VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

ZPRACOVÁNÍ PROSTOROVÉHO ZVUKU SFÉRICKÉHO MIKROFONNÍHO POLE

SPATIAL AUDIO PROCESSING OF A SPHERICAL MICROPHONE ARRAY

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jiří Tomešek

AUTHOR

VEDOUCÍ PRÁCE — Ing. Matej Liska

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Audio inženýrství**
specializace Akustika a audiovizuální technika
Ústav telekomunikací

Student: Bc. Jiří Tomešek ***ID:*** 220785
Ročník: 2 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Zpracování prostorového zvuku sférického mikrofonního pole

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s principy mikrofonních polí se zaměřením na sférické pole. Realizujte získávaní a zpracování zvuku z mikrofonního pole MEMS senzorů s TDM výstupem použitím FPGA. Navrhněte a demonstrujte spojení FPGA se softwarem Matlab, a implementujte aplikaci pro zpracování dat z MEMS mikrofonů.
V softwaru také realizujte aplikaci s jednoduchým GUI pro nastavení různého zpracování prostorového zvuku v závislosti na volbě uživatele, například v podobě formátu ambisonie vyšších řádů či beamformingu. Funkčnost demonstrujte.

DOPORUČENÁ LITERATURA:

[1] Time Division Multiplexed Audio Interface: A Tutorial [online], Cirrus Logic, 2006, Accessible from: https://gab.wallawalla.edu/~larry.aamodt/engr432/cirrus_logic_TDM_AN301.pdf
[2] CHURIWALA, Sanjay. Designing with Xilinx® FPGAs. India: Springer, 2017. ISBN 978-3-319-42437-8. Accessible from:
DOI 10.1007/978-3-319-42438-5.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 21.5.2024

Vedoucí práce: Ing. Matej Liska

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

ABSTRAKT

Diplomová práce se zabývá zpracováním prostorového zvuku sférického mikrofonního pole, jejich vlastnostmi a principy snímání. Dále jsou zde vysvětleny principy fungování MEMS mikrofonů a následná implementace. Rozhraní mezi mikrofonním polem a počítačem je vytvořeno pomocí programovatelného hradlového pole společně s USB převodníkem. Práce přibližuje vhodnou metodu softwarové implementace pro komunikace, řízení a propojení konkrétního hardwaru. Byla provedena a vysvětlena implementace jednotlivých funkcionalit pomocí programovacího jazyka VHDL v FPGA. Byl implementován příjem dat z mikrofonů pomocí TDM rozhraní, řídící logika a komunikace mezi FPGA a počítačem prostřednictvím rozhraní FTDI. V rámci práce byla také vytvořena aplikace v prostředí Matlab pro řízení FPGA a zpracování dat z mikrofonů včetně grafického uživatelského rozhraní. V aplikaci je implementována metoda ambisonie a metoda pro zpracování zvukového signálu pomocí prostorového filtrování.

KLÍČOVÁ SLOVA

Ambisonie, Beamforming, FPGA, FTDI, ICS-52000, Matlab, MEMS, mikrofony, mikrofonní pole, sférické pole, VHDL

ABSTRACT

The diploma thesis focuses on the processing of spatial sound from a spherical microphone array, their properties, and the principles of capturing. It further explains the principles of operation of MEMS microphones and subsequent implementation. The interface between the microphone array and the computer is created using a programmable gate array along with a USB converter. The thesis outlines a suitable method for software implementation for communication, control, and interconnection of specific hardware. The implementation of individual functionalities was carried out and explained using the VHDL programming language in an FPGA. Data reception from the microphones via a TDM interface, control logic, and communication between the FPGA and the computer through an FTDI interface were implemented. Within the thesis, an application was also created in the Matlab environment for controlling the FPGA and processing data from the microphones, including a graphical user interface. The application implements the ambisonics method and a method for processing the audio signal using spatial filtering.

KEYWORDS

Ambisonics, Beamforming, FPGA, FTDI, ICS-52000, Matlab, MEMS, microphones, microphones array, spherical array, VHDL

TOMEŠEK, Jiří. *Zpracování prostorového zvuku sférického mikrofonního pole*. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/159303. Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedoucí práce: Ing. Matej Liska

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Jiří Tomešek
VUT ID autora:	220785
Typ práce:	Diplomová práce
Akademický rok:	2023/24
Téma závěrečné práce:	Zpracování prostorového zvuku sférického mikrofonního pole

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné 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 závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné 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 autorského 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.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Matejovi Liskovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Cílem této diplomové práce bylo porozumět a seznámit se s principy mikrofonních polí se zaměřením na pole sférické. Získávat a zpracovávat zvuková data z mikrofonního pole MEMS senzorů s TDM výstupem použitím FPGA. Navrhnout a demonstrovat spojení FPGA se softwarem Matlab a implementace samotné aplikace pro zpracování dat z MEMS mikrofonů. V rámci aplikace také implementovat grafické uživatelské rozhraní pro nastavení různého zpracování prostorového zvuku v závislosti na volbě uživatele.

V kapitole 1 jsou popsány mikrofonní pole a jejich vlastnosti se zaměřením na pole sférické. Kapitola 2 se zabývá základními metodami pro signálové zpracování dat ze sférického mikrofonního pole, jako je tvarování přijímače pro prostorovou filtraci a ambisonie.

Práce se v kapitole 3 věnuje hardwaru, který byl použit v rámci práce. Zde jsou vysvětleny principy fungování MEMS mikrofonů, jejich využití a typy výstupních rozhraní se zaměřením na TDM rozhraní kvůli následné implementaci získávání a zpracování zvuku z mikrofonního pole MEMS senzorů s TDM výstupem. Rozhraní mezi mikrofonním polem a počítačem je vytvořeno pomocí programovatelného hradlového pole, jehož principy a možnosti použití jsou v této kapitole popsány. Pro vytvoření vhodné implementace bylo potřeba zjistit i vlastnosti programovatelného hradlového pole Digilent Arty S7-25 a mikrofonů ICS-52000, které byly přiděleny pro tuto práci.

Kapitola 4 se zabývá naprogramováním a vysvětlením implementace jednotlivých funkcionalit pomocí programovacího jazyka VHDL pro FPGA, jako je například generování periodických signálů pro řízení MEMS mikrofonů, řídící logika, implementace rozhraní pro přenos dat a příjem data z mikrofonů pomocí TDM rozhraní. K přenosu dat mezi FPGA a počítačem byl využit převodník FTDI.

V další části této kapitoly je popsána implementace funkcionalit v Matlab aplikaci, která se stará o řízení FPGA čipu, o příjem dat a zpracování dat z mikrofonů. Jako metody zpracování dat byly zvoleny Ambisonie a Beamforming, protože pro tyto metody mělo být optimalizováno přidělené sférické mikrofonní pole. Pro aplikaci bylo také navrženo grafické uživatelské rozhraní, jehož implementace je popsána v této kapitole.

1 Mikrofonní pole

Mikrofonní pole je obecně složeno z komponent, kterými jsou dva a více mikrofonů a komponenty pro signálové zpracování. Výstup takového pole je signál, který obsahuje sledovaný zvukový signál, šum, informaci o šíření a také rušení [1].

Sférické mikrofonní pole je často používáno ke snímání zvukových polí [1] a k získání jejich prostorových informací pomocí metody dekompozice rovinných vln [2]. Metoda dekompozice rovinných vln využívá sférickou Fourierovu transformaci k převedení tlakových signálů z mikrofonů na ambisonické signály popisující prostorové rozložení zvukového pole [2].

U sférického mikrofonního pole je důležitý návrh a jeho optimalizace v souladu s vlastnostmi, kterými má pole disponovat. Samotný návrh a jeho optimalizace jsou závislé na sférických harmonických funkcích. Optimální návrh sférického mikrofonního pole je především určen rozmístěním senzorů na povrchu koule a jejich vzdálenosti od jejího středu. Vlastnosti pole jsou ovlivněny také použitými senzory a konstrukcí pole, která může být otevřená, a nebo pevná uzavřená [3].

Ideální pole by mělo být schopno snímat nekonečný počet bodů na nekonečném počtu koulí různých poloměrů. To by umožnilo získat co nejvíce informací o zvukovém prostředí ve všech směrech a v různých vzdálenostech od mikrofonního pole. Uvedená vlastnost by umožnila věrně zachytit a reprodukovat různorodé zvukové scény a prostorové vlastnosti zvuku [4].

Ideální pole ale není v praxi realizovatelné, a proto při návrhu polí dochází ke kompromisům. V praxi se sférická mikrofonní pole často používají pro ambisonii [5], prostorové filtrování – beamforming [6] a odhad směru přicházejícího zvuku [6]. Samotná geometrie pole musí být navržena tak, aby nedocházelo k prostorovému aliasingu [7].

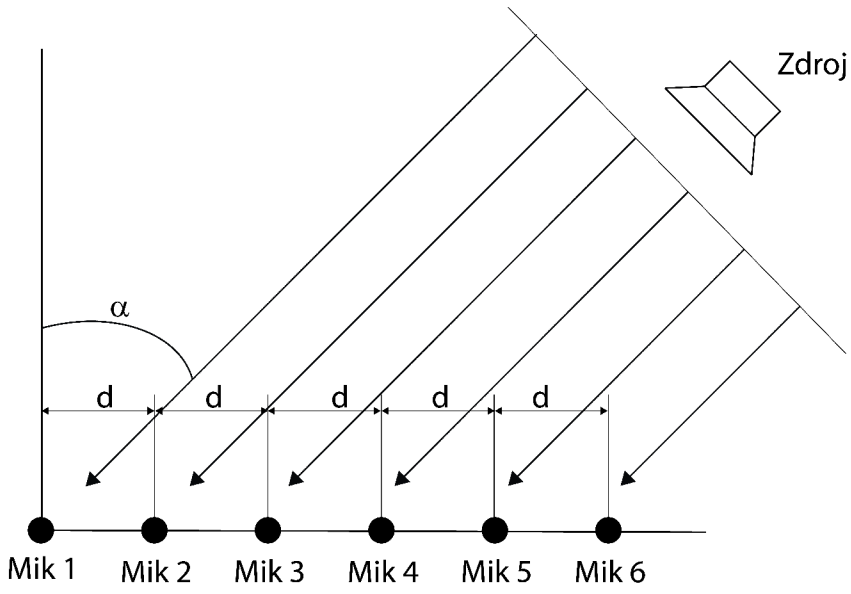
Při následném zpracování signálů, které jsou nasnímány mikrofonním polem, je potřeba řešit problematiku časové synchronizace, díky které je možné zajistit vhodné fázové podmínky [8]. Z hlediska způsobů zpracování signálů můžeme mikrofonní pole rozdělit na sumační, u kterých se signály z mikrofonů sčítají, a diferenční [9].

1.1 Lineární mikrofonní pole

Mezi standardní geometrické vzory mikrofonních polí patří lineární mikrofonní pole. Jeho výhoda spočívá v jednoduchosti s ohledem na zpracování a konstrukci. Na obrázku 1.1 je zobrazeno uniformní lineární pole. Uniformní pole má tu vlastnost, že vzdálenost jednotlivých mikrofonů od sebe je stejná. Dále je zde vyobrazen zvukový zdroj, který má vzdálenost od mikrofonního pole větší, než je vzdálenost mezi mikrofony v poli [7].

Za těchto okolností můžeme zvukový zdroj považovat za zdroj ve vzdáleném poli. Díky tomu je možné kulové plochy, které se šíří od zdroje, aproximovat plochami rovinnými. Takže je možné zvukové vlny, které se šíří k mikrofonům, považovat za vzájemně rovnoběžné [7].

Z obrázku 1.1 je patrné, že ke každému mikrofonu zvuková vlna dorazí v rozdílném okamžiku, jelikož musí ke každému mikrofonu urazit rozdílnou vzdálenost.



Obr. 1.1: Uniformní lineární pole s mikrofony a zdrojem [7].

Lineární pole má tu nevýhodu, že je možné určit úhel α pouze v rozmezí od $0\,^\circ$ do $180\,^\circ$. Problém je označován jako předozadní nejednoznačnost lineárního pole, kvůli kterému nemůže být přesně zodpovězena otázka, jestli je zdroj zvuku před nebo za mikrofonním polem. Problém lze vyřešit použitím jiné geometrie mikrofonního pole, např. sférického [10].

Směrovost pole závisí na konstrukci mikrofonního pole a způsobu tvorby výstupního signálu. V případě výstupu dat z jednotlivých mikrofonů do adaptivního výpočetního zařízení může být dynamicky měněna [11].

1.2 Sférické mikrofonní pole

Sférické mikrofonní pole je dalším typem mikrofonního pole, kdy jsou mikrofony uspořádány na povrchu sféry [12]. Sférické pole bývá konstruováno buď jako otevřené pole, nebo pevné pole. V otevřeném sférickém poli jsou mikrofony umístěny a přichyceny ve volném prostoru. V případě pevného pole jsou mikrofony umístěny a

připevněny k pevné kulové konstrukci. Tyto konstrukce se obvykle vyrábí z tvrdého plastu [13].

1.2.1 Otevřené sférické pole

Otevřené sférické pole má oproti pevnému poli několik výhod. Otevřené pole je "akusticky průhledné", tzn. že nehrozí rozptyl dopadajících vln odrazem z velké pevné koule zpět do oblasti, ve které probíhá měření. Další výhodou je velký výběr algoritmů pro konfiguraci paprsku a v neposlední řadě nízká hmotnost v případě vytváření rozměrných polí. Za určitých okolností je vhodné vytvářet velká mikrofonní pole, např. při potřebě nahrávat nízké frekvence. V případě pevné konstrukce by tato pole měla vysokou hmotnost [14].

Nevýhoda otevřené konstrukce koule je, že na frekvencích, které odpovídají uzlovým hodnotám sférických módů, vzniká nadměrné množství hluku [13].

1.2.2 Pevné sférické pole

Pevná kulová konstrukce je jednodušší na výrobu. Umožňuje do jejího vnitřku uschovat všechny elektronické součástky a kabely. Nevýhoda pevné konstrukce je, že konstrukce narušuje zvukové pole kolem svého těla kvůli tomu, že rozptyluje dopadající zvukové vlny. Další problém, který může u tohoto typu konstrukce nastat, je problém s velikostí a následně s jeho váhou. Pokud má být sférické mikrofonní pole použitelné na nízkých frekvencích, musí být využito polí s velkými rozměry. Na rozdíl od otevřených mikrofonních polí netrpí pevná pole problémy s hlukem nebo šumem [10].

Pevné sférické mikrofonní pole pracuje na principu snímání hodnot akustického tlaku na různých místech sféry poté, co dojde k rozptylu příchozí zvukové vlny na povrchu sféry, který se řídí tzv. sférickými harmonickými funkcemi [15].

1.3 Prostorové vzorkování a aliasing

Zvukové pole je v dnešní době potřeba vzorkovat několika diskrétními mikrofony, protože na trhu není dostupné řešení v podobě spojitého sférického tlakového senzoru. Prostorový aliasing vzniká hlavně u pravidelně uspořádaných polí. U prostorového vzorkování u uniformních lineárních polí záleží na vzdálenosti d umístění mikrofonů od sebe. Zde musí být dodržena podmínka, která definuje, že vzdálenost senzorů od sebe nesmí být větší, než je vlnová délka snímaného signálu [6]

$$\lambda < \frac{d}{2}, \tag{1.1}$$

kde λ je vlnová délka a d je vzdálenost mikrofonů od sebe. Pokud podmínka nebude splněna, bude docházek k nejednoznačnosti při lokalizaci zdroje zvuku.

Prostorový aliasing u sférického mikrofonního pole nastane, když jsou zvuková pole vyšších řádů zachycena pomocí pole s neadekvátním počtem mikrofonních senzorů. Musí být dodrženo, že počet mikrofonů závisí na řádu zvukového pole dle [16]

$$S \geq (N+1)^2. \tag{1.2}$$

N zde označuje řád zvukového pole a S počet snímačů. Pokud podmínka není dodržena, způsobí to, že vlastní paprsky vyššího řádu jsou aliasingovány do nižších řádů. Ve skutečnosti je vyžadováno nekonečné množství mikrofonů, aby bylo možné zachytit skutečné zvukové pole, protože ty nejsou řádově omezené a jsou reprezentovány nekonečnou řadou sférických harmonických funkcí [16].

2 Signálové zpracování sférického pole

Kapitola se zabývá teorií signálového zpracování signálů ze sférického mikrofonního pole pomocí metod pro prostorové filtrování a ambisonii.

2.1 Tvarování směrové charakteristiky přijímače

Tvarování přijímače mikrofonního pole neboli beamforming je technika, pomocí které lze provádět tvarování paprsku přijímače. Signály z mikrofonů v poli jsou považovány za vstupní signály pro následné zpracování, z nichž následně vzniká jeden výstupní signál s požadovanými vlastnostmi. Jedna taková vlastnost může být třeba zesílení signálu od zdroje zvuku, který je považován za primární, a signály od sekundárních zdrojů zvuku může potlačit. Toto zpracování je umožněno díky použití směrového filtru, který je obvykle označován jako beamformer. Techniky lze rozdělit na fixní a adaptivní. Fixní techniky mají pevně nastavené parametry, zatímco parametry u adaptivních technik se mění v závislosti na datech z mikrofonního pole. Použití jednotlivých technik je závislé na podmínkách, ve kterých bude mikrofonní pole používáno [17].

2.1.1 Rovnice pro prostorové filtrování

Signál z výstupu prostorového filtru je zde označen jako y. Prostorový filtr je definován jako akusticky tlak p(k,r, θ, ϕ) působící na kouli s poloměrem r, kde k je vlnové číslo, θ je úhel elevace a ϕ je úhel azimutu. Akustický tlak je násoben váhovací funkcí [w(k, θ, φ)]* a ještě je provedena integrace po celém povrchu [17]

$$y = \int_0^{2\pi} \int_0^{\pi} [w(k, \theta, \phi)]^* p(k, r, \theta, \phi) \sin\theta d\theta d\phi. \tag{2.1}$$

Sférické mikrofonní pole je složeno z M mikrofonů, které jsou na povrchu koule o poloměru r. Pozice mikrofonů budou značeny jako $(r, \theta_\mathrm{q}, \phi_\mathrm{q})$, kde $m = 1$ až M. Mikrofonem m je měřen akustický tlak p při vlnovém čísle k. Je označen jako: $p_\mathrm{q}(k) \equiv p(k, r, \theta_\mathrm{m}, \phi_\mathrm{m})$. Akustický tlak pak následně tvoří vektor o velikosti $M \times 1$ [17]

$$\mathbf{p} = [p_1(k), p_2(k), ..., p_\mathrm{Q}(k)]^T. \tag{2.2}$$

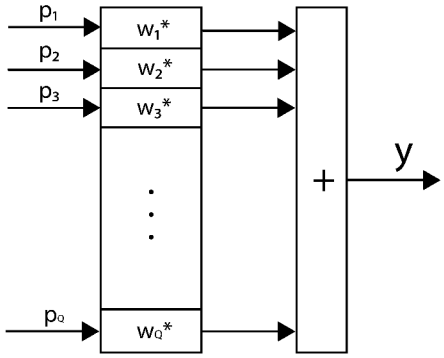
Diskrétní verze prostorového filtru s váhou $w_\mathrm{m}(k)$, kde m je číslo mikrofonu. Váhový vektor je definován jako [17]

$$\mathbf{w} = [w_1(k), w_2(k), ..., w_\mathrm{M}(k)]^T. \tag{2.3}$$

Výstup pole y je pak často definován jako vnitřní součin [17]

$$y = \mathbf{w}^H \mathbf{p}. \tag{2.4}$$

Může být taky vyjádřen pomocí blokového diagramu jako na obrázku 2.1.



Obr. 2.1: Blokový diagram beamforming systému [17].

Obecně lze říci, že problém návrhu paprsku pole nebo prostorového filtru lze označit jako navržení vektoru **w** takovým způsobem, že pro přesně definovaný vstup p se na výstupu y objeví signál s požadovanými vlastnostmi. Obvykle se při definicích vlastností pole předpokládá, že na vstup mikrofonního pole dopadá rovinná vlna s jednotnou amplitudou. Zde bude naměřený tlak nahrazen řídícím vektorem. Tento vektor obsahuje amplitudu rovinné vlny, která je měřena každým dílčím mikrofonem v poli. Vektor řízení **v** má pro mikrofonní pole ve volném poli formu [17]

$$\mathbf{v} = [v_1, v_2, ..., v_\mathrm{M}]^T, \tag{2.5}$$

kde v_m je definováno jako

$$v_\mathrm{m} = e^{\mathbf{ik}\cdot\mathbf{r}}, 1 \leq m \leq M. \tag{2.6}$$

Vektor $\mathbf{k} = (k, \theta_\mathrm{k}, \phi_\mathrm{k})$ je směrový vektor, který nese informaci o směru příchodu rovinné vlny k mikrofonnímu poli. Vektor $\mathbf{r} = (k, \theta_\mathrm{m}, \phi_\mathrm{m})$ je vektor, který udává polohu mikrofonu m. Výstup mikrofonního pole nyní může být definován podle [17] jako

$$y = \mathbf{w}^H\mathbf{v}. \tag{2.7}$$

Pokud jsou použity různé konfigurace mikrofonního pole, je tomu potřeba přizpůsobit i řídící vektor. Například pokud by bylo použito pevné mikrofonní pole, řídící vektor musí zajistit kompenzaci účinku rozptylu zvuku z koule [17].

Rovnice pole v oblasti prostoru lze upravit pro oblast sférických harmonických funkcí. V rovnici 2.8 jsou tlakové a váhové funkce vyjádřeny přes kouli a jsou označovány jako $p_\mathrm{nm}(k)$ a $w_\mathrm{nm}(k)$ pro sférické Fourierovy transformace, které jim náleží. Funkce sférických Fourierových koeficientů je možné zapsat jako [17]

$$y = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} [w_\mathrm{nm}(k)]^* p_\mathrm{nm}(k, r). \tag{2.8}$$

Rovnici je možné vyjádřit v maticovém tvaru, jestliže koeficienty za řádem N jsou rovny nule. Rovnice v maticovém tvaru je pak [17]

$$y = w_{\mathrm{nm}}^{H} p_{\mathrm{nm}}. \tag{2.9}$$

Kde vektory $\mathbf{w}_{\mathrm{nm}}$ a $\mathbf{p}_{\mathrm{nm}}$ jsou definovány jako

$$\mathbf{w}_{\mathrm{nm}} = [w_{00}(k), w_{1(-1)}(k), ..., w_{\mathrm{NN}}(k)]^{T}, \tag{2.10}$$

$$\mathbf{p}_{\mathrm{nm}} = [p_{00}(k,r), p_{1(-1)}(k,r), ..., p_{\mathrm{NN}}(k,r)]^{T}. \tag{2.11}$$

Výstup mikrofonního pole je možné přepsat do tvaru pro sférickou harmonickou doménu jako [17]

$$y = \mathbf{w}_{\mathrm{nm}}^{H} \mathbf{v}_{\mathrm{nm}}, \tag{2.12}$$

kde vektor $\mathbf{v}_{\mathrm{nm}}$ je definován jako

$$\mathbf{v}_{\mathrm{nm}} = [v_{00}, v_{1(-1)}, ..., v_{\mathrm{NN}}]^{T}, \tag{2.13}$$

kde $\mathbf{v}_{\mathrm{nm}}$ je vstup pole.

Pro otevřené mikrofonní pole lze p_{nm} a v_{nm} zapsat jako [17]

$$\mathbf{p}_{\mathrm{nm}}(k,r) = 4\pi i^{n} j_{\mathrm{n}}(kr)[\, Y_{\mathrm{n}}^{m}(\theta_{\mathrm{k}}, \phi_{\mathrm{k}})]^{*}, \tag{2.14}$$

$$\mathbf{v}_{\mathrm{nm}} = 4\pi i^{n} j_{\mathrm{n}}(kr)[\, Y_{\mathrm{n}}^{m}(\theta_{\mathrm{k}}, \phi_{\mathrm{k}})]^{*}. \tag{2.15}$$

Kde n je řád sférických harmonických funkcí, j_{n} je sférická Besselova funkce druhého řádu a Y_{n}^{m} jsou koeficienty sférických harmonických funkcí příslušného řádu.

To lze ještě zobecnit na výraz

$$\mathbf{v}_{\mathrm{nm}} = b_{\mathrm{n}}(kr)[\, Y_{\mathrm{n}}^{m}(\theta_{\mathrm{k}}, \phi_{\mathrm{k}})]^{*}, \tag{2.16}$$

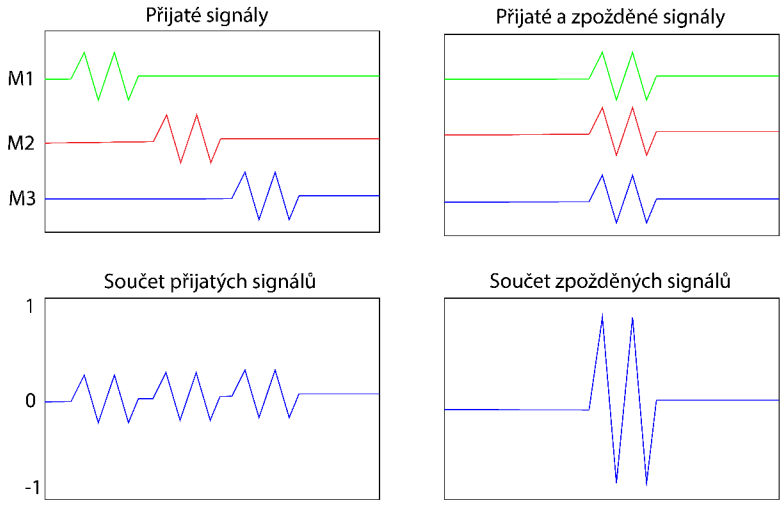
kde $b_{\mathrm{n}}(kr) = 4\pi i^{n} j_{\mathrm{n}}(kr)$. Pomocí jednoduchých úprav funkce $b_{\mathrm{n}}(kr)$, která je označována jako síla sférického módu [18], lze beamforming přizpůsobit dalším typům polí [17].

2.1.2 Metoda Delay and Sum

Jde o velmi často používaný typ beamformeru kvůli jeho jednoduchému použití a realizaci. Cílem této metody je dosáhnout maximálního výstupu ve směru příchodu zvukové vlny. Tato metoda má také velkou odolnost vůči šumu a nejistotám [17].

Delay and sum metoda využívá koeficienty zpoždění a amplitudové váhovací koeficienty. Zpoždění a váhovací koeficienty jsou aplikovány na každý z mikrofonních senzorů a výsledné signály jsou sečteny. Zpoždění pro jednotlivé mikrofony je voleno

tak, aby bylo dosaženo shodné fáze rovinné vlny pro všechny mikrofonní senzory. Tím se maximalizuje citlivost mikrofonního pole ve směru příchozí zvukové vlny [19]. Úpravou zpoždění signálu z mikrofonů lze poměrně jednoduše měnit směr hlavního laloku směrem ke zdroji zvuku. Využívá se konstruktivní a destruktivní interference. Na signály ve směru hlavního laloku se uplatňuje konstruktivní interference a jsou zesíleny. Na signály přicházejí z nežádoucího směru zase destruktivní interference a jsou potlačeny. Je možné definovat pole, které je složeno z M mikrofonních senzorů, které jsou umístěny v prostoru $\mathbf{x}_\mathrm{m} = [x_\mathrm{m}, y_\mathrm{m}, z_\mathrm{m}]$ a měří vlnoplochu $f(\mathbf{x}, t)$ [19].



Obr. 2.2: Výstupní signál pole bez použití a s použitím delay and sum metody [19].

Delay and sum beamformer je složen z části, která na vstupní signály jednotlivých mikrofonních senzorů aplikuje definované zpoždění δ_m, a z části, která na jednotlivé signály aplikuje amplitudové váhovací koeficienty w_m. Následně jsou výstupy všech senzorů sečteny. Celý tento postup je znázorněn na obrázku 2.3. Jednotlivá zpoždění, která jsou aplikována na signály z mikrofonních senzorů, jsou zvolena tak, aby došlo k maximalizování citlivosti mikrofonního pole na zvukové vlny šířící se z daného směru [19].

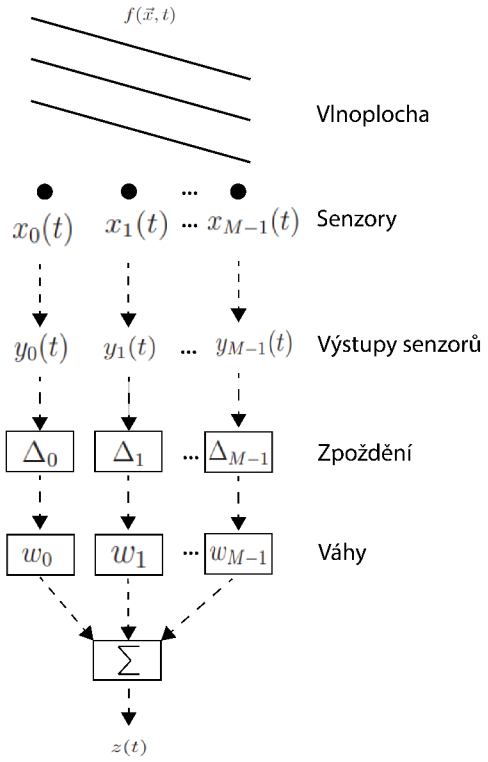
Směr hlavního laloku, ve kterém bude citlivost zvukového pole maximalizována, zle upravovat právě pomocí zpoždění a díky tomu je možné zvukové pole nasměrovat ke konkrétnímu zdroji zvuku. Pro jednotlivé mikrofonní senzory mohou být použity různé váhovací koeficienty a mohou jednotlivé snímače zeslabovat, nebo naopak zesilovat. Rozdílnými koeficienty lze upravovat tvar hlavního laloku a do jisté míry potlačit postranní laloky, aby nedocházelo k naslouchání z vedlejších směrů. Delay and sum metoda spadá do metod neadaptivních a má koeficienty nastaveny fixně nezávisle na přijímané zvukové vlně.

Výstup delay and sum metody je pak dán rovnicí [19]

$$\mathbf{z}(t) = \sum_{m=0}^{M-1} w_{\mathrm{m}} \times y_{\mathrm{m}}(t - \delta_{\mathrm{m}}), \tag{2.17}$$

kde M je celkový počet mikrofonů, w_{m} je matice váhovacích koeficientů a y_{m} je výstupní signál z mikrofonů s příslušným zpožděním.

Základní myšlenkou beamformingu je použití sad různých zpoždění pro nasměrování hlavního laloku pole do různých směrů, aby bylo dodrženo, že směr laloku odpovídá směru zdroje zvuku a tím bude zajištěn maximální výstupní výkon [19].



Obr. 2.3: Blokové schéma delay and sum metody [19].

2.1.3 Metody založené na spektrální analýze

Spektrum s minimální odchylkou MVS

Metoda MVS (Minimum Variance Spectrum) vychází z metody MV pro jednu dimenzi, kterou je možné rozšířit na dvě a více dimenzí. Základním prvkem MVS je úzkopásmový filtr, který přizpůsobuje svůj rozptyl (varianci) spektrálním složkám vstupního signálu na každé zvolené (zájmové) frekvenci. Zájmový signál na zvolené frekvenci projde úzkopásmovým filtrem bez zkreslení, zatímco ostatní frekvence jsou pomocí úzkopásmového filtru potlačeny [20].

Při výpočtu se vychází z odhadu kovarianční matice R například pomocí metody vnějšího součinu. Výsledné spektrum je poté reprezentováno pomocí maximálního zpoždění autokorelace L_1 a L_2 a definováno jako [20]

$$P(f_1, f_2) = \frac{(L_1 + 1)\,(L_2 + 1)}{e^H(f_1, f_2)\,R^{-1}\,e(f_1, f_2)}, \tag{2.18}$$

kde $e(f_1, f_2)$ je komplexní 2D Fourierův vektor [20]

$$e(f_1, f_2) = exp(-j\,2\pi\,f_2(0 : L_2)^T) \otimes exp(-j\,2\pi\,f_1(0 : L_1)^T). \tag{2.19}$$

MUSIC spektrum

Metoda MUSIC (Multiple Signal Classification) stejně jako metoda MVS může být rozšířena do více dimenzí a předpokládá se, že nasnímaný signál je ovlivněn bílým šumem. Vektor kovarianční matice je rozložen na dva ortogonální podprostory, a to na prostor signálu a prostor šumu. Opět je potřeba pomocí metod odhadnout kovarianční matici R a dále vypočítat vlastní vektory a hodnoty matice R pomocí singulárního rozkladu [20]

$$R = U \sum V^H. \tag{2.20}$$

Matice U obsahuje vlastní vektory a diagonální matice $\sum$ obsahuje seřazené singulární hodnoty matice R, které jsou všechny nenulové kvůli šumu [20].

MUSIC pracuje výhradně s podprostorem šumu, který je definován v matici U. Proto MUSIC využívá ortogonalitu mezi podprostorem signálu a podprostorem šumu, který je označován jako U_n. Spektrum MUSIC je definováno jako [20]

$$P(f_1, f_2) = \frac{(L_1 + 1)\,(L_2 + 1)}{e^H(f_1, f_2)\;U_\mathrm{n}\;U_\mathrm{n}^H\,e(f_1, f_2)}. \tag{2.21}$$

2.2 Odhad směru příchodu založený na rozkladu sférických harmonik

Metoda odhadu směru příchodu (DOA) slouží k lokalizaci a sledování zájmového signálu. Lze využít u různých typů polí, ale v této části bude metoda zaměřena na sférické pevné mikrofonní pole. Metoda vychází z předpokladu, že zvukové pole je složeno z vln rovinných. Metoda se zabývá zpracováním signálů v doméně sférických harmonik s využitím sférické Fourierovy transformace [21].

Sférická Fourierova transformace, která bývá označována i jako rozklad sférických harmonických funkcí, je definována jako [21]

$$f(\theta, \phi) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} f_\mathrm{nm}\,Y_\mathrm{n}^m(\theta, \phi), \tag{2.22}$$

kde f_{nm} jsou sférické harmonické koeficienty a Y_{n}^{m} jsou sférické harmonické funkce. Sférické harmonické koeficienty f_{nm} jsou definovány jako [21]

$$f_{\mathrm{nm}} = \int_0^{2\pi} \int_0^{\pi} f(\theta, \phi)[\, Y_{\mathrm{n}}^{m}(\theta, \phi)]^{*} sin\theta \, d\theta \, d\phi, \tag{2.23}$$

kde θ je úhel azimutu a může nabývat hodnot 0 až π a ϕ je úhel elevace a může nabývat hodnot 0 až 2π. Sférické harmonické funkce jsou definovány jako [21]

$$Y_{\mathrm{n}}^{m}(\theta, \phi) = \sqrt{\frac{(2n+1)(n-m)!}{4\pi(n+m)!}} P_{\mathrm{n}}^{m}(cos\theta) e^{im\phi}, \tag{2.24}$$

kde i je imaginární jednotka, n a m je řád a index sférické harmonické funkce. $P_{\mathrm{n}}^{m}(cos\theta)$ je zde označení pro Legendreovu funkci.

K rozkladu rovinných vln lze pak využít výše definovanou sférickou Fourierovu transformaci. Pokud budou snímány rovinné vlny přicházející ze směrů $s(\theta_{\mathrm{s}}, \phi_{\mathrm{s}})$, akustický tlak na souřadnici $p(r, \theta_{\mathrm{q}}, \phi_{\mathrm{q}})$ je dán součtem sférických harmonických funkcí a sférické Besselovy funkce jako [21]

$$p(kr, \theta_{\mathrm{q}}, \phi_{\mathrm{q}}) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \sum_{s=1}^{S} b_{\mathrm{n}}(kr) a_{\mathrm{s}}(k)[\, Y_{\mathrm{n}}^{m}(\theta_{\mathrm{s}}, \phi_{\mathrm{s}})]^{*} \, Y_{\mathrm{n}}^{m}(\theta_{\mathrm{q}}, \phi_{\mathrm{q}})[\mathrm{Pa}], \tag{2.25}$$

kde a_{s} je amplituda rovinné vlny s, k je vlnové číslo a b_{n} je síla sférického módu. Síla sférického módu má rozdílnou definici pro otevřené a pevné sférické mikrofonní pole. Pro pevné sférické mikrofonní pole je b_{n} definováno jako [21]

$$b_{\mathrm{n}}(kr) = 4\pi i^{n} \left(j_{\mathrm{n}}(kr) - \frac{j'_{\mathrm{n}}(ka)}{h'_{\mathrm{n}}(ka)} h_{\mathrm{n}}(kr) \right), \tag{2.26}$$

kde j_{n} je sférická Besselova funkce a h_{n} sférická Hankelova funkce druhého řádu a j'_{n} h'_{n} jsou jejich derivace [22]. Z výše uvedených vzorců je možné vypočítat akustický tlak na povrchu sféry v bodech snímání jako [21]

$$p_{\mathrm{nm}} = \frac{4\pi}{Q} \sum_{q=1}^{Q} p(kr, \theta_{\mathrm{q}}, \phi_{\mathrm{q}})[\, Y_{\mathrm{n}}^{m}(\theta_{\mathrm{q}}, \phi_{\mathrm{q}})]^{*}. \tag{2.27}$$

Nyní může být vypočten beamforming ve sférické harmonické doméně jako [21]

$$y(\theta_{\mathrm{l}}, \phi_{\mathrm{l}}) = \sum_{n=0}^{N} \sum_{m=-n}^{n} \frac{p_{\mathrm{nm}}}{b_{\mathrm{n}}(kr)} Y_{\mathrm{n}}^{m}(\theta_{\mathrm{l}}, \phi_{\mathrm{l}}). \tag{2.28}$$

Směr zájmu je pak nalezen jako maximum $y(\theta_{\mathrm{l}}, \phi_{\mathrm{l}})$ [21].

2.3 Ambisonie

Jde o metodu pro kódování prostorového zvuku z mikrofonního pole, která klade důraz na snímání směrových vlastností zvuku. V obvyklém vícekanálovém zvuku,

jako je třeba 5.1 a 7.1, každý kanál nese informaci odpovídající danému reproduktoru. V ambisonii tomu tak není. Zde má každý kanál informaci o daných fyzikálních vlastnostech akustického pole. Mezi tyto vlastnosti spadá akustická rychlost, tlak a akustická výchylka. Metodu ambisonie je možné rozlišovat podle řádů na [23]:

- `Nultý řád` – zaznamenává informaci o tlakovém poli v počátku souřadnicového systému. To znamená, že zaznamenává informaci ze všesměrového mikrofonu. Kanál pro tlakové pole je označován jako *W*.
- `První řád` – zaznamenává informaci o tlakovém poli v počátku souřadnicového systému a ještě informaci o akustické rychlosti v počátku souřadnicového systému. Jde o záznam tří mikrofonů s osmičkovou směrovou charakteristikou podél každé osy souřadnicového systému. Zde jsou kanály označovány jako *X, Y, Z*.
- `Druhý a vyšší řád` – přidává informace o derivátech vyšších řádů tlakového pole. Kanály bývají již označovány pouze číselně.

2.3.1 Vlastnosti

Výše uvedená technologie má oproti jiným přístupům k prostorovému zvuku jiné vlastnosti. Je založena na základě fyzikálních principů akustického pole. Není omezena na rovinné vlny, ale může zaznamenat jakékoliv obecné zvukové pole. Zjednodušeně řečeno, dokáže zachytit a reprodukovat jakýkoliv zvuk, který je možné slyšet v reálném světě. Dále je zcela nezávislá na rozmístění reproduktorů. To znamená, že pracovní postup při produkci a postprodukci ambisonie nezávisí na rozmístění reproduktorů při přehrávání. Tradiční stereo a vícesměrové přístupy jsou založeny na předpokladu, že reproduktory jsou umístěny na přesně stanovených pozicích. Toho se pak následně využívá při přehrávání, záznamu a produkci [23].

Ambisonie není založena na objektech. Má pevné kanály, které zachytávají zvuk ze všech směrů. V jiných přístupech založených na objektech, ve kterých je například každý zvukový objekt charakterizován monofonní stopou a sadou metadat pro uchování dalších vlastností o objektu, dochází k problému s paměťovou a výpočetní náročností u více objektů. Produkci založenou na objektech je možné bez větších problémů převést na ambisonické kódování. V opačném případě z ambisonie získat zvukové objekty není snadné. Součástí metody je část zabývající se nahráváním, kódováním, postprodukcí, přenosem a reprodukcí [23].

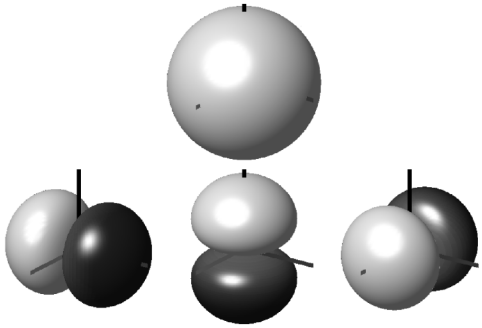
2.3.2 Nahrávání a kódování

V první řadě je potřeba zvukové pole zaznamenat nebo zakódovat. Zakódováním je myšleno vytvořit zvukové pole synteticky z monofonního záznamu. Zaznamenáním

je myšleno zaznamenat zvukové pole pomocí speciálního mikrofonu. Hlavním principem ambisonie je zakódování zvukového pole do pevného počtu kanálů. Pro 1. řád to jsou kanály čtyři. Kanál *W* je úměrný akustickému tlaku p v poslechovém bodě a kanály *X, Y, Z* odpovídají x, y, z složkám vektoru akustické rychlosti v v poslechovém bodě. To znamená, že kanál *W* kóduje akustický tlak v počátku a kanály *X, Y, Z* směr příchozího zvuku také v počátku. Počet mikrofonů M v závislosti na řádu r ambisonie je dán vztahem [23]

$$M = (r + 1)^2. \tag{2.29}$$

Prvnímu řádu odpovídá 3D zobrazení sférických harmonických funkcí na obrázku 2.4. Jde o grafické znázornění, které zobrazuje citlivost mikrofonu na zvuky přicházející z různých směrů [23].



Obr. 2.4: 3D zobrazení sférických harmonických funkcí 1. řádu [23].

2.3.3 Kódování rovinné vlny v Ambisonii prvního řádu

Rovinná vlna je dána zvukovým signálem *s(t)* a směrem příchozího zvuku. Směr příchozího zvuku od zdroje je možné vyjádřit jednotkovým vektorem $\mathbf{k}$. Ten udává směr dopadající rovinné vlny. Tento směrový vektor je možné rozložit ve sférických souřadnicích na $\mathbf{k} = (\cos\phi\cos\theta, \sin\phi\cos\theta, \sin\phi)$. Kde ϕ vyjadřuje úhel azimutu a θ vyjadřuje úhel elevace zdroje zvuku. Akustický tlak lze vyjádřit pomocí *s(t)* a $\mathbf{k}$ jako [23]

$$p(t, \mathbf{r}) = s(t + \mathbf{k} \cdot \frac{\mathbf{r}}{c}), \tag{2.30}$$

kde $\mathbf{r}$ je polohový vektor a c je rychlost šíření zvuku ve vzduchu. Rovinná vlna má také vektor akustické rychlosti $\mathbf{v}$. Ten je definován jako [23]

$$\mathbf{v}(t, \mathbf{r}) = -\frac{1}{Z} s(t + \mathbf{k} \cdot \frac{\mathbf{r}}{c}\mathbf{k}). \tag{2.31}$$

Z je akustická impedance. $Z = \rho_0 c \approx 420\,\mathrm{kg\,s^{-1}}$, kde ρ_0 je akustická hustota vzduchu. V počátku souřadnicového systému je možné vyjádřit rovnice pro jednotlivé osy jako [23]

$$p(t, \mathbf{0}) = s(t), \tag{2.32}$$

$$\mathbf{v}_\mathrm{x}(t, \mathbf{0}) = -\frac{1}{Z} s(t) \cos\phi \cos\theta, \tag{2.33}$$

$$\mathbf{v}_\mathrm{y}(t, \mathbf{0}) = -\frac{1}{Z} s(t) \sin\phi \cos\theta, \tag{2.34}$$

$$\mathbf{v}_\mathrm{z}(t, \mathbf{0}) = -\frac{1}{Z} s(t) \sin\theta. \tag{2.35}$$

Tyto čtyři rovnice tvoří základ pro ambisonii 1. řádu. Po úpravě získáme rovnice pro kanály *W, X, Y, Z*, do kterých se zakóduje virtuální zdroj zvuku *s(t)*, který přichází z úhlového směru (ϕ, θ). Zakódován je tímto způsobem [23]

$$W(t) = \frac{s(t)}{\sqrt{2}}, \tag{2.36}$$

$$X(t) = s(t) \cos\phi \cos\theta, \tag{2.37}$$

$$Y(t) = s(t) \sin\phi \cos\theta, \tag{2.38}$$

$$Z(t) = s(t) \sin\theta. \tag{2.39}$$

Ambisonii lze použít i pro 2D, v případě 2D využití se $\theta = 0$ a $Z(t) = 0$ [23].

2.3.4 Kódování obecného zvukového pole v Ambisonii

Pro obecné použití v případě více zdrojů zvuku, jejichž polohu lze přesně určit, je možné signály s_i přicházející ze směrů ϕ_i, θ_i vypočítat do kanálů ambisonie jako [23]

$$W(t) = \sum_i \frac{s_\mathrm{i}(t)}{\sqrt{2}}, \tag{2.40}$$

$$X(t) = \sum_i s_\mathrm{i}(t) \cos\phi_\mathrm{i} \cos\theta_\mathrm{i}, \tag{2.41}$$

$$Y(t) = \sum_i s_\mathrm{i}(t) \sin\phi_\mathrm{i} \cos\theta_\mathrm{i}, \tag{2.42}$$

$$Z(t) = \sum_i s_\mathrm{i}(t) \sin\theta_\mathrm{i}. \tag{2.43}$$

Záznam ve zvukovém kanálu $W(\mathrm{t})$ odpovídá záznamu virtuálního mikrofonu, který je stejně citlivý na zvuky přicházející ze všech směrů. Záznam ve zvukových kanálech $X(\mathrm{t})$, $Y(\mathrm{t})$ a $Z(\mathrm{t})$ odpovídá zvukovému záznamu v osách *x, y, z* ze tří virtuálních mikrofonů, které mají osmičkovou charakteristiku směrovosti. V uvedené technologii ambisonie je možné zakódovat i jiné než rovinné vlny. Takové kódování je však

složitější. Obecně lze říci, že je možné si vystačit jen s kódováním rovinných vln, protože každý zvukový zdroj může být zakódován jako několik rovinných vln. I kulová vlna může být od určité vzdálenosti od počátku aproximována pomocí několika rovinných vln s dostatečnou přesností. Vzdálenost musí být dostatečná k tomu, aby se kulová vlna téměř stala vlnou rovinnou. To se děje z důvodu, že se kulová vlna postupně více rozptyluje v závislosti na rostoucí vzdálenosti od zdroje [23].

2.3.5 Záznam Ambisonie pomocí pole mikrofonů

Pro 1. řád ambisonie se použijí tři mikrofony pro kanály $X(\mathrm{t})$, $Y(\mathrm{t})$ a $Z(\mathrm{t})$, kde každý jeden mikrofon bude ve směru jedné z os *x, y, z*. Všechny mikrofony by měly být ideálně uspořádány v jednom bodě prostoru. V praxi je možné se častěji setkat s umístěním mikrofonů na vrcholy geometrických útvarů. Pro ambisonii prvního řádu by to byl čtyřstěn [23].

Pro každý mikrofon je možné použít snímání pomocí formátu A i B. Formát A je způsob záznamu signálů z mikrofonů, v němž má každý mikrofon svůj kanál. U formátu B je způsob záznamu rozdílný. Jak už bylo uvedeno, využívá kanály $W(\mathrm{t})$ pro akustický tlak a $X(\mathrm{t})$, $Y(\mathrm{t})$, $Z(\mathrm{t})$ pro tlakové gradienty v osách souřadnicového systému označených jako *x, y, z*. Data z formátu A je možné převést na data ve formátu B [23].

Jestliže rozmístění mikrofonních kapslí odpovídá vrcholům kvádru, tedy nesměřují ve směrech akustických os, ale jsou uspořádány se 45 stupňovým posunem vzhledem k jednotlivým osám, je možné jednotlivé kapsle označit pomocí tří písmen. Písmena označují polohu jednotlivých mikrofonů vzhledem ke směru pohledu mikrofonů. Mikrofony je tedy možné označit jako LPH(levý-přední-horní), PPD(pravý-přední-dolní), LZD(levý-zadní-dolní), PZH(pravý-zadní-horní). 3D pohled na rozložení mikrofonních kapslí je na obrázku 2.5 [24].

Výsledné rovnice pro převod z formátu A do B pak vypadají následovně[24]:

$$W = s_{\mathrm{LPH}} + s_{\mathrm{PPD}} + s_{\mathrm{LZD}} + s_{\mathrm{PZH}}, \tag{2.44}$$

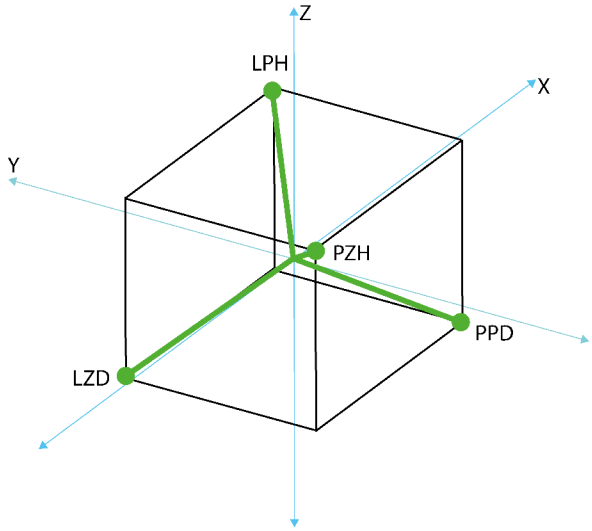
$$X = s_{\mathrm{LPH}} + s_{\mathrm{PPD}} - s_{\mathrm{LZD}} - s_{\mathrm{PZH}}, \tag{2.45}$$

$$Y = s_{\mathrm{LPH}} - s_{\mathrm{PPD}} + s_{\mathrm{LZD}} - s_{\mathrm{PZH}}, \tag{2.46}$$

$$Z = s_{\mathrm{LPH}} - s_{\mathrm{PPD}} - s_{\mathrm{LZD}} + s_{\mathrm{PZH}}. \tag{2.47}$$

Písmenem s je zde označen signál z mikrofonu a indexem poloha mikrofonu, která odpovídá obrázku 2.5.

Mezi hlavní výhody formátu B patří, že nezáleží na počtu zdrojů zvuku nebo typu a stačí přenášet jen čtyři kanály formátu B. Další výhodou je snadná manipulace např. z hlediska rotace, kdy stačí na původní kanály aplikovat rotační matici.



Obr. 2.5: 3D pohled na rozložení mikrofonních kapslí [24].

Lineární transformace, které nejsou závislé na prostorových vlastnostech, mezi něž spadá například filtrování, lze aplikovat standardním způsobem, ale za podmínky, že musí být aplikovány na všechny kanály. I nelineární transformace, mezi které patří například úprava dynamického rozsahu, je možné použít za podmínky, že budou aplikovány na všechny kanály stejně. Je možné využít i prostorové deformace. Mezi prostorové deformace patří třeba efekt zaostření na jeden směr zvukového pole [24].

2.4 Ambisonie vyšších řádů

Přidáním mikrofonních snímačů je možné docílit ambisonie vyšších řádů, které zabezpečují lepší směrové vlastnosti v porovnání s ambisonií 1. řádu. Vyšší řády rozkládají zvukové pole pomocí záznamu signálů z uskupení několika mikrofonů ve sférických harmonických funkcí. Sférické harmonické funkce jsou matematické funkce, které se používají k popisu zvukového pole na sférickém mikrofonním poli. Sférické harmonické funkce jsou matematicky popsány jako $Y_{\mathrm{lm}}(\phi, \theta)$. Kde l je řád ambisonie, m = -l, ..., l je přesně daný koeficient, ϕ udává úhel azimutu a θ udává úhel elevace. Nejvyšší použitý řád rozvoje l označuje zároveň i řád ambisonie. Každý jednotlivý řád má $2l + 1$ kanálů. Celkový počet kanálů pro ambisonii řádu l je dán vzorcem 2.29. Takže např. ambisonie 2. řádu ($l = 2$) má 9 kanálů [23].

Zvukové pole lze prezentovat jako součet zvukových polí vytvořených jednotlivými zdroji jako [23]

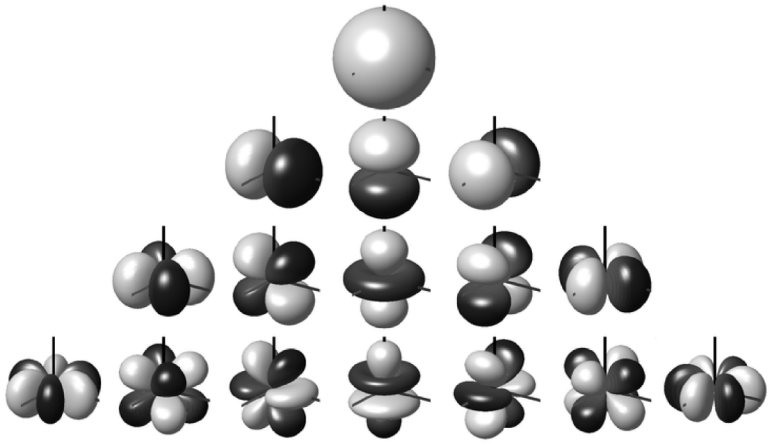
$$S(t, \phi, \theta) = \sum_{l=0}^{\infty} \sum_{m=-1}^{l} B_{\mathrm{lm}}(t)\, Y_{\mathrm{lm}}(\phi, \theta). \tag{2.48}$$

Kanály vyšších řádů ambisonie jsou totožné s $B_{\mathrm{lm}}(t)$ a obsahují již známé kanály *W, X, Y, Z*. $Y_{\mathrm{lm}}(\phi, \theta)$ je označení pro sférické harmonické funkce, které tvoří základ rozvoje.

Rozvoj zvukového pole v prostředí počátku souřadnicového systému je možné popsat pomocí sférických harmonických funkci jako [23]

$$p(\omega, \mathbf{r}) = \sum_{l=0}^{\infty} i^l j_{\mathrm{m}}(kr) \sum_{m=-l}^{l} B_{\mathrm{lm}}(\omega)\, Y_{\mathrm{lm}}(\phi, \theta), \tag{2.49}$$

kde $j_{\mathrm{m}}(kr)$ je označení pro Besselovy sférické funkce.



Obr. 2.6: 3D zobrazení sférických harmonických funkcí 0. až 3. řádu [23].

2.4.1 Kódování

Máme-li rovinnou vlnu, která vytváří signál $s(t)$, který přichází pod úhlem azimutu ϕ_{s} a úhlem elevace θ_{s}, a pokud je využito sférických harmonických funkcí, tak lze signál zakódovat jako [23]

$$B_{\mathrm{lm}}(t) = s(t)\, Y_{\mathrm{lm}}(\phi_{\mathrm{s}}, \theta_{\mathrm{s}}), \tag{2.50}$$

kde *s(t)* je výstupní signál z mikrofonu, Y_{lm} je koeficient příslušné harmonické funkce pro daný úhel elevace a azimutu, pod kterým je mikrofon umístěn na sféře.

Pro první řád ambisonie vyššího řádu by kódování do kanálů vypadalo následovně [23]

$$B_{0,0}(t) = \sqrt{2}W(t) = s(t), \tag{2.51}$$

$$B_{1,-1}(t) = \sqrt{3}Y(t) = s(t)\sqrt{3}\sin\phi_{\mathrm{s}}\cos\theta_{\mathrm{s}}, \tag{2.52}$$

$$B_{1,0}(t) = \sqrt{3}Z(t) = s(t)\sqrt{3}\sin\theta_{\mathrm{s}}, \tag{2.53}$$

$$B_{1,1}(t) = \sqrt{3}X(t) = s(t)\sqrt{3}\sin\phi_{\mathrm{s}}\cos\theta_{\mathrm{s}}. \tag{2.54}$$

V současnosti se pro kódování do ambisonie používá několik normalizací zisku a uspořádání kanálů. Mezi uspořádání zvukových kanálů patří FuMa, SID a ACN. Jako normalizace se využívají SN3D, N3D a maxN.

Uspořádání kanálů

FuMa (Furse–Malham) je uspořádání kanálu ambisonie až do třetího řádu, respektive do třetího řádu jsou označovány písmeny. Jsou zde přidány kanály druhého řádu (*RSTUV*) a třetího řádu *(KLMNOPQ)*. Rozšíření vyšších řádů jsou také definována, ale kvůli přehlednosti už nejsou označována písmeny a nejsou používána. Uspořádaní je znázorněno na obrázku 2.7 [25].

			W_0			
		Y_2	Z_3	X_1		
	V_8	T_6	R_4	S_5	U_7	
Q_{15}	O_{13}	M_{11}	K_9	L_{10}	N_{12}	P_{14}

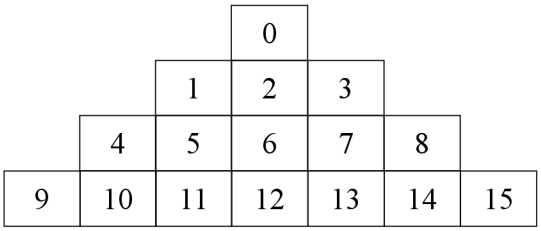
Obr. 2.7: Uspořádání FuMa [25].

SID (Single Index Designation) systém uspořádání, který je kompatibilní s 1. řádem uspořádání FuMa. Pořadí uspořádání následujících kanálů se liší. Názvy kanálů zůstávají na stejných pozicích jako u FuMa, ale od druhého řádu se mění jejich indexy. Uspořádání SID je na obrázku 2.8 [26].

			0			
		2	3	1		
	5	7	8	6	4	
10	12	14	15	13	11	9

Obr. 2.8: Uspořádání SID [26].

ACN (Ambisonics Channel Number) uspořádání kanálů ve spojení s normalizací zisku SN3D tvoří současný standard Ambix. Uspořádání ACN není kompatibilní ani s 1. řádem uspořádání FuMa nebo SID. Uspořádání ACN je na obrázku 2.9 [27].



Obr. 2.9: Uspořádání ACN [27].

Normalizace

Mezi používané normalizace zisku patří FuMa (maxN), N3D a SN3D. Pokud je zvuk zakódován pomocí jedné z normalizací, existují zde vztahy, pomocí kterých je možné jednu normalizaci převést na druhou. Je ale potřeba dodržet převod mezi stejnými kanály. Jednotlivá uspořádání s indexy kanálů, sférické harmonické funkce s normalizací zisku SN3D a s převodními koeficienty jsou níže 2.10.

řád	Uspořádání			Sférické harmonické funkce	Převodní koeficienty	
	ACN	FuMa	SID	SN3D	do N3D	do maxN
0	0	0	0	1	1	1/sqrt(2)
1	1	2	2	sin(a)cos(e)	sqrt(3)	1
1	2	3	3	sin(e)	sqrt(3)	1
1	3	1	1	cos(a)cos(e)	sqrt(3)	1
2	4	8	5	sqrt(3/4)sin(2a)(cos(e))^2	sqrt(5)	2/sqrt(3)
2	5	6	7	sqrt(3/4)sin(a)sin(2e)	sqrt(5)	2/sqrt(3)
2	6	4	8	(1/2)(3(sin(e))^2-1)	sqrt(5)	1
2	7	5	6	sqrt(3/4)cos(a)sin(2e)	sqrt(5)	2/sqrt(3)
2	8	7	4	sqrt(3/4)cos(2a)(cos(e))^2	sqrt(5)	2/sqrt(3)
3	9	15	10	sqrt(5/8)sin(3a)(cos(e))^3	sqrt(7)	sqrt(8)/5
3	10	13	12	sqrt(15/4)sin(2a)sin(e)(cos(e))^2	sqrt(7)	3/sqrt(5)
3	11	11	14	sqrt(3/8)sin(a)cos(e)(5(sin(e))^2-1)	sqrt(7)	sqrt(45)/32
3	12	9	15	(1/2)sin(e)(5(sin(e))^2-3)	sqrt(7)	1
3	13	10	13	sqrt(3/8)cos(a)cos(e)(5(sin(e))^2-1)	sqrt(7)	sqrt(45)/32
3	14	12	11	sqrt(15/4)cos(2a)sin(e)(cos(e))^2	sqrt(7)	3/sqrt(5)
3	15	14	9	sqrt(5/8)cos(3a)(cos(e))^3	sqrt(7)	sqrt(8)/5

Obr. 2.10: Uspořádání a normalizace Ambisonie [25][26][27][28][29].

3 Hardwarová sekce

V této části práce jsou popsány jednotlivé prvky signálového řetězce od samotných mikrofonů v kapitole 3.1 přes programovatelná hradlová pole v kapitole 3.2 a převodník na USB linku v kapitole 3.3, což vytváří hlavní hardwarový základ celého systému umožňujícího sběr, zpracování a přenos audio signálu v reálném čase.

3.1 MEMS mikrofony

MEMS (mikro-eletromechanický systém) mikrofony jsou elektro-akustické měniče, které obvykle obsahují MEMS mikrofon, ASIC (aplikačně specializovaný integrovaný obvod) a předzesilovač v jednom pouzdře. MEMS mikrofon převádí proměnný akustický tlak pomocí pohyblivé membrány na změny kapacity. Pohyblivá membrána je vyleptaná v křemíku. Obsahuje také stacionární perforovanou destičku, která společně s membránou tvoří kondenzátor. Využívá se zde podobný princip jako v kondenzátorových mikrofonech [30].

Integrovaný obvod tyto změny kapacity pak převádí na výstup pomocí analogových a digitálních převodníků. Na výstupu se nejčastěji využívá rozhraní pro digitální kódování [31].

Mezi hlavní výhody MEMS mikrofonů patří jejich všesměrové vlastnosti, malá spotřeba elektrické energie, vysoká mechanická odolnost proti nárazům a teplotám. Poslední vlastnost, která nemůže být opomenuta, je i jejich malá a kompaktní velikost.

3.1.1 Typy výstupního rozhraní

Výstupní signál MEMS mikrofonů může být buďto v analogové, nebo v digitální podobě. Z důvodu jednoduššího zapojení MEMS mikrofonů a následného zpracování signálu budou využity mikrofony s digitálním výstupem. Digitální MEMS mikrofony už v sobě mají integrovaný analogově – digitální (A/D) převodník přímo v jejich pouzdře. Na trhu existuje mnoho výrobců MEMS mikrofonů a každý výrobce do svých čipů integruje jiné výstupní digitální rozhraní. Nejrozšířenější typy jsou PDM (Pulse Density Modulation), TDM (Time Division Multiplexing) a I2S (Inter-IC Sound) [32].

V případě mikrofonů s analogovým výstupem by do konstrukce musely být zabudovány A/D převodníky kvůli následnému digitálnímu zpracování. Jelikož se v práci počítá s 32 mikrofony, použití mikrofonů s analogovým výstupem by vedlo k nárůstu rozměrů desek plošných spojů, vzrostl by počet signálových vodičů a vzrostla

by náročnost konstrukce. Z toho důvodu pro tuto práci byly přiděleny mikrofony s digitálním rozhraním TDM.

TDM

TDM je používán v případech, kdy je potřeba dva a více datových kanálů přenášet po jedné datové lince. TDM rozhraní použitých mikrofonů dokáže přenést po jedné datové lince až 16 datových kanálů. Funguje na principu, že každý datový kanál na datové lince má vyhrazený slot pro přenos dat. Tento slot tvoří $1/N$ šířky rámce, kde N je počet kanálů, které je požadováno přenášet [33].

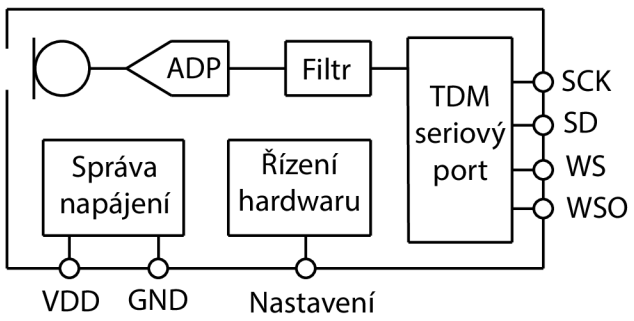
Hodinový signál pro TDM rozhraní je obvykle implementován jako puls o šířce jednoho bitu. Jeho frekvence závisí na počtu datových kanálů a vzorkovací frekvenci. Pokud bude zvolena vzorkovací frekvence $\mathrm{f_{vz}} = 48\,\mathrm{kHz}$, datový rámec pro jeden mikrofon bude mít 32 bitů a v zapojení je maximální počet mikrofonů n, což je 16. Frekvence hodinového signálu f_{clk} se vypočte z následujícího vzorce [33]

$$f_{\mathrm{clk}} = n \cdot 32 \cdot \mathrm{f_{vz}} = 16 \cdot 32 \cdot 48000 = 24,576\ [\mathrm{MHz}]. \tag{3.1}$$

TDM rozhraní nemá přesně definovaný standard, takže se implementace u jednotlivých výrobců můžou lišit a nemusí být mezi sebou kompatibilní [32]. Kvůli tomu, že TDM rozhraní umožňuje přenos 16 kanálů zvukového signálu, je vhodné použít v této práci právě mikrofony s TDM rozhraním ve dvou oddělených linkách pro 32 mikrofonů.

3.1.2 MEMS mikrofon ICS-52000

Pro tuto práci byly přiděleny mikrofony ICS-52000 s výstupním digitálním rozhraním TDM. Tento mikrofon je složen ze senzoru MEMS, obvodů pro úpravu signálu, analogově-digitálního převodníku, anti-aliasingového filtru, napájecích obvodů a 32 bitového TDM rozhraní. Blokový diagram mikrofonu je na obrázku 3.1 [33].



Obr. 3.1: Blokový diagram mikrofonu ICS-52000 [33].

Implementované TDM rozhraní podporuje zapojení až s 16 mikrofony ICS-52000, které mohou být přímo připojeny k mikrokontrolerům nebo signálovým procesorům bez použití dodatečných periferií, jako jsou audiokodeky. Díky jejich vlastní synchronizaci jsou vhodné pro použití v mikrofonních polích [33].

Výrobce udává, že tolerance citlivosti mikrofonu je ± 1 dB. Další výhodu jsou i kompaktní malé rozměry. Mikrofony mohou fungovat na různých vzorkovacích frekvencích od 8 kHz až do 48 kHz a v různých konfiguracích, co se týče počtu. Pro různé konfigurace mikrofonů z hlediska počtu i z hlediska použité vzorkovací frekvence je pro mikrofony potřeba přesně nakonfigurovat řídící signály. Mikrofony mají dva hlavní řídící signály *SCK* a *WS*. *SCK* je hodinový signál pro mikrofony a *WS* je synchronizační signál. Perioda *WS* signálu musí být dle dokumentace [33]

$$WS = \frac{1}{f_{\text{vz}}}[\text{s}]. \tag{3.2}$$

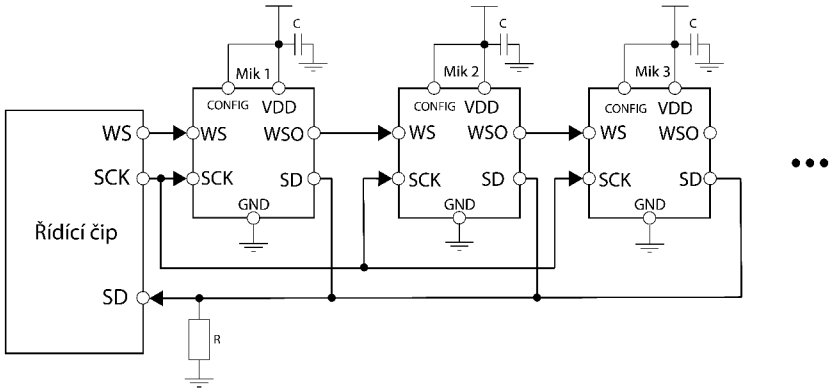
Na signál *WS* se váže signál *WSO*, který je generován na výstupních pinech mikrofonů samotnými mikrofony. Jde o zřetězený signál *WS*, kterým si mezi sebou mikrofony předávají synchronizační impuls pro zajištění synchronizace okamžiků pro odesílání dat na společnou datovou sběrnici.

Hodinový signál *SCK* je závislý na použitém počtu mikrofonů a na vzorkovací frekvenci. Frekvence signálu *SCK* je dána vzorcem [33]

$$SCK = M \cdot 32 \cdot f_{\text{vz}}\,[\text{Hz}], \tag{3.3}$$

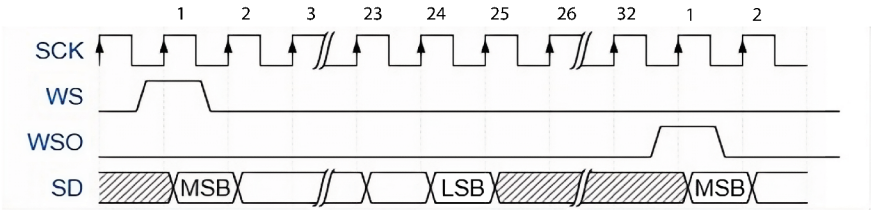
kde M je počet mikrofonů a jde o mocninu čísla 2.

Výstupní data z mikrofonů mají délku 24 bitů na kanál, jsou ve formátu dvojkového doplňku a na sběrnici jsou vysílány z výstupu *SD* (Serial Data) od nejvýznamnějšího bitu (MSB) v závislosti na náběžné hraně hodinového signálu *SCK*. Zapojení jednotlivých mikrofonů je na obrázku 3.2[33].



Obr. 3.2: Zapojení mikrofonů ICS-52000 [33].

Průběhy signálů v časové doméně jsou zobrazeny na obrázku 3.3. Z průběhů je patrné, že aby mikrofon začal vysílat data na společnou sériovou datovou sběrnici *SD*, musí první přijít *WS* synchronizační impuls na vstup mikrofonu a během něj musí dojít k náběžné hraně hodinového signálu. Podrobnější parametry mikrofonu lze nalézt v katalogovém listu výrobce [33].



Obr. 3.3: Časový diagram jednoho TDM slotu [33].

3.2 Programovatelná hradlová pole

Programovatelná hradlová pole nebo také FPGA jsou digitální integrované obvody, které jsou založeny na matici konfigurovatelných logických bloků, které jsou mezi sebou propojovány pomocí programovatelných propojení. U těchto bloků je možné konfigurovat jednotlivá propojení mezi dalšími logickými bloky a díky těmto základním elementům je možné FPGA naprogramovat tak, aby prováděla nejrůznější a i velice komplexní úlohy [34].

3.2.1 Použití FPGA

Dnešní FPGA obsahují miliony hradel a můžou obsahovat i mikroprocesorová jádra, vysokorychlostní vstupně/výstupní obvody, atd. Často jsou použity i v komunikačních zařízeních, rádiích, radarech a v DSP (Digital Signal Processing) aplikacích. Dnešní FPGA se často používají k implementaci různých návrhů integrovaných obvodů [34].

FPGA lze najít i v aplikacích pro vysokorychlostní zpracování signálů, kde se obvykle používají digitální signálové procesory. Čipy FPGA mohou obsahovat násobičky, aritmeticko-logické jednotky, velké paměti RAM a mohou dosahovat velké paralelizace zpracování dat [34].

3.2.2 Periferie

Kvůli širokým možnostem použití FPGA, které záleží hlavně na návrhu designu, je možné využít s FPGA širokou škálu periferií. Je možné použít různé zobrazovací

jednotky, DA/AD převodníky, čidla, snímače, ovládací prvky a další. Velkou výhodou FPGA je, že pokud daná periferie potřebuje nějaké rozhraní nebo převodník, je možné ho implementovat přímo do designu FPGA [34].

3.2.3 Komunikační rozhraní

V závislosti na použitém vývojovém kitu, který je popsán v kapitole 3.2.4, v této kapitole budou popsána jen rozhraní, kterými disponuje použitý vývojový kit.

Join Test Action Group (JTAG) je komunikační standard, který se využívá pro programování, ladění chování programovaných čipů. Využívá signály *TCK*, *TMS*, *TDI* a *TDO*. *TCK* je hodinový signál rozhraní, se kterým jsou ostatní signály synchronní. Signál *TMS* je pro řízení tzv. TAP řadiče, což je zjednodušeně stavový automat. Signály *TDI* (Test Data In) a *TDO* (Test Data Out) jsou vstupní a výstupní datové signály [35]. JTAG může dosahovat rychlosti přenosu až 30 Mbit/s [36].

Universal Asynchronous Receiver Transmitter (UART) je sériový komunikační protokol, který ke komunikaci využívá dva vodiče *Rx* a *Tx*. Využívá se pro komunikaci mezi mikroprocesory, mikrokontrolery, FPGA a počítačem. Zařízení s UART v roli vysílače převádí paralelní data na sériová a následně je odesílá po komunikační lince. Zařízení, které je v roli přijímače, sériová data přijímá a následně je paralelizuje. UART zároveň zajišťuje časování a další potřebné funkce pro komunikaci. Rychlost komunikace přes UART je standardně 115200 bit/s, ale může dosahovat i vyšších rychlostí [37].

3.2.4 DIGILENT Arty S7

Pro tuto práci byl přidělen vývojový kit od firmy DIGILENT, model Arty S7-25. Na tomto kitu je FPGA čip od firmy Xillinx Spartan-7, což je výkonné, cenově dostupné FPGA, které je kompatibilní s vývojovým prostředím Vivado Design Suite, které bude využito pro naprogramování čipu FPGA společně s jazykem VHDL. Tento vývojový kit obsahuje velké množství vstupně/výstupních obvodů, 256 MB DD3 paměť, 16 MB Flash paměť. Rozhraní JTAG, UART. Arty S7-25 má 100 MHz a 12 MHz zdroje hodinového signálu a speciální bloky MMCM (Mixed Mode Clock Manager) a PLL (Phase Locked Loop) pro úpravu frekvence a fáze hodinového signálu, které budou využity pro vytvoření řídících signálu jak pro FPGA, tak pro MEMS mikrofony, viz kapitola 4.1. Více parametrů je možné najít ve výrobním listu [38].

V této práci bude FPGA čip sloužit primárně jako převodník mezi TDM rozhraním mikrofonů a USB rozhraním počítače pro přenos dat a řízení mikrofonů. Dále

bude generovat hodinový signál a synchronizační signály pro MEMS mikrofony dle parametrů mikrofonů.

3.3 Převodník FTDI

Původní návrh komunikace mezi FPGA a počítačem, který byl implementován v rámci semestrální práce, na kterou tato diplomová práce navazuje, spočíval v implementaci IP jádra Data Capture, které bylo vygenerováno rozšířením HDL Verifier pro Matlab a také se počítalo s použitím jen 16 mikrofonů. Toto jádro využívalo komunikaci přes rozhraní JTAG a bylo poměrně jednoduché na implementaci. V rámci semestrální práce ale bylo zjištěno, že přenos dat tímto způsobem není vhodný, protože přenos dat i při nastavení nejvyšší možné frekvence JTAG rozhraní z FPGA do Matlabu byl příliš pomalý. Implementace nezvládala přenést ani data ze dvou mikrofonů.

Vývojový kit s FPGA obsahuje čip FTDI 2232H [38], který dle dokumentace výrobce [39] disponuje režimem synchronního FIFA 245. Režim Sync FIFO 245 umožňuje přenosovou rychlost až 40 MB/s, což je pro účely této práce více než dostatečné, protože 32 mikrofonů vyprodukuje 6,25 MB/s dat. Proto, aby mohl být tento režim využit, musí být čip FTDI připojen k FPGA specifickými piny. Vývojový kit, který byl přidělen k této práci s čipem FTDI 2232H, se jeví, že má na vývojovém kitu zapojení pouze pro režim JTAG nebo UART [38]. Výrobce kitu má i ve své dokumentaci schéma zapojení této části skryto pod záminkou, že jde "know–how". Možnosti namapovaní signálů na tyto piny nejsou ani v XDC souboru vývojového kitu, tudíž není možné tuto variantu využít.

Po důkladné rešerši dostupných možností a konzultaci byla zvolena varianta vývojového kitu UM232H-B FTDI. Jde také o FTDI čip, který rovněž disponuje zmíněným režimem synchronního FIFA. Další jeho výhodou je to, že je použitelný na velké škále operačních systémů přes Windows, Linux a i Mac OS [40].

4 Softwarová sekce

Tato část práce se bude zabývat programováním FPGA v jazyce VHDL v kapitole 4.1. Přesněji implementací generování periodických řídících signálů v kapitole 4.1.1, vytvořením základního hodinového signálu v kapitole 4.1.1, příjmem dat z mikrofonů v kapitole 4.1.2 a řízením samotného TDM rozhraní v kapitole 4.1.3. Dále implementací komunikačního rozhraní FTDI v režimu synchronního FIFA v kapitole 4.1.4. Pro funkční komunikaci je potřeba naprogramovat i samotné FTDI232, to je popsáno v kapitole 4.2.

Následně naprogramováním aplikace pro počítač v prostředí Matlab pro zpracování dat až z 32 mikrofonů ze sférického mikrofonního pole v kapitole 4.3. Pro přehlednější nahlížení do kódů je doporučeno nahlížet přímo do zdrojových souborů, které jsou součástí elektronické přílohy.

4.1 Návrh a implementace funkcionalit pro FPGA

Pro naprogramování programovatelného hradlového pole bude využito vývojové prostředí AMD Vivado Design Suite ML Edition ve verzi 2023.2 společně s jazykem pro popis chování hardwaru VHDL.

4.1.1 Generování periodických řídících signálů

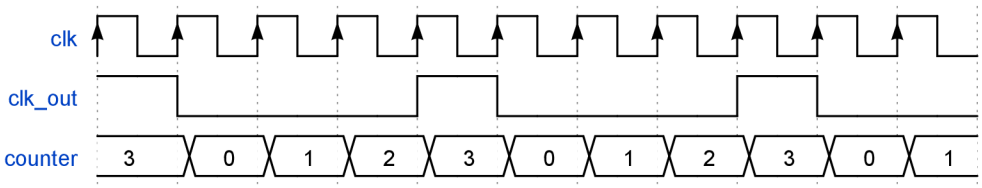
Pro generování periodických signálů, jako jsou například řídící signály *SCK* a *WS* pro mikrofony s definovanou frekvencí f, budou využity dva základní způsoby. Prvním způsobem je implementace děličky hodinového signálu, pomocí které je možné získat požadovanou nižší frekvenci f_2. Tuto metodu lze použít pouze v případě, že původní frekvence f je s požadovanou frekvencí f_2 synchronní. To znamená, že výstupní frekvence je fázově zarovnána se vstupní frekvencí. Tuto operaci lze popsat i vzorcem 4.1

$$f_2 = \frac{f}{n}[\text{Hz}], \tag{4.1}$$

kde n musí být celočíselný dělitel původní frekvence f.

Jde o způsob, jak z hodinového signálu FPGA, který je v tomto případě 100 MHz, udělat například hodinový signál s frekvencí 25 MHz. Tyto dvě frekvence jsou vůči sobě synchronní a je možné použit děličku pro získání hodinového signálu 25 MHz. Pokud by byla požadována frekvence například 24,576 MHz, tak tyto frekvence jsou vůči sobě asynchronní a pomocí jednoduché děličky není možné vytvořit signál s frekvencí 24,576 MHz. Dělička hodinového signálu funguje následovně.

Reaguje na náběžnou hranu vstupního hodinového signálu a aktuální počet hran si ukládá do paměti. Pokud počet načítaných hran odpovídá zadanému dělícímu poměru *DIV_FACT*, dělička na výstup nastaví signál do úrovně logické 1 a hodnota čítače *counter* se vynuluje, aby mohlo dojít ke zpracování další periody. Při další náběžné hraně se i logická úroveň výstupního signálu nastaví na logickou 0. Funkce je patrná i z časového diagramu na obrázku 4.1.



Obr. 4.1: Časový diagram děličky.

Druhou možností je využít speciální bloky v čipu FPGA, jako jsou bloky MMCM a PLL. Tyto bloky slouží ke generování hodinových signálů s definovanou frekvencí a fází. Lze je využít i pro generování asynchronních hodinových signálů. Tyto bloky je možné implementovat buď pomocí kódu, nebo v podobě IP jádra Clocking Wizard, které je součástí vývojového prostředí Vivado. IP jádra poskytují možnost rychle implementovat složité funkce do FPGA designů bez nutnosti navrhovat je od základů. Pro příklad je zde možné nalézt IP jádra zahrnující procesory, paměťové řadiče, rozhraní, jako jsou Ethernet nebo USB, a další komplexní bloky pro datové zpracování [41].

Clocking Wizard umožňuje použít buď bloky PLL, nebo MMCM. Počet těchto bloků na čipu je značně omezen. U desky Digilent Arty S7-25 výrobce uvádí, že na čipu jsou jen 3 bloky MMCM a 3 PLL [38]. Clocking Wizard má několik nastavitelných parametrů, jako je například vstupní a výstupní frekvence, fáze výstupního signálu, střídy a další parametry, které jsou uvedeny v dokumentaci [42].

Pomocí Clocking Wizard není možné vytvořit signály s jakoukoliv frekvencí. Má značná omezení například ve vstupní frekvenci, která je v rozsahu od 10 MHz do stovek MHz, takže není například možné jednoduše vytvořit signál s frekvencí 48 kHz, což je standardní vzorkovací kmitočet zvukového signálu. K vytvoření signálu s takovou frekvencí jen pomocí Clocking Wizard by muselo být použito kaskádní zapojení čítačů MMCM [42]. Je potřeba použít kombinaci více metod. Často není možné vytvořit přesně požadovanou frekvenci na výstupu ani v rámci MHz a je potřeba nalézt synchronní frekvenci, kterou Clocking Wizard dokáže vytvořit, a zároveň jde o synchronní frekvenci s požadovanou výstupní frekvencí. Například není možné vytvořit signál s přesnou frekvencí 24,576 MHz, což je frekvence signálu *SCK*, která je vyžadována pro 16 mikrofonů. Umožňuje vytvořit synchronní frekvenci 12,288 MHz,

kterou pak stačí zdvojnásobit pro získání frekvence 24,576 MHz. Clocking Wizard je zdarma dostupný ve standardní knihovně IP jader ve Vivado, pokud je možné jádro implementovat na dané FPGA [42].

Vytvoření základního hodinového signálu pro FPGA

Jak již bylo zmíněno FPGA musí pro mikrofony generovat signál *WS*, který má frekvenci 48 kHz. Frekvence 48 kHz je zároveň i standardní vzorkovací frekvence pro zvukové signály.

Je možné použít děličku frekvence na základní frekvenci 100 MHz. Při jejím použití ale nebude získán přesný a standardní kmitočet pro audio data a muselo by se v části pro zpracování dat použít převzorkování, které může být ztrátové a hlavně výpočetně náročné a pro použití v této práci nevýhodné.

Hodinový signál *SCK*, který mikrofony potřebují ke svému chodu, musí být s touto frekvencí synchronní. Frekvence *SCK* pak odpovídají frekvencím z tabulky 4.1. Pro vytvoření těchto frekvencí je potřeba využít obě výše zmíněné metody, protože frekvence signálu *SCK* a ani *WS* nejsou synchronní s interním hodinovým signálem 100 MHz, který je na vývojovém kitu.

Tab. 4.1: Přehled frekvencí SCK [33].

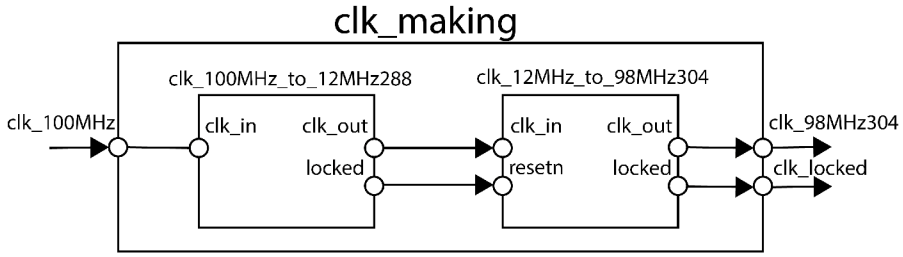
Počet mikrofonů	SCK [MHz]
1-2	3,072
3-4	6,144
5-8	12,288
9-16	24,576

Pro tuto část budou použity dvě IP jádra Clocking Wizard s bloky MMCM. První jádro *clk_wiz_100MHz_to_12MHz288* je použito k převedení vstupního hodinového signálu s frekvencí 100 MHz na frekvenci 12,288 MHz. Frekvence 12,288 MHz je jediná frekvence z tabulky 4.1, kterou je IP jádro schopno přesně vytvořit. U ostatních frekvencí vytvoří jen přibližnou frekvenci, která téměř odpovídá požadované frekvenci, ale takovou výstupní frekvenci není možné využít.

Další IP jádro Clocking Wizard *clk_wiz_12MHz288_to_98MHz304* je použito pro převedení výstupního hodinového signálu s frekvencí 12,288 MHz z výstupu předchozího jádra na frekvenci 98,304 MHz. V tomto převodu už IP jádro nemá problém s dosažením přesné frekvence. Tento hodinový signál je synchronní se všemi frekvencemi z tabulky 4.1 a taky s frekvencí signálu *WS* 48 kHz. Schválně byla vytvořena vyšší frekvence hodinového signálu než 24,576 MHz, protože v designu pro TDM

rozhraní, který bude popsán v další kapitole, bylo potřeba využít vyšší časové rozlišení. Hodinový signál s frekvencí 98,304 MHz je pak v celém návrhu využíván jako primární, ze kterého jsou odvozovány i nižší frekvence.

Kvůli přehlednosti v projektu byl vytvořen modul *clk_making*. Neobsahuje žádnou logiku, jen mezi sebou propojuje dvě výše zmíněná IP jádra a zapouzdřuje porty, které nejsou důležité pro použití v dalších modulech. Má jeden vstupní port *clk_100MHz* pro interní hodinový signál FPGA a výstupní porty *clk_98MHz304* pro nově vytvořený hodinový signál s požadovanou frekvencí a port *clk_locked*. U jader jsou také využity porty *reset* a *locked*. Jádra od samého počátku negenerují přesně stanovenou frekvenci a trvá jim nějakou dobu, než se na dané frekvenci ustálí. Proto se nejprve inicializuje první jádro, jakmile dosáhne požadované frekvence, jeho výstup *locked* je nastaven do log. 1 a tím se zruší *resetn* na druhém jádře a začne se inicializovat. Jakmile dosáhne požadované frekvence, jeho výstupní port *locked* je nastaven do log. 1. Zbytek řídící logiky na FPGA čeká, dokud není výstupní port *clk_locked* aktivní. Zapojení *clk_making* je na obrázku 4.2.

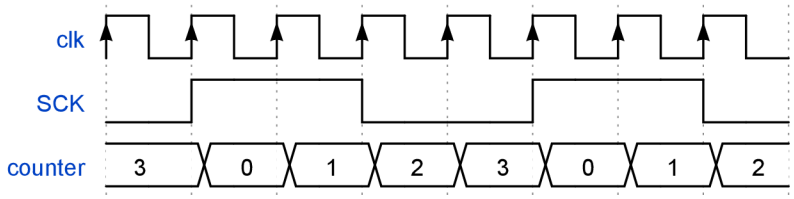


Obr. 4.2: Zapojení clk_making.

Generování signálu SCK pro mikrofony

Frekvence signálu SCK pro mikrofony musí odpovídat přesně stanovené frekvenci z tabulky 4.1. Jako další parametr, který je uveden v dokumentaci mikrofonů, je požadována střída signálu 48 až 52 %[33]. Pro vytvoření požadované frekvence pro daný počet mikrofonů byla použita jednoduchá dělička frekvence s rozšířením o adaptivní délku kladné a záporné půlperiody v závislosti na použitém počtu mikrofonů. Dělička s proměnným dělícím poměrem byla potřebná pro testování a ladění. Pro tento účel byla vytvořena entita *clk_out_mic*, která má vstupní port *clk* pro hodinový signál. V tomto případě pro 98,304 MHz. Dále vstup *clk_en*, který slouží pro povolení generování hodinového signálu, a vstup *clk_div*, který udává dělící poměr vstupního hodinového signálu. Také obsahuje výstupní port *clk_out* pro výstupní hodinový signál, který je upraven děličkou. Vstupní port *clk_div* standardně nabývá

hodnot 4, 8, 16 a 32. Celý proces je synchronizován s náběžnou hranou hodinového signálu *clk*. Časový diagram je na obrázku 4.3.



Obr. 4.3: Časový diagram generování SCK.

Pokud je připojeno 16 mikrofonů, *clk_div* je 4 a signál *clk_out* má frekvenci 24,576 MHz. Princip funkčnosti děličky byl již popsán výše, ale v tomto případě je funkcionalita děličky upravena pro generování signálu s 50 % střídou, takže dělička reaguje i na počet náběžných hran, který odpovídá polovině periody signálu. V závislosti na aktuální hodnotě čítače, jestli se nachází před polovinou periody, nebo za polovinou periody, je nastavována logická úroveň na výstupním portu. Část VHDL kódu této entity je v příloze B.2.

Generování signálu SCK pro FPGA

Jelikož s frekvencí, s jakou pracují mikrofony, musí být synchronní i další procesy na FPGA, bylo potřeba vytvořit i děličku hodinového signálu pro signál, který bude využitelný uvnitř FPGA pro další procesy, které musí být synchronní. Tento hodinový signál je vytvořen v entitě *clk_out_FPGA* viz. příloha B.3.

Tato dělička je implementována obdobně jako v předchozím případě pro signál SCK pro mikrofony, ale s tím rozdílem, že zde nebylo potřeba dosáhnout 50% střídy hodinového signálu, protože ta by v FPGA logice způsobila chyby, které by se dodatečně musely ošetřit.

Entita má opět vstupní port *clk* pro hodinový signál. V tomto případě jde o signál s frekvencí 98,304 MHz. Dále vstup *clk_en*, který slouží pro povolení generování hodinového signálu a vstup *clk_div*, který udává dělící poměr vstupního hodinového signálu. Dále také obsahuje výstupní port *clk_out* pro vytvářený hodinový signál. Pokud generování hodinového signálu není povoleno, na výstupu je logická 0.

Generování signálu WS

Další signál, který musí FPGA generovat pro správné fungování mikrofonů, je signál *WS*. Tento signál slouží k synchronizaci datových rámců, které produkují samotné mikrofony[33]. Dle dokumentace mikrofonů [33] signál *WS* musí mít frekvenci, která

odpovídá vzorkovací frekvenci. Vzorkovací frekvence je použita 48 kHz. Dále u mikrofonů ihned po přivedení napájení a hodinového signálu *SCK* nastane počáteční inicializace. Během této počáteční inicializace nesmí být generován signál *WS*. Signál *WS* může být generován až po uplynutí minimálně 10 ms. WS signál je vzorkován při náběžné hraně SCK a klesající hrana WS může přijít kdykoli před začátkem další náběžné hrany [33]. Signál *WSO*, který je generován mikrofonem, je o 32 hodinových taktů zpožděný *WS* a slouží jako vstupní *WS* signál pro další mikrofon v řetězci. O generování signálu *WSO* se starají samotné mikrofony a generování *WSO* tedy není součástí implementace v FPGA.

Pro generování *WS* signálu byla vytvořena entita *ws_mic*. Tato entita má dva generické parametry pro *startup* a *div_fact_WS*. Parametr *startup* udává počet taktů hodinového signálu pro počáteční inicializaci, během které nesmí být signál *WS* generován. Parametr *div_fact_WS* udává dělící poměr pro získání signálu *WS* s frekvencí 48 kHz. Parametr *startup* je roven 1081344 taktům, protože doba pro počáteční inicializaci t byla zvolena 11 ms a vstupní hodinový signál *clk* je 98,304 MHz. Počet taktů lze vypočítat pomocí vzorce 4.2

$$x = \frac{t}{T} = \frac{0,011}{(1.01725 \cdot 10^{-8})} = 1081344[-], \tag{4.2}$$

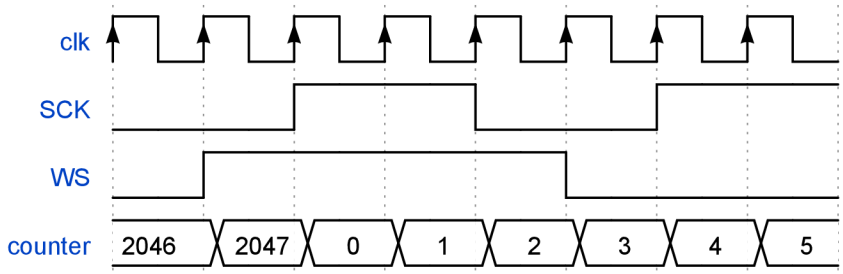
kde x je počet taktů hodinového signálu *clk* a T je perioda hodinového signálu.

Entita *ws_mic* má také několik vstupních a výstupních portů. Vstup *clk* je vstup pro hodinový signál 98,304 MHz. Port *length* je vstup, pomocí kterého je udávána délka synchronizačního impulzu *WS* v hodinových taktech, jehož délka je závislá na frekvenci hodinového signálu pro mikrofony *SCK*. Vstupní port *WS_enable* slouží k povolení generování signálu *WS* a port *WS_counter_enable* slouží k povolení počátečního odpočtu 11 ms kvůli počáteční inicializaci mikrofonů. Výstupní port *starUpEnd* slouží pro signalizaci, že doba na počáteční inicializaci mikrofonů uběhla a výstupní port *WS* je výstup pro generovaný synchronizační signál *WS*, který je následně přiveden na vstupní pin prvního mikrofonu v řetězci.

Funkcionalita této entity viz příloha B.4 je rozdělena do dvou procesů. První proces *startUp_WS* se stará o počáteční odpočet 11 ms. Jde o jednoduchý čítač *counter_1*, který jen počítá náběžné hrany hodinového signálu. V případě, že počet načítaných náběžných hran odpovídá počtu hran pro 11 ms, je výstup entity *startUpEnd* nastaven na úroveň log. 1 z důvodu signalizace pro řídící logiku, že je fáze *startup* dokončena. V opačném případě, kdy odpočet pro inicializaci není povolen, nebo pokud nebyl načítán daný počet náběžných hran, na výstupním portu entity je nastavena log. 0.

Druhý proces zajišťuje generování signálu *WS* s požadovanou frekvencí. Je zde využit opět princip jednoduché děličky hodinového signálu s rozšířením, které zajiš-

tuje proměnlivou délku impulsu v závislosti na frekvenci signálu *SCK*. Proces generování signálu *WS* závisí na vstupním portu *WS_enable*. Jelikož je *clk* 98,304 MHz, je potřeba tuto frekvenci vydělit *div_fact_WS*, který je roven 2048 pro získání frekvence 48 kHz. Signál *WS* je nastaven na úroveň log. 1 o jeden hodinový takt dříve, než přijde náběžná hrana hodinového signálu pro mikrofony *SCK* viz 4.4, protože synchronizace mikrofonů nastává v okamžiku, kdy je signál *WS* roven log. 1 a přijde náběžná hrana signálu *SCK*. Následně je kontrolována délka signálu *WS* v úrovni log. 1, protože je požadováno, aby byl *WS* opět roven log. 0 až jeden hodinový takt po sestupné hraně signálu *SCK*. Délka impulsu je entitě předávána pomocí vstupního portu *length*. Se snižující se frekvencí *SCK* bude počet hodinových taktů, kdy bude *WS* rovno log. 1, narůstat.



Obr. 4.4: Časový diagram generování WS.

4.1.2 Příjem dat z mikrofonů

Mikrofony, které jsou použity v této práci, produkují data v digitální podobě s použitím TDM rozhraní [33]. Všechny mikrofony používají pro přenos dat do FPGA jen jednu společnou datovou sběrnici. To má značnou výhodu při řešení vodivých cest pro přenos signálu, protože nemusí být použito 16 na sobě nezávislých sběrnic pro 16 mikrofonů. Díky synchronizaci pomocí signálu *WS* mikrofony přesně detekují, kdy mohou na sběrnici vysílat digitální data a kdy je naopak sběrnice obsazená vysíláním jiného mikrofonu z řetězce. Každý mikrofon má přidělený rámec pevné délky, ve kterém může vysílat data. Rámce mají délku 32 bitů, z nichž 24 bitů jsou data z mikrofonu a zbylých 8 bitů je vyplněno přesně nespecifikovanými daty [33]. Data jsou ve formátu dvojkového doplňku a jsou odesílána na sběrnici v pořadí od nejvýznamnějšího bitu po nejméně významný [33].

Pro implementaci je tedy důležité zaznamenat prvních 24 bitů a zbylých 8 se může vynechat, protože již nejde o potřebná data. Data jsou odesílána se stejnou frekvencí, jako je frekvence mikrofonního hodinového signálu *SCK*, a na stejné frekvenci tedy musí pracovat i příjem dat do FPGA. Další vlastnost mikrofonů, která

je důležitá při datovém přenosu, je doba, za jakou budou na sběrnici platná data od příchodu náběžné hrany hodinového signálu *SCK*. Dle dokumentace výrobce mikrofonů budou na sběrnici platná data v nejhorším případě do 18 ns od náběžné hrany signálu *SCK* [33]. S tím je nutné počítat a okamžik, ve kterém budou data vzorkována ze sběrnice, bude muset být alespoň o tuto časovou prodlevu zpožděn. Je vhodné do TDM rozhraní také implementovat logiku, která jednotlivé bity bude skládat do 24 bitových slov a následně odesílat k dalšímu zpracování. Mikrofony prvních 85 ms odesílají neplatná data a i to je potřeba zahrnout do implementace [33]. Jelikož sférické mikrofonní pole, které bude v této práci použito, má 2 linky po 16 mikrofonech, je zde možnost příjmu dat v režimu se dvěma linkami po 16 mikrofonech. Proto je zde potřeba implementovat sběr dat z obou linek současně.

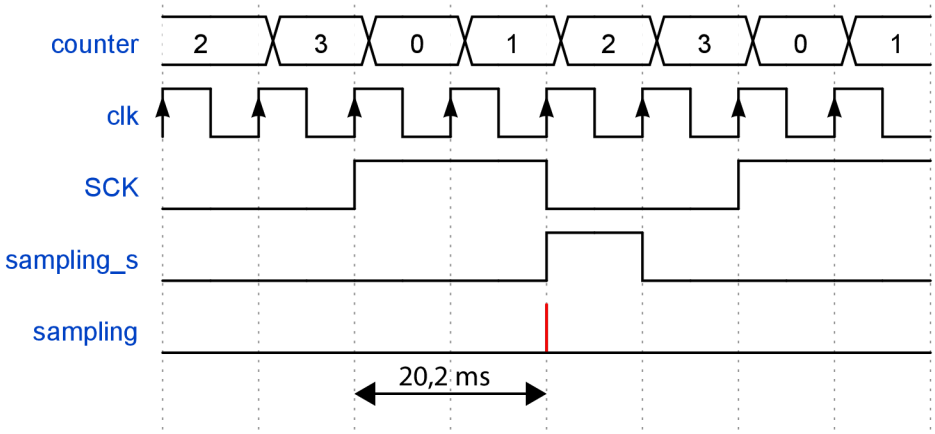
Pro příjem dat z mikrofonů byla vytvořena entita *data_from_mic*. Tato entita má několik vstupních a výstupních portů. Port *clk* je pro hodinový signál. Jelikož jde o záznam dat z mikrofonů, musí zde být připojen opět hodinový signál 98,304 MHz, aby bylo dodrženo, že frekvence jsou vůči sobě synchronní. Port *clk_fpga* je pro hodinový signál s frekvencí, která odpovídá frekvenci hodinového signálu *SCK*, jen nemá střídu 50%. Port *data_in_1* a *2* jsou porty pro příjem sériových dat z datových sběrnic, které jsou připojeny na piny FPGA. Další port *process_en* slouží k povolení funkcionality celého modulu. Je zbytečné, aby v případě, když nejsou generovány signály *SCK* a *WS*, nebo jsou na sběrnici nulová data, probíhalo vzorkování dat. Port *clk_div* opět přenáší informaci o dělícím poměru pro děličku hodinového signálu. Výstupní port *data_out* je port pro odesílání již navzorkovaných a uspořádaných dat do 32 bitových slov pro další zpracování z obou sběrnic. Port *data_ready* slouží pro signalizaci dalším entitám, že bylo navzorkováno všech 24 bitů z jednoho mikrofonu a je možné s nimi dále pracovat. Porty *in_1_en* a *in_2_en* jsou řídící signály pro rozhodnutí, ze které sběrnice vzorkovat data.

Implementace příjmu dat z TDM rozhraní je rozdělena do pěti procesů. VHDL kódy procesů jsou v příloze B.7 a B.8. Proces *non_zero_data* viz příloha B.7 slouží pro počítání hran hodinového signálu, kde počet hran odpovídá 85 ms dle výrobního listu [33]. Po tuto dobu mikrofony generují nulová/nevalidní data a ty není žádoucí zaznamenat a přenášet. Pokud čítač *zero_data_counter* napočítá odpovídající počet hran, nastaví signál *valid_data* na *true* a tím je v následujícím procesům povoleno odesílání platných dat do paměti. Počáteční odpočet je spuštěn pokaždé znovu, když řídící stavový automat opustí stav *Idle*.

Druhý proces (příloha B.7) *sampling_signal_gen* zajišťuje generování vzorkovacího impulzu. Proces funguje opět jako jednoduchá dělička hodinového signálu, která s každou náběžnou hranou hodinového signálu *clk* počítá počet náběžných hran, který je uložen do proměnné *samp_counter*. Kolikrát má být frekvence hodinového signálu podělena, opět vyjadřuje signál *clk_div*, který může nabývat hodnot

4, 8, 16 a 32. Rozdíl oproti jednoduché děličce je v implementaci generování vzorkovacího impulzu *sampling_s*, jehož zpoždění od signálu *SCK* může být docela přesně nastaveno vzhledem k časovému rozlišení frekvence hodinového signálu *clk*. Proto bylo potřeba vytvořit v modulu *clock_making* hodinový signál s vyšší frekvencí. Jak je uvedeno v dokumentaci mikrofonů [33], platná data mohou být na sběrnici v nejhorším případě se zpožděním 18 ns od náběžné hrany hodinového signálu. Perioda jednoho hodinového taktu 98,304 MHz je přibližně 10,1 ns. Proto vzorkovací impuls bude muset mít zpoždění 20,2 ns, protože 18 ns není možné touto technikou dosáhnout.

Zde se tedy vzorkovací signál vygeneruje až po dvou hodinových taktech signálu *clk* od náběžné hrany signálu *SCK*. Časový digram vzorkování je na obrázku 4.5, kde na řádku *sampling* je vyznačen okamžik vzorkování vzhledem k náběžné hraně signálu *SCK*.



Obr. 4.5: Časový diagram vzorkování dat.

Proces *counting_to_32* je proces, který pouze počítá od 0 do 31, tedy 32 náběžných hran, protože jeden datový rámec má 32 bitů. Slouží tedy pro orientaci při záznamu jednotlivých bitů, na kterou pozici do 24 bitového slova má být právě navzorkovaný bit uložen.

Logika ukládání bitů je implementována v dalším procesu *word_completing* viz příloha B.8. Proces slouží k ukládání jednotlivých datových bitů z mikrofonů do 24 bitového datového slova. Tento proces je opět synchronní s náběžnou hranou hodinového signálu *clk* a běží, pokud je povolen běh TDM rozhraní pomocí signálu *TDM_en*. V předchozím procesu *sampling_signal_gen* byl vytvářen vzorkovací impuls *sampling_s*, který zde udává okamžik, kdy má dojít ke vzorkování dat. Vzorkování probíhá pouze pro prvních 24 bitů z 32 bitového rámce, jelikož v případě posledních 8 bitů už nejde o potřebná data. Vzorkování probíhá od nevýznamnějšího

24. bitu po nultý bit a mohou být vzorkována data z obou linek zároveň. Tyto bity se postupně nasouvají do proměnných *data_reg_x_s* ze vstupních portů *data_in_x*. Orientace v rámci je zde zajištěna právě hodnotou čítače *counter* z předchozího procesu, který je synchronizován s pořadím bitů generovaných mikrofony.

Poslední proces *data_to_out* viz příloha B.8 zajišťuje přesun dat z *data_reg_x* obou linek do portu *data_to_output* a ten je následně připojen na výstupní port entity, ze kterého jsou data odebírána pro další zpracování. V tomto případě se zde data odesílají do FIFO paměti směrem do počítače. Chování procesu vychází z hodnoty čítače *counter*, který zajišťuje orientaci v tom, který bit z 32 bitů je právě na sběrnici. Pomocí vhodné implementace je zajištěno, že data se budou dále zpracovávat, jen když nepůjde o data z prvních 85 ms provozu. Pokud je již na datové sběrnici 25. bit, tudíž už proběhlo navzorkování všech potřebných 24 bitů, 24 bitová data z aktivované první mikrofonní linky jsou přiřazena na výstupní port a signalizace pro další moduly *data_ready* je nastavena na úroveň log. 1. V tomto případě bude dále na výstupní port připojena FIFO paměť a odpovídající řídící signál pro zápis do paměti. Pokud je aktivní i druhá linka, data z ní se na výstupní port dostanou v okamžiku 27. bitu a následně nastane i signalizace. Tím je ošetřen postupný zápis dat do dalších modulů bez konfliktu linek mezi sebou.

4.1.3 Řízení generování signálů SCK, WS a příjmu dat

Všechny výše uvedené entity jsou řízeny pomocí entity *TDM_FSM*, ve které je implementován stavový automat. Entita *TDM_FSM* má několik vstupních a výstupních portů. Vstupní port *clk_fpga* je pro hodinový signál. Port *startUpEnd* je vstupní port, kterým je do této entity přivedena informace z předchozí entity *ws_mic*, že uplynula doba potřebná pro inicializace mikrofonů. Port *clk_div* předává ostatním entitám informaci o tom, jaký má být použit dělící poměr základního hodinového signálu v závislosti na počtu použitých mikrofonů v řetězci.

Port *clk_en* a *ws_en* slouží pro předání informace entitám, že je povoleno generovat hodinový signál pro mikrofony *SCK* a povolení generování pro synchronizační signál *WS*. Dále *ws_counter_en* je pro povolení odpočtu 11 ms pro počáteční inicializaci mikrofonů. Port *ws_length* předává entitě pro generování signálu *WS*, jakou má mít signál *WS* délku v hodinových taktech a *TDM_en* je pro povolení vzorkování dat z mikrofonů. Port *clk_locked* slouží k předání informace o připraveném hodinovém signálu. Port *ctrl_data* slouží pro příjem řídících dat, která pocházejí z řídící aplikace na počítači. Implementace příjmu dat z PC bude popsána v další kapitole.

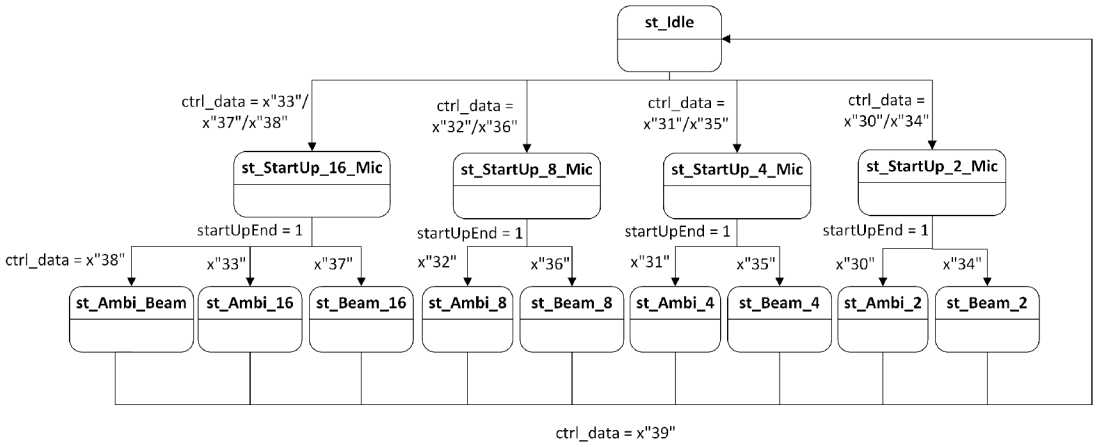
Port *LED* je následně připojen k LED diodám na vývojovém kitu, které slouží pro signalizaci uživateli/vývojáři, jaký mód je nastaven. Porty *fifo_empty* a *fifo_read_en*

slouží k řízení a přijímaní informací z FIFO paměti z části, která obsluhuje komunikaci s počítačem. Jde o FIFO paměť, ze které stavový automat přijímá řídící povely *ctrl_data*. Jelikož mikrofonní pole bude mít dvě TDM linky s 16 mikrofony, porty *in_1_en* a *in_2_en* slouží pro ovládání jednotlivých mikrofonních linek.

Stavový automat má 15 stavů, viz obrázek 4.6. První kombinační část popisuje řídící logiku stavového automatu, viz ukázka kódu v příloze B.5. V počátečním stavu *st_Idle* se nic neděje a FPGA čeká na vstup od uživatele.V závislosti na řídících datech z počítačové aplikace následně automat mění svůj aktuální stav. Stavy *st_StartUp_X_Mic* jsou stavy, během kterých dochází k počáteční inicializaci mikrofonů. Jakmile je inicializace dokončena, automat automaticky přejde do stavů, kdy jsou již mikrofony v provozu a odesílají data. Pracovní stavy jsou pojmenovány v závislosti na použití. Jedna z linek je pro mikrofony pro ambisonii a druhá linka pro beamforming. V koncové aplikaci uživatel bude moci použít jen stav *st_Ambi_Beam*, během kterého jsou v provozu obě linky současně. Stav *st_Ambi_16*, během kterého je v provozu jen linka pro ambisonii pro 16 mikrofonů, a stav *st_Beam_16* pro linku s uspořádáním mikrofonů pro beamforming. Automat v těchto stavech setrvává, dokud není přijata řídící instrukce na přechod do stavu *st_Idle*. Zbylé stavy jsou implementovány kvůli testování a ladění.

Druhá kombinační část stavového automatu již nepopisuje závislosti mezi jednotlivými stavy, ale zajišťuje přiřazení hodnot parametrů na výstupní porty, viz příloha B.6. Každý stav řídí hodnoty několika signálů. Jde o signály, které jsou následně přiřazeny na stejnojmenné výstupní porty. Jelikož v prvním stavu *Idle* není potřeba, aby FPGA cokoliv vykonávalo nebo generovalo, všechny signály jsou nastaveny do 0. Ve stavech pro inicializaci mikrofonů *st_StartUp_2_Mic* až *st_StartUp_16_Mic* je potřeba, aby se začal generovat hodinový signál pro mikrofony *SCK*. Také je nutné začít generovat hodinový signál pro FPGA se stejnou frekvencí jako *SCK* a předat ostatním entitám informaci o tom, jakou děličku mají použít. Dále je potřeba, aby se začala stopovat doba trvání inicializace a aby byla zapnuta správná LED dioda pro signalizaci. Proto tyto signály mají buďto hodnotu nastavenou v logické 1, nebo mají přiřazenou hodnotu. Ostatní signály, jako je povolení generování signálu *WS*, povolení TDM rozhraní, nebo délka signálu *WS*, jsou nastaveny do 0, protože ještě není vyžadováno jejich použití. Ve stavech *st_Ambi_X*, *st_Beam_X* a *st_Ambi_Beam* dochází k nastavení i zbývajících signálů, protože tyto stavy zaručují plnou funkčnost a již se očekává, že je potřeba generovat synchronizační signál, ale také přijímat data z mikrofonů.

V modulu jsou implementovány ještě další dva procesy, kde jeden je výstupní registr, který řídící signály z druhé kombinační části posílá na výstupní porty synchronně s hodinovým signálem. Poslední proces je řídící proces pro signály, které řídí paměť FIFO ve směru toku dat z počítače do FPGA. Má za úkol číst data z



Obr. 4.6: Řídící stavový automat.

FIFO paměti, jen pokud FIFO paměť není prázdná a pokud je automat ve stavech, ve kterých může nastat změna stavu.

Kvůli přehlednosti zde byl opět přidán modul *TDM_interface*, který sdružuje entity *clk_out_mic*, *clk_out_FPGA*, *ws_mic* a *TDM_FSM*. Tento modul neobsahuje žádnou logiku, pouze zapouzdřuje porty entit, které nejsou dále potřebné, a propojuje signály jednotlivých entit mezi sebou.

4.1.4 Přenos dat mezi počítačem a FPGA

Pro přenos dat z FPGA do počítače byl použit externí USB převodník FTDI232. Ačkoliv je tento čip a režim hojně používaný ve spojení s FPGA, není zde žádné volně dostupné IP jádro, které by se pro komunikaci mezi čipy dalo použít, proto se následující část práce bude zabývat implementací obousměrné komunikace mezi FPGA a FTDI v režimu synchronního FIFO 245. Směrem z počítačové aplikace budou přes převodník FTDI do FPGA proudit řídící data. V opačném směru komunikace budou z FPGA přes FTDI do počítačové aplikace přenášena data z mikrofonů.

Implementace režimu Sync FIFO 245

Zvolený režim používá přenos dat mezi FTDI a FPGA po 8 bitech, takže je zde použito 8 vstupně/výstupních datových signálů *D0* až *D7*. Režim se nazývá synchronním, protože FTDI čip má svůj vlastní hodinový signál, se kterým musí být synchronizována i řídící komunikační logika v FPGA. Dalším důležitým parametrem je, že FTDI má aktivní úroveň signálu v log. 0 a s tím je třeba počítat i při návrhu modulu pro FPGA [40].

Při řízení modulu synchronního FIFA je opět použit stavový automat se čtyřmi stavy. Dále je nutné počítat se dvěma hodinovými signály, z nichž jeden je hodinový signál *clk* 98,304 MHz pro řízení FPGA a mikrofonů, který byl popsán v kapitole 4.1.1, a druhý *clk_FTDI* 60 MHz pro řízení komunikace mezi FPGA a FTDI. Je patrné, že signály vůči sobě nejsou synchronní a je potřeba použít prvky k zajištění správné funkčnosti návrhu. Důležitým prvkem je také implementovat rozklad dat z mikrofonů, která jsou 32 bitová na 8 bitová slova, která FTDI umí přenášet.

Všechny potřebné funkcionality pro FTDI čip jsou zapouzdřeny do modulu *FTDI* viz příloha B.9. Modul byl navržen pro specifické potřeby této práce a není zcela univerzální. Modul *FTDI* obsahuje IP jádro Clocking Wizard, dvě IP jádra pro implementované paměti FIFO a řídící stavový automat v modulu *FTDI_SYNC_FIFO*. Modul *FTDI* má několik vstupních a výstupních portů. Obsahuje vstupní porty pro oba hodinové signály, vstupně/výstupní port *data*. Dále datové a řídící porty pro obě FIFO paměti a port pro signalizaci, že byl inicializován hodinový signál.

Synchronizace hodinových signálů

V modulu jsou dvě FIFO paměti z důvodu synchronizace dat mezi dvěma hodinovými signály, které vůči sobě nejsou synchronní. Samotný stavový automat, který řídí komunikaci, musí fungovat na frekvenci 60 MHz, což je frekvence FTDI čipu, ale samotná řídící a datová logika na FPGA funguje na frekvenci 98,304 MHz. Pro synchronizaci modulů a dat, které jsou řízeny různými hodinovými signály, jsou využity FIFO paměti, které umí pracovat s rozdílným hodinovým signálem, který nemusí být při zapisování a při čtení ani synchronní. Pokud by nebyl implementován tento synchronizační prvek, docházelo by k velkému množství metastabilních stavů a implementace by nebyla funkční.

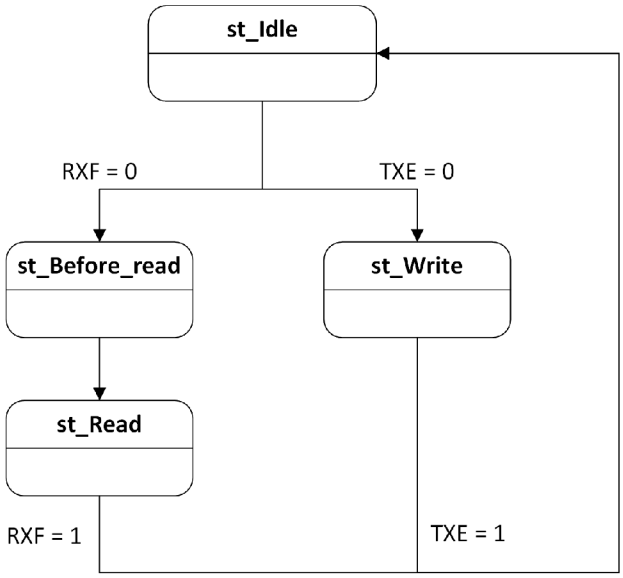
Paměť *fifo_data_to_pc* je implementována pomocí IP jádra *FIFO Generator*. Jde o paměť, která zaznamenává data z mikrofonů z modulu *TDM_interface*. Data jsou do paměti zapisována v závislosti na signalizaci, že jsou data připravena na výstupním portu modulu *TDM_interface*. Paměť je v režimu *Independent Clocks Block RAM* z důvodu, že tento režim umožňuje na vstup paměti posílat 32 bitová data, na výstupu jsou data rozložená do 8 bitových slov, která už mohou být přenášena FTDI čipem. Čtení 8 bitových slov z paměti pak podléhá stavu řídícího stavového automatu, který již pracuje na frekvenci 60 MHz.

Druhá paměť *fifo_data_from_pc* je zde z podobného důvodu. Slouží pro synchronizaci stavového automatu pro FTDI a ukládání řídících povelů pro stavový automat v *TDM_interface*. Oba stavové automaty fungují na rozdílných frekvencích, proto je důležitá synchronizace, jinak by velmi často docházelo k metastabilním stavům a návrh by nebyl funkční. Paměť je opět typu *Independent Clocks Block RAM*, ale

řídících povelů je malé množství a na jejich vyjádření stačí 8 bitů, takže zde není využita implementace, která by z 8 bitových slov skládala slova 32 bitová. Zapisování do paměti závisí na stavu řídícího stavového automatu pro FTDI a čtení z paměti závisí na řídícím stavovém automatu *TDM_FSM*.

Řízení komunikace mezi FTDI a FPGA

Celý stavový automat a podpůrné procesy pro řízení komunikace jsou implementovány v entitě *FTDI_SYNC_FIFO*. Zde implementovaný stavový automat řídí příjem a odesílání dat mezi výše popsanými FIFO pamětmi a FTDI čipem. Stavový automat má vzhledem k FPGA 4 stavy, a to *st_Idle, st_Before_Read, st_Read, st_Write*. V jeden okamžik nelze přenášet data mezi čipy obousměrně, proto musí být jednotlivé fáze odděleny. Zjednodušený stavový automat je na obrázku 4.7.



Obr. 4.7: Řídící stavový automat FTDI.

Čtení dat z FTDI do FPGA je provedeno ve dvou stavech tak, aby byly splněny požadavky dle dokumentace [40]. Jakmile čip FTDI má ve své interní paměti data z počítače, jeho signál *RXF* se nastaví na úroveň log. 0. Tuto změnu musí zaregistrovat stavový automat v FPGA. Pro úspěšné čtení musí být následně nastaven signál *OE* do log. 0 v rámci stavu *st_Before_Read*. Tím FPGA signalizuje FTDI, aby připravil data na výstupní datové piny. V následujícím hodinovém taktu je již stavový automat ve stavu *st_Read*, během kterého dojde ke čtení dat z FTDI paměti a pokud FTDI má další data v paměti, tak je chystá na datové piny do příchodu další náběžné hrany hodinového signálu FTDI. Data z FTDI se ukládají do FIFO paměti v FPGA *fifo_data_from_pc*. V tomto případě jde o řídící data pro stavový

automat *TDM_FSM*. Čtení řídících dat z FTDI má přednost před odesíláním dat do počítače.

Zápis dat z FPGA do FTDI je prováděn jen ve stavu *st_Write*. Pokud je FTDI čip připraven k zápisu, na pinu *TXE* je log. 0. Na základě této informace stavový automat přejde do stavu pro zápis, ale jen tehdy, pokud má připravená data z mikrofonů ve FIFO paměti *fifo_data_to_pc*. V rámci stavu pak nastaví potřebné řídící signály. Pokud FPGA do FTDI odesílá data a nastane požadavek na přijetí dat z počítače, odesílání dat do počítače se přeruší, ale tím nedojde ke ztrátě dat. Pouze se na okamžik přeruší čtení dat z FIFO paměti *fifo_data_to_pc*. Paměť *fifo_data_to_pc* je ale dostatečně velká, aby toto krátké přerušení v odesílání dat pokryla svou kapacitou a nedojde tak ke ztrátě dat z mikrofonů.

Důležitou součástí je zde i proces pro výstupní registr, viz příloha B.10. Ihned po připojení napájení k FPGA musí být řídící signály v log. 1. V opačném případě by FTDI začalo číst data z datových pinů v okamžiku, kdy na pinech žádná data připravená nejsou a vnitřní paměť FTDI čipu by se zaplnila neplatnými daty. Proto je u tohoto procesu implementován asynchronní reset. U výstupního registru také musel být použit fázově posunutý hodinový signál z FTDI. Ačkoliv výrobce tvrdí, že ke změnám řídících signálů může docházet v okamžiku náběžné hrany hodinového signálu, tak tato implementace nebyla spolehlivá [40]. Při komunikaci mezi čipy docházelo k metastabilním stavům, a to tak, že například FTDI čip přečetl řídící signál s náběžnou hranou hodinového signálu tak, že je na úrovni log. 0, zatímco FPGA tuto úroveň již vyhodnotilo jako log.1. To mělo za následek vynechávání, nebo zdvojování bytů v paměti. Komunikace nebyla spolehlivá. Proto zde bylo implementováno také IP jádro Clocking Wizard, aby bylo možné hodinový signál z FTDI posunout o půl periody. Posunutý hodinový signál je pak využit ve výstupním registru a díky tomu se řídící signály mění v dostatečném předstihu před jejich čtením a k výše zmíněné metastabilitě nedochází.

Samotné FTDI pro přenos dat používá obousměrné signálové cesty, proto zde pro port *data* nemohly být použity standardní jednosměrné porty in/out. Bylo potřeba využít „třístavovou vyrovnávací paměť“ (tri state buffer) [43]. Jde o vstupně/výstupní porty, u kterých lze pomocí stavu vysoké impedance nastavit, zda se má chovat jako vstupní nebo výstupní port. Řídící logika tohoto bufferu musela být implementována do posledního procesu viz příloha B.10, který se v závislosti na aktuálním stavu stavového automatu stará o směr datového toku.

4.2 Programování čipu FTDI232

Pro funkční komunikaci FTDI s FPGA a počítačem je potřeba FTDI232 naprogramovat tak, aby komunikoval v požadovaném režimu. Výrobce pro tento účel vytvořil

aplikaci *FT_Prog* a její možnosti jsou popsány v dokumentaci [44].

Zde bylo potřeba překonfigurovat defaultní nastavení. V *USB_Config_Descriptor* nastavit napájeni z USB sběrnice z počítače, povolit mu maximální odběr 500 mA a deaktivovat jeho uspání v případě, že je zařízení připojené, ale neaktivní. Popis zařízení *USB_String_Descriptor* a i sériové číslo bylo ponecháno defaultní. Toto nastavení lze ponechat, jelikož se předpokládá použití zařízení v laboratorních podmínkách. Dále bylo potřeba v sekci *Hardware_Specific* zvolit režim 245 FIFO a příslušné ovladače *D2XX Direct* pro počítač. Dle dokumentace nelze využít standardní ovladače Virtual COM Port pro režim synchronní paměti FIFO. Před naprogramováním FTDI čipu je ještě potřeba zvýšit proudový rozsah pinů na 12 mA, protože při původních 4 mA komunikace nefungovala správně. Kvůli signalizaci, že FTDI čip je v provozu, bylo u pinu C9, který je připojený k LED diodě, nastaveno její rozsvícení. Touto konfigurací byl následně čip naprogramován.

4.3 Programování aplikace v prostředí Matlab

Data z mikrofonů jsou z přes FPGA a FTDI převodník posílána na USB port počítače, ve kterém poběží aplikace pro řízení celého řetězce. Aplikace je vytvářena dle zadání v prostředí Matlab. Pro vytvoření grafického uživatelského rozhraní bude využito rozšíření *App Designer*.

4.3.1 Komunikace s FTDI

Pro komunikaci s FTDI převodníkem je možné využít standardní Virtual Com Port ovladače a nebo přímo ovladače D2XX od výrobce [45]. V této práci musely být využity ovladače D2XX, protože s nimi je možné použít režim FIFO 245 [40]. Pro komunikaci s FTDI převodníkem bylo vytvořeno následujících pět funkcí, které vycházejí z dokumentace *D2XX Programmer's Guide* [45]. Funkce jsou uloženy v adresáři Matlab/FTDI_function.

FTDI_init()

Funkce FTDI_init slouží k otevření komunikačního kanálu mezi počítačem a FTDI převodníkem. K ukládání „Handle“ na zařízení a k počátečnímu nastavení. Je potřeba nastavit komunikační režim FIFO 245, velikost vnitřní paměti na maximum a také testovat, jestli bylo připojení úspěšné. Pokud připojení neproběhlo v pořádku, je tato chyba signalizována uživateli v podobě „Error Message Boxu“. Kód funkce je ve výpisu C.1,

FTDI_write()

Funkce FTDI_write slouží k odesílání dat z počítače do FTDI převodníku a ten je následně předá FPGA. V tomto případě jde o 8 bitová data, která reprezentují řídící příkazy pro FPGA. V řídícím příkazu je zakódováno, kolik je připojeno mikrofonů a která mikrofonní linka se má spustit, popřípadě řídící příkaz pro ukončení. O to, jaký režim se má zvolit, se stará uživatel a informace o režimu je předána funkci vstupním parametrem *data*. Informace o režimu, o velikosti dat a o ukazateli na zařízení se pak předají externí funkci FT_Write z knihovny *d2xx*. Kód funkce je ve výpisu C.2.

FTDI_status()

Funkce FTDI_status slouží ke zjištění stavu FTDI převodníku, kolik má ve své vnitřní paměti dat v obou směrech komunikace. Jde o důležitou funkci jak pro odesílání, tak pro příjem dat. Kód funkce je ve výpisu C.3.

FTDI_read()

Jde o funkci pro čtení dat z paměti FTDI převodníku. Před voláním externí funkce FT_Read je potřeba alokovat paměť pro příjem dat o potřebné velikosti. Jelikož FTDI přenáší data v 8 bitové podobě, je vhodné i paměť alokovat v tomto formátu. Pokud proběhne čtení úspěšně, funkce vrací data v matici *dataRead* a počet přečtených bytů. Kód funkce je ve výpisu C.4.

FTDI_end()

Při ukončení běhu aplikace je potřeba ukončit i komunikaci s FTDI převodníkem. Když k ukončení komunikace nedojde, komunikační kanál se zařízením zůstane otevřen a při novém spuštění aplikace nebude možné se k FTDI převodníku připojit. Funkce volá funkci FT_Close z externí knihovny *d2xx*. Kód funkce je ve výpisu C.5.

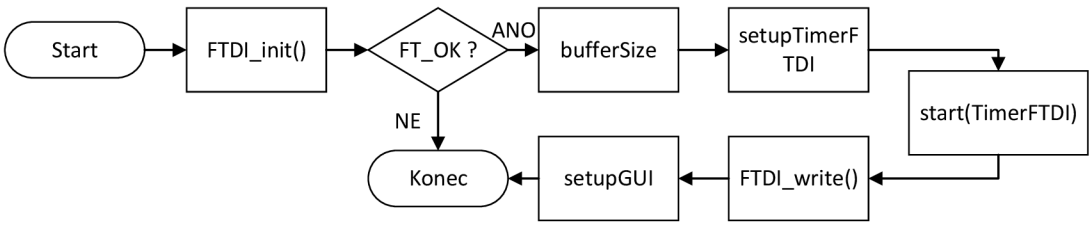
4.3.2 Příjem dat

Algoritmus pro příjem dat z FTDI převodníku byl navrhnut tak, aby docházelo k pravidelnému kontrolování stavu vnitřní paměti FTDI. Pro tento účel byl využit plánovač spuštění příkazu pomocí časovače [46], který spouští daný algoritmus v pravidelných intervalech. Časovač je nastaven tak, aby se příjem dat spouštěl dostatečně často, aby byl pokryt potřebný objem dat, který je potřeba přenést. Nastavení časovače je v příloze C.6.

Po zapnutí nahrávání uživatelem pomocí GUI se spustí proces, který je nakreslen v diagramu 4.8. Na počátku spuštění běhu aplikace proběhne pokus o navázání

komunikace a nastavení FTDI převodníku pomocí funkce FTDI_init() 4.3.1. Jestliže neproběhne správně, aplikace se nespustí. V opačném případě dojde k nastavení *bufferu* pro zpracování dat a nastavení velikosti paměti *chunkSizeFTDI*, do které se budou ukládat data z FTDI. Velikosti pamětí se nastavují v závislosti na režimu. Dále je nastaven a spuštěn časovat, který spouští v pravidelných intervalech funkci pro příjem a ukládání dat.

V této fázi již probíhá komunikace mezi aplikací, FTDI a FPGA, ale ještě není zahájen tok dat. Proto je volána funkce *FTDI_write()* 4.3.1, které jsou předány informace o režimu, v jakém mají mikrofony fungovat. V závislosti na zvoleném režimu je přeš FTDI převodník do FPGA vyslán řídící příkaz, který zahájí inicializaci mikrofonů, které začnou odesílat data do FPGA. Data FPGA předzpracuje a začne posílat do FTDI, s nímž komunikuje funkce, kterou řídí *timerFTDI*. V poslední řadě dojde ke změnám v GUI.



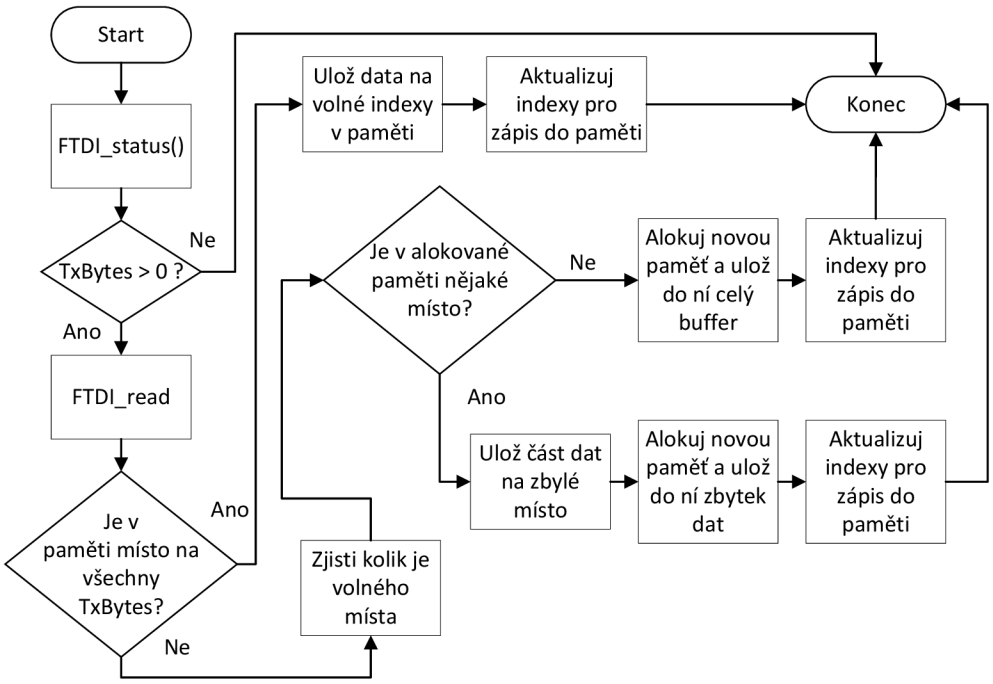
Obr. 4.8: Inicializace příjmu dat.

Časovač *timerFTDI*, který se stará o řízení příjmu dat, spouští funkci *readingDataFromFTDI*, jejíž vývojový diagram je na obrázku 4.9. Jakmile uplyne požadovaná perioda spouštění, dojde k provedení funkce. Jako první aplikace zjistí stav vnitřní paměti FTDI převodníku. Jestli FTDI nemá v paměti žádná data, proces se ukončí. V opačném případě je volána funkce *FTDI_read*, které se jako vstupní parametr předává počet bytů *TxBytes*, které má z FTDI paměti přečíst, a vrací počet přijatých bytů a 8 bitová data.

Následuje proces ukládání dat. Pro ukládání dat se využívá pole buněk (cell array). Tato datová struktura byla zvolena z důvodu, že pomocí ní lze jednoduše vytvářet dynamicky alokovanou paměť. Bylo by neefektivní při každém čtení alokovat novou paměť nebo překopírovávat data do nové větší paměti. Z tohoto důvodu se alokují bloky paměti, které mohou pojmout 1 sekundu záznamu. Pole buněk umožňuje mít buňky specifické velikosti a jakmile je buňka zaplněná daty, tak je možné alokovat novou paměťovou buňku a vřadit ji do pole na následující index. Tento princip ukládání dat je postupně využit v celé práci.

Jakmile jsou přijata data, proběhne kontrola, jestli je v aktuální buňce místo pro všechna přijatá data. Pokud ano, data jsou uložena na příslušné adresy a proběhne aktualizace pozice zapisovacích indexů. Pokud se do paměti nevlezou všechna přijatá data, zjistí se, kolik paměti je v aktuální buňce volné. Jestliže je v buňce ještě volná paměť, část přijatých dat se zapíše právě do této volné paměti. Následně proběhne alokace nové buňky, proběhne aktualizace indexů pro zápis a zbytek neuložených dat se uloží do nově alokované paměti. Pokud v aktuální buňce už žádná volná paměť nezbývá, ihned dojde k alokování nové paměti, uložení dat a změně indexů. Proces běží, dokud uživatel nedá povel k zastavení.

Na příjem dat navazuje předzpracování dat, které je popsáno v kapitole níže 4.3.3. Během předzpracování dochází i k uvolňování jednotlivých paměťových buněk, aby byly sníženy paměťové nároky.



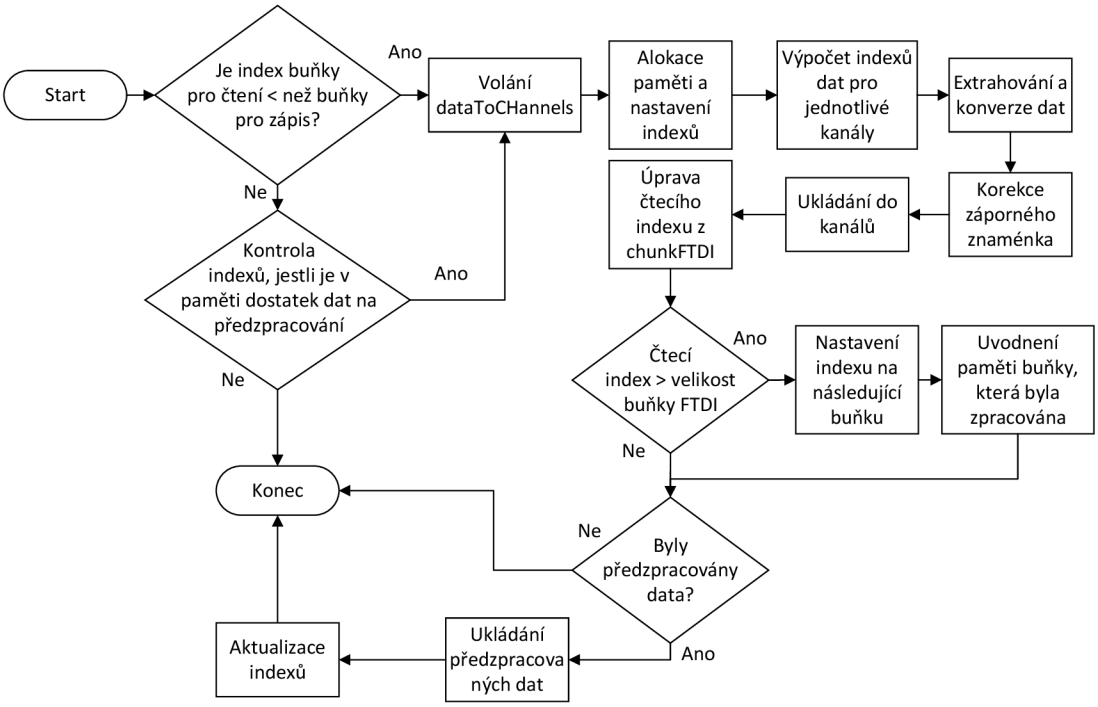
Obr. 4.9: Příjem a ukládání dat.

4.3.3 Předzpracování dat

Data z FTDI jsou přijímána v 8 bitové podobě a je potřeba z nich poskládat 24 bitová data ve dvojkovém doplňku, aby se dala dále použít a také uspořádat do kanálů. Předzpracování dat je opět pravidelně prováděno pomocí časovače *timerInChannel* po buffrech, které pojmou 10 ms záznamu a spouští se ve stejný okamžik jako příjem dat.

Celý proces předzpracování dat začíná kontrolou, jestli již byla z FTDI přijata a uložena data. Jestli index buňky pro zápis a pro čtení není shodný, což je stav kdy, jsou data zapisována do vyšší buňky, než probíhá předzpracování, není potřeba kontrolovat přímo datové indexy. Pokud je buňka, ze které se data čtou, plná, nemůže nastat situace, kdy by došlo ke čtení dat, která ještě ani nebyla přijata.

Pokud jsou indexy buněk pro zápis a čtení shodné, musí nastat kontrola přímo datových indexů, aby nenastala výše zmíněná situace, že by byla předzpracována data, která nebyla přijata. Pokud v paměti není dostatek dat na zpracování bufferu, k předzpracování nedojde.



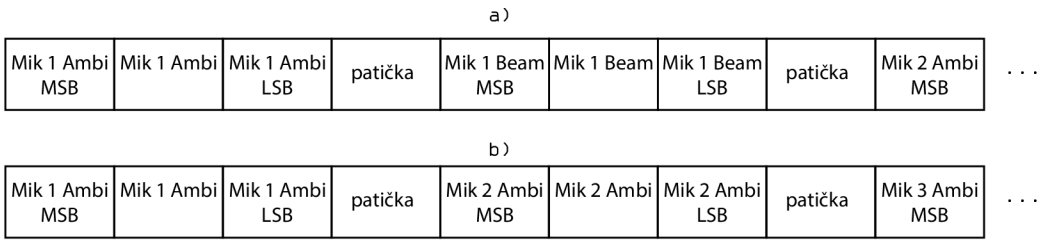
Obr. 4.10: Předzpracování dat.

Samotné předzpracování dat se děje ve funkci *dataToChannels()*. Funkce má vstupní parametr pro buffer dat na předzpracování, velikost bufferu a volbu režimu. V závislosti na režimu je provedena počáteční konfigurace předzpracování. Kolik kanálů se má vytvořit, počáteční indexy pro ukládání dat a alokace paměti, která bude následně naplněna předzpracovanými daty. Zde jsou ponechány i režimy pro méně než 16 mikrofonů na linku kvůli testování a ladění. Pokud je spuštěna jen jedna linka mikrofonů, data jsou ukládána do matice, kde sloupce představují kanály. Pokud jsou spuštěny obě linky zároveň, tak prvních 16 sloupců je pro data

z mikrofonní linky pro ambisonii a od 17. sloupce se ukládají kanály z linky pro beamforming.

Na počátku předzpracování jsou vypočítány všechny indexy nejvýznamnějších bytů, na kterých se nacházejí všechna 8 bitová data, která přísluší jednomu kanálu. Data v *chunkFTDI* a i ve zpracovávaném bufferu jsou uložena po bytech v pořadí jako na obrázku 4.11. Z pozic nejvýznamnějších bytů jsou pak odvozeny i indexy zbývajících bytů. Následně jsou provedeny bitové posuny 8 bitových dat na příslušné pozice. Mikrofony produkují 24 bitová data, takže $data_0$ na indexu se posunou o 16 bitů, $data_1$ na index + 1 o 8 bitů a $data_2$ na index + 2 se neposunují. Dále je provedena kompletace těchto posunutých bytů. Důležitým krokem je pak převod na int32 a kontrola hodnoty na pozici 24. bitu. Dosud se s daty pracovalo jako uint, ale při převodu na int je potřeba doplnit 1 na horních 8 bitů, pokud má jít o záporné hodnoty, aby byla získána reprezentace dat ve dvojkovém doplňku.

V dalším kroku už jsou jen předzpracovaná data ukládána do příslušných sloupců, které patří jednotlivým kanálů. Předzpracování dat v rámci jednoho kanálu se provádí pomocí vektorizace, která je v prostředí Matlab efektivnější než pomocí zanořených smyček.



Obr. 4.11: Uspořádání dat v paměti, a) kombinovaný režim, b) jedna linka.

Po předzpracování dat je proveden návrat do funkce *preprocessDataToChannels*(), kde je provedena kontrola, jestli již byla zpracována celá buňka. Pokud byly předzpracována všechna data v rámci jedné buňky, nastane aktualizace indexů na následující buňku a uvolnění paměti aktuální buňky, aby se snížily paměťové nároky. Předzpracována data jsou ukládána do buněk *chunkAudio* obdobným způsobem jako v případě příjmu dat z FTDI převodníku v kapitole 4.3.2. Nyní už jsou data upravena do použitelných datových struktur a roztříděna do kanálů.

4.3.4 Zpracování Ambisonie

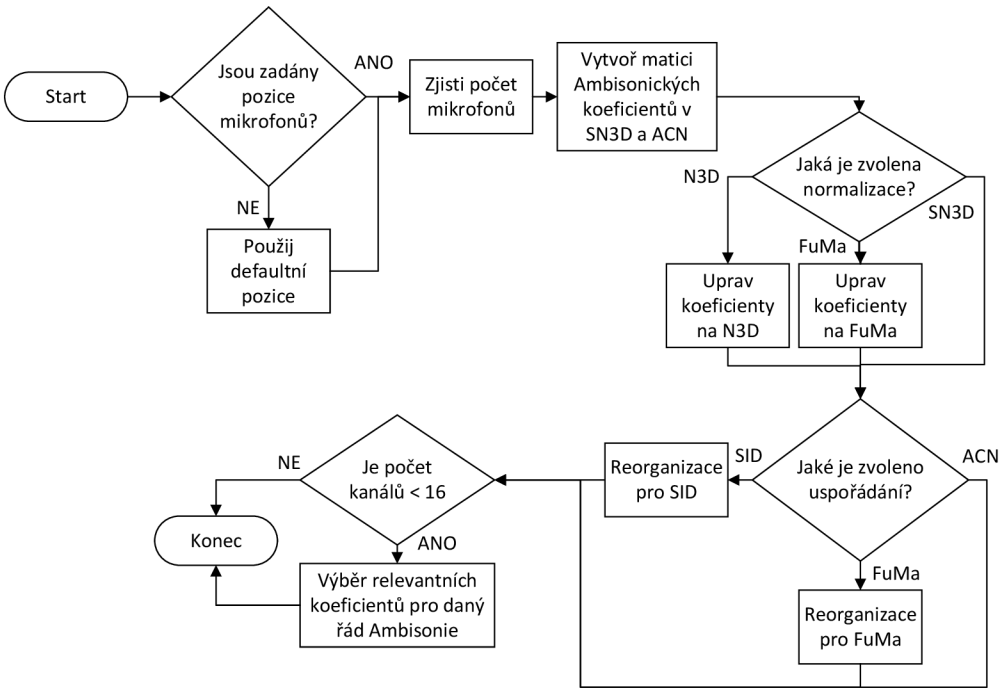
Zpracování dat do formátu ambisonie lze rozdělit do dvou hlavních částí. První část se zabývá výpočtem ambisonických koeficientů v závislosti na zvolené normalizaci

a pořadí kanálů v kapitole 4.3.4. Druhá část pak samotným procesem kódování a ukládáním zakódovaných dat v kapitole 4.3.4.

Výpočet koeficientů Ambisonie

Pro tento proces je napsána funkce *AmbiSetting()*, která v závislosti na zadané normalizaci, uspořádání kanálů a na pozicích mikrofonů vypočítá koeficienty až pro ambisonii 3. řádu.

Vstupními parametry funkce je možné zvolit jednu ze 3 normalizací. Buďto *SN3D*, *N3D*, anebo *FuMa*. Také je možné vybrat jedno ze tří uspořádání kanálů, a to *ACN*, *SID* a *FuMa*. Algoritmus pro výpočet ambisonických koeficientů je na obrázku 4.12.



Obr. 4.12: Výpočet koeficientů Ambisonie.

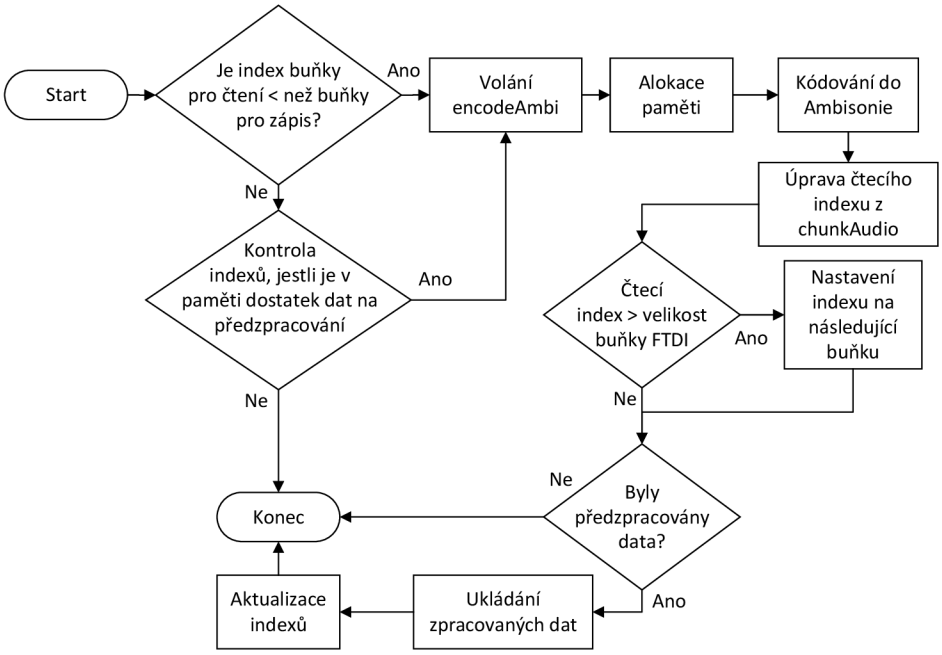
Funkce standardně počítá s pozicemi mikrofonů pro ambisonii 3. řádu, které jsou definovány uvnitř funkce. Jde o pozice mikrofonů, které náleží sférickému mikrofonnímu poli, jehož konstrukcí a návrhem se zabývá souběžně zpracovávaná bakalářská práce. Je zde ale i možnost zadat vlastní pozice mikrofonů ve stupních v případě použití na jiném sférickém mikrofonním poli.

Po výběru mikrofonů je zjištěn počet mikrofonů, se kterými se bude dále pracovat. Informace o počtu je klíčová pro správný řád ambisonie. Následně se vždy

vytvoří matice koeficientů pro 3. řád ambisonie, kde řádky odpovídají koeficientům pro jeden mikrofon. Tato matice se vytvoří s normalizací SN3D a uspořádáním ACN, což je dnes nejrelevantnější formát ambisonie.

Pokud uživatel zadá jinou normalizaci, matice je vynásobena převodními koeficienty buď na normalizaci FuMa, nebo N3D. Dále je možné zvolit uspořádání kanálů. Matice se standardně vytvořena v uspořádání ACN, ale je zde možnost kanály reorganizovat do pořadí FuMa, nebo SID.

Jako poslední je provedena kontrola, jestli je požadována ambisonie 3. řádu, pro kterou je potřeba 16 zvukových kanálů. Jestliže by funkce byla použita například pro ambisonie 2. řádu, na kterou je potřeba 9 mikrofonů, matice se ořeže na potřebný počet koeficientů. Funkce pak vrací matici koeficientů ambisonie a také pozice mikrofonů.



Obr. 4.13: Proces zpracování Ambisonie.

Kódování do Ambisonie

Samotný proces kódování probíhá v pravidelných intervalech opět pomocí časovače, který volá funkci *processAmbi()*. Funkce se stará o kontrolu dostupných dat, o samotné kódování dat po buffrech definované velikosti a ukládání již zakódovaných dat do paměti.

Jako vstupní parametr funkce musí být funkci dány koeficienty pro kódování do ambisonie, které byly získány pomocí funkce *AmbiSetting()*, která je popsána výše. Během procesu zpracování je jako první zjištěn stav dat, jestli je co zpracovávat. Pokud ano, je volána funkce *encodeAmbi()*, která zajišťuje hlavní část zakódování dat. Pro efektivní provedení funkce je nejdříve alokována paměť, do které se následně ukládají již zpracovaná data. Jde o poměrně jednoduchý proces, ve kterém jsou zvuková data násobena ambisonickými koeficienty a následně je provedena jejich sumace do příslušných kanálů.

Následně je implementována logika ukládání do buněk obdobným způsobem jako v případě procesu pro předzpracování dat v kapitole 4.3.3. Celý proces je znázorněn na obrázku 4.13.

4.3.5 Zpracování Beamforming

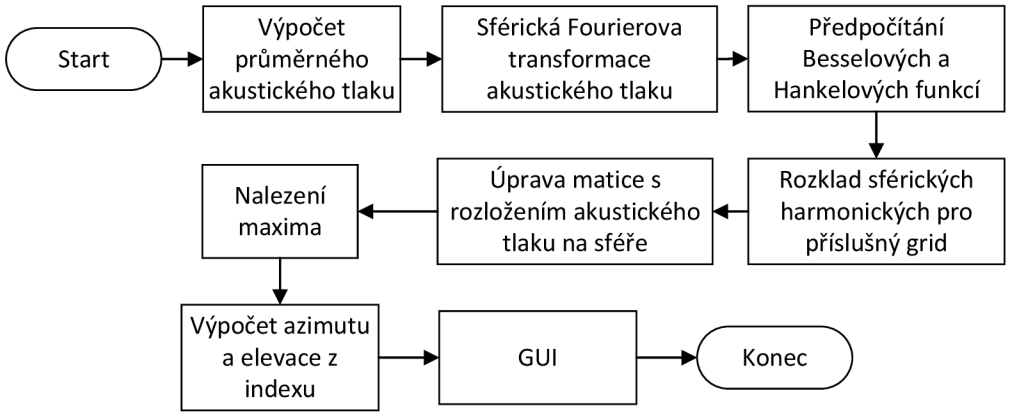
Metoda zpracování signálu beamforming je opět prováděna již během nahrávaní signálu, a to pomocí pravidelného volání funkce *beamforming()* obdobně jako v případě ambisonie. Zpracování beamformingu lze rozdělit do dvou částí. První část se zabývá lokalizací směru zdroje zvuku. Pro odhad lokalizace byla využita metoda rozkladu sférických harmonických funkcí 2.2. Rozklad sférických harmonických funkcí je implementován ve funkci *beamProcess()* a následné nalezení maxima ve funkci *beamformingPlot()*.

Algoritmus je zobrazen v diagramu 4.14 a funguje následovně. Jako první je z příslušného datového bufferu vypočítán průměrný akustický tlak v závislosti na parametrech mikrofonu. Následně je na jednotlivé akustické tlaky z jednotlivých snímačů aplikována sférická Fourierova transformace 2.27 pro získání akustických tlaků na povrchu sféry v místech snímačů. Pro optimalizaci algoritmu bylo následně potřeba předpočítat Besselovy a Hankelovy sférické funkce druhého řádu a jejich derivace pro získání síly sférického módu 2.26 pomocí funkcí *shBesselFncX* a *shHankelFncX*. Předpočítání bylo zvoleno, aby se nemusely v hlavní smyčce opakovaně počítat. Díky tomu poklesla výpočetní náročnost algoritmu. Následně je pomocí několika zanořených smyček proveden rozklad akustického tlaku na povrchu sféry pro příslušné úhly mřížky.

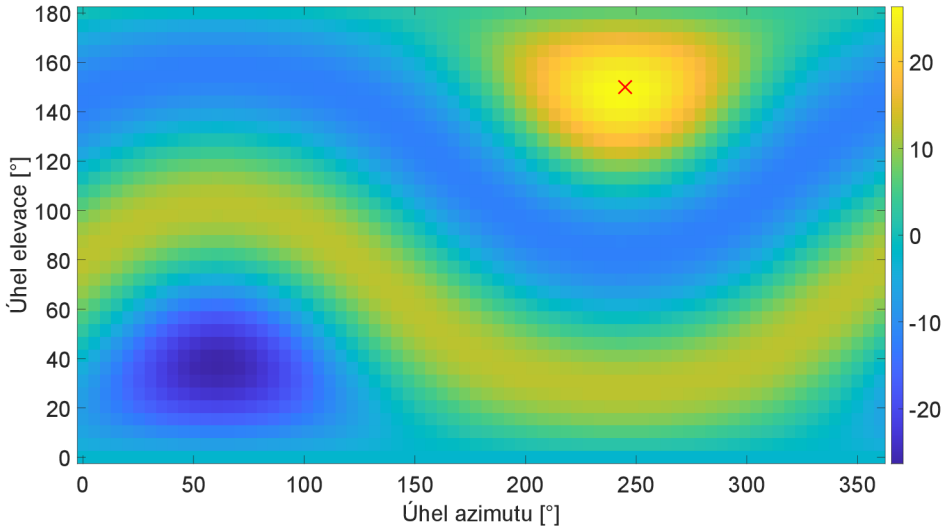
Úhly samotné mřížky povrchu koule a jejich příslušné sférické harmonické koeficienty společně s koeficienty pro snímače jsou počítány na počátku spuštění aplikace ve funkci *beamSetting()*. Není nutné je počítat v každém cyklu výpočtu, protože pozice mikrofonů a rozložení mřížky se nemění. Ve funkci *beamSetting()* jsou volány funkce jako *getSpherHarm()*. Tato funkce vrací sférické harmonické koeficienty příslušného řádu v komplexním tvaru pro danou souřadnici popsanou azimutem a elevací. Dále je zde funkce *myGrid*, která generuje rozložení mřížky na povrchu sféry,

pro kterou se pak bude následně počítat rozklad.

Nyní je získána matice viz 4.15 akustických tlaků na povrchu sféry a ve funkci *beamformingPlot()* je upravena do řádků a sloupců, v nichž je nalezeno maximum akustického tlaku, a z indexů maxima je vypočítán azimut a elevace. Azimut a elevace jsou pak vykresleny v GUI.



Obr. 4.14: Proces odhadu směru zdroje zvuku.



Obr. 4.15: Rozložení akustického tlaku na sféře.

Maximum, které označuje lokalizovaný směr zdroje zvuku, je v matici 4.15 vyznačeno červeným křížem. Nyní je lokalizován zdroj zvuku a jeho úhel azimutu a

elevace. Druhá část metody beamforming se zabývá prostorovým filtrováním. Prostorové filtrování je realizováno pomocí funkce *myBeamWeighting()*. Jako směrový vzor byl využit vzor hyperkardiody dle [6]. Při váhování kanálů je potřeba nejprve vypočítat jednotlivé váhy pro každý senzor. Nejprve byl vypočten relativní úhel mezi odhadnutým směrem, ve kterém se nachází zdroj zvuku, a jednotlivými senzory. K tomu byl využit převod uhlů azimutu a elevace do souřadnicového systému *x, y, z*, následně vypočten relativní rozdíl a tento rozdíl převeden zpět do relativního úhlu θ v radiánech. Pomocí směrového vzoru hyperkardiody pak byl pro každý snímač a příslušný relativní úhel spočten útlum daného kanálu. Zvukové kanály snímačů, které se blíží směru zdroje zvuku, jsou tak potlačeny minimálně, zatímco zvukové kanály směrových přijímačů, které mají větší relativní úhel od směru zdroje zvuku, jsou zeslabeny. Sférické mikrofonní pole se tak chová jako jeden hyperkardiodní snímač, který se natáčí v závislosti na lokalizovaném směru zdroje zvuku.

Na závěr je provedena sumace jednotlivých kanálů do jednoho. Tento kanál je pak stejně jako v případě implementace ambisonie ukládán do příslušné buňkové struktury a po dokončení nahrávání může být exportován.

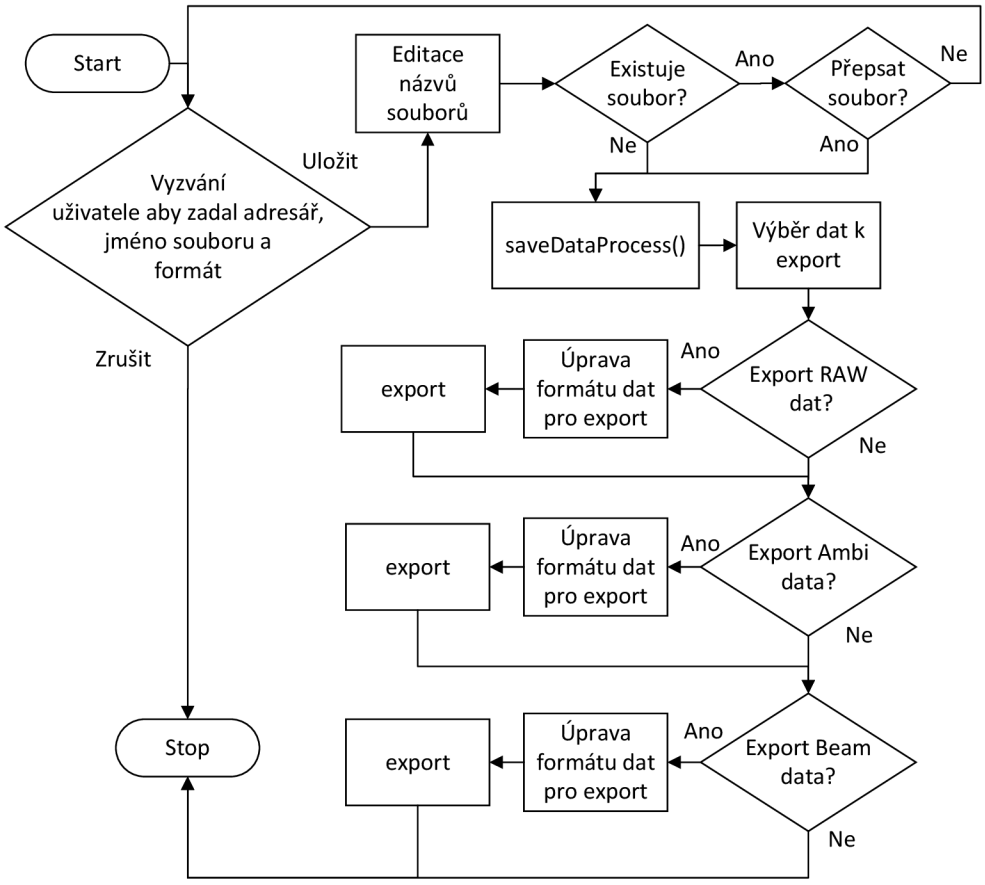
4.3.6 Export dat

Proces pro export dat do zvukového souboru je důležitou součásti aplikace, aby data získaná nahráváním mohla být i dále využita. Celý proces probíhá v hlavní funkci pro export *exportData()* a je znázorněn na obrázku 4.16. Soubory jsou ukládány pod názvem souboru s přidanou příponou podle toho, o jaká data a kanál jde. Například pokud bude zadán název souboru *nahrávka* a půjde o čistá zvuková data 1. kanálu z linky pro ambisonii, tak výsledný název souboru bude *nahrávka_RAW_data_Ambi_ch_1.wav*.

Jestliže je zavolána funkce pro export dat, jako první je uživatel dotázán, aby zadal adresář, kam se mají data uložit. Dále název souboru a formát. Zde je na výběr ze 3 datových formátů wav, flac a mp3. Následně je provedena kontrola, jestli již data pod stejným názvem neexistují, aby nedošlo k jejich nechtěnému přepsání. Pokud data pod stejným názvem již existují, následuje dotaz, jestli mají být data přepsána. Pokud ne, nastane výzva, aby uživatel zadal nový název. V opačném případě nebo pokud soubor se stejným názvem neexistuje, je volána funkce *saveDataProcess()*.

Zde dojde k dotazu, jaká data uživatel chce exportovat. Možnosti výběru závisí na zvoleném režimu nahrávání. Na výběr jsou surová zvuková data označena jako RAW, následně data zakódovaná do ambisonie a data po beamformingu. Následně dojde k exportu dat podle zvolených možností. Každý kanál je uložen pod jedinečným názvem. Před samotným exportem je potřeba provést sjednocení buněk, ve kterých jsou data uložena. A také upravit rozsah dat pro export pomocí normalizace na

rozsah hodnot od -1 do 1, kterou vyžaduje Matlab funkce *audiowrite()*, která je pro export do zvukového souboru využita. Jelikož data z mikrofonů přicházejí ve formátu 24 bitů na vzorek, je toto rozlišení zachováno i při exportu.



Obr. 4.16: Export dat do zvukového souboru.

4.3.7 Ovládání pomocí GUI

Pro zjednodušení ovládání programu bylo do aplikace zakomponováno grafické uživatelské rozhraní. Matlab obsahuje rozšíření pro tvorbu GUI App Designer, které pro tuto část práce bylo využito. Výsledné GUI je zobrazeno na obrázku 4.17. Samotné GUI společně s vazbami na zpracování dat je součástí souboru *app.mlapp*. GUI je rozděleno na dvě části a to na část pro nastavení a zobrazení.

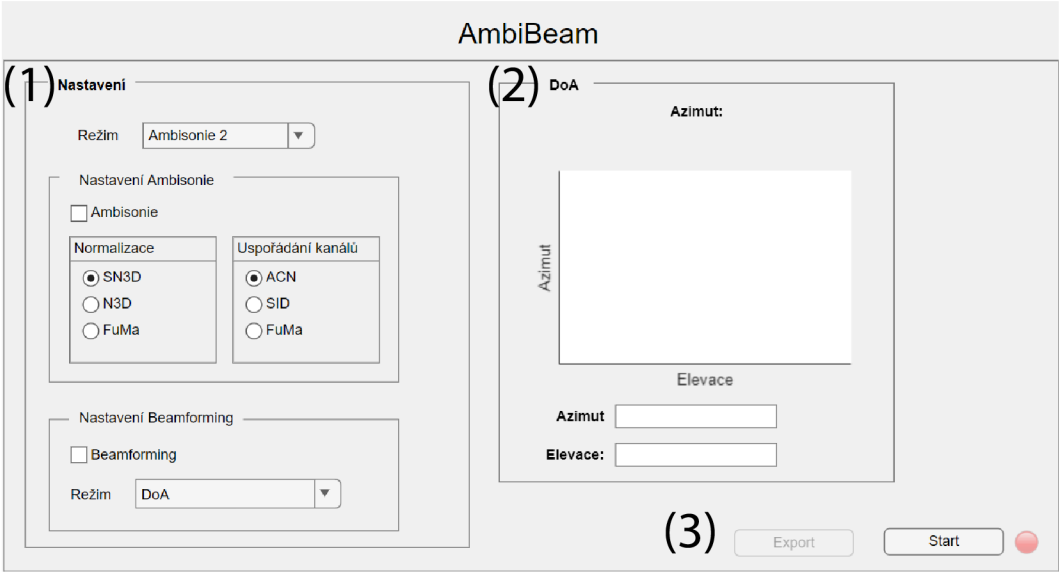
V části (1) je nastavení aplikace. Režim aplikace nastavuje uspořádání mikrofonů, která jsou připojena k FPGA. Kvůli testování a ladění hardwaru zde byly ponechány i možnosti pro všechny kombinace mikrofonů, ale samotné zpracování ambisonie a beamformingu je možné provádět jen s 16 mikrofony na lince, takže jen v režimech

Ambisonie 16, *Beamforming 16* a jejich kombinace *AmbiBeam*. Záznam zvuku ze snímačů se provádí pro všechny režimy.

Pokud uživatel zvolí jiný režim než tyto tři, v GUI se znemožní výběr a povolení ambisonie a beamformingu a zamezí se tím špatnému nastavení aplikace. Pokud bude uživatel v režimu, ve kterém je možné použít ambisonii nebo beamforming, a přejde do režimu, ve kterém to není možné, dojde k deaktivaci zvolených metod.

Dále je zde nastavení samotné ambisonie. Aby se při nahrávání zvuk kódoval rovnou do ambisonie, se dá aplikaci najevo pomocí komponenty *checkBox* v nastavení ambisonie. Ve výchozím nastavení je ambisonie v uspořádání ACN a normalizaci SN3D. To lze ale také v GUI nastavit na uvedené tři normalizace *SN3D*, *N3D*, *FuMa* a uspořádání *ACN*, *SID*, *FuMa*. Je možné zvolit pouze jedno uspořádání a jednu normalizaci, aplikace není přizpůsobena kódování do ambisonie pro více normalizaci a uspořádání zároveň.

Část nastavení pro beamforming obsahuje opět povolení zpracování metody beamforming pomocí komponenty *checkBox*. Beamforming lze spustit v režimu *DoA*, který pouze lokalizuje směr pozice zdroje zvuku, anebo v režimu prostorového filtrování.



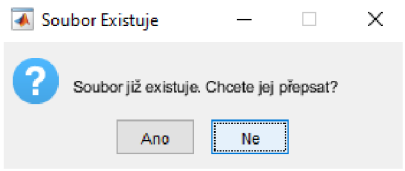
Obr. 4.17: Grafické uživatelské rozhraní.

V části (2) je zobrazení odhadu směru pozice zdroje zvuku a výpis jednotlivých složek souřadnic azimutu a elevace. Zobrazení *DoA* funguje jen v případě, že je v provozu Beamforming v jednom ze dvou režimů.

Dále má aplikace v části (3) tlačítka pro ovládání aplikace. Pravé tlačítko Start/Stop slouží k zahájení nahrávání společně se zpracováním dat. V závislosti na stavu aplikace mění svůj vzhled a rozsvěcuje signalizaci nahrávání napravo. Po dokončení

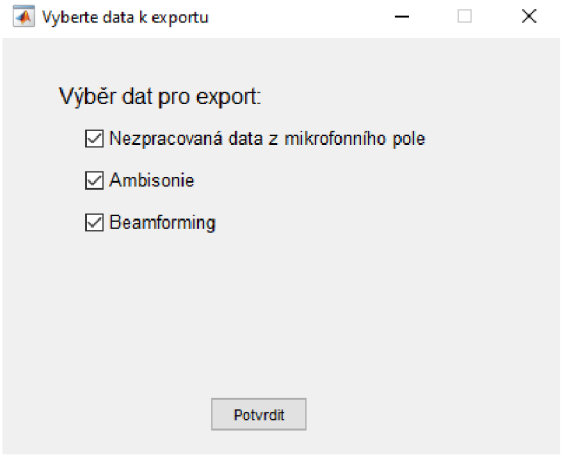
nahrávání, se stop tlačítkem chod aplikace zastaví, dojde i k ukončení běhu FPGA a umožní se volba *Export*. Celý proces Export byl podrobně popsán v kapitole 4.3.6, ale zde bude vysvětleno GUI, které export doprovází.

Po zvolení exportu dat je zobrazen standardní ukládací dialog operačního systému, kde uživatel zadá název a formát souboru. Následně se na pozadí provede kontrola, jestli soubor se stejným názvem i s příponou, kterou k názvu automaticky doplní aplikace, již existuje. Pokud ano, zobrazí se dialogové okno s dotazem, jestli se mají soubory přepsat 4.18.



Obr. 4.18: Dialogové okno existujícího souboru.

Po vyřešení kolize dojde k zobrazení dialogového okna 4.19, kde uživatel může vybrat, co si přeje uložit. Možnosti výběru opět závisí na režimu aplikace. Všechny tři možnosti jsou uskutečnitelné pouze v režimu *AmbiBeam*. V režimu *Ambisonie 16* není možné exportovat data pro beamforming a v režimu *Beamforming 16* zas není možné exportovat data pro ambisonii. Pokud je zvolen jiný režim pro 2-8 mikrofonů, je možné exportovat pouze zvuková data bez použití metod zpracování. Po exportu dat dojde ke smazání dat z paměti aplikace pomocí funkce *resetMem()*, aby mohlo být spuštěno nové nahrávání.

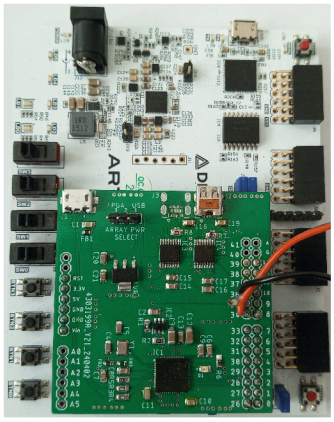


Obr. 4.19: Dialogové okno výběr souborů.

5 Zhodnocení

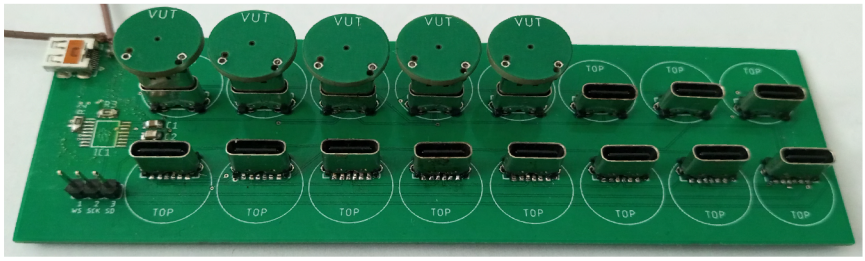
Výsledkem práce byla realizace získávání a zpracování zvuku ze sférického mikrofonního pole MEMS senzorů s TDM rozhraním za pomocí FPGA. V FPGA bylo realizováno řízení mikrofonů, za pomocí pravidelně generovaných řídících signálu *SCK* a *WS*. Frekvence a tvar průběhů byly nejprve odsimulovány v prostředí Vivado a následně změřeny osciloskopem se sondou, která byla kalibrována do 150 MHz. Počáteční 85 ms inicializace mikrofonů byla pro ověření pouze odsimulována. Pro řízení generování těchto signálů byl vytvořen stavový automat, který na základě řídících příkazů z počítačové aplikace mění nastavení řídících signálů v závislosti na počtu mikrofonů na lince. Jeho funkčnost byla nejprve ověřena pomocí simulace ve Vivado a následně ověřena na použitém FPGA.

Komunikace mezi FPGA a počítačem byla prvně realizována pomocí IP jádra Data Capture, které bylo získáno pomocí Matlab rozšíření HDL Verifier. Tato realizace se ukázala jako nepoužitelná pro potřeby této práce, protože přenos byl pomalý. Z tohoto důvodu byl do řetězce implementován USB převodník FTDI232. Pro jeho implementaci musel být napsán samostatný modul, které zajišťoval komunikaci mezi FPGA a FTDI. Modul pro správu FTDI byl také odsimulován, ale samotnou komunikace mezi počítačem a FPGA přes FTDI nebylo možné simulovat. Ověření funkčnosti ve směru komunikace z počítače do FPGA bylo provedeno pomocí skriptu v prostředí Matlab, který přes FTDI převodník posílal řídící signály do FPGA a FPGA na ně reagovalo prostřednictvím LED diod a následně i signály *SCK* a *WS*. V opačném směru byla komunikace ověřena pomocí naprogramovaného modulu v FPGA, který generoval přesně stanovenou posloupnost čísel a přes FTDI tyto čísla byla odesílána do prostředí Matlab a následně vyhodnocena jejich správnost. Díky tomu bylo možné ověřit, jestli nedochází k deformaci a ztrátě dat.



Obr. 5.1: FPGA s navrženým FTDI modulem.

Jako poslední modul v rámci FPGA byl realizován příjem dat TDM výstupu MEMS mikrofonů. Jako všechny moduly byl nejprve odsimulován a následně ověřen na reálném hardwaru. Ověření funkčnosti bylo realizováno pomocí přípravku s mikrofony, ke kterému bylo připojeno FPGA. Postupně se byly vyzkoušeny všechny mikrofony samostatně pro nastavené FPGA pro linku se dvěma mikrofony, tím byly vyřazeny nefunkční mikrofony, které by způsobily špatně odhalitelné chyby v celém řetězci. Následně bylo do řetězce přidáváno více mikrofonů a měněno nastavení FPGA pro daný počet mikrofonů na lince. Důležitou částí příjmu dat, bylo naladit impuls vzorkování dat z datové sběrnice na přesně definovaný okamžik, kdy jsou na sběrnici data dostupná. Vzorkovací impuls musel být posunut o 20,2 ms od náběžné hrany hodinového signálu *SCK* pro mikrofony. Kombinace mikrofonů 2, 4 a 8 byly bez problému zprovozněny a data z mikrofonů ověřeny pomocí zpracování v počítači, kdy se z 8 bitových dat z FTDI vytvořily zvukové signály a ty bylo možné přehrát. U nastavení pro 16 mikrofonů nastal problém. Pro kombinaci 16 mikrofonů se na přípravku podařilo zprovoznit maximálně 13 mikrofonů. Po připojení dalších mikrofonů, byl signál z pole mikrofonů kompletně degradován.



Obr. 5.2: Testovací přípravek s MEMS mikrofony.

Při diagnostice chyby bylo zjištěno, že na testovacím přípravku dochází ke ztrátě napětí u řídících signálů a na pinech dalších mikrofonů dochází k úbytku napětí na $U_{\check{s}\check{s}} = 1{,}5\,\mathrm{V}$, což neodpovídá 3,3 V logice, kterou produkuje FPGA, a další mikrofony v řetězci tak nerozeznají úrovně log. 0 a log. 1.

V prostředí Matlab byla naprogramována aplikace pro řízení a získávání dat z FPGA. Základ aplikace tvořily již ověřené testovací skripty pro komunikaci a získávaní dat, ale modifikované pro použití v aplikaci s grafickým uživatelským rozhraním. Jelikož se celý postprocessing děje v reálném čase hned během nahrávání, bylo potřeba ověřit, jestli aplikace zvládá dostatečně rychle přijímat data z FTDI převodníku a zpracovávat je. Kdyby získávání dat bylo pomalé, došlo by k přetečení vnitřní paměti FTDI převodníku, následně i k přetečení FIFO paměti v FPGA a tím ke ztrátě dat. Pro testování byla do aplikace implementována další paměťová struktura, která do své paměti ukládala stavy FTDI paměti. Z nasbíraných dat o stavu

paměti během nahrávání pak bylo možné ověřit dostatečnou rychlost komunikace mezi Matlab aplikací a FTDI.

Rychlost předzpracování 8 bitových dat do 32 bitových byla během nahrávání kontrolována pomocí záznamu indexů čtení a zápisu. Algoritmus pro předzpracování musel být několikrát optimalizován, aby doba trvání výpočtu nepřekročit periodu volání.

Metoda ambisonie pro 16 kanálů, která je v aplikaci implementována, je také prováděna hned během nahrávání. Původní implementace byla provedena pomocí několika zanořených smyček *for*. Rychlost zpracování přidělené paměti dat, ale nebyla optimální, a proto se výpočet implementoval pomocí maticového výpočtu, pro který je Matlab optimalizovaný. Po maticové optimalizaci byly dosažené výsledky kontrolovány s původní snadnou implementací pomocí *for* smyček.

Metoda beamforming, byla realizována taky pro 16 mikrofonů. Pro lokalizaci směru zdroje zvuku byla využita metoda rozkladu sférických harmonických funkcí. Metoda je náročná na výpočet a nebylo možné dosáhnout optimální doby provádění. Byla zde snaha docílit paralelního zpracování metody, ale Matlab nemá moc možností paralelního výpočtu jako jiné programovací jazyky. Pomocí funkcí *parfor()*, *parfeval()* a *spmd()* výpočet trval ještě déle. Proto zde byl zvolen kompromis a musela být zvětšena perioda opakování a pro ni přizpůsobena velikost zpracovávaného množství dat. Na rozklad sférických harmonických funkcí již byla navázána lokalizace směru, vytvoření vektoru váhovacích koeficientů a prostorová filtrace pomocí beampaternu. Funkce byla ověřena simulací, ve které byly signály s mikrofonního pole generovány v závislosti na směru zdroje zvuku a následně byla provedena kontrola, jestli referenční směr odpovídá odhadnutému.

Algoritmy pro ambisonii a beamforming nemohly být ověřeny na sférickém mikrofonním poli, protože sférické mikrofonní pole, jehož návrhem a konstrukcí se zabývala jiná práce, nebylo dokončeno. Je možné, že pro použití v reálných podmínkách budou muset být algoritmy upraveny, protože se zde objeví problémy, které simulace neodhalily.

Pro aplikaci bylo také vytvořeno grafické uživatelské rozhraní pro snadné ovládání aplikace s možností exportu zvukových dat do zvukových souborů.

Závěr

V teoretické části práce byly vysvětleny principy mikrofonních polí se zaměřením na sférická mikrofonní pole viz kapitola 1. Dále byly v práci popsány základní metody pro signálové zpracování dat z mikrofonního pole v kapitole 2, jako je tvarování přijímače pro prostorovou filtraci, lokalizace směru zdroje zvuku pomocí rozkladu sférických harmonických funkcí a ambisonie. V hardwarové části práce 3 byly vysvětleny principy fungování MEMS mikrofonů se zaměřením na mikrofony s TDM rozhraním kvůli následné implementaci. Rozhraní mezi mikrofonním polem a počítačem bylo vytvořeno za pomocí FPGA a FTDI převodníku, jejichž možnosti použití byly popsány v teoretické části 3.2 a 3.3.

Pro vytvoření vhodné implementace bylo potřeba zjistit vlastnosti FPGA a mikrofonů ICS-52000, které byly přiděleny pro tuto práci. Následně byla provedena a vysvětlena implementace jednotlivých modulů v kapitole 4.1 pomocí programovacího jazyka VHDL na FPGA pro generování periodických signálů, na kterých závisí funkčnost MEMS mikrofonů. Dále byl implementován příjem dat z mikrofonů pomocí TDM rozhraní, pro správnou funkčnost celého návrhu musela být naprogramována i řídící logika. V další fázi bylo implementováno rozhraní FTDI v režimu FIFO 245, které se stará o komunikace mezi FPGA a počítačem. Po naprogramování jednotlivých modulů byla ověřena jejich funkčnost pomocí simulací a měření.

V rámci diplomové práce se na testovacím přípravku o jedné TDM lince povedlo zprovoznit až 13 mikrofonů z 16. Problém nebyl v softwarové sekci, ale v návrhu hardwaru, kterým se tato práce nezabývala. Testování neprobíhalo přímo na sférickém mikrofonním poli, protože sférické mikrofonní pole, jehož návrhem a konstrukcí se zabývala jiná práce, nebylo dokončeno.

Dále byla naprogramována aplikace v prostředí Matlab viz kapitola 4.3, která zpracovává vícekanálový signál prostorového zvuku. Byla navržena pro zpracování dat z mikrofonního pole, které mělo mít optimalizované rozmístění 16 mikrofonů pro ambisonii a 16 mikrofonů pro beamforming. Aplikace umožňuje nahrání zvukových dat až z 32 mikrofonů ze sférického mikrofonního pole současně. Aplikace převádí data z FPGA na zvuková data roztříděná do kanálů. Má implementovanou metodu ambisonie s nastavením normalizace a uspořádáním kanálů a metodu pro prostorové filtrování, u které je lokalizace směru zdroje zvuku realizována pomocí rozkladu sférických harmonických funkcí a následné filtrování pomocí řízeného paprsku, který směrovou charakteristikou odpovídá hyperkardiodě. Pro snadné ovládání aplikace bylo vytvořeno grafické uživatelské rozhraní, které umožňuje snadné ovládání.

Výsledky této práce poskytují teoretický základ a popisují softwarový balík, který byl otestován a změřen, spolu s celkovým konceptem propojení, snímání a záznamu prostorového zvuku pomocí mikrofonního pole.

Literatura

[1] MEYER, Jens a ELKO, Gary, 2002. A highly scalable spherical microphone array based on an orthonormal decomposition of the soundfield. Online. *IEEE International Conference on Acoustics Speech and Signal Processing*. S. II-1781-II-1784. ISBN 0-7803-7402-9. Dostupné z: <https://doi.org/10.1109/ICASSP.2002.5744968.> [cit. 2023-12-01].

[2] RAFAELY, Boaz, 2005. Analysis and Design of Spherical Microphone Arrays. Online. *IEEE Transactions on Speech and Audio Processing*. S. 135-143. ISSN 1558-2353. Dostupné z: <https://ieeexplore.ieee.org/document/1369318> [cit. 2023-09-10].

[3] ABHAYAPALA, Thushara a WARD, Darren, 2002. Theory and design of high order sound field microphones using spherical microphone array. Online. *IEEE International Conference on Acoustics, Speech, and Signal Processing*. S. II-1949-II-1952. ISBN 0-7803-7402-9. Dostupné z: <https://ieeexplore.ieee.org/abstract/document/5745011> [cit. 2024-05-03].

[4] JIN, Craig; EPAIN, Nicolas a PARTHY, Abhaya, 2013. Design, Optimization and Evaluation of a Dual-Radius Spherical Microphone Array. Online. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. S. 193-204. ISSN 2329-9304. Dostupné z: <https://ieeexplore.ieee.org/document/6645392> [cit. 2024-05-03].

[5] ZOTTER, Franz a FRANK, Matthias, 2019. Ambisonics. Online. Springer. ISBN 978-3-030-17207-7. Dostupné z: <https://doi.org/https://doi.org/10.1007/978-3-030-17207-7>. [cit. 2024-05-03].

[6] COHEN, Israel; BENESTY, Jacob a GANNOT, Sharon, 2010. Speech Processing in Modern Communication. Online. Springer. ISBN 978-3-642-11130-3. Dostupné z: <https://link.springer.com/book/10.1007/978-3-642-11130-3. [cit. 2024-03-03].

[7] BENESTY, Jacob a CHEN, Jingdong, 2012. *Study and Design of Differential Microphone Arrays*. Springer International Publishing. ISBN 978-3-642-33753-6.

[8] BRANDSTEIN, Michael a WARD, Darren, 2001. Microphone Arrays Signal Processing Techniques and Applications. Online. Springer. ISBN 978-3-662-04619-7. Dostupné z: <https://doi.org/10.1007/978-3-662-04619-7>. [cit. 2024-05-03].

[9] BENESTY, Jacob; CHEN, Jingdong a COHEN, Israel, 2015. Design of Circular Differential Microphone Arrays. Online. Springer. ISBN 978-3-319-14842-7. Dostupné z: <https://link.springer.com/book/10.1007/978-3-319-14842-7>. [cit. 2024-05-03].

[10] DU, Chaoran; LECLERE, Quentin a LI, Baojiang, Design and evaluation of open spherical microphone arrays. In *24th International congress on sound and vibration*, London, United Kingdom: ICVS, 2017. HAL-01595873. Dostupné z: <https://www.researchgate.net/publication/337679960_DESIGN_AND_EVALUATION_OF_OPEN_SPHERICAL_MICROPHONE_ARRAYS>.

[11] PICHLER, Thomas, 2016. Circular Differential Microphone Arrays. Online, Diplomová práce. Graz: Graz University of Technology, Signal Processing and Speech Communications Laboratory. Dostupné z: <https://www.spsc.tugraz.at/student-projects/circular-differential-microphone-arrays.html>. [cit. 2024-05-03].

[12] PLESSAS, Peter, 2009. Rigid Sphere Microphone Arrays for Spatial Recording and Holography. Online, Diplomová práce, vedoucí Franz Zotter. Graz: Graz University of Technology. Dostupné z: <https://plessas.mur.at/rnd/da/Thesis_Plessas.pdf>. [cit. 2024-05-03].

[13] BALMAGES, Ilya a RAFAELY, Boaz, 2007. Open-Sphere Designs for Spherical Microphone Arrays. Online. *IEEE Transactions on Audio, Speech and Language Processing*. Roč. 15, č. 2, s. 727-732. ISSN 1558-7916. Dostupné z:<https://doi.org/10.1109/TASL.2006.881671.>. [cit. 2023-12-01].

[14] BENESTY, Jacob; CHEN, J. a HUANG, Yiteng, 2008. *Microphone array signal processing*. Dostupné z:<https://doi.org/https://doi.org/10.1007/978-3-540-78612-2>. Online. Berlin: Springer. ISBN 978-3-540-78612-2. [cit. 2023-12-01].

[15] NOVOTNÝ, Martin, 2019. Zpracování zvuku pro sférická videa - sada výukových úloh. Online, Bakalářksá práce, vedoucí Rund František. Praha: České vysoké učení technické v Praze. Dostupné z: <https://dspace.cvut.cz/handle/10467/82837>. [cit. 2024-05-03].

[16] JARRETT, Daniel P.; HABETS, E.A.P a NAYLOR, Patrick A., 2017. *Theory and Applications of Spherical Microphone Array Processing*. Online. Springer Topics in Signal Processing. Cham: Springer International Publishing. ISBN 978-3-319-42211-4. Dostupné z:<https://doi.org/https://doi.org/10.1007/978-3-319-42211-4>. [cit. 2023-12-01].

[17] RAFAELY, Boaz, 2018. *Fundamentals of Spherical Array Processing*. Online. 2. Springer International Publishing. ISBN 978-3-319-99561-8. Dostupné z:<https://doi.org/https://doi.org/10.1007/978-3-319-99561-8>. [cit. 2023-12-01].

[18] GAO, Weijian; ZHAO, Hangfang a XU, Wen, 2016. Direction of arrival estimation based on spherical harmonics decomposition. Online. Monterey, CA, USA: IEEE. ISBN 978-1-5090-1537-5. Dostupné z: https://doi.org/10.1109/OCEANS.2016.7761288. [cit. 2024-01-03].

[19] GRYTHE, Jørgen, 2015. Beamforming algorithms-beamformers. Online. S. 5. Dostupné z:<https://api.semanticscholar.org/CorpusID:17996998.>. [cit. 2023-10-22].

[20] GIERLICH, Roland, 2015. Joint estimation of spatial and motional radar target parameters by multidimensional spectral analysis. Online. Dresden, Germany: IEEE. ISBN 978-3-9540-4853-3. ISSN 2155-5753. Dostupné z: <https://doi.org/10.1109/IRS.2015.7226282>. [cit. 2024-05-05].

[21] GAO, Weijian; ZHAO, Hangfang a XU, Wen, 2016. Direction of arrival estimation based on spherical harmonics decomposition. Online. In: . Monterey, CA, USA, s. 5. ISBN 978-1-5090-1537-5. Dostupné z: <https://doi.org/10.1109/OCEANS.2016.7761288>. [cit. 2024-05-05].

[22] Spherical Bessel Function, c2024. Online. ScienceDirect. Dostupné z: <https://www.sciencedirect.com/topics/mathematics/spherical-bessel-function> . [cit. 2024-05-14].

[23] ARTEAGA, Daniel, 2023. Introduction to Ambisonics. Online. S. 31. Dostupné z: <https://doi.org/10.5281/zenodo.7963105>. [cit. 2023-09-20].

[24] FARINA, Angelo, 2006. *A-format to B-format conversion*. Online. Dostupné z:<http://pcfarina.eng.unipr.it/Public/B-format/A2B-conversion/A2B.htm>. [cit. 2023-10-20].

[25] Ambisonic component ordering, c2024. Online. Audiokinetic. Dostupné z: <https://www.audiokinetic.com/fr/library/edge/?source=Help&id=ambisonics_channel_ordering>. [cit. 2024-05-03].

[26] DANIEL, Jerome, 2003. Spatial Sound Encoding Including Near Field Effect: Introducing Distance Coding Filters and a Viable, New Ambisonic Format. Online. S. 15. Dostupné z: <https://www.researchgate.net/publication/229100838_Spatial_Sound_Encoding_Including_Near_Field_Effect_Introducing_Distance_Coding_Filters_and_a_Viable_New_Ambisonic_Format/references>. [cit. 2024-05-03].

[27] FARINA, Angelo, 2022. Explicit Ambix formulas for High Order Ambisonics. Online. Angelo Farina's personal Home Page. Dostupné z: <http://www.angelofarina.it/Aurora/HOA_explicit_formulas.htm>. [cit. 2024-05-03].

[28] MEDIA ARTS AND TECHNOLOGY UNIVERSITY OF CALIFORNIA, SANTA BARBARA. Ambisonics. Online. Dostupné z: <https://w2.mat.ucsb.edu/240/D/notes/Ambisonics.html>. [cit. 2024-05-03].

[29] FARINA, Angelo, 2022. ACN-N3D formulas for High Order Ambisonics. Online. Angelo Farina's personal Home Page. Dostupné z: <http://www.angelofarina.it/Aurora/HOA_ACN_N3D_formulas.htm>. [cit. 2024-05-03].

[30] STMICROELECTRONICS,c2023. *MEMS Microphones*. Online. Dostupné z:<https://www.st.com/en/mems-and-sensors/mems-microphones.html>. [cit. 2023-9-12].

[31] FOX, Arthur, 2020. *What Is A MEMS Microphone*. Online. Dostupné z: <https://bit.ly/3RmO4t9>. [cit. 2023-09-12].

[32] LEWIS, Jerad, 2012. Common Inter-IC Digital Interfaces for Audio Data Transfer. Online. S. 3. Dostupné z: <https://www.analog.com/media/en/technical-documentation/technical-articles/ms-2275.pdf>. [cit. 2023-09-12].

[33] INVENSENSE, c2017. ICS-52000 Datasheet. Online. Dostupné z:<https://invensense.tdk.com/download-pdf/ics-52000-datasheet/>. [cit. 2023-09-12].

[34] MAXFIELD, Clive, 2004. *The Design Warrior's Guide to FPGAs*. Newnes. ISBN 0-7506-7604-3.

[35] NICOLLE, Jean. *JTAG 1 - What is JTAG*. Online. Fpga4fun.com. Dostupné z: <https://www.fpga4fun.com/JTAG1.html>. [cit. 2023-10-21].

[36] DIGILENT. *JTAG SMT1 Reference Manual*. Online. Dostupné z:<https://digilent.com/reference/programmers/jtag-smt1/reference-manual>. [cit. 2023-10-21].

[37] DAWOUD, Peter a DAWOUD, Shenouda, 2020. *Serial Communication Protocols and Standards RS232/485, UART/USART, SPI, USB, INSTEON,Wi-Fi and WiMAX*. River Publishers. ISBN 978-87-7022-154-2.

[38] DIGILENT. *Arty S7*. Olnine. Dostupné z URL: <https://digilent.com/reference/programmable-logic/arty-s7/start>.

[39] FTDI CHIP, c2024. Products FT2232HQ. Online. FTDI Chip. Dostupné z: <https://ftdichip.com/products/ft2232hq/> . [cit. 2024-05-04].

[40] FTDI CHIP, c2024. Products FT232HQ. Online. FTDI Chip. Dostupné z: <https://ftdichip.com/products/ft232hq/> . [cit. 2023-12-01].

[41] AMD XILINX, c2023. *Intellectual Property*. Online. Dostupné z:<https://www.xilinx.com/products/intellectual-property.html>. [cit. 2023-11-05].

[42] AMD XILINX. Clocking Wizard v6.0 LogiCORE IP Product Guide. Online. Dostupné z:<https://docs.xilinx.com/r/en-US/pg065-clk-wiz>. [cit. 2023-11-05].

[43] Tutorial – What is a Tri-State Buffer, c2022. Online. NAND LAND. Dostupné z: <https://nandland.com/create-tri-state-buffer-in-vhdl-and-verilog/>. [cit. 2023-12-01].

[44] TDI FT_PROG Utility, c2024. Online. FTDI Chip. Dostupné z: <https://ftdichip.com/wp-content/uploads/2020/07/AN_124_User_Guide_For_FT_PROG.pdf>. [cit. 2024-05-04].

[45] D2XX Programmer's Guide, c2023. Online. FTDI CHIP. FTDI Chip. Dostupné z: <https://ftdichip.com/wp-content/uploads/2023/09/D2XX_Programmers_Guide.pdf>. [cit. 2024-02-04].

[46] Timer, c2024. Online. MathWorks. Dostupné z: <https://www.mathworks.com/help/matlab/ref/timer.html>. [cit. 2024-02-02].

Seznam symbolů a zkratek

ACN	ambisonics channel number / číslo kanálu ambisonie
AD	analogově – digitální
ADP	analogově – digitální převodník
Ambi	ambisonie
ASIC	aplikačně specializovaný integrovaný obvod
Beam	beamforming
B_n	síla sférického módu
C	kapacita
clk	hodinový signál
d	vzdálenost
DA	digitálně – analogový
DIV FACT	dělícíc poměr
DoA	direction of arrival / směr příchodu
DSP	digital signal processing / číslicové zpracování signálu
f_clk	frekvence hodinového signálu
FIFO	first in first out / první dovnitř první ven
FPGA	field programmable gate array / programovatelné hradlové pole
FSM	finite state machine / konečný stavový automat
f_vz	vzorkovací frekvence
FuMa	Furse – Malham
GND	země elektronických obvodů
GUI	grafické uživatelské rozhraní
h_n	Hankelova sférikální funkce
i	imaginární jednotka

I2S	inter – ic sound
IP Core	intellectual property core
JTAG	joint test action group
k	vlnové číslo
L	zpoždění autokorelace
λ	vlnová délka
LED	elektro – luminiscenční dioda
LSB	nejméně významný bit
LZD	levý – zadní – dolní
MEMS	mikro – elektromechanickcý systém
Mik	mikrofon
MMCM	mixed mode clock manager
MSB	nevýznamnější bit
MV	minimum variance
MVS	minimum variance spectrum
MUSIC	multiple signal classification
n	řád sférické harmonické funkce
P_{n}^{m}	Legendrova funkce
ϕ	úhel azimutu
PLL	phase lock loop
PPD	pravý – přední – dolní
PZH	pravý – zadní – dolní
R	odpor
RAM	polovodičová paměť s přímým přístupem
ρ_0	akustiká hustota vzduchu

S	počet snímačů
SCK	hodinový signál pro mikrofon
SD	sériová dat z mikrofonů
SID	single index designation / označení jediným indexem
s(t)	zvukový signál
TAP	test access point
TCK	test clock
TDI	test data in
TDM	time division multiplexing
TDO	test data out
θ	úhel elevace
TMS	test mode select
UART	universal asynchronous receiver transmitter
USB	univerzální sériová sběrnice
V	vektor řízení
VDD	napájecí napětí
w	váhovací funkce
WS	word select / výběr slova
WSO	word select out
Y_{n}^{m}	koeficient sférické harmonické funkce
Z	akustika impedance

Seznam příloh

A Obsah elektronické přílohy

Elektronická příloha obsahuje adresář se zdrojovými kódy pro FPGA v jazyce VHDL a adresář se zdrojovými kódy pro prostředí Maltab. Pro vývoj zdrojových kódu v jazyce VHDL bylo použito prostředí Vivado ML Edition 2023.2 a pro aplikaci Matlab ve verzi R2023b. Pro testování byl využit notebook s procesorem Intel Core i7-4900MQ, s operační pamětí 16 GB a systémem Windows 10.

Z důvodu rozsáhlosti projektu nemohl být celý projekt v programu Vivado umístěn do školního informačního systému, a proto je celý projekt vytvořený v programu Vivado i projekt vytvořený v prostředí Matlab dostupný na adrese: `https://vutbr-my.sharepoint.com/:f:/g/personal/xtomes07_vutbr_cz/EtF2a-3z_8FPpznYChPCFzcB9_RcW45aJzSfsFm4qhUStQ?e=KIQc7i`.

- `/` — kořenový adresář přiloženého archivu
  - `Matlab` — zdrojové soubory počítačové aplikace
    - `app.mlapp` — spustitelný soubor
    - `ftd2xx.lib` — knihovna výrobce pro komunikaci s FTDI
    - `ftd2xx.h` — hlavičkový soubor ke knihovně ftd2xx.lib
    - `Ambisonie` — adresář s funkcemi pro ambisonii
    - `Beamforming` — adresář s funkcemi pro beamforming
    - `ExportData` — adresář s funkcemi pro export dat
    - `FTDI_function` — adresář s funkcemi pro komunikaci s FTDI
  - `FPGA` — adresář se zdrojovými kódy ve VHDL
    - `CLK_MAKING` — adresář s VHDL kódy pro tvorbu hodinového signálu
    - `FTDI` — adresář s VHDL kódy pro komunikace s FTDI
    - `TDM_interface` — adresář s VHDL kódy pro řízení mikrofonů
    - `XDC` — adresář s xdc souborem
    - `Top.VHDL` — soubor top modulu VHDL návrhu
    - `generovani_dat.VHDL` — soubor ve VHDL pro testování komunikace
    - `TOP_TB.VHDL` — soubor pro simulaci VHDL návrhu

B Implementace funkcionalit v jazyce VHDL

Výpis B.1: Příklad implementace děličky hodinového signálu v jazyce VHDL.

```
entity clk_divider is
  generic(
    DIV_FACT : positive := 4
  );
  port (
    clk      : in  std_logic;
    clk_out  : out std_logic
  );
end clk_divider;
architecture Behavioral of clk_divider is
  signal counter : unsigned(2 downto 0) := (others => '0');
begin
  process (clk) begin
    if rising_edge (clk) then
      if counter = (DIV_FACT - 1) then
        counter <= (others => '0');
        clk_out <= '1';
      else
        counter <= counter + 1;
        clk_out <= '0';
      end if;
    end if;
  end process;
end Behavioral;
```

Výpis B.2: Implementace generování hodinového signálu SCK pro mikrofony.

```
entity clk_out_mic is                                                    1
  port (                                                                 2
    clk           : in std_logic;                                        3
    clk_en        : in std_logic;                                        4
    clk_div       : in  std_logic_vector(5 downto 0);                    5
    SCK           : out std_logic                                        6
  );                                                                     7
end clk_out_mic;                                                         8
architecture Behavioral of clk_out_mic is                                9
  signal counter: unsigned(5 downto 0) := (others => '0');               10
  signal SCK_s  : std_logic := '0';                                      11
begin                                                                    12
  clock_divider : process (clk) begin                                    13
    if rising_edge (clk) then                                            14
      if clk_en = '1' then                                               15
        if counter = (unsigned(clk_div) - 1) then                        16
          SCK_s <= '1';                                                  17
          counter <= (others => '0');                                    18
        elsif counter >= 0 and counter <                                 19
        (unsigned(clk_div) / 2 - 1) then                                 20
          SCK_s <= '1';                                                  21
          counter <= counter +1;                                         22
        else                                                             23
          SCK_s <= '0';                                                  24
          counter <= counter + 1;                                        25
        end if;                                                          26
      else                                                               27
        counter <= (others => '0');                                      28
        SCK_s <= '0';                                                    29
      end if;                                                            30
    end if;                                                              31
  end process clock_divider;                                             32
  SCK <= SCK_s;                                                          33
```

Výpis B.3: Implementace generování hodinového signálu SCK pro FPGA.

```
entity clk_out_FPGA is                                                    1
  port (                                                                  2
    clk          : in  std_logic;                                         3
    clk_en       : in  std_logic;                                         4
    clk_div      : in  std_logic_vector(5 downto 0);                      5
    clk_out      : out std_logic                                          6
  );                                                                      7
end clk_out_FPGA;                                                         8
architecture Behavioral of clk_out_FPGA is                                9
  signal counter   : unsigned(5 downto 0) := (others => '0');             10
  signal clk_out_s : std_logic := '0';                                    11
begin                                                                     12
                                                                          13
clock_divider : process (clk) begin                                       14
  if rising_edge (clk) then                                               15
    if clk_en = '1' then                                                  16
      if counter = (unsigned(clk_div) - 1) then                           17
        clk_out_s <= '1';                                                 18
        counter <= (others => '0');                                       19
      else                                                                20
        counter <= counter + 1;                                           21
        clk_out_s <= '0';                                                 22
      end if;                                                             23
    else                                                                  24
      clk_out_s <= '0';                                                   25
      counter <= (others => '0');                                         26
    end if;                                                               27
  end if;                                                                 28
end process clock_divider;                                                29
clk_out <= clk_out_s;                                                     30
end Behavioral;                                                           31
```

Výpis B.4: Implementace generování synchronizačního signálu WS.

```
startUp_WS : process (clk) begin
  if rising_edge (clk) then
    if WS_counter_enable = '1' then
      if counter_1 = (startup - 2) then
        counter_1 <= (others => '0');
        startUpEnd_s <= '1';
      else
        counter_1 <= counter_1 + 1;
        startUpEnd_s <= '0';
      end if;
    else
      counter_1 <= (others => '0');
      startUpEnd_s <= '0';
    end if;
  end if;
end process startUp_WS;
WS_gen : process (clk) begin
  if rising_edge (clk) then
    if WS_enable = '1' then
      if counter_2 >= (div_fact_WS - 2) then
        WS_s <= '1';
        if counter_2 = (div_fact_WS - 1) then
          counter_2 <= (others => '0');
        else
          counter_2 <= counter_2 + 1;
        end if;
      elsif counter_2 <= (unsigned(length)/2 - 1) then
        counter_2 <= counter_2 + 1;
        WS_s <= '1';
      else
        counter_2 <= counter_2 + 1;
        WS_s <= '0';
      end if;
    else
       counter_2 <= (others => '0');
       WS_s <= '0';
     end if;
  end if;
end process WS_gen;
```

Výpis B.5: TDM_FSM první kombinační část.

```
combinatory_part_1 : process (pres_state, ctrl_data,
startUpEnd, clk_locked) begin
  case pres_state is
    when st_Idle => if ctrl_data = x"30" and clk_locked = '1'
        then next_state <= st_StartUp_2_Mic;
      elsif ctrl_data = x"31" and clk_locked = '1'
        then next_state <= st_StartUp_4_Mic;
                    ...........
      elsif ctrl_data = x"38" and clk_locked = '1'
        then next_state <= st_StartUp_16_Mic;
      else  next_state <= st_Idle; end if;
    when st_StartUp_2_Mic => if startUpEnd = '1' and
        ctrl_data = x"30" then next_state <= st_Ambi_2;
      elsif startUpEnd = '1' and ctrl_data = x"34"
        then next_state <= st_Beam_2;
      else  next_state <= st_StartUp_2_Mic; end if;
                    ..........
    when st_StartUp_16_Mic => if    startUpEnd = '1' and
        ctrl_data = x"33" then next_state <= st_Ambi_16;
      elsif startUpEnd = '1' and ctrl_data = x"37"
        then next_state <= st_Beam_16;
      elsif startUpEnd = '1' and ctrl_data = x"38"
        then next_state <= st_Ambi_Beam;
      else  next_state <= st_StartUp_16_Mic; end if;
    when st_Ambi_2 => if ctrl_data = x"39"
        then next_state <= st_Idle;
      else next_state <= st_Ambi_2; end if;
    when st_Ambi_4 => if ctrl_data = x"39"
        then next_state <= st_Idle;
      else next_state <= st_Ambi_4; end if;
                    ...........
    when st_Beam_16 => if ctrl_data = x"39"
        then next_state <= st_Idle;
      else next_state <= st_Beam_16; end if;
    when st_Ambi_Beam => if ctrl_data = x"39"
        then next_state <= st_Idle;
      else next_state <= st_Ambi_Beam; end if;
    end case;
  end process combinatory_part_1;
```

Výpis B.6: TDM_FSM druhá kombinační část.

```
combinatory_part_2 : process (pres_state) begin
  case pres_state is
    when st_Idle => clk_en_s <= '0';
                    ws_en_s  <= '0';
                    ws_counter_en_s <= '0';
                    ws_length_s <= (others => '0');
                    led_s <= (others => '0');
                    clk_div_s <= (others => '0');
                    TDM_en_s <= '0';
                    in_1_en_s <= '0';
                    in_2_en_s <= '0';
    when st_StartUp_16_Mic => clk_en_s <= '1';
                    ws_en_s <= '0';
                    ws_counter_en_s <= '1';
                    ws_length_s <= (others => '0');
                    led_s <= "0000";
                    clk_div_s <= "000100";
                    TDM_en_s <= '0';
                    in_1_en_s <= '0';
                    in_2_en_s <= '0';
                ..................
    when st_Ambi_16 => clk_en_s <= '1';
                       ws_en_s <= '1';
                       ws_counter_en_s <= '0';
                       ws_length_s <= "000001";
                       led_s <= "0001";
                       clk_div_s <= "000100"; -- 4
                       TDM_en_s <= '1';
                       in_1_en_s <= '1';
                       in_2_en_s <= '0';
    when st_Beam_16 => clk_en_s <= '1';
                       ws_en_s <= '1';
                       ws_counter_en_s <= '0';
                       ws_length_s <= "000001";
                       led_s <= "0001";
                       clk_div_s <= "000100"; -- 4
                       TDM_en_s <= '1';
                       in_1_en_s <= '0';
                       in_2_en_s <= '1';
                ..................
  end case;
end process combinatory_part_2;
```

Výpis B.7: Příjem dat z mikrofonů 1. část.

```
non_zero_data : process (clk) begin
  if rising_edge(clk) then
    if process_en = '1' then
      if valid_data = false then
        if zero_data_cnt = (zero_time - 1) then
          valid_data <= true;
          zero_data_cnt <= 0;
        else
          zero_data_cnt <= zero_data_cnt + 1;
          valid_data <= false;
        end if;
      end if;
    else
      zero_data_cnt <= 0;
      valid_data <= false;
    end if;
  end if;
end process non_zero_data;

sampling_signal_gen : process (clk) begin
  if rising_edge (clk) then
    if process_en = '1' then
      if samp_counter = (unsigned(clk_div) - 1) then
        samp_counter <= 0;
        sampling_s <= '0';
      elsif samp_counter = 0 then
        samp_counter <= samp_counter + 1;
        sampling_s <= '1';
      else
        samp_counter <= samp_counter + 1;
        sampling_s <= '0';
      end if;
    else
      samp_counter <= 0;
      sampling_s <= '0';
    end if;
   end if;
end process sampling_signal_gen;
```

Výpis B.8: Příjem dat z mikrofonů 2. část.

```
reg : process (clk) begin -- pridavani bitu do bufferu
  if rising_edge (clk) then if process_en = '1' then
      if sampling_s = '1' then
        if counter < 24 then
          if in_1_en = '1' then
            data_reg_1_s(23 - counter) <= data_in_1;
          end if;
          if in_2_en = '1' then
            data_reg_2_s(23 - counter) <= data_in_2;
          end if;
        end if;
      end if;
  end if; end if;
end process reg;
data_to_out : process (clk) begin
  if rising_edge (clk) then
    if process_en = '1' then
      if sampling_s = '1' then
        if valid_data = true then
          data_ready_s <= '0';
          if in_1_en = '1' then
            if counter = 25 then
              data_out_s(31 downto 8) <= data_reg_1_s;
              data_out_s(7 downto 0) <= (others => '0');
            elsif counter = 26 then
              data_ready_s <= '1';
            end if;
          end if;
          if in_2_en = '1' then
            if counter = 27 then
              data_out_s(31 downto 8) <= data_reg_2_s;
              data_out_s(7 downto 0) <= x"01";
            elsif counter = 28 then
              data_ready_s <= '1';
            end if;
          end if;
        end if;
      else
        data_ready_s <= '0';
      end if;
    else
```

Výpis B.9: Modul FTDI.

```
component FTDI_SYNC_FIFO                                                    1
  port(                                                                     2
    clk_ftdi : in std_logic;                                                3
    data     : inout  std_logic_vector(7 downto 0);                         4
    rxf      : in     std_logic;                                            5
    txe      : in     std_logic;                                            6
    rd       : out    std_logic;                                            7
    oe       : out    std_logic;                                            8
    wr       : out    std_logic;                                            9
    siwu     : out    std_logic;                                            10
        .....                                                               11
  );                                                                        12
end component;                                                              13
                                                                            14
component fifo_data_from_pc                                                 15
  port (                                                                    16
    wr_clk : in  std_logic;                                                 17
    rd_clk : in  std_logic;                                                 18
    din    : in  std_logic_vector(7 downto 0);                              19
        .....                                                               20
  );                                                                        21
end component;                                                              22
                                                                            23
component fifo_data_to_pc                                                   24
  port (                                                                    25
    wr_clk       : in  std_logic;                                           26
    rd_clk       : in  std_logic;                                           27
    din          : in  std_logic_vector(31 downto 0);                       28
        .....                                                               29
  );                                                                        30
end component;                                                              31
                                                                            32
component clk_wiz_delay                                                     33
  port (                                                                    34
    clk_out          : out    std_logic;                                    35
    locked           : out    std_logic;                                    36
    clk_in           : in     std_logic                                     37
  );                                                                        38
end component;                                                              39
```

Výpis B.10: Modul FTDI SYNC FIFO.

```
-- Output register
process (clk_ftdi_delay, clk_delay_locked, clk_locked) begin
  if clk_locked = '0' then
    rd   <= '1';
    wr   <= '1';
    oe   <= '1';
    siwu <= '1';
    data_from_PC_to_fpga_wr_en <= '0';
    data_from_fpga_to_PC_rd_en <= '0';
    write_en_next_s <= '0';
  elsif rising_edge(clk_ftdi_delay) then
    rd   <= rd_s;
    wr   <= wr_s;
    oe   <= oe_s;
    siwu <= '1';
    data_from_PC_to_fpga_wr_en <= data_from_PC_to_fpga_wr_en_s
    data_from_fpga_to_PC_rd_en <= data_from_fpga_to_PC_rd_en_s
    write_en_next_s <= write_en_s;
  end if;
end process;

-- Read Data from FTDI/Write Data to FTDI
process(clk_ftdi) begin
  if rising_edge(clk_ftdi) then
   if pres_st = st_Read then
     if rxf = '0' then
       data <= (others => 'Z');
       data_from_pc_to_fpga_fifo <= data;
     end if;
   elsif pres_st = st_Write then
     if txe = '0' and write_en_next_s = '1'
     and write_en_s = '1' then
       data_from_pc_to_fpga_fifo <= (others => 'Z');
       data <= data_from_fpga_to_PC_from_fifo;
     end if;
   end if;
  end if;
end process;
```

C Implementace aplikace v prostředí Matlab

Výpis C.1: Funkce FTDI_init.

```
function [status, ftHandle] = FTDI_init()
FT_OK = 0;

deviceId = 0;
ftHandle = libpointer('voidPtr', 0);
status = calllib('d2xx', 'FT_Open', deviceId, ftHandle);

if status == FT_OK
    calllib('d2xx', 'FT_ResetDevice', ftHandle);
    calllib('d2xx', 'FT_SetBitMode', ftHandle, 0xff, 0x40);
    calllib('d2xx', 'FT_SetTimeouts', ftHandle, 1, 1);
    calllib('d2xx', 'FT_SetUSBParameters', ftHandle,...
    65536, 65536);
    calllib('d2xx', 'FT_SetFlowControl', ftHandle,...
    0x0100 , 0, 0);
else
    errordlg('Zařízení␣nebylo␣nalezeno','Device␣Error');
    return;
end
end
```

Výpis C.2: Funkce FTDI_write.

```
function bytesWritten = FTDI_write(ftHandle, mode)

    switch mode
        case "Ambisonie 2"
            dataToWrite = uint8('0');
        case "Ambisonie 4"
            dataToWrite = uint8('1');
        case "Ambisonie 8"
            dataToWrite = uint8('2');
        case "Ambisonie 16"
            dataToWrite = uint8('3');
        case "Beamforming 2"
            dataToWrite = uint8('4');
        case "Beamforming 4"
            dataToWrite = uint8('5');
        case "Beamforming 8"
            dataToWrite = uint8('6');
        case "Beamforming 16"
            dataToWrite = uint8('7');
        case "AmbiBeam"
            dataToWrite = uint8('8');
        case "stop"
            dataToWrite = uint8('9');
    end

    BytesWritten = libpointer('uint32Ptr', 1);

    [~, ~, bytesWritten] = calllib('d2xx', 'FT_Write',...
    ftHandle, dataToWrite, length(dataToWrite), BytesWritten);
end
```

Výpis C.3: Funkce FTDI_status.

```
function [status, RxQueue, TxQueue, EventStatus] =
FTDI_status(ftHandle)

    RxQueue = libpointer('uint32Ptr', uint32(0));
    TxQueue = libpointer('uint32Ptr', uint32(0));
    EventStatus = libpointer('uint32Ptr', uint32(0));

    [status , RxQueue, TxQueue, EventStatus] = ...
    calllib('d2xx', 'FT_GetStatus', ftHandle,...
    RxQueue, TxQueue, EventStatus);

end
```

Výpis C.4: Funkce FTDI_read.

```
function [status, dataRead, bytesRead] = ...
FTDI_read(ftHandle, numBytesToRead)
    buffer = libpointer('uint8Ptr', zeros(1, numBytesToRead));
    bytesRead = libpointer('uint32Ptr', 0);

    status = calllib('d2xx', 'FT_Read', ftHandle,...
    buffer, numBytesToRead, bytesRead);

    if status == 0
        dataRead = buffer.Value;
        bytesRead = bytesRead.Value;
    else
        return
    end
end
```

Výpis C.5: Funkce FTDI_end.

```
function FTDI_end(ftHandle)
    calllib('d2xx', 'FT_Close', ftHandle);
end
```

Výpis C.6: Nastavení časovače pro příjem dat.

```
function timerFTDI = setupTimerFTDI(app)                                    1
    timerFTDI = timer;                                                      2
    timerFTDI.Period = 0.004;                                               3
    timerFTDI.TasksToExecute = Inf;                                         4
    timerFTDI.ExecutionMode = 'fixedRate';                                  5
    timerFTDI.BusyMode = 'drop';                                            6
    timerFTDI.TimerFcn = @(myTimerObj, thisEvent)...                        7
    readingDataFromFTDI(app, app.chunkSizeFTDI, app.ftHandle);              8
end                                                                         9
```