ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra telekomunikační techniky

Výukové úlohy

s číslicovým signálovým procesorem

Diplomová práce

leden 2015

Diplomant: **Bc. Tomáš Vaňáč**

Vedoucí práce: **Prof. Ing. Pavel Zahradník, CSc.**

Čestné prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Dále prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé diplomové práce nebo její části se souhlasem katedry.

V Praze dne 5. ledna 2015

Tomáš Vaňáč

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra telekomunikační techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Vaňáč Tomáš**

Studijní program:
Obor: Sítě elektronických komunikací

Název tématu: **Výukové úlohy s číslicovým signálovým procesorem.**

Pokyny pro vypracování:

Pro výukový přípravek DSK s číslicovým signálovým procesorem TMS320C6748 navrhněte a odlaďte výukové úlohy na téma spektrální analyzátor, AM přijímač, Cannyho hranový operátor.

Seznam odborné literatury:

[1] www.ti.com [on-line]
[2] www.analog.com [on-line]

Vedoucí: prof. Ing. Pavel Zahradník, CSc.

Platnost zadání: do konce letního semestru 2014/2015

prof. Ing. Boris Šimák, CSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 23. 1. 2014

Anotace:

Tato práce si klade za cíl vysvětlit principy zpracování audio a video signálu na vývojové desce C6748 LCDK, která je osazena číslicovým signálovým procesorem TMS320C6748 a potřebnými vstupně výstupními obvody. S využitím těchto znalostí byly zpracovány 3 samostatné úlohy, které mají sloužit jako podpora výuky číslicového zpracování signálů. V první úloze je řešen spektrální analyzátor vstupního audio signálu, druhá úloha se týká jednoduchého AM přijímače s využitím A/D převodníku ADS16006EVM a poslední úloha zahrnuje implementaci algoritmu pro Cannyho hranový detektor. Popisu a demonstraci zmíněných úloh je věnována značná část práce.

Klíčová slova:

DSP, LCDK, C6748, CCS, MCASP, VPIF, EMIF, AIC31, I2C, EDMA, LCD, FFT, AM, kodek, raster, signál, zpracování signálu, video, audio, buffer, spektrální analyzátor, frekvence, spektrum, přijímač, Canny, Sobel, detektor hran, Gaussův filtr.

Summary:

This thesis describes the principles of audio and video processing on the C6748 development kit (LCDK), which is equipped with the TMS320C6748 digital signal processor and necessary input and output circuits. There have been created 3 separate examples to serve as a support for teaching of the digital signal processing by using this knowledge. A spectrum analyzer is discussed in the first example, the second example relates to a simple AM receiver with the A/D converter ADS1606EVM and the last example involves an implementation of an algorithm for the Canny edge detector. Description and demonstration of these examples are contained in this work.

Key words:

DSP, LCDK, C6748, CCS, MCASP, VPIF, EMIF, AIC31, I2C, EDMA, LCD, FFT, AM, codec, raster, signal, signal processing, video, audio, buffer, spectrum analyzer, frequency, spectrum, receiver, Canny, Sobel, edge detector, Gaussian filter.

1 Obsah

2 Úvod

Tato diplomová práce se věnuje tvorbě a popisu vybraných demonstračních úloh na digitálním signálovém procesoru. Úlohy jsou programovány na jediném typu přípravku, konkrétně na vývojové desce C6748 LCDK od firmy Texas Instruments. Na té je obsažen digitální signálový procesor TMS320C6748 a vedle mnoha dalších subsystémů jsou obsaženy také vstupní a výstupní obvody pro možnost přivedení audio a video signálů. Na tomto přípravku jsou postupně demonstrovány 3 samostatné úlohy:

- Spektrální analyzátor
- AM přijímač
- Cannyho hranový detektor

Vzhledem k tomu, že jednotlivé úlohy jsou navrženy jako podpůrný prostředek pro výuku číslicového zpracování signálů, je v této práci kladen důraz nejen na jejich podrobné vysvětlení, popis a názornost, ale také na souvislosti, které jsou potřebné k jejich vytvoření. Proto se v první kapitole nejprve seznámíme se samotným modulem C6748 LCDK, přičemž se zaměříme především na popis těch částí, které budeme dále využívat v našich úlohách. Pro úlohu AM přijímač je navíc využit AD převodník ADS1606EVM, jehož popisu je věnována následující kapitola. Dále je uvedeno schéma zapojení pro jednotlivé úlohy, seznam použitých zařízení a jejich nastavení.

Tím máme vyřešen popis všech potřebných zařízení a můžeme přistoupit k samotnému programování a tvorbě úloh. Ještě předtím se ale obeznámíme s vývojovým prostředím Code Composer Studio v5.1 (dále jen CCS), v němž jsou všechny úlohy naprogramovány. V kapitole je uveden nástin instalace, ukázka potřebného nastavení, seznam pro nás důležitých knihoven a také stručný popis balíčku StarterWare. V něm jsou obsaženy knihovny ovladačů pro LCDK subsystémy a jednoduché ukázkové příklady, které posloužily jako základ pro naše úlohy.

Následně se již dostáváme k hlavnímu bodu této práce, kterým je rozbor zadaných úloh. Nejprve je vždy uveden teoretický základ úlohy a poté podrobný popis její implementace na modulu C6748 LCDK. Algoritmy a funkce programu mohou být v této práci pro názornější pochopení předvedeny v neúplné podobě, příp. jen jako zobecněné části kódu. Kompletní projekty a kód všech úloh společně s potřebnými knihovnami jsou přiloženy k této práci na CD vloženém do kapsy na zadní straně desek. Úlohy a části kódu lze využít dle uvážení.

První zpracovanou úlohou je spektrální analyzátor. V této úloze je nad vstupním audio signálem (např. přímo ze zvukové karty v PC) prováděno zpracování rychlou Fourierovou transformací a výsledek je pomocí vlastních funkcí zobrazován na obrazovce monitoru jako graf frekvenčního spektra vstupního signálu společně s osami, měřítky a jednoduchými popisky. Vedle různých možností nastavení FFT, vzorkovací frekvence a použití odlišných váhovacích oken nabízí úloha také dobrý základ k různým

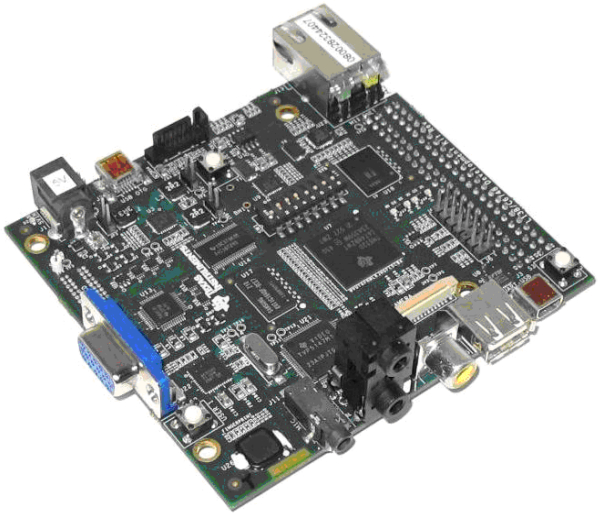
modifikacím. Lze například vytvořit vlastní funkce nad vstupním audio signálem s možností sledovat na obrazovce jejich vliv na frekvenční spektrum.

Druhou úlohou je AM přijímač, který využívá AD převodníku ADS1606EVM připojeného k LCDK modulu. Tento převodník poskytuje způsob, jak převést data ze vstupního přijímacího obvodu do digitalizované podoby a jak je přenést na EMIFA rozhraní LCDK modulu. Poté jsou získaná data vhodně zpracována a je provedena digitální AM demodulace pro získání původního nemodulovaného signálu. Nakonec je tento signál poslán na audio výstup k poslechu.

Poslední vypracovanou úlohou je Cannyho hranový detektor. Součástí příkladu je proces zachycení, zpracování, zobrazení video signálu a k tomu je implementován kompletní algoritmus pro Cannyho hranový detektor a možnost porovnat jej např. s detektorem hran založeným na Sobelově operátoru. Dále tato úloha zahrnuje možnost zobrazit si histogram obrazu, který lze uplatnit při tvorbě vlastních algoritmů a funkcí nad obrazem. Pro příklad využití je implementována funkce pro ekvalizaci histogramu. Jako základ může tato úloha dobře posloužit např. pro testování vlastností ostatních hranových detektorů, pro jejich přímé porovnání, dále pro implementaci různých obrazových filtrů (již obsahuje Gaussův filtr jako součást algoritmu Cannyho hranového detektoru), ale i pro mnohé další operace s obrazem, které se uživatel rozhodne vytvořit.

3 Vývojový modul C6748 LCDK

Pro implementaci vybraných úloh byl zvolen modul C6748 LCDK (C6748 DSP Development Kit) od firmy Texas Instruments. Ten má díky své vybavenosti a množství vstupně výstupních rozhraní vhodné předpoklady pro zpracování audio a video signálů v reálném čase. Vedle signálového procesoru TMS320C6748 totiž obsahuje řadu dalších komponent, které nám dovolí tvorbu zamýšlených aplikací v rámci jediné vývojové desky. Proto se nejprve budeme věnovat popisu tohoto modulu a především těch komponent, které budeme dále potřebovat ve zvolených úlohách.



Obrázek 1 - Vývojový modul C6748 LCDK

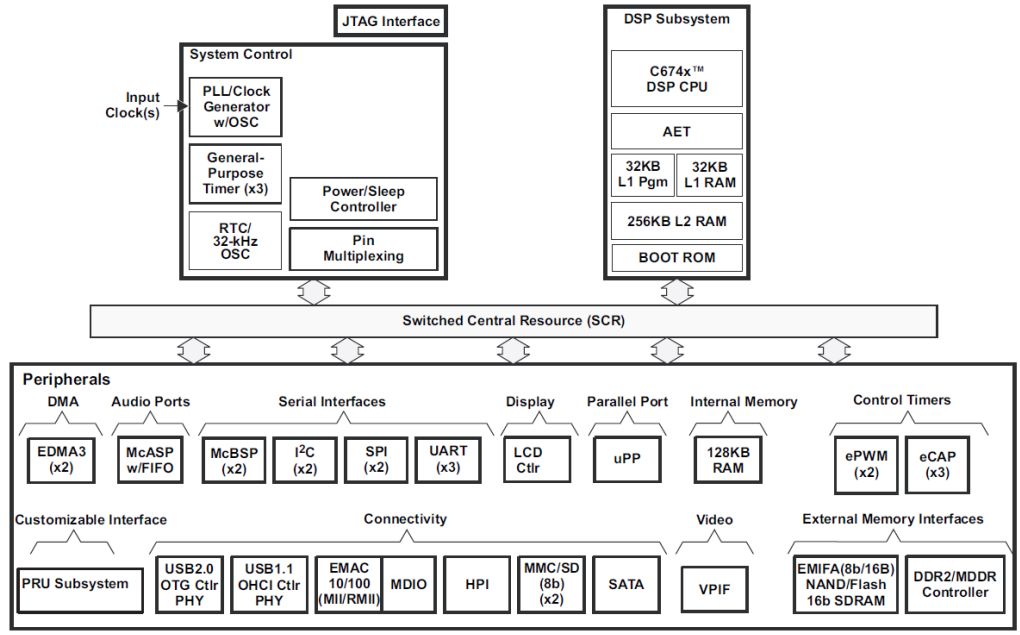
3.1 Specifikace LCDK

V následujícím bodovém přehledu jsou uvedeny hlavní charakteristiky modulu C6748 LCDK. Podrobnější specifikace a popis jednotlivých částí modulu lze nalézt v technickém manuálu (TMS320C6748 DSP Technical Reference Manual - SPRUH79a, příp. TMS320C674x Processor Peripherals Overview - SPRUFK9f).

- TMS320C6748 DSP procesor s pevnou/plovoucí čárkou na frekvenci 456MHz
- 128 MB DDR2 SDRAM na 150MHz
- 128 MB 16b NAND FLASH
- EDMA3, 2x Master/Slave I^2C sběrnice
- Mini-USB-UART port, mini-USB OTG port (USB 2.0)
- USB Host port (USB 1.1)
- 10/100 Mbps Ethernet
- MCASP, AIC31 stereo codec
- LCD kontrolér a Video Port Interface (VPIF)
- VGA port, kompozitní video vstup
- 3 audio porty (Line IN & Line OUT & MIC IN)
- Rozhraní pro připojení externí paměti EMIFA

3.2 Komponenty LCDK

Zde si postupně představíme důležité subsystémy, které využijeme v řešených úlohách a které jsou standardně obsaženy na LCDK modulu - viz blokové schéma procesoru TMS320C6748 a jeho periferních obvodů (viz obr. 2).



Obrázek 2 - Blokové schéma procesoru TMS320C6748 (převzato z [1])

U popisovaných subsystémů a periferních obvodů jsou vždy uvedeny jejich základní vlastnosti a jejich hlavní funkce. Dále je nastíněn příklad programové konfigurace a použití daného subsystému ve vybraných úlohách.

3.2.1 Digitální signálový procesor TMS320C6748

Digitální signálové procesory jsou obecně svou architekturou a instrukčním souborem optimalizovány pro zpracování digitalizovaných signálů v reálném čase. Standardně mají Harvardskou architekturu (s oddělenou pamětí pro data a pro program), zahrnují CPU se speciální násobičkou schopnou provádět operaci násobení s přičítáním a mají hardwarovou podporu kruhových bufferů a bitové reverzace (výhodné pro výpočetní operace typu FIR, FFT). Vzhledem k těmto vlastnostem a důrazu na nízkou spotřebu jsou využívány v široké množině aplikací (např. datová komprese, modulace, filtrace, zpracování obrazu a zvuku) v zařízeních, která dnes a denně používáme.

Na námi zvoleném vývojovém modulu máme k dispozici DSP procesor TMS320C6748, tedy procesor z řady C6000, což je nejvýkonnější řada od firmy Texas Instruments. TMS320C6748 kombinuje výhody procesorů s pevnou řádovou čárkou a procesorů s plovoucí řádovou čárkou do jedné architektury pro snadnější vývoj aplikací. Procesor pracuje na frekvenci 456MHz s architekturou VLIW a vykonává až 8 32-bitových instrukcí za strojový cyklus (VLIW - very long instruction word, velmi dlouhé instrukce rozdělené do většího množství polí, přičemž každé pole obsahuje kód pro jednu operaci,

což dovoluje zpracovávat více jednoduchých instrukcí paralelně). Procesor má dvouúrovňovou vyrovnávací (cache) paměť pro načítání instrukcí a dat z pomalé paměti. Na 1. úrovni je programová cache paměť 32KB L1P a datová cache paměť 32KB L1D. Na 2. úrovni je programová cache paměť L2P sestávající z paměťového prostoru 256KB. L2 cache paměť lze konfigurovat jako adresovanou paměť, vyrovnávací paměť, nebo jako kombinaci obou. Pro rychlý přenos dat mezi jednotlivými vyrovnávacími paměťmi podporuje procesor interní přímý přístup do paměti IDMA (Internal direct memory access). Součástí čipu procesoru je též sdílená paměť 128KB RAM dostupná z obou úrovní cache pamětí. Procesor disponuje šesti ALU jednotkami a dvěma násobičkami (jedna pro násobení s pevnou a druhá pro násobení s plovoucí řádovou čárkou). Je podporována adresace po bytech (po 8, 16, 32, nebo 64b). Výkonnost procesoru se blíží až k 3648 MIPS (Milion Operation per Second) a 2746 MFLOPS (Milion Floating point Operations per Second).

3.2.2 Enhanced Direct Memory Access (EDMA)

EDMA, tedy rozšířený přímý přístup do paměti, umožňuje přímou výměnu dat mezi dvěma oblastmi mapované paměti bez účasti CPU. To vede k efektivnějšímu využití procesoru, který může mezitím zpracovávat další instrukce. C6748 LCDK obsahuje EDMA3 řadič, který zahrnuje 2 kanálové kontroléry, 3 kontroléry přenosu a 64 nezávislých DMA kanálů.

Kanálové kontroléry (EDMA3CC) slouží jako uživatelské rozhraní pro konfiguraci EDMA3, pro vyhodnocení priorit příchozích požadavků, nebo přerušení od periferií. Dále tyto kontroléry potvrzují žádosti o přenos pro kontroléry přenosu. Zahrnují parametry RAM (PaRAM), kanálové řídící registry a registry přerušení.

Kontroléry přenosu (EDMA3TC) zodpovídají za přenos dat. Žádosti o přenos potvrzené od EDMA3CC obsahují vlastnosti přenosu, na jejichž základě zvolí EDMA3TC zdrojovou a cílovou adresu pro daný přenos.

DMA kanály představují v případě LCDK modulu 64 programovatelných kanálů pro provedení rozdílných na sobě nezávislých přenosů.

Příkladem použití EDMA3 a jedním z programovatelných DMA kanálů je např. uložení dat ze sériového rozhraní McASP0 (viz kapitola 3.2.8) do paměti. EDMA3 řadič může být v tomto případě nastaven tak, že pokud dojde k dokončení přenosu, nastane přerušení, na jehož základě je inicializován další přenos. Pro úspěšné zpracování dat v reálném čase platí, že následné operace s daty uloženými do paměti musejí být dokončeny dříve, než je dokončen další DMA přenos. V řešených úlohách budeme EDMA3 aplikovat podobně jako ve zmíněném příkladu. U 1. a 3. úlohy využijeme EDMA3 také pro přenos obrazových dat mezi periferiemi VPIF/LCDC a pamětí a u 2. úlohy pro přenos dat mezi rozhraním EMIFA a pamětí.

3.2.3 Sběrnice I^2C

Inter-Integrated Circuit (I^2C) je sériová sběrnice, která slouží k připojení a řízení subsystémů na LCDK modulu. Sběrnici představují 2 vodiče, z nichž jeden přenáší hodinový signál SCL (Synchronous Clock) a druhý tvoří datový kanál SDA (Synchronous Data). Po této sběrnici lze periferním zařízením posílat, či od nich přijímat data o šířce

až 8b a o rychlosti až 400Kbps. I^2C umožňuje zapínat, či vypínat periferní obvody a jsou na ní podporovány události přímého přístupu do paměti DMA a přerušení. Sběrnice funguje na principu Master/Slave, kde Master iniciuje začátek a konec komunikace a generuje hodinový signál SCL.

Z připojených a v zadaných úlohách použitých subsystémů na sběrnici I^2C můžeme jmenovat např. stereo kodek AIC31, či dekodér TVP5147. Např. kodek AIC31 je v úlohách nejprve konfigurován jako Slave zařízení, poté je pro něj inicializováno rozhraní I^2C0. Toto rozhraní je následně využíváno pro konfiguraci ADC a DAC převodníku, pro konfiguraci formátu dat a přenosových parametrů (např. velikost jednoho přenosového slotu, či zda přenášet oba stereo kanály, nebo jen levý/pravý).

3.2.4 Interrupt Controller (INTC)

DSP Interrupt Controller (INTC) je řadič přerušení, který se stará o správu přerušení pro procesor. Události na zařízeních, které mohou vyvolat přerušení, kombinuje řadič až do 12 prioritizovaných CPU přerušení. Zdroj přerušení pro každé z 12 přerušení lze volit uživatelsky z tabulky, ve které jsou uvedena všechna přípustná přerušení pro procesor TMS320C6748 (tabulka 5-6 v technickém manuálu - SPRS590D).

V řešených úlohách se bez přerušení neobejdeme. Nejprve jsou proto v programu nadefinována potřebná přerušení (např. pro McASP audio přenosy) a poté je vždy na začátek běhu programu vloženo volání funkce pro inicializaci řadiče přerušení, bez které by proces vyhodnocování přerušení a volání obslužných funkcí nefungoval. Pokud je vše nastavené správně, bude McASP vždy po dokončení přenosu vyvolávat přerušení, po němž dojde k zahájení přenosu nového. To nám zajistí neustálé občerstvování zpracovávaných dat.

3.2.5 Paměti

3.2.5.1 Vestavěné na LCDK

Na desce LCDK lze využít vedle interních cache pamětí procesoru také paměť 128 MB DDR2 SDRAM, která je dostupná jak procesoru, tak i některým periferiím. Další vestavěnou pamětí je NAND FLASH 128 MB, na které je uložen firmware, případně zde mohou být uloženy projekty demo aplikací.

3.2.5.2 Rozhraní EMIFA

External memory interface A (EMIFA) je rozhraní, přes které můžeme k procesoru připojit externí paměťový prostor pro program i pro data s až 16b širokou datovou sběrnicí. Podporovány jsou:

Asynchronní paměti:

- NOR (8-b, nebo 16-b široká datová sběrnice)
- NAND (8-b, nebo 16-b široká datová sběrnice)

Synchronní paměť (podpora až do rychlosti 100MHz):

- 16-b SDRAM s adresovatelným prostorem až 128-MB

Rozhraní EMIFA využijeme ve 2. úloze nestandardně a namísto externí paměti jeho prostřednictvím připojíme k LCDK modulu převodník ADS1606EVM. Bližší informace a způsob konfigurace EMIFA rozhraní jsou obsaženy v kapitole věnované rozboru úlohy AM přijímač (viz kap. 8.2.2.2).

3.2.6 General-Purpose Input/Output (GPIO)

General-Purpose Input/Output (GPIO) je rozhraní poskytující přístup k univerzálním pinům procesoru, které lze konfigurovat buď jako výstupy, nebo jako vstupy. Pokud je pin nastaven jako výstupní, lze jeho stav nastavovat prostřednictvím příslušného registru. V případě, že je pin nastaven jako vstupní, lze zjistit stav vstupu přečtením stavu interního registru. Všechny GPIO piny lze využít také jako zdroje přerušení a zdroje pro generování EDMA událostí s možností nastavení, zda má být přerušení generováno při náběžné, nebo při sestupné hraně vstupního signálu.

V úloze AM přijímač je jeden z dostupných pinů nastaven jako vstupní s přerušením závislým na sestupné hraně vstupního signálu. Na vybraný pin je připojen Data Ready signál z AD převodníku, který poté slouží jako impulz k přečtení dat z výstupu AD převodníku, jejich uložení do vnitřního registru EMIFA a k následnému přenesení prostřednictvím EDMA do paměti.

3.2.7 Audio Codec AIC3106

Na LCDK modulu je na vstupu a výstupu audio periferií zařazen obvod TLV320AIC3106, tedy audio stereo kodek s nízkou spotřebou. Tento kodek komunikuje s procesorem C6748 přes sériové rozhraní McASP (viz následující podkapitola) a je řízen po sběrnici I^2C. Jeho hlavní funkcí je obousměrný převod mezi analogovou a digitální reprezentací signálu (ADC/DAC). Mimo jiné umožňuje nastavit některé parametry formátu audio dat. ADC i DAC podporují vzorkovací frekvence 8-96kHz, lze nastavit vstupní zisk a výstupní útlum. ADC má obsažen programovatelný předzesilovač a automatické řízení zisku (automatic gain control - AGC) pro mikrofonní vstupy s nízkou úrovní signálu. Kodek umí pracovat s 16, 24, nebo 32-bitovou reprezentací digitálního signálu. Maximální přípustná efektivní hodnota napětí vstupního signálu kodeku je 0,707V.

Pro ovládání kodeku AIC3106 využijeme na LCDK sběrnici I^2C. Pro přenos audio dat po sériové sběrnici jsou podporovány protokoly SPI a I^2S (Integrated Interchip Sound). I^2S protokol umožňuje přenos digitalizovaných audio dat mezi dvěma integrovanými obvody. V řešených aplikacích využívajících vstupních a výstupních audio signálů nastavíme typ přenosového protokolu kodeku právě na I^2S protokol, který poté budeme nastavovat také na sériové sběrnici McASP (viz následující podkapitola). Dále budeme pro kodek konfigurovat především vzorkovací frekvenci a mód určující, zda bude přenesen a uložen do paměti stereo signál, nebo jen jeden z kanálů (pravý, nebo levý). V řešených úlohách zpracováváme jen jeden z kanálů, vzorkovací frekvenci kodeku máme nastavenou na 48kHz a pokud chceme, můžeme ji snadno měnit v dovoleném rozsahu (8 – 96kHz).

3.2.8 Multichannel Audio Serial Port (McASP)

Multichannel Audio Serial Port (McASP) je sériové rozhraní navržené pro potřeby vícekanálových audio aplikací vhodné pro přenos multiplexu s časovým dělením (TDM) pomocí sériového protokolu I^2S. TDM rámec se může skládat z 2-32 time slotů, příp. z 1 time slotu (dávka). Velikost time slotu může být 8, 12, 16, 20, 24, 28, nebo 32b. McASP se skládá z vysílací a z přijímací části, které mohou pracovat synchronizovaně, nebo zcela nezávisle s odlišným časováním, s různými módy přenosu a s jinou bitovou reprezentací dat. McASP zahrnuje až 16 serializátorů, které mohou být zapnuty a nastaveny buď jako vysílací, nebo jako přijímací.

Na LCDK slouží McASP jako obousměrný prostředník mezi kodekem AIC3106 a procesorem, případně pamětí při použití EDMA. V řešených úlohách budeme McASP rozhraní konfigurovat jak pro příjem, tak pro odesílání audio dat a budeme využívat EDMA pro rychlé ukládání dat do paměti. Nejprve nastavíme mód přenosu McASP na I^2S protokol, nakonfigurujeme podporu EDMA, stanovíme délku slotu a počet slotů na rámec pro režim TDM. Dále zapneme systémové hodiny zvlášť pro přijímací, zvlášť pro vysílací sekci, zapneme synchronizaci vysílací a přijímací sekce, inicializujeme jeden serializátor pro přijímací a jeden pro vysílací sekci. Nakonfigurujeme vstupní a výstupní McASP piny, nadefinujeme McASP přerušení a nakonec zapneme přijímací a vysílací sekci rozhraní McASP. Proces uložení audio dat do paměti, či jejich odeslání na výstup již vyžaduje složitější strukturu programu. Nyní jsme si uvedli pouze stručný postup konfigurace McASP pro LCDK modul. Podrobnější popis McASP a konkrétní provedení konfigurace si ukážeme v kapitole věnované rozboru úlohy spektrální analyzátor (viz kap. 7.2.2).

3.2.9 Video dekodér TVP5147M1

Video dekodér TVP5147M1 je na LCDK umístěn na vstup kompozitního video signálu, kde převádí vstupní kompozitní (podporovány jsou formáty PAL, NTSC a SECAM kompozitní a S-video) analogový video signál na digitální komponentní video signál (YCrCb – konkrétně formát 4:2:2, lze volit mezi 10b a 20b). Video dekodér dále umožňuje programovatelné zesílení a offset vstupního signálu, synchronizaci, zahrnuje adaptivní hřebenové filtry pro potlačení obrazového šumu a artefaktů. Přes I^2C rozhraní lze ovládat obrazové vlastnosti jako kontrast, jas, či sytost. Video dekodér obsahuje 2 10-bitové A/D převodníky s výkonností 30 MSPS. Po oddělení jasových a chrominančních složek jsou tyto složky zpracovány odděleně 2 procesory. Digitalizovaný YCrCb video signál z výstupu video dekodéru pokračuje na rozhraní VPIF (viz následující podkapitola).

Tento video dekodér použijeme pouze u úlohy Cannyho hranový detektor, kde budeme mít na jeho vstup připojenou videokameru. Programové spuštění dekodéru je jednoduché, přes I^2C rozhraní provedeme inicializaci video dekodéru pro příjem kompozitního video signálu a dále nemusíme nic dalšího nastavovat.

3.2.10 Video Port Interface (VPIF)

Video Port Interface (VPIF) je rozhraní, které zajišťuje příjem a odesílání video dat, které za pomoci EDMA přenosu po snímcích ukládá do paměti, nebo naopak z paměti

načítá. VPIF má 2 vstupní kanály pro příjem a 2 výstupní kanály pro odesílání. Podporovány jsou formáty PAL a NTSC progresivního i prokládaného typu, HDTV a SDTV.

V případě LCDK je vstupem ve VPIF digitalizovaný komponentní video signál z video dekodéru TVP5147M1. VPIF rozhraní budeme potřebovat pouze u úlohy Cannyho hranový detektor a to jen ve směru pro příjem video dat. Programové zprovoznění VPIF rozhraní zahrnuje inicializaci, nastavení EDMA přenosu, bufferů, potřebných přerušení a další podružnou konfiguraci. Strukturu si podrobněji vysvětlíme v kapitole o úloze Cannyho hranový detektor (viz kap. 9.2.2). Nyní si stručně uvedeme důležité parametry, které přes rozhraní VPIF budeme nastavovat. Prvním takovým parametrem je velikost jednoho dávkového přenosu mezi VPIF a pamětí s využitím EDMA. Nastavuje se skrze registr DMA Request Size Control Register (REQSIZE) a na výběr je velikost 32B, 64B, 128B a 256B. V případě řešené úlohy používáme velikost 128B. Dále je nutné zvolit a spustit kanál 0 na VPIF pro příjem dat z video dekodéru. Pro další zpracování dat je důležitou volbou formát zachyceného videa, přičemž máme na výběr ze 2 možností, SD NTSC 720x480I, nebo SD PAL 720x576I. Pro úlohu Cannyho hranový detektor si zvolíme formát SD PAL 720x576I.

3.2.11 LCD Controller (LCDC)

Máme data z video vstupu uložená a již zpracovaná v paměti a nyní je chceme zobrazit na monitoru. O to se v C6748 stará LCD Controller (LCDC), který podporuje zobrazení na asynchronní LCD (s mapovanou pamětí) i synchronní LCD (rastr) rozhraní. LCDC řadič se skládá ze dvou nezávislých řadičů, přičemž současně může fungovat pouze jeden z nich:

- *Raster Controller* – obsluhuje zobrazení pro synchronní LCD. Poskytuje časování a data pro konstantní obnovu snímků na pasivním displeji. Podporuje široké množství barevných i monochromatických formátů. Video data jsou zpracována a ukládána po snímcích do bufferů, pomocí EDMA přenosu jsou poté směrována na externí zobrazovací zařízení.
- *LCD Interface Display Driver (LIDD)* – obsluhuje zobrazení pro asynchronní LCD. Poskytuje plně programovatelné řídící signály (CS, WE, OE, ALE) a výstupní data pro externí displeje.

LCDC, přesněji Raster Controller využijeme pro zobrazení zpracovaných snímků na LCD monitoru v úlohách spektrální analyzátor a Cannyho hranový detektor. Pro zobrazení nastavíme EDMA přenos, v němž stanovíme ukazatel na buffer v paměti se snímkem připraveným k odeslání a definujeme šířku a výšku obrazu (640x480). Pro LCDC Raster Controller konfigurujeme hodiny a časování parametrů pro horizontální i pro vertikální zobrazení, dále mód zobrazení (TFT, nebo STN) a barevnou paletu zobrazovaných dat (barevná, nebo monochromatická). Pro spuštění zobrazovací rutiny je nutné ještě nadefinovat přerušení a na závěr nezapomenout spustit Raster Controller. Podrobnější popis je uveden v kapitolách rozboru příslušných úloh (viz kap. 7.2.5).

3.2.12 Video DAC THS8135

Rychlý trojitý D/A převodník THS8135 je optimalizovaný pro použití ve video aplikacích. Podporuje řadu video formátů (RGB i YCrCb), rozlišení až 1600x1200 s obnovovací frekvencí do 85Hz. Výkonnost převodníku dosahuje 240MSPS, je 10-bitový s přídavnými obvody pro tvorbu synchronizačních pulzů a zatemňovacích intervalů.

Na LCDK modulu je tento D/A převodník umístěn za LCDC řadič, z nějž zpracovává data přímo pro externí LCD monitor připojený k modulu přes VGA rozhraní. Programově není nutné převodník konfigurovat, pro správné zobrazení obrazu uloženého v paměti postačí nastavit LCDC řadič.

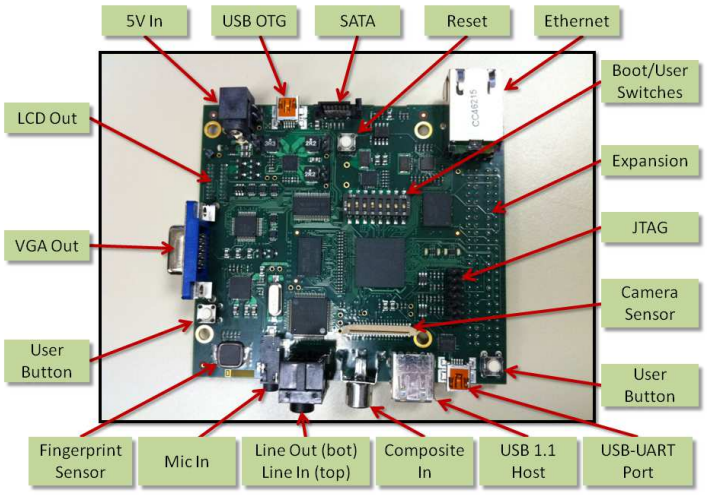
3.2.13 Power and Sleep Controller (PSC)

Power and Sleep Controller (PSC) je řadič zodpovědný za správu napěťového zapínání a vypínání systémů procesoru, za zapínání a vypínání systémových hodin a za reset. Především se používá k řízení výkonu subsystémů umístěných na čipu procesoru a periferních subsystémů procesoru.

Prakticky se s tímto řadičem v programu setkáme při zapínání dílčích systémů, se kterými chceme v řešených aplikacích pracovat, tedy např. VPIF, LCDC, McASP, GPIO, či EDMA3. Pro spuštění daného subsystému využijeme v programu funkci z knihovny psc.h, pomocí níž snadno zvolíme, který subsystém chceme uvést do provozu a jakým způsobem jej chceme napájet. V řešených úlohách používáme subsystémy s nepřerušovaným stálým napájením.

3.3 Konektory LCDK

Pro připojení externích zařízení k modulu C6748 LCDK je na modulu přítomna řada konektorů (označeny J1 – J16). Zde si krátce zmíníme ty, které budeme potřebovat v řešených úlohách (přehled všech je na obr. 3). Zaměříme se také na přídavný externí JTAG emulátor, bez kterého se neobejdeme při nahrávání zkompilovaných programů z PC do paměti na LCDK modulu a při ladění kódu.



Obrázek 3 - Přehled vstupů a výstupů modulu C6748 LCDK (převzato z [2])

3.3.1 JTAG emulátor XDS100v2

Pro připojení PC k modulu C6748 LCDK, pro možnost nahrávání vytvořených programů do paměti modulu a pro ladění programu potřebujeme externí JTAG emulátor, protože ač má LCDK modul 14-pinový TI JTAG konektor (J6), nemá zabudovaný vlastní JTAG emulátor. Existuje více možných variant externích emulátorů, my jsme zvolili JTAG emulátor XDS100v2 (viz obr. 4).

Obrázek 4 - JTAG emulátor XDS100v2

XDS100v2 emulátor představuje rozhraní mezi JTAG a USB. Obsahuje mini USB a 14-pinový TI konektor, pomocí kterého jej lze přímo připojit na konektor J6 LCDK modulu. Je napájen z USB, podporuje USB 2.0, dostupné jsou ovladače pro operační systémy Windows XP, Vista, 7, 8 a Linux. Emulátor je kompatibilní s vývojovým prostředím Code Composer Studio IDE™ od verze 4 výše a také s nástrojem UniFlash™ od Texas Instruments. Z podporovaných funkcí emulátoru zmíníme podporu ladění programu z vývojového prostředí Code Composer Studio - emulace připojení/odpojení modulu, čtení z paměti, zápis do paměti, čtení stavu registrů, nahrání programu, spuštění, zastavení, krokování programu, možnost použití zarážek běhu programu (tzv. breakpoints), reset registrů a paměti.

3.3.2 Audio vstupy a výstupy

Vstupní audio zařízení (mikrofon, výstup zvukové karty,..) máme možnost připojit přes konektory J10 (Line In – horní zdířka) a J11 (MIC In), které dále přenáší audio signál na kodek AIC3106. Oba konektory jsou typu 3,5mm Jack. V řešených úlohách lze libovolně zvolit, který z těchto vstupů použít, program to rozezná.

Reproduktory či sluchátka připojíme do konektoru J10 (Line Out – dolní zdířka).

3.3.3 Video vstupy a výstupy

Kameru připojíme na konektor J9 typu RCA Jack, tedy vstup pro kompozitní video signál. Vstupním obvodem pro tento konektor je video dekodér TVP5147M1.

Monitor připojíme přes VGA rozhraní na konektor J8, který je obsluhován LCDC řadičem a video DAC převodníkem THS8135.

3.3.4 Konektor pro rozhraní EMIFA

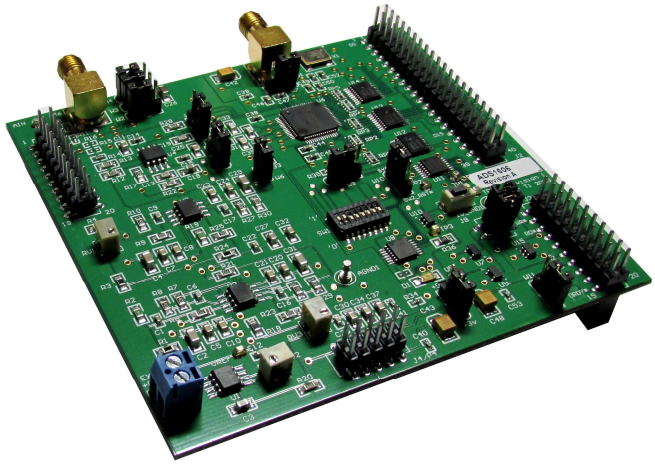
Pro připojení externího A/D převodníku ADS1606EVM použijeme 46-pinový konektor J14 pro rozhraní EMIFA, jehož popis je uveden v kapitole 5.

3.3.5 Napájení modulu LCDK

Pro úplnost přehledu použitých konektorů na LCDK modulu v řešených aplikacích uvedeme také napájecí konektor J1, přes nějž je modul napájen napětím +5V.

4 Modul ADS1606EVM

U úlohy AM přijímač využijeme externí A/D převodník pro digitalizaci analogového signálu přijatého ze vstupního přijímacího obvodu. Výstup převodníku poté připojíme na EMIFA rozhraní C6748 LCDK modulu. Pro tyto účely jsme vybrali A/D převodník ADS1606EVM (viz obr. 5), který si představíme v této kapitole.



Obrázek 5 - Modul ADS1606EVM

Modul ADS1606EVM obsahuje delta sigma A/D převodník s rychlostí vzorkování 5 MSPS, s šířkou pásma 2,45 MHz, s odstupem signálu od šumu až 88 dB a s možností zvolit si zdroj referenčního napětí. Součástí převodníku je také konfigurovatelná fronta FIFO pro výstupní data.

4.1 Analogové rozhraní modulu

Analogový signál lze do převodníku přivést buď přes SMA konektor J6, nebo přes 20-pinový konektor J1/P1. Vstupní signál je nejprve převeden na rozdílový signál a poté je vztažen k souhlasnému napětí přivedeného ze subsystému pro tvorbu referenčního napětí. Následně je již signál přiveden na vstupy AINP a AINN převodníku ADS1606. V analogové části se dále nacházejí subsystémy pro tvorbu systémových hodin a referenčního napětí. Pomocí příslušných jumperů lze vybrat zdroj referenčního napětí (interní, nebo externí), či zda budou použity interní, nebo externí hodiny (CLK). Pokud jsou použity interní hodiny, dovoluje převodník použít buď interní 40-MHz oscilátor, nebo externí přes konektor J5. Při volbě externího referenčního napětí je nutné zajistit na konektoru J7 napětí v rozsahu 8 – 40 V.

4.2 Převodník ADS1606

Vstupní signál je vzorkován delta sigma modulátorem rychlostí 40 MSPS (při f_{CLK} = 40 MHz). Modulátorem s rozdílovým analogovým vstupním signálem (V_{IN} = AINP - AINN), který je měřen oproti rozdílovému referenčnímu napětí (V_{REF} = V_{REFP} - V_{REFN}) tak, že největší rozsah vstupního signálu je dán interně

jako ±1,467V_{REF}. Tedy pro V_{REF} = 3V je rozsah vstupního napětí ±4,4V. Pokud je vstupní signál mimo rozsah, dojde k nastavení digitálního výstupu OTR (out of range). Za modulátorem následuje lineární fázový digitální filtr s nízkým zvlněním, který provádí decimaci dat z modulátoru na výslednou rychlost 5 MSPS.

4.3 Digitální rozhraní modulu

Digitální výstup převodníku je 16-bitový (8000h – 7FFFh) na 40-pinovém konektoru J2/P2. Vyčítání dat z převodníku je ovládáno přes jednoduché paralelní rozhraní na konektrou J3/P3. Sestupná hrana výstupního signálu $\overline{DRDY}$ (J3/P3 – pin 19) indikuje, že jsou dostupná nová data. Pro aktivování výstupní sběrnice pro přenos 16-bitového digitálního signálu musí být oba řídící signály $\overline{CS}$ (J3/P3 – pin 1) a $\overline{RD}$ (J3/P3 – pin 5) nastaveny na stav low. Pokud jsou tyto signály nastaveny jinak, je výstupní rozhraní ve stavu vysoké impedance. Reset převodníku lze provádět manuálně tlačítkem SW2, nebo nastavením řídícího signálu $\overline{RESET}$ (J3/P3 – pin 3) na stav low. Při vypnuté frontě FIFO je doporučeno provádět reset se sestupnou hranou CLK nastavením řídícího signálu $\overline{RESET}$ na low. Tím jsou všechny digitální obvody nulovány, digitální výstupy jsou nastaveny na low a signál $\overline{DRDY}$ je nastaven na high. Po skončení resetu je po druhé vzestupné hraně CLK nastaven signál $\overline{DRDY}$ na low. Dalších 47 cyklů signálu $\overline{DRDY}$ je vhodných pro vyrovnání digitálního filtru před dalším načtením dat. V praxi se vyplatí reset periodicky využívat, pokud máme více A/D převodníků a potřebujeme jejich synchronizaci.

Převodník obsahuje také nastavitelnou vyrovnávací paměť typu FIFO (First-in First-out) pro výstupní data. FIFO umožňuje dočasně uložit digitalizovaná data před odesláním na výstup. Pomocí DIP switch přepínače SW1 (switch 1, 2, 3) lze nastavit hloubku paměti FIFO (0 - 14). FIFO paměť je nastavena před spuštěním převodníku a poté se již nemění.

4.4 Napájení modulu

Pro plnou funkčnost modulu ADS1606EVM je nutné zajistit definované napájení přes konektor J4/P4:

- Pro analogové rozhraní modulu zdroj napětí ± 6 Vdc připojený na piny 1 a 2
- Pro napájení analogové části ADC zdroj napětí 5 Vdc připojený na pin 3
- Pro napájení digitálního části ADC zdroj napětí 3 Vdc připojený na pin 9
- Pro digitální I/O rozhraní modulu zdroj napětí 3, nebo 5 Vdc připojený na pin 10

V řešené úloze je použit pouze jeden 3 Vdc zdroj napětí připojený na pin 9 konektoru J4/P4, přičemž jumper W8 je nastaven do polohy 1-2. Tím je zvolena úroveň napájecího napětí 3 Vdc pro digitální I/O rozhraní modulu. To je výhodné zejména při připojení digitálního I/O rozhraní převodníku k EMIFA rozhraní LCDK modulu, které má rovněž úroveň napětí 3 Vdc. Lze tedy přímo propojit příslušné vstupy a výstupy modulu převodníku a LCDK modulu.

5 Zapojení, konfigurace a použitá zařízení

V této kapitole si uvedeme možnosti fyzického nastavení LCDK modulu, především nastavení DIP switch přepínače. Poté si postupně pro všechny 3 řešené úlohy stanovíme schéma zapojení a seznam použitých zařízení. Speciálně pro úlohu AM přijímač si navíc popíšeme připojení A/D převodníku a schéma vstupního přijímacího obvodu.

Vysvětlivky pro bloková schémata zapojení uvedená v této kapitole:

- Modré bloky představují obvody osazené na modulu C6748 LCDK
- Černé bloky představují vstupní a výstupní konektory na modulu C6748 LCDK (označení J1-J16 odpovídá značení na LCDK)
- Bílé bloky představují externí zařízení
- Šipky znázorňují směr toku dat

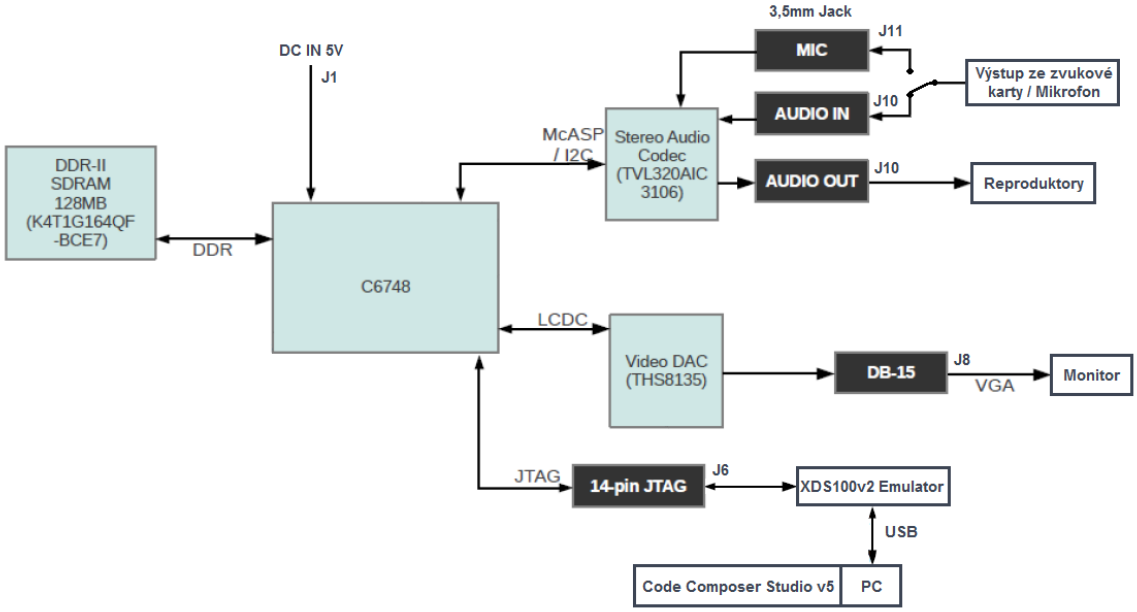
5.1 Modul C6748 LCDK

Zde zmíníme fyzicky nastavitelné prvky a indikátory, které se nacházejí na desce modulu C6748 LCDK. A především si určíme, jak tyto prvky a indikátory využijeme v řešených úlohách.

Na LCDK modulu jsou osazeny 2 uživatelsky programovatelná tlačítka (S2, S3) obsluhovaná pomocí rozhraní GPIO signálů (General-purpose input/output – piny procesoru, jejichž chování může řídit uživatel) a tlačítko reset (S1), kterým lze resetovat celý modul pro vymazání obsahu registrů a pamětí. Dále je na desce DIP switch s 8 nastavitelnými přepínači (SW1). Přepínače 1-4 stanovují bootovací pravidla. Lze zvolit ze čtyř možností (boot z vlastní BOOT ROM procesoru, z rozhraní UART2, z paměti NAND 16, nebo z MMC/SD karty). Např. při nastavení přepínačů 1-4 do polohy OFF, ON, ON, ON zvolíme boot z paměti NAND, kde je uložená demo aplikace (více informací o jejím spuštění lze najít v krátkém návodu Getting Started Guide přiloženém k LCDK modulu). Zbylé 4 přepínače (5 - 8) jsou uživatelsky konfigurovatelné, v řešených úlohách je však nebudeme potřebovat. Pro vybrané úlohy nastavíme všech 8 přepínačů do polohy OFF, tedy bootování z vlastní BOOT ROM procesoru. Na LCDK modulu se nachází i 7 indikačních LED diod (D1-D7). Dioda D1 indikuje přítomnost napájecího napětí 5V. D2 se rozsvítí, pokud napájecí napětí přesáhne 5,8V. D3 indikuje, že je deska napájena buď z konektoru J1, nebo z USB. Funkci dalších 4 diod (D4-D7) může definovat uživatel pomocí rozhraní GPIO signálů. Na modulu jsou ještě 4 jumpery (JP1 – JP4), které necháme nezapojené.

5.2 Úloha 1 – Spektrální analyzátor

V úloze spektrální analyzátor, jehož blokové schéma je na obrázku 6, připojujeme současně audio a video zařízení. Úloha je založena na zpracování dat ze vstupního audio zařízení (mikrofon, výstup zvukové karty, příp. jiný zdroj audio signálu) a následném zobrazení jejich spektra na monitoru (libovolný monitor s VGA vstupem a rozlišením obrazu alespoň 640x480 pixelů). Lze připojit také výstupní zařízení pro poslech vstupního signálu, není to však nezbytné. Pro ověřování tvaru, amplitudy, frekvence vstupního, příp. výstupního audio signálu můžeme použít osciloskop.



Obrázek 6 - Blokové schéma zapojení úlohy spektrální analyzátor

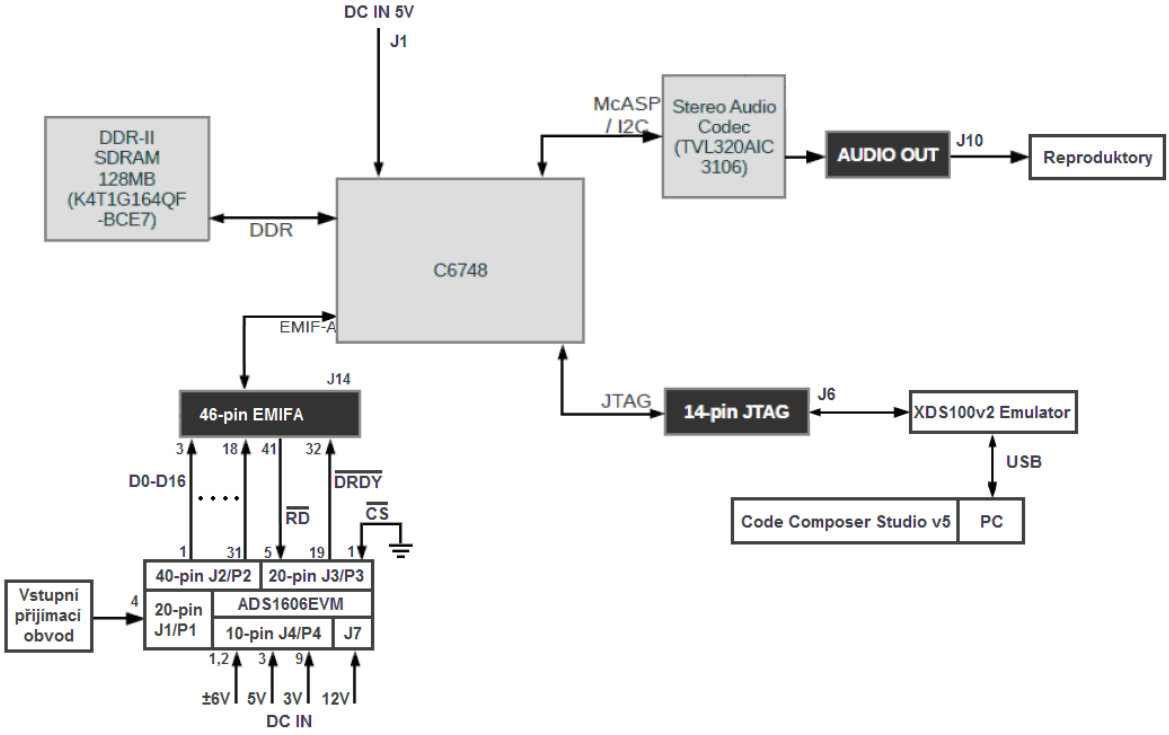
Použitá zařízení:

- Modul LCDK C6748 verze A5
- XDS100v2 JTAG emulátor
- Notebook s vývojovým prostředím Code Composer Studio v5.1
- Zvuková karta - integrovaná Realtek ALC269 (příp. mikrofon)
- Reproduktory (příp. sluchátka)
- Monitor s rozlišením alespoň 640x480
- Osciloskop

5.3 Úloha 2 – AM přijímač

V úloze AM přijímač připojujeme přes EMIFA rozhraní na konektor J14 modulu C6748 LCDK převodník ADS1606EVM (viz blokové schéma na obrázku 7, kde jsou vedle šipek znázorněna i čísla příslušných pinů). Propojení se skládá z 16 datových signálů (D0 - D15) a ze 3 řídících signálů ($\overline{CS}$, $\overline{RD}$ a $\overline{DRDY}$), přičemž chip select $\overline{CS}$ signál je připojen na zem, což znamená, že je trvale ve stavu low a čtení dat z převodníku je proto

řízeno pouze read $\overline{RD}$ signálem po obdržení sestupné hrany data ready $\overline{DRDY}$ signálu. Na vstup převodníku je připojen analogový vstupní signál ze vstupního přijímacího obvodu, pro testování a zprovoznění převodníku byl používán generátor funkcí s frekvencí 10 kHz. Převodník je napájen třemi různými zdroji napětí (viz kap. 4.4) a na konektor J7 má připojen zdroj napětí 12 Vdc pro vytvoření referenčního napětí. Na výstupní straně je připojeno audio zařízení pro poslech zpracovaného vstupního signálu (reproduktory, či sluchátka).



Obrázek 7 - Blokové schéma zapojení úlohy AM přijímač

Použitá zařízení:

- Modul LCDK C6748 verze A5
- XDS100v2 JTAG emulátor
- Notebook s vývojovým prostředím Code Composer Studio v5.1
- A/D převodník ADS1606EVM
- Zdroje napětí ±6 Vdc, 3 Vdc, 5 Vdc, 12 Vdc
- Vstupní přijímací obvod
- Reproduktory, příp. sluchátka
- Osciloskop, generátor funkcí

5.3.1 Modul ADS1606EVM

Modul ADS1606EVM obsahuje řadu jumperů a DIP switch přepínačů. V tabulkách na následující stránce je uvedeno jejich nastavení použité v úloze AM přijímač. Pro úplnost je přiložena také tabulka s přehledem dostupných konektorů na modulu převodníku.

Tabulka 1 - Přehled nastavení jumperů na modulu převodníku

Jumper	Popis	Nastavení pro úlohu č. 2	
W1	Volba zdroje AIN_N signálu	1 – 2	Interní
W2	Volba zdroje VREFP napětí	1 – 2	Ze subsystému pro tvorbu referenčního napětí
W3	Volba zdroje VREFN napětí	1 – 2	
W4	Volba zdroje VMID napětí	1 – 2	
W6	Volba připojení VCM napětí	1 – 2	
W5	Volba zdroje AIN signálu	1 – 2	Externí z J6, nebo J1/P1
W7	Volba zdroje hodinového signálu	1 – 2	Interní
W8	Volba napájení digitálního I/O rozhraní	1 – 2	Napájení 3 Vdc
W9	Volba zdroje $\overline{RESET}$ signálu	1 – 2	Přiveden z W10
W10	Volba zdroje $\overline{WR}$ signálu	2 – 3	Manuální, přes SW2
W11	Volba $\overline{DRDY}$ signálu	1 – 2	Přerušení na sestupnou hranu
J8	Nastavení adresy převodníku	1 – 2 3 – 4	Nevyužito, nastaveno na 0xA0000000

Pozn.: Jumper v poloze 1 – 2 = 

Tabulka 2 - Nastavení DIP switche SW1 na modulu převodníku

Pozice	Popis	Nastavení pro úlohu č. 2	
1	FIFO2	0	Úroveň FIFO = 0
2	FIFO1	0	
3	FIFO0	0	
4	Bez přidělené funkce	0	
5	Bez přidělené funkce	0	
6	Bez přidělené funkce	0	
7	Vypnutí převodníku	1	Převodník nevypnut
8	Externí referenční napětí	1	Externí reference

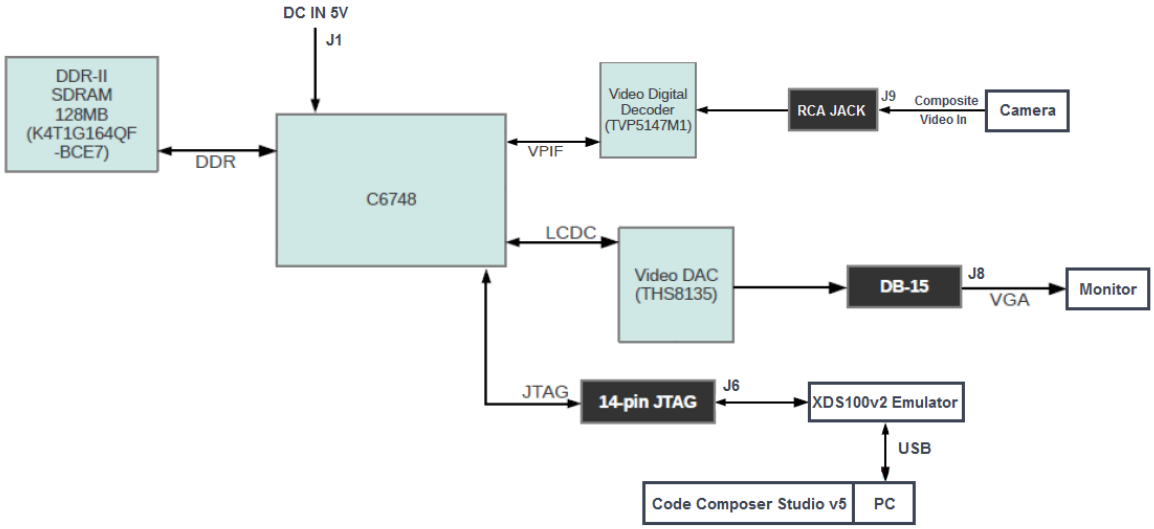
Na modulu převodníku se nachází také tlačítko SW2, které je v úloze používáno pro možnost manuálního resetu převodníku.

Tabulka 3 - Přehled konektorů na modulu převodníku

Konektor	Popis	Použití v úloze č. 2
J1/P1	20 pinů, analogové rozhraní	Vstupní signál AIN (pin 4), AGND (piny 9, 11, 13, 17, 19)
J2/P2	40 pinů, 16 digitálních výstupů	16-bitový výstup (piny 1, 3, 5, ... , 31), DGND (piny 2, 4, 6, ... , 32)
J3/P3	20 pinů, řídící signály	$\overline{CS}$ (pin 1), $\overline{RD}$ (pin 5), $\overline{DRDY}$ (pin 19)
J4/P4	10 pinů, napájení	Viz kap. 4.4, AGND (pin 5), DGND (pin 6)
J5	Externí hodinový signál	Nezapojen
J6	Analogový vstup	Nezapojen
J7	Externí reference	Připojen zdroj napětí 12 Vdc

5.4 Úloha 3 – Cannyho hranový detektor

Úloha Cannyho hranový detektor využívá pouze obvody pro zpracování video signálu. Vstupním zařízením je v této úloze videokamera poskytující kompozitní video signál na vstupu LCDK modulu. Na výstup je stejně jako v první úloze připojen monitor (opět s rozlišením minimálně 640x480 pixelů), v tomto případě použitý pro zobrazení video signálu snímaného videokamerou a zpracovaného procesorem.



Obrázek 8 - Blokové schéma zapojení úlohy Cannyho hranový detektor

Použitá zařízení:

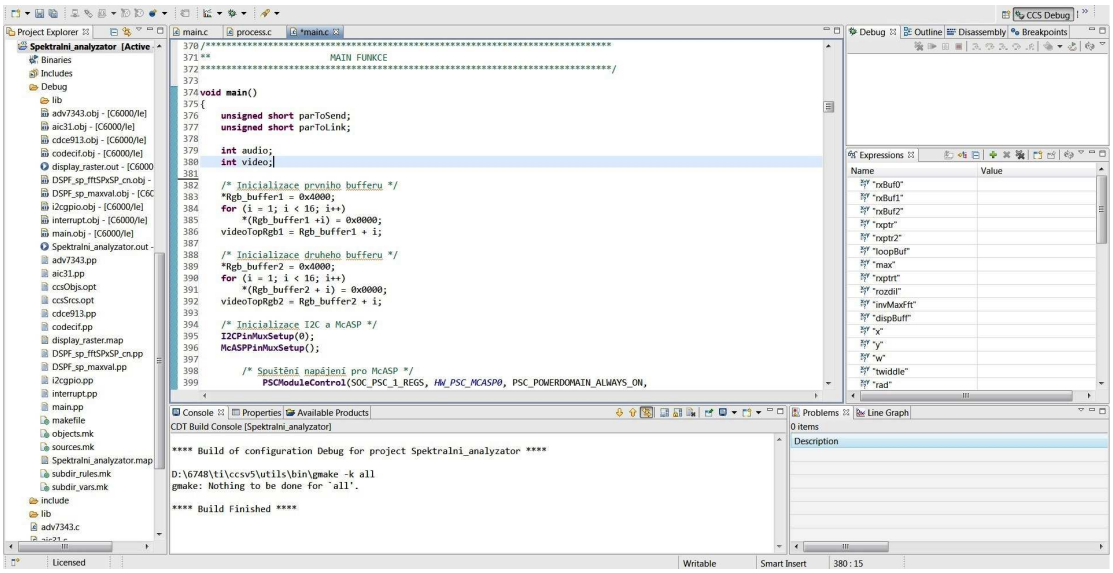
- Modul LCDK C6748 verze A5
- XDS100v2 JTAG emulátor
- Notebook s vývojovým prostředím Code Composer Studio v5.1
- Kamera se senzorem CMOS 1/3“, mikrofonem a s DC napájením 6 - 9V, pracuje v normě PAL-CCIR
- Monitor s rozlišením alespoň 640x480

6 Code Composer Studio v5.1

Předtím, než se začneme věnovat samotnému řešení vybraných úloh, představíme si hlavní nástroj, který využijeme při jejich vytváření a ladění. Tím nástrojem je vývojové prostředí Code Composer Studio ve verzi 5.1.0.09000 založeném na rozšířené platformě Eclipse v3.6. Code Composer Studio (dále také CCS) dovoluje vytvářet aplikace pracující v reálném čase v programovacím jazyce C. Zahrnuje množství funkcí, mj. grafické rozhraní pro psaní programu (editor programu, systémová konzole, okna pro správu projektů, pro ladění programu, pro konfiguraci připojení k cílovým zařízením, pro zobrazení chyb, atd.), dále obsahuje simulátor, kompilátor, assembler, linker (sestavovací program) a podporu ladění programu za chodu.

Aplikace tvořené v CCS jsou strukturovány do projektů. Projekt v CCS obsahuje všechny soubory nebo cesty k souborům, které jsou nutné k vytvoření zkompilovaného spustitelného souboru dané aplikace. Jsou v něm také uloženy informace o tom, jak tyto soubory využít pro správné sestavení spustitelného souboru.

Vytvořené a bez syntaktických chyb přeložené projekty lze nahrát přímo z prostředí CCS přes USB a příslušný JTAG emulátor do paměti procesoru, následně spustit, případně přejít k jejich ladění. CCS obsahuje funkce pro ladění zahrnující sledování proměnných, prohlížení obsahu paměti a registrů, vložení zarážek do programu (breakpoints), grafické znázornění průběhu proměnných a měření doby provedení algoritmů.



Obrázek 9 - Ukázka prostředí Code Composer Studia

V této kapitole se nejprve stručně podíváme na instalaci a základní nastavení CCS. Následně si vypíšeme přehled doplňkových balíčků, které jsou využívány v řešených úlohách a je proto potřeba mít je rovněž nainstalované. V další podkapitole se zaměříme na strukturu projektu, jak se zakládá/importuje a jaké typy souborů obsahuje. Poté si zmíníme několik důležitých nastavení, která je nutné provést pro zprovoznění vybraných úloh. Provedeme nastavení JTAG emulátoru pro připojení LCDK modulu a uvedeme si příklad spuštění programu a jeho ladění. Nakonec si vypíšeme kompletní přehled knihoven pro jednotlivé řešené úlohy.

6.1 Instalace a potřebné doplňky

Vývojové prostředí Code Composer Studio v5 je přiloženo na SD kartě dodávané společně s C6748 LCDK modulem. Případně lze instalační soubor stáhnout z webu Texas Instruments, přehled verzí a odkazy ke stažení na oficiální TI wiki jsou dostupné zde:
http://processors.wiki.ti.com/index.php/Download_CCS

Podporovány jsou operační systémy Windows od verze XP výše a Linux (oficiálně testovány distribuce Ubuntu, Redhat a Fedora). Samotná instalace je intuitivní, doporučeno je instalovat CCS mimo adresář Program Files (pro systémy Windows Vista, 7 a 8), v opačném případě je nutné spouštět CCS jako administrátor. CCS totiž ukládá v základním nastavení konfigurační data do instalačního adresáře. Při prvním spuštění CCS nabízí vybrat si pracovní adresář, se kterým poté standardně pracuje a do nějž se ukládají vytvořené projekty. Takových pracovních adresářů může mít uživatel více, potom lze mezi nimi dle potřeby přepínat.

Pouze čistá instalace vývojového prostředí Code Composer Studio nám ke zprovoznění vytvořených aplikací stačit nebude. Dále si proto uvedeme softwarové balíčky, které také nainstalujeme, protože obsahují vývojové nástroje použité v řešených úlohách.

6.1.1 StarterWare v1.20.03.03

Hlavním vývojovým balíčkem, který využijeme ve všech řešených úlohách je StarterWare v1.20.03.03. Ve StarterWare jsou obsaženy tzv. Device Abstraction Layer (DAL) knihovny a ukázkové aplikace, které demonstrují možnosti použití periferií na modulu C6748 LCDK. Adresář balíčku StarterWare je poměrně obsáhlý, ukážeme si proto jen jeho hlavní části a ty, které dále využijeme:

- *Drivers* – obsahuje zdrojové kódy pro API rozhraní jednotlivých periferií (API - Application Programming Interface - rozhraní pro programování aplikací)
- *Examples* – obsahuje ukázky aplikací, které nastiňují způsob použití ovladačů periferií a knihoven
- *Nandlib* – obsahuje zdrojové kódy a hlavičkové soubory pro knihovnu NAND
- *Build* – obsahuje všechny soubory potřebné k překladu knihoven a ukázkových aplikací (CCS projekty, makefiles – určuje postup při překladu a definuje vztah mezi zdrojovými soubory)
- *Binary* – obsahuje zkompilované knihovny a spustitelné soubory pro všechny ukázkové aplikace, ovladače a knihovny obsažené ve StarterWare
- *Include* – obsahuje hlavičkové soubory pro velkou část knihoven a ovladačů ve StarterWare
- *Platform* – obsahuje zdrojové kódy pro inicializaci některých subsystémů na modulu C6748 LCDK a pro nastavení pinmux (Pin Multiplexing)
- *System_config* – obsahuje systémové zdrojové kódy pro API rozhraní přerušení a API rozhraní vyrovnávací paměti
- *Utils* – obsahuje zdrojové kódy pro další užitečné funkce

Pro řešené úlohy využijeme z balíčku StarterWare především knihovny, ovladače a hlavičkové soubory pro inicializaci, konfiguraci a ovládání některých periferií. Ukázkové aplikace dostupné ve složce Examples představují základ, z něhož vycházejí řešené úlohy (konkrétně ukázkové příklady mcaspPlayBk a vpif_lcd_loopback). Kompletní seznam použitých knihoven a hlavičkových souborů je uveden v kapitole 6.5.

6.1.2 DSPLIB v3.1.0.0

Balíček DSP Library je souhrnem zdrojových kódů běžně používaných a vysoce optimalizovaných funkcí pro zpracování digitálních signálů. Jednotlivé funkce jsou přizpůsobené pro snadnou implementaci a podporují použití procesorů s pevnou i s plovoucí řádovou čárkou. V DSP Library jsou obsaženy zejména tyto funkce:

- Adaptivní filtrace
- Korelace
- Rychlá Fourierova transformace
- Filtrace, konvoluce
- Operace s maticemi

6.1.3 MathLIB v3.0.1.1

Jak název napovídá, balíček Math Library zahrnuje zdrojové kódy funkcí týkajících se matematických výpočtů a funkcí. Funkce z tohoto balíčku jsou optimalizované pro procesory s plovoucí řádovou čárkou a lze je využít i pro náročné aplikace v reálném čase. V Math Library jsou zahrnuty především tyto funkce:

- Trigonometrické funkce (sin, cos, arctan)
- Logaritmus, exponent
- Odmocnina
- Převrácená hodnota
- Dělení

6.1.4 BIOS C6 Software Development Kit v2.0

V BIOS C6SDK v2.0 jsou obsaženy jak všechny výše zmíněné balíčky, tak i mnohé další a k tomu je dodáván v balení modulu C6748 LCDK na přiložené SD kartě. V řešených úlohách jsme použili právě tento soubor knihoven, nástrojů, ovladačů a demonstračních příkladů. Konkrétně nainstalovaný adresář BIOS C6SDK v2.0 obsahuje:

- SYS BIOS 6.33.1.25 RTOS
- Platform Development Kit (PDK)
  - SYS BIOS Drivers
  - Chip Support Register Library (CSLR)
  - StarterWare
  - NDK Support Package (NSP)
- EDMA3 Low Level Driver

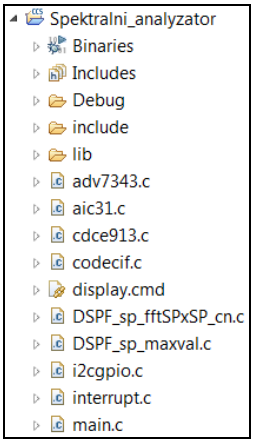
- Interprocess Communication (IPC)
- Network Development Kit (NDK)
- DSP Library (DSPLIB)
- Image Library (IMGLIB)
- Floating Point Math Library (MATHLIB)
- Demonstrační úlohy
- Eclipse RTSC Tools (XDC)

V našich aplikacích využijeme jen poměrně malou část ze všeho, co BIOS C6SDK nabízí. Proto nebudeme popisovat všechny jeho součásti a pro tuto práci se spokojíme s uvedeným seznamem. Více informací o všech uvedených knihovnách a nástrojích lze nalézt v dokumentu bios_c6sdk_2_0_user_guide.pdf, který se nachází v instalačním adresáři BIOS C6SDK. Popis jednotlivých balíčků lze nalézt také v jejich samostatných vydáních, či na oficiální TI wiki.

Odkaz ke stažení BIOS C6SDK na oficiálním TI webu zde:
http://software-dl.ti.com/dsps/dsps_public_sw/c6000/web/bios_c6sdk/latest/index_FDS.html

6.2 Projekt

V CCS zakládáme pro vytvoření nové aplikace nový projekt, případně importujeme stávající do pracovního adresáře, který právě používáme. Založení nového projektu je proto první krok, který po spuštění CCS uděláme. Jednoduše volbou Project > New CCS Project se nám otevře okno, kde zadáme název projektu, výstupní typ zvolíme Executable, vybereme umístění projektu na disku (standardně je nabídnut právě zvolený pracovní adresář), dále zadáme typ procesoru, pro který chceme aplikaci vytvářet, můžeme si otevřít rozšířené nastavení, které si podrobněji probereme o podkapitolu dále a nakonec máme možnost zvolit si z přednastavených šablon projektů. Co se zvoleného procesoru týče, zde vybereme rodinu procesorů – Family C6000 a variantu Generic C674x Device (pokud budeme mít v nabídce konkrétnější možnost, tedy C6748, zvolíme tu). Volbu typ připojení bychom neměli mít možnost po předchozích volbách měnit, proto jeho nastavení provedeme zvlášť po založení projektu (viz kapitola 6.3). Po nastavení všech zmíněných voleb můžeme vše potvrdit a vlevo ve stromu projektů se nám objeví nově vytvořený projekt.



Obrázek 10 - Příklad struktury projektu v CCS studiu

Při importu existujícího projektu je postup jednodušší. Zvolíme File > Import a v otevřeném okně poté vybereme Existing CCS/CCE Eclipse Project. Pokračujeme na další stránku, kde vybereme importovaný projekt a zaškrtneme volbu pro zkopírování dat projektu do pracovního adresáře na disku. Po potvrzení máme projekt importovaný a můžeme s ním začít pracovat. Struktura projektu vytvořeného v CCS5 může vypadat podobně jako na obrázku č. 10.

Projekt v sobě zahrnuje různé typy souborů:

- *Binaries* – obsahuje binární soubor s příponou .out, který slouží k přímému nahrání do paměti procesoru a obsahuje celý přeložený program
- *Includes* – obsahuje hlavičkové soubory (.h), případně jen cesty do adresářů, kde se nacházejí
- *Debug* – obsahuje zdrojové soubory a makefiles, z nichž je kompilován výstupní binární soubor .out
- *Include* – vytvořený uživatelem, obsahuje hlavičkové soubory (.h)
- *Lib* – vytvořený uživatelem, obsahuje zkompilované knihovny (.lib)
- *Zdrojové soubory programu (.c)*
- *Linker command files (.cmd)* – umožňuje definovat systémovou paměť a paměť, do které budou alokované jednotlivé sekce programu, každý projekt musí obsahovat soubor typu .cmd

6.3 Důležitá nastavení

Po založení nového projektu se budeme věnovat nastavení cest k použitým hlavičkovým souborům a knihovnám, dále rozšířenému nastavení projektu a v poslední části této podkapitoly provedeme nastavení připojení LCDK modulu v prostředí CCS.

Všechny hlavičkové soubory a knihovny, které použijeme při psaní programu, musíme buď vložit přímo do adresáře projektu, nebo zadat v nastavení projektu jejich místo uložení. Pro většinu hlavičkových souborů aplikujeme druhý způsob, volíme Project > Properties > CCS Build > C6000 Compiler > Include Options, kam zadáme všechny adresáře, ve kterých se nacházejí do programu zahrnuté hlavičkové soubory (seznam viz kap. 6.5). Pro zahrnutí knihoven (.lib) do projektu postupujeme obdobně, volíme Project > Properties > CCS Build > C6000 Linker > File Search Path a zde vyplníme cesty, kde se knihovny nacházejí a také celé jejich názvy (opět seznam viz kap. 6.5).

Dále se podíváme na další obecné nastavení projektu. Volíme Project > Properties > CCS General, kde je jednak nastavení typu procesoru, který jsme zvolili při založení projektu, a dále rozšířené nastavení čítající 5 dalších voleb. První volbou určíme, jakou použijeme endianitu (pořadí bajtů číselného typu v paměti). Zvolíme little endian (nejméně významný bajt na místo s nejnižší adresou). Druhou volbou je nastavení verze překladače. Pro řešené úlohy byla použita verze TI v7.3.1. Následuje volba výstupního formátu, kde je na výběr EABI (ELF) a legacy COFF (vysvětlení o odstavec níže). Jedná se o důležitou položku, na které závisí funkčnost řešených úloh. Ty jsou stavěny pro EABI (ELF). Čtvrtou položkou je nastavení souboru Linker command file, kam zadáme název .cmd souboru, který používáme při překladu projektu. Poslední položkou je výběr Runtime support (RTS) library, knihovny, kterou používá kompilátor pro implementaci zabudovaných funkcí při běhu programu. V řešených úlohách nemusíme tuto knihovnu zadávat, necháme tedy pole prázdné.

Eabi (ELF) a legacy COFF ABI jsou dvě varianty rozhraní ABI (Application Binary Interface), což je soubor pravidel, dle kterých kompilátor provádí sestavení a propojení samostatných zdrojových souborů a knihoven do spustitelného celku. Patří sem mnoho pravidel, mj. např. formát výstupního souboru (pro Eabi – formát ELF), způsob, jak jsou argumenty předávány funkcím, nebo kolik bitů mají datové typy char, int, long a další. Embedded ABI (EABI), které použijeme v řešených úlohách, je novější ABI, podporované od verze kompilátoru 7.2.x. Toto ABI by postupně mělo plně nahradit dřívější COFF ABI. Hlavní změny EABI oproti COFF ABI jsou např. objektově sdílené soubory ELF, datový typ long je namísto 40-bitový nově 32-bitový a v EABI již není před symboly vkládán znak podtržítka.

Před spuštěním zkompilovaných programů provedeme poslední důležitý krok. Tím je nastavení způsobu připojení k C6748 LCDK modulu. Zobrazíme si okno pro nastavení připojení volbou View > Target Configuration. Zde můžeme definovat připojení pro jednotlivé projekty, nebo nadefinovat uživatelské připojení, které bude použito, pokud je projekt bez speciálního nastavení. Nadefinujeme si nyní uživatelské připojení. V okně pro správu připojení zvolíme New Target Configuration, v novém okně zadáme název připojení např. c6748, umístění necháme předvolené a potvrdíme. Otevře se základní nastavení, kde v položce connection vybereme XDS100v2 USB emulátor a v Board or Device zaškrtneme TMS320C6748. Poté jen uložíme zadanou konfiguraci, v okně správy připojení stiskneme pravé tlačítko na nově vytvořené připojení c6748.ccxml a nastavíme jej jako výchozí volbou Set as Default. Nastavení připojení je nyní hotové.

6.4 Spouštění a ladění programu

Pokud jsme provedli nastavení dle předchozích kapitol a máme napsaný program, který chceme vyzkoušet na LCDK modulu, zkusíme jej nejprve přeložit volbou Project > Build Project. Při úspěšném překladu konzole vypíše hlášku Build Finished a okno se seznamem chyb bude prázdné. V opačném případě nezbývá, než najít vypsanou chybu a opravit ji. Po úspěšném překladu připojíme přes USB a JTAG emulátor modul C6748 LCDK a stiskneme Run > Debug (nebo jen zkratku F11). Tato volba provede kompilaci projektu, připojení LCDK modulu a nahrání projektu do paměti procesoru na LCDK. Po úspěšném nahrání projektu stačí stisknout v okně Debug tlačítko Resume, čímž se spustí běh programu.

Pro pozastavení programu slouží tlačítko Suspend. Pro pokračování běhu programu stačí zvolit znovu Resume, pro restart běhu programu je vhodné nejprve provést systémový reset tlačítkem Reset > System Reset a poté tlačítkem Restart program spustit od začátku. Další volbou je tlačítko Terminate, kterým ukončíme běh programu a odpojíme LCDK modul od CCS. Předtím, než toto provedeme, je opět vhodné provést reset LCDK, abychom předešli problémům při dalším spuštění programu.

Zbylé volby slouží pro ladění programu. Můžeme krokovat program po jednotlivých instrukcích a do zdrojového kódu si můžeme vkládat zarážky, kterými určíme, kde se má běh programu zastavit, abychom mohli dle uvážení např. nahlížet na stav proměnných, registrů, či na obsah paměti.

6.5 Knihovny použité v řešených úlohách

Zde si uvedeme přehled a krátký popis významnějších knihoven použitých při řešení vybraných úloh, především těch, se kterými jsme pracovali přímo v hlavní části programu. Pro jednotlivé úlohy si poté uvedeme seznam adresářů s hlavičkovými soubory, které zaneseme do Include Options. Obdobně uvedeme seznam knihoven, které zaneseme do File Search Path (nastavení obojího viz kap. 6.3).

6.5.1 Přehled důležitých knihoven

Hlavičkové soubory:

- aic31.h - obsahuje funkce pro ovládání kodeku AIC3106
- dspcache.h - poskytuje funkce pro obsluhu cache pamětí
- edma.h - obsahuje funkce pro ovládání EDMA
- edma_event.h - obsahuje výčet EDMA událostí
- emifa.h - obsahuje funkce pro ovládání EMIFA
- gpio.h - obsahuje funkce pro ovládání GPIO
- hw_emifa2.h - obsahuje definice a makra registrů pro EMIFA
- hw_syscfg0_C6748.h - obsahuje hardwarové registry pro modul SYSCFG0
- hw_types.h - obsahuje definice a makra registrů periferií
- i2c.h - obsahuje funkce pro konfiguraci I2C sběrnice
- interrupt.h - poskytuje funkce pro obsluhu přerušení na DSP
- lcdkC6748.h - obsahuje funkce pro nastavení pinmux jednotlivých subsystémů na LCDK
- math.h - zahrnuje optimalizované matematické funkce
- mcasp.h - obsahuje funkce pro ovládání McASP
- psc.h - obsahuje funkce pro uvedení vybraného subsystému do požadovaného stavu
- raster.h - obsahuje funkce pro ovládání LCDC raster řadiče
- soc_C6748.h - obsahuje informace o periferiích pro C6748 SOC
- stdio.h - základní knihovna pro standardní vstup a výstup
- tvp5147.h - obsahuje funkce pro ovládání TVP5147 dekodéru
- vpif.h - obsahuje funkce pro ovládání VPIF řadiče

Zkompilované knihovny

- drivers.lib - obsahuje zdrojové kódy pro API rozhraní periferií
- utils.lib - obsahuje další funkce, např. zpoždění
- platform.lib - obsahuje zdrojové kódy pro inicializaci některých subsystémů na modulu C6748 LCDK a pro nastavení pinmux (Pin Multiplexing)
- system_config.lib - obsahuje systémové zdrojové kódy pro API rozhraní přerušení a API rozhraní vyrovnávací paměti
- vpif_lab.lib - knihovna převzatá od pana Ing. V. Svobody, který v ní zahrnul zdrojové kódy potřebné pro proces zachycení video signálu, uložení do paměti a zobrazení na displeji

6.5.2 Knihovny pro úlohu 1 – Spektrální analyzátor

Include Options:

- Adresář include v pracovním adresáři projektu
- V adresáři balíčku StarterWare:
    - \include
    - \include\c674x
    - \include\c674x\c6748
    - \include\hw
- Adresář bios_6_33_01_25\packages\ti\bios\include

File Search Path:

- drivers.lib
- utils.lib
- platform.lib
- system_config.lib

6.5.3 Knihovny pro úlohu 2 – AM přijímač

Include Options:

- Adresář include v pracovním adresáři projektu
- V adresáři balíčku StarterWare:
    - \include
    - \include\c674x
    - \include\c674x\c6748
    - \include\hw

File Search Path:

- drivers.lib
- utils.lib
- platform.lib
- system_config.lib

6.5.4 Knihovny pro úlohu 3 – Cannyho hranový detektor

Include Options:

- Adresář incl\c674x v pracovním adresáři projektu

File Search Path:

- drivers.lib
- utils.lib
- platform.lib
- img_algs_vs.lib
- system_config.lib
- vpif_lab.lib

7 Úloha č. 1 – Spektrální analyzátor

7.1 Teoretický rozbor

První řešenou úlohou na modulu C6748 LCDK je spektrální analyzátor, tedy aplikace, pomocí níž je možné provádět analýzu vstupního signálu ve frekvenční oblasti. Potřebné zapojení a nastavení Code Composer Studia máme popsané v kapitolách 5.2 a 6. Tato úloha spočívá v zobrazení amplitudového frekvenčního spektra vstupního audio signálu na displeji monitoru. Proto nejprve připojujeme na audio vstup LCDK modulu zdroj audio signálu a na VGA výstup monitor. Ze subsystémů na LCDK modulu využijeme vedle procesoru kodek AIC3106, sériové rozhraní McASP, EDMA, řadič přerušení, PSC, paměť DDR2, LCDC řadič a video DA převodník THS8135.

Předtím, než si ukážeme implementaci úlohy, seznámíme se s procesem zpracování a uložení vstupního audio signálu, podíváme se na rychlou Fourierovu transformaci a řekneme si o procesu zobrazení a obnově snímků na displeji. Kromě těchto 3 základních kroků nutných pro implementaci úlohy budeme v mezikrocích řešit přizpůsobení formátu dat a implementujeme také řadu doplňujících funkcí týkajících se dalšího zpracování vstupního signálu a zkvalitnění prezentace výsledků na monitoru.

7.1.1 Proces zpracování vstupního audio signálu

V úloze spektrální analyzátor se nejprve snažíme dosáhnout toho, abychom měli v paměti buffer, který bude obsahovat data odpovídající aktuálnímu vstupnímu audio signálu, tzn. že bude neustále občerstvován novými daty. Navíc provedeme ukládání dat do paměti pomocí EDMA, abychom co nejméně zatěžovali procesor. Pro přehlednost si uveďme postup, jakým lze proces zpracování a uložení vstupního audio signálu realizovat na LCDK modulu:

- Připojení zdroje audio signálu na vstup LCDK modulu (Line In, nebo MIC)
- Definování vstupních bufferů
- Inicializace I^2C rozhraní pro ovládání kodeku AIC3106 a rozhraní McASP
- Inicializace a konfigurace kodeku AIC3106
- Inicializace a konfigurace rozhraní McASP
- Inicializace a konfigurace rozhraní EDMA
- Inicializace řadiče přerušení
- Nastavení přerušení pro McASP a EDMA
- Konfigurace DMA parametrů
- Konfigurace kodeku AIC3106 a McASP pro I^2S mód
- Aktivace procesu zpracování vstupního signálu a jeho uložení do paměti

Názorněji si jednotlivé body probereme v části zabývající se programovou implementací tohoto procesu. V případě, že se vše povedlo, máme buffer naplněný vstupními daty a chceme s ním dále pracovat. Musíme se proto seznámit s tím, v jakém formátu se data do bufferu uložila a jakých hodnot nabývají. Nesmíme zapomenout také na to, že v závislosti na nastavení kodeku AIC3106 se do něj mohou ukládat data z obou

kanálů, pravého i levého. My ovšem budeme pracovat jen s jedním z nich. Strukturu bufferu si proto ukážeme v kap. 7.2.3. Nyní uvažujme, že máme buffer naplněný souvislými daty z jednoho kanálu představujícími jednotlivé vzorky vstupního signálu. Takový buffer se budeme snažit získat i při programovém řešení úlohy, abychom se mohli věnovat vlastním operacím nad vstupním signálem - provedení váhování signálu oknem, provedení FFT transformace a převedení výsledných dat do podoby vhodné pro zobrazení.

7.1.2 Diskrétní Fourierova transformace (DFT)

Vzhledem k tomu, že v úloze pracujeme s diskrétní reprezentací signálu, jednou z variant pro převod signálu z časové oblasti do frekvenční je diskrétní forma Fourierovy transformace (DFT). Ta pracuje s posloupností reprezentující vstupní signál v časové oblasti x[n] s konečným počtem vzorků N, přičemž počítá pro N vzorků vstupní posloupnosti N hodnot frekvenčního spektra.

Pro přímou DFT platí vztah:

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n] \cdot e^{\frac{-j2\pi nk}{N}},\ 0 \le n \le N-1$$

kde, $x[n]$ je posloupnost reprezentující signál v časové oblasti, $X[k]$ je posloupnost vzorků spektra a N je délka transformace.

Spektrum vypočtené spojitou i diskrétní Fourierovou transformací je komplexní s krokem Δf (rozlišení spektra). Obsahuje reálnou i imaginární část:

$$X_k = re\{X_k\} + j \cdot im\{X_k\} = |X_k| \cdot e^{-j\varphi_k},$$

kde φ_k je fáze komplexního spektra. Pokud posouváme signál v čase, mění se fáze komplexního spektra.

Rozlišení ve spektru je dáno vzdáleností sousedních čar:

$$\Delta f = \frac{1}{N \cdot \Delta t} = \frac{f_S}{N},$$

kde N je délka transformace, f_S je vzorkovací frekvence a Δt je vzorkovací perioda.

Vztah pro maximální frekvenci:

$$f_{MAX} = f_N = \frac{1}{2 \cdot \Delta t} = \frac{f_S}{2} = \frac{N}{2} \cdot \Delta f$$

V úloze pracujeme s reálnou podobou vstupního signálu. V takovém případě platí sudá symetrie v reálných hodnotách spektra X_k ($re\{X_k\} = re\{X_{N-k}\}$) a lichá symetrie v imaginárních hodnotách spektra X_k ($im\{X_k\} = -im\{X_{N-k}\}$). Hodnoty X_k v rozsahu $\langle f_N, f_S \rangle$ jsou komplexně sdružené s hodnotami X_k v rozsahu $\langle 0, f_N \rangle$. Je proto možné ve výsledku použít jen část $\langle 0, f_N \rangle$ a pracovat tak s tzv. jednostranným spektrem (způsob, kterým je řešena implementace úlohy).

7.1.3 Rychlá Fourierova transformace (FFT)

Dostáváme se k transformaci, která je použita v této úloze, k rychlé Fourierově transformaci (FFT). Ta přináší v podstatě stejné výsledky jako DFT transformace, ovšem její výhodou je výrazně nižší výpočetní náročnost a přizpůsobení pro architekturu DSP procesorů. Provedení DFT transformace představuje výpočetní náročnost aritmetických operací kvadraticky závislou na N - $O(N^2)$. Oproti tomu FFT transformace redukuje počet násobení na složitost O(N/2 log N) a oproti DFT je mnohem efektivnější (např. při N = 1024 je potřebný počet operací pro FFT 205x nižší než při použití DFT).

FFT transformace je založena na využití periodicity a symetrii exponenciály ($W_N = e^{-j\frac{2\pi}{N}}$):

$$W_N^{kn} = W_N^{(k+N)n}$$
$$W_N^{k(n+\frac{N}{2})} = -W_N^k$$

FFT transformace typu DIT (decimation in time) probíhá následovně:

- Vstupní posloupnost x[n] musí být délky 2^n
- Postupně probíhá rozklad vstupního signálu na sudou $x_1[n]$ a lichou $x_2[n]$ posloupnost:

$$x_1[n] = x[2n]$$
$$x_2[n] = x[2n+1]$$

- DFT transformace ze dvou dílčích sudých a lichých posloupností potom vypadá takto:

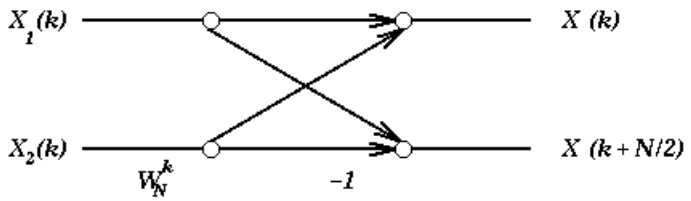
$$X(k) = \sum_{n=0}^{N/2-1} x_1[n] \cdot W_N^{2nk} + \sum_{n=0}^{N/2-1} x_2[n] \cdot W_N^{(2n+1)k} =$$
$$= \sum_{n=0}^{N/2-1} x_1[n] \cdot W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} x_2[n] \cdot W_{N/2}^{nk} =$$
$$= X_1(k) + W_N^k \cdot X_2(k)$$

- Tedy výsledné spektrum lze získat sečtením spekter sudé a liché posloupnosti. Tyto posloupnosti můžeme dále a dále dělit na další podružné sudé a liché posloupnosti, až zůstanou v každé jednotlivé dílčí posloupnosti pouze 2 prvky (proto musí být vstupní posloupnost délky 2^n), jejichž DFT získáme dle vztahu pro tzv. motýlek:

$$A = a + W_N^k \cdot b$$
$$B = a - W_N^k \cdot b$$

Obecně:

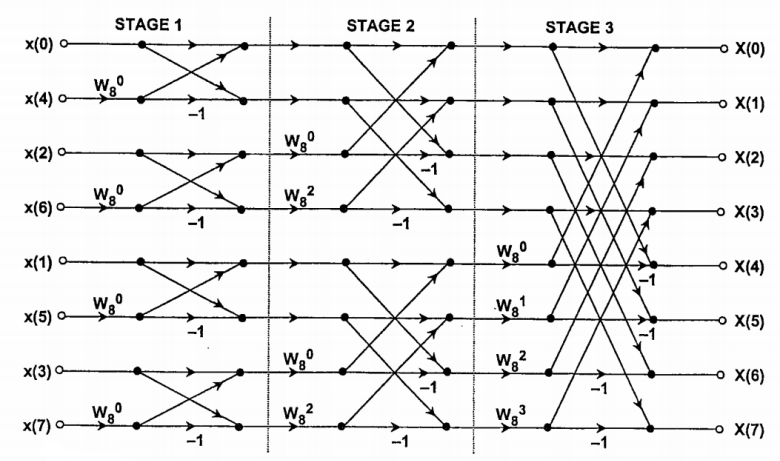
$$X(k) = X_1(k) + W_N^k \cdot X_2(k)$$
$$X(k + N/2) = X_1(k) - W_N^k \cdot X_2(k)$$



Obrázek 11 - Grafické znázornění motýlku

- Znázorněný motýlek obsahuje dva komplexní součty a jeden komplexní součin.

- Motýlek tvoří základ FFT algoritmu. Na obrázku 12 je ukázka FFT DIT algoritmu pro N=8. Na tomto příkladě je FFT složena ze 3 stupňů a pořadí na vstupu je dáno bitovou reverzací vzorků.



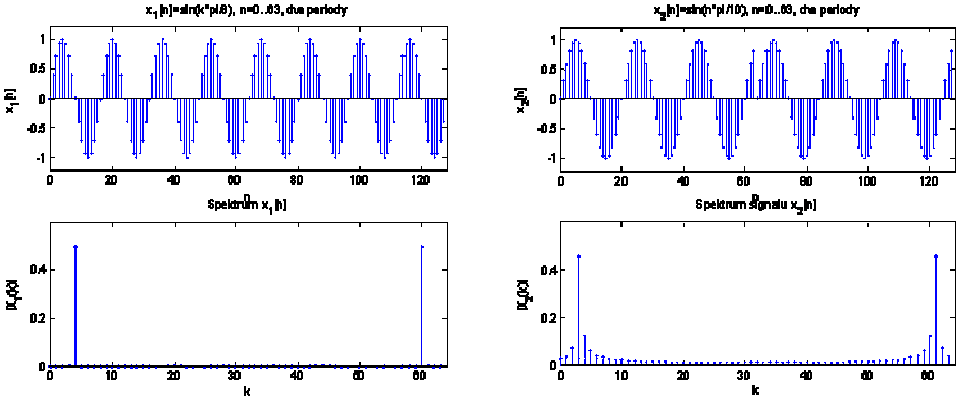
Obrázek 12 - Algoritmus FFT DIT pro N=8

Jednotlivé stupně se dělí na sekce, přičemž platí, že n-tý stupeň rozkladu obsahuje $2^{\log_2 N - n}$ sekcí (v příkladu jsou např. v prvním stupni 4 sekce). Sekce obsahují dle stupně rozkladu 2^{n-1} motýlků (v příkladu např. ve 2. stupni obsahuje sekce 2 motýlky). Vynásobením sekcí a motýlků získáme počet motýlků v daném stupni rozkladu.

V programovém řešení je potom možné zobrazený algoritmus implementovat tak, že motýlek je samostatná funkce volaná z každé sekce v jednotlivých stupních rozkladu FFT. Hlavní část FFT funkce řeší přípravu dat a jejich setřídění pro každý stupeň rozkladu a zajišťuje volání funkce motýlek.

7.1.4 Váhové funkce

Váhové funkce slouží k potlačení jevu, kterému se říká prosakování ve spektru. To lze vysvětlit na příkladu harmonického signálu. Pokud je tento signál periodický a délka tohoto signálu je celistvým násobkem jeho periody, potom k prosakování ve spektru nedochází. Pokud je však délka signálu necelistvým násobkem jeho periody, k prosakování ve spektru dochází, tzn. že dojde k rozprostření energie části spektra do okolí hlavní spektrální čáry (viz srovnání na obr. 13).



Obrázek 13 - Spektrum a) bez vlivu prosakování, b) s prosakováním ve spektru

Řešením nežádoucího jevu prosakování ve spektru může být zarovnání signálu na násobek periody, což ovšem není příliš praktické. Proto se používají váhové funkce, tedy datové posloupnosti, kterými signál násobíme před provedením Fourierovy transformace:

$$s_v(n) = s(n) \cdot w(n),$$

kde $s_v(n)$ je váhovaný signál, s(n) původní signál a w(n) je použitá váhová funkce

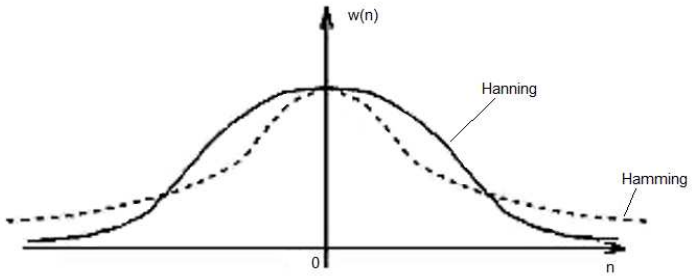
Ve spektru to odpovídá konvoluci původního spektra a spektra váhové funkce. Existuje mnoho druhů váhovacích oken. Nejzákladnějším je obdélníkové okno délky N, které nahradí všechny vzorky za N-tým vzorkem nulami. To vede k velmi dobrému spektrálnímu rozlišení, ovšem protože má spektrum pravoúhlého okna malý odstup postranních laloků (13,3dB), může dojít k nežádoucímu maskování signálů s malou amplitudou. Mnohem lepší odstup postranních laloků za cenu horšího spektrálního rozlišení má např. často používané Hammingovo okno (odstup hlavního od postranního laloku - 42,5dB), příp. Hanningovo okno (odstup hlavního od postranního laloku 31,5dB). Obě jsou součástí řešení úlohy spektrální analyzátor.

Funkce pro Hammingovo okno:

$$w(n) = \begin{cases} 0{,}54 + 0{,}46\cos(n\pi / L), & -L \le n \le L, \\ 0, & jinde. \end{cases}$$

Funkce pro Hanningovo okno:

$$w(n) = \begin{cases} 0{,}5 + 0{,}5\cos(n\pi / L), & -L \le n \le L, \\ 0, & jinde. \end{cases}$$



Obrázek 14 - Hammingovo a Hanningovo okno

7.1.5 Proces zobrazení snímků na displeji

Pokud máme na vstupní data aplikovanou rychlou Fourierovu transformaci a máme je uložena v normalizované podobě vhodné k vykreslení na připojeném monitoru, čeká nás poslední úkol, kterým je konfigurace výstupního rozhraní LCDK modulu pro zobrazení požadovaných snímků na displeji. Formát dat vhodný pro zobrazení znamená, že výsledek z FFT transformace jsme normalizovali a poskládali takovým způsobem, aby odpovídal použitému rozlišení displeje a aby výsledné zobrazení grafu vhodně využívalo zobrazovací prostor. Graf zobrazujeme v černobílém provedení (graf a popisky bílé, pozadí černé), proto potřebujeme jen hodnoty bílé a černé pro nastavení barvy pixelu a nemusíme se v této úloze zabývat video formátem výstupních dat (ten je vysvětlen v úloze Cannyho hranový detektor). Zobrazovací funkce je uvedena v kap. 7.2.4.10. Nyní se podívejme na postup, jakým lze realizovat proces zobrazení na LCDK modulu:

- Připojení monitoru na VGA výstup LCDK modulu
- Definování výstupních bufferů
- Inicializace a konfigurace LCDC raster řadiče
- Nastavení přerušení pro LCDC řadič
- Konfigurace parametrů pro DMA přenos
- Aktivace přenosu
- Aktivace přerušení na konci každého snímku, požadavek na nový

Názorněji si jednotlivé body opět probereme v části zabývající se programovou implementací tohoto procesu. Pokud je vše správně, na monitoru se nám zobrazí jeden nebo dva grafy dle zvolené funkce (viz implementace) a periodicky se obnovují dle změn vstupního audio signálu.

7.2 Implementace

Nyní se dostáváme k samotnému programovému řešení úlohy spektrální analyzátor. Začneme strukturou projektu úlohy a vývojovým diagramem hlavní části programu. Dále popíšeme programové řešení zpracování audio signálu a formát zvukových dat ukládaných do paměti. Poté se budeme věnovat vysvětlení a ukázce vytvořených funkcí, ukážeme si programové řešení procesu zpracování dat pro jejich zobrazení na monitoru a na závěr provedeme testování a ukázku vytvořeného programu pro zvolenou konfiguraci parametrů.

7.2.1 Struktura programu

Hlavní část zdrojového kódu úlohy spektrální analyzátor je uložena v souboru **main.c**, ovšem projekt úlohy obsahuje i další podpůrné zdrojové soubory:

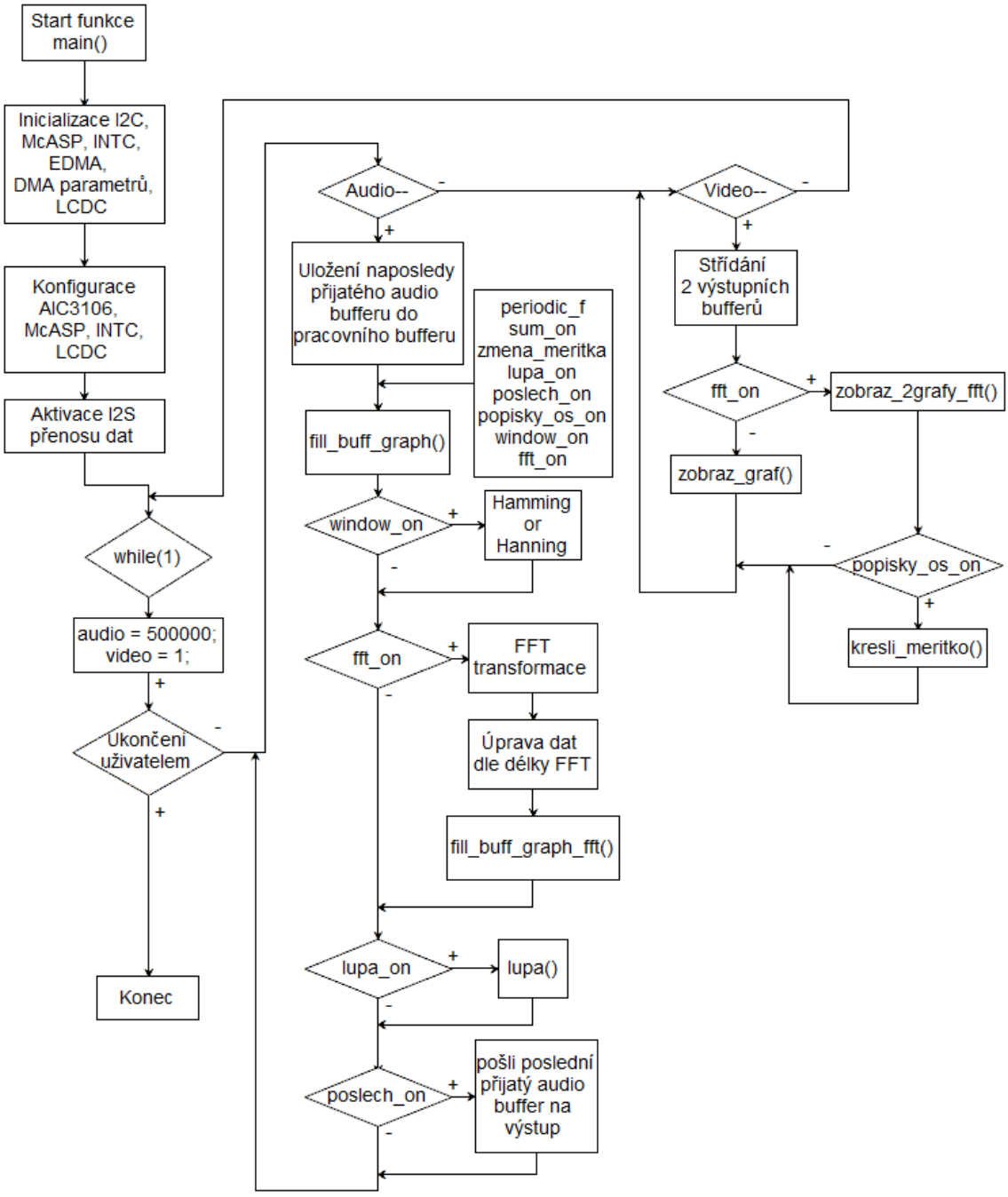
- **aic31.c** – obsahuje definici registrů a funkcí pro řízení kodeku AIC3106
- **codecif.c** – zahrnuje funkce pro konfiguraci I2C rozhraní, přes které je ovládán kodek AIC3106
- **DSPF_sp_fftSPxSP_cn.c** – funkce z knihovny DSPLIB pro výpočet komplexní dopředné FFT mixed radix transformace s bitovou reverzací (viz kap. 7.2.4.3)
- **DSPF_sp_maxval.c** – funkce z knihovny DSPLIB, která ze vstupního pole vrací prvek s nejvyšší hodnotou
- **interrupt.c** – obsahuje funkce pro správu událostí přerušení

Na obrázku 15 je znázorněn vývojový diagram konečné podoby programu v hlavní funkci `main()`.

Podíváme-li se na vývojový diagram, vidíme, že v hlavní funkci `main()`, která začíná inicializací a konfigurací jednotlivých subsystémů, se nachází v nekonečném cyklu `while(1)` další dva vnořené while cykly s podmínkou danou parametry pojmenovanými audio a video. První while cyklus provádí zachycení přijatých audio dat, jejich uložení do pracovního bufferu a následné zpracování. Provedení funkcí volaných v tomto cyklu je závislé na parametrech typu bool (true, nebo false) zadaných před spuštěním programu (např. `window_on`, `fft_on`, atd.). Druhý while cyklus provádí zobrazení zpracovaných dat na monitoru.

Při opakování nekonečného cyklu `while(1)` se na jeho začátku vždy nastaví parametry audio a video na zadanou hodnotu (viz diagram na obr. 15), což způsobí, že cyklus s parametrem audio je proveden mnohonásobně vícekrát než cyklus s parametrem video. Základním předpokladem je totiž práce s celými buffery. Na začátku máme plný buffer, jehož velikost lze zadat (s ohledem na zvolenou délku FFT transformace). Celý jeho obsah zkopírujeme do pracovního bufferu, nad nímž provádíme další operace a výsledek zobrazujeme v grafu. Důležité je, že pracujeme s celými buffery vždy najednou, tzn. že i na monitoru vždy zobrazujeme kompletně nové výsledky. Musíme proto ošetřit periodu obnovování grafu na monitoru, abychom zajistili čitelnost zobrazovaných dat pro pozorovatele. To je řešeno právě jednoduchým poměrem provedení zmíněných dvou while cyklů. V základním nastavení se počítá s jedním provedením cyklu video a definovatelným počtem provedení cyklu audio. Potom čím větší je hodnota parametru audio, tím delší je perioda obnovy zobrazených grafů. V diagramu

na obrázku 15 jsou zadané experimentálně ověřené hodnoty parametrů audio a video, při nichž se zobrazené grafy obnovují s periodou 0,4s.



Obrázek 15 - Vývojový diagram úlohy 1 - spektrální analyzátor

Toto řešení je optimální pouze pro zkoumání periodických vstupních signálů, pro které je navíc implementována funkce synchronizace časové základny. Nevadí nám proto, že zobrazujeme vždy jen x-tý buffer, protože výsledky se liší jen mírně vlivem neperiodických vlivů (např. šum). U neperiodických signálů je zjevnou nevýhodou, že nezobrazujeme vstupní výsledky kontinuálně pro vstupní signál. Zobrazujeme vždy jen výsledky pro x-tý buffer a přicházíme tak o analýzu množství vstupních dat. Vzhledem k tomu, že proces zpracování audio signálu je mnohem rychlejší, než následné zobrazení

výsledků (i s ohledem na nutné zdržení pro čitelnost dat na monitoru), je nutné pro kontinuální analýzu neperiodických dat využít paměť. Tedy více pracovních bufferů, do kterých postupně ukládáme výsledky ze zpracovaných dat a poté je postupně se zadanou periodou zobrazujeme na monitoru. Finální podoba k práci přiložené úlohy takové řešení neobsahuje a je zaměřena především na zobrazení periodických funkcí.

7.2.2 Zachycení vstupních audio dat

V kódu programu je definována celá řada proměnných a konstant. Mnohé z nich lze konfigurovat a měnit tak parametry a chování daných částí programu. Nebudeme se zde věnovat jejich výpisu, postupně si však některé z nich zmíníme při popisu programových funkcí. Případně lze pro jejich přehled nahlédnout do zdrojového souboru main.c, na jehož začátku jsou definovány a okomentovány. Hlavní funkce `main()` začíná voláním procedur pro inicializaci a konfiguraci všech potřebných LCDK subsystémů. Nyní se proto budeme podrobněji věnovat této části programu - konkrétně zprovoznění procesu zpracování a uložení audio dat do paměti. Dále uvedené funkce obsahují parametry a registry, jejichž hodnoty jsou nadefinovány v použitých knihovnách. Obecně se parametry funkcí pomocí číselných hodnot nezadávají a jsou skryty za slovně vyjádřenými proměnnými. V dalším výkladu se proto budeme držet tohoto zápisu (pro zjištění hodnot je nutné nahlédnout do příslušné knihovny).

První funkce volané z `main()` jsou `I2CPinMuxSetup(0)` a `McASPPinMuxSetup()` z knihovny lcdkC6748.h. Těmito funkcemi zvolíme, které piny procesoru budou použity pro sběrnici I2C a které pro McASP, což nám umožní s těmito periferiemi dále pracovat.

Následují funkce pro spuštění a nastavení požadovaného stavu periferií McASP a také EDMA3. Pro obojí je funkce totožná, pouze s jinými parametry. O spuštění McASP se stará funkce z knihovny psc.h:

```
PSCModuleControl(SOC_PSC_1_REGS,HW_PSC_MCASP0,PSC_POWERDOMAIN_ALWAYS_ON,
                                 PSC_MDCTL_NEXT_ENABLE);
```

`SOC_PSC_1_REGS`	základní adresa PSC mapovaných registrů
`HW_PSC_MCASP0`	číslo modulu (pro McASP = 7)
`PSC_POWERDOMAIN_ALWAYS_ON`	nastavení stálého napájení modulu
`PSC_MDCTL_NEXT_ENABLE`	nastavení příznaků modulu

Inicializaci I2C rozhraní pro kodek AIC31 zajistí funkce z knihovny codecif.h:

```
I2CCodecIfInit(SOC_I2C_0_REGS,INT_CHANNEL_I2C, I2C_SLAVE_CODEC_AIC31);
```

`SOC_I2C_0_REGS`	základní adresa registrů použitých pro kodek
`INT_CHANNEL_I2C`	číslo kanálu pro nastavení systémového přerušení
`I2C_SLAVE_CODEC_AIC31`	slave adresa kodeku na rozhraní I2C

Inicializujeme také EDMA3 řadič pomocí funkce z knihovny edma.h:

```
EDMA3Init(SOC_EDMA30CC_0_REGS, 0);
```

`SOC_EDMA30CC_0_REGS`	základní adresa registrů použitých pro EDMA3
`0`	číslo fronty, ke které jsou mapovány žádosti DMA Master kanálu

Inicializaci INTC řadiče přerušení obstarává funkce **`IntDSPINTCInit()`** z knihovny interrupt.h. Tato funkce zajistí, aby byla všechna maskovatelná přerušení na začátku vypnuta. V dalších krocích je provedeno nastavení požadovaných přerušení voláním funkcí **`EDMA3IntSetup()`** a **`McASPErrorIntSetup()`.** Obě funkce se nacházejí v hlavním zdrojovém souboru main.c a obsahují totožnou proceduru lišící se pouze zadanými parametry:

```
IntRegister(C674X_MASK_INT6, McASPErrorIsr);
IntEventMap(C674X_MASK_INT6, SYS_INT_MCASP0_INT);
IntEnable(C674X_MASK_INT6);
```

Pro nastavení přerušení na EDMA3 i na McASP jsou použity 3 funkce z knihovny interrupt.h. První funkce přiřadí zadanou obsluhu přerušení (`McASPErrorIsr`) k jednomu z maskovatelných přerušení (`C674X_MASK_INT6`). Druhá funkce provádí mapování vybrané systémové události (`SYS_INT_MCASP0_INT`) do maskovatelného přerušení procesoru (`C674X_MASK_INT6`). Třetí funkcí je zvolené přerušení procesoru zapnuto.

Pro zprovoznění chodu přerušení je ještě nutné globální povolení přerušení v procesoru. To provedeme funkcí **`IntGlobalEnable()`** opět z knihovny interrupt.h.

Po nastavení obsluhy přerušení je prováděna žádost o přidělení DMA kanálu pro příjem a DMA kanálu pro vysílání. Toto provádí funkce z knihovny edma.h (příklad pro vysílací kanál):

```
EDMA3RequestChannel(SOC_EDMA30CC_0_REGS,EDMA3_CHANNEL_TYPE_DMA,
                    EDMA3_CHA_MCASP0_TX, EDMA3_CHA_MCASP0_TX, 0);
```

`SOC_EDMA30CC_0_REGS`	základní adresa registrů použitých pro EDMA3
`EDMA3_CHANNEL_TYPE_DMA`	typ kanálu, v našem případě DMA
`EDMA3_CHA_MCASP0_TX`	číslo kanálu pro požadovanou událost
`EDMA3_CHA_MCASP0_TX`	číslo kanálu, dle kterého je generováno přerušení po úspěšném, či neúspěšném dokončení přenosu
`0`	číslo fronty, ke které jsou mapovány žádosti DMA Master kanálu

Nyní se v kódu dostáváme k volání čtyř funkcí, které se nacházejí v hlavním zdrojovém souboru main.c a které se postupně starají o inicializaci DMA parametrů, o konfiguraci kodeku AIC31, o konfiguraci rozhraní McASP a o aktivaci procesu příjmu

a vysílání dat na I2S. Vzhledem k tomu, že tyto funkce jsou již obsáhlejší a jejich součástí je množství dalších procedur, zaměříme se především na popis jejich činnosti a na důležité části.

7.2.2.1 Funkce I2SDMAParamInit()

První funkce slouží k inicializaci DMA parametrů, které jsou součástí kanálového kontroléru EDMA3CC. Je využita struktura EDMA3CCPaRAMEntry z knihovny edma.h, jenž obsahuje deklaraci potřebných DMA parametrů a poskytuje vhodný formát pro jejich uživatelskou konfiguraci. Níže je uveden kód definice parametrů v EDMA3CCPaRAMEntry před začátkem běhu programu. V tomto případě se jedná o definici parametrů pro EDMA přenos z rozhraní McASP do paměti RAM, tedy pro příjem dat. Obdobně je definována struktura pro odesílání dat.

```
static struct EDMA3CCPaRAMEntry const rxDefaultPar =
{     (unsigned int)(EDMA3CC_OPT_DAM | (0x02<<8u)), /* Opt field */
      (unsigned int) SOC_MCASP_0_DATA_REGS, /* source address */
      (unsigned short)(BYTES_PER_SAMPLE), /* aCnt */
      (unsigned short)(1), /* bCnt */
      (unsigned int) rxBuf0, /* dest address */
      (short) (0), /* source bIdx */
      (short)(BYTES_PER_SAMPLE), /* dest bIdx */
      (unsigned short)(PAR_RX_START*SIZE_PARAMSET),/*link address */
      (unsigned short)(0), /* bCnt reload value */
      (short)(0), /* source cIdx */
      (short)(0), /* dest cIdx */
      (unsigned short) 1 /* cCnt */
}
```

`opt`	OPT pole setu parametrů
`srcAddr`	byte adresa zdroje
`aCnt`	počet bytů v každém vzorku
`bCnt`	počet vzorků v každém slotu
`destAddr`	byte adresa cíle
`srcBIdx`	index mezi po sobě následujícími vzorky zdroje
`destBIdx`	index mezi po sobě následujícími vzorky cíle
`linkAddr`	adresa pro propojení (automatické přehrání setu parametrů)

Funkce **`I2SDMAParamInit`**() provádí potřebné změny některých parametrů v EDMA3CCPaRAMEntry dle zvolených vstupních podmínek, aby mohl být uskutečněn první EDMA přenos pro příjem i pro vysílání. Do těchto podmínek patří zejména konfigurovatelné proměnné - počet vzorků na jeden přenášený buffer, počet bytů na jeden vzorek, počet bufferů pro příjem a počet bufferů pro vysílání.

V úloze spektrální analyzátor je proměnná počet vzorků na jeden přenášený buffer z důvodu dalších operací stanovena jako pětinásobek zvolené velikosti FFT transformace (např. pro `FFTSIZE = 512` je rovna 2560 vzorkům). Jeden vzorek se v úloze skládá ze 4B (délka jednoho slova je 16b a je přenášen pravý i levý kanál). Přijímací buffery jsou v úloze použity 3 a stejný počet je i bufferů vysílacích.

Popisovaná funkce začíná převzetím počátečních parametrů ze struktury rxDefaultPar (viz definice výše) a jejich zkopírováním do základního setu parametrů, což je na C6748 DMA kanál 0 definovaný pro příjem z rozhraní McASP. Základní set je inicializován k přijetí dat bufferem `rxBuf0` (jeden ze 3 bufferů pro příjem). V dalším kroku je povoleno DMA přerušení pro zahájení nového přenosu po skončení předchozího a základní set parametrů je spojen s propojeným setem parametrů začínajícím na hodnotě parametru `PAR_RX_START` (v této úloze roven 40). Následující příjem již nastane pouze přes takto propojené sety parametrů (v této úloze jsou 2 sety propojených parametrů 40 a 41 pro příjem). V prvním propojeném setu parametrů dojde k příjmu druhého vzorku pouze, pokud je již celý první vzorek uložen v bufferu `rxBuf0`, třetí vzorek je přijmut pouze, pokud je celý druhý vzorek uložen v bufferu `rxBuf1`, atd.

Podobným způsobem probíhá inicializace DMA parametrů pro posílání dat z paměti na McASP. DMA kanál je v tomto případě 1 a základní set je inicializován pro odesílání z loop bufferu, jehož velikost lze konfigurovat. Základní set parametrů je spojen s propojeným setem parametrů začínajícím na hodnotě parametru `PAR_TX_START` (v této úloze roven 42). Všechny další sety parametrů pro vysílání jsou propojeny samy na sebe (narozdíl od setů parametrů pro příjem, které jsou propojeny vždy na jiný PaRAM set).

7.2.2.2 Funkce AIC31I2SConfigure()

Tato funkce slouží ke konfiguraci kodeku AIC31, k čemuž využívá funkcí z knihovny aic31.h prostřednictvím rozhraní I2C. Nejprve je volána funkce pro reset kodeku a jeho uvedení do výchozího stavu. Dále je provedena konfigurace formátu dat pro operace kodeku (v úloze zvolen typ I2S pro I2S mód kodeku) a také konfigurace šířky slotu (v úloze nastavena na implicitně 16b). V pořadí třetí zde volaná funkce dovoluje volbu, které ze dvou sekcí ADC/DAC budou na kodeku využívány (v úloze nastaven režim both tedy použití obou sekcí). Taktéž lze prostřednictvím posledního parametru této funkce měnit vzorkovací frekvenci kodeku (v úloze mezi 8 – 96 kHz). Poslední dvě funkce slouží k inicializaci ADC a DAC sekce kodeku. Níže je uvedena ukázka procedury provádějící popsanou činnost.

```
AIC31Reset(SOC_I2C_0_REGS);

while(delay--);

AIC31DataConfig(SOC_I2C_0_REGS, AIC31_DATATYPE_I2S, SLOT_SIZE, 0);
AIC31SampleRateConfig(SOC_I2C_0_REGS,AIC31_MODE_BOTH,SAMPLING_RATE);
AIC31ADCInit(SOC_I2C_0_REGS);
AIC31DACInit(SOC_I2C_0_REGS);
```

7.2.2.3 Funkce McASPI2SConfigure()

Konfiguraci sériového rozhraní McASP obstarává funkce, která obsahuje mnoho dalších vnořených funkcí z knihovny mcasp.h. Začíná stejně jako u předchozí funkce resetem rozhraní McASP a nastavením formátu dat pro příjem/vysílání

na formát I2S se stanovením délky slova (16b) a délky slotu (16b). Následuje funkce pro konfiguraci synchronizačního signálu pro příjem/vysílání. Tato funkce má nastaven režim I2S, šířku synchronizačního signálu rovnou délce slova a lze si zvolit, zda má být synchronizace na náběžnou, nebo na sestupnou hranu. Dalších několik funkcí zajišťuje výběr a konfiguraci systémových hodin pro přijímací a posléze i pro vysílací sekci. Vybrat lze interní hodiny na McASP, externí hodiny procesoru, příp. lze zvolit kombinace obou. V této úloze jsou použity externí hodiny. U hodin je stanovena polarita určující, zda budou data přijímána/vysílána na náběžnou, nebo na sestupnou hranu a lze dělit hodnotu systémových hodin jednou z předdefinovaných hodnot (v úloze 32). Po konfiguraci hodin následuje spuštění synchronizace mezi přijímací a vysílací sekcí.

Inicializace rozhraní McASP pokračuje nastavením vysílacích a přijímacích slotů, které jsou 2 v případě módu I2S. Konfigurují se serializéry, jeden jako přijímací a jeden jako vysílací. Důležitou částí jsou funkce pro nastavení činnosti vstupních a výstupních pinů McASP. Na vstupní piny jsou nastaveny systémové hodiny, synchronizační signál a serializér pro příjem dat. Na výstupní piny je nastaven serializér pro výstup, který je připojen na vstup kodeku AIC31.

Poslední částí konfigurace McASP je nastavení přerušení vzniklých chybami na přijímací, nebo na vysílací sekci. To zajišťuje funkce (zobrazena níže), jejíž hlavní parametr se může skládat z několika subparametrů stanovujících podmínky, při kterých dojde k přerušení. Jedná se především o problémy typu chyba při DMA přenosu, selhání hodin, či selhání synchronizace.

```
McASPRxIntEnable   (SOC_MCASP_0_CTRL_REGS,    MCASP_RX_DMAERROR
                                            | MCASP_RX_CLKFAIL
                                            | MCASP_RX_SYNCERROR
                                            | MCASP_RX_OVERRUN);
```

7.2.2.4 Funkce I2SDataTxRxActivate()

Poslední z této čtveřice funkcí aktivuje příjem a odesílání dat na rozhraní McASP. Před voláním této funkce je nutné mít připravené DMA parametry, mít nakonfigurovaný kodek AIC31 a rozhraní McASP. Tělo této funkce vypadá následovně:

```
McASPRxClkStart(SOC_MCASP_0_CTRL_REGS, MCASP_RX_CLK_EXTERNAL);
McASPTxClkStart(SOC_MCASP_0_CTRL_REGS, MCASP_TX_CLK_EXTERNAL);

EDMA3EnableTransfer(SOC_EDMA30CC_0_REGS, EDMA3_CHA_MCASP0_RX,
                            EDMA3_TRIG_MODE_EVENT);
EDMA3EnableTransfer(SOC_EDMA30CC_0_REGS,EDMA3_CHA_MCASP0_TX,
                            EDMA3_TRIG_MODE_EVENT);

McASPRxSerActivate(SOC_MCASP_0_CTRL_REGS);
McASPTxSerActivate(SOC_MCASP_0_CTRL_REGS);

McASPRxEnable(SOC_MCASP_0_CTRL_REGS);
McASPTxEnable(SOC_MCASP_0_CTRL_REGS);
```

Nejprve jsou spuštěny externí hodiny pro přijímací i vysílací sekci, dále je povolen EDMA pro přenos, jsou aktivovány serializéry na McASP a na závěr je aktivováno rozhraní McASP pro příjem a pro vysílání.

7.2.3 Buffer a formát zvukových dat

Pro práci se zvukovými daty uloženými do paměti a pro jejich rychlé zpracování a přesouvání je využívána struktura bufferů. Buffer je paměťový prostor o definované velikosti, který je často využíván pro dočasné uložení dat ze vstupních zařízení před jejich zpracováním a přesunem na další místo. V této úloze jsou definovány 3 vstupní a 3 výstupní audio buffery o velikosti dané proměnnou `AUDIO_BUFF_SIZE` (počet vzorků na jeden buffer). Například pro zpracování vstupních dat je vždy využit poslední naplněný vstupní buffer z trojice `rxBuf0`, `rxBuf1` a `rxBuf2`. Poté, co jsou data z tohoto bufferu přijata, je připraven k naplnění dalšími daty ze vstupu prostřednictvím EDMA. Buffer tedy nemusí být a většinou ani není jen jeden, což je výhodné především pro urychlení zpracování dat. Zatímco v jednom z bufferů právě probíhá zpracování dat, další je v tu samou chvíli plněn novými vstupními daty. Výhodou také je, že k bufferům nemusíme přistupovat pomocí individuálního adresování na každé paměťové místo bufferu zvlášť. Stačí pouze znát ukazatel na začátek bufferu v paměti a pozici, na které se nacházejí požadovaná data (v našem případě posloupnost audio vzorků).

V programu je pro tři přijímací i pro tři vysílací buffery definováno pole ukazatelů, které dovoluje snadný výběr např. naposledy naplněného bufferu, nebo výběr dalšího bufferu k naplnění.

Definice pole ukazatelů na přijímací buffery:

```
static unsigned int const rxBufPtr[NUM_BUF] =
{     (unsigned int) rxBuf0,
      (unsigned int) rxBuf1,
      (unsigned int) rxBuf2   }
```

Parametr NUM_BUF představuje volbu jednoho ze tří bufferů a jsou za něj dosazovány např. proměnné `lastFullRxBuf`, nebo `nxtBufToRcv`. Nastavení těchto proměnných probíhá během obsluhy přerušení. Na přijímací straně je po dokončení DMA přenosu a naplnění jednoho ze vstupních bufferů volána obsluha přerušení, během níž je prováděna funkce `McASPRxDMAComplHandler()`. Součástí této funkce je změna stavu proměnné `lastFullRxBuf` pro indikaci, že přijetí nového bufferu bylo dokončeno:

```
lastFullRxBuf = (lastFullRxBuf + 1) % NUM_BUF;
```

Dále jsou aktualizovány DMA parametry pro příjem dalších dat a je aktivován DMA přenos pro příjem dat do následujícího bufferu:

```
nxtParToUpdate =  PAR_RX_START + parOffRcvd;
parOffRcvd = (parOffRcvd + 1) % NUM_PAR;
BufferRxDMAActivate(nxtBufToRcv,nxtParToUpdate,
                        PAR_RX_START + parOffRcvd);
```

Na konci funkce je změněn stav proměnné `nxtBufToRcv` k určení dalšího bufferu pro příjem dat po dokončení stávajícího přenosu:

```
nxtBufToRcv = (nxtBufToRcv + 1) % NUM_BUF;
```

Při zpracování vstupních dat budeme pracovat vždy s posledním naplněným bufferem. Nad ním budeme provádět zamýšlené operace, jejichž výsledky budeme ukládat do dalších bufferů připravených pro přenos na výstupní zařízení. Abychom však mohli operace nad těmito daty provádět, je nutné znát formát, v jakém jsou jednotlivé audio vzorky v bufferu uloženy.

Pro každý vzorek vstupního audio signálu je vyhrazeno 32b, přičemž v těchto 32b je obsažen vzorek pro pravý i pro levý kanál, každý tedy o velikosti 16b. V úloze budeme vybírat vzorky z levého kanálu a vzhledem k tomu, že máme pomocný vstupní buffer `rxptrt` definovaný s datovým typem unsigned short, je jeho posloupnost vzorků taková, že první hodnota v bufferu je celých 16b z prvního vzorku levého kanálu, druhá hodnota v bufferu je nulová, třetí hodnota obsahuje celých 16b z prvního vzorku pravého kanálu, čtvrtá hodnota je nulová, atd. Ve funkci pro zpracování vstupních dat je proto vybrán pouze každý čtvrtý prvek ze vstupního pomocného bufferu, ostatní data jsou zahozena. Tím si zajistíme, že při dalším zpracování budeme mít pouze data z levého audio kanálu.

7.2.4 Funkce pro zpracování dat

V této kapitole se budeme zabývat nekonečným cyklem `while(1)`, v jehož těle jsou volány všechny funkce provádějící operace nad vstupními daty. Začneme podcyklem `while(audio--)`, v němž je vedle volání funkcí pro zpracování zvukových dat prováděna také změna DMA parametrů pro odeslání dat na výstup pro poslech (jen pokud proměnná `poslech_on = true`). Cyklus `while(audio--)` začíná vždy podmínkou:

```
if(lastFullRxBuf != lastSentTxBuf){
```

Tzn. pokud je poslední naplněný buffer různý od posledního odeslaného bufferu, jsou prováděny další funkce, resp. celé tělo cyklu `while(audio--)` je závislé na splnění této podmínky. Na začátku cyklu je poté provedena aktualizace stavu proměnné pro určení bufferu k odeslání dat na audio výstup. Poslední naplněný buffer (`lastFullRxBuf`) je následně zkopírován do prvního pomocného bufferu `rxptrt`:

```
memcpy((void *)rxptrt, (void *)rxBufPtr[lastFullRxBuf], AUDIO_BUF_SIZE);
```

Nyní již máme vše připraveno pro zahájení operací nad vstupními daty, které jsme si uložili do pomocného bufferu `rxptrt`. Postupně si nyní ukážeme činnost a nástin všech naprogramovaných procedur, které lze z těla cyklu `while(1)` spustit. Po vysvětlení funkcí z podcyklu `while(audio--)` se přesuneme do podcyklu `while(video--)`, který slouží k zobrazení výsledků na displeji a zobrazení odpovídajících popisků.

7.2.4.1 *Funkce fill_buff_graph()*

Touto funkcí je prováděna transformace dat ze vstupního pomocného bufferu `rxptrt` do formátu vhodného pro další zpracování a pro zobrazení původního signálu v grafu na displeji. Dále se uvnitř této funkce nachází procedury pro zapnutí periodické funkce, pro možnost změny měřítka na ose y při přesáhnutí maximální hodnoty a pro zapnutí sledování šumu. Tyto procedury jsou spouštěny v závislosti na počátečním nastavení příslušných parametrů a popíšeme si je samostatně v dalších podkapitolách.

Funkce **`fill_buff_graph()`** je definována s těmito parametry:

```
static void fill_buff_graph(unsigned short *buff, unsigned int vyska,
      unsigned int hranice, bool sum, bool meritko);
```

`*buff`	buffer s naposledy přijatými audio vzorky
`vyska`	rozsah osy y grafu zobrazeného na displeji
`hranice`	vyjadřuje posunutí grafu na obrazovce směrem nahoru od určené pozice
`sum`	vyjadruje volbu, zda zapnout sledování šumu
`meritko`	vyjadřuje volbu, zda zapnout možnost změny měřítka na ose y

V případě, že jsou ve funkci **`fill_buff_graph()`** vypnuty všechny další procedury, provádí se jen dva cykly for. První for cyklus prochází celý vstupní buffer a vybírá audio vzorky z levého kanálu. K těmto vzorkům přičítá vždy hodnotu 32768. Vstupní vzorky mají datový typ **`unsigned int`** (s rozsahem 0-65535), tedy datový typ neobsahující záporná čísla. Abychom získali rozložení hodnot vzorků kolem nuly, posuneme touto operací nulu na hodnotu 32768. Všechny záporné hodnoty tak budeme mít v intervalu <0, 32767> a všechny kladné hodnoty v intervalu <32768, 65535>. Další funkcí prvního for cyklu je stanovení minimální a maximální hodnoty ze všech vzorků v prohlíženém bufferu. Do proměnné `rozdil` poté ukládáme rozdíl mezi maximální a minimální hodnotou.

Druhý for cyklus prochází vzorky z prvního for cyklu. Na začátku tohoto cyklu je od všech vzorků odečítána hodnota rovná hodnotě nejmenšího vzorku. Tím si zajistíme, že všechny vzorky budou v rozsahu <0,`rozdil`>. Hlavní činností druhého for cyklu je naplnit dva různé buffery transformovanými hodnotami audio vzorků.

První buffer se jmenuje `ingraph_buff`, má datový typ **`unsigned char`** a slouží k uložení vstupních audio vzorků ve formátu, ve kterém budou poté zobrazeny na monitoru jako graf vstupního signálu v časové oblasti. Na ose y můžeme pro hodnoty vzorků využít rozsah 0-255. Vzhledem k tomu, že na obrazovce budeme zobrazovat dva grafy nad sebou, jeden se vstupním signálem v časové oblasti a jeden po FFT transformaci ve frekvenční oblasti, volíme výšku jednoho grafu a tedy i rozsah hodnot osy y v rozmezí 0-190. Parametr `vyska` ve funkci **`fill_buff_graph()`** je tedy roven hodnotě 190. Lze jej měnit, ovšem je přitom potřeba dbát na velikost zobrazovacího prostoru a na překrývání s dalšími grafy a popisky na obrazovce. Do bufferu ingraph_buff jsou ukládány vzorky v rozsahu (<0,`vyska`>)+hranice. Nula vstupního signálu je v tomto rozsahu představována hodnotou `vyska`/2. Parametr

`hranice` (implicitně nastaven na hodnotu 10) poskytuje možnost posunou graf na obrazovce výše a oddělit jej tak od spodního grafu, případně si udělat místo na další zobrazované prvky. Předpokladem v základním nastavení je, že součet parametrů `vyska+hranice` nepřesáhne hodnotu 210 (z důvodu velikosti zobrazovacího prostoru).

Druhý buffer nese název `input_buff`. Data jsou do něj ukládána v nezměněném rozsahu, jsou pouze posunuta o `-rozdil`/2 tak, aby nula byla v novém rozsahu hodnot představována skutečně nulou. Ve výsledku jsou tedy hodnoty v bufferu `input_buff` v intervalu <`-rozdil`/2, `rozdil`/2>. Tento buffer je dále využíván jako vstupní buffer pro funkci váhování oknem. Případně, pokud je váhování vypnuto, je vstupním bufferem přímo pro funkci provádějící FFT transformaci.

7.2.4.2 Funkce pro váhování signálu oknem

Funkci pro váhování signálu oknem lze jednoduše spustit, či vypnout pomocí definovatelného parametru `window_on`. Pokud je `window_on=true`, potom je prováděno váhování oknem. Součástí programu jsou dvě váhovací funkce, mezi nimiž lze volit. První funkce provádí váhování pomocí Hammingova okna – **`get_hamming()`** a druhá pomocí Hanningova okna – **`get_hanning()`**. Funkce jsou obdobné, pouze s jinými transformačními vztahy.

Ukážeme si proto pouze funkci **`get_hamming()`**. Ta obsahuje následující for cyklus:

```
for (i = 0; i < FFTSIZE; i++) {
   multiplier = (0.54 - 0.46 * cos(2*PI*i/(FFTSIZE-1)));
   window_buff[i] = multiplier * input_buff[i];
}
```

Výsledkem funkce je buffer `window_buff` obsahující váhované vzorky. Tento buffer je použit jako vstupní buffer ve funkci provádějící FFT transformaci.

7.2.4.3 Funkce pro FFT transformaci

Provedení FFT transformace provádí funkce **`get_fft(window_buff)`**, jejímž jediným parametrem je volitelný buffer (standardně `window_buff` s váhovanými vzorky signálu). Zbylé parametry používané uvnitř funkce jsou globální proměnné (především důležitý parametr délka FFT transformace).

Hlavním prvkem funkce **`get_fft()`** je funkce **`DSPF_sp_fftSPxSP_cn()`** z knihovny DSPLIB pro výpočet komplexní dopředné FFT mixed radix transformace s bitovou reverzací. Její definice je následující:

```
void DSPF_sp_fftSPxSP_cn (int N, float *ptr_x, float *ptr_w,
              float *ptr_y, unsigned char *brev, int n_min,
              int offset, int n_max);
```

`N`	počet komplexních vzorků vyjadřujících délku FFT transformace (dovolený rozsah - $8 \leq N \leq 8192$)
`*ptr_x`	ukazatel na komplexní vstupní data

`*ptr_w`	ukazatel na komplexní tzv. twiddle faktory
`*ptr_y`	ukazatel na komplexní výstupní data
`*brev`	ukazatel na tabulku bitové reverzace se 64 položkami
`n_min (radix)`	nejmenší fft motýlek používaný při výpočtu
`offset`	index na dílčí FFT transformaci od začátku hlavní FFT
`n_max`	velikost hlavní FFT transformace

Pro použití této funkce musíme mít definovanou, či vypočtenou řadu parametrů. Délka FFT transformace je dána velikostí globální proměnné `FFTSIZE`.

Na základě délky FFT transformace stanovíme hodnotu parametru n_min (označovaného také jako radix), který představuje nejmenší fft motýlek použitý při výpočtu FFT. Pro provedení transformace musí být délka FFT transformace vždy násobkem mocniny dvou (2^n). V případě, že n je liché, zvolíme radix roven 2 (např. pro `FFTSIZE` = 8). Pokud je n sudé, zvolíme radix roven 4 (např. pro `FFTSIZE = 16`). Proto funkce **`get_fft()`** začíná podmínkou, v níž je volena velikost parametru radix.

Dalším potřebným parametrem je ukazatel na pole obsahující tzv. twiddle faktory. Tyto faktory představují exponenciály W_N^k vysvětlené v kapitole 7.1.4. Pro jejich výpočet je volána funkce **`tw_gen`**`(w, FFTSIZE)` z knihovny DSPLIB. Ta na základě zvolené délky FFT transformace vypočítá všechny potřebné twiddle faktory a uloží je do pole `w[2*FFTSIZE].`

Vzhledem k tomu, že funkce **`DSPF_sp_fftSPxSP_cn`**`()` je napsána pro transformaci komplexních proměnných a náš vstupní signál (ve `window_buff`) obsahuje pouze reálnou část, je nutné vstupní posloupnost vzorků nejprve drobně upravit. Funkce **`DSPF_sp_fftSPxSP_cn`**`()` uvažuje posloupnost vstupních vzorků, v níž vždy lichý vzorek představuje reálnou část a sudý vzorek imaginární část. Provedeme proto jedním for cyklem vložení nuly za každý vzorek vstupní posloupnosti. Tím určíme, že imaginární část všech vzorků vstupujících do FFT transformace bude vždy nulová. Výsledek této operace je uložen v poli `x[2*FFTSIZE].`

Posledním chybějícím parametrem, který je nutné před použitím funkce **`DSPF_sp_fftSPxSP_cn`**`()` definovat, je tabulka bitové reverzace pro prohození vzorků, které jsou zpřeházené procesem decimace v čase. Hodnoty tabulky uložené v poli `brev[64]` převezmeme z knihovny DSPLIB.

Po těchto přípravách je uvnitř funkce **`get_fft`**() volána funkce pro FFT transformaci v tomto tvaru:

```
DSPF_sp_fftSPxSP_cn(FFTSIZE, x, w, y, brev, rad, 0, FFTSIZE);
```

Po provedení transformace máme její výsledek uložen v poli `y[2*FFTSIZE]`. Uvnitř funkce **`get_fft`**() však ještě zůstáváme. Nachází se zde též procedura pro normalizaci výstupních dat z FFT transformace. To je prováděno zejména proto, abychom měli hodnoty výstupních dat v určitém definovaném rozsahu a formátu vhodném pro použití v zobrazovacích funkcích. Jsou zde proto zahozeny vzorky výstupní posloupnosti s imaginární částí. Dalším krokem je nalezení maximální hodnoty pomocí funkce **`DSPF_sp_maxval`**`(y,FFT_HALF_SIZE)` z knihovny DSPLIB.

Maximální hodnotou následně vydělíme všechny vzorky z výstupní posloupnosti, čímž zajistíme, že budou všechny v rozsahu <0,1>. Jelikož potřebujeme pouze polovinu ze všech výstupních FFT vzorků (výsledné frekvenční spektrum je symetrické), na závěr nahradíme nulami všechny vzorky z druhé poloviny posloupnosti vzorků.

7.2.4.4 Funkce fill_buff_graph_fft()

Po provedení FFT transformace jsme si její výsledky uložili do druhého pomocného bufferu `rxptrt2` a voláme funkci definovanou následovně:

```
void fill_buff_graph_fft(float *buff, unsigned int vyska,
                                      unsigned int hranice);
```

`*buff`	buffer s vypočtenými vzorky frekvenčního spektra
`vyska`	rozsah osy y grafu zobrazovaného na displeji
`hranice`	vyjadřuje na displeji posunutí grafu směrem nahoru od určené pozice

Tato funkce slouží k převodu hodnot vzorků frekvenčního spektra do takového formátu, aby je bylo možné v odpovídající podobě zobrazit na monitoru. Oproti obdobné funkcionalitě ve funkci **`fill_buff_graph()`** je zde situace jednoduší, protože vzorky máme již normované procedurou z funkce **`get_fft()`**. Provedeme proto pouze jednoduchou operaci, ve které vynásobíme každý vzorek parametrem `vyska` a k tomu přičteme parametr `hranice`. Oba parametry lze volitelně měnit, ovšem musíme dbát na limity velikosti plochy zobrazovacího zařízení a na překryvy s dalšími zobrazovanými prvky. Implicitně je parametr `vyska = 190` a `hranice = 20`.

7.2.4.5 Změna měřítka

Získali jsme spektrum vstupního signálu, které je připravené pro převedení do dvourozměrného vyjádření v grafu. To provádí zobrazovací funkce popsaná v kapitole 7.2.4.10. Ještě před touto funkcí si vysvětlíme několik dalších doplňkových procedur, které lze zapínat a vypínat před spuštěním běhu programu. Jednou z těchto procedur je možnost zmenšení měřítka. Ta se nachází uvnitř funkce **`fill_buff_graph()`** a její spuštění je závislé na nastavení dvoustavového parametru `zmena_meritka`. Tato procedura se vztahuje pouze ke grafu vstupního signálu a hlídá, aby hodnoty vzorků vstupního signálu nepřesáhly dovolenou hranici. V případě, že ji přesáhnou, tedy v případě, kdy se ocitnou mimo zobrazovací prostor, je provedena změna měřítka osy y zobrazovaného grafu.

Na začátku funkce **`fill_buff_graph()`** je nastavena proměnná `max_val`, kterou je dána maximální hodnota vstupního signálu zobrazitelná v grafu. Tzn., že všechny vzorky s vyšší hodnotou nebudou v grafu zobrazeny, pokud bude procedura změny měřítka vypnuta. Jestliže ji zapneme a jediný vzorek ze všech zobrazovaných přesáhne hodnotu proměnné `max_val`, je proměnná `max_val` zvětšena o `max_val*1,5`. To zajistí 1,5x zmenšení měřítka osy y grafu průběhu vstupního signálu, protože hodnotou `max_val` jsou děleny všechny vzorky v převodním vztahu funkce **`fill_buff_graph()`**. Zobrazený vstupní signál bude vždy kompletní, protože hodnota

`max_val` je zvětšována v cyklu **`while`**`(rozdil > max_val)` do té doby, než přesáhne maximální hodnotu vstupního signálu.

7.2.4.6 Sledování šumu

Tato jednoduchá procedura se opět nachází v těle funkce **`fill_buff_graph`**`()` a její provedení je závislé na dvoustavovém parametru `sum_on`. Princip činnosti je velmi podobný jako u procedury změny měřítka. I zde využíváme proměnnou `max_val`, kterou zde nastavujeme na velmi nízkou úroveň v případě, že je maximální hodnota vstupního signálu menší než určená mez. Pokud je tedy signál pod úrovní stanovené meze (nastavené na velmi nízkou hodnotu), jedná se zpravidla o šum, který si povolením této procedury zvětšíme na celý zobrazovací prostor a získáme tak detail šumu.

7.2.4.7 Synchronizace časové základny

Poslední procedurou uvnitř funkce **`fill_buff_graph`**`()` je synchronizace časové základny. Tu má smysl použít, pokud víme, že vstupní signál je periodický. V takovém případě nastavíme parametr `periodic_f = true`. Procedura synchronizace časové základny plní podobný úkol jako stejnojmenná funkce u osciloskopů. Tzn., že zajišťuje stabilizaci periodického signálu, aby jeho zobrazený úsek začínal vždy ve stejném místě periody. Princip činnosti procedury je takový, že v prvním for cyklu je nalezen určitý bod z předem daného rozmezí hodnot vstupního signálu a pozice tohoto bodu v posloupnosti vstupních vzorků je uložena do pomocné proměnné. Ve druhém for cyklu je první vzorek vstupního signálu v bufferu `rxptrt` nahrazen vzorkem z pozice dané pomocnou proměnnou a další vzorky jsou nahrazeny vzorky následujícími za pomocnou proměnnou. Tím došlo k posunutí začátku zobrazovaného signálu na určenou pozici a s každým novým bufferem bude začátek signálu znovu posunut do stejného místa periody. Zjednodušený zápis procedury vypadá následovně:

```
for(i=1; i<AUDIO_BUF_SIZE; i++){
     if((buf[i]>=0) && (buf[i]<=rozdil/20) && buf[i]<=buf[i+1]){
          pred = i;
          i=AUDIO_BUF_SIZE;
    }
}
for(j=0; j<AUDIO_BUF_SIZE - pred; j++){
     buf[j] = buf[j + pred];
}
```

7.2.4.8 Lupa

Procedura nazvaná lupa je věnována možnosti úpravy zobrazení grafu frekvenčního spektra. Její spuštění závisí na dvoustavovém parametru `lupa_on`. Konkrétně dovoluje výřez zadaného úseku spektra a jeho zobrazení. Funkce má definovány parametry **`int`** `n` a **`int`** `posun`. Parametr `n` udává, kolika násobný bude výřez úseku grafu (lze volit v rozmezí 2x - 16x). Parametr `posun` definuje v jakém místě na ose x bude začínat výřez úseku grafu (maximálně může mít hodnotu 256).

7.2.4.9 *Zoom out*

Poslední volitelná procedura závislá na dvoustavovém parametru `zoom_out` je spuštěna pouze, pokud je délka FFT transformace alespoň 2048 vzorků. Jejím úkolem je vynechat v takovém případě ze zobrazované posloupnosti každý x-tý vzorek (dle zvolené délky FFT transformace), aby bylo možné zobrazit celé vypočítané frekvenční spektrum na obrazovce.

7.2.4.10 *Zobrazovací funkce*

Pro zobrazení výsledků na monitoru je nutné převést hodnoty vzorků původního signálu a hodnoty vzorků frekvenčního spektra do dvourozměrného prostoru, který představuje jednotlivé pixely. Tyto pixely mohou nabývat pouze dvou stavů, buď černá (0), nebo bílá (65535). Nebudeme zde proto potřebovat znát video formát dat, ve kterém jsou zpracována pro zobrazení na displeji (případně je vysvětlen v rozboru úlohy Cannyho hranový detektor – viz kap. 9.1.2 a 9.1.3). Černou barvou je vyplněno pozadí, bílou jsou vykresleny grafy a popisky.

Pro volání zobrazovacích funkcí se musíme přesunout do cyklu `while(video--)`. Odtud probíhá výběr, kterou ze zobrazovacích funkcí použijeme. Pro úlohu byly vytvořeny tři varianty. První funkce **`zobraz_graf()`** dovoluje zobrazení jednoho grafu přes většinu plochy displeje. V tomto grafu lze zobrazit původní signál, případně váhovaný signál. Druhou zobrazovací funkcí je **`zobraz_2grafy()`**, která vytváří na obrazovce dva grafy nad sebou (každý zaujímá polovinu obrazovky). Díky této funkci lze provést přímé srovnání dvou signálů (v úloze pro srovnání původního a váhovaného signálu). Poslední funkce je pojmenována **`zobraz_2grafy_fft`**() a umožňuje opět zobrazení dvou grafů nad sebou, přičemž spodní graf je vždy pro zobrazení frekvenčního spektra.

Vzhledem k tomu, že všechny tři funkce jsou založeny na stejném principu, ukážeme si zde pouze funkci **`zobraz_2grafy_fft`**().

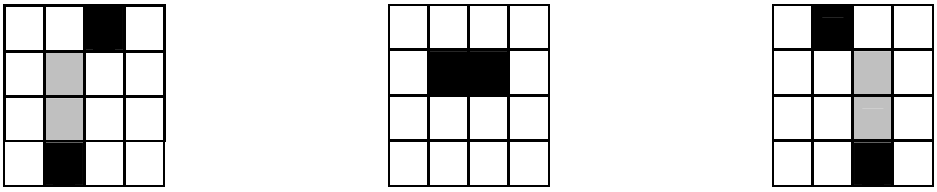
```
void zobraz_2grafy_fft(unsigned char *buff_graf1,
                       unsigned char *buff_graf2,
                       unsigned int pingpong);
```

`*buff_graf1`	buffer 1 – vstupní, či váhovaný signál
`*buff_graf2`	buffer 2 – vypočtené frekvenční spektrum
`pingpong`	parametr, dle kterého dochází ke střídání dvou používaných video bufferů - mění se na konci cyklu `while(video--)`

Na začátku funkce jsou nastaveny parametry `side = 40` (jak daleko od levého okraje displeje bude začínat graf), `bottom_first = 250` (jak daleko od spodního okraje obrazovky bude spodní okraj 1. grafu) a `bottom_second = 20` (jak daleko od spodního okraje obrazovky bude spodní okraj 2. grafu). Funkce začíná for cyklem, který postupně prochází od levého dolního rohu všechny sloupce zobrazovacího prostoru (640x480). V tomto for cyklu se nacházejí 2 další for cykly. První řeší zobrazení horního grafu, přičemž prochází řádky 250-460 (číslování řádků od spodního kraje obrazovky), což je dáno parametry `bottom_first` a `480-bottom_second`.

Druhý řeší zobrazení spodního grafu a prochází přes řádky 20 – 230, což odpovídá parametrům `bottom_second` a `480-bottom_first`.

Ve for cyklu pro horní graf je několik vnořených podmínek, které zajišťují, abychom nepřesáhli zobrazovací prostor na ose x a abychom nepokračovali ve vykreslování, pokud je počet vzorků signálu menší než zobrazovací prostor na ose x. Další vnořené podmínky již rozhodují, zda je pixel bodem grafu a bude proto nastaven na bílou barvu, nebo není a bude nastaven na černou barvu. Zde lze již objasnit, proč procházíme obraz po sloupcích zleva doprava a v každém sloupci řádky od spodního levého okraje po horní okraj obrazovky.



Obrázek 16 - Funkce mezi dvěma pixely a) rostoucí, b) konstantní, c) klesající funkce

Na obrázku 16 jsou patrné tři možné situace dvou sousedních vzorků signálu vyjádřených již jako pixely (na obrázku označeny černě) v dvourozměrném uspořádání. V prvním případě je v daném místě funkce rostoucí. Abychom zajistili, že mezi těmito body bude spojnice a neměli jen bodový graf, procházíme sloupec s levým pixelem po řádcích odzdola nahoru a nastavíme všechny pixely v cestě jako bod grafu (označeny šedivou), dokud nedosáhneme hodnoty pravého sousedního pixelu. Ve druhém případě jde o konstantní funkci, u které pouze nastavíme pixel příslušející hodnotě vzorku a pokračujeme na další sloupec. Poslední případ je pro funkci klesající, pro kterou je situace analogická funkci rostoucí.

Zvláštními podmínkami jsou nastaveny na bílou barvu také osy. Když se aktuální sloupec rovná parametru `side`, jsou všechny řádky tohoto sloupce v rozmezí obou vnořených for cyklů zbarveny na bílou (osa y) a jestliže aktuální řádek se rovná parametru `bottom_first`, nebo `bottom_second`, je vytvářena osa x horního i dolního grafu. Poslední částí prvního vnořeného for cyklu je přenos hodnoty aktuálního pixelu (uloženého do proměnné `px`) na danou pozici video bufferu `Rgb_buffer1`, nebo `Rgb_buffer2`. Oba video buffery se střídavě plní, což zajišťuje parametr `pingpong` nastavený před voláním funkce **`zobraz_2grafy_fft`**(). Video buffery ovšem představují posloupnost dat bez dvourozměrného vyjádření. V této posloupnosti jsou za sebe řazeny jednotlivé řádky od horního okraje obrazovky po spodní okraj obrazovky. Převod aktuální pozice na obrazovce do tohoto bufferu vypadá následovně:

```
Rgb_buffer1[((480-(radek+1))*640)+(sloupec)]=(px);
```

Po skončení prvního vnořeného for cyklu následuje druhý vnořený for cyklus pro zobrazení průběhu frekvenčního spektra ve spodním grafu. Zde narozdíl od horního grafu nebudeme spojovat sousední pixely průběhu signálu, ale budeme vždy tvořit spektrální čáry od nuly až do hodnoty dané velikostí příslušného vzorku frekvenčního spektra. To je mnohem jednodušší situace než v horním grafu a je řešena jedinou

podmínkou při procházení sloupce odzdola nahoru a nastavování pixelů na bílou barvu, dokud nenarazíme na pixel příslušející dané hodnotě vzorku frekvenčního spektra. Na konci druhého for cyklu je opět převod hodnoty aktuálního pixelu na danou pozici jednoho z video bufferů `Rgb_buffer1`, nebo `Rgb_buffer2` (totožné jako u prvního for cyklu).

7.2.4.11 Funkce pro kreslení popisků

Poslední funkcí volanou z cyklu `while(video--)` před odesláním video bufferu s výsledky na displej je funkce pro přidání názvů, popisků a měřítek os grafů. To řeší funkce s názvem **`kresli_meritko()`**, jejíž spuštění je závislé na dvoustavovém parametru `popisky_os_on`.

```
void kresli_meritko(int poz_x, int poz_y, int img_width,
                    int img_height, unsigned int pingpong);
```

`poz_x`	vybraný referenční sloupec obrazovky
`poz_y`	vybraný referenční řádek obrazovky
`img_width`	šířka displeje = 640
`img_height`	výška displeje = 480
`pingpong`	parametr, dle kterého dochází ke střídání video bufferů

Z této funkce je dle různých podmínek (zvolená vzorkovací frekvence, délka FFT transformace, povolení, či zakázání doplňkových funkcí) prováděno četné volání funkce **`kresli_retezec()`** pro kreslení požadovaných textových řetězců na vybraných místech zobrazovacího prostoru.

```
void kresli_retezec(char *retezec, int poz_x, int poz_y,
               int img_width, int img_height, unsigned int pingpong);
```

`*retezec`	pole znaků, resp. řetězec k zobrazení
`poz_x`	počáteční sloupec řetězce
`poz_y`	počáteční řádek řetězce
`img_width`	šířka displeje = 640
`img_height`	výška displeje = 480
`pingpong`	parametr, dle kterého dochází ke střídání dvou používaných video bufferů - mění se na konci cyklu `while(video--)`

Zde oproti předešlé funkci přibyl parametr pole znaků, kam lze zadat libovolný řetězec, který chceme zapsat na určené místo zobrazovacího prostoru (`poz_x, poz_y`). Funkce **`kresli_retezec()`** obsahuje for cyklus, ze kterého je pro každý znak z pole `*retezec` volána funkce **`vyber_znak()`**.

```
void vyber_znak(char znak, int poz_x, int poz_y, int img_width,
                           int img_height, unsigned int pingpong);
```

Parametry funkce **vyber_znak()** jsou opět velmi podobné, liší se pouze parametrem `znak`, který slouží pro definici hledaného znaku z řetězce. Funkce obsahuje switch – case strukturu pro výběr hledaného znaku. Každý znak je určen pro zobrazení v prostoru 7x8 pixelů, což je patrné z obrázku 17.

```
case '2':
        {
                unsigned short znak[] = {0,1,1,1,0,0,0,
                                         1,0,0,0,1,0,0,
                                         0,0,0,0,1,0,0,
                                         0,0,0,1,0,0,0,
                                         0,0,1,0,0,0,0,
                                         0,1,0,0,0,0,0,
                                         1,1,1,1,1,0,0,
                                         0,0,0,0,0,0,0};

                kresli_znak(znak, poz_x, poz_y, img_width, img_height, pingpong);
        }
            break;
```

Obrázek 17 - Příklad definice znaku

Znak uložený v poli `znak[]` je v posledním kroku převeden do video bufferu na zadanou pozici pomocí funkce **`kresli_znak()`** – viz obrázek 17. Ta pomocí dvou vnořených for cyklů prochází od zadaného místa prostor 7x8 pixelů a dle pole `znak[]` jim přiřadí bílou, nebo černou barvu. Vše rovnou ukládá do jednoho ze dvou video bufferů, který je zvolen na základě parametru `pingpong`.

7.2.5 Zobrazení na displeji

Poté, co jsme naplnili daty výstupní video buffery, podíváme se nyní na závěrečnou část úlohy, kterou je proces zobrazení dat z video bufferu na monitoru. V hlavní funkci **`main()`** se proto nachází inicializace potřebných zařízení. Nejprve jsou inicializovány oba již zmíněné pravidelně se střídající video buffery `Rgb_buffer1` a `Rgb_buffer2`, do nichž se ukládají výsledky k zobrazení na displeji. Následuje volání funkce **`SetupIntc()`** pro konfiguraci přerušení. Funkce je bez parametrů a má stejnou strukturu jako funkce **`McASPErrorIntSetup()`** popsaná v kapitole 7.2.2. Stejným způsobem je zvolena obsluha přerušení (`LCDIsr`), která je přiřazena jednomu z maskovatelných přerušení procesoru (`C674X_MASK_INT4`). Vybranému přerušení je přiřazena zadaná systémová událost (`SYS_INT_LCDC_INT`) a v poslední části je zvolené přerušení procesoru aktivováno. Obsluha přerušení `LCDIsr` po každém skončeném snímku provede zobrazení nového snímku z připraveného výstupního video bufferu a pokud je dostupný nově naplněný buffer pro zobrazení, aktualizuje se parametr `updated`, kterým je nově naplněný buffer nastaven jako příští výstupní buffer.

Další volanou inicializační funkcí je **`SetUpLCD()`**. Tou je prováděna inicializace LCDC Raster řadiče a skládá se z celkem dvanácti dílčích funkcí. Je zde obsažena funkce pro spuštění a nastavení požadovaného stavu periferie LCDC. Další je funkce pro volbu pinů procesoru použitých pro LCDC. Následuje funkce pro nastavení obnovovací frekvence **`RasterClkConfig()`**. Funkce **`RasterDMAConfig`**() řeší konfiguraci DMA uvnitř LCDC řadiče.

```
RasterDMAConfig(unsigned int baseAddr, unsigned int frmMode,
      unsigned int bustSz, unsigned int fifoTh, unsigned int endian);
```

`baseAddr`	základní adresa LCD modulu
`frmMode`	určuje, zda je použit jeden, nebo dva video buffery (v úloze použity dva)
`bustSz`	určuje velikost dávky u DMA (v úloze nastavena na hodnotu 16)
`fifoTh`	Hodnota, která definuje, kdy může LCDC Raster Controller číst ze vstupní FIFO (v úloze nastavena na hodnotu 8)
`endian`	určuje zda použít pro uspořádání dat způsob big endian, nebo ne (v úloze je big endian vypnut)

V pořadí další funkce se jmenuje `RasterModeConfig()`.Touto funkcí je nastaven mód zobrazení (TFT, nebo STN) a paleta zobrazovaných dat (barevná, nebo monochromatická). V této úloze je zvolen mód TFT s barevnou paletou zobrazovaných dat. Následuje funkce `RasterLSBDataOrderSelect()` pro seřazení dat ve video bufferu od nejméně po nejvíce významný bit. V poslední části jsou tři funkce pro nastavení polarity různých parametrů pro časování LCDC Raster kontroléru a pro nastavení horizontálních a vertikálních parametrů pro časování.

Po inicializační funkci **`SetUpLCD()`** je volána funkce `RasterDMAFBConfig()`, kterou jsou pro oba video buffery nastaveny počáteční a koncové hodnoty adres. Další je funkce `RasterEndOfFrameIntEnable`(). Ta po skončení každého snímku aktivuje přerušení. Na závěr je aktivován LCDC Raster kontrolér pomocí funkce `RasterEnable()`.

7.2.6 Testování

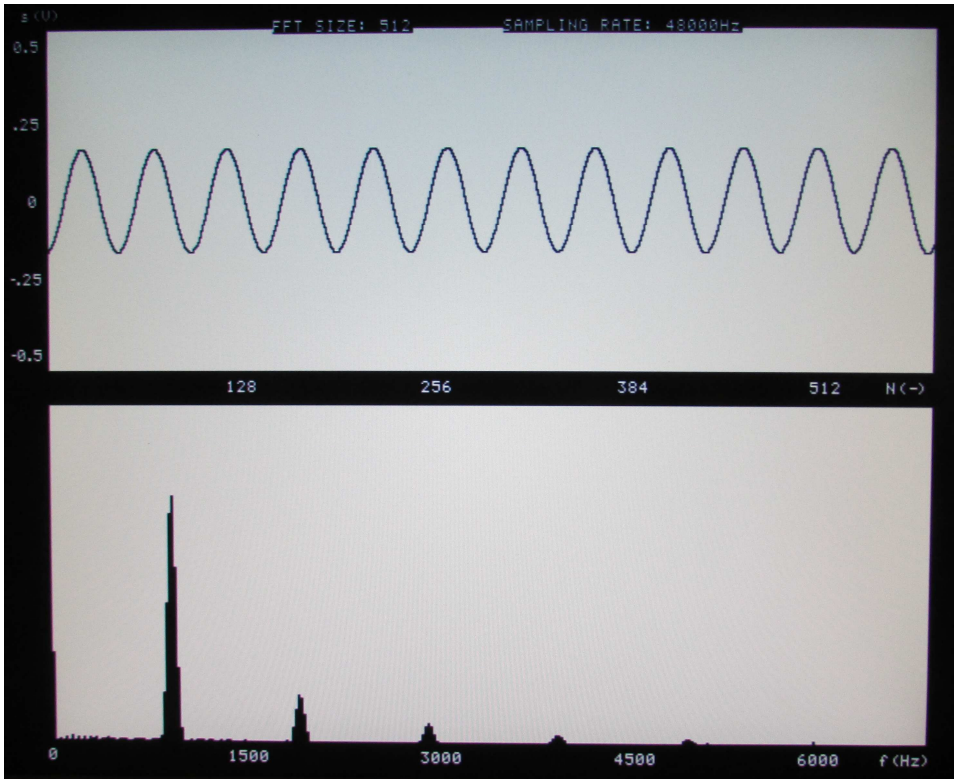
Výsledné zobrazení na monitoru, tedy i výsledek implementace úlohy spektrální analyzátor ukazují obrázky 18 a 19. Horní graf je vstupní signál, dolní graf je spektrum tohoto signálu. Pro tento případ byla zvolena vzorkovací frekvence 48kHz, délka FFT transformace 512 vzorků, bylo povoleno váhování signálu Hammingovo oknem a na true byly nastaveny parametry `periodic_f`, `popisky_os_on`, `window_on` a `fft_on`.

Pro generování periodických signálů různých tvarů a s různými frekvencemi ze zvukové karty byly využívány především tyto dva nástroje:

- Signální generátor v2.3.0 – pro levý kanál lze generovat signály ve tvaru sinus, obdélník, pila a šum o frekvenci 22Hz - 22kHz
- Tone Generator v3.0.0.4 – oproti prvnímu nabízí navíc další typy signálů (např. generování impulzů)

7.3 Zhodnocení

Konečné zpracování úlohy spektrální analyzátor nabízí možnost zobrazení dvou samostatných grafů na monitoru. V horním grafu je průběh vstupního signálu ze zvukové karty, ve spodním grafu je vypočítané frekvenční spektrum vstupního signálu. Dále si lze zobrazit popisky os, měřítko a hodnoty aktuálně zvolených parametrů vzorkovací frekvence a délky transformace. Vzorkovací frekvenci vstupního signálu lze volit v rozmezí 8 – 96kHz a délku transformace od 8 do 8192 vzorků.



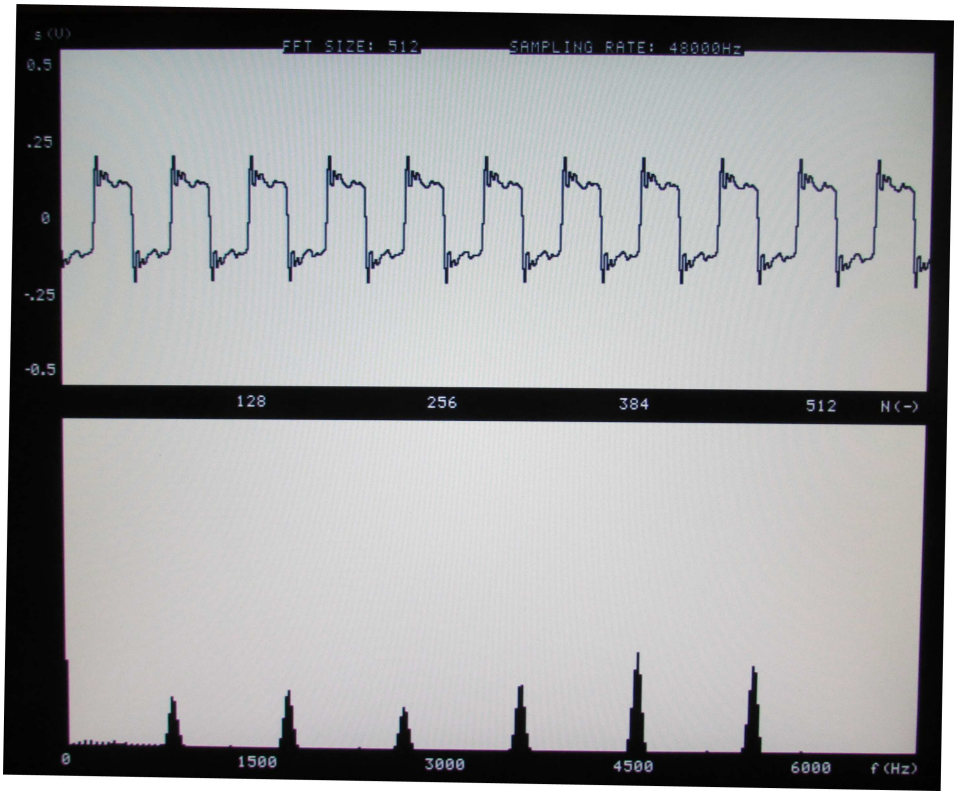
Obrázek 18 - Spektrum sinusového signálu o frekvenci 1000Hz

Vedle hlavní funkce zobrazení frekvenčního spektra byla v úloze vypracována řada doplňkových funkcí, které je možné před spuštěním programu zapínat, či vypínat. Patří sem zejména možnost váhování vstupního signálu oknem, dále synchronizace časové základny pro periodické signály, automatické zmenšení měřítka na ose y při přesáhnutí maximálně zobrazitelné amplitudy, možnost sledování i velmi malých amplitud signálu (zejména šumu), či možnost výřezu daného místa ve frekvenčním spektru.

Mezi důležité součásti programu patří zobrazovací funkce, které dovolují převedení průběhu vstupního signálu i frekvenčního spektra do dvourozměrného vyjádření v grafu na monitoru. Mají na starosti vykreslení os grafů i průběhů signálů a přímo se starají o naplnění výstupních video bufferů. Užitečnou je také navržená funkcionalita pro vykreslování textových řetězců na zadaných místech obrazovky, pomocí níž jsou zobrazovány popisky a měřítka os.

Do popisu úlohy je pro co nejlepší pochopení zahrnut podrobný popis procesu ukládání dat ze vstupního audio zařízení do paměti na LCDK modulu a popis přenesení zpracovaných dat z video bufferů na výstupní zobrazovací zařízení.

Úloha může být snadno rozšířena o další operace, v každém cyklu jsou naplněny pomocné buffery např. vstupním signálem, váhovaným signálem, či frekvenčním spektrem signálu. Lze proto jednoduše zařadit do programu další funkce a procedury, které tak mají okamžitý přístup ke vstupním datům a které mohou zpracovaná data pomocí zobrazovacích funkcí snadno převádět do formy pro zobrazení.



Obrázek 19 - Spektrum obdélníkového signálu o frekvenci 900Hz

8 Úloha č. 2 – AM přijímač

8.1 Teoretický rozbor

Druhá úloha je věnována vytvoření AM přijímače, kde C6748 LCDK modul plní funkci digitálního AM demodulátoru. Přijatý amplitudově modulovaný signál z přijímacího obvodu je nejprve pomocí modulu převodníku ADS1606EVM převeden na 16-bitový digitální signál a dále je posílán přes rozhraní EMIFA do paměti na LCDK modulu. Na uloženou posloupnost vzorků je aplikován algoritmus pro digitální AM demodulaci a výsledek je posílán na audio výstup k poslechu. Schéma pro tuto úlohu je uvedeno v kapitole 5.3. Na LCDK modulu pro tuto úlohu využijeme včetně DSP procesoru subsystémy EMIFA, GPIO, EDMA, řadič přerušení, PSC, paměť DDR2, kodek AIC3106 a sériové rozhraní McASP .

V teoretické části pro úlohu AM přijímač si popíšeme navržený vstupní přijímací obvod, dále uvedeme vlastnosti AD převodu na modulu ADS1606EVM, princip čtení dat z výstupu převodníku pomocí rozhraní EMIFA a EDMA přenos pro uložení dat z EMIFA do bufferu v paměti. Především však je zde věnován prostor popisu digitální AM demodulace, kterým se budeme řídit při implementaci na LCDK modulu.

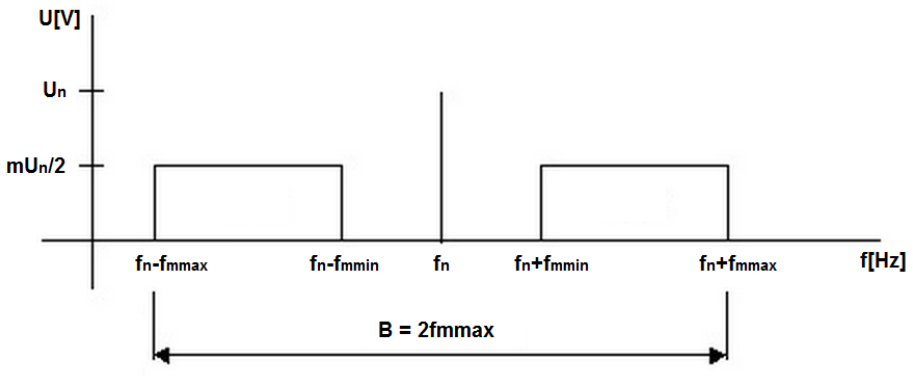
8.1.1 Amplitudová modulace a demodulace

Amplitudová modulace je nejstarším typem modulace, při níž se lineárně mění amplituda nosné vlny v závislosti na modulačním signálu. V základním provedení je amplitudově modulovaný signál složen ze dvou postranních pásem a z nepotlačené nosné vlny.

Průběh amplitudově modulovaného signálu je dán vztahem:

$$u(t) = U_n\left[1 + m\cos(\omega_m t)\right]\cos(\omega_n t),$$

kde $m = U_m / U_n$ představuje hloubku amplitudové modulace (zajišťuje přenos informace o hlasitosti). V praxi je vždy 0 < m < 1, v případě nedodržení této podmínky dochází k přemodulování a ke zkreslení průběhu výstupního signálu.



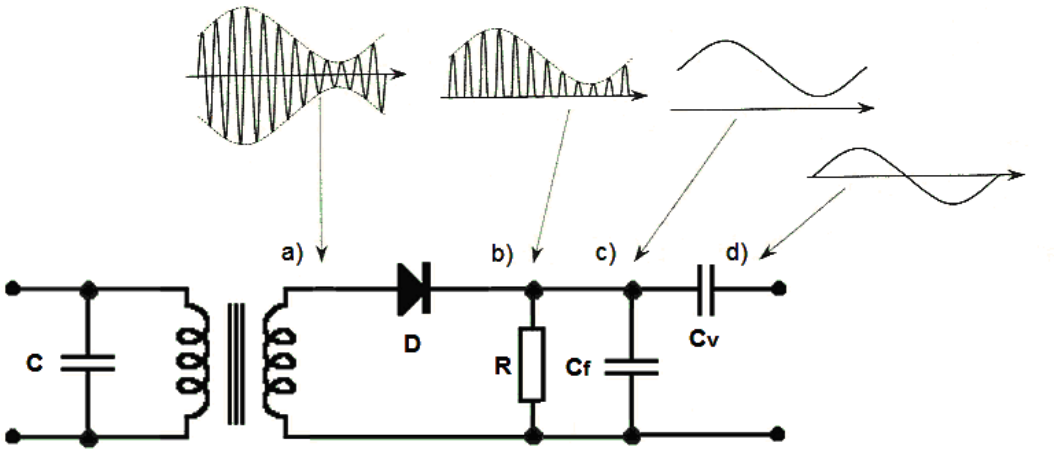
Obrázek 20 - Spektrum amplitudově modulovaného signálu

Vztah pro amplitudově modulovaný signál v rozepsané podobě:

$$u(t) = U_n \cos(\omega_n t) + \frac{U_m}{2}\cos((\omega_n - \omega_m)t - \varphi) + \frac{U_m}{2}\cos((\omega_n + \omega_m)t + \varphi)$$

Spektrum amplitudově modulovaného signálu se skládá ze samotné nosné vlny o amplitudě U_n a ze dvou postranních modulačních složek o amplitudě $mU_n/2$. Obě postranní modulační složky jsou symetricky rozloženy po stranách nosného kmitočtu na frekvencích $f_n + f_m$ a $f_n - f_m$. Z obrázku 20 je patrné, že amplitudy složek postranních pásem mohou dosahovat pro maximální hloubku modulace (tj. m = 1) nejvýše poloviny amplitudy nosné frekvence. Šířka pro 1 přenosový kanál B (šířka přenosového pásma) vyhrazeného pro šíření 1 amplitudově modulovaného signálu je dána dvojnásobkem maximální modulační frekvence $f_{m\max}$ ($B = 2 f_{m\max}$). Např. pro střední vlny je šířka přenosového kanálu B = 9 kHz, což znamená, že nejvyšší povolená modulační frekvence je 4,5 kHz.

Zmíníme také princip amplitudové demodulace, kterou lze v nejjednodušším provedení realizovat pomocí diodového detektoru znázorněného na obrázku 21.



Obrázek 21 - Schéma jednoduchého AM demodulátoru

Jedná se o obvod jednocestného usměrňovače s kondenzátorem na výstupu. Na vstup je přiveden vysokofrekvenční symetrický amplitudově modulovaný signál (a). Dioda D propustí pouze kladné půlvlny signálu (b), které na rezistoru R vytvoří proměnný úbytek napětí a kondenzátor Cf potlačí zbylé vysokofrekvenční složky signálu (c). Původní nemodulovaný nízkofrekvenční signál je získáván z vazebního kondenzátoru C_V, který nepropustí stejnosměrnou složku (d).

8.1.2 Vstupní přijímací obvod

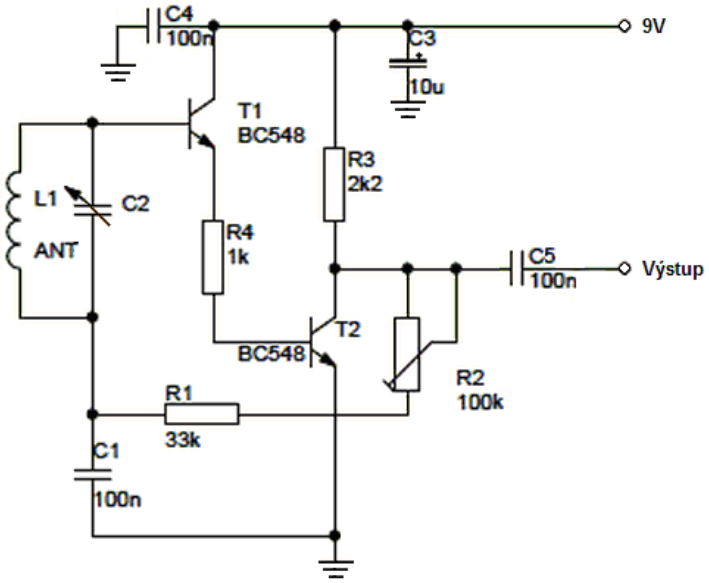
Úkolem vstupního přijímacího obvodu je příjem amplitudově modulovaného signálu, resp. výběr nosného kmitočtu pro požadovanou rozhlasovou stanici, jeho zesílení a odeslání na vstup AD převodníku. V úloze budeme přijímat rozhlasové stanice v pásmu středních vln (500 – 1600 kHz), kde lze nalézt např. rozhlasovou stanici ČRo Dvojka/ Plus

na frekvenci 639 kHz (vysílač Liblice, okres Kolín). Návrh jednoduchého přijímacího obvodu je uveden na obrázku 22. Pro výběr nosné frekvence se na vstupu nachází rezonanční obvod, který je tvořen feritovou anténou a ladícím kondenzátorem.

Rezonanční frekvence je dána vztahem:

$$f_R = \frac{1}{2\pi\sqrt{L_1 C_2}}$$

Pokud máme $L_1 = 200\mu H$, $C_{2\min} = 50\,pF$ a $C_{2\max} = 500\,pF$, potom dle zmíněného vztahu můžeme pomocí ladícího kondenzátoru naladit obvod na vybraný rezonanční kmitočet z rozsahu 500 – 1600 kHz.



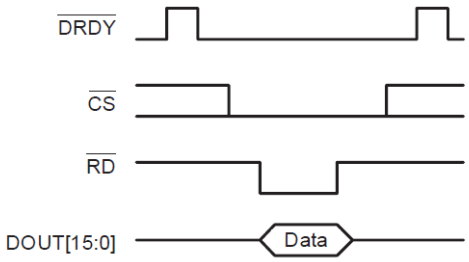
Obrázek 22 - Návrh zapojení pro vstupní přijímací obvod

Pro minimální zatížení rezonančního obvodu L1 a C2 je připojen tranzistor T1 jako emitorový sledovač zajišťující vysokou vstupní impedanci pro zesilovač tvořený tranzistorem T2 v zapojení se společným emitorem. Sériové zapojení R1 a R2 představuje regenerativní zpětnou vazbu, přičemž pomocí proměnného rezistoru R2 lze nastavit optimální zpětnou vazbu. V případě nadměrné zpětné vazby může dojít k nežádoucímu rozkmitání obvodu, naopak pokud je malá vazba, klesá citlivost přijímače. Napájecí napětí navrženého obvodu je 4,5 - 9V.

8.1.3 Analogově digitální převod a proces čtení dat z převodníku

Pro analogově digitální převod byl vybrán modul ADS1606EVM, jehož popis a vlastnosti obsahuje kapitola č. 4. Nyní si uveďme především informace potřebné pro úlohu AM přijímač. Dle známého vzorkovacího teorému musí být vzorkovací frekvence alespoň dvojnásobná oproti nejvyšší frekvenci vzorkovaného signálu ($f_{vz} = 2\,f_{\max}$), aby nedošlo k nežádoucímu zkreslení signálu. Převodník ADS1606

má vzorkovací frekvenci 5 MHz, což je zcela vyhovující pro vzorkování signálu rozhlasových stanic v pásmu středních vln (500 – 1600 kHz). Vzhledem k tomu, že na modulu ADS1606EVM operujeme s rozdílovým referenčním napětím V_{REF} = 3V, lze na vstup modulu přivést napětí v dovoleném rozsahu ±1,467V_{REF}, tedy ±4,4V. Pokud je tento rozsah překročen, rozsvítí se červená dioda na modulu převodníku. Převodník pracuje s vzorkovací frekvencí 5Mhz, což znamená, že každých 200ns je k dispozici nový vzorek na 16-bitovém výstupu, který potřebujeme přečíst. O tom, že jsou připravena nová data, informuje sestupná hrana data ready signálu z modulu převodníku, který je přiveden na jeden z GPIO vstupů LCDK modulu. V takovém případě je generováno přerušení a z rozhraní EMIFA je nastavením read signálu na stav low zahájeno čtení dat z výstupu modulu převodníku. Celý proces je znázorněn na obrázku 23 (v příkladu je rovněž chip select signál, který je v úloze přiveden na zem - trvale ve stavu low).



Obrázek 23 - Ukázka jednoho cyklu čtení dat z převodníku

8.1.4 Přenos dat do paměti LCDK modulu

V předešlé podkapitole jsme si zjednodušili situaci při přerušení iniciovaném sestupnou hranou data ready signálu na pouhé přečtení nových dat z modulu převodníku. Ve skutečnosti je ovšem při přerušení zahájen EDMA přenos, je provedeno přečtení nových dat a ta jsou poté přístupná v příslušném hardwarovém registru pro EMIFA. Tento registr je nastaven jako zdroj dat pro EDMA přenos, jsou nakonfigurovány rovněž i ostatní parametry pro EDMA přenos, je provedena inicializace EDMA rozhraní a je definován vstupní buffer jako cíl EDMA přenosu. Po těchto nezbytných krocích již může probíhat EDMA přenos, který v každém cyklu přenese 1 vzorek o 16 bitech z výstupu převodníku do definovaného bufferu. Při samotné AM demodulaci můžeme pracovat např. s posloupností 2048 vzorků, proto použijeme buffer s totožnou velikostí. Po naplnění bufferu vždy spustíme demodulaci a začneme jej plnit opět od začátku.

8.1.5 Digitální AM demodulace

V paměti máme buffer o zadané délce naplněný daty získanými z výstupu AD převodníku. Tato data představují digitalizovanou podobu přijatého amplitudově modulovaného signálu. Nyní tedy můžeme využít signálového procesoru ke zpracování a demodulaci vstupního signálu. Prvním krokem je aplikace pásmové propusti pro potlačení všech frekvencí kromě pásma požadovaného nosného kmitočtu. Tento krok není nezbytně nutný, pokud je dostatečná selektivita vstupního přijímacího obvodu.

Samotná digitální demodulace AM signálu je založena na jednoduchém součinu amplitudově modulovaného signálu s výrazem $\cos(\omega_n t)$.

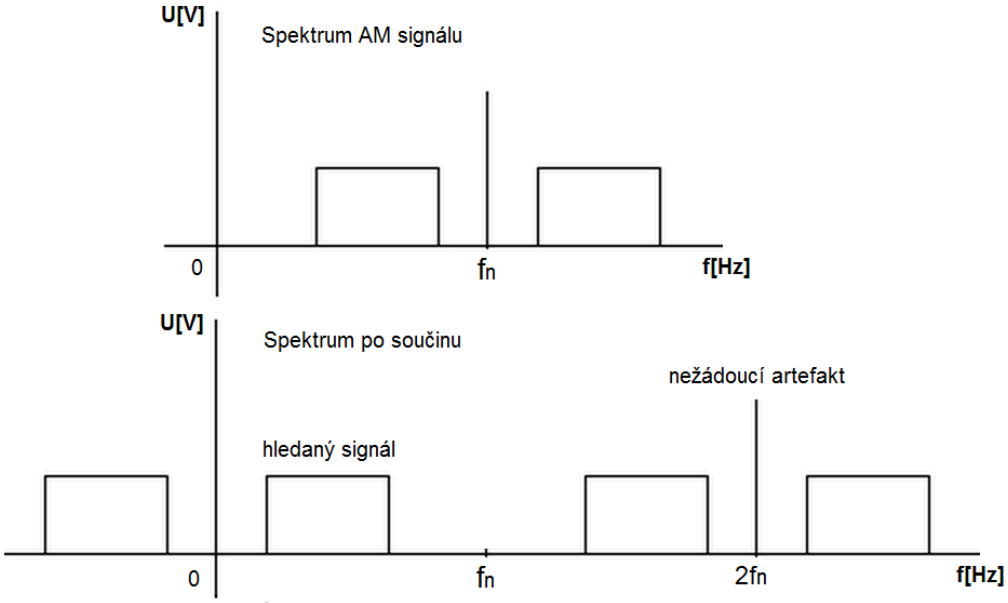
Připomeňme si vztah pro amplitudově modulovaný signál:

$$u(n) = [U_n + U_m \cos(\omega_m t)]\cos(\omega_n t)$$

Nyní tento signál vynásobme výrazem $\cos(\omega_n t)$ a dle goniometrické funkce $\cos\alpha\cos\beta = \frac{1}{2}[\cos(\alpha-\beta)+\cos(\alpha+\beta)]$ získáme:

$$u(n) = [U_n + U_m \cos(\omega_m t)]\cos(\omega_n t)\cos(\omega_n t) =$$
$$= [U_n + U_m \cos(\omega_m t)]\cdot\left[\frac{1}{2} + \frac{1}{2}\cos(2\omega_n t)\right] =$$
$$= \left[\frac{U_n}{2} + \frac{U_m}{2}\cos(\omega_m t)\right] + \left[\frac{U_n}{2} + \frac{U_m}{2}\cos(\omega_m t)\right]\cos(2\omega_n t)$$

Výsledkem součinu amplitudově modulovaného signálu s výrazem $\cos(\omega_n t)$ je signál, jehož amplitudové spektrum (viz obr. 24) obsahuje jednak původní nemodulovaný signál, ale také signál na nosné frekvenci $2f_n$, který je nežádoucí a potřebujeme jej potlačit.

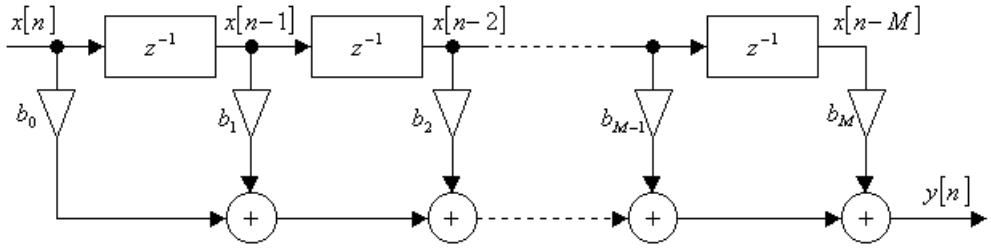


Obrázek 24 - Spektrum AM signálu po součinu s výrazem cos(ω_nt)

Pro odstranění nežádoucího signálu aplikujeme v dalším kroku číslicový FIR filtr dolní propusti pro potlačení frekvencí větších než $f_{m\max}$.

FIR filtry (finite impuls response) s konečnou impulsovou odezvou jsou vzhledem ke svým vlastnostem výhodné pro digitální zpracování signálu. Neobsahují zpětnou vazbu, jsou vždy stabilní a lze je realizovat pouze za pomoci základních operací násobení a sčítání. Pro dosažení vysoké strmosti je ovšem nutné vyššího řádu filtru oproti

IIR filtrům. Na obrázku 25 je uvedena přímá struktura FIR filtru. Příklad implementace filtru si ukážeme v kap. č. 8.2.3.



Obrázek 25 - Přímá struktura FIR filtru

Pro soustavu na obrázku platí diferenční rovnice:

$$y[n] = b_0 x[n] + b_1 x[n-1] + ... + b_M x[n-M]$$

Přenosová funkce soustavy je dána vztahem (v Z-transformaci):

$$H[z] = \frac{Y[z]}{X[z]} = b_0 + b_1 z^{-1} + ... + b_M z^{-M} = \sum_{i=0}^{M} b_i z^{-i}$$

8.1.6 Přenos zpracovaného signálu na audio výstup

Pro odesílání zpracovaných dat na audio výstup k poslechu si definujeme výstupní buffer v paměti, do kterého se ukládají data ve formátu vhodném pro odeslání na rozhraní McASP. Odeslání na McASP rozhraní je prováděno pomocí EDMA přenosu. Nový přenos je zahájen vždy po dokončení algoritmů pro zpracování signálu. Pro přehlednost si uveďme postup, jakým lze přenos zpracovaného signálu na audio výstup realizovat na LCDK modulu:

- Definování výstupních bufferů
- Inicializace I^2C rozhraní pro ovládání kodeku AIC3106 a rozhraní McASP
- Inicializace a konfigurace kodeku AIC3106
- Inicializace a konfigurace rozhraní McASP
- Inicializace a konfigurace rozhraní EDMA a DMA parametrů
- Inicializace řadiče přerušení
- Nastavení přerušení pro McASP a EDMA
- Konfigurace kodeku AIC3106 a McASP pro I^2S mód
- Aktivace procesu zpracování vstupního signálu a jeho uložení do paměti

Názorněji si jednotlivé body probereme v části zabývající se programovou implementací tohoto procesu. Značná část informací je uvedena v popisu úlohy spektrální analyzátor s tím rozdílem, že tam jsme prováděli konfiguraci vstupního i výstupního přenosu z audio kodeku (viz kap. č. 7.2.2 a 7.2.3)

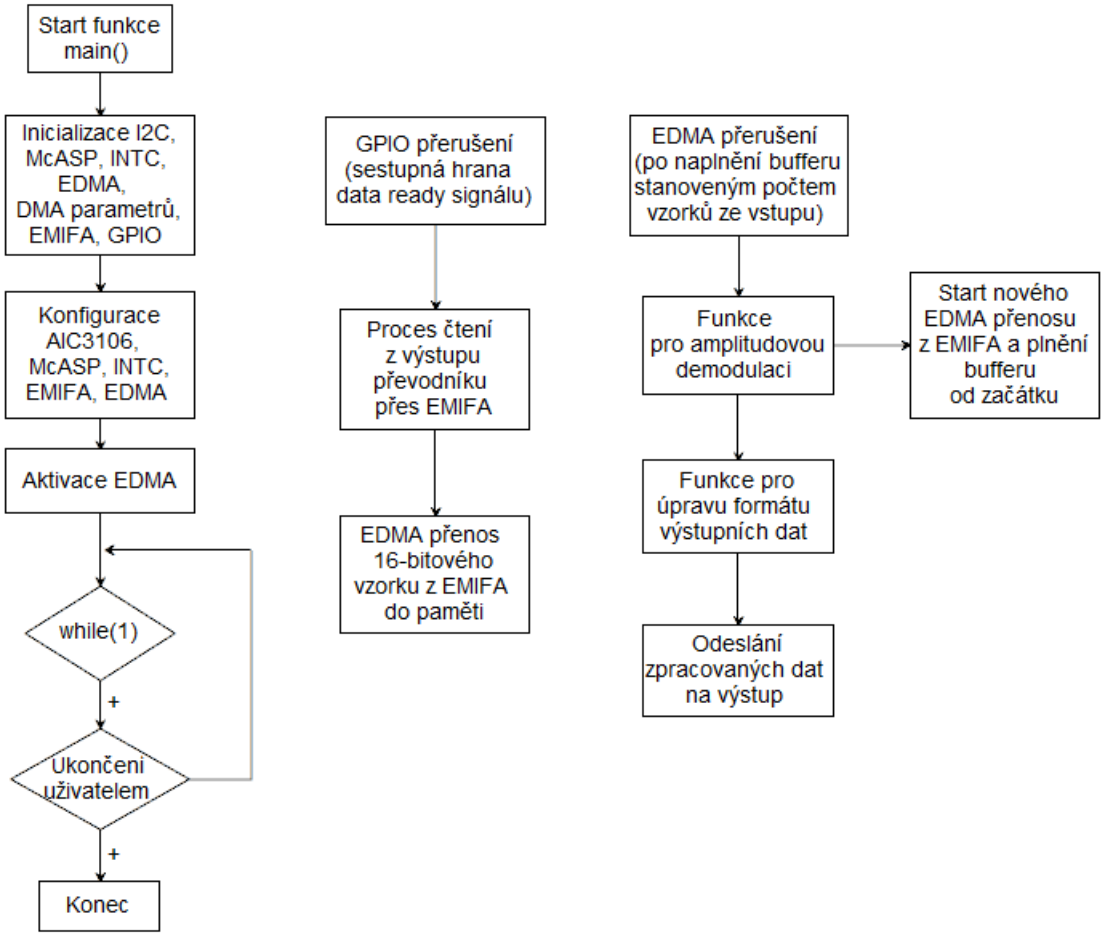
8.2 Implementace

V předešlých kapitolách jsme si popsali konfiguraci, zprovoznění a zapojení modulu převodníku ADS1606EVM (viz kap. 4 a 5.3) a vytvořili jsme návrh vstupního přijímacího obvodu připojovaného na vstup modulu převodníku pro výběr požadované nosné frekvence (viz kap. 8.1.2). Tato kapitola obsahuje popis programového řešení čtení dat z převodníku, popis způsobu přenosu a ukládání dat do paměti, návrh funkce pro AM demodulaci a odeslání zpracovaných dat na audio výstup LCDK modulu.

8.2.1 Struktura programu

Kromě zdrojového souboru **main.c**, který obsahuje hlavní část zdrojového kódu úlohy AM přijímač, jsou v projektu úlohy další podpůrné zdrojové soubory:

- **aic31.c** – obsahuje definici registrů a funkcí pro řízení kodeku AIC3106
- **codecif.c** – zahrnuje funkce pro konfiguraci I2C rozhraní, přes které je ovládán kodek AIC3106
- **interrupt.c** – obsahuje funkce pro správu událostí přerušení
- **EMIFAPinmuxSetup.c** – obsahuje funkci pro konfiguraci EMIFA pinmux



Obrázek 26 - Vývojový diagram úlohy 2 - AM přijímač

Na obrázku 26 je znázorněn vývojový diagram programu pro úlohu AM přijímač. V hlavní funkci **main()** se tentokrát nachází pouze volání inicializačních a konfiguračních funkcí, nekonečný cyklus **while(1)** neobsahuje žádný algoritmus. Významné kroky se zde dějí v rámci procedur přerušení. Nejprve je v případě sestupné hrany na vstupu GPIO5[11] generováno přerušení pro přečtení výstupních dat z modulu převodníku a pro zahájení nového EDMA přenosu z EMIFA rozhraní do paměti. V paměti je postupně plněn vstupní buffer, který má shodnou délku s parametrem `bCnt` nastaveného v paramsetu pro EDMA přenos z EMIFA. Při naplnění tohoto bufferu je `bCnt = 0`, EDMA přenos je kompletní a je generováno přerušení, po němž jsou vynulovány příslušné bity v EDMA registru IPR. Tyto bity vyjadřují, že byl dokončen EDMA přenos a tím, že je vynulujeme, dovolíme start nového EDMA přenosu. Dále je v rutinně tohoto přerušení volána funkce pro provedení samotné AM demodulace, na kterou navazují funkce pro transformaci dat do podoby vhodné k odeslání na výstup a funkce pro závěrečné odeslání dat na audio výstup prostřednictvím EDMA, McASP a kodeku AIC3106.

8.2.2 Čtení dat z převodníku a ukládání do bufferu v paměti

V hlavní funkci **main()** se nachází volání funkcí pro inicializaci a konfiguraci všech potřebných LCDK subsystémů. Pro provedení procesu čtení dat z modulu převodníku a jejich ukládání do bufferu v paměti konfigurujeme subsystémy GPIO, EMIFA, INTC a EDMA. V následujících podkapitolách si předvedeme způsob nastavení zmíněných subsystémů. Uvedené funkce opět obsahují parametry a registry, jejichž hodnoty jsou nadefinovány v použitých knihovnách z balíčku StarterWare.

8.2.2.1 Konfigurace GPIO vstupu

Pro konfiguraci GPIO rozhraní a vybraného GPIO pinu jako vstupního generujícího přerušení při sestupné hraně jsou ve funkci **main()** volány 4 funkce.

První je funkce **PSCInit()**, odkud je volána funkce **PSCModuleControl**() z knihovny psc.h pro spuštění GPIO rozhraní. Příklad a vysvětlení této funkce je uveden v popisu implementace úlohy spektrální analyzátor (viz kap. 7.2.2).

Druhou funkcí je **GPIOPinMuxSetup()**, kterou jsou konfigurovány I/O piny procesoru pro přístup k vybraným subsystémům, v tomto případě poskytuje tato funkce přístup k pinu 11 z GPIO banky 5. Funkce pracuje na úrovni systémových registrů, které nastavuje prostřednictvím funkce pro přístup k systémovým registrům **HWREG()** z knihovny hw_types.h.

Třetí volanou funkcí **InterruptInit()** je provedena inicializace INTC řadiče přerušení prostřednictvím funkce **IntDSPINTCInit()** z knihovny interrupt.h. Tato funkce zajistí, aby byla všechna maskovatelná přerušení na začátku vypnuta. Dále je odsud nastaveno globální povolení přerušení v procesoru, které je provedeno funkcí **IntGlobalEnable()** opět z knihovny interrupt.h.

V poslední funkci **AD1606InterruptInit()** je provedena konfigurace vybraného GPIO pinu a je povoleno přerušení pro tento pin. V úloze jsme pro přivedení data ready signálu z modulu převodníku zvolili pin 11 z GPIO banky 5, který je snadno dostupný na konektoru J14 LCDK modulu na EMIFA rozhraní jako jeden z adresových pinů. Vzhledem k tomu, že adresové piny na EMIFA nevyužijeme,

můžeme použít 1 adresový pin a přenastavit jeho funkci na GPIO vstup. Z **AD1606InterruptInit()** jsou volány následující 3 funkce:

```
GPIODirModeSet(SOC_GPIO_0_REGS, 92, GPIO_DIR_INPUT);
```

SOC_GPIO_0_REGS	základní adresa registrů pro GPIO
92	číslo pro vybraný GPIO pin
GPIO_DIR_INPUT	parametr pro nastavení GPIO pinu jako vstupu

```
GPIOIntTypeSet(SOC_GPIO_0_REGS, 92, GPIO_INT_TYPE_FALLEDGE);
```

SOC_GPIO_0_REGS	základní adresa registrů pro GPIO
92	číslo pro vybraný GPIO pin
GPIO_INT_TYPE_FALLEDGE	parametr pro nastavení přerušení generovaného při sestupné hraně vstupního signálu

```
GPIOBankIntEnable(SOC_GPIO_0_REGS, 5);
```

SOC_GPIO_0_REGS	základní adresa registrů pro GPIO
5	číslo GPIO banky

Nejprve je tedy pin 11 z GPIO banky 5 nastaven jako vstupní, poté pro tento vstup nastaveno přerušení závislé na sestupné hraně vstupního signálu a na závěr je povoleno přerušení pro celou GPIO banku 5. Nemusíme konfigurovat rutinu přerušení zvlášť pro pin 11 GPIO banky 5, protože EDMA přenos na C6748 LCDK lze spouštět v rámci GPIO pouze při události celé GPIO banky.

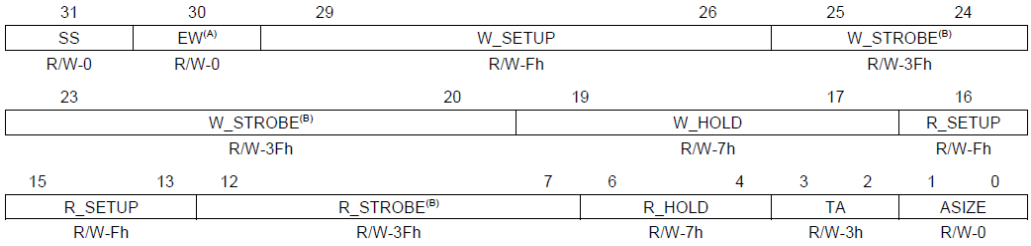
8.2.2.2 Konfigurace EMIFA rozhraní

EMIFA rozhraní, které využíváme pro propojení modulu převodníku a LCDK modulu, nejprve spustíme funkcí **PSCModuleControl**() volané z funkce **PSCInit()**.

Následuje funkce **EMIFASetup()**, která obsahuje konfiguraci pro EMIFA rozhraní. Na jejím začátku voláme funkci **EMIFAPinMuxSetup()**, která se nachází v podružném zdrojovém souboru EMIFAPinMuxSetup.c. Jedná se o upravenou verzi obdobné procedury z balíčku StarterWare. Funkce je upravena tak, abychom poskytli procesoru přístup pouze k těm pinům, které využijeme při spojení s modulem převodníku. Jsou to piny pro data D0 – D15, chip select CSn2 a read OEn. Funkce pracuje na úrovni systémových registrů jako obdobná funkce pro GPIO uvedená v předchozí kapitole.

EMIFA rozhraní provozujeme pro připojení modulu převodníku v asynchronním režimu. V tomto režimu lze k EMIFA připojit až 4 zařízení najednou, protože máme k dispozici 4 konfigurační registry se stejnou strukturou pro asynchronní režim CE2CFG – CE5CFG. Vzhledem k tomu, že máme jediné zařízení připojené k EMIFA, můžeme si vybrat, který registr použijeme. Signál chip select můžeme propojit s příslušným vstupem modulu převodníku, nebo jej na převodníku připojíme

přímo na zem, čímž jej pocháme trvale v aktivním stavu. Z konfiguračních registrů je v úloze používán registr CE2CFG. Jeho struktura je následující:



Obrázek 27 - Asynchronní konfigurační registr CE2CFG

Ve funkci **`EMIFASetup()`** se věnujeme především konfiguraci registru CE2CFG. K tomu jsou implementovány dále uvedené 4 funkce, na nichž si vysvětlíme jednotlivé parametry v registru CE2CFG. Před začátkem konfigurace je provedeno vynulování všech bitů registru CE2CFG. To je nutné, protože bez tohoto kroku by poslední z dále uvedených funkcí vzhledem ke své vnitřní syntaxi přestala fungovat.

```
EMIFAAsyncDevDataBusWidthSelect(SOC_EMIFA_0_REGS,EMIFA_CHIP_SELECT_2,
                                EMIFA_DATA_BUSWITTH_16BIT);
```

Touto funkcí je nastaven parametr ASIZE v registru CE2CFG na hodnotu 1, což znamená použití 16-bitové datové sběrnice pro spojení s modulem převodníku.

```
EMIFAAsyncDevOpModeSelect(SOC_EMIFA_0_REGS,EMIFA_CHIP_SELECT_2,
                                EMIFA_ASYNC_INTERFACE_NORMAL_MODE);
```

Tato funkce slouží k volbě mezi normálním a Strobe módem asynchronního rozhraní na EMIFA nastavením parametru SS. V normálním módu je chip select signál aktivní po celý cyklus čtení, zatímco ve Strobe módu je aktivní pouze po dobu tzv. Strobe periody v cyklu čtení, tzn. cca po dobu, kdy je současně aktivní read signál. Pro úlohu nastavíme normální mód, ovšem vzhledem k tomu, že chip select signál z EMIFA můžeme ponechat nezapojený, není pro nás tento parametr potřebný.

```
EMIFAExtendedWaitConfig(SOC_EMIFA_0_REGS,EMIFA_CHIP_SELECT_2,
                                EMIFA_EXTENDED_WAIT_DISABLE);
```

Další funkce ponechává parametr EW na hodnotě 0 pro vypnutí tzv. módu prodlouženého čekacího cyklu (Extended Wait Mode). Tento režim spočívá v tom, že lze dle potřeby využít EMA_WAIT pinu na rozhraní EMIFA k prodloužení doby Strobe periody, resp. doby pro čtení z externího zařízení.

```
EMIFAWaitTimingConfig(SOC_EMIFA_0_REGS,EMIFA_CHIP_SELECT_2,
              EMIFA_ASYNC_WAITTIME_CONFIG(1, 2, 1, 1, 2, 1, 0));
```

Poslední funkce nastavuje zbylých 7 parametrů registru CE2CFG, konkrétně W_SETUP, W_STROBE, W_HOLD, R_SETUP, R_STROBE, R_HOLD a TA.

Parametr TA představuje minimální počet hodinových cyklů EMIFA mezi 2 čtecími cykly. Nastavíme jej na 0. Další parametry představují doby pro cykly čtení a zápisu. Pro nás jsou důležité pouze parametry pro čtení, které jsou zadávány v počtech hodinových cyklů EMIFA rozhraní. EMIFA pracuje na frekvenci 25MHz, což znamená, že 1 hodinový cyklus trvá cca 40ns. Pokud tedy R_SETUP=1, R_STROBE=2 a R_HOLD=1, je výsledná doba celého cyklu čtení cca 160ns, přičemž Strobe perioda (doba, kdy je read signál aktivní, kdy probíhá čtení) trvá cca 80ns. Z převodníku přichází nový vzorek a sestupná hrana data ready signálu každých 200ns, což znamená, že při tomto nastavení jsme schopni číst každý vzorek bez vynechání a doba čtecího cyklu nepřesahuje dobu mezi dvěma data ready signály.

8.2.2.3 Konfigurace EDMA přenosu

V předchozích 2 kapitolách jsme si předvedli konfiguraci GPIO a EMIFA rozhraní při propojení s modulem převodníku. Až nyní si ukážeme, jak takto nakonfigurované subsystémy dále využijeme při implementaci EDMA přenosu. Pokud bychom přicházející data z modulu převodníku četli přímo s využitím procesoru bez EDMA, stíhali bychom číst cca každý 5. – 7. vzorek. Proto potřebujeme konfigurovat EDMA přenos, který bude spuštěn vždy při obdržení sestupné hrany data ready signálu a ve kterém bude současně zahrnuto čtení nových dat z výstupu modulu převodníku prostřednictvím EMIFA rozhraní. To EMIFA rozhraní umožňuje.

Postup pro inicializaci a konfiguraci EDMA přenosu je obdobný jako v úloze spektrální analyzátor. Nejprve ve funkci **`EDMA3Initialize`**`()` pomocí funkce `PSCModuleControl`() zapneme kanálový kontrolér EDMA3CC_0 a kontrolér přenosu EDMA3TC_0. Součástí je také inicializace EDMA3 řadiče ve funkci `EDMA3Init`() z knihovny edma.h.

Dále je provedena žádost o přidělení DMA kanálu pro událost vzniklou na GPIO bance 5, v níž se nachází vstupní pin pro data ready signál (GPIO5[11]). To provádí funkce z knihovny edma.h:

```
EDMA3RequestChannel(SOC_EDMA30CC_0_REGS, EDMA3_CHANNEL_TYPE_DMA,
              EDMA3_CHA_GPIO_BNKINT5, EDMA3_CHA_GPIO_BNKINT5, 0);
```

`SOC_EDMA30CC_0_REGS`	základní adresa registrů použitých pro EDMA3
`EDMA3_CHANNEL_TYPE_DMA`	typ kanálu, v našem případě DMA
`EDMA3_CHA_GPIO_BNKINT5`	číslo kanálu pro požadovanou událost
`EDMA3_CHA_GPIO_BNKINT5`	číslo kanálu, dle kterého je generováno přerušení po úspěšném, či neúspěšném dokončení přenosu
`0`	číslo fronty, ke které jsou mapovány žádosti DMA Master kanálu

Konfigurace, nastavení paramsetu a spuštění EDMA přenosu se děje ve funkci:

```
EmifaReceiveData(EDMA3_CHA_GPIO_BNKINT5, EDMA3_CHA_GPIO_BNKINT5,
                                                    emif_rbuffer);
```

Na začátku funkce `EmifaReceiveData()` je definován nový paramset dle struktury EDMA3CCPaRAMEntry z knihovny edma.h, jenž obsahuje deklaraci potřebných DMA parametrů a poskytuje vhodný formát pro jejich uživatelskou konfiguraci. Jednotlivé parametry EDMA přenosu z EMIFA rozhraní do bufferu `emif_rbuffer` jsou nadefinovány dle následujícího příkladu.

```
paramSet.srcAddr = SOC_EMIFA_CS2_ADDR;
paramSet.destAddr = (unsigned int) buffer;
paramSet.aCnt = 2;
paramSet.bCnt = 1000;
paramSet.cCnt = 1;
paramSet.srcBIdx = 0;
paramSet.destBIdx = 2;
paramSet.srcCIdx = 0;
paramSet.destCIdx = 2;
paramSet.linkAddr = 0xFFFFu;
paramSet.bCntReload = 0;
paramSet.opt = 0x00000000u;
paramSet.opt |= ((EDMA3CC_OPT_SAM) << EDMA3CC_OPT_SAM_SHIFT);
paramSet.opt |= ((tccNum << EDMA3CC_OPT_TCC_SHIFT) & EDMA3CC_OPT_TCC);
paramSet.opt |= (1 << EDMA3CC_OPT_TCINTEN_SHIFT);
```

Zdrojová adresa (srcAddr) pro přenos je nastavena na hardwarový registr, `SOC_EMIFA_CS2_ADDR` (0x60000000), v němž je vždy po cyklu čtení z modulu převodníku dostupný nový 16-bitový vzorek. Cílová adresa (destAddr) je nastavena ukazatelem na začátek bufferu `emif_rbuffer`. Další parametr aCnt = 2 (počet bytů v každém vzorku) vyjadřuje, že při každém EDMA přenosu spuštěném sestupnou hranou data ready signálu na GPIO vstupu, jsou přeneseny první 2B, tzn. prvních 16b ze zdrojové adresy `SOC_EMIFA_CS2_ADDR`, tedy právě 1 právě přečtený vzorek. Druhý parametr bCnt = 2048 (počet vzorků v každém slotu) pro druhou dimenzi přenosu nám říká, že přenos 2B vzorku bude proveden 2048x a vzhledem k tomu, že parametr cCnt (počet slotů v každém rámci) pro třetí dimenzi je roven 1, tak po provedení 2048 takových přenosů je teprve vyvoláno přerušení dané kompletním dokončením EDMA přenosu. Parametr srcBIdx (index mezi po sobě následujícími vzorky zdroje) je nastaven na 0, protože zdrojovou adresou je hardwarový registr a s dalšími vzorky se v něm nechceme nikam posouvat. Naopak parametr destBIdx (index mezi po sobě následujícími vzorky cíle) je nastaven na 2, což znamená, že každý další vzorek uložíme o 2B dále v bufferu `emif_rbuffer`. Parametry srcCIdx a destCIdx jsou nastaveny totožně a vzhledem k tomu, že třetí dimenze není využita, nejsou tyto parametry pro EDMA přenos v úloze podstatné. Následuje parametr linkAddr, který představuje link na další paramset po dokončení EDMA přenosu. V rámci tohoto přenosu používáme pouze 1 paramset, proto je tento parametr rovněž nevyužit. Stejná situace je i s parametrem bCntReload, který slouží k přehrání hodnoty bCnt parametru, když dosáhne parametr bCnt hodnoty 0. Ani tento parametr nepotřebujeme. Poslední je OPT pole setu parametrů, kde je především v posledním řádku povoleno EDMA3 přerušení a je nastaven DMA kanál (tccNum = `EDMA3_CHA_GPIO_BNKINT5`), dle kterého je generováno přerušení po úspěšném, či neúspěšném dokončení EDMA přenosu.

Po nastavení paramsetu voláme funkci pro nastavení právě nakonfigurovaného paramsetu pro zadaný DMA kanál:

```
EDMA3SetPaRAM(SOC_EDMA30CC_0_REGS, EDMA3_CHA_GPIO_BNKINT5,
                                        &paramSet);
```

Poslední funkce slouží k aktivaci EDMA přenosu v režimu spouštění v závislosti na události na GPIO bance 5 (sestupná hrana na pinu GPIO5[11]):

```
EDMA3EnableTransfer(SOC_EDMA30CC_0_REGS, EDMA3_CHA_GPIO_BNKINT5,
                                        EDMA3_TRIG_MODE_EVENT);
```

8.2.3 Návrh funkce pro digitální AM demodulaci

Vzhledem k tomu, že algoritmus pro provedení digitální AM demodulace není v čase dokončení této práce dokončen, uvedeme si v této kapitole pouze postup, jakým jej později implementujeme a odladíme. Vstupem pro tento algoritmus je buffer `emif_rbuffer` se vstupním digitalizovaným modulovaným AM signálem. Impulzem pro provedení algoritmu demodulace je přerušení generované vlivem dokončení EDMA přenosu dat z EMIFA rozhraní a naplnění bufferu `emif_rbuffer`. Obsah bufferu se přenese do vstupního pracovního bufferu `demod_inbuffer` a buffer `emif_rbuffer` se začne plnit od začátku. V algoritmu pro digitální AM demodulaci postupujeme dle kroků, které jsme si uvedli v teoretickém rozboru (viz kap. 8.1.5). Nejprve tedy provedeme součin vstupního modulovaného signálu s výrazem $\cos(\omega_n t)$, tedy s jeho nosnou vlnou. Abychom mohli provést součin nosné vlny s modulovaným signálem, potřebujeme nejprve oddělit nosnou vlnu od modulovaného signálu a až poté provést jejich součin. Získání digitální nosné vlny dosáhneme aplikováním číslicového FIR filtru typu pásmové propusti pro výběr požadované nosné frekvence. V programu toto provedeme pomocí funkce **`DSPF_sp_fir_gen()`** z knihovny DSPLIB:

```
void DSPF_sp_fir_gen (const float *x, const float *h, float *y,
                                    int nh, int ny);
```

`*x`	ukazatel na buffer se vstupními daty, musí mít velikost $ny + nh - 1$
`*h`	ukazatel na buffer s koeficienty filtru, pole musí mít nh koeficientů reverzně seřazených $\{h(nh-1),...,h(1),h(0)\}$
`*y`	ukazatel na výstupní buffer, musí mít velikost ny
`nh`	počet koeficientů filtru, musí být dělitelný 4 a větší než 0
`ny`	počet prvků na výstupu, musí být dělitelný 4 a větší než 0

Vstupním bufferem je `demod_inbuffer`, výstupní buffer je `carrier_buffer` a koeficienty filtru a jejich počet (řád filtru) určíme pomocí nástroje FDATOOL v prostředí MATLAB a uložíme je do bufferu `h_buffer`.

V dalším kroku provedeme v jednoduchém for cyklu potřebný součin:

```
demod_inbuffer[i] = demod_inbuffer[i] * carrier_buffer[i]
```

Poslední operací je aplikace číslicového FIR filtru typu dolní propust pro oddělení původního nemodulovaného signálu od nežádoucí vysokofrekvenční složky. Opět uplatníme funkci **`DSPF_sp_fir_gen`**`()` z knihovny DSPLIB, přičemž koeficienty filtru požadované dolní propusti stanovíme s využitím nástroje FDATOOL v prostředí MATLAB. Výstupní buffer funkce a celého algoritmu pro digitální AM demodulaci je `demod_outbuffer`.

8.2.4 Přenos zpracovaného signálu na audio výstup

Proces konfigurace přenosu výstupních dat z bufferu `demod_outbuffer` na audio výstup probíhá stejným způsobem, jaký jsme již použili a popsali při implementaci úlohy spektrální analyzátor (viz kap. 7.2.2). Použijeme rovněž totožnou strukturu výstupního audio bufferu, proto před zahájením EDMA přenosu nejprve převedeme data do požadovaného formátu (viz kap. 7.2.3). Je nutné si dát pozor především na nastavení vzorkovací frekvence audio kodeku AIC31 (dovolená vzorkovací frekvence 8 – 96 kHz), aby korespondovala se vzorkovací frekvencí námi demodulovaného signálu. V opačném případě musíme před odesláním dat do audio kodeku upravit vzorkovací frekvenci demodulovaného signálu na požadovanou úroveň pomocí interpolace/decimace signálu. Impulzem pro spuštění EDMA přenosu dat z výstupního bufferu na rozhraní McASP je naplnění výstupního audio bufferu novými daty.

8.3 Zhodnocení

Cílem úlohy AM přijímač je vytvořit AM demodulátor, kde celý proces AM demodulace probíhá na DSP. Nicméně úloha AM přijímač nebyla k datu odevzdání této práce prakticky dokončena. Pro nevyřešené části úlohy byl v rámci rozboru úlohy proveden principiální návrh, který se však od výsledné podoby může lišit dle nově zjištěných skutečností. Zejména se jedná o kapitolu návrhu vstupního přijímacího obvodu a kapitolu návrhu procedury pro provedení digitální AM demodulace. Tyto 2 části nemohly být řešeny před dokončením potřebného propojení modulu převodníku ADS1606EVM s LCDK modulem prostřednictvím rozhraní EMIFA s funkčním EDMA přenosem do paměti.

Naopak dokončeno bylo spojení modulu převodníku ADS1606EVM a LCDK modulu s využitím rozhraní EMIFA. Byl vyřešen způsob inicializace a konfigurace EMIFA rozhraní pro asynchronní režim s 16-bitovým paralelním rozhraním. Při testování AD převodu byl na vstupu modulu převodníku používán generátor funkcí s nejvyšší frekvencí 10kHz. Pro zprovoznění modulu převodníku je nutné zajistit požadované napájení a nastavit řadu jumperů a DIP switch přepínače (viz kap. 5.3). Při testování modulu převodníku bylo dále zjištěno, že pro spolehlivou funkcionalitu je vhodné použít externí zdroj referenčního napětí, proto jsme připojovali na vstup J7 zdroj napětí 12 Vdc.

Stav úlohy k datu odevzdání této práce je takový, že právě testujeme vytvořený EDMA přenos z EMIFA rozhraní do bufferu v paměti. Spuštění EDMA přenosu se děje

vlivem přerušení daném sestupnou hranou data ready signálu z modulu převodníku na GPIO vstupu LCDK modulu. Konfigurace GPIO vstupu a přerušení dané sestupnou hranou data ready signálu je odzkoušeno, rovněž jsou generovány čtecí signály pro aktivaci výstupu převodníku a je zahájeno čtení dat z výstupu modulu převodníku. Přečtené vzorky jsou přístupné z systémového registru CE2CFG na EMIFA, odkud probíhá EDMA přenos.

Dalším krokem v řešení úlohy bude využití FIFO režimu na výstupu modulu převodníku. To modul převodníku umožňuje, požadovaná úroveň FIFO lze nastavit pomocí 3 pinů na DIP switch přepínači. Zároveň s tím bude nutné upravit režim čtení z EMIFA rozhraní tak, aby při každé sestupné hraně data ready signálu byla v zadaném intervalu provedena posloupnost stanoveného počtu čtecích impulzů v daném neměnném intervalu. V tomto případě budeme při EDMA přenosu přenášet více vzorků najednou dle nastavené úrovně FIFO. Dále bude provedena realizace vstupního přijímacího obvodu a bude odladěn algoritmus pro digitální AM demodulaci. Úspěšným výsledkem celé úlohy má být poslech naladěné rozhlasové stanice na audio výstupu LCDK modulu. Na výsledné podobě úlohy budeme po odevzdání této práce i dále pracovat, aby byla dokončena v rámci následujícího semestru a mohla být použita při výuce.

9 Úloha č. 3 – Cannyho hranový detektor

9.1 Teoretický rozbor

Třetí úlohou řešenou na modulu C6748 LCDK je Cannyho hranový detektor. Cílem této úlohy je implementovat na LCDK modulu funkční algoritmus pro provedení detekce hran Cannyho metodou. Schéma zapojení a nastavení Composer Studia pro tuto úlohu jsme si ukázali již v kapitole 5.4 a v kapitole 6. Vstupními daty pro tuto úlohu jsou kamerou zachycené snímky, které ukládáme v bufferech do paměti LCDK. Na tyto snímky aplikujeme úplný algoritmus pro Cannyho hranový detektor a výsledný obraz složený z detekovaných hran původního snímku zobrazujeme na monitoru. Na LCDK pro tuto úlohu využijeme včetně DSP procesoru subsystémy video dekodér TVP5147M1, VPIF, EDMA, řadič přerušení, PSC, paměť DDR2, LCDC řadič a video DAC THS8135.

Před samotnou implementací úlohy si popíšeme proces zpracování zachyceného snímku a jeho uložení do paměti. Podíváme se na video formáty dat, v nichž jsou ukládány a zobrazovány jednotlivé snímky, vysvětlíme si princip Cannyho hranového detektoru a zmíníme také detektor hran založený na Sobelově operátoru. Sobelův detektor je rovněž součástí implementace, aby poskytl příležitost k přímému srovnání výsledku s Cannyho detektorem. Na možnost srovnání ať již s původním snímkem, nebo s výsledkem jiného detektoru byl při implementaci kladen důraz. Proto je možné si zobrazovací prostor monitoru rozdělit na více částí a v těchto částech zobrazovat v režimu zrcadlení totožné, avšak odlišně zpracované části snímku. Popsán je také proces zobrazení zpracovaných snímků na displeji.

9.1.1 Proces zpracování zachyceného snímku

Novinkou oproti předešlým dvěma úlohám je zde využití periférií a subsystémů pro uložení snímků z připojené videokamery do paměti na LCDK. Data z kamery jsou přiváděna přes kompozitní vstup a postup pro zpracování a uložení zachyceného snímku lze shrnout do těchto bodů:

- Alokace a inicializace potřebných video bufferů
- Inicializace I^2C rozhraní pro ovládání TVP5147
- Inicializace a konfigurace video dekodéru TVP5147
- Inicializace a konfigurace VPIF
- Inicializace adres bufferů pro první snímek
- Inicializace řadiče přerušení
- Aktivace VPIF pro zachycení nového snímku

Po těchto krocích máme funkční proces zachycení a zpracování nových snímků. Ty se ukládají do zvoleného bufferu v paměti ve formátu YCbCr 4:2:2. V další podkapitole se budeme věnovat právě tomuto formátu.

9.1.2 Barevný model YCbCr 4:2:2

Barevný model YCbCr představuje způsob, jakým jsou v paměti LCDK kódovány zachycené snímky z videokamery. V tomto modelu představuje Y jasovou (tzv. luma) komponentu a Cb, Cr jsou chrominanční komponenty zastupující barevné složky. Dvě chrominanční komponenty se používají z důvodu úspory paměťového místa (o 50%). Lidské oko je totiž citlivé na malé změny jasu v obraze, ale již nedokáže rozlišit malé změny v chrominančních signálech. Proto lze některé chrominanční složky vyřadit, aniž by se to viditelně projevilo na výsledné kvalitě obrazu.

V úloze pracujeme s modelem YCbCr 4:2:2, u kterého na každé 4 jasové vzorky připadají 2 vzorky Cb a 2 vzorky Cr. Ke všem Y vzorkům je tedy přiřazena jedna z chrominančních složek, přičemž druhá chrominanční složka je převzata z předcházejícího vzorku. Způsob uložení video signálu v YCbCr do bufferu v paměti LCDK je patrný z obrázku 28.

Y_1	Cb_1	Y_2	Cr_1	Y_3	Cb_2	Y_4	Cr_2

Obrázek 28 - Způsob uložení dat v barevném modelu YCbCr

Přepočet barev z RGB modelu pro vytvoření jednotlivých složek YCbCr se provádí podle těchto doporučených vztahů:

$$Y = 0.299R + 0.587G + 0.114B$$
$$Cb = -0.1687R - 0.3313G + 0.5B + 128$$
$$Cr = 0.5R - 0.4187G - 0.0813B + 128$$

9.1.3 Barevný model RGB 5:6:5

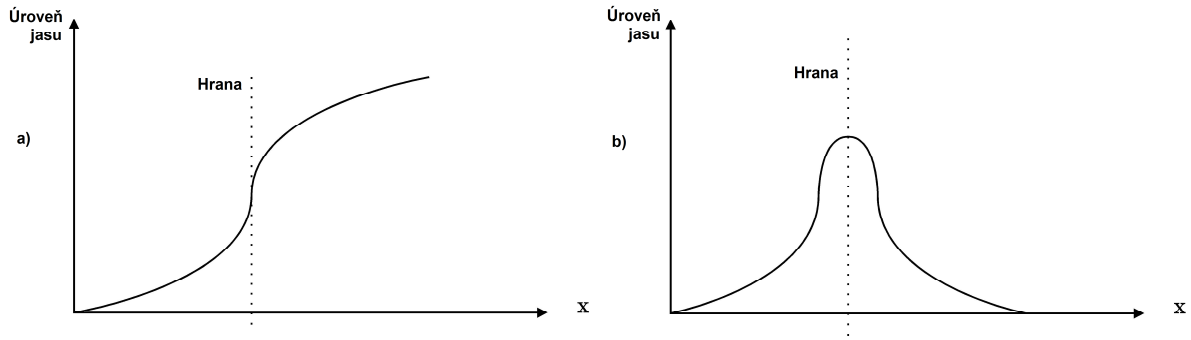
Barevný model RGB 5:6:5, často označovaný také jako High Color, představuje digitální způsob uložení RGB informace dle znalostí o vnímání barev lidským okem. Je ověřeno, že lidské oko běžně nerozezná více než 64 odstínů jedné barvy (výjimku tvoří pouze zelená barva). Proto byl vytvořen model RGB 5:6:5 pro uložení jedné barevné kombinace tří základních barev do 16 bitů (5 bitů pro červenou, 6 bitů pro zelenou a 5 bitů pro modrou barvu). Celkově lze v 16 bitovém High Color módu zobrazit až 65 536 barev.

V úloze model RGB 5:6:5 využijeme při zobrazení snímků na monitoru a při aplikaci hranových detektorů na jednotlivé snímky. Proto je po uložení zachyceného snímku do bufferu v paměti provedena konverze z modelu YCbCr 4:2:2 do RGB 5:6:5. Veškeré další operace jsou tedy prováděny s tímto barevným modelem.

9.1.4 Hranové detektory

Hranové detektory se používají pro nalezení a zobrazení takových míst v obraze, kde se nacházejí hrany. Hrany mohou vznikat na rozhraní dvou objektů, různou hloubkou, odrazivostí povrchu, odlesky, vlivem stínů, apod. V obraze lze za hranu považovat takové místo, v němž se strmě mění úroveň jasu. Hrana popisuje rychlost změny a směr největšího růstu jasové funkce f(x, y). Skládá se z hranových bodů, což jsou pixely, které mají velkou hodnotu gradientu (vektor, jehož směr udává směr největší změny jasu v daném místě a jehož velikost odpovídá velikosti změny jasu). Čím větší je hodnota gradientu v daném

místě, tím větší je předpoklad, že se jedná o hranu a tím ostřejší je to hrana. Ukázka jasové funkce hrany je patrná na obrázku 29a.



Obrázek 29 - Příklad a) jasové funkce hrany, b) 1. derivace hrany

Pro detekci hran v obraze existují metody založené na:

- hledání prvních derivací a jejich maxim (Sobel, Canny)
- hledání průchodů druhých derivací nulou (Marr-Hildreth)
- lokální aproximaci obrazové funkce parametrickým modelem, např. polynomem dvou proměnných (Haralick)

V úloze budeme implementovat především Cannyho hranový detektor, který pro nalezení hran využívá metodu hledání maxim prvních derivací. Pro tuto metodu existuje několik operátorů, z nichž často používaný je operátor Sobelův.

9.1.4.1 Sobelův hranový detektor

Příklad první derivace jasové funkce hrany je uveden na obrázku 29b. Provedením první derivace jasové funkce získáme její gradient, tedy směr a velikost největší změny:

$$|\nabla f(x,y)| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}\,, \qquad \varphi = arctg(\frac{\partial f / \partial y}{\partial f / \partial x})$$

Pro diskrétní vyjádření gradientu lze tento vztah přibližně nahradit rozdílem hodnot úrovní jasu 2 sousedních pixelů např. ve směru x:

$$\nabla f(x) = \frac{\partial f}{\partial x} \approx f(x+1, y) - f(x, y)$$

To je již způsob, jakým je v úloze implementován hranový detektor založený na hledání prvních derivací. Výběr sousedních pixelů, které jsou od sebe odečítány v každém bodě obrazu, řeší tzv. konvoluční maska. Používají se např. Robertsův operátor (značně náchylný na šum), Prewittové operátor (všem bodům v daném směru přikládá stejnou váhu), nebo Sobelův operátor (přikládá větší váhu bodům blíže ke středu masky, díky čemuž má lepší odolnost proti šumu).

Ke stanovení gradientu všech bodů obrazu využijeme principu diskrétní konvoluce. To je lineární operace, při které využíváme konvoluční masky (matice např. o velikosti 3x3) s definovanými koeficienty. Vybranou masku (zvolený operátor) přikládáme k obrazu a pohybujeme se po jednotlivých pixelech. Pro každý pixel se provádí součin koeficientů masky s příslušnými sousedícími pixely a výsledná hodnota gradientu pixelu je dána sumou těchto součinů.

Obecný zápis diskrétní konvoluce:

$$f(x,y) * h(x,y) = \sum_{i=-k}^{k} \sum_{j=-l}^{l} f(x+i, y+j) \cdot h(i,j),$$

kde f(x,y) je vstupní hodnota pixelu (x,y), h(x,y) je konvoluční maska přiřazená pixelu (x,y) a i, j (-k až +k, -l až l) jsou meze, které vyjadřují počet sloupců a řádků použité konvoluční masky, tedy prostor kolem pixelu (x,y), který bude využíván při konvoluci.

Konvoluční maska 3x3 pro Sobelův operátor:

ve směru x: $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$, y: $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ a ↗: $\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$

Z konvoluční masky pro Sobelův operátor je zřejmé, že větší váha je kladena bodům blíže ke středu (narozdíl od operátoru Prewittové, kde je váha všech bodů stejná), což pomáhá při detekci zašumělých hran. Dle toho, jakou masku použijeme, získáme gradient jasové funkce v určitém směru (operátory ve směru x, y a diagonálním).

Dle velikosti vypočtených gradientů se rozhodujeme, zda se jedná, či nejedná o hranu. To lze určit pevně nastaveným prahem, případně práh nastavovat automaticky (vysvětleno v části implementace úlohy). Ve výsledku budou jako hrany zobrazeny pouze ty obrazové body, jejichž gradient přesáhl stanovený práh. Nastavená úroveň prahu hraje velkou roli v ostrosti a pravdivosti detekovaných hran. Pokud je práh nastaven nízko, mohou být zobrazené hrany příliš široké, navíc se např. vlivem šumu mohou zobrazovat i hrany, které hranami nejsou. Naopak při vysoko nastaveném prahu může dojít ke ztrátě částí, nebo i celých hran.

9.1.4.2 Cannyho hranový detektor

Z hlediska hranové detekce jsme si vysvětlili metodu pomocí první derivace na příkladu Sobelova operátoru. Tyto detektory však nesplňují všechny požadavky, které od detektorů hran očekáváme:

- detekce všech hran v obraze s co nejmenším počtem chyb
- přesnost polohy detekované hrany
- jednoznačnost při detekci hrany (hrany nesmí být zdvojené)

U detekce hran pomocí prvních derivací je zejména patrná citlivost na šum, nejednoznačnost a některé hrany nejsou často vůbec nalezeny (velmi závisí na nastavení prahu). Cannyho algoritmus pro detekci hran je proto navržen tak, aby potlačil zmíněné negativní jevy a správně detekoval všechny hrany v obraze.

Popišme si nyní princip algoritmu pro Cannyho hranový detektor. Skládá se z těchto částí:

a) Aplikace Gaussova filtru pro potlačení šumu

Pro vyhlazení obrazu a potlačení šumu nejprve aplikujeme na snímek Gaussův filtr, což je prováděno pomocí diskrétní 2D konvoluce. Používáme konvoluční masku s gaussovským rozložením koeficientů pro výpočet nových hodnot všech pixelů v obraze. V tomto rozložení směrem ke středu masky koeficienty rostou a jejich hodnoty odpovídají průběhu Gaussovy křivky.

Dvourozměrné Gaussovo rozložení je dáno vztahem:

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

kde x, y jsou souřadnice pixelu a σ je směrodatná odchylka (parametr strmosti funkce – standardně od 1 do 1,4).

Pomocí tohoto vztahu jsou tedy vypočteny koeficienty konvoluční masky, která je poté aplikována na celý obraz. Ukázku konvoluční masky použité v úloze lze nalézt v části implementace úlohy.

b) Výpočet velikosti a směru gradientu

Zde využijeme metodu první derivace vysvětlenou v předcházející kapitole. Z dostupných operátorů zvolíme Sobelův, protože je méně náchylný na šum. Jsou vypočteny velikosti gradientů pro dvě masky - ve směru x a ve směru y. Celková velikost gradientu je rovna:

$$G(x,y) = \sqrt{G_x^2 + G_y^2}$$

Směr gradientu vyjadřující orientaci hrany v daném bodě je vypočítán jako:

$$\varphi(x,y) = arctg\,\frac{G_x}{G_y}$$

Směr gradientu je dále aproximován do jedné ze čtyř možných hodnot (0°, 45°, 90°, 135°), čímž si zajistíme, že detekované hrany bude možné zobrazit v maticovém uspořádání pixelů. Při implementaci tohoto kroku jsou výsledkem dvě pole - jedno s uloženou velikostí gradientů a druhé s uloženým směrem gradientů.

c) Nalezení lokálních maxim – ztenčení

Po zjištění velikosti a směru gradientů jednotlivých pixelů snímku jsou v tomto kroku vybírány pouze lokální maxima gradientů a jsou odebrány body, které nejsou maximem (nonmaximum suppression). Tím je zajištěno, že hrana bude detekována v místě nejvyššího gradientu z oblasti, kterou prochází. U každého zkoumaného pixelu se testují sousední pixely, které jsou ve směru a proti směru jeho gradientu. Pokud jsou velikosti gradientů těchto sousedních pixelů nižší než velikost gradientu zkoumaného pixelu, je tento pixel uznán jako hrana. Například v případě vodorovné hrany musí být velikost gradientu horních a dolních sousedních pixelů nižší, aby byly zkoumané pixely označeny jako hrana. Pokud není podmínka splněna, je zkoumaný pixel označen jako bod, kterým neprochází hrana. Tímto krokem jsme stanovili, kde přesně se v obraze nacházejí hrany a zařídili jsme, aby nebyly zobrazovány paralelní hrany, což způsobilo ztenčení a zpřesnění všech detekovaných hran.

d) Prahování s hysterezí

V klasickém Sobelově detektoru často dochází k tomu, že jsou buď detekovány hrany, které nejsou hranami, nebo se nepovede některé hrany detekovat vůbec. To velmi závisí na nastavení prahové úrovně.

V Cannyho algoritmu je tento nedostatek vyřešen provedením tzv. prahování s hysterezí. Z předešlého kroku máme detekovány hrany, ovšem nepřidělili jsme jim žádný význam. To je nutné, protože předchozí krok detekuje i ty nejmenší hrany, které mají své lokální maximum. Proto jsou při prahování s hysterezí nastaveny dvě prahové úrovně - minimální a maximální. Pokud je velikost gradientu v místě označeném jako hrana větší než maximální práh, potom je přímo bez dalších kroků označen jako hrana pro zobrazení na monitoru. Jestliže leží hodnota mezi minimálním a maximálním prahem, potom je bod uznán jako hrana pouze v případě, že sousedí s bodem, který již byl jako hrana označen dříve. Všechny ostatní pixely jsou označeny jako místa, kterými neprochází žádná hrana.

Výhody Cannyho hranového detektoru oproti detektorům založených na metodě první derivace jsou především:

- znatelně lepší odolnost proti šumu
- snížení počtu chybně detekovaných hran
- jednoznačnost a přesnost (detekovány hrany s tloušťkou jednoho pixelu)

9.1.5 Proces zobrazení zpracovaných snímků na displeji

Pro závěrečné zobrazení snímků s detekovanými hranami na monitoru je využit zcela totožný postup jako u úlohy spektrální analyzátor (viz kapitola 7.1.5).

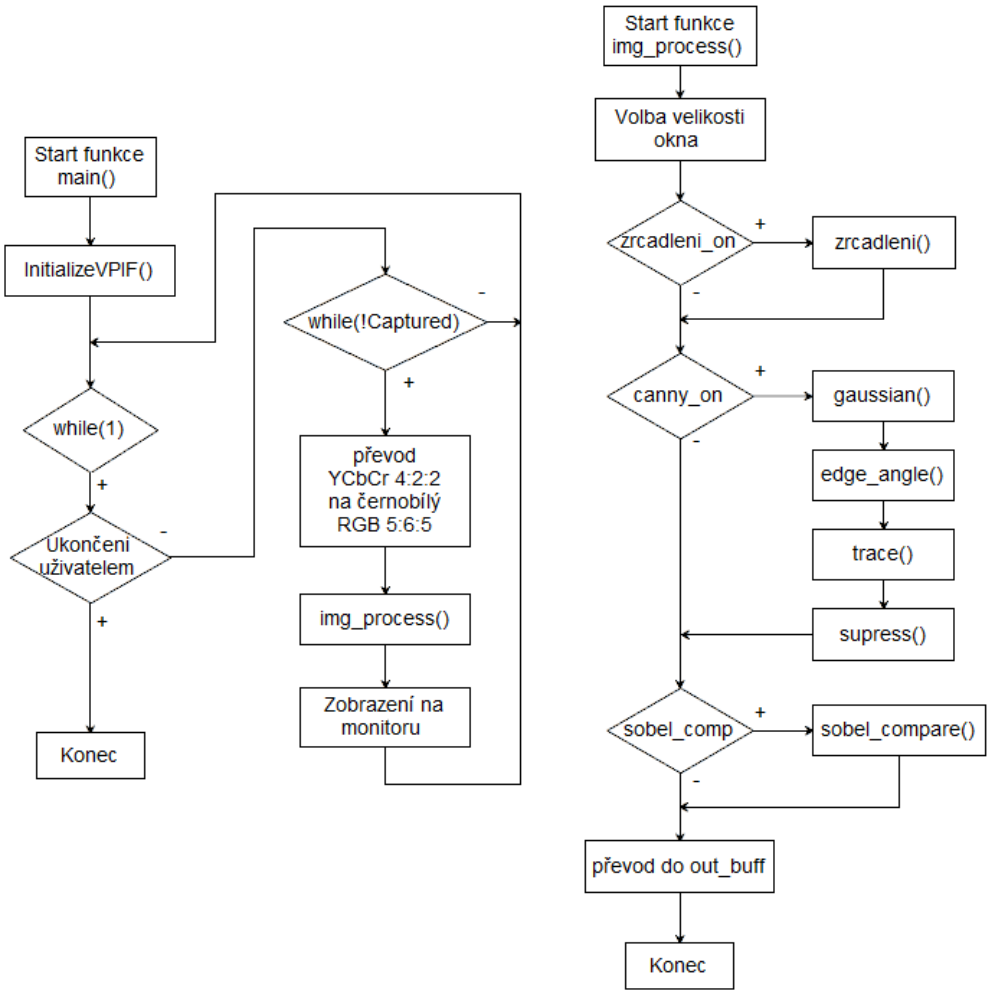
9.2 Implementace

Implementace úlohy Cannyho hranový detektor je věnována především samotnému řešení Cannyho algoritmu. Nejprve si v této kapitole popíšeme strukturu projektu úlohy a ukážeme si vývojový diagram hlavní části programu. Dále si nastíníme proces zpracování a uložení snímku do paměti z demonstrační úlohy vpif_lcd_loopback. Následuje hlavní část s popisem Cannyho algoritmu a doplňkových funkcí. Proces zobrazení zpracovaných dat na monitoru je velmi podobný tomu z úlohy spektrální analyzátor, zde si proto ukážeme pouze místo, odkud probíhá přenos dat z výstupních video bufferů. Na závěr si i u této úlohy předvedeme ukázku z výsledné podoby programu.

9.2.1 Struktura programu

Nejpodstatnější část úlohy Cannyho hranový detektor se nachází ve zdrojových souborech **main.c** a **process.c**, což jsou jediné dva zdrojové soubory v projektu úlohy. Další funkce jsou zkompilované v knihovnách vpif_lab.lib a img_algs_vs.lib.

Na obrázku 30 je znázorněn vývojový diagram konečné podoby programu v hlavní funkci `main()` a v procesní funkci `img_process()`.



Obrázek 30 - Vývojový diagram úlohy 3 - Cannyho hranový detektor

Pro zachycení a uložení dat z kamery a pro zobrazení zpracovaných dat na monitoru je využita knihovna od pana inženýra Svobody, která se používá ve výuce signálových procesorů. V knihovně je obsažena ukázková úloha vpif_lcd_loopback z balíčku StarterWare. Tato demonstrační úloha je upravena tak, aby bylo možné ji volat pomocí jediné funkce **`InitializeVPIF()`**, čímž docílíme veškerých inicializačních a konfiguračních procedur pro zpracování a uložení snímků z videokamery i pro zobrazení snímků na monitoru. Druhá převzatá knihovna je img_algs_vs.lib, v níž se nachází funkce **`cbcr422sp_to_rgb565_gray()`** pro převedení uloženého snímku v paměti z modelu YCbCr 4:2:2 do černobílého RGB 5:6:5. Pro Cannyho algoritmus potřebujeme pouze jasovou informaci, proto dále pracujeme pouze s černobílým obrazem.

Po uložení snímku do paměti a převodu do černobílého formátu je volána funkce **`img_process()`**, která obsahuje veškeré operace, které se snímkem provedeme, než ho pošleme na výstup k zobrazení. Součástí této funkce je zejména Cannyho algoritmus, ale nachází se zde i funkce pro možnost zrcadlení poloviny zobrazovací plochy a funkce pro srovnání Cannyho detektoru s detektorem jiného typu. Po provedení Cannyho algoritmu je výsledek zobrazen na monitoru na konci funkce **`main()`**.

9.2.2 Proces uložení zachyceného snímku do paměti

Inicializace a kompletní spuštění procesu ukládání nových snímků z kamery do paměti je prováděno funkcí **`InitializeVPIF()`**, která je součástí knihovny vpif_lab.lib. Za touto funkcí je skryta modifikovaná demonstrační úloha vpif_lcd_loopback, v níž je vyřešen způsob uložení snímku do paměti a jeho následné zobrazení.

Nejprve jsou alokovány 2 luma buffery a 2 buffery pro chrominanční složky, do nichž budou ukládány příchozí snímky. Poté je provedena inicializace I2C rozhraní pro nastavení video dekodéru TVP5147. Dekodér je v dalším kroku nastaven pro příjem kompozitního typu videa. Následuje funkce **`SetUpVPIFRx()`**, která slouží ke konfiguraci rozhraní VPIF. Je v ní nastaveno VPIF přerušení po zachycení spodního řádku každého snímku, je určen způsob ukládání videa v barevném modelu YCbCr s 8 bity na jeden vzorek a nakonec je konfigurován VPIF kanál 0 pro příjem dat z video dekodéru ve formátu PAL 720x576I. Další funkce **`VPIFDMARequestSizeConfig()`** slouží ke stanovení velikosti jednoho dávkového přenosu mezi VPIF a pamětí s využitím DMA. Poté jsou inicializovány adresy bufferů pro uložení prvního snímku, je nastavena obsluha přerušení a spuštěno přerušení pro VPIF. V obsluze přerušení **`VPIFIsr`**() je po každém uloženém snímku prováděna aktualizace adres pro uložení nového snímku. Ve výsledku máme každý nový snímek uložen do bufferů `*videoTopY` pro luma komponenty a do `*videoTopC` pro chrominanční komponenty.

9.2.3 Převod z modelu YCbCr 4:2:2 do RGB 5:6:5

Pro převod z barevného modelu YCbCr 4:2:2 do černobílého modelu RGB 5:6:5 je po uložení snímku do paměti volána funkce **`cbcr422sp_to_rgb565_gray()`** z knihovny img_algs_vs.lib. Jedná se o upravenou funkci z ukázkové úlohy vpif_lcd_loopback. Navíc je zde prováděn převod z barevného modelu RGB 5:6:5 do černobílého RGB 5:6:5, který potřebujeme pro provedení implementace Cannyho hranového detektoru. Pro bližší

informace o převodu lze nahlédnout do zdrojového kódu funkce v úloze vpif_lcd_loopback, kde je uveden rovněž srozumitelný komentář.

RGB kódování všech stupňů šedi je vždy vyjádřeno rovností všech tří složek (R,G,B), které v modelu 5:6:5 nabývají hodnot 0-31. Tzn, že bílá je zapsána jako (31,31,31), černá je (0,0,0) a středně šedá je (15,15,15). K určení stupně šedi z každého barevného pixelu je nutné vypočítat jeho hodnotu z různě váhovaných složek červené, zelené a modré. To je řešeno dle vztahu:

$$Y = 0.299R + 0.587G + 0.114B$$

Koeficienty, kterými jsou váhovány barevné složky, mají svůj původ ve vnímání barev lidským okem. Nejvyšší důležitost má z toho důvodu zelená barva, nejmenší pak modrá barva. Vypočtenou hodnotu Y přiřadíme všem třem barevným složkám (Y,Y,Y), čímž získáme požadovaný stupeň šedi daného pixelu.

Pokud máme např. pixel s hodnotou (21,5,12), potom:

$$Y = 0.299(21) + 0.587(5) + 0.114(12) = 11$$

To znamená, že odpovídající úroveň šedi je zapsána jako (11,11,11). V binárním 16-bitovém zápisu, se kterým pracujeme v úloze je potom zápis této úrovně šedi:

01011 001011 01011.

9.2.4 Cannyho hranový detektor

Implementace algoritmu pro Cannyho hranový detektor se skládá ze čtyř samostatně volaných funkcí, v nichž jsou postupně prováděny základní kroky algoritmu popsané v kapitole 9.1.4.2.

9.2.4.1 Aplikace Gaussova filtru

Gaussovu filtraci provádí funkce:

```
gaussian(*buff, img_width);
```

Parametr `*buff` je vstupní buffer obsahující černobílý snímek, `img_width` určuje šířku filtrované oblasti od levého kraje.

Je definována maska s Gaussovo rozložením koeficientů:

$$h = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Využíváme principu 2D konvoluce, kdy procházíme pomocí dvou vnořených for cyklů po řádcích všechny pixely, provádíme součin koeficientů masky s příslušnými sousedními pixely a výsledná nová hodnota pixelu je dána sumou těchto součinů.

9.2.4.2 Velikost a směr gradientu pomocí Sobelova operátoru

Po skončení Gaussovy filtrace je volána funkce **`edge_angle`**`(img_width)`, která provádí výpočet velikosti a směru gradientu všech pixelů. Parametr `img_width` představuje i zde šířku funkcí zpracovávané oblasti od levého okraje.

Pro nalezení gradientů použijeme Sobelovu masku ve směru x a ve směru y:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Opět procházíme pomocí dvou vnořených for cyklů celý obraz pixel po pixelu a na každý pixel aplikujeme 2D konvoluci se Sobelovými maskami ve směru x, i ve směru y. Pro každý pixel tím vypočteme velikost gradientu G_X a G_Y. Výsledná velikost a směr gradientu jsou vypočteny dle vztahů uvedených v kapitole 9.1.4.2 b). V závěru funkce je provedena aproximace směru gradientu. Každému pixelu je přiřazena jedna ze čtyř hodnot možných směrů gradientu (0°, 45°, 90°, 135°). V programu je toto řešeno čtyřmi podmínkami s nastaveným rozmezím (např. od 22,5 do 67,5 je přiřazena hodnota 45°). Výsledky funkce jsou uloženy do bufferů `gradient[][]` a `edgeDir[][]`.

9.2.4.3 Prahování s hysterezí

Při implementaci jsou narozdíl od teoretického popisu prohozeny kroky prahování s hysterezí a ztenčení. To znamená, že nejprve jsou dle prahových úrovní nalezeny všechny hrany a až poté jsou vyhledána lokální maxima a odstraněny paralelní hrany. Výhodou je, že po provedení prahování víme, kde se nacházejí všechny skutečné hrany, což využijeme při hledání vedlejších hran.

Pro provedení prahování s hysterezí je volána funkce **`trace`**`(img_width)`. Tato funkce nejprve dle horního definovaného prahu `horniprah` (nastaven např. na hodnotu 36) rozhoduje, zda se jedná, či nejedná o hranu. Pokud je hodnota pixelu větší než `horniprah`, je pixel označen jako hrana a ve switch – case struktuře je podle směru gradientu pixelu volána funkce s parametry danými směrem gradientu:

```
findEdge(rowShift,colShift,row,col,dir,dolniprah,img_width);
```

`rowShift`	posun o stanovený počet řádků od daného pixelu
`colShift`	posun o stanovený počet sloupců od daného pixelu
`row`	řádek pixelu
`col`	sloupec pixelu
`dir`	Směr gradientu pixelu

`dolniprah`	dolní práh pro prahování s hysterezí
`img_width`	šířka funkcí zpracovávané oblasti

Od zkoumaného pixelu, který byl označen jako hrana, prochází funkce **`findEdge`**`()` pixely ve směru daném parametrem `dir` a parametry posunu `rowShift` a `colShift`. Pokud má následující pixel stejný směr gradientu a velikost gradientu vyšší, než je `dolniprah`, potom je označen jako hrana. Ve while cyklu se pokračuje na další pixel, který opět podrobíme testu, zda náleží zkoumané hraně. While cyklus skončí, pokud má pixel jiný směr gradientu, když má velikost gradientu menší než `dolniprah`, nebo pokud se nachází na okraji zobrazovací plochy.

Výsledkem této funkce je nalezení všech hran v obraze, které splnily podmínky dané prahováním s hysterezí. V pomocném bufferu tak máme všechny pixely představující hrany převedené na bílou barvu, zatímco všechny ostatní pixely jsou nastaveny na černou barvu.

9.2.4.4 Nalezení lokálních maxim - ztenčení

Posledním krokem algoritmu pro Cannyho hranový detektor je potlačení detekovaných paralelních hran, aby zůstaly jen ty „nejsilnější“. Proto voláme funkci **`supress`**`(img_width)`. V té opět dvěma vnořenými for cykly procházíme všechny pixely, přičemž pokud byl v předchozím kroku pixel označen jako hrana (jeho hodnota odpovídá bílé barvě), je pro tento pixel ze switch – case struktury volána dle směru gradientu funkce:

```
suppressNonMax(rowShift,colShift,row,col,dir,dolniprah,img_width);
```

Funkce **`suppressNonMax`**`()` obsahuje stejné parametry jako funkce **`findEdge`**`()` z předcházejícího kroku. Této funkci se podobá svým provedením. Hlavní změnou jsou odlišně nastavené parametry `rowShift` a `colShift`, které určují posun na další zkoumaný pixel. Nyní neprocházíme pixely ve směru hrany, ale procházíme pixely ve směru kolmém na hranu a to pro oba směry od hrany. Ve funkci se proto nachází dva while cykly, každý pro jeden směr od hrany. Pro pokračování cyklu musí být splněna podmínka, že další pixel má stejný směr gradientu a že je to pixel v předešlém kroku označený jako hrana. Pokud tyto podmínky pixel splňuje, je do pomocného bufferu `nonMax` uložena jeho pozice a velikost gradientu. Navíc je inkrementována proměnná, která určuje počet pixelů v šířce hrany. Po prozkoumání všech pixelů v obou směrech kolmých na hranu v místě zkoumaného pixelu je naplněn buffer `nonMax` velikostmi gradientů všech těchto pixelů. Potom je z pixelů vybrán ten s nejvyšší velikostí gradientu, který představuje lokální maximum zkoumaného místa hrany. V úplném závěru se všechny pixely kromě těch, které jsou lokálními maximy, nastaví na černou barvu. Výsledkem funkce jsou v ideálním případě hrany o šířce jednoho pixelu uložené v bufferu `pom_buff`. Tento buffer již lze poslat na výstup k zobrazení na monitoru a ke sledování úspěšnosti implementace Cannyho hranového detektoru.

9.2.5 Doplňkové funkce

Společně s implementací algoritmu pro Cannyho detektor je i tato úloha doplněna o další doplňkové funkce. Samostatně lze vyzkoušet jiný typ detektoru, přičemž implementovány jsou detektory založené na první derivaci. Tyto detektory lze poté s využitím funkce pro rozdělení obrazovky a funkce pro zrcadlení přímo porovnávat s Cannyho hranovým detektorem.

9.2.5.1 Zrcadlení

Funkce **`zrcadleni()`** je jednoduchá funkce, kterou lze povolit nastavením dvoustavové proměnné `zrcadleni_on` na true. Předpokladem této funkce je, že při dalších operacích pracujeme pouze s polovinou zobrazovací plochy o velikosti `img_width/2 * img_height`. Cílem funkce je, abychom jednu polovinu zobrazovací plochy překlopili do druhé poloviny a získali tak dvě poloviny s totožnou obrazovou informací. Každou polovinu můžeme v dalších operacích zpracovávat jiným způsobem a ve výsledku tak můžeme srovnávat např. dvě různé metody detekce hran.

Provedení funkce spočívá v procházení všech pixelů z levé poloviny zobrazovací plochy, přičemž každý pixel je přenesen do pixelu na stejném řádku o `img_width/2` sloupců dále.

9.2.5.2 Sobelův hranový detektor

Z jednoduchých hranových detektorů založených na první derivaci je ve funkci **`sobel()`** implementován Sobelův detektor. Sobelův operátor je použit i v algoritmu pro Cannyho hranový detektor. Implementace samotného Sobelova detektoru je v principu stejná. Použijeme dvě masky - ve směru x a ve směru y. 2D konvolucí určíme v jediném for cyklu velikost gradientu všech pixelů a jako hrany označíme pouze ty pixely, které mají velikost gradientu vyšší, než je definovaný práh. Výsledek poté můžeme rovnou zobrazit na monitoru.

9.2.5.3 Porovnání

Pro využití funkce zrcadlení a pro možnost porovnání Cannyho detektoru s jiným detektorem hran je navržena funkce **`sobel_compare()`**, kterou lze povolit, nebo zakázat pomocí dvoustavové proměnné `sobel_comp`. Jedná se o totožné provedení Sobelova detektoru hran jako ve funkci **`sobel()`** doplněném o funkcionalitu zobrazení výsledku detekce v pravé polovině obrazovky. Do této funkce můžeme dosadit i jiný hranový detektor, či jiný způsob zpracování obrazu. Také můžeme nechat pravou polovinu obrazovky s původním nezpracovaným snímkem pro vyhodnocení úspěšnosti detekce hran.

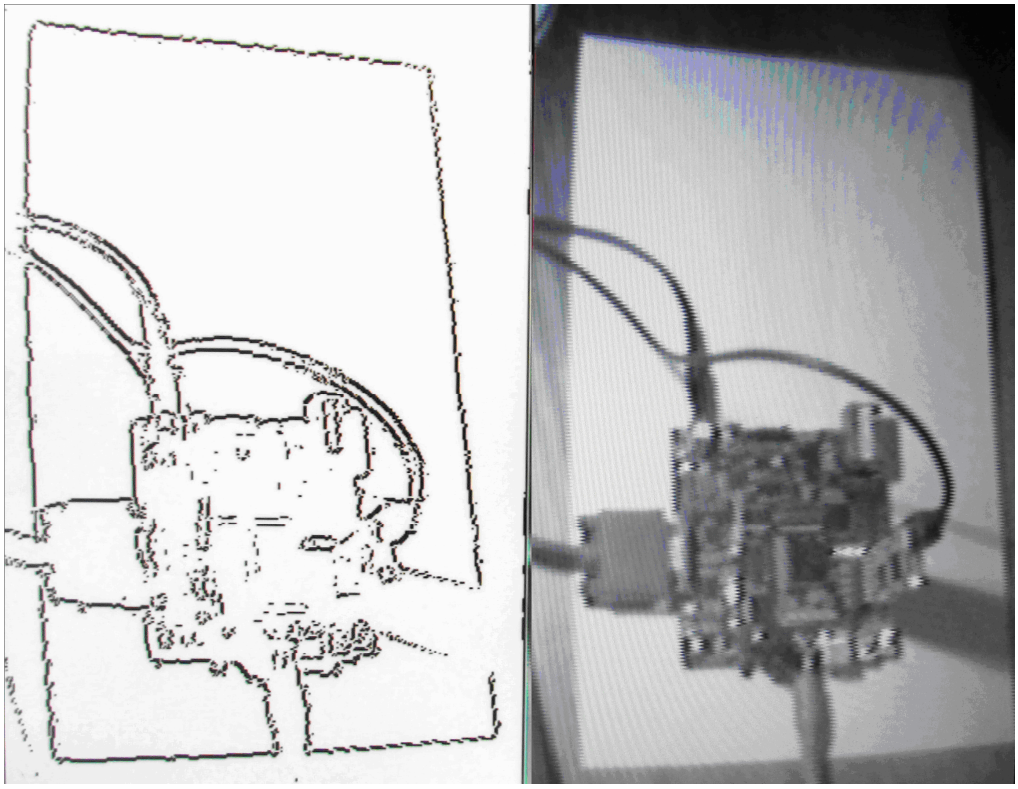
9.2.6 Proces zobrazení zpracovaného snímku na monitoru

Posledním úkolem je zobrazení výsledků detekce hran na monitoru. Zpracovaný pomocný buffer `pom_buff` proto přenášíme do bufferu `out_buff`, což znamená do jednoho z výstupních video bufferů `videoTopRgb1`, nebo `videoTopRgb2`. Tyto buffery jsou střídavě ve funkci **`main()`** posílány k zobrazení. Proces inicializace subsystémů potřebných k zobrazení snímku a jejich spuštění je obsažen ve funkci **`Initialize()`**. Provedení

zobrazení je v principu stejné jako proces zobrazení u úlohy spektrální analyzátor v kapitole 7.2.5.

9.3 Zhodnocení

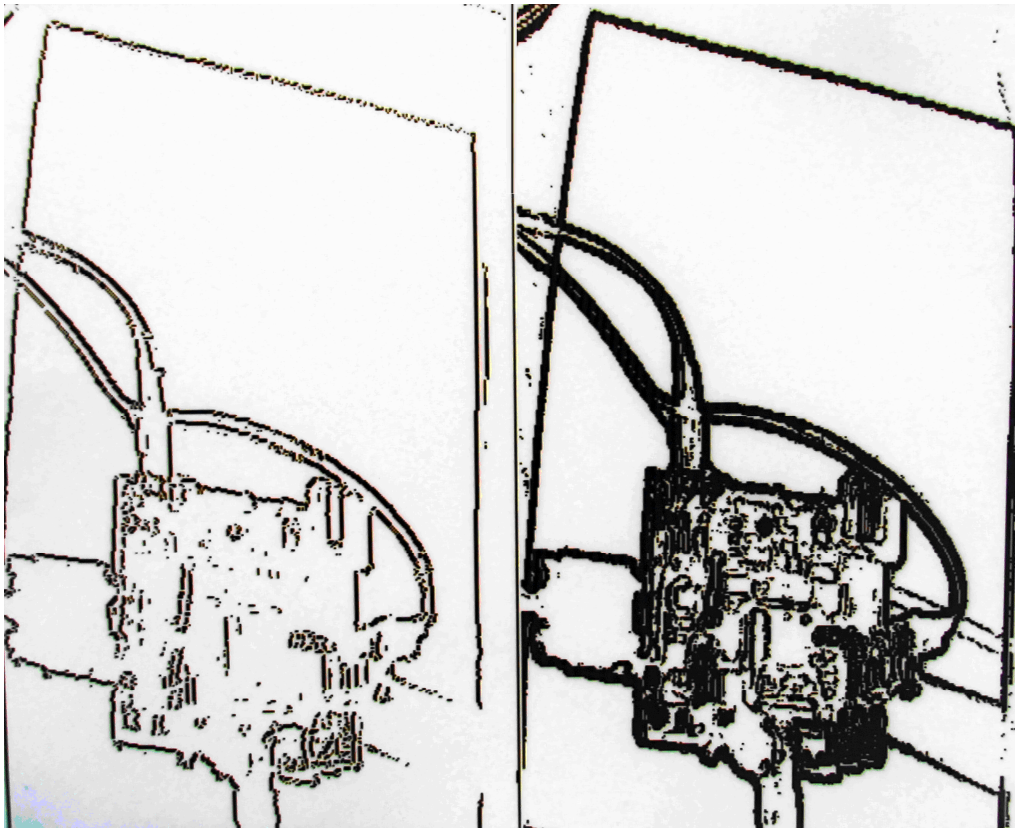
V úloze Cannyho hranový detektor byl implementován úplný algoritmus pro detekci hran Cannyho metodou skládající se ze čtyř definovaných kroků. Nejprve je provedena Gaussova filtrace snímku pro potlačení šumu, v dalším kroku je spočtena velikost a směr gradientů všech pixelů, poté je provedeno prahování s hysterezí pro nalezení všech dostatečně “silných” hran a nakonec jsou nalezeny lokální maxima gradientů a potlačeny nežádoucí paralelní hrany. K zachycení, uložení a zobrazení snímků je využita knihovna používaná při výuce zpracování obrazu na signálových procesorech od pana inženýra V. Svobody. Je vhodná zejména z důvodu oddělení inicializační a konfigurační části kódu od procesních funkcí pro zpracování obrazu. Lze se proto věnovat pouze samotným obrazovým operacím, aniž bychom museli konfigurovat jednotlivé subsystémy na LCDK modulu.



Obrázek 31 - Ukázka výsledku algoritmu Cannyho hranového detektoru

Kromě hlavního algoritmu pro Cannyho hranový detektor obsahuje úloha podružné funkce pro možnost porovnání dvou různých detektorů a pro viditelnější vyhodnocení úspěšnosti detekce hran Cannyho metodou. Obraz si tak lze rozdělit na dvě poloviny, přičemž je dále možné si zvolit, že na pravé polovině se bude zrcadlit obraz z levé poloviny. Úloha v základu umožňuje porovnání Cannyho detektoru s původním zachyceným obrazem, nebo se Sobelovým detektorem hran. Do porovnávací funkce přitom není složité dosadit jiný druh detektoru. Ve výsledku můžeme například mít na levé

polovině obrazovky provedenou detekci hran Cannyho metodou a na pravé polovině obrazovky na stejné polovině původního snímku můžeme mít detekci hran pomocí Sobelova detektoru. Všechny funkce včetně té pro Cannyho detektor lze jednoduše vypínat, či zapínat. Úloha je proto jednoduše modifikovatelná a přizpůsobená pro vložení dalších algoritmů a funkcí.



Obrázek 32 - Porovnání výsledku Cannyho a Sobelova hranového detektoru

Cannyho algoritmus je ovšem mnohem výpočetně náročnější než např. Sobelův detektor. To je velmi dobře znát i v této úloze. Zatímco při implementaci Sobelova detektoru je obraz plynulý a procesor dostatečně stíhá zpracovávat nově příchozí snímky a zobrazovat je na monitoru, při spuštění Cannyho algoritmu je patrné výrazné zhoršení, obnova snímků je pomalejší a procesor nestíhá zpracovávat všechny příchozí snímky. Nejlepším způsobem jak toto zlepšit vedle optimalizace zdrojového kódu je zmenšit oblast pro aplikaci Cannyho algoritmu. Již při provádění algoritmu pouze nad polovinou obrazu je znatelné výrazné zrychlení obnovy snímků. Oblast lze zvolit libovolně velkou a pokud budeme chtít plynulou obnovu snímků v reálném čase při provádění Cannyho algoritmu, budeme muset zmenšit oblast ještě méně než na polovinu.

10 Závěr

Cílem diplomové práce bylo vytvoření 3 výukových úloh pro digitální signálový procesor na vývojovém modulu C6748 LCDK. Při psaní této práce byl kladen důraz zejména na postupné seznámení se se všemi aspekty, které jsou nutné k pochopení dané problematiky. Z toho důvodu byl v první části práce proveden popis všech subsystémů a periferií na LCDK modulu, které jsme potřebovali při implementaci vybraných úloh. Záměrem popisu všech částí LCDK modulu bylo rychlé seznámení se s jejich činností. Pro plné pochopení však doporučujeme nahlédnout do příslušných obsáhlých technických a uživatelských manuálů uvedených na konci této práce v seznamu použité literatury. Vzhledem k použití modulu převodníku ADS1606EVM v úloze AM přijímač byla tomuto převodníku věnována samostatná kapitola, která obsahuje všechny jeho důležité vlastnosti a informace potřebné k pochopení jeho činnosti a způsobu použití. Celou práci lze mimojiné rozdělit na HW a na SW část. Proto hned po popisu použitých modulů následuje kapitola, která obsahuje konfiguraci obou modulů a schémata zapojení pro jednotlivé úlohy.

Zde se práce láme a začíná SW část. První kapitola této části má za úkol provést seznámení s vývojovým prostředím Code Composer Studio, jenž bylo použito při programování všech úloh. Důraz je opět kladen zejména na důležitá nastavení CCS pro jednotlivé úlohy, je zde uveden postup vytvoření, či importu projektu, postup připojení k LCDK modulu a nahrání projektu do paměti procesoru. V kapitole byl dále uveden seznam potřebných softwarových balíčků, výpis těch nejdůležitějších knihoven a pro zprovoznění úloh důležitý seznam použitých knihoven pro jednotlivé úlohy.

Další hledisko, jak lze práci rozdělit, je na část popisující nástroje potřebné ke zprovoznění jednotlivých úloh a na část popisující provedení úloh samotných, přičemž popis každé úlohy je vždy rozdělen na teoretický rozbor a na implementaci úlohy.

První zpracovanou úlohou byl spektrální analyzátor. Tato úloha nabízí možnost zobrazení dvou samostatných grafů na monitoru. V horním grafu je průběh vstupního signálu ze zvukové karty, ve spodním grafu je vypočítané frekvenční spektrum vstupního signálu. Dále si lze zobrazit popisky os, měřítko a hodnoty aktuálně zvolených parametrů vzorkovací frekvence a délky transformace. Vedle hlavní funkce zobrazení frekvenčního spektra byla v úloze vypracována řada doplňkových funkcí, které je možné před spuštěním programu zapínat, či vypínat. Patří sem zejména možnost váhování vstupního signálu oknem, dále synchronizace časové základny pro periodické signály, automatické zmenšení měřítka na ose y při přesáhnutí maximálně zobrazitelné amplitudy, možnost sledování i velmi malých amplitud signálu (zejména šumu), či možnost výřezu daného místa ve frekvenčním spektru.

Úloha může být snadno rozšířena o další operace, v každém cyklu jsou naplněny pomocné buffery např. vstupním signálem, váhovaným signálem, či frekvenčním spektrem signálu. Lze tedy jednoduše zařadit do programu další funkce a procedury, které mají okamžitý přístup k požadovaným signálům, lze je jednoduše zapínat, či vypínat a jejich výsledky mohou být pomocí implementovaných zobrazovacích funkcí snadno převáděny do formy pro zobrazení.

Druhou úlohou byl AM přijímač. V této úloze bylo cílem vytvoření kompletního AM přijímače, kde celý proces AM demodulace probíhá na DSP a kde k digitalizaci přijatého modulovaného signálu slouží modul převodníku ADS1606EVM připojený k LCDK modulu prostřednictvím rozhraní EMIFA. Nicméně úloha AM přijímač nebyla k datu odevzdání této práce prakticky realizována. Dokončeno bylo spojení modulu převodníku ADS1606EVM a LCDK modulu s využitím rozhraní EMIFA. Byl vyřešen způsob inicializace a konfigurace EMIFA rozhraní pro asynchronní režim s 16-bitovým paralelním rozhraním. Dále byl nakonfigurován EDMA přenos dat z rozhraní EMIFA do paměti závislý na sestupné hraně data ready signálu z modulu převodníku přiváděného na GPIO vstup LCDK modulu.

Pro nevyřešené části úlohy byl v rámci rozboru úlohy v této práci proveden principiální návrh. Zejména se jedná o kapitolu návrhu vstupního přijímacího obvodu a kapitolu návrhu procedury pro provedení digitální AM demodulace.

Dalším krokem v řešení úlohy bude využití FIFO režimu na výstupu modulu převodníku a s tím související přenastavení režimu čtení z rozhraní EMIFA a úprava parametrů EDMA přenosu. Dále bude provedena realizace vstupního přijímacího obvodu a bude odladěn navržený algoritmus pro digitální AM demodulaci. Úspěšným výsledkem celé úlohy má být poslech naladěné rozhlasové stanice na audio výstupu LCDK modulu. Na výsledné podobě úlohy budeme po odevzdání této práce i nadále pracovat, aby byla dokončena v rámci následujícího semestru a mohla být použita při výuce.

Třetí a poslední úlohou byl Cannyho hranový detektor. V této úloze byl implementován úplný algoritmus pro detekci hran Cannyho metodou skládající se ze čtyř definovaných kroků. Nejprve je provedena Gaussova filtrace snímku pro potlačení šumu, v dalším kroku je spočtena velikost a směr gradientů všech pixelů, poté je provedeno prahování s hysterezí pro nalezení všech dostatečně “silných” hran a nakonec jsou nalezeny lokální maxima gradientů a potlačeny nežádoucí paralelní hrany.

Kromě hlavního algoritmu pro Cannyho hranový detektor obsahuje úloha podružné funkce pro možnost porovnání dvou různých detektorů. Obraz si tak lze rozdělit na dvě poloviny, přičemž je dále možné si zvolit, že na pravé polovině se bude zrcadlit obraz z první poloviny. Úloha v základu umožňuje porovnání Cannyho detektoru s původním zachyceným obrazem, nebo se Sobelovým detektorem hran. Do porovnávací funkce přitom není složité dosadit jiný druh detektoru. Všechny funkce včetně té pro Cannyho detektor lze jednoduše vypínat, či zapínat. Úloha je proto jednoduše modifikovatelná a přizpůsobená pro vložení dalších algoritmů a funkcí.

11 Seznam použité literatury

[1] TEXAS INSTRUMENTS, Inc. *TMS320C6748 DSP Technical Reference Manual (SPRUH79A)* [online]. Dallas, © 2011. Dostupné z: http://www.ti.com/lit/ug/spruh79a/spruh79a.pdf

[2] L138/C6748 Development Kit (LCDK). In: *Texas Instruments Wiki* [online]. 23. 5. 2012. Dostupné z: http://processors.wiki.ti.com/index.php/L138/C6748_Development_Kit_(LCDK)

[3] XDS100. In: *Texas Instruments Wiki* [online]. 8. 8. 2008. Dostupné z: http://processors.wiki.ti.com/index.php/XDS100

[4] BIOS C6SDK 2.0 Getting Started Guide. In: *Texas Instruments Wiki* [online]. 20. 8. 2012. Dostupné z: http://processors.wiki.ti.com/index.php/BIOS_C6SDK_2.0_Getting_Started_Guide

[5] StarterWare Getting Started 01.20.XX.XX. In: *Texas Instruments Wiki* [online]. 27. 10. 2011. Dostupné z: http://processors.wiki.ti.com/index.php/StarterWare_Getting_Started_01.20.XX.XX

[6] StarterWare 01.20.01.01 User Guide. In: *Texas Instruments Wiki* [online]. 27. 10. 2011. Dostupné z: http://processors.wiki.ti.com/index.php/StarterWare_01.20.01.01_User_Guide

[7] Category:Code Composer Studio v5. In: *Texas Instruments Wiki* [online]. 20. 8. 2010. Dostupné z: http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v5

[8] TEXAS INSTRUMENTS, Inc. *TMS320C6748™ Fixed- and Floating-Point DSP (SPRS590F)* [online]. Dallas, © 2008, revised March 2014. Dostupné z: http://www.ti.com/lit/ds/symlink/tms320c6748.pdf

[9] TEXAS INSTRUMENTS, Inc. *TMS320C674x/OMAP-L1x Processor Peripherals Overview Reference Guide (SPRUFK9F)* [online]. Dallas, © 2011. Dostupné z: http://www.ti.com/lit/ug/sprufk9f/sprufk9f.pdf

[10] TEXAS INSTRUMENTS, Inc. *TMS320C674x/OMAP-L1x Processor General-Purpose Input/Output (GPIO) User's Guide (SPRUFL8B)* [online]. Dallas, © 2010. Dostupné z: http://www.ti.com.cn/cn/lit/ug/sprufl8b/sprufl8b.pdf

[11] TEXAS INSTRUMENTS, Inc. *TMS320C674x/OMAP-L1x Processor External Memory Interface A (EMIFA) User's Guide (SPRUFL6F)* [online]. Dallas, © 2010. Dostupné z: http://www.ti.com/lit/ug/sprufl6f/sprufl6f.pdf

[12] TEXAS INSTRUMENTS, Inc. *TMS320C6748/46/42 and OMAP-L138 Processor Enhanced Direct Memory Access (EDMA3) Controller User's Guide (SPRUGP9B)* [online]. Dallas, © 2010. Dostupné z: http://www.ti.com/lit/ug/sprugp9b/sprugp9b.pdf

[13] TEXAS INSTRUMENTS, Inc. *TMS320C674x/OMAP-L1x Processor Video Port Interface (VPIF) User's Guide (SPRUGJ9B)* [online]. Dallas, March 2011. Dostupné z: http://www.ti.com/lit/ug/sprugj9b/sprugj9b.pdf

[14] TEXAS INSTRUMENTS, Inc. *TMS320C674x DSP Cache User's Guide (SPRUG82A)* [online]. Dallas, © 2009. Dostupné z: http://www.ti.com/lit/ug/sprug82a/sprug82a.pdf

[15] TEXAS INSTRUMENTS, Inc. *TMS320C6748 DSP System Reference Guide (SPRUGJ7D)* [online]. Dallas, © 2010. Dostupné z: http://d1.amobbs.com/bbs_upload782111/files_38/ourdev_628170WESIFV.pdf

[16] TEXAS INSTRUMENTS, Inc. *OMAP-L138 & C6748 LCDK v.A5 Schematics* [online]. March 2012. Dostupné z: http://processors.wiki.ti.com/images/6/69/OMAP-L138_C6748_LC_Dev_Kit_Ver_A5_.zip

[17] TEXAS INSTRUMENTS, Inc. *TLV320AIC3106 Low-Power Stereo Audio CODEC for Portable Audio/Telephony (SLAS509F)* [online]. 2006, revised December 2014. Dostupné z: http://www.ti.com/lit/ds/symlink/tlv320aic3106.pdf

[18] TEXAS INSTRUMENTS, Inc. *THS8135 10-bit 240-MSPS Video DAC with Tri-level Sync and Video (ITU-R.BT601)-Compliant Full-scale Range (SLAS343B)* [online]. 2001, revised April 2013. Dostupné z: http://www.ti.com/lit/ds/slas343b/slas343b.pdf

[19] TEXAS INSTRUMENTS, Inc. *TVP5147PFP Data Manual (SLES099C)* [online]. 2007. Dostupné z: http://www.ti.com/lit/ds/symlink/tvp5147.pdf

[20] TEXAS INSTRUMENTS, Inc. *ADS1605 and ADS1606 EVM User's Guide (SLAU122A)* [online]. Dallas, © 2004. Dostupné z: http://www.ti.com/lit/ug/slau122a/slau122a.pdf

[21] TEXAS INSTRUMENTS, Inc. *ADS1606, 16-Bit, 5MSPS Analog-to-Digital Converter (SBAS274H)* [online]. Dallas, © 2003, revised May 2007. Dostupné z: http://www.ti.com/lit/ds/symlink/ads1606.pdf

[22] CASTILLE, Kyle. TEXAS INSTRUMENTS, Inc. *TMS320C6000 EMIF to External FIFO Interface (SPRA543)* [online]. May 1999. Dostupné z: http://www.ti.com/lit/an/spra543/spra543.pdf

[23] DHARIA, Anuj a Rosham GUMMATTIRA. TEXAS INSTRUMENTS, Inc. *Signal Processing Examples Using the TMS320C67x Digital Signal Processing Library (DSPLIB) (SPRA947A)* [online]. June 2009. Dostupné z: http://www.ti.com/lit/an/spra947a/spra947a.pdf

[24] TEXAS INSTRUMENTS, Inc. *Code Composer Studio ™ Integrated Development Environment (IDE) v5 Quick Start Guide (SPRM382A)* [online]. © 2011. Dostupné z: http://processors.wiki.ti.com/images/5/52/CCStudio_v5_QSG_9-11.pdf

[25] GELB, Ben. *A Software Defined Radio Receiver* [online]. June 8, 2004. Dostupné z: http://www.tjhsst.edu/~rlatimer/techlab/Gelbpaper04.pdf

[26] BECKER, Dirk. UNIVERSITY OF EAST LONDON. *Lab-Report Digital Signal Processing: Amplitude Modulation using DSP methods* [online]. 28. 1. 1999. Dostupné z: http://www.dbecker.de/sites/default/files/amdigital.pdf

[27] GREEN, Bill. *Canny Edge Detection Tutorial* [online]. 2002. Dostupné z: http://dasl.mem.drexel.edu/alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html

[28] KUNTZ, Noah. *Canny Tutorial* [online]. 2006. Dostupné z: *http://www.pages.drexel.edu/~nk752/Research/cannyTut2.html*

[29] BRUCHANOV, Martin. *Základy zpracování obrazů* [online]. 2009. Dostupné z: http://bruxy.regnet.cz/web/programming/CZ/zaklady-zpracovani-obrazu/

[30] QURESHI, Shehrzad. *Embedded Image Processing on the TMS320C6000(TM) DSP: Examples in Code Composer Studio(TM) and MATLAB*. Springer, 2005. 1st edition. ISBN 978-0-387-25280-3.

[31] REAY, Donald. HERIOT-WATT UNIVERSITY. *Digital Signal Processing and Applications with the OMAP-L138 eXperimenter*. Hoboken, New Jersey, vydavatelství John Wiley & Sons, Inc., © 2012. ISBN 978-0-470-93686-3.

[32] DOBEŠ, Michal. *Zpracování obrazu a algoritmy v C#*. Praha, vydavatelství BEN, 2008. ISBN 978-80-7300-233-6.

12 Seznam obrázků

13 Seznam tabulek

14 Seznam zkratek

ADC / DAC – Analog to Digital Converter / Digital to Analog Converter
AINN – Analog Input Negative
AINP – Analog Input Positive
ALU – Arithmetic Logic Unit
AM – Amplitude Modulation
API – Application Programming Interface
CCS – Code Composer Studio
CEnCFG – CEn Configuration Register
CE (CS) – Chip Enable (Chip Select)
CLK – Clock
CPU – Central Processing Unit
DSP – Digital Signal Processor
DFT - Discrete Fourier Transform
EDMA3 – Enhanced Direct Memory Access
EDMA3CC – EDMA3 Channel Controller
EDMA3TC – EDMA3 Transfer Controller
EMIFA – External Memory Interface A
FIFO – First-In, First-Out
FFT – Fast Fourier Transform
GPIO – General Purpose Input/Output
I2C - Inter-Integrated Circuit
I2S – Integrated Interchip Sound
IDMA – Internal direct memory access
INTC – Interrupt Controller
JTAG – Joint Test Action Group
LCDC – LCD Controller
LCDK – L138/C6748 Developement Kit
McASP – Multichannel Audio Serial Port
OTR – Out of Range
PaRAM – Parameter RAM
PSC – Power Sleep Controller
RD – Read
RGB – Red, Green, Blue Color System
SMA – SubMiniature version A Connector
SOC – System on a Chip
TDM – Time-Division Multiplexing
UART – Universal Asynchronous Receiver/Transmitter
VGA – Video Graphics Array
VPIF – Video Port Interface
VLIW – Very Long Instruction Word
YCrCb – Luminance, Red Chrominance, Blue Chrominance Color System