

ANOTACE

Práce se zabývá vývojem zařízení pro obousměrný bezdrátový přenos digitalizované hlasové informace. Zařízení sestává z analogového vstupu vhodného pro připojení mikrofону, analogového výstupu pro sluchátka, nebo malý reproduktor, řídicího mikroprocesoru a RF modulu pro bezdrátovou komunikaci. Komprese audio signálu je řešena softwarově.

KLÍČOVÁ SLOVA

Digitální komprese audio signálu; bezdrátová komunikace

ANNOTATION

This document describes device for full-duplex wireless voice communication. Basic parts of the device are analogue input for microphone, analogue output for headphones or speaker, microprocessor and RF module handling wireless communication. Compression of audio signal is managed by microprocessor.

KEYWORDS

Digital compression of audio signal; wireless communication

STREIT, J. *Digitální audiořetězec s bezdrátovou komunikací*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav automatizace a měřicí techniky, 2014. 53 s., 23 s. příloh. Bakalářská práce. Vedoucí práce: doc. Ing. Zdeněk Bradáč, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma Digitální audiořetězec s bezdrátovou komunikací jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne

.....

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce doc. Ing. Zdeňku Bradáčovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne

.....

OBSAH

1	Úvod.....	8
2	Základní koncept.....	9
2.1	Analogový vstup a výstup.....	9
2.2	Komprese audio signálu.....	9
2.3	Řídicí mikroprocesor	10
2.4	RF modul	10
3	Bootloader	11
3.1	První spuštění bootloaderu.....	11
3.2	SAM-BA Monitor	11
3.2.1	Xmodem protokol.....	12
3.3	Způsob programování	13
3.4	Spuštění bootloaderu z uživatelské aplikace	16
3.5	PC klient	16
4	Audio kodek	17
4.1	Komunikace s kodekem.....	17
4.1.1	Nastavení kodeku	17
4.1.2	Audio data	19
4.2	Základní nastavení kodeku	20
4.2.1	Hodiny	21
4.2.2	Vzorkovací frekvence.....	21
4.2.3	Komunikace.....	21
4.2.5	Výstupní signál.....	23
4.2.6	Nastavení hlasitosti.....	24
5	Speex	25
5.1	Přizpůsobení kodeku.....	25
5.1.1	Optimalizace.....	25
5.1.2	Nastavení kompilace	25
5.1.3	Systémové funkce.....	26
5.2	API.....	27
5.2.1	SpeexBits.....	27
5.2.2	Enkodér.....	28
5.2.3	Dekodér	29
5.3	Příklad použití kodeku	31
5.3.1	Enkodér.....	31
5.3.2	Dekodér	32
6	Komunikační protokol	33
6.1	Řízení komunikace	34
6.2	Identifikace zařízení.....	35

6.3	Skupiny zařízení	36
7	RF modul.....	37
7.1	Komunikace s modulem	37
7.2	Nastavení modulu	38
7.2.1	RF přenos.....	39
7.2.2	Správce paketů.....	41
7.3	Obsluha modulu	43
8	Závěr	45
	Literatura	46
	Seznam obrázků	49
	Seznam tabulek	50
	Seznam zkratk	51
	Seznam příloh.....	53

1 ÚVOD

Tato práce se zabývá návrhem malého mobilního zařízení, umožňujícího uživatelům v reálném čase vzájemně komunikovat pomocí hlasu.

Hlavní výhodou navrhovaného zařízení oproti komerčně dostupným technologiím je možnost obousměrné komunikace bez nutnosti přepínání směru. Toto většina dostupných vysílaček neumožňuje. Umí to sice mobilní telefony, jejich hlavní nevýhodou ovšem je, že jejich provoz je zpoplatněn.

Systém pracuje v ISM¹ pásmu na frekvenci 866 MHz. Z důvodů efektivního využití frekvenčního spektra je nutná minimální šířka pásma. Z toho přímo vyplývá požadavek na minimální datový tok. Audio signál je tedy nutno před přenosem komprimovat.

Práce navazuje na [1], kde jsou podrobně rozvedeny důvody výběru konkrétního řešení a návrh hardwaru. Hlavním cílem této práce je doplnění programového vybavení k již navrženému hardwaru.

¹ *Industrial, Scientific and Medical* – volná pásma, spadající pod všeobecné licence Českého telekomunikačního úřadu (ČTU); není zde vyloučeno rušení [30]

2 ZÁKLADNÍ KONCEPT

V této kapitole jsou vyjmenovány základní části celého systému. Ke každé je ve stručnosti uvedeno, proč byl zvolen právě tento postup. Podrobnější postup při návrhu a porovnání s alternativními řešeními je v [1].

2.1 Analogový vstup a výstup

Většina dnes používaných mikroprocesorů disponuje analogově-digitálním převodníkem (AD převodníkem). Tento by se dal přímo použít k digitalizaci vstupního signálu. Nicméně je zdě několik problémů. Prvním a největším je úroveň vstupního signálu. Rozsah a rozlišení běžných AD převodníků nedovolují připojit mikrofon přímo na vstup převodníku. Signál z mikrofonu je potřeba napřed zesílit. Navíc protože dynamický rozsah lidského sluchu je asi 130 dB [2], mělo by toto zesílení být nastavitelné.

Poněkud omezenější výběr je v oblasti digitálně-analogových převodníků (DA převodníků). DA převodník se ovšem pro daný účel dá nahradit pomocí PWM² výstupu, na kterém je připojen RC článek, fungující jako dolní propust. Opět je zde ale stejný problém s hlasitostí. Navíc, a to hlavně, by bylo nutné softwarově převzorkovat výstupní signál, aby vzorkovací frekvence byla mimo slyšitelné pásmo.

Všechny tyto problémy řeší použitý audio kodek. Pro své značně široké možnosti byl vybrán kodek TLV320AIC3101 od firmy Texas Instruments.

2.2 Komprese audio signálu

Název práce je Digitální audiořetězec s bezdrátovou komunikací. Z pojmu „řetězec“ vyplývá, že by se komunikace mělo účastnit více zařízení. Dostupné hardwarové obvody pro kompresi audiosignálu jsou navrženy pouze pro zpracování jednoho stereofonního signálu. Z toho plyne, že by vzájemně mohly komunikovat pouze 2 zařízení. Pro komunikaci většího počtu je tedy potřeba provádět kompresi softwarově. V tom případě je počet účastníků omezen pouze dostupným výpočetním výkonem řídicího mikroprocesoru.

Z dostupných open source kodeků³, vhodných pro hlas, byl testován Opus a Speex. Vybrán byl Speex, díky menší náročnosti na operační paměť systému.

² PWM – Pulse Weight Modulation – pulzně šířková modulace

³ Z hlediska hardwaru je audio kodek součástka, starající se o všechny nezbytné náležitosti převodu mezi analogovým a digitálním audio signálem. Nemá ovšem v naprosté většině případů nic společného s kompresí signálu.

V kontextu softwaru naopak audio kodek znamená algoritmus řešící kompresi a dekompresi audio signálu.

2.3 Řídicí mikroprocesor

Kvůli značné výpočetní náročnosti komprese audio signálu nelze použít jednoduchý osmibitový mikrokontrolér. Byl vybrán mikroprocesor od firmy Atmel AT91SAM3U2CA-AU. Jedná se o 32bitový procesor postavený na jádře ARM® Cortex®-M3 revision 2.0 [3]. Hlavní důvody byly dostatečný výpočetní výkon, obsahuje potřebná rozhraní pro připojení audio kodeku a RF modulu a integrovaná paměť ROM, obsahující bootloader.

2.4 RF modul

Hlavním kritériem pro výběr RF modulu bylo frekvenční pásmo, ve kterém pracuje a předpokládaný dosah. Vzhledem k zadání (sub-gigahertzové bezdrátové komunikační rozhraní) a k české legislativě, přichází v úvahu pouze pásmo 433 MHz a 866 MHz. Z dostupných modulů byl vybrán RFM22B-868-S1, pracující v pásmu 866 MHz.

Mezi jeho hlavní přednosti patří [4]:

- Vysílací výkon +20 dBm (0,5 W – maximální povolený vysílací výkon v daném pásmu)
- Citlivost přijímače -121 dBm
- Přenosová rychlost (0,123 ÷ 256) kbps
- Podpora paketů
- Jednoduše nastavitelná komunikační frekvence pomocí nastavení kanálu

3 BOOTLOADER

Jedním z hlavních důvodů pro výběr procesoru od firmy Atmel byl z výroby přednahráný bootloader. Díky němu nebylo nutné kupovat programátor. Stačí procesor připojit k počítači prostřednictvím integrovaného USB portu, nebo UARTu.

V procesoru se nachází paměť typu ROM, ve které je z výroby nahráný Atmel SAM-BA® Monitor. SAM je rodina 32bitových mikroprocesorů firmy Atmel, postavených na jádře ARM®. BA je zkratka z Boot Assistant [5].

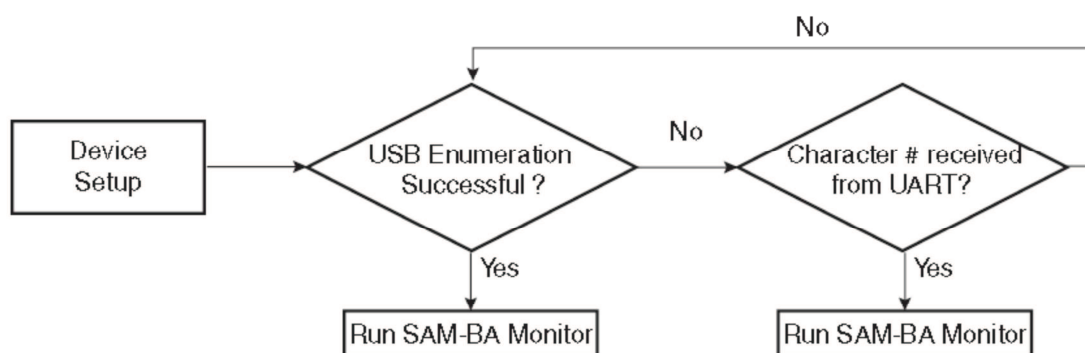
3.1 První spuštění bootloaderu

Procesor obsahuje několik GPNVM bitů (General Purpose Non-Volatile Memory). Jedná se o speciální bity, které si uchovávají svou hodnotu i při odpojení napájení procesoru (nebo resetu). Jejich nastavení ovlivňuje, z které paměti procesor po resetu načte program. Bit GPNVM1 určuje, jestli procesor spustí uživatelský program z interní Flash paměti (bit nastaven), nebo SAM-BA Monitor z interní ROM paměti (bit vynulován).

Kromě možnosti měnit hodnotu bitu softwarově se tento dá vynulovat hardwarově, přivedením log. 1 na pin Erase. Tím se vymaže interní Flash paměť a všechny GPNVM bity. Po resetu procesor spustí z interní ROM paměti SAM-BA Monitor.

3.2 SAM-BA Monitor

Po svém spuštění SAM-BA monitor nastaví hodiny procesoru, inicializuje komunikační periférie (UART, případně USB) a čeká na příkaz od nadřazeného systému (počítače).



Obrázek 1: Vývojový diagram inicializace SAM-BA monitoru [3]

Nastavení hodin znamená výběr vhodného zdroje taktovací frekvence pro procesor. Možnosti jsou externí krystal, externí hodinový signál na pinu XIN, nebo interní 12 MHz RC oscilátor. Pokud jsou hodiny přivedeny zvenku (včetně externího krystalu), je pomocí interního 32 kHz RC oscilátoru změřena jejich frekvence. Jestliže se tato nerovná 12 MHz, nebo je použit interní 12 MHz RC oscilátor, není povoleno USB,

protože hodiny nejsou považovány za dostatečně přesné pro jeho bezproblémový provoz. V tom případě je možná pouze komunikace přes UART.

Protože UART byl kvůli své jednoduchosti použit dále během ladění aplikace, byl zvolen i pro programování. Komunikace přes USB nebyla zatím řešena. V budoucnu je ovšem velice pravděpodobný přechod z UARTu na USB, protože USB disponuje podstatně širšími možnostmi.

Komunikace po UARTu má následující parametry: 115200 bps, 8 bitů, žádná parita, 1 stop bit (115200 8N1).

SAM-BA monitor se ovládá pomocí několika jednoduchých textových příkazů.

Akce	Příkaz	Parametry	Příklad
Zapiš bajt	O	Adresa, Hodnota#	O200001,CA#
Přečti bajt	o	Adresa,#	o200001,#
Zapiš půl slovo (2 bajty)	H	Adresa, Hodnota#	H200002,CAFE#
Přečti půl slovo (2 bajty)	h	Adresa,#	h200002,#
Zapiš slovo (4 bajty)	W	Adresa, Hodnota#	W200000,CAFEDCA#
Přečti slovo (4 bajty)	w	Adresa,#	w200000,#
Pošli soubor	S	Adresa,#	S200000,#
Přijmy soubor	R	Adresa, délka#	R200000,1234#
Spusť program	G	Adresa#	G200200#
Zobraz verzi	V	#	V#

Tabulka 1: Příkazy SAM-BA monitoru [3]

Hodnoty všech parametrů jsou hexadecimální čísla. Není potřeba zarovnávat je na maximální šířku. Rozsah adres je 0x00000000 – 0xFFFFFFFF. Všechny příkazy končí znakem # (ASCII 0x23).

Odpověď na jakýkoli příkaz je <CR><LF>[hodnota]>. Hodnotu vrací pouze příkazy pro čtení (o, h, w) a požadavek na verzi (V). Pro příkazy zápisu vypadá vrácený řetězec následovně: "\r\n>".

3.2.1 Xmodem protokol

Pro posílání a přijímání souborů používá SAM-BA monitor Xmodem protokol. Jedná se o jednoduchý protokol, sestávající z paketů zajištěných 16bitovým CRC⁴. Přijetí paketu musí být protistranou potvrzeno, jinak musí být paket poslán znovu.

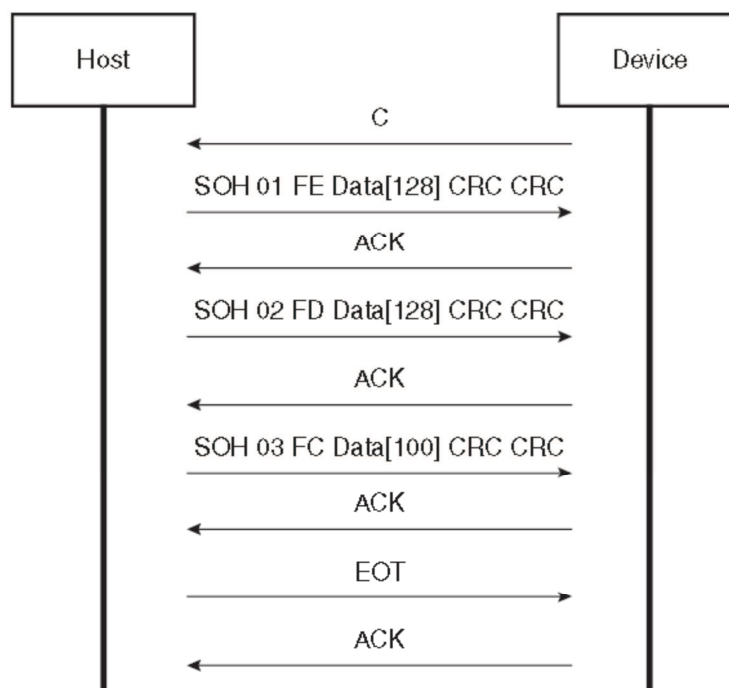
⁴ CRC – Cyclic Redundancy Check

Struktura paketu je následující:

<SOH><blk #><255-blk #><--128 data bytes--><CRC>

- <SOH>: ASCII Start Of Header = 0x01
- <blk #>: číslo paketu (1 byte); začíná se od 1; každý paket má číslo o jedna vyšší, než paket předchozí; dojde-li k přetečení, paket následující po paketu s číslem 0xFF má číslo 0x00 (NE 0x01)
- <255-blk #>: první doplněk k blk = 0xFF – blk
- <CRC>: 2 bytes CRC16CCITT, MSB first

Přijetí paketu přijímač potvrdí posláním ASCII znaku <ACK> = 0x06. Došlo-li během příjmu k chybě (nesedí číslo paketu, nebo CRC), vrátí přijímač <NAK> = 0x15. Konec přenosu signalizuje vysílač znakem <EOT> = 0x04.



Obrázek 2: Příklad Xmodem protokolu [3]

3.3 Způsob programování

K programování interní Flash paměti procesoru slouží Enhanced Embedded Flash Controller (EEFC). Jeho součástí jsou page buffer, Command register a Status Register. Postup programování je následující:

- Do page bufferu se nahrají data o velikosti jedné stránky Flash paměti (pro vybraný procesor 256 bajtů). **Do page bufferu se musí zapisovat vždy celé slovo (4 bajty) naráz.**
- Do Command registru se nahraje číslo zapisované stránky a příkaz pro zápis stránky (eventuálně příkaz pro smazání a zápis, pokud Flash paměť nebyla předem smazána).

- Ve Status registru se kontroluje bit FRDY (bit 0). V okamžiku, kdy je tento bit nastaven, je programování hotovo.

Tento způsob umožňuje naprogramovat Flash paměť pomocí příkazů *W* (zapiš slovo) a *w* (přečti slovo). Napřed je pomocí série příkazů *W* zapsána celá stránka do page bufferu (zápis se provádí na adresu, na které má být dané slovo v paměti umístěno. Interní logika EEFC se postará o zápis do page bufferu). Poté se pomocí příkazu *W* zapíše do Command registru číslo právě zapsané stránky spolu s příkazem pro zápis. Nakonec se pomocí příkazu *w* vyčítá obsah Status registru tak dlouho, dokud je bit FRDY 0. Toto se opakuje s každou další stránkou, dokud není požadovaná část paměti naprogramována.

Značná nevýhoda tohoto postupu je, že pro přenesení 4 bajtů užitečné informace (data do Flash paměti), musí po lince projít až 19 bajtů jedním směrem a 3 bajty opačným. Je to dáno za prvé tím, že se binární data přenáší v textové podobě (na jeden bajt potřebují dva znaky) a za druhé tím, že pro každé slovo musím znovu přenášet jeho adresu. Neefektivita tohoto přenosu je dobře patrná z toho, že v rámci jednoho přenosu činí užitečná data pouze cca 18 %.

Podstatně větší zdržení ovšem vzniká na straně PC při „přepínání“ mezi posíláním dat a příjmem odpovědi. Každý zápis, nebo čtení z COM portu znamená volání systémové funkce a předání řízení operačnímu systému. Protože pro vývoj běžně používané operační systémy (v tomto případě MS Windows) nejsou systémy reálného času, nedá se zajistit plynulý datový tok.

Experimentálně bylo zjištěno, že naprogramování 1 kB paměti trvá průměrně 4,4 s. Naprogramování celých 128 kB paměti použitého procesoru pak téměř 10 minut.

Toto prodlení by vývoj značně zdržovalo, proto je nutné použít jiný způsob. Vhodným řešením se zdá být použití Xmodem protokolu pro přenos celé stránky. Bohužel SAM-BA monitor zapisuje data do paměti tak jak přicházejí bajt po bajtu. Do page bufferu se ovšem musí zapisovat po slovech. Výsledkem tudíž bylo, že v každém slově se objevil čtyřikrát jeho poslední bajt.

Atmel tento problém řeší pomocí malých programů, takzvaných appletů, které se pomocí Xmodem protokolu nahrají do paměti RAM procesoru a pomocí příkazu *G* spustí. Různé varianty těchto appletů pak umožňují kromě interní paměti Flash programovat i externí paměti k procesoru připojené.

Nicméně všechny oficiální applety od Atmelu ke své funkci vyžadují, aby k procesoru byla připojena externí paměť RAM. Navržený hardware bohužel externí RAM nedisponuje. Proto bylo nutno naprogramovat vlastní applet, který by fungoval pouze s interní pamětí RAM.

Postup programování je pak následující:

1. Do paměti RAM se pomocí Xmodem protokolu (příkaz *S*) nahraje applet
2. Na předem dané adresy v paměti RAM se nahraje nastavení appletu (viz **Chyba! Nenalezen zdroj odkazů.**)
3. Na nastavenou adresu se nahraje stránka (případně několik) dat

4. Spustí se applet (příkazem *G*)
5. Bod 3 a 4 se opakuje, dokud nejsou naprogramovány všechny stánky.
6. Spustí se naprogramovaný kód (spustí se applet s příkazem 2)

Jméno	Adresa	Datový typ	Význam
source_address	0x2007C800	const uint32_t* const	Adresa, odkud se kopírují data; musí být zarovnaná na celé slovo
destination_address	0x2007C804	uint32_t*	Adresa, kam se kopírují data; musí být zarovnaná na celé slovo; po zápisu se automaticky zvedne o 4
start_page	0x2007C808	uint32_t	Číslo zapisované stránky, po zápisu se automaticky inkrementuje
pages_count	0x2007C80C	const uint32_t	Počet zapisovaných stránek
page_size	0x2007C810	const uint32_t	Velikost stránky v bajtech
cmd	0x2007C814	uint32_t	Příkaz (viz Tabulka 2: Nastavení appletu)

Tabulka 2: Nastavení appletu

Příkaz	Hodnota	Popis
Zapiš stránky	1	Zkopíruje stránku do page bufferu, pošle EEFC příkaz pro zapsání stránky a počká na dokončení zápisu.
Spust' program z interní Flash paměti	2	Nastaví GPNVM bit 1 a resetuje procesor
Spust' program z interní paměti ROM (SAM-BA monitor)	3	Vynuluje GPNVM bit 1 a resetuje procesor

Tabulka 3: příkazy appletu

3.4 Spuštění bootloaderu z uživatelské aplikace

Dostane-li uživatelská aplikace požadavek na vstup do bootladeru, mělo by stačit smazat GPNVM bit 1 a resetovat procesor. V praxi ovšem aplikace při pokusu o smazání GPNVM bitu „zamrzne“. Podle pozorovaného chování to vypadá, že okamžitě po smazání bitu dojde k přemapování paměti. Podle dokumentace k procesoru [3] by ovšem k přemapování paměti mělo dojít pouze při resetu.

Řešením je použití stejného appletu jako při programování. Zdrojový kód appletu je přibaleno k uživatelské aplikaci jako statická data (při kompilaci uživatelské aplikace). Při příjmu požadavku na spuštění bootladeru je applet nakopírován do paměti RAM a spuštěn s příkazem 3 (viz Tabulka 3).

Protože kód je načten z paměti RAM, smazání GPNVM bitu 1 a přemapování paměti Flash ho nijak neovlivní.

Jelikož applet musí být umístěn na konkrétní adrese, musí být část kódu uživatelské aplikace, která ho kopíruje a spouští, napsána v assembleru a používat pouze pracovní registry procesoru. Jinak by mohlo dojít buď k neúmyslnému přepsání appletu, nebo by tento nemusel být vůbec spuštěn.

3.5 PC klient

Protože vytvořený applet kvůli své jednoduchosti není kompatibilní s již existujícími, bylo nutno vytvořit klientský software pro PC, který by programování zajišťoval. Toho bylo dosaženo přidáním podpory daného protokolu do modulu Programátor programu Lorris [6] (multiplatformní open source software, obsahující sadu nástrojů pro vývoj Embedded zařízení).

4 AUDIO KODEK

Audio kodek je integrovaný obvod, který zprostředkovává rozhraní mezi analogovou a digitální částí zařízení z hlediska audio signálu. Mezi jeho nejdůležitější funkce patří:

- přizpůsobení amplitudy vstupního signálu
- napájení mikrofonu, pokud je vyžadováno
- digitalizace vstupního analogového signálu
- převod výstupního signálu z digitálního na analogový (plus převzorkování a odstranění vzorkovací frekvence, aby tato nepůsobila ve výsledném signálu rušivě)
- nastavení hlasitosti výstupního signálu
- impedanční a výkonové přizpůsobení výstupu

Všechny informace o použitém kodeku TLV320AIC3101 byly čerpány z [7].

4.1 Komunikace s kodekem

Kodek obsahuje dvě komunikační rozhraní. Pomocí I²C se provádí nastavení parametrů. Pomocí I²S jsou kodeku posílána audio data pro DA převodníky a vyčítány AD převodníky.

Dále je potřeba ovládat $\overline{\text{RESET}}$ pin. Kodek po připojení napájení vyžaduje reset, jinak komunikace s ním neprobíhá korektně. Délka resetovacího pulzu musí být minimálně 10 ns.

Volitelně je možno do kodeku přivést externí hodinový signál pomocí pinu MCK. Alternativou je použití bitových hodin I²S linky.

4.1.1 Nastavení kodeku

Kodek obsahuje celkem 256 osmibitových registrů, sloužících pro jeho nastavení. Tyto registry jsou rozděleny do dvou bank po 128 registrech. Registr 0 v každé bance slouží pro nastavení aktuální banky. Ne všechny registry jsou využité.

Banka jedna obsahuje nastavení koeficientů integrovaných digitálních filtrů. Banka nula pak všechna ostatní nastavení.

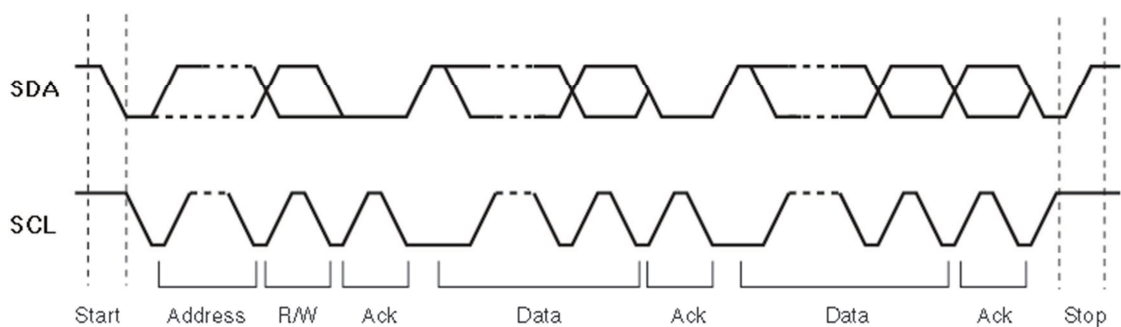
Po resetu je vždy vybraná banka nula.

Nastavování kodeku probíhá přes I²C sběrnici. Jedná se o polo duplexní synchronní sériovou sběrnici. Tato sestává ze dvou vodičů: SCL, který přenáší hodiny a SDA, který přenáší data. Oba vodiče jsou upínacími rezistory nastaveny na log. 1. Zařízení připojená ke sběrnici pak v případě potřeby přivedou na požadovaný vodič log. 0. Tímto je realizována logická funkce AND, čímž je umožněno připojit ke sběrnici více zařízení typu Master bez nebezpečí vzniku hazardních stavů, pokud by se do sběrnice pokusilo zapsat více zařízení naráz. Součástí specifikace I²C [8] jsou metody jak vícenásobný simultánní zápis detekovat a vyrovnat se s ním.

Komunikaci po sběrnici vždy řídí jedno zařízení, označované jako Master. Ten může komunikovat vždy pouze s jedním zařízením typu Slave v rámci daného přenosu. Master přenos začíná (posláním START, nebo Repeated START), určuje, s kým bude komunikovat (pole Slave address), směr přenosu a ukončuje přenos (posláním STOP). Dále Master generuje hodinový signál, čímž určuje rychlost komunikace. Přijetí každého bajtu musí být zařízením (Slave v režimu zápisu, Master v režimu čtení) potvrzeno pomocí ACK (acknowledgment) bitu.

Každý přenos sestává z:

- START bitu
- R/ \overline{W} bitu, určujícího směr přenosu
- sedmibitové adresy Slave zařízení, se kterým chce Master komunikovat
- ACK bitu, potvrzujícího úspěšné přijetí
- libovolného počtu datových bajtů, vždy následovaných ACK bitem
- STOP bitu (eventuálně Repeated START bitu, pokud za právě dokončeným přenosem neprodleně následuje další).



Obrázek 3: I²C přenos [3]

Pro zápis hodnoty do registru kodeku je potřeba poslat následující:

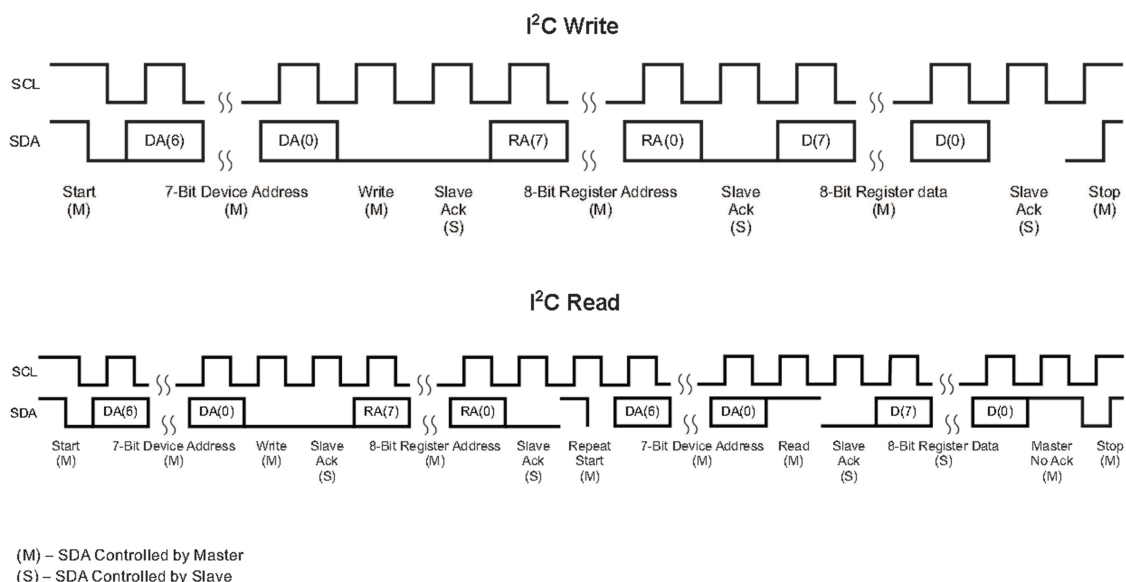
- START
- adresu kodeku 0x18 s příkazem pro zápis
- adresu vybraného registru v rozsahu 0 ÷ 127
- hodnotu, kterou chci do registru zapsat
- STOP

Pro přečtení hodnoty registru je potřeba následující:

- poslat START
- poslat adresu kodeku 0x18 s příkazem pro zápis
- poslat adresu vybraného registru v rozsahu 0 ÷ 127
- poslat Repeated START
- poslat adresu kodeku 0x18 s příkazem pro čtení
- přečíst kodekem poslaná data

- každý přečtený bajt kromě posledního musí být potvrzen posláním ACK bitu; poslední bajt, který má být vyčten, je následován NACK bitem, čímž se kodeku naznačuje ukončení přenosu
- poslat STOP

V rámci jednoho přenosu je možno zapsat nebo přečíst více po sobě následujících registrů.



Obrázek 4: Komunikace s audio kodekem pomocí I²C [7]

4.1.2 Audio data

Přenos audio dat mezi kodekem a procesorem probíhá pomocí I²S sběrnice. Ač název vypadá podobně, jedná se o zcela jinou sběrnici než dříve popsaná sběrnice I²C.

Jedná se o plně duplexní synchronní sériovou sběrnici, primárně určenou právě pro přenos digitálního audia. Sběrnice sestává celkem ze čtyř vodičů:

- TD – transmit data – pro přenos dat z mikroprocesoru do DA převodníků audio kodeku
- RD – receive data – pro přenos dat z AD převodníků kodeku do mikroprocesoru
- BCLK – bit clock – vzorkování TD a RD probíhá na náběžné hraně signálu
- WCLK – word clock – hrana signálu určuje začátek přenosu, polarita signálu určuje, jestli jsou data určena pro levý (log. 0), nebo pravý (log. 1) kanál.

Data jsou přenášena nejvýznamnějším bitem napřed.

Signál BCLK může být také využit jako zdroj hodinového signálu pro audio kodek, místo pinu MCLK. Tato možnost ovšem momentálně není využita. Dále je uvedeno proč.

Jak audio kodek, tak procesor umožňují generovat nezávisle (oba, kterýkoli, nebo žádný) oba hodinové signály. Audio kodek ovšem nedisponuje vyrovnávací pamětí.

Frekvence WCLK tedy musí odpovídat přímo vzorkovací frekvenci AD a DA převodníků. Frekvence BCLK se pak určí následovně:

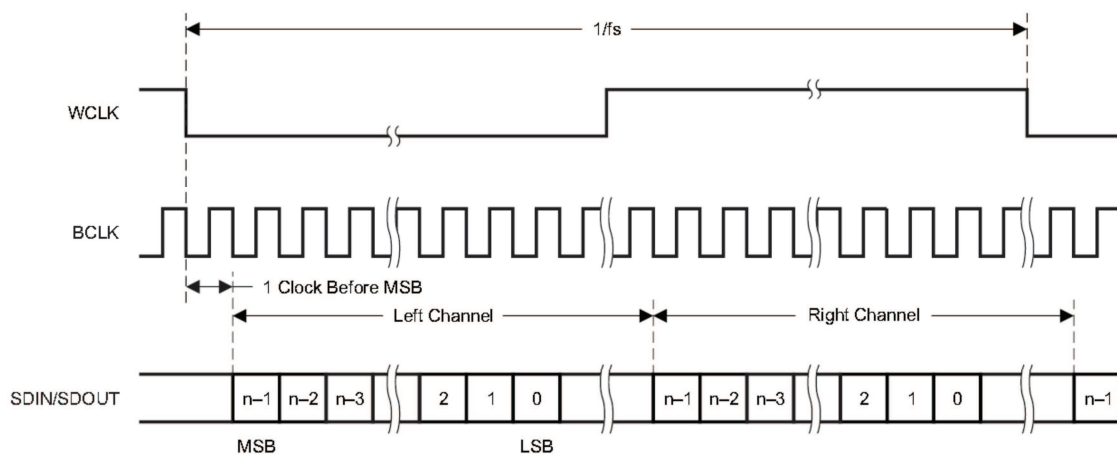
$$f_{BCLK} = 2 \times f_{WCLK} \times W_{len} \times N_{dev} \quad (1)$$

kde

- f_{BCLK} je frekvence BCLK v Hz
- f_{WCLK} je frekvence WCLK v Hz
- W_{len} je délka jednoho vzorku audio signálu v bitech – podle nastavení kodeku buď 16, 20, 24, nebo 32
- N_{dev} je počet zařízení při využití TDM (Time-Division Multiplexing) [9], v tomto případě vždy 1

Pro zajištění optimální funkce je potřeba, aby WCLK i BCLK měli minimální jitter. Z toho důvodu je na MCLK vstup kodeku přiveden hodinový signál s frekvencí 12 MHz z krystalového oscilátoru mikroprocesoru. Kodek pak pomocí interní frekvenční násobičky a děliček generuje oba signály, WCLK i BCLK. Prakticky se tak na I²S sběrnici chová jako Master.

Další výhodou tohoto uspořádání je, že při požadavku na změnu vzorkovací frekvence stačí změnit nastavení kodeku. Není potřeba měnit zároveň i nastavení komunikačního rozhraní na mikroprocesoru.



Obrázek 5: I²S komunikace [7]

4.2 Základní nastavení kodeku

V této kapitole je stručně popsáno základní použité nastavení kodeku. Kompletní možnosti nastavení kodeku jsou velmi rozsáhlé, a jsou podrobně uvedeny v [7].

Pro zlepšení čitelnosti kódu výsledného programu bylo vytvořeno API, zapouzdřující kompletně celé nastavení kodeku. Všechna jeho nastavení jsou prováděna pomocí tohoto API (viz příloha A).

4.2.1 Hodiny

Většina nastavení kodeku, týkající se frekvence, nebo času, se odvíjí od referenční frekvence $f_{S(ref)}$. Tato musí být buďto 48 kHz (aktuálně použitá), nebo 44,1 kHz (odvozeno od audio CD). Navíc kodek interně potřebuje frekvenci rovnou 256násobku $f_{S(ref)}$. Kodek disponuje nastavitelnou frekvenční násobičkou a děličkou, které umožňují vytvoření požadované frekvence z širokého rozsahu vstupních frekvencí (v tomto případě 12 MHz).

Nastavení hodin (viz **Obrázek 6**):

- Zdroj MCLK 12 MHz
- PLL
 - $P = 1$
 - $R = 1$
 - $K = 8,192$
 - $J = 8$
 - $D = 1920$

4.2.2 Vzorkovací frekvence

Vzorkovací frekvence f_S se určí z referenční frekvence podělením hodnotou $NDAC$:

$$f_S = \frac{f_{S(ref)}}{NDAC} \times DRA \quad (2)$$

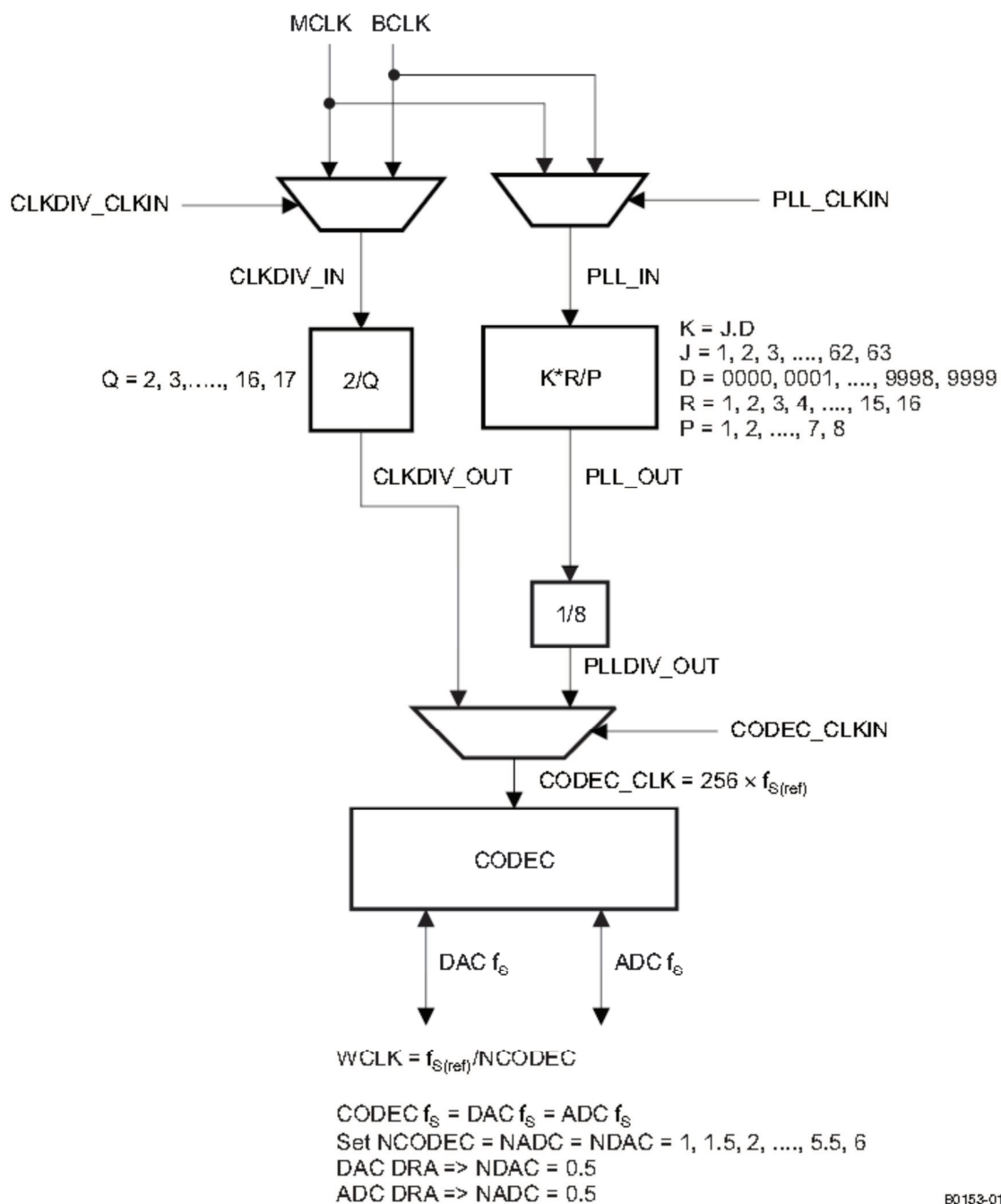
Parametr $NDAC$ může být jedna z hodnot 1; 1,5; 2; 2,5; 3; 3,5; 4; 4,5; 5; 5,5; 6. Parametr DRA může nabývat hodnot 1 nebo 2. Umožňuje zdvojnásobit vzorkovací frekvenci. Díky tomu je možno dosáhnout vzorkovací frekvence až 96 kHz.

Kvůli požadavku na minimální šířku pásma je potřeba zvolit vzorkovací frekvenci co nejnižší. Frekvenční složky lidského hlasu, zajišťující srozumitelnost, se pohybují v rozsahu přibližně 300 Hz ÷ 3,4 kHz [10]. Podle vzorkovacího teorému musí být vzorkovací frekvence minimálně dvakrát vyšší, než maximální frekvence vzorkovaného signálu. V kombinaci s možnostmi kodeku vychází vzorkovací frekvence 8 kHz. Podle rovnice (2) tedy $NDAC = 6$.

4.2.3 Komunikace

Jak již bylo zmíněno, audio data si kodek s mikroprocesorem vyměňuje přes I²S sběrnici, na které se chová jako Master (generuje hodiny). Hodiny jsou generovány pouze při komunikaci (pokud je AD i DA převodník vypnutý, není důvod hodiny posílat).

Šířka slova byla zvolena 16 bitů. Kodek sice disponuje 24bitovými převodníky, nicméně 16 bitů je pro danou aplikaci více než dostačující.



Obrázek 6: Nastavení hodin audio kodeku [7]

4.2.4 Vstupní signál

Kromě interního mikrofону obsahuje navržený hardware také konektor pro připojení sluchátek s mikrofonom. Jedná se čtyř pinový konektor typu Jack 3,5 mm, jaké se dnes běžně používají u chytrých telefonů⁵. Kodek by měl umět rozpoznat připojení

⁵ Bohužel existují dvě varianty zapojení tohoto konektoru, které se vzájemně liší pořadím pinů GND a MIC (viz **Obrázek 7**). V rámci větší kompatibility by bylo vhodné mezi audio kodek a konektor vložit integrovaný obvod, který umožnil použití obou variant. Například TS3A226AE od firmy Texas Instruments [30].

konektoru konfiguraci připojeného zařízení (jedná-li se pouze o sluchátka, nebo o sluchátka s mikrofonom). Bohužel tuto funkci se nepovedlo uvést do provozu. Z toho důvodu je zatím vstup nastaven na pevně integrovaný mikrofón.



Obrázek 7: Headset connector [11]

Byl použit běžný elektretový mikrofón. Tento typ mikrofónů obsahuje zabudovaný jednostupňový tranzistorový zesilovač, proto potřebuje napájení. To je schopen poskytnout přímo audio kodek. Podle [7] je pro optimální funkci rozpoznávacího bloku potřeba mít co nejvyšší napětí. Podle [12] je maximální povolené napětí mikrofónu 10 V, tudíž není limitující. Napájecí napětí tedy by zvoleno $AVDD = 3,3\text{ V}$.

Mikrofón je připojen diferenciálně k levému vstupnímu diferenciálnímu kanálu (MIC1LP/LINE1LP, MIC1LM/LINE1LM).

Na vstupu je aktivováno automatické nastavení hlasitosti (AGC – Automatic Gain Control). Výchozí nastavení AGC se ukázalo jako dostačující.

Výstupní signál z AGC je přiveden na levý AD převodník.

Externí vstup je připojen na pravý kanál (MIC2R-LINE2R). Tento je možné interně připojit jak na levý, tak na pravý AD převodník (přes odpovídající AGC). Díky tomu je možné vybrat si zdroj signálu na úrovni kodeku, snímat současně oba signály, nebo je dokonce analogově mixovat dohromady (s nastavitelným poměrem).

4.2.5 Výstupní signál

Navržený hardware má dva výstupy. Jedním je $8\ \Omega$ Reprodukter, diferenciálně připojený k tomu určenému výstupu kodeku (HPLCOM, HPRCOM). Druhým je stereofonní výstup, připojený přes kapacitní vazbu na konektor pro sluchátka.

Do reproduktoru je možno přivést buď signál z levého DA převodníku, nebo analogový mono mix stereofonního signálu z obou DA převodníků (součet obou signálů

podělených dvěma). Do sluchátek je možno přivést stereofonní signál, signál z levého DA převodníku (do obou sluchátek stejný), nebo opět mono mix.

Ve výchozím nastavení je jak do reproduktoru, tak do sluchátek přiveden výstup levého DA převodníku.

Vzhledem k již zmíněné nefunkčnosti automatické detekce připojení sluchátek je jako primární výstup použit reproduktor. Na sluchátka je možno přepnout ručně.

Ochrana výstupu proti zkratu je aktivní a nastavená na omezení dodávaného proudu.

4.2.6 Nastavení hlasitosti

O nastavení hlasitosti vstupního signálu se stará AGC. Výstupní úroveň je nastavena na -5,5 dB, aby byl maximálně využit rozsah AD převodníku. V případě potřeby by se dal signál zeslabit omezením maximálního zesílení AGC. Ostatní parametry AGC jsou ponechány ve výchozím nastavení.

Výstupní hlasitost jak reproduktoru, tak sluchátek je nastavena na -38,7 dB. Nastavení hlasitosti se dá jednoduše změnit podle aktuálních požadavků. Rozsah hlasitosti je 0 dB až -78,3 dB s krokem přibližně 0,5 dB (se vzrůstajícím útlumem signálu se velikost kroku mírně mění, viz příloha A.5).

5 SPEEX

Jak je uvedeno v [1], jsou pro komprese digitálního hlasu volně dostupné 3 kodeky⁶: Codec2, Opus a Speex.

Nejmenšího datového toku (tudíž největší komprese) dosahuje Codec2 – 1200 bps [13]. Bohužel Codec2 používá výpočty v plovoucí čárce a použitý mikroprocesor nedisponuje FPU⁷ a softwarová emulace je pomalá. Pro použití tohoto kodeku by i za předpokladu FPU byl potřeba přibližně 2,5krát větší výpočetní výkon [13].

Opus je z vybraných kodeků nejflexibilnější. Dá se použít jak pro aplikace vyžadující minimální datový tok, tak pro aplikace vyžadující především maximální kvalitu audia [14]. Experimentálně bylo zjištěno, že Opus pro kompresi mono audia vyžaduje 40838 B paměti RAM. Použitý procesor disponuje pouze 32 kB [3], tudíž je Opus nepoužitelný.

Zbývá tedy Speex. Jedná se o kodek postavený na metodě CELP – Code Excited Linear Prediction. Popis principů CELP je v [15].

Byla použita verze Speex 1.2-RC1.

Při nastavení kvality na 3 (viz Tabulka 4) bude výsledný datový tok pouze 8000 bps.

5.1 Přizpůsobení kodeku

Speex je distribuován ve formě generických zdrojových kódů, napsaných v jazyce C. Díky tomu je možné Speex jednoduše přizpůsobit požadavkům konkrétní aplikace.

5.1.1 Optimalizace

V [1] je uvedeno, že jedním z důvodů pro výběr mikroprocesoru byla optimalizovaná verze kodeku Speex pro jeho jádro (ARM® Cortex®-M3 [3]). Tato optimalizace spočívá v přepsání kritických úseků kódu do assembleru. Byla vytvořena firmou STMicroelectronics pro její rodinu mikroprocesorů STM32F [16]. Přesto že jádro mikroprocesoru, a tedy i instrukční sada, jsou stejné, nepodařilo se kód zkompileovat pro použitý mikroprocesor. Důvodem je odlišná syntaxe některých částí inline assembleru. Z časových důvodů se nepodařilo ji převést na syntaxi odpovídající použitému překladači. Do budoucna by bylo vhodné se tomuto problému více věnovat.

5.1.2 Nastavení kompilace

Speex umožňuje pomocí definování určitých symbolů nastavit, které části knihovny se budou požívat. Bohužel ne všechny tyto přepínače jsou zdokumentované. Zdrojový kód Speexu je rozsáhlý a vyhledání všech možností přímo v kódu by zabralo značné množství času. Zde jsou proto uvedeny pouze využité možnosti a jejich stručný popis.

⁶ V této kapitole se pojmem kodek myslí kompresní algoritmus.

⁷ FPU – Floating Point Unit – hardware umožňující počítání s plovoucí desetinnou čárkou.

- `FIXED_POINT` – všechny výpočty jsou prováděny s pevnou desetinou čárkou (tzn. V celých číslech)
- `DISABLE_FLOAT_API` – nezahrne do kompilace funkce jakkoli využívající plovoucí desetinou čárku. Protože se v plovoucí čárce nic nepočítá (díky `FIXED_POINT`), jsou tyto zbytečné.
- `DISABLE_VBR` – zakáže Variable Bit Rate – proměnou přenosovou rychlost. Tato funkce by sice mohla přinést zvýšení kvality přenosu při zachování průměrného datového toku, ale je implementovaná pouze v plovoucí desetinné čárce
- `USE_KISS_FFT` – použití knihovny Kiss FFT [17] pro Fourierovu transformaci. Alternativou je `USE_SMALL_FFT`. Doposud nebylo experimentálně ověřeno, která metoda by byla vhodnější.
- `USE_ALLOCA` – pro dynamickou alokaci paměti jsou použity standartní systémové funkce. Alternativou je `VAR_ARRAYS`, kdy by si Speex měl na začátku staticky alokovat maximální potřebné množství paměti a neprovádět dynamickou alokaci. Zdá se ovšem, že tato možnost nefunguje korektně.
- `OS_CUSTOM_SUPPORT` – Speex nepoužívá přímo systémové funkce. Místo toho má vlastní funkce, které volají originální systémové. V případě potřeby je možno tyto jeho funkce předefinovat a vyhnout se tak volání systémových funkcí. Toto bylo nezbytně nutné, jak je popsáno dále.

5.1.3 Systémové funkce

Základní předpoklad je, že na daném zařízení běží nějaký operační systém, který zajišťuje standartní výstup, správu paměti, ukončování procesů v případě kritické chyby, apod.

Kvůli úspoře výpočetního výkonu však není žádný operační systém použit. Bylo proto nutné dopsat některé funkce, které běžně poskytuje operační systém. Využití některých dalších bylo nahrazeno dostupnými možnostmi.

Dopsány byly následující systémové funkce:

- `void* _sbrk (int ptrdif)`
Tato funkce zajišťuje dynamickou alokaci paměti. Jsou na ní postaveny běžně používané funkce jako `malloc` a `free`. Parametr `ptrdiff` určuje kolik místa je potřeba alokovat (kladná hodnota), nebo uvolnit (záporná hodnota). Funkce vrací ukazatel na začátek právě alokovaného bloku paměti (v případě alokace), nebo na konec bloku právě uvolněného (v opačném případě). Specifikace funkce je v [18]. Implementace je následující. Na začátku programu je staticky alokovan blok paměti o velikosti 16 kB. Dále je definován ukazatel, ukazující na začátek tohoto bloku. Při volání funkce `_sbrk` je k tomuto ukazateli přičtena hodnota

ptrdiff. Pokud výsledek ukazuje dovnitř alokovaného bloku paměti, je vrácena původní hodnota ukazatele (před přičtením). V opačném případě je globální proměnná `errno` nastavena na `NOMEM` (12) a je vráceno `(void*)-1`.

- `void _exit(int status)`

Pokud Speex detekuje interní chybu, pokusí se ukončit svůj proces voláním funkce `exit`. Specifikace funkce je v [19]. Implementace je následující: vypsání hlášky na debug výstup a reset mikroprocesoru.

Následující funkce z implementace Speex byly pozměněny:

- `void _speex_fatal(const char *str, const char *file, int line)`
- `void speex_warning(const char *str)`
- `void speex_warning_int(const char *str, int val)`
- `void speex_notify(const char *str)`
- `void _speex_putc(int ch, void*)`
- `void print_vec(float *vec, int len, const char *name)`

Výše zmíněné funkce slouží k informování uživatele. V originále využívají systémovou funkci `fprintf` a buď standartní výstup `stdout`, nebo chybový výstup `stderr`. Funkce byly pozměněny tak, aby pro výstup používali formátovací třídu `format` s výstupem nasměřovaným na debug.

5.2 API

Kompletní dokumentace k Speex API⁸ je k dispozici v [20]. Zde jsou vypsány pouze použité části.

5.2.1 SpeexBits

`SpeexBits` je datový typ, reprezentující proud bitů mezi enkodérem a dekodérem.

Použité funkce pro manipulaci se `SpeexBits`:

- `void speex_bits_init (SpeexBits *bits)`

Inicializuje strukturu `SpeexBits`

Parametry:

- `bits` – ukazatel na strukturu, která má být inicializována

- `void speex_bits_reset (SpeexBits *bits)`

Vymaže uložená data a uvede strukturu do výchozího stavu (jako po inicializaci)

Parametry:

- `bits` – ukazatel na strukturu, která má být resetována

- `void speex_bits_destroy (SpeexBits *bits)`

Uvolní všechnu paměť spojenou se strukturou `SpeexBits`

⁸ API – Application Programming Interface – rozhraní, umožňující programátorovi využívat služeb knihovny

Parametry:

- bits – ukazatel na strukturu, která má být smazána
- void speex_bits_read_from (SpeexBits *bits, char *bytes, int len)

Uloží do struktury data z paměti. Neprovádí kopírování dat, pouze nastaví interní ukazatele a ostatní prvky struktury.

Parametry:

- bits – ukazatel na strukturu, do které mají být data uložena
- bytes – ukazatel na blok dat v paměti
- len – délka bloku dat v bajtech
- int speex_bits_write (SpeexBits *bits, char *bytes, int max_len)

Zapíše data ze struktury do paměti.

Parametry:

- bits – ukazatel na strukturu obsahující data
- bytes – ukazatel na předem alokovaný blok paměti, do kterého mají být data uložena
- max_len – maximální počet bajtů, které je možno do paměti uložit (velikost alokovaného bloku)

Návratová hodnota: počet bajtů, které byly do paměti uloženy.

5.2.2 Enkodér

- void * speex_encoder_init (const SpeexMode *mode)

Vytvoří datovou strukturu reprezentující enkodér.

Parametry:

- mode – ukazatel na strukturu obsahující základní nastavení enkodéru. Na výběr je některá ze tří výchozích hodnot (jedná se o globální struktury, je potřeba předávat jejich adresu):
 - speex_nb_mode – vhodný pouze pro řeč, minimální datový tok, vzorkovací frekvence 8 kHz
 - speex_wb_mode – kompromis mezi kvalitou a datovým tokem, vzorkovací frekvence 16 kHz
 - speex_uwb_mode – nejlepší kvalita, vzorkovací frekvence 32 kHz

Návratová hodnota: ukazatel na strukturu reprezentující enkodér

- void speex_encoder_destroy (void *state)

Uvolní veškerou paměť alokovanou enkodérem.

Parametry:

- state – ukazatel na strukturu reprezentující enkodér
- int speex_encode_int (void *state, spx_int16_t *in, SpeexBits *bits)

Zkomprimuje jeden audio rámeček (standartně 20 ms).

Je-li požadován více než jeden audio rámeček ve výsledném datovém paketu, doporučená metoda je zavolat vícekrát `speex_encode_int`, než je vytvořen paket voláním `speex_bits_write`.

Parametry:

- `state` – ukazatel na strukturu reprezentující enkodér
- `in` – ukazatel na vstupní audio data – pole hodnot typu `int16_t` v rozsahu $\pm 2^{15}$. **Během komprimace jsou tato data změněna!**
- `bits` – ukazatel na strukturu `SpeexBits`, do které jsou zapsána zkomprimovaná data.

Návratová hodnota: 0, pokud zkomprimovaný rámeček není třeba odesílat (pouze DTX⁹), jinak 1.

- `int speex_encoder_ctl (void *state, int request, void *ptr)`

Funkce sloužící k nastavování, nebo vyčítání parametrů enkodéru.

Parametry:

- `state` – ukazatel na strukturu reprezentující enkodér
- `request` – kód požadované operace. Použité kódy:
 - `SPEEX_GET_FRAME_SIZE = 3`
`ptr` je ukazatel na `int`, do kterého je uložena velikost audio rámce ve vzorcích (ne v bajtech)
 - `SPEEX_SET_QUALITY = 4`
`ptr` je ukazatel na `int` v rozsahu $0 \div 10$ viz Tabulka 4.
 - `SPEEX_RESET_STATE = 26`
`ptr = NULL`
- `ptr` – ukazatel sloužící pro předávání dat viz parametr `request`

5.2.3 Dekodér

- `void * speex_decoder_init (const SpeexMode *mode)`

Vytvoří datovou strukturu reprezentující dekodér.

Parametry:

- `mode` – ukazatel na strukturu obsahující základní nastavení dekodéru. Na výběr je některá z tří výchozích hodnot (jedná se o globální struktury, je potřeba předávat jejich adresu):
 - `speex_nb_mode` – vhodný pouze pro řeč, minimální datový tok, vzorkovací frekvence 8 kHz
 - `speex_wb_mode` – kompromis mezi kvalitou a datovým tokem, vzorkovací frekvence 16 kHz
 - `speex_uwb_mode` – maximální kvalita zvuku, vzorkovací frekvence 32 kHz

⁹ DTX – Discontinuous Transmission – umožňuje snížit průměrný datový tok tím, že neodesílá rámce, ve kterých není řeč, ale pouze statický šum pozadí.

Návratová hodnota: ukazatel na strukturu reprezentující dekodér

- `void speex_decoder_destroy (void *state)`

Uvolní veškerou paměť alokovanou dekodérem.

Parametry:

`state` – ukazatel na strukturu reprezentující dekodér

- `int speex_decode_int (void *state, SpeexBits *bits, spx_int16_t *out)`

Dekóduje jeden audio rámeček (standardně 20 ms).

Parametry:

- `state` – ukazatel na strukturu, reprezentující dekodér
- `bits` – ukazatel na strukturu `SpeexBits`, ve které jsou komprimovaná data, nebo `NULL`, pokud byl paket ztracen
- `out` – ukazatel na pole typu `int16_t`, do kterého jsou uložena dekódovaná audio data. Pole musí být dostatečně velké, aby se do něj vešel jeden audio rámeček.

Návratová hodnota:

- 0 – vše v pořádku
- -1 – konec přenosu
- -2 – vstupní data obsahují chybu

- `int speex_decoder_ctl (void *state, int request, void *ptr)`

Funkce sloužící k nastavování, nebo vyčítání parametrů dekodéru.

Parametry:

- `state` – ukazatel na strukturu reprezentující dekodér
- `request` – kód požadované operace. Použité kódy:
 - `SPEEX_GET_FRAME_SIZE = 3`
`ptr` je ukazatel na `int`, do kterého je uložena velikost audio rámce ve vzorcích (ne v bajtech)
 - `SPEEX_RESET_STATE = 26`
`ptr = NULL`
- `ptr` – ukazatel sloužící pro předávání dat viz parametr `request`

Speex quality [-]	Datový tok [bps]	Výpočetní náročnost [MFLOPS ¹⁰]	popis ¹¹
0	2150	6,0	Pouze pro přenos šumu
1	3950	10,5	Značný šum / zkreslení, dobrá srozumitelnost
2	5950	9,0	Značný šum / zkreslení, dobrá srozumitelnost
3, 4	8000	10,0	Občasný šum / zkreslení
5, 6	11000	14,0	Znatelné zkreslení pouze ve sluchátkách
7, 8	15000	11,0	Rozdíl mezi vstupem a výstupem rozeznatelný pouze za použití dobrých sluchátek
9	18200	17,5	Rozdíl mezi vstupem a výstupem prakticky nerozeznatelný
10	24600	14,5	Pro hlas vstup a výstup bez rozdílu, dobrá kvalita i pro hudbu

Tabulka 4: Datový tok a výpočetní náročnost enkodéru vs. nastavená kvalita [21]

5.3 Příklad použití kodeku

Pro použití kodeku je nezbytné připojit hlavičku speex.h:

```
#include <speex/speex.h>
```

5.3.1 Enkodér

Vytvoření potřebných datových struktur:

```
SpeexBits bits;
void *enc_state;
```

Jejich inicializace:

```
speex_bits_init(&bits);
enc_state = speex_encoder_init(&speex_nb_mode);
```

Nastavení požadované kvality:

```
int quality = 4;
speex_encoder_ctl(enc_state, SPEEX_SET_QUALITY, &quality);
```

Zjištění velikosti audio rámce:

```
int frame_size;
speex_encoder_ctl(enc_state, SPEEX_GET_FRAME_SIZE, &frame_size);
```

Další proměnné potřebné pro funkci enkodéru:

```
// pole, ze kterého se načítají vstupní audio data
int16_t* input_frame = malloc(frame_size * sizeof(int16_t));
// pole, do kterého jsou ukládána komprimovaná data
static const int MAX_NB_BYTES = 128;
char byte_ptr[MAX_NB_BYTES];
```

¹⁰ FLOPS – FLOating-point OPerations per Second – počet operací v plovoucí čárce za sekundu.
MFLOPS – Mega FLOPS = 10⁶ FLOPS. [30]

¹¹ Objektivní měření je značně problematické, pouze subjektivní dojem.

```
// kolik bajtů bylo do výstupního pole skutečně uloženo
int nbBytes;
Tím je inicializace hotova. Dále se pro každý rámec provede:
speex_bits_reset(&bits);
speex_encode_int(enc_state, input_frame, &bits);
nbBytes = speex_bits_write(&bits, byte_ptr, MAX_NB_BYTES);
Po dokončení komprimování uvolnění použitých zdrojů:
speex_bits_destroy(&bits);
speex_encoder_destroy(enc_state);
free(input_frame);
```

5.3.2 Dekodér

Vytvoření potřebných datových struktur:

```
SpeexBits bits;
void *dec_state;
```

Jejich inicializace:

```
speex_bits_init(&bits);
dec_state = speex_decoder_init(&speex_nb_mode);
```

Zjištění velikosti audio rámce:

```
int frame_size;
speex_decoder_ctl(dec_state, SPEEX_GET_FRAME_SIZE, &frame_size);
```

Další proměnné potřebné pro funkci dekodéru:

```
// pole, do kterého se ukládají dekódovaná audio data
int16_t* output_frame = malloc(frame_size * sizeof(int16_t));
// pole, ze kterého jsou načítána komprimovaná data
char input_bytes[128];
// kolik bajtů bylo do výstupního pole skutečně uloženo
int nbBytes; // délka komprimovaného rámce, viz speex_bits_write
```

Tím je inicializace hotova. Dále se pro každý rámec provede:

```
speex_bits_read_from(&bits, input_bytes, nbBytes);
speex_decode_int(dec_state, &bits, output_frame);
```

Po dokončení komprimování uvolnění použitých zdrojů:

```
speex_bits_destroy(&bits);
speex_decoder_destroy(dec_state);
free(output_frame);
```

6 KOMUNIKAČNÍ PROTOKOL

Aby spolu mohla zařízení vzájemně komunikovat, musí používat stejný komunikační protokol. V případě digitálního audio řetězce jsou požadavky na komunikaci následující:

- Každé zařízení musí být schopno posílat data všem ostatním
- Možnost nezávislé komunikace více skupin zařízení na stejném kanále
- Minimální datový tok
- Maximální využití hardwarových prostředků

Použitý kodek Speex je podstatně odolnější vůči ztraceným paketům, než proti poškozeným [21]. Proto je důležité rozpoznávat chyby v přenosu. Naopak není nezbytně nutné znovu posílat chybně přijaté pakety. Vypadne-li sem tam nějaký, na srozumitelnost komunikace to nebude mít podstatný vliv. Pokud by v pořádku nedorazila většina paketů, je problém pravděpodobně v přenosové cestě. V tom případě je potřeba hledat řešení jinde, například zvolit jiný kanál. Přeposílání chybně přijatých paketů by problém velice pravděpodobně nevyřešilo.

Aby byly všechny výše zmíněné požadavky splněny, bylo nutné navrhnout vlastní komunikační protokol. Navržený protokol z velké části využívá hardwarové podpory komunikace pomocí paketů, kterou poskytuje použitý RF modul (viz kapitola 7.2.2). Ta sestává z:

- detekce začátku paketu.
- nastavitelné hlavičky délky $0 \div 4$ bajty.
- nastavitelné délky paketu v rozmezí $0 \div 255$ bajtů.
- kontroly přenášených dat pomocí CRC¹².

Všechny tyto prvky jsou využity. Mikroprocesor zpracovává hlavičku, délku paketu a data. Zbytek paketu je v režii RF modulu.



Obrázek 8: Struktura datového paketu [4]

¹² CRC – Cyclic Redundancy check – metoda detekce chyb v datech [30].

Část paketu	Velikost v bajtech
Preamble	4
Sync word	2
Header	1
Length	1
Data	40
CRC	2

Tabulka 5: Velikost jednotlivých částí paketu

V **Chyba! Nenalezen zdroj odkazů.** jsou uvedeny velikosti jednotlivých částí paketu. Důležitá je především délka datové části paketu. Ostatní položky mají většinou délku doporučenou výrobcem RF modulu (viz [4]). Pro přenos všech ostatních nezbytných informací, kromě datové části paketu, je potřeba celkem 10 bajtů. Při daném datovém toku 8000 bps a délce rámce 20 ms (viz kapitola 5), vychází délka jednoho rámce následovně:

$$l = \frac{8000}{8} \times 0.02 = 20 [B]. \quad (3)$$

Posílat každý rámec zvlášť by tudíž znamenalo navýšení datového toku o 50 %. Kvůli možné ztrátě paketu není dobré posílat naráz více rámců. Kompromis jsou 2 rámce na paket, čímž je dána délka datové části paketu 40 bajtů. Celková délka paketu je pak 50 bajtů.

6.1 Řízení komunikace

Každé zařízení vždy vysílá všem ostatním a musí od všech ostatních přijímat. Při bezdrátové komunikaci pomocí rádiových vln je toho dosaženo tím, že všechna zařízení pracují na stejné frekvenci. Jediné, co je potřeba zajistit je, aby vždy vysílalo pouze jedno zařízení. Kvůli dosažení maximální flexibility a minimalizaci přebytečné komunikace byl použit modifikovaný token ring. Při tomto způsobu komunikace si zařízení postupně předávají právo vysílat (token).

Počet komunikujících zařízení není předem znám a může se v průběhu času měnit. Proto poté, co poslední zařízení odvysílá svá data, je zařazena pauza, trvající minimálně stejně dlouho, jako nejdelší možný vyslaný rámec. Během této pauzy má nové zařízení šanci zapojit se do komunikace. Poté přechází vysílací právo opět na první zařízení a celý cyklus se opakuje.

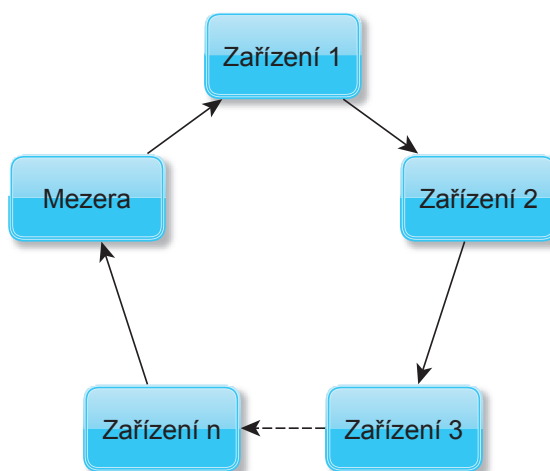
Maximální počet komunikujících zařízení je dán délkou paketu, délkou audia v paketu a přenosovou rychlostí:

$$n_{max} < t_A \times \frac{r_{bps}}{l_p} = 40 \text{ ms} \times \frac{100 \text{ kbps}}{400 \text{ b}} = 10 \quad (4)$$

kde

- t_A je délka audia v jednom paketu = 40 ms.
- r_{bps} je přenosová rychlost = 100 kbps.
- l_p je délka paketu v bitech = $50 \times 8 = 400 \text{ b}$.

Z implementačních důvodů byl maximální počet zařízení zaokrouhlen na 8.



Obrázek 9: Token ring

Protože datový tok každého zařízení je konstantní, musí být konstantní i perioda komunikačního cyklu, nezávisle na počtu zúčastněných zařízení. Perioda je dorovnáвана délkou mezery na konci cyklu.

6.2 Identifikace zařízení

Aby bylo možné určit, od kterého zařízení pochází právě přijatý rámec a pořadí zařízení při vysílání, musí mít každé zařízení unikátní identifikátor. Kvůli jednoduchosti je použito pořadí, v jakém se zařízení zapojilo do komunikace. Než se nové zařízení zapojí do komunikace, musí proběhnout kompletní cyklus komunikace všech zařízení, aby bylo možno určit jejich počet. Tomuto počtu poté odpovídá identifikátor nového zařízení. První zapnuté zařízení tudíž má identifikátor 0.

Zařízení si díky tomu také udržují přehled o aktuálním počtu zúčastněných členů. To je důležité především pro první zařízení (s identifikátorem 0), které musí vědět, kdy začít nový cyklus.

Identifikátor je posílán v nejnižších třech bitech hlavičky paketu.

Když nějaké zařízení ukončí komunikaci, ostatní zařízení s vyšším identifikátorem si jej o jedno zmenší. Tím se zabrání vytváření prázdných mezer uprostřed komunikačního cyklu.

Ukončení komunikace jednoho člena není ostatním nijak oznamováno. Pokud se v daném počtu některý identifikátor neobjeví, je jeho vlastník považován za neaktivního a je z komunikace vyloučen. Přijetí poškozeného paketu není považováno za absenci daného identifikátoru.

6.3 Skupiny zařízení

Aby nedocházelo ke zbytečnému plýtvání šířkou pásma, může na jedné frekvenci komunikovat více zařízení „nezávisle“ na sobě. To je umožněno díky nastavení skupiny v hlavičce paketu. Jedná se o dva bity, následující identifikátor zařízení. Princip je takový, že zařízení ignoruje všechny přijaté pakety, které mají v hlavičce jinou skupinu, než je na zařízení nastavena.

Token se samozřejmě předává přes všechna zařízení na stejné frekvenci, nezávisle na skupině.

Délka identifikátoru skupiny je dva bity, tudíž mohou existovat až 4 skupiny. Větší počet není třeba, protože na jedné frekvenci může komunikovat maximálně 8 zařízení. A aby se dalo hovořit o komunikaci, musí se toho procesu účastnit minimálně dvě zařízení.



Obrázek 10: Hlavička paketu

- DID[0-2] – Device IDentifier – identifikátor zařízení v rozsahu $0 \div 7$
- GID[0-1] – Group IDentifier – identifikátor skupiny v rozsahu $0 \div 3$

7 RF MODUL

Jak je uvedeno v kapitole 2.4, byl vybrán modul RFM22B. Kromě výše uvedených důvodů také pro svou značnou flexibilitu.

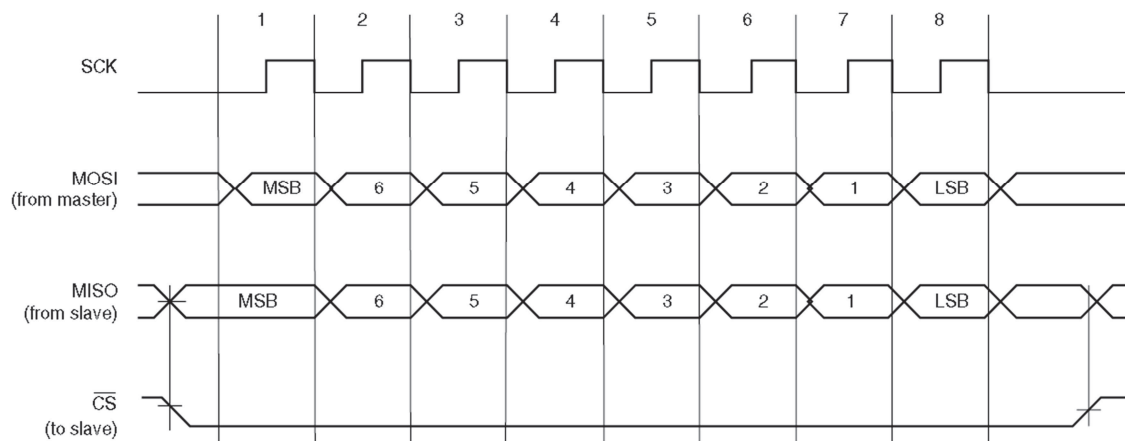
7.1 Komunikace s modulem

Komunikace s modulem probíhá pomocí SPI¹³ rozhraní. Jedná se o plně duplexní synchronní sériové rozhraní. Komunikaci řídí mikroprocesor (Master). RF modul je vždy Slave. Sběrnice sestává ze čtyř vodičů:

- MOSI – Master Output Slave Input – data jdoucí z mikroprocesoru do modulu
- MISO – Master Input Slave Output – data jdoucí z modulu do mikroprocesoru
- SCK – hodiny (generované Masterem)
- \overline{CS} – Chip Select – pomocí log. 0 na tomto vodiči vybírá Master, se kterým zařízením typu Slave chce komunikovat (každý Slave má svůj vlastní chip select, díky čemuž je počet zařízení na sběrnici prakticky neomezen)

Princip komunikace je jednoduchý – každých 8 taktů hodinového signálu si řídící mikroprocesor vymění s vybraným zařízením typu Slave jeden bajt.

Existují 4 režimy, lišící se polaritou hodinového signálu a pořadím hrany hodinového signálu, na které se vzorkují datové signály. RF modul používá režim 0.



Obrázek 11: SPI režim 0

Kromě SPI je z modulu do mikroprocesoru přiveden ještě signál \overline{IRQ} . Nastavením log. 0 na tomto vodiči signalizuje modul mikroprocesoru, že došlo k nějaké události, kterou by měl mikroprocesor obsloužit.

¹³ SPI – Serial Peripheral Interface – sériové rozhraní pro připojení periférií

Komunikace s modulem probíhá pomocí osmibitových registrů. Modul obsahuje celkem 128 těchto registrů [4].

Každá transakce sestává vždy minimálně z 16 bitů. První poslaný bit, \bar{R}/W , určuje, jestli se bude do modulu zapisovat (log. 1), nebo se z něj bude číst (log. 0). Následuje 7bitová adresa, nejvýznamnější bit je první. Následuje minimálně jeden bajt dat. Směr určuje dříve poslaný bit \bar{R}/W . Data v opačném směru jsou ignorována.

Je-li komunikace delší než 16 bitů, je adresa registru pro každý následující bajt o jedno zvětšena. Díky tomu je možno jedním přenosem zapsat do / přečíst z několika po sobě jdoucích registrů. Výjimkou je adresa 0x7F, která slouží pro komunikaci s interní pamětí typu FIFO¹⁴.

7.2 Nastavení modulu

Modul disponuje značně širokými možnostmi nastavení. Bohužel je k němu velice špatná dokumentace. To práci s modulem značně komplikuje a výrazně to omezuje jeho možnosti. Výrobce k modulu poskytuje dokument v MS Excel [22]. V tom je možné nastavit požadované parametry přenosu a nechat si vypočítat nastavení registrů. Tato metoda je sice vhodná pro první experimenty s modulem, ale značně omezuje možnosti jeho průběžného přizpůsobování aktuálním požadavkům aplikace. Postup, který dokument používá k získání požadovaných hodnot, je bohužel skryt a nedá se jednoduše extrahovat.

Bylo zjištěno, že integrovaný obvod Si4432 firmy Silicon Labs, je totožný s integrovaným modulem RF22B, použitým v modulu RFM22B. k tomuto obvodu je navíc k dispozici podstatně kvalitnější dokumentace (datasheet [23], Register descriptions [24] a Progrmmers guide [25]).

Bohužel ne všechna nastavení registrů, určená podle dokumentace, se shodovala s nastaveními určenými podle MS Excel. Nastavení z MS Excel se jako jediné ukázalo být funkční.

V následujících kapitolách je použité nastavení podrobně rozebráno.

¹⁴ FIFO – First In First Out – Používá se jako vyrovnávací paměť

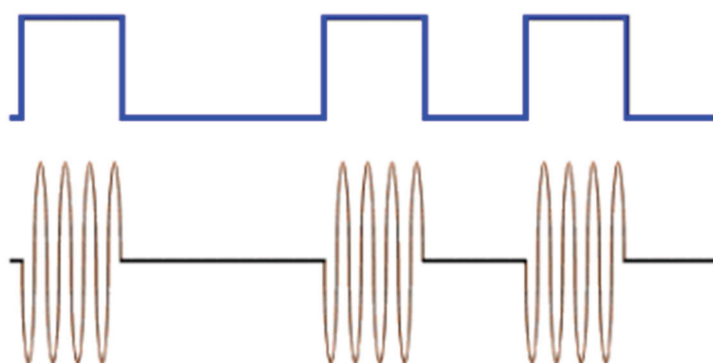
7.2.1 RF přenos

Z hlediska RF komunikace jsou důležité především 2 parametry: nosná frekvence a typ modulace užitečného signálu na nosnou vlnu.

Základní typy modulace jsou amplitudová a frekvenční. Třetí, méně často používaný typ, je fázová modulace.

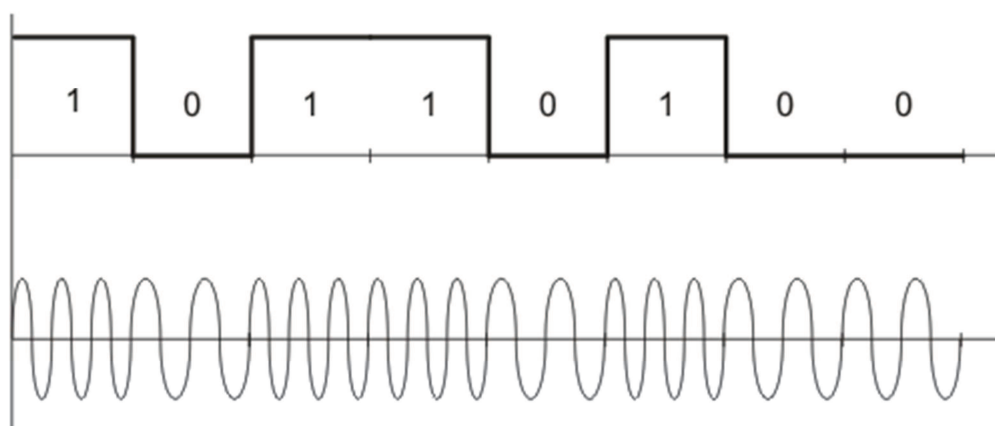
Amplitudová modulace je historicky starší. V dnešní době se více využívá frekvenční modulace. Výhodou amplitudové modulace je nižší energetická náročnost a zabraná šířka pásma na kanál. Nevýhodou je vyšší náchylnost na rušení oproti frekvenční modulaci.

Použitý modul disponuje možností amplitudové modulace. Jedná se o nejjednodušší metodu pro přenos digitální informace, OOK – On / Off Keying. Je znázorněna na Obrázek 12. Jediným parametrem u tohoto typu modulace je frekvence nosné vlny.

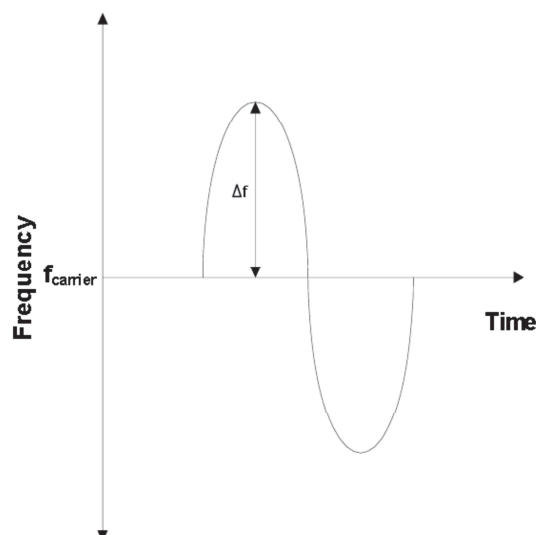


Obrázek 12: OOK modulace [26]

Převážně kvůli vyšší odolnosti vůči rušení byla zvolena frekvenční modulace. U frekvenční modulace je kromě frekvence nosné vlny ještě důležitá odchylka frekvence Δf . Ta udává, o kolik se změní frekvence nosné vlny pro přenos log. 1 a log. 0.



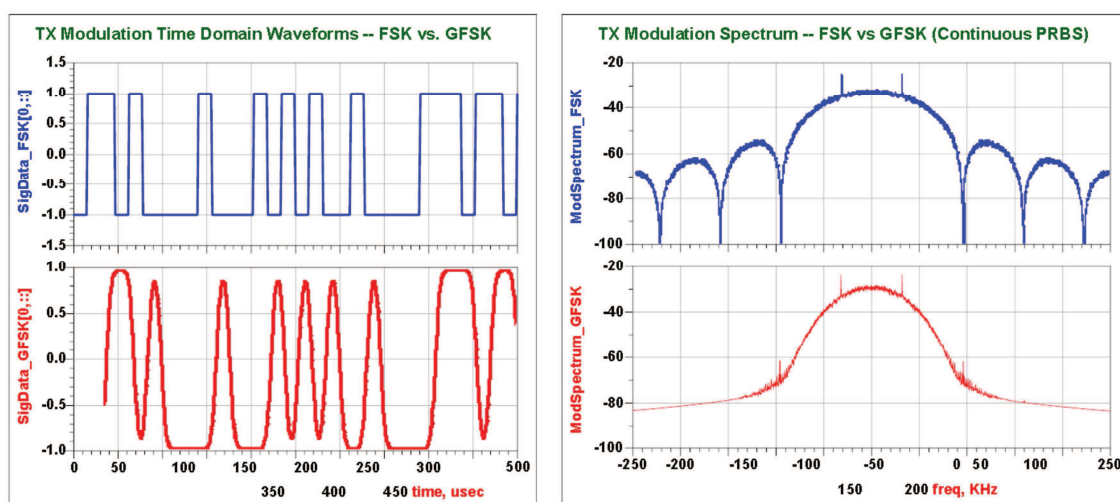
Obrázek 13: FSK modulace



Obrázek 14: Odchylka frekvence u frekvenční modulace [4]

Minimální velikost odchylky frekvence je závislá na frekvenci modulačního signálu. Jinak řečeno na rychlosti přenosu dat. Pro optimální funkci musí být frekvenční odchylka rovna, nebo větší, než přenosová rychlost (100 kbps), tudíž 50 kHz.

Modul umožňuje výběr mezi dvěma variantami frekvenční modulace: FSK (Frequency Shift Keying) a GFSK (Gaussian Frequency Shift Keying). Rozdíl mezi nimi je v plynulosti změny frekvence. Zatímco u FSK je změna skoková, u GFSK dochází k přechodu od $f_c + \Delta f$ k $f_c - \Delta f$ (nebo naopak) plynule. Rozdíl je patrný na následujícím obrázku. Zvolena byla modulace GFSK.



Obrázek 15: FSK vs. GFSK [4]

Nastavení nosné frekvence probíhá prakticky ve dvou krocích. V prvním se nastaví základní frekvence, označená jako kanál 0. Tato frekvence je prakticky dána impedančním přizpůsobením hardwaru. V tomto případě je to 866 MHz.

Dále se nastaví šířka kanálu. Ta by měla být minimálně čtyřnásobek odchylky frekvence Δf , aby nedocházelo k přeslechům mezi sousedními kanály. V tomto případě tedy 200 kHz.

Poté stačí již nastavit jen číslo kanálu a modul si sám nastaví správnou nosnou frekvenci. To umožňuje jednoduše a rychle přepínat mezi kanály.

Na nastavení nosné frekvence se také podílí AFC - Automatic Frequency Control. Působením různých vnějších vlivů, například různá teplota komunikujících zařízení, dochází k rozcházení jejich frekvencí. AFC za běhu kalibruje frekvenci interního oscilátoru přijímače podle přijímané frekvence. Tím se výrazně zlepšuje kvalita přenosu.

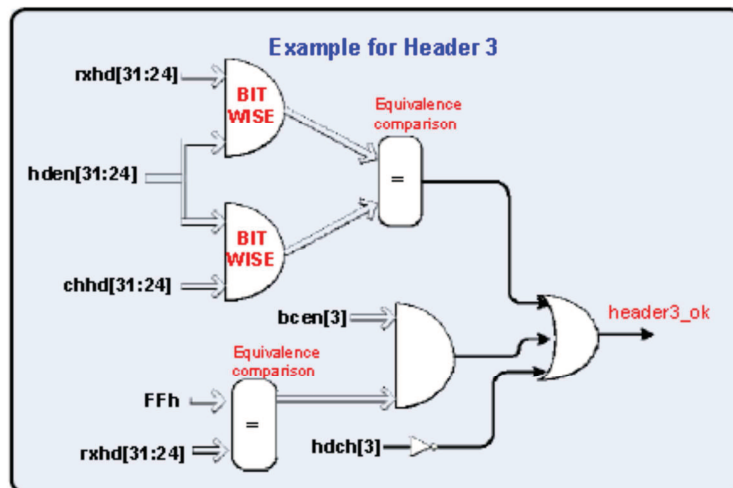
7.2.2 Správce paketů

To, že se RF modul sám stará vytváření a zpracování přijatých paketů, výrazně snižuje zatížení řídicího mikroprocesoru, především na straně přijímače. Mikroprocesor musí pouze zahájit příjem, počkat na signál z RF modulu, že data byla v pořádku přijata a vyčíst je.

Struktura paketu je znázorněna na Obrázek 8. Zvolené délky jednotlivých polí jsou uvedeny v **Chyba! Nenalezen zdroj odkazů.** Význam jednotlivých polí je uveden zde:

- Preamble – jedná se o sekvenci střídajících se jedniček a nul, indikující začátek vysílání. Tato sekvence je dobře rozpoznatelná, i když není zachycena od začátku. Během ní provádí AFC korekce frekvence. Byla zvolena preamble délky 32 bitů (doporučeno výrobcem).
- Sync Word – 1 až 4 bajty, následující za preamble. Musí od ní být dobře rozlišitelné. Sync Word udává začátek paketu. Jsou využity 2 bajty s hodnotou 0x2DD4 (výchozí hodnota).
- Header – uživatelsky nastavitelná hlavička paketu v délce 0 až 4 bajty. Je určena převážně pro přenos adresy zařízení. RF modul umí porovnávat přijatou hodnotu s přednastavenou a signalizovat příjem paketu pouze pokud se shodují. Vyhodnocování probíhá pro každý bajt zvlášť a je znázorněno na **Obrázek 16**. V této aplikaci je využit pouze jeden bajt hlavičky. Její kontrola se neprovádí přímo v modulu, místo toho ji zpracovává řídicí mikroprocesor.
- Length – délka datové části paketu v bajtech, v rozsahu 0 až 255. Pokud jsou vždy posílány pakety konstantní délky, může tato být nastavena v přijímači a vysílači a nemusí být součástí paketu. Kvůli zpětné kompatibilitě s případnými budoucími rozšířeními je délka posílána.
- Data
- CRC – nepovinný kontrolní součet, spočítaný buď z celého paketu, nebo jen z jeho datové části. Neodpovídá-li přijatá hodnota hodnotě vypočítané z přijatých dat, je řídicímu mikroprocesoru nahlášen příjem poškozeného paketu. K dispozici je několik možných algoritmů výpočtu [22]:

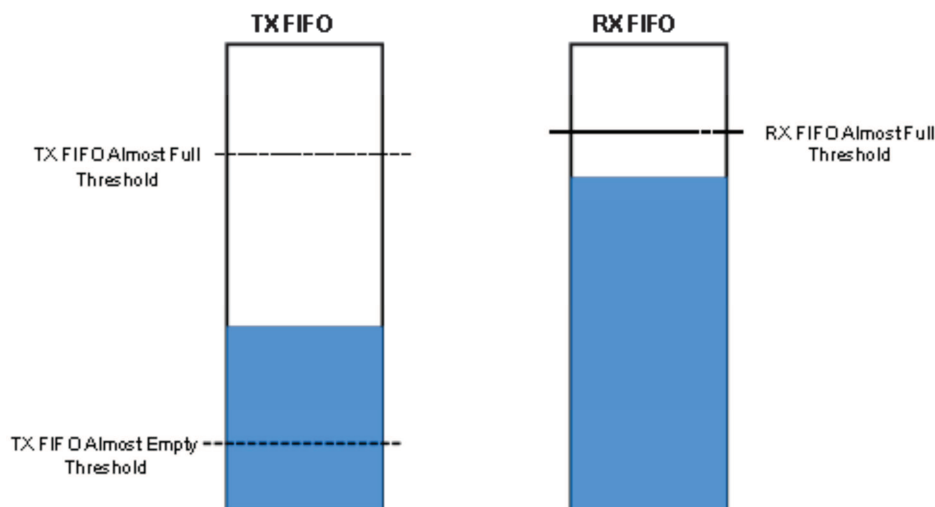
- CRC16-CCITT (použito v této aplikaci)
- CRC16-IBM
- IEC16 (pravděpodobně IEC 870-5-2 CRC)
- BIACHEVA



Obrázek 16: Vyhodnocení hlavičky paketu [4]

Se správou paketů úzce souvisí interní vyrovnávací paměť typu FIFO. RF modul obsahuje dvě vyrovnávací paměti, každá o velikosti 64 bajtů. Jedna je využívána pro příjem dat, druhá pro vysílání. Pro pakety s délkou nepřesahující velikost paměti je vhodné nahrát celý paket do paměti a teprve potom zahájit vysílání. Při příjmu je naopak celý paket uložen do paměti, odkud ho řídící mikroprocesor vyčte.

Pro delší pakety ovšem nutné průběžně doplňovat vysílaná data a vyčítat přijatá. Aby to bylo možné, umí tyto paměti generovat požadavek na obsluhu, pokud jsou téměř plné, nebo prázdné (pouze TX FIFO), viz Obrázek 17.

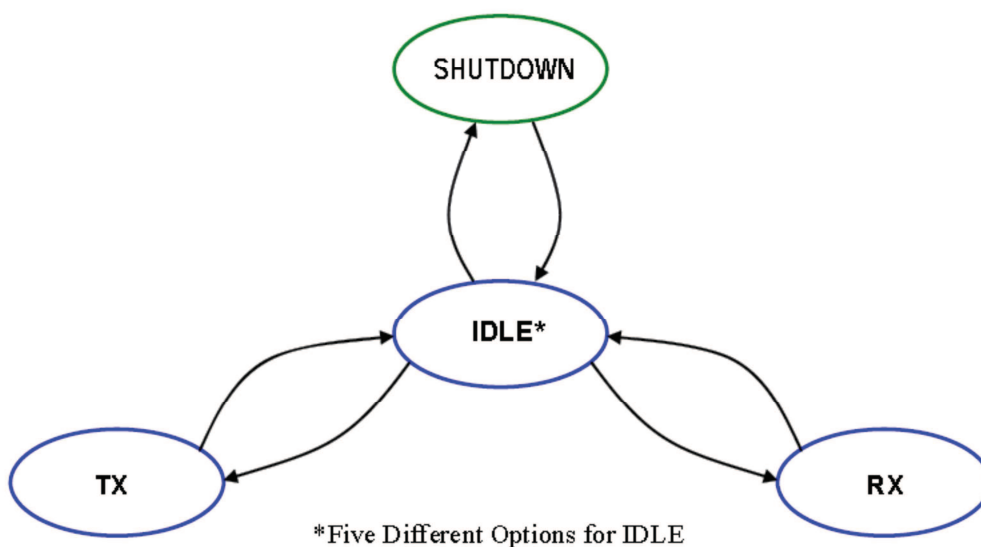


Obrázek 17: Vyrovnávací paměti [4]

7.3 Obsluha modulu

RF modul se může nacházet v jednom z následujících stavů (viz **Obrázek 18**):

- Shutdown – nejnižší spotřeba, ztráta nastavení modulu, ovládán hardwarově pinem modulu, nevyužit
- Idle
 - Standby – velmi nízká spotřeba, funkční pouze komunikační rozhraní
 - Sleep – velmi nízká spotřeba, kromě komunikačního rozhraní aktivní ještě časovač, umožňující periodicky probouzet celý systém
 - Sensor – nízká spotřeba, kromě dříve uvedeného je navíc aktivní interní AD převodník, umožňující například měření napájecího napětí (hlídání podvybití baterií), nebo teploty (interní teplotní čidlo)
 - Ready – aktivní hlavní oscilátor, umožňuje rychle přejít do přijímacího, nebo vysílacího módu
 - Tune – interní frekvenční násobička zůstává naladěná na nastavenou nosnou frekvenci, umožňuje nejrychlejší přechod mezi TX a RX módem za cenu vyšší spotřeby
- RX state – nastavením modulu do RX módu se aktivuje přijímač a čeká se na přijetí paketu (pokud je využit správce paketů), po přijetí paketu se modul vrací do nastaveného Idle módu
- TX state – nastavením modulu do TX módu se aktivuje vysílač a odešle se paket, čekající ve vyrovnávací paměti (pokud je využit správce paketů), po odvysílání paketu se modul vrací do nastaveného Idle módu



Obrázek 18: Módy RF modulu [23]

O korektní přepínání mezi módy (zapínání a vypínání nezbytných komponent, potřebných pro daný mód) se stará interní logika modulu.

Po startu aplikace je modul inicializován (komunikační rozhraní, RF přenos, správce paketů, povolení žádostí o obsluhu). Povoleny jsou následující žádosti o obsluhu:

- valid packet received – signalizuje příjem paketu, paket je vyčten z vyrovnávací paměti
- CRC error – signalizuje příjem poškozeného paketu, obsah vyrovnávací paměti je vymazán
- packet sent – paket byl odeslán, přepnutí modulu do RX módu

Modul je přepnut do RX módu a zařízení se snaží zapojit do komunikačního cyklu (viz kapitola 6). Pokud v daném časovém intervalu není zachycena žádná komunikace, považuje se zařízení za jediné v systému, přidělí si identifikátor 0 a začne nový komunikační cyklus odesláním paketu.

8 ZÁVĚR

Bylo navrženo, vyrobeno a otestováno zařízení pro obousměrný bezdrátový přenos digitalizované hlasové informace. Zařízení je postaveno na 32bitovém mikroprocesoru s jádrem ARM AT91SAM3U2C od firmy Atmel. Analogové vstupy a výstupy zařízení řeší audio kodek TLV320AIC3101 od firmy Texas Instruments. O bezdrátovou komunikaci se stará RF modul RFM22B-868-S1. Komunikace probíhá v ISM pásmu 866 MHz.

Kvůli maximálnímu využití frekvenčního spektra při rádiové komunikaci byl požadován minimální datový tok. Toho bylo dosaženo softwarovou komprimací signálu pomocí kodeku Speex. Výsledný datový tok, včetně všech náležitostí, nutných pro spolehlivou bezdrátovou komunikaci, je 10 kbps. V případě nutnosti by se dal datový tok ještě zmenšit přibližně na polovinu, ovšem za cenu nižší kvality přenášeného signálu.

Díky nízkému datovému toku a použití softwarové komprimace může vzájemně komunikovat až 8 zařízení.

V rámci dalšího pokračování projektu je potřeba navrhnout nový, menší hardware, řešící několik drobnějších nedostatků toho současného (například diverzifikaci antén). Dále je potřeba přidat správu napájení (baterie). Z hlediska softwaru je třeba se zaměřit na optimalizaci využití hardwarových prostředků a vylepšení bezdrátové komunikace (větší robustnost). Zvláštní pozornost je třeba věnovat spotřebě energie celého zařízení.

LITERATURA

- [1] J. STREIT.: Digitální audiořetězec z bezdrátovou komunikací [Online]. aktualizováno 2013
- [2] Anon.: WikiSkripta [Online]. aktualizováno 2014-3-3 [cit. 2014-5-14]. Dostupné z URL: http://www.wikiskripta.eu/index.php/Pr%C3%A1h_sluchu_a_sluchov%C3%A9_pole
- [3] Atmel.: SAM3U Series Complete [Online]. aktualizováno 2012-2-12 [cit. 2014-5-23]. Dostupné z URL: <http://www.atmel.com/Images/doc6430.pdf>
- [4] Hope Microelectronics.: RFM22B/23B ISM TRANSCEIVER [Online]. aktualizováno 2006 [cit. 2014-5-15]. Dostupné z URL: http://www.hoperf.com/upload/rf/RFM22B_23B.pdf
- [5] Atmel.: Atmel SAM-BA In-system Programmer [Online]. aktualizováno 2014 [cit. 2014-5-14]. Dostupné z URL: <http://www.atmel.com/tools/atmelsam-bain-systemprogrammer.aspx?tab=overview>
- [6] Vojtěch Boček.: Lorris Toolbox [Online]. aktualizováno 2014 [cit. 2014-5-25]. Dostupné z URL: <http://tassadar.github.io/Lorris/cz/index.html>
- [7] Texas Instruments.: LOW-POWER STEREO AUDIO CODEC FOR PORTABLE AUDIO/TELEPHONY [Online]. aktualizováno 2008-12 [cit. 2014-5-19]. Dostupné z URL: <http://www.ti.com/lit/ds/symlink/tlv320aic3101.pdf>
- [8] NXP.: I2C-bus specification and user manual [Online]. aktualizováno 2014-4-4 [cit. 2014-5-19]. Dostupné z URL: http://www.nxp.com/documents/user_manual/UM10204.pdf
- [9] Anon.: Time-division multiplexing [Online]. aktualizováno 2014-2-21 [cit. 2014-5-19]. Dostupné z URL: http://en.wikipedia.org/wiki/Time-division_multiplexing
- [10] Anon.: Wikipedie [Online]. aktualizováno 2014-4-11 [cit. 2014-5-19]. Dostupné z URL: <http://cs.wikipedia.org/wiki/Hlas>
- [11] David.: maild.org [Online]. aktualizováno 2014-1-31 [cit. 2014-5-19]. Dostupné z URL: <http://maild.org/wordpress/wp-content/uploads/2014/01/4conductorplug4ap-235x300.jpg>
- [12] pro-signal.: Electret Condenser Microphone [Online]. aktualizováno 2011-3-15 [cit. 2014-5-19]. Dostupné z URL: <http://www.farnell.com/datasheets/1505849.pdf>

- [13] ROWETEL.: Codec 2 [Online]. aktualizováno 2014 [cit. 2014-5-21].
Dostupné z URL: <http://www.rowetel.com/blog/?page_id=452>
- [14] Anon.: Opus [Online]. aktualizováno 2013-12-5 [cit. 2014-5-21]. Dostupné
z URL: <<http://www.opus-codec.org/>>
- [15] Jean-Marc Valin.: Speex [Online]. aktualizováno 2007-5-23 [cit. 2014-5-
21]. Dostupné z URL: <<http://www.speex.org/docs/manual/speex-manual/node9.html>>
- [16] STMicroelectronics.: Vocoder demonstration using a Speex audio codec on
STM32F101xx and STM32F103xx microcontrollers [Online].
aktualizováno 2008-10 [cit. 2014-5-21]. Dostupné z URL:
<http://www.st.com/st-web-ui/static/active/jp/resource/technical/document/application_note/CD00204907.pdf>
- [17] Mark Borgerding.: sourceforge [Online]. aktualizováno 2013-6-20 [cit.
2014-5-21]. Dostupné z URL: <<http://sourceforge.net/projects/kissfft/>>
- [18] The Open Group.: The Single UNIX ® Specification [Online].
aktualizováno 1997 [cit. 2014-5-21]. Dostupné z URL:
<<http://pubs.opengroup.org/onlinepubs/7908799/xsh/brk.html>>
- [19] The Open Group.: The Single UNIX ® Specification [Online].
aktualizováno 1997 [cit. 2014-5-21]. Dostupné z URL:
<http://pubs.opengroup.org/onlinepubs/7908799/xsh/_exit.html>
- [20] Anon.: Speex Documentation [Online]. aktualizováno 2007-5-23 [cit.
2014-5-21]. Dostupné z URL: <<http://www.speex.org/docs/api/speex-api-reference/index.html>>
- [21] Jean-Marc Valin.: Speex [Online]. aktualizováno 2007-12-8 [cit. 2014-5-
22]. Dostupné z URL: <<http://www.speex.org/docs/manual/speex-manual.pdf>>
- [22] HopeRF.: RF22B 23B 31B 42B 43B Register Settings_RevB1-v5 [Online].
aktualizováno 2011-5-3 [cit. 2014-5-24]. Dostupné z URL:
<http://www.hoperf.com/upload/rf/RF22B%2023B%2031B%2042B%2043B%20Register%20Settings_RevB1-v5.xls>
- [23] Silicon Laboratories.: Si4430/31/32 ISM TRANSCEIVER [Online].
aktualizováno 2010-10 [cit. 2014-5-24]. Dostupné z URL:
<<http://www.silabs.com/Support%20Documents/TechnicalDocs/Si4430-31-32.pdf>>
- [24] Silicon Laboratories.: AN440 [Online]. aktualizováno 2013-7 [cit. 2014-5-
24]. Dostupné z URL:
<<http://www.silabs.com/Support%20Documents/TechnicalDocs/AN440.pdf>>

- [25] Silicon Laboratories.: AN415 [Online]. aktualizováno 2010-7 [cit. 2014-5-24]. Dostupné z URL:
<<http://www.silabs.com/Support%20Documents/TechnicalDocs/AN415.pdf>>
- [26] Kaspars.: All About Circuits [Online]. aktualizováno 2011-6-01 [cit. 2014-5-24]. Dostupné z URL:
<http://forum.allaboutcircuits.com/cache.php?url=http://www.technologiyuk.net/telecommunications/telecom_principles/images/am_on_off_keying.gif>
- [27] Texas Instruments.: Autonomous Audio Headset Switch with Reduced GND Switch RON and FM Capability [Online]. aktualizováno 2013-7 [cit. 2014-5-19]. Dostupné z URL:
<<http://www.ti.com/lit/ds/symlink/ts3a226ae.pdf>>
- [28] Anon.: Wikipedia [Online]. aktualizováno 2014-5-19 [cit. 2014-5-24]. Dostupné z URL:
<http://en.wikipedia.org/wiki/Cyclic_redundancy_check>
- [29] Anon.: Wikipedie [Online]. aktualizováno 2013-3-10 [cit. 2014-5-22]. Dostupné z URL: <<http://cs.wikipedia.org/wiki/FLOPS>>
- [30] Anon.: Wikipedie [Online]. aktualizováno 2013-4-4 [cit. 2014-5-14]. Dostupné z URL: <http://cs.wikipedia.org/wiki/P%C3%A1smo_ISM>

SEZNAM OBRÁZKŮ

Obrázek 1: Vývojový diagram inicializace SAM-BA monitoru [3]	11
Obrázek 2: Příklad Xmodem protokolu [3]	13
Obrázek 3: I ² C přenos [3]	18
Obrázek 4: Komunikace s audio kodekem pomocí I ² C [7]	19
Obrázek 5: I ² S komunikace [7]	20
Obrázek 6: Nastavení hodin audio kodeku [7]	22
Obrázek 7: Headset connector [11]	23
Obrázek 8: Struktura datového paketu [4]	33
Obrázek 9: Token ring	35
Obrázek 10: Hlavička paketu	36
Obrázek 11: SPI režim 0	37
Obrázek 12: OOK modulace [26]	39
Obrázek 13: FSK modulace	39
Obrázek 14: Odchylka frekvence u frekvenční modulace [4]	40
Obrázek 15: FSK vs. GFSK [4]	40
Obrázek 16: Vyhodnocení hlavičky paketu [4]	42
Obrázek 17: Vyrovnávací paměti [4]	42
Obrázek 18: Módy RF modulu [23]	43

SEZNAM TABULEK

Tabulka 1: Příkazy SAM-BA monitoru [3].....	12
Tabulka 2: Nastavení appletu	15
Tabulka 3: příkazy appletu	15
Tabulka 4: Datový tok a výpočetní náročnost enkodéru vs. nastavená kvalita [21]	31
Tabulka 5: Velikost jednotlivých částí paketu	34

SEZNAM ZKRATEK

- CS – Chip Select
- IRQ – Interrupt Request
- ACK – Acknowledge (ASCII character, or I²C bit)
- AD – analogově-digitální (převodník)
- AFC – Automatic Frequency Control
- AGC – Automatic Gain Control
- API – Application Programming Interface
- ARM – Advanced RISC Machine
- ASCII – American Standard Code for Information Interchange
- AVDD – Analogové napájecí napětí
- b – bit
- B – bajt
- BCLK – bit clock
- bps – bits per second
- CD – Compact Disc
- CELP – Code-Excited Linear Prediction
- CR – Carriage return (ASCII character)
- CRC – Cyclic Redundancy Check
- ČTÚ – Český telekomunikační úřad
- DA – digitálně-analogový (převodník)
- DID – Device Identifier
- DPS – Deska plošných spojů
- DRA – Double Rate
- DTX – Discontinuous Transmission
- EEFC – Enhance Embedded Flash Controller
- EOT – End Of Transmission (ASCII character)
- FFT – Fast Fourier Transformation
- FIFO – First In First Out
- FLOPS – Floating-point Operations per Second
- FRDY – Flash Ready
- FSK – Frequency Shift Keying
- GFSK – Gaussian Frequency Shift Keying
- GID – Group Identifier
- GND – Ground
- GPNVM – General Purpose Non-Volatile Memory
- HPLCOM – High-power Left Common output
- HPRCOM – High-power Right Common output
- I²C – Inter-Integrated Circuit

- I²S – Integrated Interchip Sound
- ISM – Industrial, Scientific and Medical
- kB – kilobyte (1024 B)
- kbps – kilobits per second (1000 bps)
- LF – Line Feed (ASCII character)
- MCK – Master Clock (microprocessor)
- MCLK – Master Clock (audio kodek)
- MFLOPS – MegaFLOPS
- MIC – microphone
- MISO – Master Input Slave Output
- MOSI – Master Output Slave Input
- MS – Microsoft
- NACK – Negative Acknowledge (I²C bit)
- NAK – Negative Acknowledge (ASCII character)
- OOK – On-OFF Keying
- PC – Personal Computer
- PLL – Phase-Locked Loop
- PWM – Pulse Weight Modulation
- RAM – Random Access Memory
- RC – Resistor-Capacitor
- RD – Receive Data
- RF – Radio Frekvency
- ROM – Read-Only Memory
- RISC – Reduced Instruction Set Computing
- RX – Receiver
- SAM-BA – SAM Boot Assistant
- SCK – Serial Clock (SPI)
- SCL – Serial Clock (I²C)
- SDA – Serial Data (I²C)
- SPI – Serial Peripheral Interface
- TD – Transmit Data
- TX - Transmitter
- UART – Universal Asynchronous Receiver/Transmitter
- USB – Universal Serial Bus
- WCLK – Word Clock

SEZNAM PŘÍLOH

A	Audio kodek api – referenční manuál	I
A.1	Pin	I
A.2	I ² C	I
A.3	I ² S	II
A.4	Audio kodek	II
A.5	Ovládání hlasitosti	XII
B	Schéma zapojení	XIV
C	Seznam součástek	XIX
D	DPS	XXI
E	Obsah CD	XXIII

A AUDIO KODEK API – REFERENČNÍ MANUÁL

Knihovna pro práci s audio kodekem TLV320AIC3101 je napsána tak, aby byla platformě nezávislá. Toho je dosaženo využitím několika tříd, poskytujících jednotné rozhraní k hardwaru. Jsou to třídy `pin_t`, `i2c_t` a `i2s_t`.

A.1 Pin

Třída `pin_t` umožňuje nezávislé ovládání jednotlivých pinů mikroprocesoru. Musí na ní být definovány následující datové typy:

- `enum peripheral_select { PA, PB };`

A následující metody:

- `void make_peripheral(const peripheral_select& peripheral)`
Nastaví, která z periférií mikroprocesoru je k pinu připojena. Využívá `i2c_t` a `i2s_t`.
- `void make_low()`
Nastaví pin jako výstupní s hodnotou log. 0.
- `void set_low()`
Nastaví na pinu log. 0.
- `void set_high()`
Nastaví na pinu log. 1.

A.2 I²C

Komunikace s audio kodekem probíhá přes I²C linku. Tu zapouzdřuje třída `i2c_t`, na které musí být definovány následující metody:

- `bool write(const uint8_t& dev_addr, const uint8_t& reg_addr, const uint8_t& data)`
- `bool write(const uint8_t& dev_addr, const uint8_t& reg_addr, const uint16_t& data)`
Zapíše data do registru.

Parametry:

- `dev_addr` – fyzická adresa zařízení na sběrnici
- `reg_addr` – adresa registru, do kterého se má zapsat
- `data` – data, která se mají do registru zapsat

Návratová hodnota: `true` pokud funkce uspěla, jinak `false`.

- `int read(const uint8_t& dev_addr, const uint8_t& reg_addr, const uint8_t&)`
- `int read(const uint8_t& dev_addr, const uint8_t& reg_addr, const uint16_t&)`
Přečte data z registru.

Parametry:

- `dev_addr` – fyzická adresa zařízení na sběrnici
- `reg_addr` – adresa registru, ze kterého se má číst

- Prázdný parametr, sloužící pouze pro určení velikosti čtených dat

Návratová hodnota: Přečtená hodnota pokud funkce uspěla, jinak -1.

Vzhledem k tomu, že komunikace po I²C sběrnici se nemusí vždy podařit, je vhodné, aby funkce měly interní timeout a nedocházelo k zamrzání aplikace.

A.3 I²S

Třída `i2s_t` zajišťuje rozhraní pro přenos digitálního audio signálu mezi kodekem a mikroprocesorem. Musí na ní být definovány následující metody:

- `bool empty()`
Vrátí `true`, pokud nejsou k dispozici nová data z kodeku, jinak `false`.
- `uint32_t read()`
Vrátí naposled přijatá data z kodeku.
- `void write(const uint32_t& data)`
Zapíše data do kodeku.

A.4 Audio kodek

Audio kodek je reprezentován třídou `audio_codec_t`. Na ní jsou definované následující datové typy:

- `enum sample_rate_t`
 - `F48kHz` = 0
 - `F32kHz` = 1
 - `F24kHz` = 2
 - `F19200Hz` = 3
 - `F16kHz` = 4
 - `F12kHz` = 6
 - `F10667Hz` = 7
 - `F9600Hz` = 8
 - `F8kHz` = 10
 - `F96kHz` = 16 + 0
 - `F64kHz` = 16 + 1
 - `F38400Hz` = 16 + 3

Nastavení vzorkovací frekvence převodníků (pouze pokud $f_{S(\text{ref})}$ je 48 kHz).

- `enum mic_bias_t`
 - `MICBIAS_DISABLED` = 0
 - `MICBIAS_2V` = 1
 - `MICBIAS_2V5` = 2
 - `MICBIAS_AVDD` = 3
- `enum stereo_output_config_t`
 - `SINGE_ENDED` = 0
 - `PSEUDO_DIFF` = 1
 - `FULLY_DIFF` = 2
- `enum fsref_t`
 - `FSref48kHz` = 0
 - `FSref44_1kHz` = 1
- `enum audio_interface_mode_t`

- I2S = 0
- DSP = 1
- LEFT_JUSTIFIED = 2
- RIGHT_JUSTIFIED = 3
- enum audio_interface_word_length_t
 - W16BITS = 0
 - W20BITS = 1
 - W24BITS = 2
 - W32BITS = 3

U většiny metod jejich název, případně jména a typy parametrů, dostatečně popisují jejich funkce. Proto je zde, až na výjimky, uveden pouze seznam metod.

Nastavovací metody vrací `true`, pokud uspějí, v opačném případě `false` (viz A.2 I2C). Vyčítající metody vrací hodnoty ve stejném rozsahu, jako korespondující nastavovací metody. Výjimkou je, pouze pokud selžou. V tom případě vrací `-1`.

Definované metody:

- `audio_codec_t (i2c_t &contorl_interface, i2s_t data_interface, pin_t rst_pin)`
Konstruktor.
- `bool init (const sample_rate_t& sample_rate = F48kHz, const fsref_t f& fsref = FSref48kHz, const audio_interface_word_length_t& word_len = W16BITS, const clock_t::clk_in_source_t& clock_in = clock_t::MCLK, const bool& bclk_master = true, const bool& wclk_master = true, const uint8_t& pll_J = 8, const uint16_t& pll_D = 1920, const uint8_t& pll_R = 1, const uint8_t& pll_P = 1, const high_power_output_stage_t::common_mode_voltage_t& common_mode_voltage = high_power_output_stage_t::CMV_1V65, const bool& left_analog_mic = true, const bool& right_analog_mic = true)`
- `bool set_sample_rate (const sample_rate_t& sample_rate)`
- `int get_sample_rate ()`
- `bool set_FSref (const fsref_t& fsref)`
- `int get_FSref ()`
- `bool enable_3D_effect (const bool& enable = true)`
- `bool disable_3D_effect ()`
- `int is_3D_effect_enabled ()`
- `bool set_mic_bias (const mic_bias_t& mic_bias)`
- `int get_mic_bias ()`
- `int is_headset_detected ()`
- `int was_headset_status_changed ()`
- `bool mute_while_resync (const bool& en = true)`
- `bool dont_mute_while_resync ()`
- `int is_muted_while_resync ()`
- `bool set_capless_driver ()`
- `bool set_ac_coupled_driver ()`

- `int is_ac_coupled_driver ()`
- `bool set_stereo_output_configuration (const stereo_output_config_t& config)`
- `int get_stereo_output_configuration ()`
- `bool use_analog_microphone (const bool& left = true, const bool& right = true)`
- `bool use_digital_microphone (const bool &left = true, const bool& right = true)`
- `int is_left_microphone_analog ()`
- `int is_right_microphone_analog ()`
- `bool enable_i2c_error_detection (const bool& en = true)`
- `bool disable_i2c_error_detection ()`
- `int is_i2c_error_detection_enabled ()`
- `int i2c_error_occured ()`
- `bool set_audio_interface_mode (const audio_interface_mode_t& mode)`
- `int get_audio_interface_mode ()`
- `bool set_audio_interface_word_length (const audio_interface_word_length_t& mode)`
- `int get_audio_interface_word_length ()`
- `bool software_reset ()`
- `void shutdown ()`
- `void print_settings ()`
- `int read_ADC_flags ()`
- `i2s_t& data_interface ()`
- `bool empty () const`
Viz A.3 I2S
- `uint32_t read () const`
Viz A.3 I2S
- `void write (const uint32_t& data)`
Viz A.3 I2S

Kromě těchto metod jsou definovány následující objekty:

- `clock`
Nastavení hodin kodeku.
 - `enum clk_in_source_t`
 - `MCLK = 0`
 - `GPII2 = 1`
 - `BCLK = 2`
 - `enum clk_in_selectin_t`
 - `PLLDIV_OUT = 0`
 - `CLKDIV_OUT = 1`
 - `bool set_clk_div_source(const clk_in_source_t& source)`
 - `int get_clk_div_source()`
 - `bool set_clk_pll_source(const clk_in_source_t& source)`
 - `int get_clk_pll_source()`
 - `bool set_codec_clock_source(const clk_in_selectin_t& source)`
 - `int get_codec_clock_source()`
 - `bool enable_pll(const bool& en = true)`
 - `bool disable_pll()`
 - `int is_pll_enabled()`
 - `bool set_div_Q(const uint8_t& value)`

- int get_div_Q()
- bool set_pll_P(const uint8_t& value)
- int get_pll_P()
- bool set_pll_R(uint8_t value)
- int get_pll_R()
- bool set_pll_J(const uint8_t& value)
- bool set_pll_D(const uint16_t& value)
- int get_pll_D()
- bool set_pll(const uint8_t& J, const uint16_t& D, const uint8_t& R, const uint8_t& P, const bool& enable = true)
- bool set_bclk_slave_mode()
- bool set_bclk_master_mode()
- int is_bclk_master()
- bool set_wclk_slave_mode()
- bool set_wclk_master_mode()
- int is_wclk_master()
- bool enable_3_state_dout(const bool& en = true)
- bool disable_3_state_dout()
- int is_3_state_dout_enabled()
- bool enable_continuous_clock_output(const bool& en = true)
- bool disable_continuous_clock_output()
- int is_continuous_clock_output_enabled()
- bool enable_256_clock_transfer_mode(const bool& en = true)
- bool disable_256_clock_transfer_mode()
- int is_256_clock_transfer_mode_enabled()
- bool set_data_offset(const uint8_t& value)
- int get_data_offset()
- left_adc
- right_adc

Nastavení AD převodníků.

- enum high_pass_filter_settings_t
 - DISABLED = 0
 - HPF0045FS = 1
 - HPF0125FS = 2
 - HPF025FS = 3

Nastavení filtru typu horní propust. Možnosti jsou: vypnuto, $0,0045 \times f_{S(\text{ref})}$, $0,0125 \times f_{S(\text{ref})}$ a $0,025 \times f_{S(\text{ref})}$.

- volume_t – viz A.5 Ovládání hlasitosti
- bool power_on(const bool& on = true)
- bool power_off()
- int is_powered()
- int was_overflow()
- bool enable_resync(const bool& en = true)
- bool disable_resync()
- int is_resync_enabled()
- bool use_high_pass_filter_default_coeffs()
- bool use_high_pass_filter_programmable_coeffs()
- int using_high_pass_filter_default_coeffs()
- bool set_high_pass_filter(const high_pass_filter_settings_t& settings)
- int get_high_pass_filter_settings()
- bool use_programmable_filters(const bool& use = true)
- int using_programmable_filters()

- line2_L_to_left_adc
line2_R_to_left_adc
line2_L_to_right_adc
line2_R_to_right_adc

Nastavení propojení mezi vstupy a AD převodníky.

- bool set_volume(const uint8_t& value)

Hodnoty v rozsahu 0 ÷ 8. Výsledné zesílení:

$$A = -1,5 \times value [dB] \quad (1)$$

Je-li zadána větší hodnota, je vstup odpojen.

- int get_volume()
- bool mute(const bool& mute = true)
- bool unmute()
- int is_muted()
- line1_LP_to_left_adc
line1_RP_to_left_adc
line1_RP_to_right_adc
line1_LP_to_right_adc

Nastavení propojení mezi diferenciálními vstupy a AD převodníky. Disponují stejným nastavením hlasitosti, jako ostatní vstupy (viz předchozí bod). Navíc obsahují následující metody:

- bool set_single_ended()
- bool set_differential()
- int is_differential()

- left_agc
right_agc

Nastavení automatické kontroly hlasitosti.

- enum target_level_t
 - A5_5dB = 0
 - A8dB = 1
 - A10dB = 2
 - A12dB = 3
 - A14dB = 4
 - A17dB = 5
 - A20dB = 6
 - A24dB = 7
- enum attack_time_t
 - At8ms = 0
 - At11ms = 1
 - At16ms = 2
 - At20ms = 3
- enum new_attack_time_t
 - NAT7ms = 0
 - NAT8ms = 1
 - NAT10ms = 2
 - NAT11ms = 3
- enum decay_time_t
 - Dt100ms = 0
 - Dt200ms = 1
 - Dt400ms = 2

- Dt500ms = 3
- o enum new_decay_time_t
 - NDt50ms = 0
 - NDt150ms = 1
 - NDt250ms = 2
 - NDt350ms = 3
- o enum noise_gate_hysteresis_t
 - H1dB = 0
 - H2dB = 1
 - H3dB = 2
 - DISABLED = 3
- o bool enable(const bool& en = true)
- o bool disable()
- o int is_enabled()
- o bool set_target_level(const target_level_t& level)
- o int get_target_level()
- o bool set_attack_time(const attack_time_t& time)
- o int get_attack_time()
- o bool set_decay_time(const decay_time_t& time)
- o int get_decay_time()
- o bool set_max_gain(const uint8_t& gain)
- o int get_max_gain()
- o bool set_noise_gate_hysteresis(const noise_gate_hysteresis_t& hysteresis)
- o int get_noise_gate_hysteresis()
- o bool set_noise_gate_threshold(const uint8_t& threshold)
- o int get_noise_gate_threshold()
- o bool enable_clip_stepping(const bool& en = true)
- o bool disable_clip_stepping()
- o int is_clip_stepping_enabled()
- o int get_applied_gain()
- o bool set_noise_detection_debounce(const uint8_t& debounce)
- o int get_noise_detection_debounce()
- o bool set_signal_detection_debounce(const uint8_t& debounce)
- o int get_signal_detection_debounce()
- o bool use_new_attack_time()
- o bool use_attack_time()
- o int using_new_attack_time()
- o bool set_new_attack_time(const new_attack_time_t& time)
- o int get_new_attack_time()
- o bool set_new_attack_time_multiplication(const uint8_t& time)
- o int get_new_attack_time_multiplication()
- o bool use_new_decay_time()
- o bool use_decay_time()
- o int using_new_decay_time()
- o bool set_new_decay_time(const new_decay_time_t& time)
- o int get_new_decay_time()
- o bool set_new_decay_time_multiplication(const uint8_t& time)
- o int get_new_decay_time_multiplication()

- left_dac
right_dac

Nastavení DA převodníků

- o enum channel_selection_t
 - MUTE = 0
 - THIS_CHANNEL = 1
 - SECOND_CHANNEL = 2
 - MONO_MIX = 3
 - o enum output_path_t
 - MIX_INPUT = 0
 - LINE_INPUT = 1
 - HIGH_POWER_INPUT = 2
 - o enum master_volume_t
 - INDEPENDENT = 0
 - RIGHT = 1
 - LEFT = 2
 - o enum quescent_current_t
 - DEFAULT = 0
 - PLUS_50 = 1
 - PLUS_100 = 3
 - o volume_t – viz A.5 Ovládání hlasitosti
 - o bool set_channel(const channel_selection_t& channel)
 - o int get_channel()
 - o bool power_on(const bool& on = true)
 - o bool power_off()
 - o int is_powered()
 - o int was_overflow()
 - o bool set_output_path(const output_path_t& path)
 - o int get_output_path()
 - o bool set_master_volume(const master_volume_t& master)
 - o int get_master_volume()
 - o bool set_quescent_current(const quescent_current_t& increase)
 - o int get_quescent_current()
 - o bool enable_digital_effects_filter(const bool& en = true)
 - o bool disable_digital_effects_filter()
 - o int is_digital_effects_filter_enabled()
 - o bool enable_deemphasis_filter(const bool& en = true)
 - o bool disable_deemphasis_filter()
 - o int is_deemphasis_filter_enabled()
 - o bool enable_resync(const bool& en = true)
 - o bool disable_resync()
 - o int is_resync_enabled()
- line_out_L
line_out_R

Nastavení signálových výstupů.

- o bool set_volume(const uint8_t& value)
 - value – hodnota v rozsahu 0 ÷ 9 dB.
 - o int get_volume()
 - o bool mute(const bool& mute = true)
 - o bool unmute()
 - o int is_muted()
 - o int are_gains_applied()
 - o bool power_on(const bool& value = true)
 - o bool power_off()
 - o int is_powered()

- Následující objekty jsou typu `volume_t` (viz A.5 Ovládání hlasitosti). Ovládají amplitudu přivedeného z daného bloku na výstup.
 - `pga_L`
 - `dac_L1`
 - `pga_R`
 - `dac_R1`

- `hplout`
`hprout`

Nastavení silových výstupů. Obsahují stejná nastavení jako signálové výstupy.

Navíc obsahují následující datové typy a metody:

- enum `common_mode_voltage_t`
 - `CMV_1V35` = 0
 - `CMV_1V5` = 1
 - `CMV_1V65` = 2
 - `CMV_1V8` = 3
- enum `soft_stepping_t`
 - `ONE_FS` = 0
 - `TWO_FS` = 1
 - `DISABLED` = 2
- bool `set_high_impedance()`
- bool `set_weak_drive()`
- int `is_high_impedance()`
- bool `enable_short_circuit_protection(const bool& en = true)`
- bool `disable_short_circuit_protection()`
- int `is_short_circuit_protection_enabled()`
- bool `set_short_circuit_shutdown()`
- bool `set_short_circuit_current_limit()`
- int `is_short_circuit_shutdown()`
- int `is_short_circuit_detected()`
- int `was_short_circuit_detected()`
- bool `set_common_mode_voltage(const common_mode_voltage_t& voltage)`
- int `get_common_mode_voltage()`
- bool `set_volume_soft_stepping(const soft_stepping_t& soft_stepping)`
- int `get_volume_soft_stepping()`
- bool `set_AVDD_as_weak_driver()`
- bool `set_bandgap_as_weak_driver()`
- int `is_AVDD_weak_driver()`
- bool `set_power_on_delay(const uint8_t& delay)`
- int `get_power_on_delay()`
- bool `set_driver_ramp_up_step_time(const uint8_t& value)`
- int `get_driver_ramp_up_step_time()`

- `hplcom`

Nastavení levého komplementárního silového výstupu. Obsahuje stejná nastavení jako silové výstupy a navíc následující:

- enum `mode_t`
 - `DIFF_HPLOUT` = 0
 - `VCM` = 1
 - `SINGLE_ENDED` = 2

- o `bool set_mode(const mode_t& mode)`
 - o `int get_mode()`
 - **hprcom**
 Nastavení pravého komplementárního silového výstupu. Obsahuje stejná nastavení jako silové výstupy a navíc následující:
 - o `enum mode_t`
 - `DIFF_HPROUT` = 0
 - `VCM` = 1
 - `SINGLE_ENDED` = 2
 - `DIFF_HPLCOM` = 3
 - `HPLCOM_FEEDBACK` = 4
 - o `bool set_mode(const mode_t& mode)`
 - o `int get_mode()`
 - **line2_RP**
line1_RM
line1_RP
line2_LP
line1_LM
line1_LP
 Nastavení přímého analogového propojení když je kodek v úsporném režimu.
 - o `bool enable(const bool& en = true)`
 - o `bool disable()`
 - o `int is_enabled()`
 - **filters**
 Sdružuje nastavení koeficientů digitálních filtrů. Každý filtr se skládá z koeficientů čitatele N a koeficientů jmenovatele D. Koeficienty jsou přístupné přes metody filtrů:
 - o `coeff_t& N(const uint8_t& index)`
 - o `coeff_t& D(const uint8_t& index)`
 Vrácený objekt typu `coeff_t` má následující metody, pomocí kterých se dá s hodnotou daného koeficientu manipulovat:
 - o `bool set(const int16_t& value)`
 - o `int get()`
 Jsou definovány následující filtry:
 - o `left_effects`
`right_effects`
 Koeficienty N0, N1, N2, N3, N4, N5, D1, D2, D4, D5
 - o `left_de_emphasis`
`right_de_emphasis`
 Koeficienty N0, N1, D1
 - o `left_high_pass`
`right_high_pass`
 Koeficienty N0, N1, D1
- Příklad použití:**
- ```
audio_codec.filters.left_effects.N(0).set(1234);
```
- Zápis do neexistujícího koeficientu nemá žádný efekt a je vždy úspěšný (návratová hodnota je `true`). Čtení neexistujícího koeficientu vrátí hodnotu 0. Selže-li funkce pro čtení, je vrácena hodnota 32768.

- `attenuation_3D`

objekt typu `coeff_t`, na kterém jsou definovány následující metody:

- `bool set(const int16_t& value)`
- `int get()`

Sežde-li funkce pro čtení, je vrácena hodnota 32768.

## A.5 Ovládání hlasitosti

Všechny objekty typu `volume_t`, sloužící k ovládání hlasitosti, mají následující metody:

- `bool set_volume(const uint8_t& value)` - viz následující tabulka
- `int get_volume()`
- `bool mute(const bool& mute = true)`
- `bool unmute()`
- `int is_muted()`

| value | Gain<br>[dB] | value | Gain<br>[dB] | value | Gain<br>[dB] | value        | Gain<br>[dB] |
|-------|--------------|-------|--------------|-------|--------------|--------------|--------------|
| 0     | 0,0          | 30    | -15,0        | 60    | -30,1        | 90           | -45,3        |
| 1     | -0,5         | 31    | -15,5        | 61    | -30,6        | 91           | -45,8        |
| 2     | -1,0         | 32    | -16,0        | 62    | -31,1        | 92           | -46,2        |
| 3     | -1,5         | 33    | -16,5        | 63    | -31,6        | 93           | -46,7        |
| 4     | -2,0         | 34    | -17,0        | 64    | -32,1        | 94           | -47,4        |
| 5     | -2,5         | 35    | -17,5        | 65    | -32,6        | 95           | -47,9        |
| 6     | -3,0         | 36    | -18,0        | 66    | -33,1        | 96           | -48,2        |
| 7     | -3,5         | 37    | -18,6        | 67    | -33,6        | 97           | -48,7        |
| 8     | -4,0         | 38    | -19,1        | 68    | -34,1        | 98           | -49,3        |
| 9     | -4,5         | 39    | -19,6        | 69    | -34,6        | 99           | -50,0        |
| 10    | -5,0         | 40    | -20,1        | 70    | -35,1        | 100          | -50,3        |
| 11    | -5,5         | 41    | -20,6        | 71    | -35,7        | 101          | -51,0        |
| 12    | -6,0         | 42    | -21,1        | 72    | -36,1        | 102          | -51,4        |
| 13    | -6,5         | 43    | -21,6        | 73    | -36,7        | 103          | -51,8        |
| 14    | -7,0         | 44    | -22,1        | 74    | -37,1        | 104          | -52,2        |
| 15    | -7,5         | 45    | -22,6        | 75    | -37,7        | 105          | -52,7        |
| 16    | -8,0         | 46    | -23,1        | 76    | -38,2        | 106          | -53,7        |
| 17    | -8,5         | 47    | -23,6        | 77    | -38,7        | 107          | -54,2        |
| 18    | -9,0         | 48    | -24,1        | 78    | -39,2        | 108          | -55,3        |
| 19    | -9,5         | 49    | -24,6        | 79    | -39,7        | 109          | -56,7        |
| 20    | -10,0        | 50    | -25,1        | 80    | -40,2        | 110          | -58,3        |
| 21    | -10,5        | 51    | -25,6        | 81    | -40,7        | 111          | -60,2        |
| 22    | -11,0        | 52    | -26,1        | 82    | -41,2        | 112          | -62,7        |
| 23    | -11,5        | 53    | -26,6        | 83    | -41,7        | 113          | -64,3        |
| 24    | -12,0        | 54    | -27,1        | 84    | -42,2        | 114          | -66,2        |
| 25    | -12,5        | 55    | -27,6        | 85    | -42,7        | 115          | -68,7        |
| 26    | -13,0        | 56    | -28,1        | 86    | -43,2        | 116          | -72,2        |
| 27    | -13,5        | 57    | -28,6        | 87    | -43,8        | 117          | -78,3        |
| 28    | -14,0        | 58    | -29,1        | 88    | -44,3        | 118 -<br>127 | Mute         |
| 29    | -14,5        | 59    | -29,6        | 89    | -44,8        |              |              |

Tabulka závislost skutečného zesílení na nastavené hodnotě

## Závislost skutečného zesílení na nastavené hodnotě

