VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

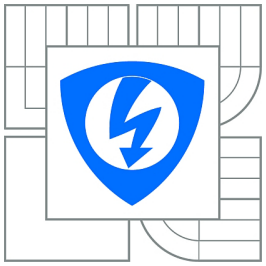
SYSTÉM PRO MONITOROVÁNÍ A NASTAVENÍ SIGNÁLOVÉHO PROCESORU V DIGITÁLNÍM REPRODUKTOROVÉM SYSTÉMU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE JAKUB LANÍK
AUTHOR

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

SYSTÉM PRO MONITOROVÁNÍ A NASTAVENÍ SIGNÁLOVÉHO PROCESORU V DIGITÁLNÍM REPRODUKTOROVÉM SYSTÉMU

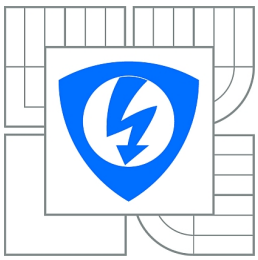
SYSTEM FOR MONITORING AND SETUP OF SIGNAL PROCESSOR IN DIGITAL LOUDSPEAKER SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — JAKUB LANÍK
AUTHOR

VEDOUCÍ PRÁCE — DOC. ING. TOMÁŠ KRATOCHVÍL, PH.D.
SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student:	Bc. Jakub Laník	***ID:***	115215
Ročník:	2	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

Systém pro monitorování a nastavení signálového procesoru v digitálním reproduktorovém systému

POKYNY PRO VYPRACOVÁNÍ:

Proveďte rozbor problematiky zpracování signálu v DSP s ohledem na aplikaci digitálního reproduktorového systému. Uvažujte nadřízený SW s GUI, který bude umožňovat komunikaci se zvoleným DSP a dále ovládat virtuální digitální reproduktorový systém, reprezentovaný vývojovým kitem s DSP. SW by měl být připraven pro budoucí implementaci na skutečném HW se shodným DSP.

Realizujte kompletní SW pro kontrolu digitálního reproduktorového systému s obousměrnou komunikací s DSP. Pro realizaci SW použijte Matlab, LabView nebo jiný objektově orientovaný systém.

Odlaďte a ověřte funkčnost systému včetně GUI podle pokynů vedoucího práce.

DOPORUČENÁ LITERATURA:

[1] SMÉKAL, Z., SYSEL, P. Signálové procesory. Praha: Sdělovací technika, 2006.

[2] JÁN, J. Číslicové zpracování, filtrace a analýzy signálů. Brno: Vutium, 2002.

[3] SMITH, W. S. The Scientist and Engineer's Guide to Digital Signal Processing [online]. Dostupné na www: http://www.dspguide.com

Termín zadání:	11.2.2013	***Termín odevzdání:***	24.5.2013

Vedoucí práce: doc. Ing. Tomáš Kratochvíl, Ph.D.
Konzultanti diplomové práce:

prof. Dr. Ing. Zbyněk Raida
Předseda oborové rady

ABSTRAKT

Projekt se zabývá teoretickým návrhem digitálního reproduktorového systému a jeho nadřazeným ovládacím softwarem s GUI. Tento digitální systém je složen z jedné hlavní jednotky Master, která má za úkol digitalizovat různé audio signály a poté je streamovat přes ethernet do podružných jednotek Slave. Ovládací software má za úkol tento systém monitorovat a konfigurovat.

KLÍČOVÁ SLOVA

ARM, Cortex M4, Ethernet, Qt, USB, STM32F4

ABSTRACT

This project solves teoretical design of digital reproductor system and controlling GUI software. This digital system is composed of one master unit, which digitalize different audio sources and stream them by ethernet to slave units. The controlling software is be able to set and monitor the audio system.

KEYWORDS

ARM, Cortex M4, Ethernet, Qt, USB, STM32F4

LANÍK, J. *Systém pro monitorování a nastavení signálového procesoru v digitálním reproduktorovém systému*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 73 s. Vedoucí diplomové práce doc. Ing. Tomáš Kratochvíl, Ph.D..

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Systém pro monitorování a nastavení signálového procesoru v digitálním reproduktorovém systému“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Doc. Ing. Tomáši Kratochvílovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé semestrální práce.

V Brně dne ....

(podpis autora)

Výzkum realizovaný v rámci této diplomové práce byl finančně podpořen projektem CZ.1.07/2.3.00/20.0007 **Wireless Communication Teams** operačního programu **Vzdělávání pro konkurenceschopnost**.

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Finanční podpora byla poskytnuta Evropským sociálním fondem a státním rozpočtem České republiky.

Tento příspěvek vzniknul za podpory projektu CZ.1.07/2.3.00/20.0007 WICOMT, financovaného z operačního programu Vzdělávání pro konkurenceschopnost

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Digitální reproduktorový systém je složen z centrální jednotky, která má za úkol přijímat a případně digitalizovat audio signály z různých zdrojů (např. CD přehrávač) a v různých formátech (analogově, SPDIF), tyto signály zpracovat např. filtrovat a v posledním kroku distribuovat po dostatečně rychlém přenosovém médiu do dalšího bloku. Tento blok, slave, má za úkol přijatý digitální signál z centrální jednotky vhodně upravit a převést na analogový signál pro buzení reproduktorů.

Celý projekt "Digitální reproduktorvý systém" je rozdělen na tři samostatné části, kde každý člověk řeší svou část. Každá část by měla být schopna pracovat alespoň pro demonstrační účely autonomně a také musí být schopna pracovat s ostatními celky jako jeden systém.

Tento projekt tedy řeší návrh a realizaci ovládacího software s grafickým uživatelským rozhraním GUI, který bude mít za úkol kompletní nastavení celého systému a jeho monitorování.

Pro vývoj ovládacího software byl také napsán testovací program, který simuluje jednotku master pro jednodušší implementaci komunikačních protokolů a nepotřebnosti skutečného hardwaru pro testování.

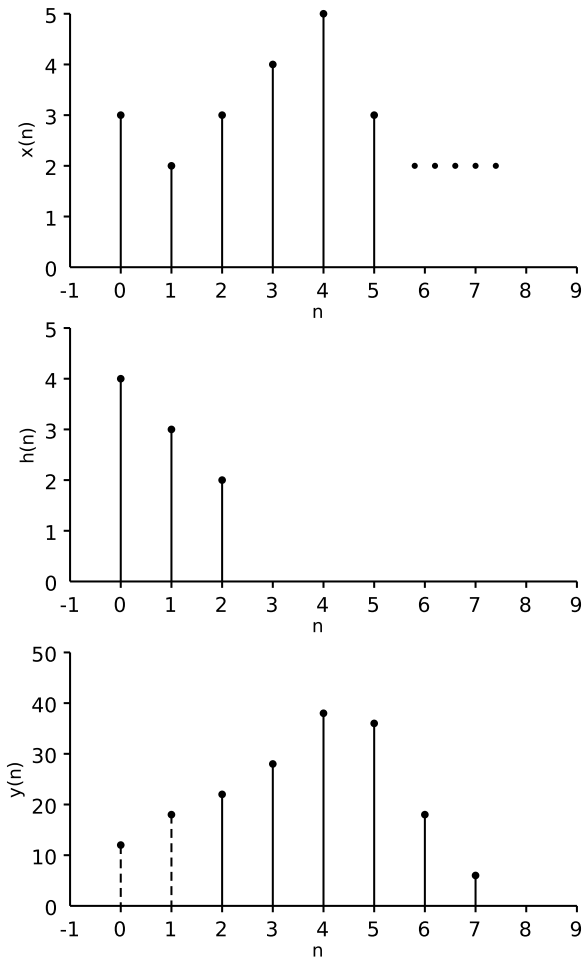
1 ZPRACOVÁNÍ SIGNÁLU V DSP

Digitální signálový procesor neboli DSP [1] [5] je mikrokontrolér speciálně uzpůsoben na matematické operace probíhající v reálném čase. Velmi důležitá součást DSP je jednotka, která umožňuje provádět matematické operace s čísly s plovoucí desetinou čárkou neboli FPU [4].

V tomto projektu bude DSP použit hlavně k realizaci signálových filtrů pomocí výpočtu konvoluce [2] [5] v časové oblasti:

$$y(n) = x(n) * h(n) = \sum_{i=0}^{M} x(i) \cdot h(n-i) = \sum_{i=0}^{M} h(i) \cdot x(n-i) \tag{1.1}$$

y(n) ... výstupní signál
x(n) ... vstupní signál
h(n) ... impulsní charakteristika systému - odezva systému na jednotkový impuls
M ... délka impulsní charakteristiky



Obr. 1.1: Grafické znázornění konvoluce

Z obrázku 1.1 je vidět přechodový jev před ustálením filtru (první dvojce vzorků) a časové zpoždění výstupního signálu $y(n)$. Tento jev se v praxi projevuje zpožděním systému o několik vzorků, které přímo závisí na délce impulsní charakteristiky systému.

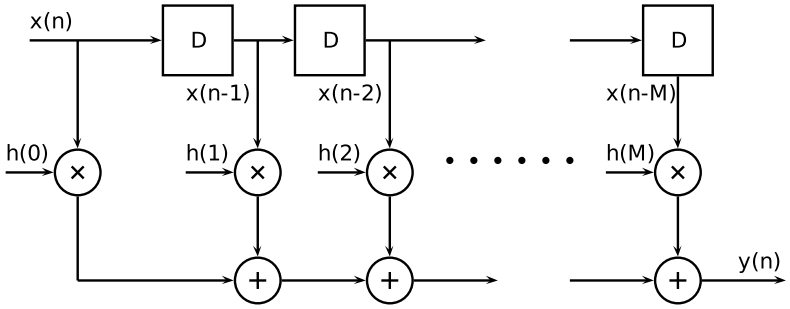
1.1 Filtry

Jak bylo psáno výše, DSP bude primárně použit pro realizaci filtrů, existují dva hlavní typy filtrů [3] [2]:

- Filtr s konečnou impulsní charakteristikou - Finite impulse response (FIR)
- Filtr s nekonečnou impulsní charakteristikou - Infinite impulse response (IIR)

1.1.1 Filtr FIR

Princip FIR filtru je patrný z obrázku 1.2. Signál je postupně zpožďován o jednotlivé vzorky v blocích D (delay) a jednotlivě zpožděné vzorky jsou násobeny příslušnými koeficienty filtru $h(0..M)$ a poté sečteny.



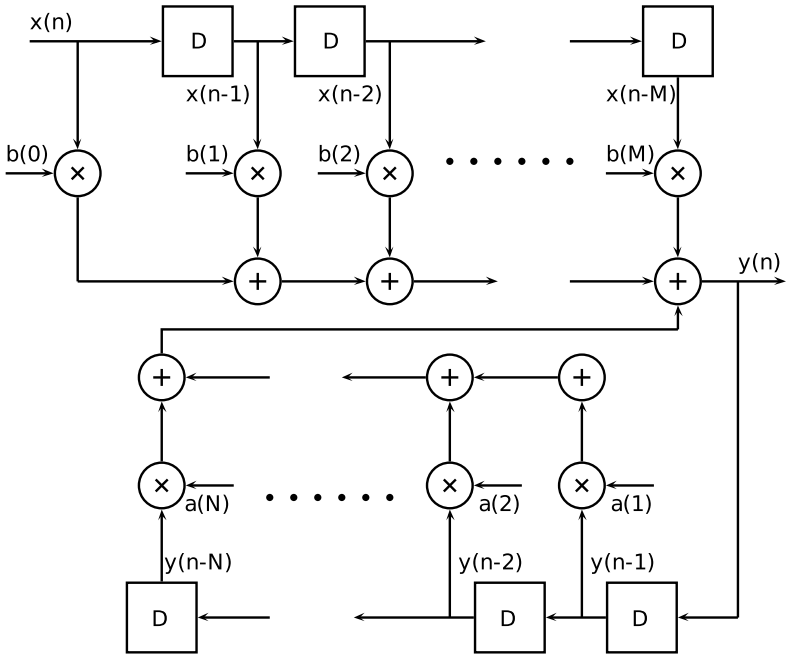
Obr. 1.2: Blokové schéma FIR filtru

FIR filtr je nerekurzivní tzn. výsledný vzorek bude mít hodnotu závislou pouze na hodnotách vstupních vzorků. Tato vlastnost umožňuje realizaci FIR filtru přímo pomocí konvoluce (vzorec 1.1). FIR filtr má všechny póly v počátku, takže je zajištěna stabilita filtru. Naopak nevýhoda FIR filtru je nutnost použití daleko vyšších řádů filtru pro dosažení stejně strmé frekvenční charakteristiky než v případě IIR filtru, což má také vliv na potřebný výpočetní výkon procesoru.

1.1.2 Filtr IIR

Princip funkce filtru IIR lze pozorovat z obrázku 1.3. IIR filtr se principiálně skládá z dvou částí, dopředná část jako v případě FIR filtru a zpětnovazební část. Jedná se

tedy o rekurzivní realizaci filtru, IIR filtr již nejde realizovat přímo pomocí konvoluce. Zásadní nevýhody IIR filtru jsou především vždy nelineární fázová charakteristika a také riziko nestability, neboť je minimálně jeden pól IIR filtru mimo počátek. Naopak hlavní výhoda je pak potřeba nižšího řádu filtru.



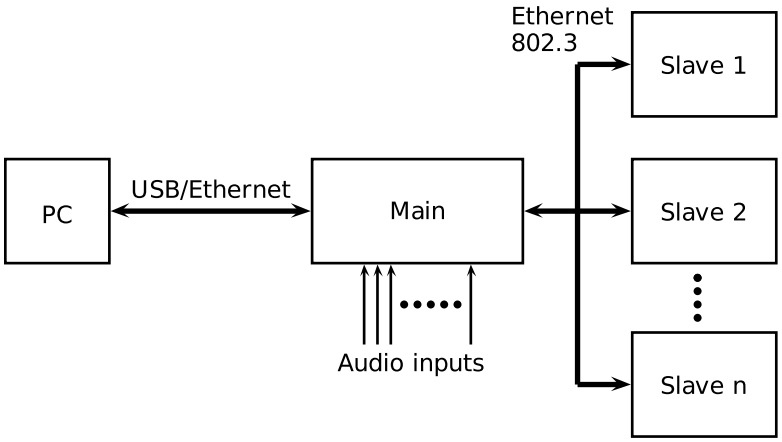
Obr. 1.3: Blokové schéma IIR filtru

Výstupní signál je potom dán vztahem:

$$y(n) = \sum_{i=1}^{N} (a(i) \cdot y(n-i)) + \sum_{j=0}^{M} (b(j) \cdot x(n-j)) \tag{1.2}$$

2 DEFINICE

Bloková struktura audio systému je na obrázku 2.1.



Obr. 2.1: Blokové schéma digitálního audio systému

2.1 Hlavní distribuční jednotka

Hlavní jednotka (na obrázku 2.1 "Main"), jak již bylo psáno výše, má za úkol přijímat audio signály (analogové, digitální), zpracovávat a distribuovat. Musí být schopna komunikovat s uživatelem pomocí dálkového IR ovládání a v neposlední řadě musí být schopná komunikovat s PC kvůli nastavení a monitorování celého systému. Ke všem těmto účelům musí také odpovídat výpočetní výkon použitého mikrokontroléru.

Pro komunikaci s podružnými (slave) jednotkami bylo zvoleno komunikační rozhraní IEEE 802.3 100BASE-Tx [10] z důvodu dostatečně vysoké přenosové rychlosti a možnosti komunikace na vzdálenost až 100m.

Pro komunikaci s osobním počítačem (PC) bylo zvoleno rozhraní USB [8] spolu s rozhraním ethernet. Hlavní důvod pro výběr USB k tomuto účelu je především kvůli tomu, že v dnešní době je USB na každém počítači. Rychlost USB v módu Full speed (FS) je teoreticky 12Mbit/s [8], což je pro potřeby nastavení a monitoringu systému plně dostačující. Tato rychlost by měla být dostačující k přenosu několika datových proudů v reálném čase. V případě módu High speed (HS) je teoretická rychlost USB až 480Mbit/s [9]. Při realizaci tohoto projektu by měla být dostačující rychlost 12Mbit/s. Ethernet byl k ovládání zvolen z důvodu, že celý audio systém je postaven na bázi ethernetu, takže není potřeba dalšího kabelu. Další výhoda je v nepotřebnosti ovladače na PC straně.

Distribuční jednotka musí být také schopna vhodně zdigitalizovat (ne)symetrický analogový vstupní NF signál, alespoň v kvalitě audio CD (44.1KHz, 16 bitů) [6]. Také musí být schopna přijmout již digitální signál S/PDIF [7].

2.2 Podružná jednotka

Hlavním úkolem podružné jednotky ("Slave") je příjem datových streamů přes rozhraní IEEE 802.3, převedení do analogové podoby a výkonové zesílení pro buzení reproduktorů nebo případně pomocí zesilovače pracujího na principu PWM modulace.

Dalším úkolem této jednotky je příjem povelů dálkového IR ovládání, které budou zaslány přes rozhraní ethernet zpět do řídící jednotky, která vykoná žádané změny ve zpracování signálu.

V neposlední řadě musí mít tato jednotka dostatečně výkoný výstup pro řízení motorku, který bude použit pro natáčení reproduktoru. Další požadavek na slave jednotku je měření teplot a univerzální vstupy/výstupy.

3 HARDWAROVÉ ŘEŠENÍ

Pro tento projekt byl vybrán digitální signálový procesor od firmy STMicroelectronics STM32F407VG [11] s jádrem ARM Cortex-M4F [13], který pokrývá všechny výše uvedené požadavky (ethernet RMII, USB, FPU), maximální taktovací frekvence jádra 168MHz by měla poskytnout dostatečný výpočetní výkon pro potřeby tohoto projektu.

Pro základní použití procesoru byl zakoupen vývojový kit STM32F4DISCOVERY [12] s výše uvedeným procesorem, ke kterému bude připojena přídavná deska plošných spojů s ethernetovým PHY driverem KSZ8031RNL [14], který je vybaven rozhraním RMII [14], transformátorem a konektorem RJ45. Schéma je v příloze A na obrázcích A.1 a A.2, deska plošných spojů potom na obrázku A.3 a rozložení součástek na obrázku A.4. Vývojový kit STM32F4DISCOVERY obsahuje také uživatelské USB rozhraní pro přímou komunikaci s procesorem, která bude využita pro komunikaci s ovládacím software. Na této vývojové desce bude možno realizovat virtuální digitální audio systém pro demonstraci ovládacího software.

4 OVLÁDACÍ SOFTWARE

Ovládací software je realizován v objektově orientovaném programovacím jazyce *C++* s grafickým frameworkem, který by měl být multiplatformní a hlavně volně dostupný i s vývojovými nástroji.

4.1 Požadavky

- Komunikace přes USB
- Komunikace přes rozhraní ethernet
- Zobrazení časového průběhu audiosignálu
- Výpočet spektra přijatého audio signálu
- Přehledné grafické rozhraní

4.2 Srovnání frameworků

Výše uvedeným hlavním požadavkům vyhovuje prakticky pouze trojce použitelných frameworků:

- WxWidgets [15]
- Qt [16]
- GTK+ [17]

4.2.1 WxWidgets

WxWidgets je opensource GUI framework podporující překlad pro různé platformy (Windows, Linux, MacOS). Jako komplexní vývojový nástroj lze použít volně dostupné prostředí Code::Blocks s integrovanou aplikací wxSmith na grafické vytváření formulářů.

4.2.2 Qt

Qt je také opensource GUI framework pro různé platformy. Vhodné vývojové prostředí Qt-Creator dodávané přímo od firmy Digia (dříve vyvíjeno firmou Nokia) je také volně dostupné. Tohle vývojové prostředí oproti Code::Blocks také umí kruciální funkci code completion ("doplňování kódu"), což do obrovské míry zrychluje vývoj

aplikace, dostupná dokumentace k frameworku Qt je na vyšší úrovni než k wxWidgets (projevuje se zde také fakt, že wxWidgets je vyvíjeno pouze hrstkou nadšenců a Qt je vyvíjeno firmou Digia). Z tohoto hlavního důvodu (rychlost vývoje aplikace) byl pro použití v tomto projektu vybrán framework Qt.

4.2.3 GTK+

GTK+ je také opensource multiplatformní GUI framework. Bohužel pro vývoj aplikací v C/C++ pro GTK+ neexistuje žádné komplexní vývojové prostředí jako ve výše uvedených frameworcích. Lze použít např. framebuilder Glade v kombinaci s textovým editorem např. Vim a vhodným kompilátorem. Vhodné komplexní prostředí pro vývoj GTK aplikací, bohužel pouze v jazyce Pascal, je Lazarus.

4.3 USB

Jak již bylo napsáno výše pro komunikaci s hardwarem bylo vybráno rozhraní USB, jelikož vyvíjené zařízení se bude na USB chovat jako proprietární zařízení, je potřeba napsat vlastní ovladače pro operační systém. Tento problém řeší volně dostupná opět multiplatformní knihovna pro práci s USB v uživatelském prostoru, což znamená, že není potřeba psát ovladač na úrovni jádra operačního systému. Tato knihovna je známá pod projektem *libusb* [18]. Knihovna libusb je také s výhodou portována přímo do frameworku Qt jako komponenta.

Knihovna je schopna pracovat se všemi třemi typy endpointů (přerušení, bulk, isochronous) a poskytuje synchronní nebo asynchronní metody pro práci s USB. Experimentálně se podařilo dosáhnout reálné přenosové rychlosti procesorem STM32F407 ve FS módu USB cca 6Mbit/s, což je pro potřeby tohoto projektu postačující přenosová rychlost.

Jelikož MCU STM32F407 podporuje až 3 endpointy + 1 kontrolní endpoint, komunikace bude rozdělena na 2 endpointy:

1. Nastavení systému a monitoring v módu přerušení, nízký datový přenos (cca 1-10kbit/s)

2. Přenos datových streamů do PC v módu isochronous, vysoký datový přenos (cca 3-4Mbit/s)

4.4 Ethernet

4.4.1 Nízkoúrovňové rozhraní pro konfiguraci

Pro komunikaci s master jednotkou a zároveň pro komunikaci mezi master jednotkou a slave jednotkami bylo vymyšleno ethernetové rozhraní na nízké úrovni v jazyce C jako struktury s vlastnostmi jednotlivých jednotek a jejich patřičných vstupů/výstupů. Soubor btypes.h, jehož výpis je uveden v příloze B, je použit ve stejné formě v ovládacím softwaru tak ve firmwaru DSP.

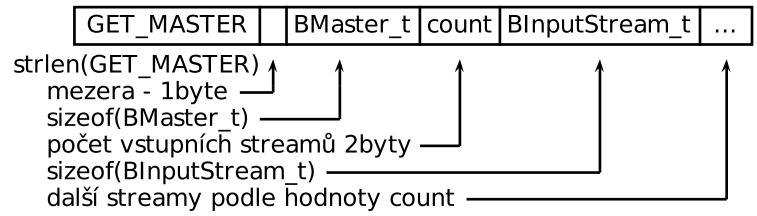
Hlavičkový soubor je rozdělen do několika hlavních částí:

- Výběr datových typů podle použité platformy (PC/ARM) kvůli stejné délce datových typů na obou platformách,
- výběr komunikačních portů,
- řetězce pro zadávání příkazů,
- tabulky různých řetězců (pásma ekvalizéru, názvy výstupů atd.),
- počet koeficientů číslicového ekvalizéru a počet měřených teplot,
- struktury pro nastavení vstupního streamu, výstupního streamu, slave jednotky, master jednotky a struktura BMain_t, která zastřešuje všechny jednotky a jejich datové streamy,
- makra pro vyplnění a vyčtení hlavičky UDP audio streamu.

Popis kominukace pro příkazy GET

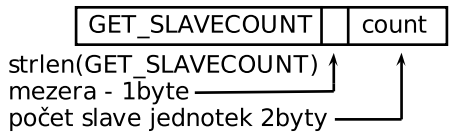
Příkazy `GET_MASTER, GET_SLAVECOUNT a GET_SLAVE` jsou posílány řídícím softwarem pouze po připojení k hlavní jednotce pro zjištění všech informací o celém audio systému, tyto příkazy fungují na principu požadavek-odpověď. Data jsou posílána přímo z paměti bez žádných konverzí z důvodu rychlosti, díky tomuto by mohl nastat problém při různých endianitách použitých platforem, v tomto případě jsou však obě platformy little-endian.

- Řídící software zašle na IP adresu master jednotky a předem stanovený port příkaz GET_MASTER a master jednotka odpoví:



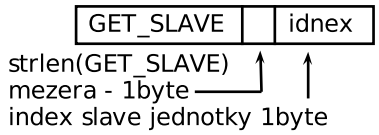
tedy vlastnosti master jednotky a všechny její vstupní audio streamy.

- Na příkaz GET_SLAVECOUNT jednotka master odpoví:

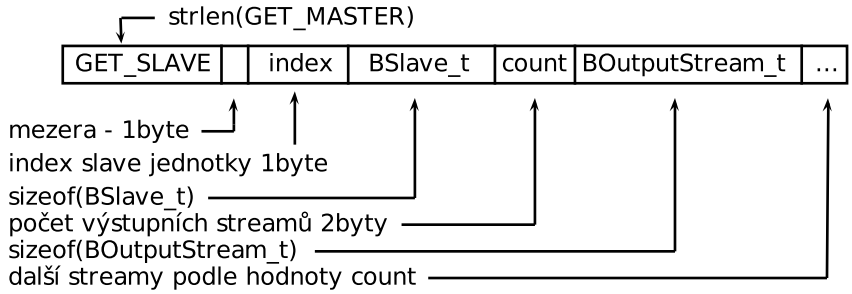


  počet připojených slave jednotek.

- Příkaz GET_SLAVE funguje následovně:
  ovládací software pošle:



  master odpoví:

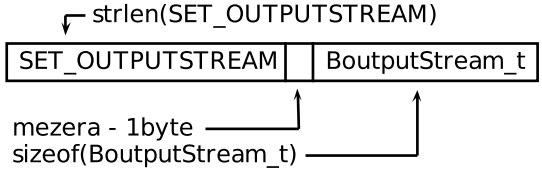


  Odpověď je tedy souhrn vlastností konkrétní slave jednotky a všechny její výstupní audio streamy.

Popis příkazů SET

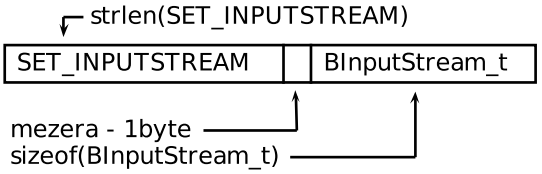
Další sada příkazů `SET_OUTPUTSTREAM`, `SET_INPUTSTREAM`, `SET_TEMPERATURE`, `SET_GPIO`, `SET_SLAVE_EQUALISER`, `SET_SLAVENAME`, `SET_SLAVEINPUT` je použita pro nastavování vlastností master jednotky a slave jednotek. Tyto příkazy jsou pouze jednosměrné ve směru z ovládacího software do master jednotky. Doručení změn do slave jednotek je poté úkol master jednotky.

- Příkaz SET_OUTPUTSTREAM má za úkol nastavit vlastnosti výstupního streamu v určité slave jednotce podle sériového čísla (které musí být v systému unikátní) jednotky a identifikátoru výstupního streamu (což je vše uloženo ve struktuře BoutputStream_t):

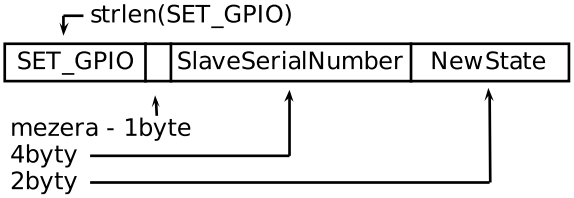


- Příkaz SET_INPUTSTREAM má za úkol nastavit vlastnosti vstupního streamu v master jednotce (která je v systému pouze jedna) a podle identifikátoru

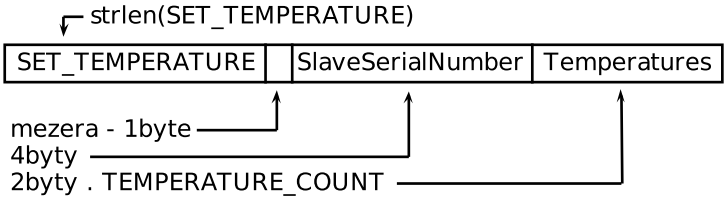
vstupního streamu (opět uloženo přímo ve struktuře BInputStream_t):



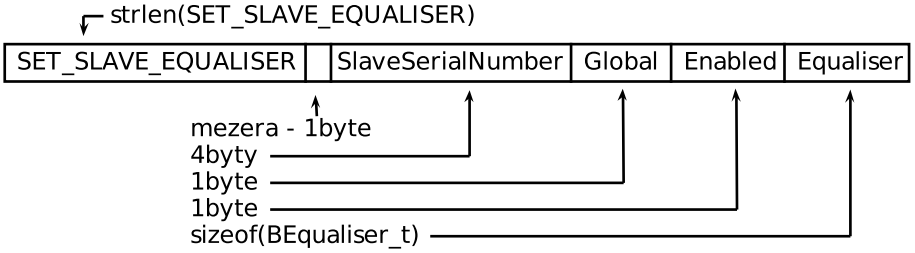
- Příkaz SET_GPIO jak již název napovídá má za úkol nastavit stavy univerzálních digitálních výstupů slave jednotky opět podle sériového čísla jednotky:



- Příkaz SET_TEMPERATURE není v ovládacím software implementován, je použit pouze pro nastavení teplot v master jednotce od slave jednotky:

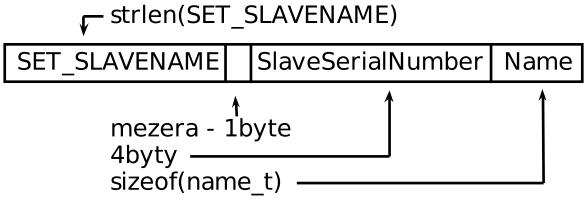


- Příkaz SET_SLAVE_EQUALISER nastavuje digitální globální ekvalizér, který je společný pro všechny výstupní streamy slave jednotky:

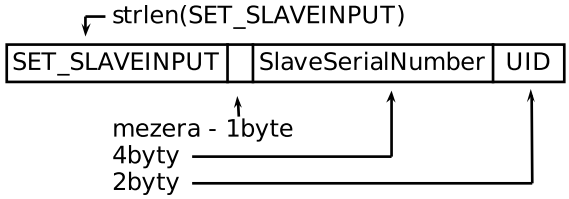


Data Global určují zda použít tento globální ekvalizér pro všechny výstupní streamy nebo používat nastavení ekvalizérů každého výstupního streamu. Data Enabled určují jestli úplně vypnout všechny ekvalizéry slave jednotky.

- Příkaz SET_SLAVENAME nastavuje uživatelský název slave jednotky:



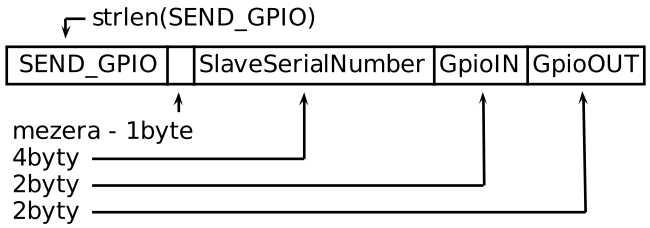
- Poslední příkaz SET_SLAVEINPUT nastavuje identifikátor vstupního připojeného zařízení ke slave jednotce:



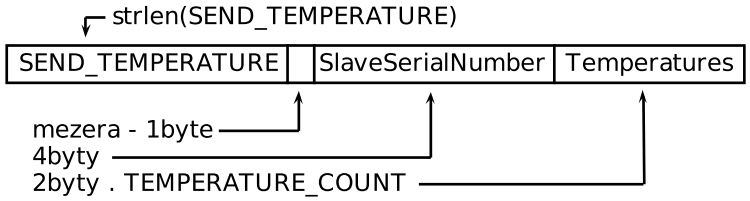
Popis příkazů SEND

Další sada příkazů `SEND_GPIO, SEND_TEMPERATURE, SEND_OUTPUTSTREAM, SEND_INFRACOMMAND, SEND_SLAVE_EQUALISER, SEND_SLAVEREFRESH` slouží opět k jednosměrné komunikaci ovšem ve směru z master jednotky do ovládacího software v PC.

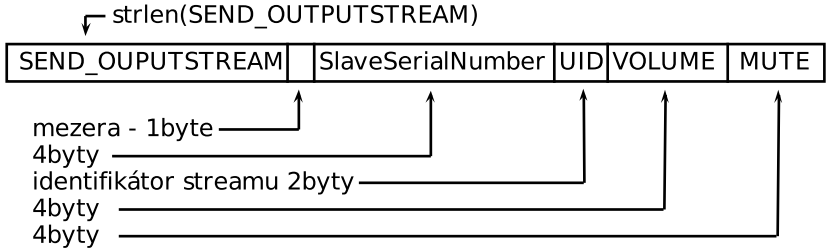
- Příkaz SEND_GPIO posílá do ovládacího softwaru stav změněných digitálních vstupů/výstupů:



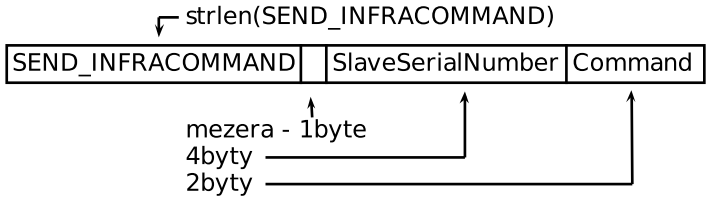
- Příkaz SEND_TEMPERATURE posílá do ovládacího software nové teploty, které přijme ze slave jednotky:



- Příkaz SEND_OUTPUTSTREAM pošle do ovládacího software nové nastavení výstupního streamu (hlasitost a mute), které bude možno měnit přímo na jednotce:

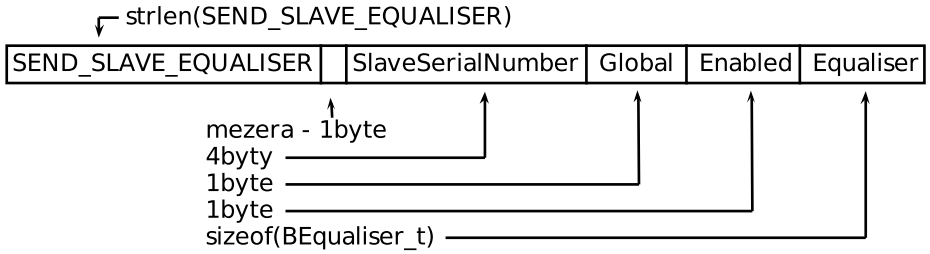


- Příkaz SEND_INFRACOMMAND informuje ovládací software o přijatém povelu z dálkového ovládání:

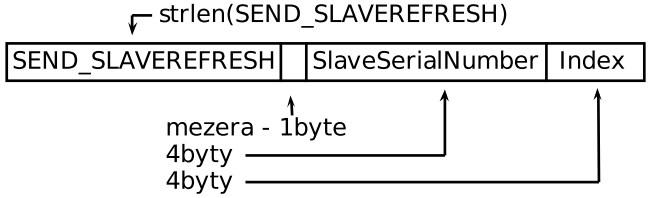


  V případě, že sériové číslo slave jednotky bude nula tak software předpokládá, že povel z dálkového ovladače přijala přímo master jednotka.

- Příkaz SEND_SLAVE_EQUALISER posílá nové nastavení ekvalizéru do ovládacího SW:



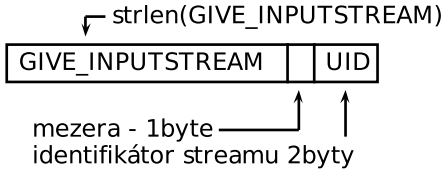
- SEND_SLAVEREFRESH upozorňuje ovládací SW, aby si vyžádal znovu informace o slave jednotce:



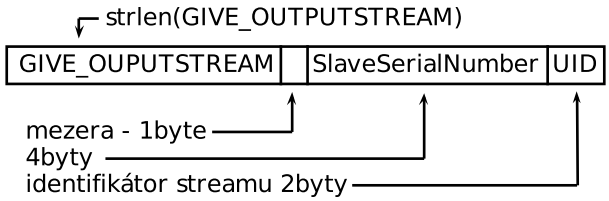
Popis příkazů GIVE

Poslední sada příkazů `GIVE_STOPOUTPUTSTREAM, GIVE_STOPINPUTSTREAM, GIVE_OUTPUTSTREAM, GIVE_INPUTSTREAM` slouží k vyžádání/zrušení zasílání audio streamů od master jednotky do ovládacího software pro zobrazování časového průběhu atd.

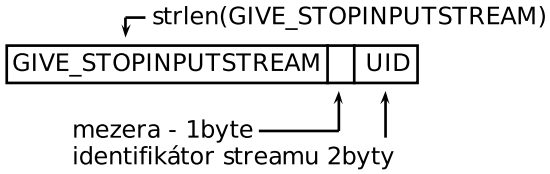
- Příkaz GIVE_INPUTSTREAM vyžádá vstupní stream podle jeho UID:



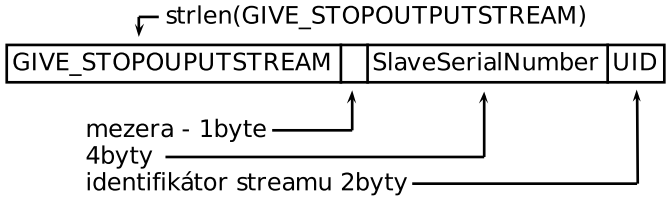
- Příkaz GIVE_OUTPUT vyžádá výstupní stream podle jeho UID a sériového čísla slave jednotky:



- Příkaz GIVE_STOPINPUTSTREAM zruší zasílání vstupního streamu podle jeho UID:



- Příkaz GIVE_STOPOUTPUTSTREAM opět zruší zasílání výstupního streamu podle sériového čísla slave jednotky a UID streamu:



4.4.2 Struktura paketu audio streamu

Audio streamy jsou posílány protokolem UDP, což je nespolehlivá služba, která nezaručuje doručení, což by nemělo v případě občasného přijetí paketu, ve kterém bude několik chybných vzorků, vadit. Naopak výhoda je v rychlosti přenosu oproti protokolu TCP. Pakety jsou posílány na jednom portu, který je definován v souboru *btypes.h* (příloha B) na konkrétní IP adresy slave jednotek, případně pomocí broadcastu pokud budou zároveň vyžádány ovládacím softwarem.

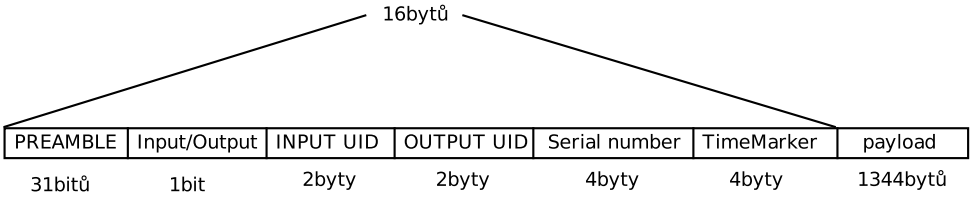
Struktura paketu je vidět na obrázku 4.1, ze kterého je vidět, že paket je složen z dvojce hlavní částí:

- hlavička 16 bytů
- data 1344 bytů

Hlavička je složena z preambule pro jasné rozeznání typu paketu (pokud bude přijatý paket mít jinou preambuli bude zahozen). Dále je v hlavičce vyhrazen jeden bit pro rozeznání zda je stream vstupní nebo výstupní, v běžném provozu jsou posílány pouze pakety výstupních streamů, vstupní streamy mohou být posílány pouze

na vyžádání ovládacího software. Další dvojce bytů určuje identifikátor vstupního streamu, který je nutný také pouze na straně ovládacího SW pro jasnou identifikaci vstupního streamu. Dále je umístěn identifikátor výstupního streamu a sériové číslo slave jednotky. Díky těmto informacím je slave jednotka schopna poznat zda je paket určen pro ni nebo má být zahozen. Identifikátor výstupního streamu je určen pro směrování audio dat do správného reproduktoru. Poslední čtyři byty v hlavičce jsou určeny pro časovou značku (přímo systémový čas mikrokontroléru), která je určena pro časovou synchronizaci.

Zbytek paketu je vyplněn vlastními daty, 336 audio vzorků jako 4-bytová čísla s plovoucí desetinnou čárkou (float IEEE 754) v rozsahu od -1 do +1. Dlužno dodat, že ve data jsou uspořádána v pořadí little-endian.



Obr. 4.1: Struktura paketu audio streamu

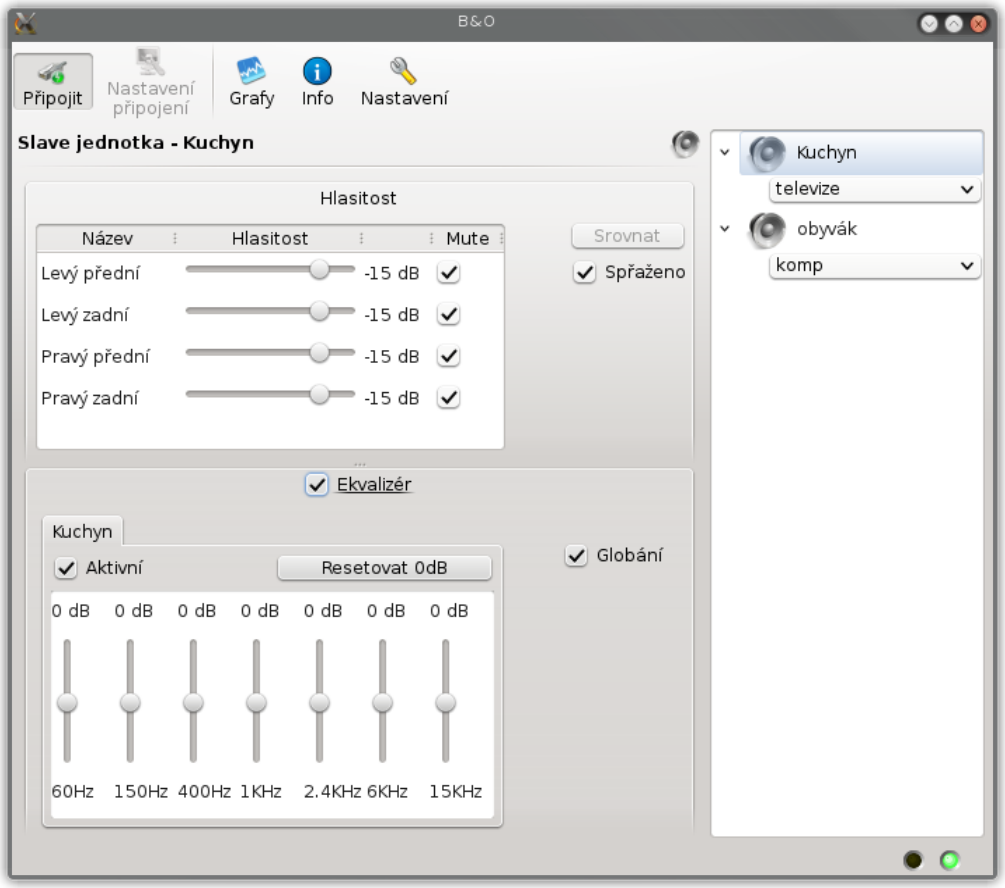
5 NÁVRH GUI

Tato část práce se zabývá převážně uživatelskými možnostmi aplikace. Ovládací software je sestaven z hlavního okna, kde je možno uživatelsky nastavovat základní parametry systému, hlavní okno je na obrázku 5.1 a z oken pro komplexní nastavení systému (nastavení citlivostí, seskupení vstupů do zařízení atd.).

5.1 Hlavní formulář

Hlavní formulář je na obrázku 5.1 a je rozdělen do několika sekcí:

- Výběr slave jednotky
- Nastavení hlasitosti
- Konfigurace ekvalizéru



Obr. 5.1: Hlavní formulář aplikace

Slave jednotky

V pravé části formuláře uživatel volí slave jednotku, kterou chce konfigurovat, podle vybrané jednotky je automaticky přepínána celá levá část formuláře. Dále má uživatel možnost přehledně volit zdroje vstupního signálu pro všechny slave jednotky v celém systému.

Hlasitost

Zde uživatel může nastavovat hlasitosti jednotlivých kanálů slave jednotky, případně vstup úplně zatlumit pomocí funkce Mute. Tlačítko Srovnat má za úkol nastavit všechny hlasitosti na stejnou úroveň, *QCheckBox* Spřaženo zajišťuje, že všechny hlasitosti jsou měněny stejně.

Ekvalizér

Uživatel má k dispozici sedmi-pásmový ekvalizér, kde má různé možnosti konfigurace:

- Ekvalizér úplně vypnutý,
- společné nastavení pro všechny výstupní streamy slave jednotky,
- nastavení pro každý stream slave jednotky zvlášť.

Také existuje tlačítko pro resetování nastavení všech pásem ekvalizéru na hodnotu 0dB.

Nástrojový pruh

V nástrojovém pruhu jsou tlačítka s těmito funkcemi:

- **Připojit** - připojí program k master jednotce audio systému, po úspěšném připojení je rozsvícena zelená led dioda ve stavovém pruhu úplně napravo, žlutá dioda vlevo slouží k indikaci probíhající komunikace.
- **Nastavení připojení** - tlačítko vyvolá dialog pro konfiguraci připojení k hardware, více popsáno v 5.1.1.
- **Grafy** - tlačítko vyvolá okno pro práci s časovými průběhy (viz 5.2).
- **Info** - tlačítko vyvolá okno pro zobrazení obecných informací audio systému (viz 5.3).
- **Nastavení** - tlačítko vyvolá dialog pro komplexní nastavení audio systému, více popsáno v 5.4.

5.1.1 Dialog nastavení připojení

Dialog slouží pro nastavení připojení k audio systému a je zobrazen na obrázku 5.2. Uživatel zde má možnost vybrat typ připojení (USB/ethernet) a případně zadat IP adresu master jednotky. V neposlední řadě uživatel může zvolit zda-li se software má automaticky pokusit připojit k audio systému po spuštění. (Tyto informace jsou uchovávány v souboru *bang.ini*, který je vytvořen ve stejném adresáři jako spustitelný soubor).

Obr. 5.2: Dialog nastavení připojení k jednotce master

5.2 Dialog s časovými průběhy

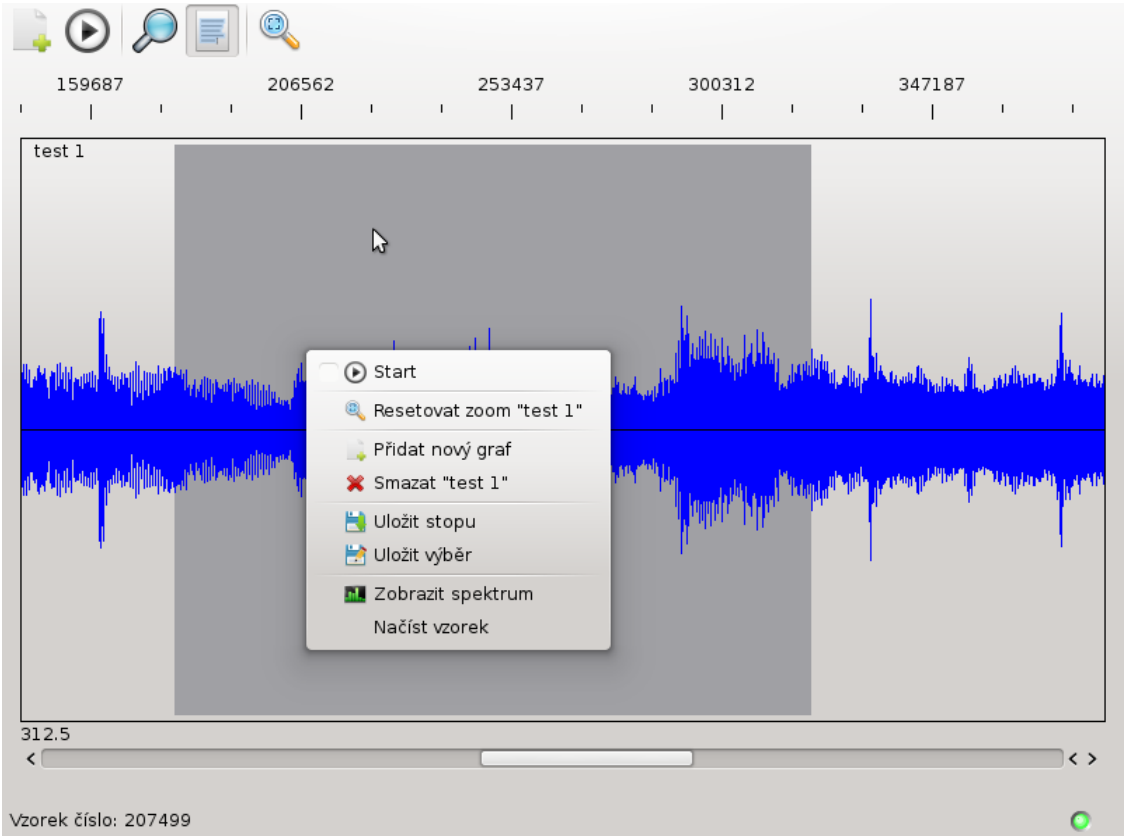
Tento dialog umožňuje uživateli zaznamenat časový průběh libovolného vstupního či výstupního streamu. Formulář s načteným průběhem je na obrázku 5.3.

Možnost pohybu v grafu je pomocí posuvníku v dolní části okna nebo pomocí kolečka myši. Horizontální zvětšování je potom možno stisknutím klávesy Ctrl a kolečkem myši. Tyto vlastnosti byly odpozorovány z nahrávacího programu Audacity [19].

Dialog má horní nástrojový pruh, který má tyto akce:

- **Nový časový průběh** - vyvolá okno s výběrem streamu (viz 5.2.1). Je možno mít více časových průběhů současně, ty jsou pak automaticky skládány pod sebe a jejich posun a zvětšování v ose x jsou synchronizovány.
- **Start** - vyšle master jednotce požadavek o zasílání/zastavení streamů, které jsou zobrazeny.
- **Režim vertikálního zvětšování** - v tomto režimu je změněn kurzor myši na lupu a kliknutím na časový průběh dojde k jeho vertikálnímu zvětšování, případně při držení klávesy Ctrl a klikání naopak zmenšování

- **Režim výběru** - v tomto režimu je kurzor změněn na standardní kurzor výběru textu. Kliknutím a tažením lze vybrat úsek časového průběhu
- **Reset vertikálního zoomu** - tlačítko nastaví na všech časových průbězích vertikální zvětšení na původní hodnotu 1.



Obr. 5.3: Dialog s načteným časovým průběhem

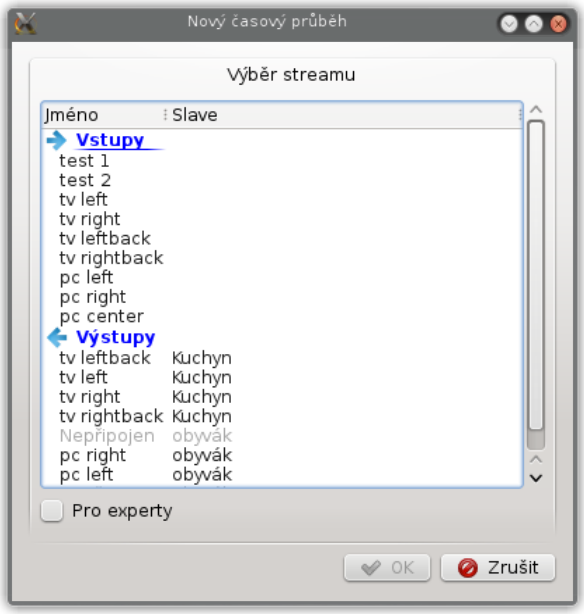
Po kliknutí pravým tlačítkem na libovolný časový průběh je zobrazeno kontextové menu, které je také vidět na obrázku 5.3 a obsahuje tyto funkce:

- **Start** - stejná funkce jako v nástrojovém pruhu, vyšle požadavek na odesílání/zastavení streamů.
- **Resetovat zoom** - nastaví vertikální zvětšení na původní hodnotu 1, ovšem pouze u konkrétního časového průběhu.
- **Přidat nový graf** - opět stejná funkce jako z nástrojového pruhu.
- **Smazat** - vymaže časový průběh, u kterého bylo vytvořeno kontextové menu.
- **Uložit stopu** - vyvolá standardní dialog pro uložení souboru. Časový průběh je případně uložen jako wav soubor.

- **Uložit výběr** - principiálně stejná funkce jako předchozí, ale bude uložen pouze výběr vzorků.
- **Zobrazit spektrum** - zobrazí dialog pro práci se spektrem, kterému předá rozsah vybraných vzorků (viz 5.2.2).
- **Načíst vzorek** - funkce je pouze pro testování a má za úkol načíst do časového průběhu vzorky ze souboru.

5.2.1 Dialog s nastavením časového průběhu

Tento dialog je zobrazen při akci "Nový časový průběh" nadřazeného dialogu. Uživatel může zvolit, který stream má být zaznamenáván v novém časovém průběhu. Volba pro experty slouží k zobrazení identifikátorů streamů a byla hojně používána při testování. Dialog je zobrazen na obrázku 5.4.

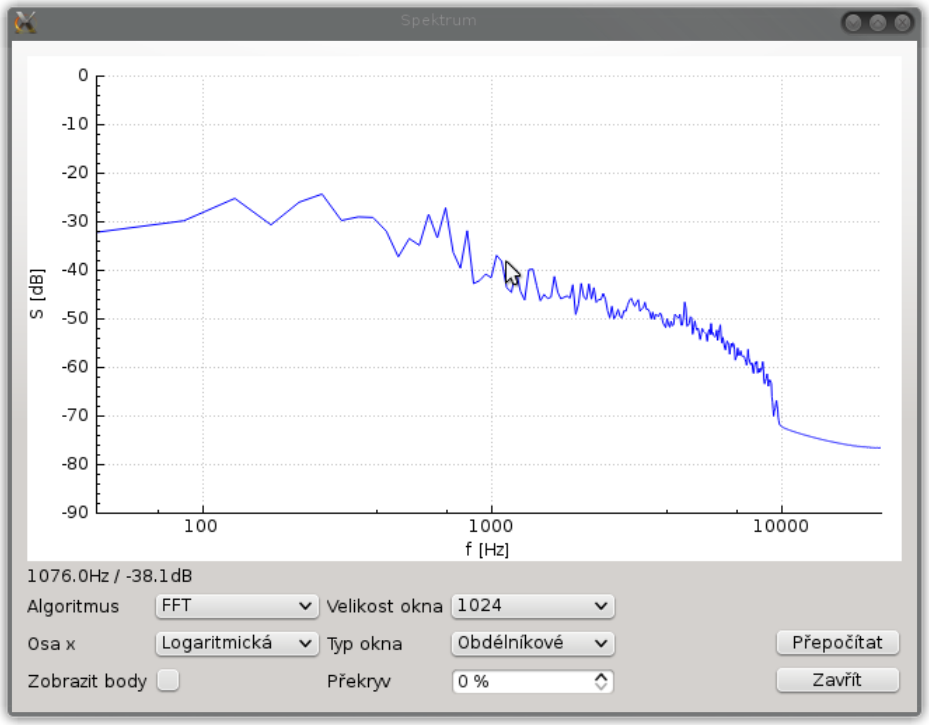


Obr. 5.4: Dialog pro výběr streamu nového časového průběhu

5.2.2 Dialog pro práci se spektrem

Dialog pro práci se spektrem je vidět na obrázku 5.5. Uživatel lze může volit parametry transformace - velikost okna, typ okna, případně překryv. Dále lze nastavit parametry zobrazení - frekvenční osa lineární/logaritmická, případně zobrazit přímo vypočtené body jako křížky. Tlačítko přepočítat slouží k novému načtení vzorků z nadřazeného dialogu s časovými průběhy.

Při pohybu kurzoru v oblasti spektra je pod grafem zobrazována frekvence z rastru frekvenčního kroku a velikost frekvenční složky v dB. V případě zvolení většího okna než je počet vzorků je zde zobrazeno hlášení, že výpočet nelze provést.



Obr. 5.5: Ukázka dialogu pro práci se spektrem

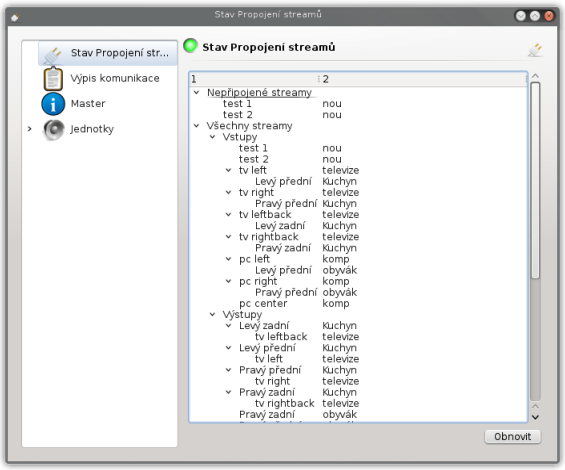
5.3 Dialog obecné informace

Tento dialog je určen pro zobrazení obecných informací o celém audio systému a je složen z několika přepínatelných widgetů pomocí stromu vlevo:

- Stav propojení streamů
- Výpis komunikace
- Obecné informace o master jednotce a slave jednotkách (výpis příkazů dálkového ovládání, záznam měřených teplot, sledování univerzálních vstupů/výstupů atd.)

5.3.1 Stav propojení streamů

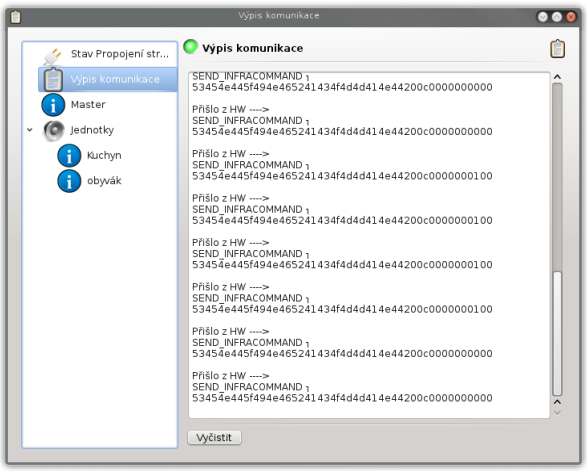
Tento widget pouze zobrazuje aktuální propojení vstupních a výstupních streamů podle jejich UID identifikátorů. Widget je zobrazen na obrázku 5.6. Jediný ovládací prvek je tlačítko pro znovu načtení obsahu widgetu.



Obr. 5.6: Widget "Stav propojení streamů"

5.3.2 Výpis komunikace

Tento widget zachycuje a zobrazuje veškerou komunikaci mezi master jednotkou a ovládací aplikací. Widget je zobrazen na obrázku 5.7.



Obr. 5.7: Widget "Výpis komunikace"

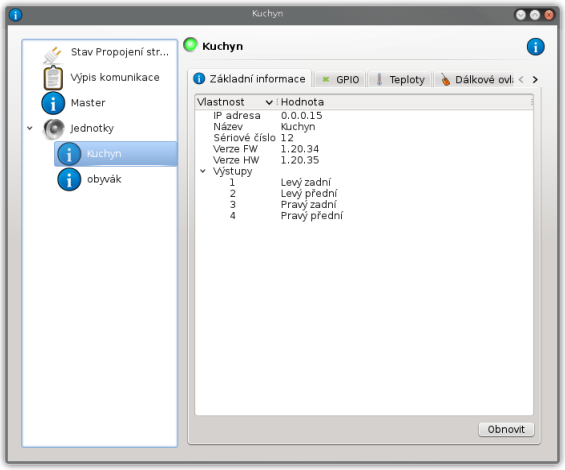
5.3.3 Výpis základních informací

Tento widget je rozdělen do více karet:

- **Základní informace** - sériová čísla, verze HW atd.
- **GPIO** - sledování a ovládání univerzálních vstupů/výstupů
- **Teploty** - záznam měřených teplot
- **Dálkové** ovládání - záznam přijatých příkazů od dálkového ovládání

Základní informace

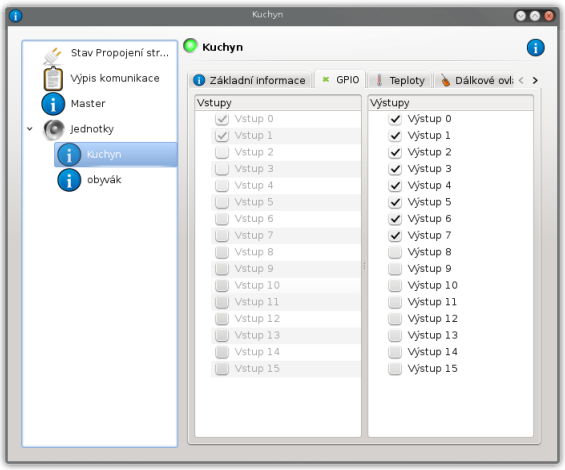
Widget je zobrazen na obrázku 5.8, jsou zde vidět základní informace o slave jednotce např IP adresa, sériové číslo atd.



Obr. 5.8: Widget "Základní informace"

GPIO

Widget je zobrazen na obrázku 5.9, zde lze sledovat stavy 16 vstupních a výstupních pinů, výstupní piny lze také modifikovat.

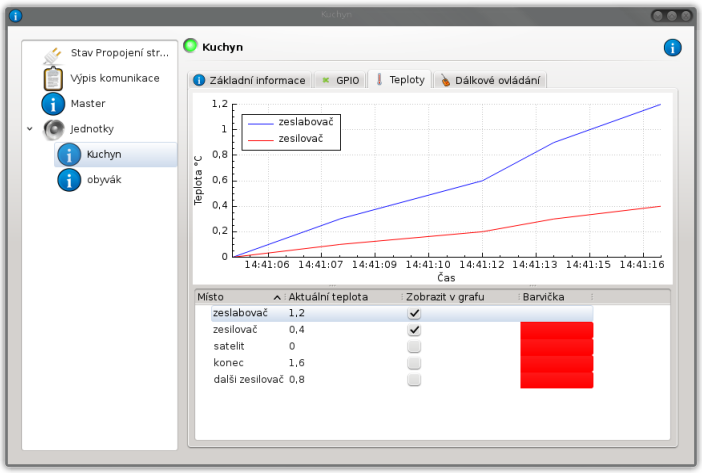


Obr. 5.9: Widget "GPIO"

Teploty

Widget je zobrazen na obrázku 5.10. Widget zobrazuje aktuální teploty, které jsou měřeny přímo systémem a odesílány ovládacímu SW. Existuje zde také možnost

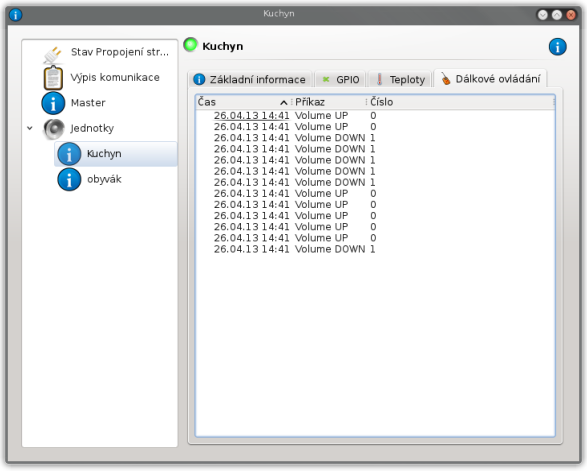
vybrané teploty zobrazit jako časovou závislost v grafu. U každé teploty lze zvolit jinou barvu závislosti dvojklikem na barevně vyplněné pole v tabulce níže. Ten vyvolá standardní dialog pro výběr barvy.



Obr. 5.10: Widget "Teploty"

Dálkové ovládání

Widget je zobrazen na obrázku 5.11. Zde je časový záznam všech přijatých povelů dálkového ovládání.



Obr. 5.11: Widget "Dálkové ovládání"

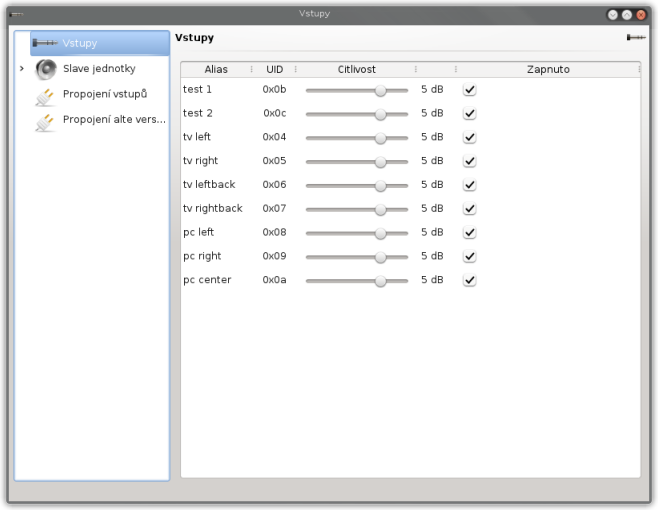
5.4 Formulář nastavení systému

Formulář je určen pro komplexní nastavení celého reproduktorového systému a opět je složen z několika dílčích widgetů:

- **Vstupy** - zde lze nastavit základní vlastnosti vstupních streamů (názvy atd.).
- **Slave jednotky** - poskytuje seznam slave jednotek a nastavení jejich vlastností.
- **Nastavení slave jednotky** - Stejné jako výše, pouze lze nastavovat jen jednu konkrétní slave jednotku.
- **Propojení vstupů** - zajišťuje seskupení vstupních streamů do vstupních zařízení.
- **Propojení vstupů alternativní verze** - také zajišťuje seskupení vstupních streamů do vstupních zařízení, pouze jiný vzhled.

5.4.1 Vstupy

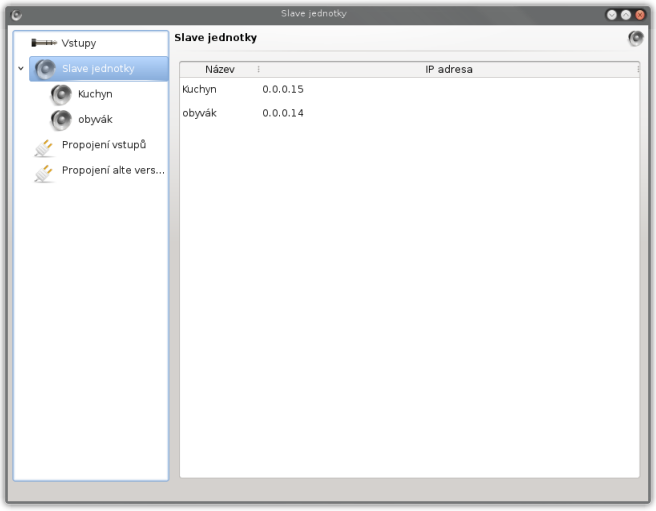
Widget Vstupy je na obrázku 5.12. Zde má uživatel možnost pojmenovat jednotlivé vstupy hlavní jednotky pro svoji přehlednost, takto pojmenované vstupy pak jsou vidět v celé aplikaci. Dále může nastavit citlivost jednotlivých vstupů, tato vlastnost je výhodná v případě, že různý zdroj má jinou sílu signálu a při přepnutí celého zdroje signálu (audio výstup TV, CD přehrávač, gramofon) by musel uživatel ještě měnit hlasitost. Uživatel může také aktivovat jenom ty vstupy které jsou opravdu používány.



Obr. 5.12: Widget "Vstupy"

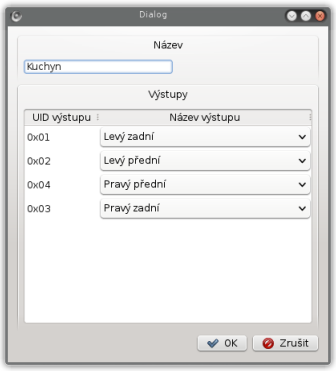
5.4.2 Slave jednotky

Karta Slave jednotky je na obrázku 5.13, zde je zobrazen seznam všech slave jednotek připojených do sítě. Uživatel si opět pro přehlednost může pojmenovat jednotky např. podle umístění. Tyto názvy jsou opět zobrazeny v celé aplikaci místo IP adres nebo sériových čísel.



Obr. 5.13: Widget "Slave jednotky"

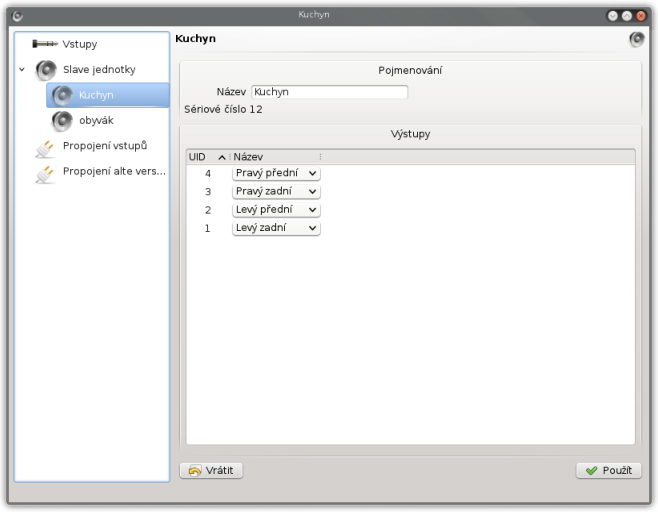
Při dvojtém kliknutí na konkrétní jednotku v tabulce je zobrazen dialog s nastavením slave jednotky, tento dialog je na obrázku 5.14. Zde má uživatel možnost nastavit výše zmiňovaný název a dále pro zvýšení univerzálnosti, existuje možnost pojmenovat jednotlivé audio výstupy (pravý přední, levý zadní, center atd.)



Obr. 5.14: Dialog nastavení slave jednotky

5.4.3 Nastavení slave jednotky

Tento widget má stejnou funkci jako předchozí, pouze jsou přímo vygenerovány dílčí nastavení pro všechny slave konkrétní slave jednotky. Widget je na obrázku 5.15.



Obr. 5.15: Widget "Nastavení slave jednotky"

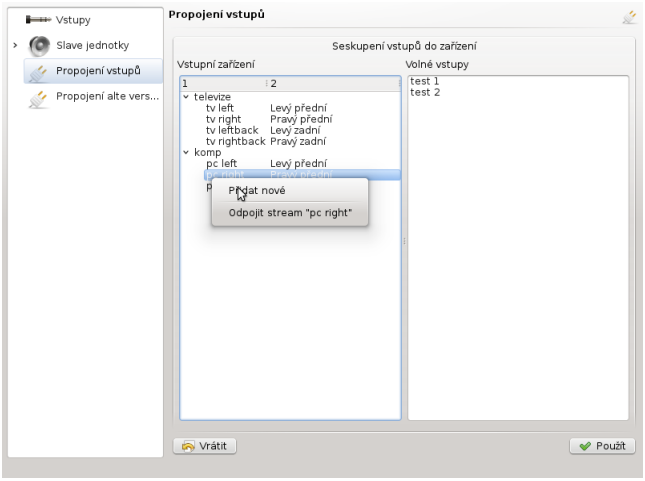
5.4.4 Propojení vstupů

Widget spojení vstupů je na obrázku 5.16. V tomto widgetu je v pravé části seznam všech volných vstupních streamů, které ještě nejsou přiřazeny žádnému vstupnímu zařízení. V pravé části jsou již pak zobrazeny vstupní zařízení ve formě stromu a jejich položky jsou připojené vstupní streamy. Poklepání na přiřazení vstupního streamu je vyvolán *QComboBox*. Pro výběr správného přiřazení streamů. Pravým kliknutím tlačítka myši je vyvoláno kontextové menu s volbami (závisí na místě kliknutí):

- **Přidat nové** - je zobrazen dialog pro zadání jména nového zařízení.
- **Odpojit stream** - tento stream bude od zařízení odpojen a přesunut do pravé části k volným streamům.
- **Odstranit zařízení** - zařízení bude smazáno a jeho streamy budou automaticky přesunuty do seznamu volných streamů.

Volné streamy jsou připojovány k zařízení pomocí funkce *Drag n Drop*, tzn. v pravé části je nutno stream uchopit a táhnout na zvolené zařízení.

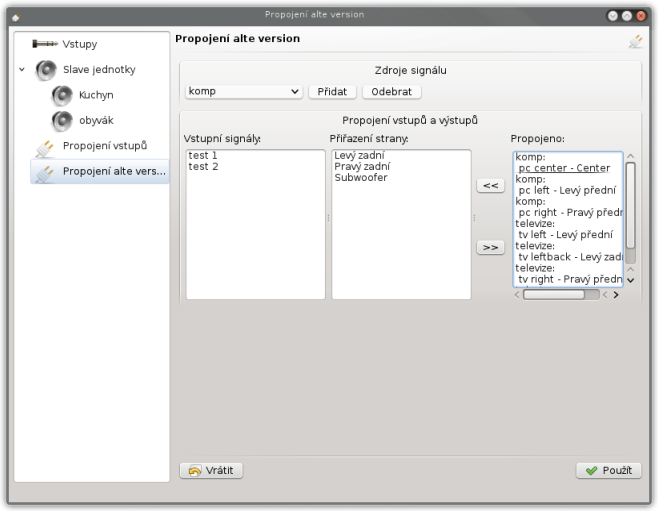
Díky tomuto nastavení pak uživatel může jednoduše přepínat celé vstupy v hlavním formuláři.



Obr. 5.16: Widget "Propojení vstupů"

5.4.5 Propojení vstupů alternativní verze

Widget s alternativním vzhledem je na obrázku 5.17. Tento widget funguje tak, že uživatel nejdříve vybere vstupní zařízení, kterému chce připojit volný vstup, poté vybere přiřazení strany a stiskne tlačítko spojit "»". Případně může naopak spojení zrušit tlačítkem rozpojit "«".



Obr. 5.17: Widget "Propojení vstupů" alternativní verze

6 PROGRAMÁTORSKÝ MODEL

Celý software je koncipován jako hiearchie tříd, které vzájemně spolupracují avšak se snahou o minimální provázanost. Software je primárně složen ze dvou hlavních částí a to třídy s formuláři (GUI) a sadou tříd (v C++ odděleno pomocí namespace BaO) speciálně určených pro souhrn vlastností a rozhraní pro celý digitální reproduktorový systém. Projekt je členěn do takovéto struktury adresářů:

```
Projekt
├── 3rdParty - knihovny třetích stran
│   ├── fftw - knihovna pro výpočty dft [21]
│   └── sndfile - knihovna pro práci se soubory wav [20]
├── bang - sada tříd BaO
├── components - sestava widgetů
│   ├── infoDlg - widgety pro dialog s informacemi
│   ├── mainDlg - widgety pro hlavní okno aplikace
│   ├── settingDlg - widgety pro okno s komplexním nastavením
│   └── treePages - vlastní widget přepínání karet pomocí stromu
├── forms - všechny formuláře a dialogy aplikace
├── interface - pouze soubor btypes.h
└── widgetTrack - sada komponent pro zobrazování audio dat
```

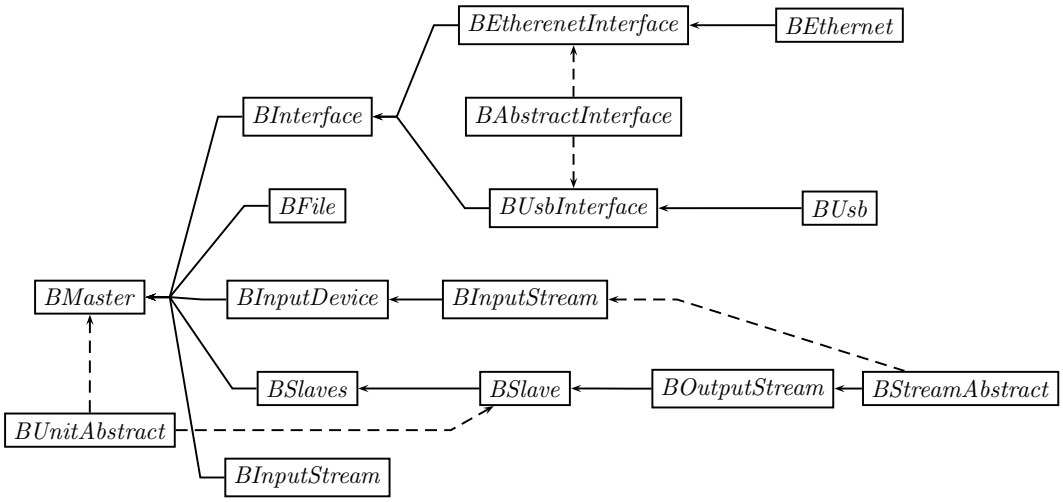
6.1 Sada tříd BaO

Všechny třídy jsou děděny ze základní třídy používané v Qt QObject (tzn. již není možný jednoduchý přechod k jinému frameworku nebo ke standardním knihovnám C++). Tato třída poskytuje základní funkce, které jsou možné v Qt, primárně možnost využití signálů/slotů (událostí), které jsou pak použity pro komunikaci mezi jednotlivými třídami a se samotným uživatelským rozhraním. Graf hierarchie jednotlivých tříd doplněn o pohled dědičností je vidět na obrázku 6.1. (čárkovaně jsou označeny dědičnosti tříd). Na tomto obrázku je vidět, že třída *BMaster* schraňuje všechny objekty celého audio systému.

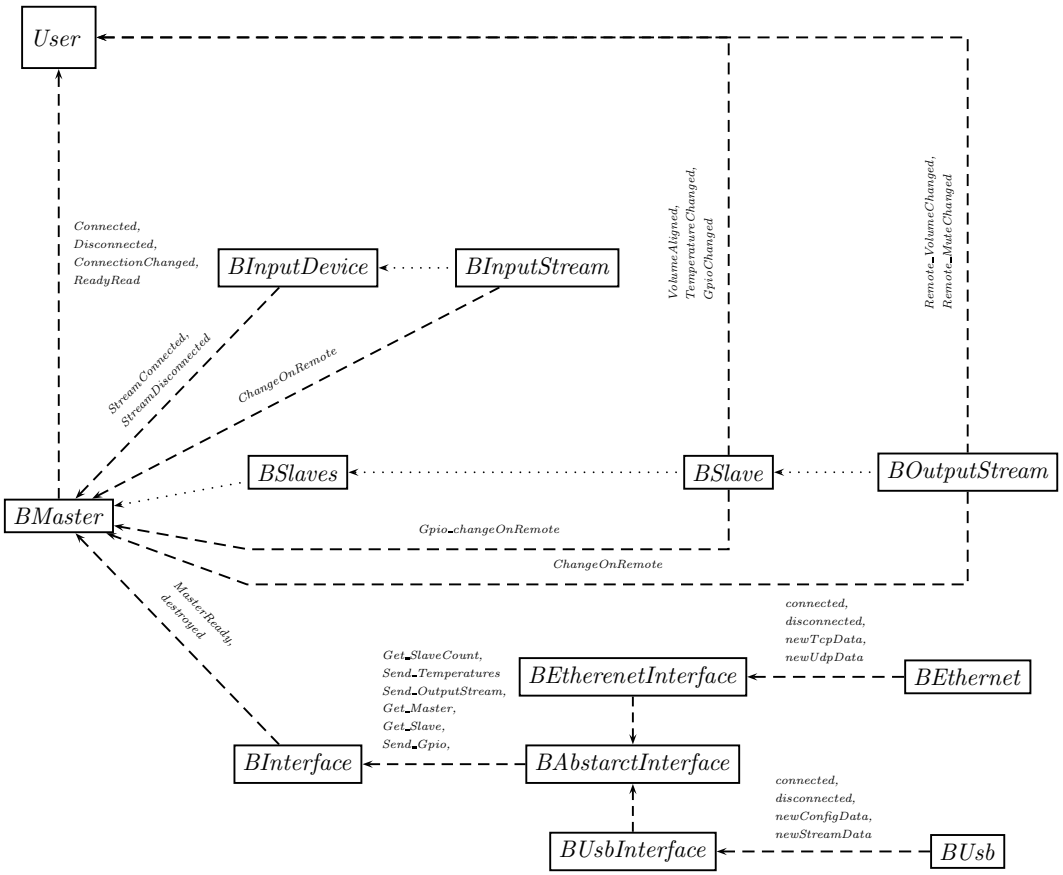
Třídy pro komunikaci s jednotkou master pomocí rozhraní USB nejsou v této fázi projektu ještě implementovány.

Některé třídy přímo odpovídají vlastnostem celých jednotek a jsou to tyto třídy:

- Jednotlivé instance třídy *BInputStream* odpovídají přímo vstupním audio streamům, které jsou fyzicky obsaženy v jednotce master.
- Instance tříd *BOutputStream* mají přímo vlastnosti výstupních streamů, které jsou naopak fyzicky obsaženy v jednotce slave.
- Instance tříd *BSlave* odpovídají přímo jednotkám slave a zahrnují také kontejner všech jejich výstupních streamů *BOutputStream*.



Obr. 6.1: Graf hierarchie tříd BaO doplněn o dědičnosti



Obr. 6.2: Graf spolupráce tříd BaO pomocí signálů/slotů

- Třída *BMaster* odpovídá přímo jednotce master a lze vytvořit pouze jednu instanci této třídy v aplikaci. Instance má všechny vlastnosti jednotky master a obsahuje také kontejner vstupních zařízení *BInputDevice*, který obsahuje kontejner streamů *BInputStream*, které patří k tomuto zařízení. Třída *BMaster* poté ještě obsahuje kontejner vstupních streamů *BInputStream*, které nejsou připojeny k žádnému vstupnímu zařízení *BInputDevice*.

Pro vyjasnění spolupráce jednotlivých tříd pomocí signálů/slotů je uveden obrázek 6.2, kde je jednoznačně vidět, které třídy komunikují mezi sebou pomocí signálů/slotů. Popis těchto signálů a slotů je poté uveden v kapitolách o jednotlivých třídách. Také jsou uvedeny signály, které může uživatel tříd použít pro svoje potřeby.

6.1.1 BStreamAbstract

Třída *BStreamAbstract* je pouze abstraktní třídou, která je poté děděna třídami *BInputStream* a *BOutputStream*, zároveň tato třída dědí z třídy *BMath*, kde jsou pouze jednoduché matematické operace (přepočty dB na absolutní jednotky atd.). Třída obsahuje privátní proměnné:

```
private:
    QString Name;
    int UID;


private:
    virtual void connectStream
        (BStreamAbstract * stream,
        bool emitSignal = true) = 0;
```

Proměnná *Name* určuje uživatelský název datového proudu a *UID* určuje unikátní identifikátor, který je již uživatelsky neměnný. Tento identifikátor musí být přímo uložen v jednotkách master/slave podle počtu vstupních/výstupních streamů.

Mimo přístupových funkcí k těmto proměnným obsahuje třída tzv. pure virtual method v překladu "čistou virtuální metodu", která nesmí mít ve třídě implementaci, ale musí být implementována v odvozených třídách.

6.1.2 BInputStream

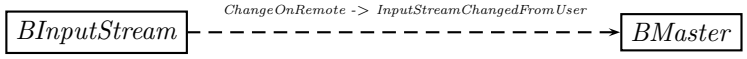
Tato třída obsahuje informace o vstupním audio streamu, jak již bylo napsáno výše tato třída dědí z *BStreamAbstract*. Třída zahrnuje tyto důležité prvky:

```
public:
    void If_SetStream(const BInputStream_t * in);
```

```
    void connectStream(BStreamAbstract *stream,
        bool emitSignal = true);
    void disconnectStream(BOutputStream * stream);

signals:
    void ChangeOnRemote(const BInputStream * stream);

private:
    int Gain;
    float GainAbsolute;
    bool Active;
    QString Side;
    BInputDevice * deviceParent;
    QMap<unsigned int , BOutputStream *> OutputConnection;

public slots:
    void SetGain(int Gain);
    void SetActive(bool active);
```

Proměnná *Gain* určuje zisk vstupního streamu v dB, která je přepočtena pomocí interní metody na absolutní zisk *GainAbsolute*, tato hodnota je posílána do master jednotky. Proměnná *Active* určuje zdali je vstupní stream vůbec aktivován. Proměnná *Side* určuje přiřazení kanálu ke straně (levý, pravý, center atd.). Tyto vlastnosti je možno nastavovat a číst pomocí přístupových metod, které nejsou uvedeny přímo v textu výše. Nakonec proměnná *OutputConnection* je mapa s hodnotami ukazatelů na připojené výstupní streamy a klíčem identifikátor streamu pro rychlé vyhledávání pomocí UID. Mapa byla zvolena z důvodu, že v praxi může nastat případ, kdy bude více vstupních streamů přehráváno na více jednotkách současně.

Signál *ChangeOnRemote* je vyslán, když uživatel změní přes uživatelské rozhraní vlastnosti streamu. Tento signál je po vytvoření streamu připojen na slot *InputStreamChangedFromUser* ve třídě *BMaster*, ve kterém je zajištěno odeslání změny do jednotky master. Komunikace s třídou *BMaster* je znázorněna na obrázku 6.3.



Obr. 6.3: Znázornění komunikace třídy *BInputStream* s okolím

Metoda *If_SetStream* Slouží k nastavení streamu pomocí nízkoúrovňové struktury, která je přijata přes rozhraní od jednotky master, struktura je uvedena v příloze B. Metoda *connectStream* slouží k připojení výstupního streamu. V případě, že

je stream již připojen funkce neudělá nic. Její druhý parametr emitSignal říká, jestli má být vyslán signál *ChangeOnRemote*. Toho je využito pokud dojde ke změně ze strany uživatele, pokud je funkce volána při nastavení podle master jednotky tak signál generován není. Metoda *disconnectStream* odpojí připojený výstupní stream, pokud stream nebyl připojen funkce nic neudělá.

Slot *SetGain* je použit k nastavování proměnné gain, který je v GUI připojen přímo na signály z posuvníků *QSlider*. Úplně stejně funguje slot *SetActive*, který je připojen přímo na signály z *QCheckBox*.

6.1.3 BOutputStream

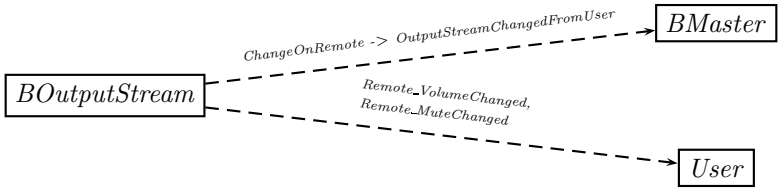
Třída obsahuje informace o výstupním audio streamu, který je přehráván slave jednotkou. Třída *BOutputStream* také dědí z třídy *BStreamAbstract* a obsahuje tyto důležité prvky:

```
private:
    int Volume;
    float VolumeAbsolute;
    bool Mute;
    BInputStream * InputConnection;
    BEqualiser LocalEqualiser;
signals:
    void Remote_VolumeChanged(int vol);
    void Remote_MuteChanged(bool mute);
    void ChangeOnRemote(const BOutputStream * stream);
public:
    void If_SetStream(const BoutputStream_t * str,
        const BMaster * master);
    void connectStream(BStreamAbstract *stream,
        bool emitSignal = true);
    void RemoteChanged(bool mute, float vol);
```

Proměnná *Volume* určuje zeslabení výstupního streamu ve slave jednotce, tato hodnota je v dB a na je interně automaticky přepočtena na absolutní hodnotu, která je v proměnné *VolumeAbsolute*. Proměnná *Mute* určuje jestli je výstupní kanál úplně tlumen (vypnut). Proměnná *BInputStream* ukazuje na připojený vstupní audio stream a proměnná *LocalEqualiser* typu *BEqualiser* obsahuje informace o nastavení ekvalizéru výstupního streamu.

Komunikace pomocí signálů je opět znázorněna na obrázku 6.4. Signál *Remote_VolumeChanged* je generován v případě, že je přes rozhraní přijata nová hodnota hlasitosti. Signál *Remote_MuteChanged* funguje úplně shodně. Tyto signály jsou

poté přímo připojeny přímo na GUI tzn. na komponenty *QSlider* a *QCheckBox*.



Obr. 6.4: Komunikace třídy *BOuputStream* s okolím

Metoda *If_SetStream* funguje úplně stejně jako u vstupního streamu. Slouží k nastavení výstupního streamu pomocí nízkoúrovňové struktury, která je uvedena v příloze B. Metoda *connectStream* slouží k připojení vstupního streamu, parametr emitSignal opět rozhoduje jestli bude generován signál *ChangeOnRemote* jako v případě vstupního streamu. Metoda *RemoteChanged* je volána v případě změny přímo v jednotce master, pokud přijde příkaz SEND_OUTPUTSTREAM (4.4.1). Metoda má za úkol nastavit hodnoty *Volume* a *Mute* podle vstupních parametrů, zároveň generuje signály *Remote_MuteChanged* a *Remote_VolumeChanged*.

6.1.4 BEqualiser

Třída *BEqualiser* obsahuje celkové nastavení ekvalizéru a to uživatelské nastavení jednotlivých pásem v dB. Do jednotky master jsou zasílány právě hodnoty 0-31 jako index v tabulce, kde má jednotka uložena již předem vypočtené koeficienty číslicových filtrů, které jsou implementovány v jednotce master. Také je zde zaznamenáno zdali je ekvalizér aktivován či nikoliv.

6.1.5 BInputDevice

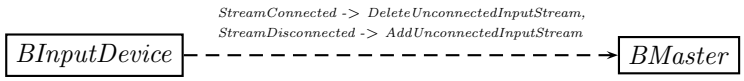
Třída *BInputDevice* obsahuje vstupní audio streamy jednoho vstupního zařízení a umožňuje k těmto streamům přístup, jejich připojování a odpojování. Důležitými prvky jsou:

```
private:
    QString Name;
    QMap<unsigned int, BInputStream *> InputStreamList;
    unsigned int UID;
signals:
    void StreamDisconnected(BInputStream * stream);
    void StreamConnected(BInputStream * stream);
public slots:
```

```
void DisconnectInputStream(BInputStream * stream);
void ConnectInputStream(BInputStream * stream);
```

Proměnná *Name* je uživatelský název vstupního zařízení. Proměnná *UID* je unikátní identifikátor vstupního zařízení, který je po vytvoření třídy neměnný a je zpřístupněn pouze pro čtení. Proměnná *InputStreamList* je asociativní kontejner ukazatelů na vstupní datové streamy *BInputStream*, které jsou připojeny k zařízení. *QMap* umožňuje rychlé vyhledávání pomocí UID jednotlivých streamů.

Sloty *DisconnectInputStream* a *ConnectInputStream* mají za úkol přidávat(mazat) vstupní datové streamy do vstupního zařízení, zároveň generují signály *StreamConnected* a *StreamDisconnected*, které jsou připojeny ke třídě *BMaster*, kde jsou naopak přidány nebo odebrány ze seznamu nepřipojených vstupních streamů. Komunikace této třídy s okolím z pohledu signálů/slotů je zobrazena na obrázku 6.5.



Obr. 6.5: Komunikace třídy *BInputDevice* s okolím

6.1.6 BUnitAbstract

Třída *BUnitAbstract* je pouze abstraktní třída, ze které jsou odvozeny třídy *BSlave* a *BMaster*. Třída obsahuje obecné informace společné pro obě jednotky master a slave:

```
protected:
    QString Name;
    QHostAddress IPAddressActual;
    unsigned int SerialNumber;
    unsigned int SWVersion;
    unsigned int HWVersion;
```

Význam jednotlivých proměnných je patrný z jejich názvů, *Name* je název jednotky, *SerialNumber* je sériové číslo jednotky, *SWVersion* je verze firmwaru a *HWVersion* je verze hardwaru. Třída také obsahuje přístupové metody k těmto vlastnostem.

6.1.7 BSlave

Třída *BSlave* je velmi rozsáhlá třída a má tyto důležité prvky:

```
private:
    QMap<BOutputStream *, unsigned int>
        OutputStreamList;
    BInputDevice * ConnectedInputAudioDevice;
    BEqualiser GlobalEqualiser;
    bool VolumeBound;
    bool Mute;
    bool EqForAll;
    uint16_t GPIO_in;
    uint16_t GPIO_out;
    int Temperature[TEMPERATURE_COUNT];
signals:
    void VolumeAligned(BaO::BSlave *);
    void TemperatureChanged(const BSlave *);
    void TemperatureChanged();
    void GpioChanged(BaO::BSlave *);
    void GpioChanged();
    void Gpio_changeOnRemote(const BSlave *);
public:
    void If_SetSlave(const BSlave_t * slave);
    void If_SetTemperatures
        (const int16_t * temperatures);
    void If_SetGpio(uint16_t in, uint16_t out);
    void Remote_changeGpio(uint16_t in, uint16_t out);
    void User_changeGpioOutput(uint16_t out);
public slots:
    void AlignVolume(void);
    void ConnectInputDevice(QString & nejm);
    void ConnectInputDevice(int idx);
    void ConnectInputDevice(BInputDevice * device);
```

OutputStreamList je seznam všech výstupních streamů slave jednotky, tento seznam je naplněn pouze jednou, při vytváření třídy a dále už je neměnný, protože skutečná slave jednotka má také určitý počet streamů, který je schopna přehrát. Seznam je naplněn právě pomocí připojené jednotky v systému. *ConnectedInputAudioDevice* je připojené vstupní zařízení *BInputDevice*.

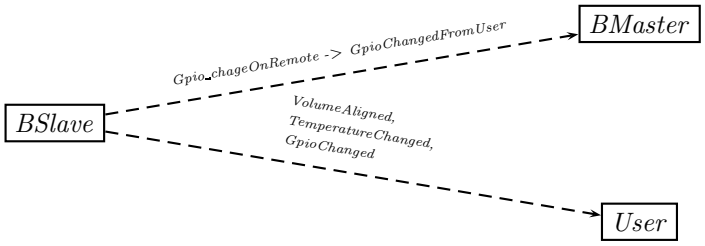
GlobalEqualiser je nastavení globálního ekvalizéru tzn. bude použito jedno nastavení ekvalizéru pro všechny výstupní streamy, o tom jestli bude použit jeden ekvalizér nebo dílčí ekvalizéry ve streamech rozhoduje proměnná *EqForAll*.

Proměnná *VolumeBound* oznamuje jestli jsou hlasitosti výstupních streamů svázány, pomocí této proměnné jsou pak propojeny komponenty *QSlider* mezi sebou a pohyb s jedním posuvníkem ovlivní hlasitosti všech výstupních streamů slave jednotky.

Proměnné *GPIO_in* a *GPIO_out* jsou stavy univerzálních vstupů/výstupů slave jednotky. Pole *Temperature* jsou hodnoty teplot měřených slave jednotkou.

Signál *VolumeAligned* je generován ve funkci *AlignVolume*, která má za úkol zprůměrovat hlasitosti všech streamů, signál je poté připojen ke GUI, kde jsou poté přenastaveny hodnoty posuvníků s hlasitosmi.

Signál *Gpio_changeOnRemote* je generován ve funkci *User_changeGpioOuput*, která je volána v případě, že uživatel změní pomocí GUI stavy univerzálních výstupů. Tento signál je poté připojen ke slotu *GpioChangedFromUser* třídě *BMaster*, který zajistí odeslání změn přes rozhraní do jednotky master. Komunikace pomocí signálů s okolím je zobrazena na obrázku 6.6.



Obr. 6.6: Komunikace třídy *BSlave* s okolím pomocí signálů

Přetížené signály *TemperatureChanged* a *GpioChanged* jsou generovány ve funkcích *If_SetTemperatures* a *If_SetGpio*, které jsou volány pokud přes komunikační rozhraní přijdou příkazy SEND_TEMPERATURE nebo SEND_GPIO (4.4.1). Tyto signály jsou opět napojeny ke GUI.

Přetížené sloty *ConnectInputDevice* mají za úkol připojit ke slave jednotce vstupní audio zařízení včetně všech streamů.

6.1.8 BSlaves

Třída *BSlaves* je pouze kontejner slave jednotek *BSlave* a zahrnuje tyto prvky:

```
private:
    QMap<unsigned int ,BSlave *> SlaveList;
public slots:
    void AddSlave(BSlave * slave);
```

Proměnná *SlaveList* je výše zmiňovaný kontejner slave jednotek, který je opět naplněn pouze jednou po připojení k master jednotce a dále je neměnný. Funkce *AddSlave* slouží k přidání slave jednotky do kontejneru.

Tato třída je použita v třídě *Bmaster* jako privátní proměnná, takže jedině tato třída může použít funkci *AddSlave*, pro uživatele je tato proměnná (a její funkce) nepřístupná.

6.1.9 BMaster

Třída *BMaster* je nejvíce rozsáhlá třída a mimo jiné poskytuje všechny potřebné přístupové metody k objektům *BSlave*, *BInputStream*, *BOutputStream*. Třída zahrnuje tyto důležité prvky:

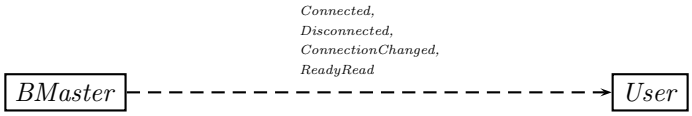
```
public:
    void If_SetMaster(BMaster_t mas);
    void If_AddInputDeviceStream
        (BInputStream_t in);
    void If_AddSlave(BSlave_t slav ,
        const QList<BoutputStream_t> * list);
    void ConnectRemote(CommunicationInterface_t type,
        QHostAddress * address = NULL);
    void DisconnectRemote();

public slots:
    void AddInputDevice(BInputDevice * device);
    void AddUnconnectedInputStream(BInputStream * stream);
    void DelUnconnectedInputStream(BInputStream * stream);

signals:
    void Connected(void);
    void Disconnected(void);
    void ConnectionChanged(bool);
    void ReadyRead(void);
```

Metody začínající prefixem *If_* jsou určeny k nastavení vlastností podle skutečné jednotky master, které jsou získány pomocí komunikačního rozhraní. Funkce *If_SetMaster* slouží k nastavení vlastností instance *BMaster*, funkce *If_AddInputDevice* slouží k přidání vstupního zařízení a funkce *If_AddSlave* slouží k přidání slave jednotky.

Metody *ConnectRemote* a *DisconnectRemote* jsou určeny pro navázání/ukončení spojení s jednotkou master, kde parametrem je komunikační rozhraní.

Veřejné sloty *AddInputDevice*, *AddUnconnectedInputStream* a *DelUnconnectedStream* slouží k přidávání vstupních zařízení, které je možno již uživatelsky definovat.

Signály *Connected*, *Disconnected* a *ConnectionChanged* jsou generovány při připojení/odpojení master jednotky a signál *ReadyRead* je generován v případě, že jsou již po připojení k master jednotce vytvořeny a inicializovány vlastnosti všech objektů, které jsou přijaty přes komunikační rozhraní od jednotky master. Komunikace třídy s je znázorněna na obrázku 6.7.



Obr. 6.7: Znázornění komunikace třídy *BMaster* pomocí signálů

6.1.10 BEthernet

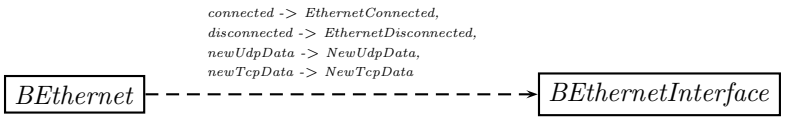
Třída *BEthernet* zajišťuje nejnižší úroveň rozhraní pro komunikaci s master jednotkou přes rozhraní ethernet a obsahuje tyto důležité prvky:

```
public slots:
    void TcpSend(QByteArray & data);
signals:
    void connected(void);
    void disconnected(void);
    void newUdpData(QByteArray & data);
    void newTcpData(QByteArray & data);
```

Slot *TcpSend* pouze uloží surová data bez jakéhokoliv zpracování (zpracování má na starost třída *BEthernetInterface*) do výstupní fronty, která je postupně na pozadí sama vyprazdňována a její obsah je odesílán přes TCP socket do master jednotky na přesně stanoveném portu CONFIGURATION_TCP_PORT (příloha B).

Signály *connected* a *disconnected* jsou generovány při sestavení/zrušení TCP spojení s master jednotkou.

Signály *newUdpData* a *newTcpData* jsou generovány při přijmu dat od master jednotky přes rozhraní ethernet. Signály jsou připojeny k vyšší vrstvě *BEthernetInterface* ke zpracování. Znázornění komunikace pomocí signálů této třídy s okolím je na obrázku 6.8



Obr. 6.8: Znázornění komunikace třídy *BEthernet* pomocí signálů

6.1.11 BAbstractInterface

Třída *BAbstractInterface* je pouze abstraktní třídou, která nemůže být přímo deklarována, jelikož opět obsahuje virtuální metody bez implementace. Z této třídy jsou odvozeny třídy *BEthernetInterface* a *BUsbInterface*, které díky tomu poskytují naprosto stejné rozhraní vyšší třídě *BInterface*. Třída obsahuje tyto prvky:

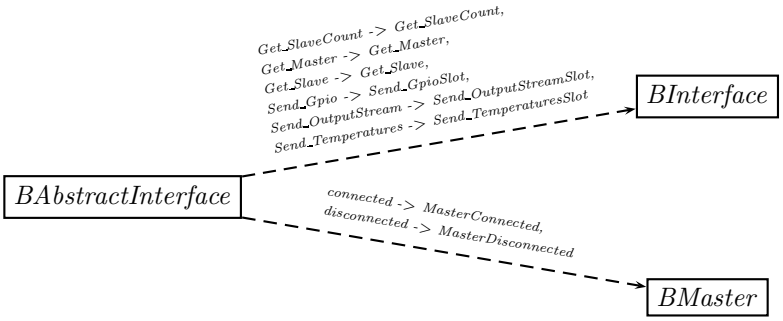
```
public:
    virtual void GetSlaveCount() = 0;
    virtual void GetMaster() = 0;
    virtual void GetSlave(int idx) = 0;

public slots:
    virtual void SetInputStream
        (const BInputStream * stream) = 0;
    virtual void SetOutputStream
        (const BOutputStream * stream) = 0;
    virtual void SetGpio
        (const BSlave * slave) = 0;

signals:
    void Get_SlaveCount(int count);
    void Get_Master(BMaster_t master,
        QList<BInputStream_t> * stream);
    void Get_Slave(BSlave_t slave,
        QList<BoutputStream_t> * stream,
        int idx);
    void Send_Gpio(uint32_t serialNumber,
        uint16_t in, uint16_t out);
    void Send_OutputStream(uint32_t serialNumber,
        uint16_t uid, port_float_t vol,
        bool_t mute);
    void Send_Temperatures(uint32_t serialNumber,
```

```
        int16_t * temperatur );
    void connected(void);
    void disconnected(void);
```

Funkce *GetSlaveCount*, *GetMaster* a *GetSlave* mají v reálné implementaci za úkol pomocí nižší vrstvy poslat do master jednotky dotaz. Po přijetí odpovědí a zpracování v interních slotech musí tyto interní sloty také generovat signály *Get_SlaveCount*, *Get_Master* a *Get_Slave* pro vyšší vrstvu.

Sloty *SetInputStream*, *SetOutputStream* a *SetGpio* musí v implementaci odvozených tříd zajistit odeslání příkazu s nastavením vstupního, výstupního streamu nebo univerzálních výstupů.

Signály *Send_Gpio*, *Send_OutputStream*, *Send_OutputStream* a *Send_Temperatures* musí být generovány pokud master jednotka vyšle změny těchto vlastností. A konečně signály *connected* a *disconnected* musí být generovány pokud je úspěšné připojení k master jednotce nebo je naopak spojení přerušeno. Znázornění komunikace pomocí signálů je na obrázku 6.9.



Obr. 6.9: Znázornění komunikace odvozených tříd z *BAbstractInterface* pomocí signálů

6.1.12 BEthernetInterface

Třída *BEthernetInterface* je odvozená třída z třídy *BAbstractInterface*, která jí dává povinnost implementovat výše uvedené metody pro komunikaci s vyšší vrstvou a také metody pro komunikaci s jednotkou master, v tomto případě přes rozhraní ethernet. Třída zahrnuje tyto důležité prvky:

```
public:
    void GetSlaveCount();
    void GetMaster();
    void GetSlave(int idx);
```

```
public slots:
    void SetInputStream(const BInputStream * stream);
    void SetOutputStream(const BOutputStream * stream);
    void SetGpio(const BSlave * slave);
private slots:
    void NewTcpData(QByteArray & data);
    void NewUdpData(QByteArray & data);
    void EthernetConnected(void);
    void EthernetDisconnected(void);
```

Metody *GetSlaveCount*, *GetMaster* a *GetSlave* byly již popsány v 6.1.11, v této třídě jsou pouze implementovány pro odesílání datových rámců GETSLAVECOUNT, GETMASTER a GETSLAVE podle 4.4.1.

Metody *SetInputStream*, *SetOutputStream* a *SetGpio* mají za úkol zpracovat data z odpovídajících tříd, vytvořit datové rámce podle 4.4.1 a ty po té odeslat pomocí třídy *BEthernet* do jednotky master.

Privátní slot *NewTcpData* je připojen na signál *NewTcpData* z třídy *BEthernet* a slouží k převedení přijatého rámce na vyšší úroveň a vyslání patřičného signálu podle typu přijatého příkazu. Signály jsou opět popsány v abstraktní třídě *BAbstractInterface* v 6.1.11.

Privátní slot *NewUdpData* je připojen k signálu *NewUdpData* z třídy *BEthernet* a slouží ke zpracování datového audio streamu.

Privátní sloty *EthernetConnected* a *EthernetDisconnected* jsou také připojeny k signálům *connected* a *disconnected* z třídy *BEthernet* a pouze vyšlou další signál pro vyšší vrstvu *connected* a *disconnected*, které jsou opět děděny z abstraktní třídy.

6.1.13 BInterface

Třída *BInterface* je nejvyšší vrstvou komunikace s jednotkou master. Poskytuje rozhraní pro zasílání změn do jednotky master a naopak také přímo provádí změny přijaté z jednotky master na nejvyšší úrovni. Třída obsahuje tyto hlavní prvky:

```
public:
    BInterface(QObject * par,
        CommunicationInterface_t interface);
public slots:
    void Set_Gpio(const BSlave * slave);
    void Set_OutputStream
        (const BOutputStream * stream);
    void Set_InputStream
        (const BInputStream * stream);
```

```
private slots:
    void Get_SlaveCount(int count);
    void Get_Master(BMaster_t master,
        QList<BInputStream_t> * stream);
    void Get_Slave(BSlave_t slave,
        QList<BoutputStream_t>
        *stream, int idx);
    void Send_GpioSlot(uint32_t serialNum,
        uint16_t in, uint16_t out);
    void Send_OutputStreamSlot(uint32_t serNum,
        uint16_t uid, port_float_t vol,
        bool_t mute);
    void Send_TemperaturesSlot(uint32_t serNum,
        int16_t * temper);
```

Konstruktor je v této třídě velmi důležitý, neboť přebírá parametr, přes které rozhraní má probíhat veškerá komunikace s jednotkou master.

Veřejné sloty *Set_Gpio*, *Set_OutputStream*, *Set_InputStream* jen přepošlou argument do jednotky master pomocí rozhraní, které bylo vybrano v konstruktoru.

Privátní slot *Get_SlaveCount* přijímá počet připojených slave jednotek.

Privátní slot *Get_Master* přijímá informace o master jednotce a všech jejich vstupních streamech, na jejichž základě nastaví proměnnou *master* pomocí metody *If_SetMaster* a přidá do něj streamy pomocí metody *If_AddInputDeviceStream*.

Privátní slot *Get_Slave* přijímá informace o jednotce slave a všech jejich výstupních streamech a přidá do seznamu jednotek pomocí metody *If_AddSlave*.

Privátní slot *Send_GpioSlot* nastaví podle sériového čísla slave jednotky stavy univerzálních vstupů a výstupů ve správné instanci třídy *BSlave* pomocí metody *If_SetGpio* třídy *BSlave*.

Privátní slot *Send_OutputStreamSlot* slouží ke zpracování informací o změně nastavení výstupního streamu a podle sériového čísla slave jednotky a identifikátoru streamu nastaví hodnotu hlasitosti a stav zatlumení pomocí metody *RemoteChanged*, která generuje signály zmiňované v 6.1.3 pro zobrazení změn v GUI.

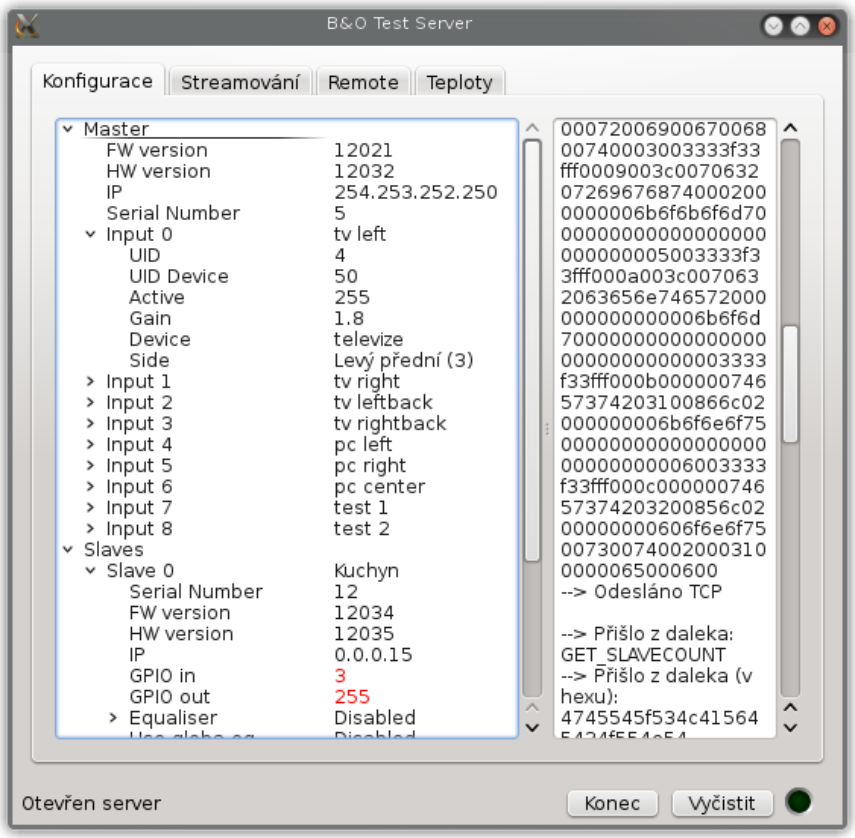
Privátní slot *Send_TemperaturesSlot* funguje na podobném principu jako slot *Send_OutputStreamSlot*, akorát nastavuje teploty pomocí metody *If_SetTemperatures* v instanci *BSlave*, která je vyhledána pomocí sériového čísla.

7 SIMULACE AUDIO SYSTÉMU NA PC

Tento testovací program má za úkol plně emulovat jednotku master, tzn. má sadu vlastností, které bude mít skutečná jednotka master implementovaná na DSP. Dále obsahuje vlastnosti od jednotlivých slave jednotek, které jsou zapojeny do systému. Tyto slave jednotky bude skutečná master jednotka schopna vyhledat a zpřístupnit přes svoje rozhraní. V této aplikaci jsou tyto vlastnosti zadány přímo v programu, což pro účely testování komunikačního protokolu a vývoje hlavní řídící aplikace plně postačuje.

7.1 Konfigurace

Testovací program je schopen přijímat příkazy protokolem TCP podle definovaného rozhraní (v kapitole 4.4.1) a měnit podle nich některé vlastnosti, které jsou přehledně zobrazeny v této kartě, nebo naopak je schopen posílat do řídícího softwaru změny, které mohou nastat na reálném hardwaru (změna teploty, změna stavu univerzálních vstupů/výstupů atd.).

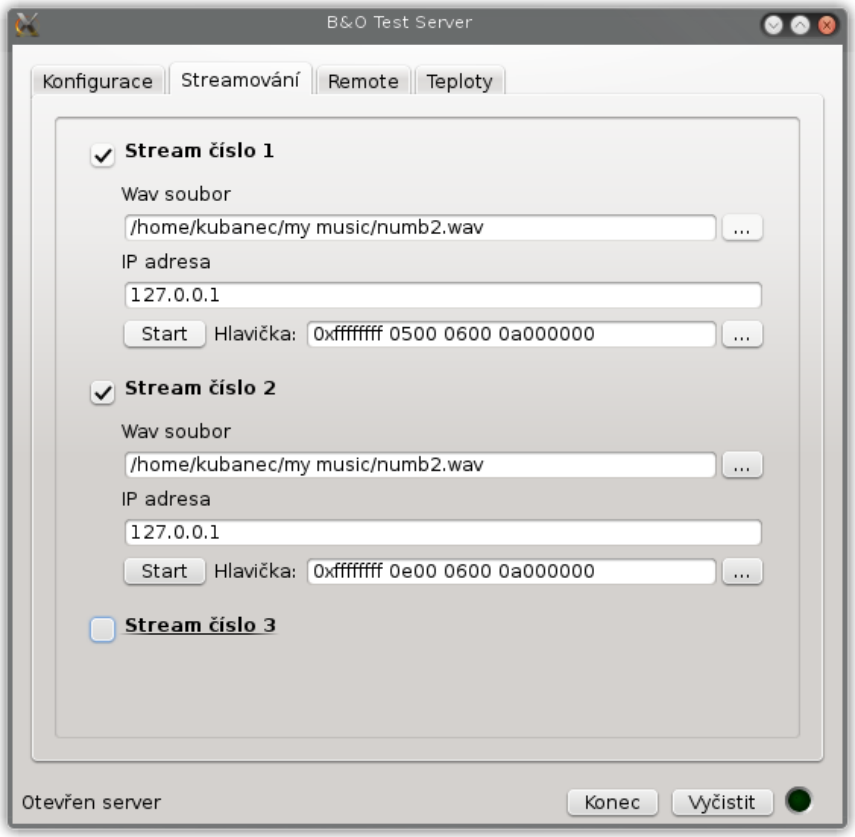


Obr. 7.1: Hlavní formulář testovací aplikace

Okno testovacího programu s kartou konfigurace je na obrázku 7.1, na kterém je vidět souhrn některých vlastností v levé části a v pravé části je vidět výpis komunikace, tlačítko ve stavovém řádku Start/Konec spouští/uzavírá TCP server pro připojení klientů (v tomto případě řídící software) a zelený indikátor ukazuje zda je připojen klient. Červeně zobrazené vlastnosti je možno přepsat a jejich změna bude zaslána do ovládacího softwaru ke zpracování. Soupis vlastností, které je možno měnit přímo ve fyzických jednotkách je popsán v kapitole 4.4.1.

7.2 Streamování

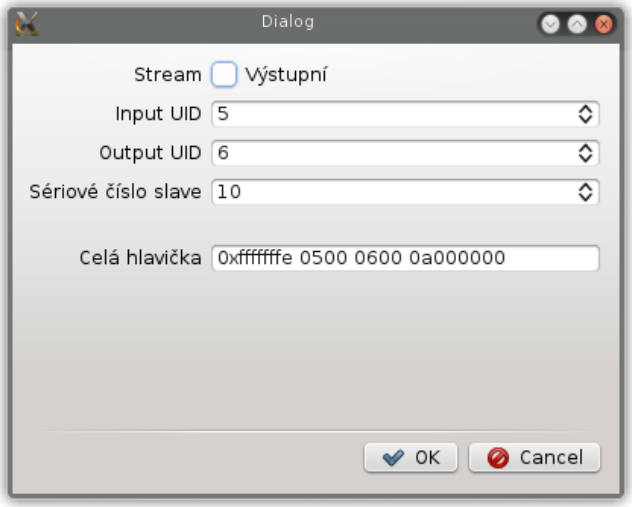
Dále je tento program schopen posílat datové streamy protokolem UDP, podle výše definovaného protokolu v kapitole 4.4.2, které je schopen řídící software zobrazit jako časový průběh, spektrum, uložit do souboru atd.



Obr. 7.2: Testovací aplikace - karta pro streamování audio dat

Tuto funkci má na starost karta, která je vidět na obrázku 7.2. Karta umožňuje vytvořit teoreticky nekonečno datových UDP streamů, každému streamu lze nastavit jiný wav soubor (soubor musí splňovat určité podmínky např. 48KHz vzorkovací frekvence), dále je možno nastavit IP adresu, na kterou má být datový tok odesílán

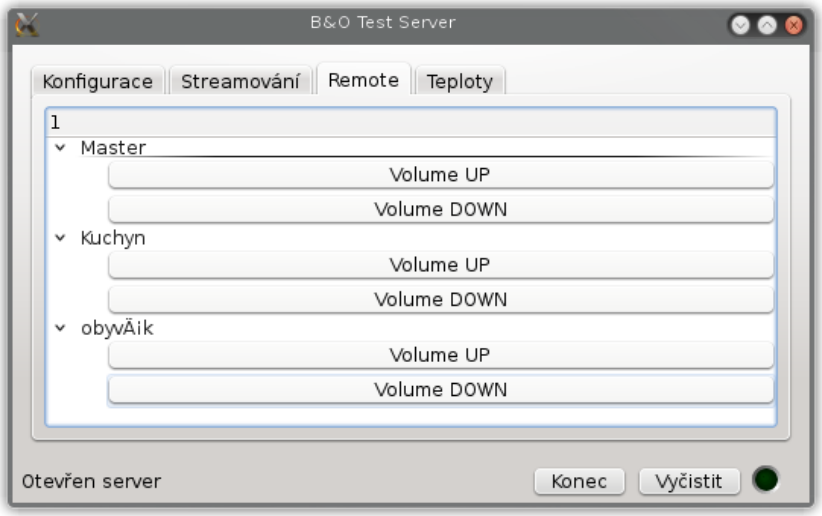
a hlavně konfigurace UDP hlavičky. K tomuto účelu byl vytvořen dialog pro přesné nastavení hlavičky, který je vidět na obrázku 7.3. Hlavička je poté skládána do datového toku pomocí maker ze souboru btypes.h pro zajištění konzistence dat na všech komponentách systému, které pracují s UDP streamy.



Obr. 7.3: Dialog pro nastavení hlavičky UDP streamu

7.3 Dálkové ovládání

Program je dále schopen zasílat příkazy dálkového ovládání, které umí ovládací software zaznamenávat. Karta určená k tomuto účelu je vidět na obrázku 7.4.

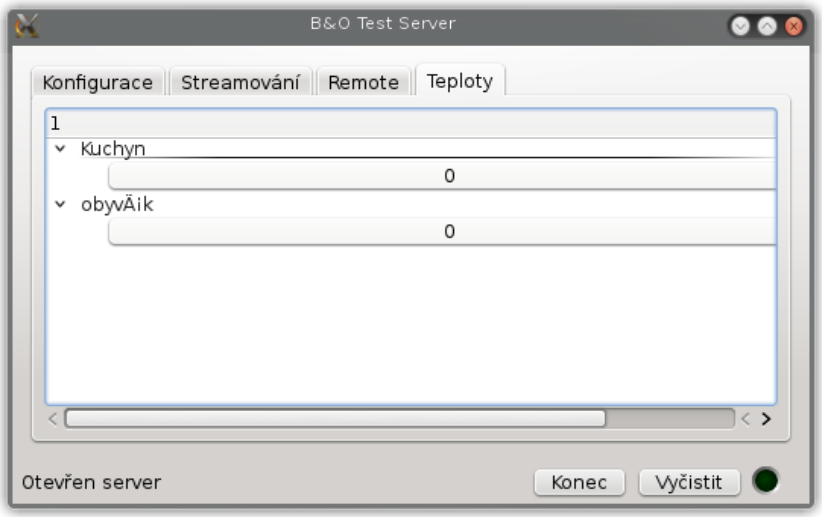


Obr. 7.4: Testovací aplikace - Dálkové ovládání

7.4 Teploty

Jako poslední funkce testovacího programu je odesílání teplot do ovládacího SW, které je ovládací software také schopen zaznamenat a také je případně zobrazit v grafu. Karta pro odesílání teplot je vidět na obrázku 7.5.

Při stisknutí tlačítka jsou odeslány vždy všechny teploty, jejich počet je definován také v btypes.h. Při každém dalším stisknutí jsou teploty interně inkrementovány, což je vhodné pro testování grafické závislosti v ovládacím software.



Obr. 7.5: Testovací aplikace - Teploty

8 ZÁVĚR

Diplomová práce řeší problematiku ovládacího software pro digitální reproduktorový systém. A to jak teoretický návrh funkce celého systému, návrh použitých komunikačních protokolů, návrh grafického rozhraní.

Po této fázi následovala fáze vlastního vývoje řídícího software, kde byly postupně implementovány a otestovány třídy pro práci s digitálním reproduktorovým systémem. Testování probíhalo pomocí zmiňované testovací aplikace.

Během vývoje tříd BaO bylo zároveň vyvíjeno a testováno i grafické uživatelské rozhraní, které je složeno přibližně z 20 ovládacích oken a dialogů. V neposlední řadě padlo velké úsilí na vývoj sestavy widgetů pro zobrazení a práci se vzorky audiosignálu. Funkce a vzhled tohoto widgetu byl inspirován několika nahrávacími programy, ovšem cílem projektu nebylo vyvinout nahrávací software do studia.

Celý koncept software a jeho komunikačních protokolů byl vymyšlen tak, aby nebylo problematické případné rozšiřování o nové funkce a vlastnosti. Samozřejmě není možno dělat jednoduše změny v hierarchii tříd BaO, jelikož tyto třídy jsou, navzdory snaze o neprovázanost, příliš provázané.

Bohužel se již nepovedlo implementovat a otestovat komunikaci přes rozhraní USB, což byl autorův původní plán. Je potřeba dodat, že k testování této komunikace je nepostradatelný skutečný hardware s funkčním USB rozhraním na, kterém by bylo možno testovat, ovšem komunikace pomocí rozhraní ethernet by měla být dostačující. V případě vývoje USB rozhraní by bylo vhodné použít stejný protokol jako v případě ethernetového rozhraní. Stačilo by pouze implementovat dvojci tříd *BUsb* a *BUsbInterface* a díky dědičnosti z abstraktní třídy *BAbstractInterface* by měla být zajištěna veškerá funkcionalita jako v případě ethernetu.

LITERATURA

[1] Smékal, Z., Sysel, P. *Signálové procesory.* 1. vydání. Praha: Sdělovací technika, 2006. 283 s. ISBN 80-86645-08-8

[2] Ján, J. *Číslicová filtrace, analýza a restaurace signálů* (druhé rozsírené vydání),VUTIUM Brno, 2002

[3] Kozumplík, J., Kolář, R., Ján, J. *Číslicové zpracování signálů: V prostředí Matlab.* Brno, 2001.

[4] Floating-point unit. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-03-29]. Dostupné z: <`http://en.wikipedia.org/wiki/Floating-point_unit`>

[5] SMITH, Steven. The Scientist and Engineer's Guide to Digital Signal Processing [online]. 1997 [cit. 2012-03-29]. ISBN 0-9660176-3-3. Dostupné z: <`http://www.dspguide.com/`>

[6] Red Book (CD standard). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 19 March 2012 [cit. 2012-03-29]. Dostupné z: <`http://en.wikipedia.org/wiki/Red_Book_(CD_standard)`>

[7] SPDIF. *The Tech-FAQ* [online]. [2011] [cit. 2012-03-29]. Dostupné z: <`http://www.tech-faq.com/spdif.html`>

[8] COMPAQ COMPUTER CORPORATION, Digital Equipment Corporation, IBM PC Company, Intel Corporation, Microsoft Corporation, NEC, Northern Telecom. *Universal Serial Bus Specification Revision 1.0* [online]. January 15, 1996 [cit. 2012-03-29]. Dostupné z: <`http://fl.hw.cz/docs/usb/usb10doc.pdf`>

[9] *Universal Serial Bus Specification Revision 2.0* [online]. 10/20/2000 [cit. 2012-03-29]. Dostupné z: <`http://fl.hw.cz/docs/usb/ecn1.pdf`>

[10] Ethernet (IEEE802.3). *A Made IT project* [online]. [2000] [cit. 2012-03-29]. Dostupné z: <`http://ckp.made-it.com/ieee8023.html`>

[11] ST Microelectronics. *RM0090 Reference manual : STM32F405xx, STM32F407xx, STM32F415xx and STM32F417xx advanced ARM-based 32-bit MCUs* [online]. 2011, 1316s., 16-Sep-2011 [cit. 2012-03-11]. Doc ID 018909 Rev 1. Dostupné z: <`http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/REFERENCE_MANUAL/DM00031020.pdf`>

[12] ST Microelectronics. *User manual : STM32F4DISCOVERY STM32F4 high-performance discovery board* [online]. [s.l.] : [s.n.], 2012 [cit. 2012-03-11] Dostupné z: <http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/USER_MANUAL/DM00039084.pdf>

[13] ARM. *Cortex-M4: Technical Reference Manual* [online]. 2010, 121 s. [cit. 2012-03-29]. Dostupné z: <http://www.google.cz/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCkQFjAA&url=http%3A%2F%2Finfocenter.arm.com%2Fhelp%2Ftopic%2Fcom.arm.doc.ddi0439c%2FDDI0439C_cortex_m4_r0p1_trm.pdf&ei=v7d0T8icJYGVOq6ukOkK&usg=AFQjCNFkM_NA0wMJ2ugKRTCS9BUiU2DPuw>

[14] MICREL. *KSZ8021RNL / KSZ8031RNL: 10Base-T/100Base-TX PHY with RMII Support* [online]. 2010 [cit. 2012-04-02]. Dostupné z: <http://www.farnell.com/datasheets/1447923.pdf>

[15] *WxWidgets: Cross-Plaform GUI Library* [online]. 2007- [cit. 2012-04-06]. Dostupné z: <http://www.wxwidgets.org/>

[16] NOKIA. *Qt* [online]. 2011 [cit. 2012-04-06]. Dostupné z: <http://doc.qt.nokia.com/>

[17] The GTK+ Project. The GTK+ Project [online]. 2007 [cit. 2012-04-30]. Dostupné z: <http://www.gtk.org/>

[18] *Libusb* [online]. 2009-, 03/03/12 [cit. 2012-04-06]. Dostupné z: <http://www.libusb.org/>

[19] AUDACITY TEAM. *Audacity* [online]. 21.1.2013 [cit. 2013-04-29]. Dostupné z: <http://audacity.sourceforge.net/>

[20] ERIK DE CASTRO LOPO. *Libsndfile* [online]. 1999, 2011 [cit. 2013-04-29]. Dostupné z: <http://www.mega-nerd.com/libsndfile/>

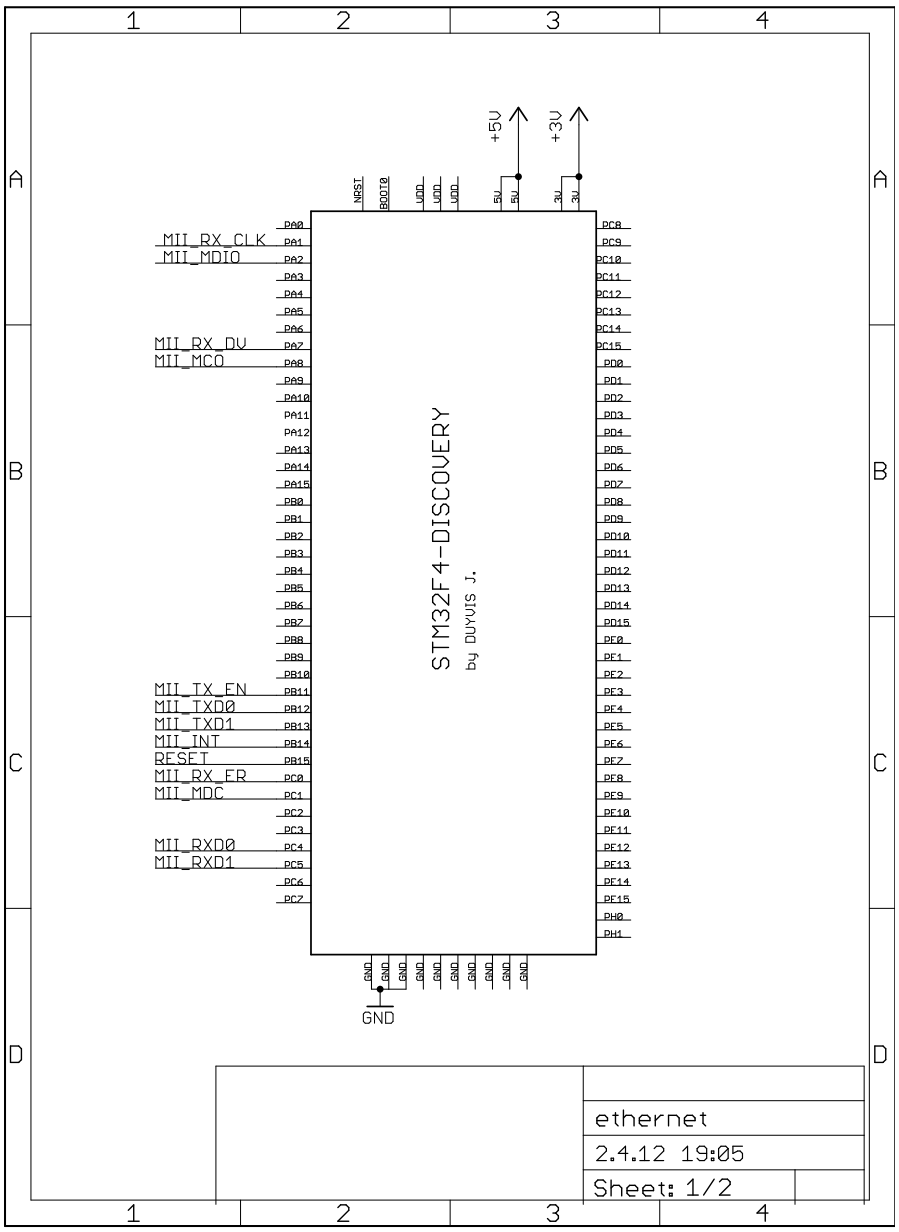
[21] Frigo, M., Johnson, S., FFTW [online]. 2003 [cit. 2013-04-29]. Dostupné z: <http://www.fftw.org/>

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

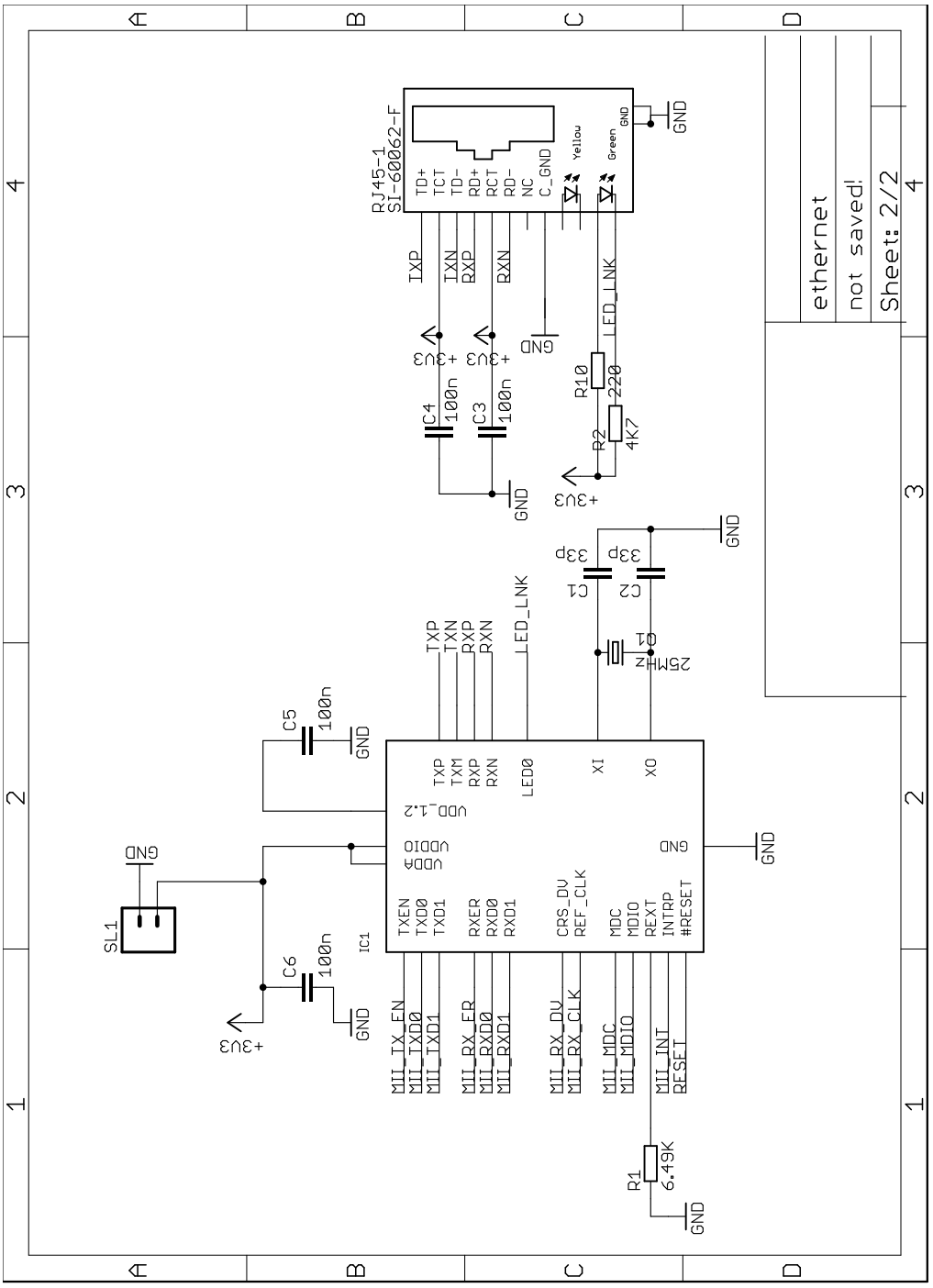
GUI	Graphical user interface
DSP	Digital signal processor
FPU	Floating-point unit
FIR	Finite impulse response
IIR	Infinite impulse response
PC	Personal computer
FS	Full speed
HS	High speed
NF	nízkofrekvenční
A/D	analog to digital
USB	Universal Serial Bus

SEZNAM PŘÍLOH

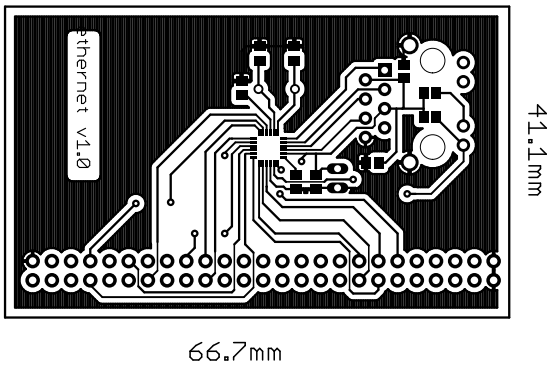
A ETHERNETOVÝ MODUL



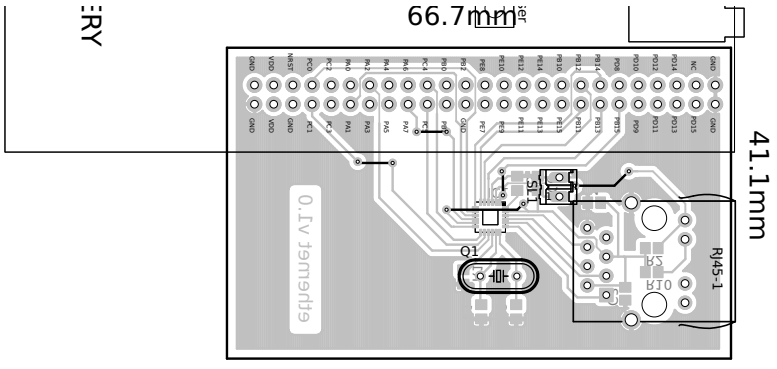
Obr. A.1: Schéma zapojení vývojového modulu pro ethernet, list 1



Obr. A.2: Schéma zapojení vývojového modulu pro ethernet, list 2



Obr. A.3: Deska plošného spoje pro modul ethernetu



Obr. A.4: Rozložení součástek modulu pro ethernet

B VÝPIS SOUBORU BTYPES.H

```
#ifndef BTYPES_H
#define BTYPES_H

#ifdef QT_CORE_LIB
#include <Qt>
#include <QStringList>

typedef quint8      uint8_t;
typedef quint16     uint16_t;
typedef quint32     uint32_t;
typedef qint8       int8_t;
typedef qint16      int16_t;
typedef qint32      int32_t;
typedef float       port_float_t;
typedef quint8     port_bool_t;

#define pFALSE      ((port_bool_t) 0)
#define pTRUE       ((port_bool_t) ~0)

#define to8bit()    toUtf8()
#define from8bit(x)  QString::fromUtf8(x)


typedef enum
{
    Refused, Closed
} CommunicationError_t;
#define COMMUNICATION_ERROR_TABLE trUtf8("Nelze␣navázat␣spojení") \
    << trUtf8("Spojení␣bylo␣přerušeno")

namespace BaO
{
typedef enum
{
    Usb, Ethernet
} CommunicationInterface_t;

#define COMM_INTERFACE_LIST "USB" << "ETHERNET"


#define INFRA_COMMANDS_LIST "Volume␣UP" << "Volume␣DOWN"
#define SIDE_LIST_QSTRING trUtf8("Center") << trUtf8("Subwoofer") \
    << trUtf8("Levý␣zadní") \
    << trUtf8("Levý␣přední") << trUtf8("Pravý␣zadní") \
    << trUtf8("Pravý␣přední") << trUtf8("nic")

#define SIDE_LIST_SHORT_QSTRING trUtf8("Cen") << trUtf8("Sub") \
    << trUtf8("LZ") << trUtf8("LP") << trUtf8("PZ") \
    << trUtf8("PP") << trUtf8("nic")

#define BAND_LIST "60Hz" << "150Hz" << "400Hz" << "1KHz" \
    << "2.4KHz" << "6KHz" << "15KHz"

#define TEMPERATURE_NAMES_LIST trUtf8("satelit") << trUtf8("zesilovač") \
    << trUtf8("další␣zesilovač") << trUtf8("zeslabovač") << trUtf8("konec")
```

```
#if (QT_VERSION < 0x050000)
    #define setSectionResizeMode setResizeMode
#endif
}

#define INI_FILE "bang.ini"

#else
#include "ch.h"
#include "hal.h"

typedef float       port_float_t;
typedef uint8_t     port_bool_t;

#endif

#ifdef QT_CORE_LIB
namespace BaO
{
#endif

typedef enum {ANALOG_UP_1, ANALOG_DW_1} remoteCommand_t;

#ifdef QT_CORE_LIB
}
#endif

#define CONFIGURATION_TCP_PORT          15152
#define UDP_PORT                        25001
#define SAMPLE_COUNT                    336
#define UDP_HEADER_LENGTH               16
#define PACKET_LENGTH              ((SAMPLE_COUNT * 4) + UDP_HEADER_LENGTH)

#define GET_MASTER                      "GET_MASTER"
#define GET_SLAVECOUNT                  "GET_SLAVECOUNT"
#define GET_SLAVE                       "GET_SLAVEN"
#define SEND_GPIO                       "SEND_GPIO"
/// asynchronní - posiláš podle sebe
#define SEND_TEMPERATURE                "SEND_TEMP"
    // SERIAL4bytyTEPLOTY2byty*počet
#define SEND_OUTPUTSTREAM               "SEND_OUTSTREAM"
#define SEND_INFRACOMMAND               "SEND_INFRACOMMAND"
    // SERIAL4byjtyCOMMAND2byty
#define SEND_SLAVE_EQUALISER            "SEND_EQUALISER"
    // SERIAL4bytyGLOBAL1byteENALED1byteSIZEOF(Bequaliser)
#define SET_OUTPUTSTREAM                "SET_OUTPUTSTREAM"
#define SET_INPUTSTREAM                 "SET_INPUTSTREAM"
#define SET_GPIO                        "SET_GPIO"
#define SET_TEMPERATURE                 "SET_TEMPERATURE"
#define SET_SLAVE_EQUALISER             "SET_EQUALISER"
    // SERIAL4bytyGLOBAL1byteENALED1byteSIZEOF(bequaliser)

//pouzije slave...

#define GIVE_INPUTSTREAM                  "GIVE_INPUTSTREAM"
#define GIVE_STOPINPUTSTREAM            "GIVE_STOPINPUTSTREAM"
#define GIVE_OUTPUTSTREAM                 "GIVE_OUTPUTSTREAM"
#define GIVE_STOPOUTPUTSTREAM           "GIVE_STOPOUTPUTSTREAM"
```

```
#define SET_SLAVENAME                    "SET_SLAVENAME"
    // ůSERIAL4bytyNAME16byt
#define SET_SLAVEINPUT                   "SET_SLAVEINPUT"
    // SERIAL4bytyUID2byty

#define SEND_SLAVEREFRESH                "SEND_SLAVEREFRESH"
    //SERIAL4bytyCISLOuTebeVPolu4byty

#define EQUALISER_COEF_COUNT             7
#define TEMPERATURE_COUNT                5

typedef enum
{
    NO_SIDE = 6, SIDE_CENTER = 0, SIDE_SUBWOOFER =1,
    LEFT_REAR = 2, LEFT_FRONT = 3, RIGHT_REAR = 4,
    RIGHT_FRONT = 5 , SIDE_ENUM_LAST
} side_enum_t;

typedef char name_t[16];

/*****************************************************
 * input stream
 *****************************************************/
struct BInputStream_t
{
    /// tímhle jenom vynásobit data z A/D
    port_float_t Gain;                   //readwrite
    /// má vzorkovat nebo ne
    port_bool_t  Active;                 //readwrite
    /// fixní nastavíš jenom na začátku programu
    uint16_t UID;                        //readonly
    /// nadřazený balík vstupních ůstream (např CD bude mít číslo 100)
    uint16_t DeviceUID;                  //readwrite
    /// uživatelský název vstupu (např CD left)
    name_t name;                         //readwrite in PC
    /// uživatelský název balíku ůvstup (např CD)
    name_t DeviceName;
    /// side_enum_t
    uint8_t Side;
}  __attribute__ ((__packed__));

typedef struct BInputStream_t BInputStream_t;

/*****************************************************
 * output stream
 *****************************************************/
struct BEqualiser_t
{
    uint8_t Coef[EQUALISER_COEF_COUNT]; //readwrite
    port_bool_t Active;                      //readwrite
}   __attribute__ ((__packed__));

typedef struct BEqualiser_t BEqualiser_t;

/// nastavení výstupu ůmže pro každý slave jinak
/// slave - celý systém pro jednu místnost (např 5+1)
// output stream = reprák
struct BoutputStream_t
{
```

```
    BEqualiser_t Equaliser;            //readwrite

    port_bool_t Mute;                   //readwrite
    port_float_t Volume;                //readwrite
    /// připojený vstup z A/D do konkrétního repráku
    uint16_t InputStreamConnectionUID;  //readwrite
    /// nadřazená jednotka v místnosti (hlavní jednotka - subwoofer)
    uint32_t SlaveSerialNumber;         //readOnly
    /// tohle nastaví PC - nezajímá tě
    uint8_t Side;                       //readOnly
    /// fixní identifikátor reproduktoru (např 100-103)
    uint16_t UID;                       //readonly
} __attribute__ ((__packed__));

typedef struct BoutputStream_t BoutputStream_t;

/*****************************************************
 * Slave
 *****************************************************/
struct BSlave_t
{
    uint32_t IP;                //readonly in PC
    uint32_t SerialNumber;      //readonly
    uint32_t FW_Version;        //readonly
    uint32_t HW_Version;        //readonly
    int16_t  Temperatures[TEMPERATURE_COUNT];    //readonly in PC
    uint16_t GPIO_In;           //readonly in PC
    uint16_t GPIO_Out;          //readwrite
    //30
    name_t   name;                //readwrite in PC
    /// vstupní připojené zařízení
    uint16_t ConnectDeviceUID;

    BEqualiser_t Equaliser;            //readwrite
    //výběr jesli použít lokální nebo globální eq
    port_bool_t useGlobalEq;
    //globálně vypne nebo zapne eqalizéry
    port_bool_t EqualisersEnabled;

    /// seznam výstupních ůstream
    BoutputStream_t ** Outputs; //readonly in PC
    uint16_t OutputCount;       //readonly in PC
} __attribute__ ((__packed__));

typedef struct BSlave_t BSlave_t;

#ifdef QT_CORE_LIB
struct BSlaves_t
{
    uint16_t Count;
    BSlave_t ** Slaves;
} __attribute__ ((__packed__));

typedef struct BSlaves_t BSlaves_t;
#endif

/*****************************************************
 * Master
 *****************************************************/
```

```
struct BMaster_t
{
    uint32_t IP;            //readonly in PC
    uint32_t SerialNumber;  //readonly
    uint32_t FW_Version;    //readonly
    uint32_t HW_Version;    //readonly
}  __attribute__ ((__packed__));

typedef struct BMaster_t BMaster_t;

/*****************************************************
 * Main
 *****************************************************/
typedef struct
{
    BMaster_t * Master;
#ifndef QT_CORE_LIB
    BSlave_t ** Slaves;
    // ještě si sem přihodit počet slave jednotek
    uint8_t SlaveCount;
#else
    BSlaves_t * Slaves;
#endif

    BInputStream_t ** InputStreams;
    uint16_t InputStreamCount;

    BoutputStream_t ** OutputStreams;
    uint16_t OutputStreamCount;

} BMain_t;


#define MEMBER_SIZE(type, member) (sizeof(((type *)0)->member))

/*****************************************************
 * Plnění UDP hlavičky
 *****************************************************/
/*****************************************************
 * 0xFF FF FF F(F/E vstup/vystup) | 2byty INPUT UID |
 * 2byty OUTPUT UID | 4byty slave serial number |
 * 4byty timemarker
 *****************************************************/

#define _UDP_FILL(ptr,data,offset,size) {memcpy(ptr + offset,&data,size);}
#define _UDP_EXTRACT(ptr,retVal,offset,size) {memcpy(&retVal,ptr+offset,size);}
#define TIME_MARKER_SIZE 4

#define STREAM_TYPE_INPUT 1
#define STREAM_TYPE_OUTPUT 0

//pokud se bude posila input stream tak bych chtěl obě uid stejny
//nejde pouzit s konstantou jako argument ale musi to byt promenna
#define UDP_FILL_PREAMBLE(ptr) \
    {memset(ptr,0xff,3); ptr[3] |= 0xFE;}
#define UDP_FILL_STREAM_TYPE(ptr,type0vystup1vstup) \
    {ptr[3] &= 0xFE ;ptr[3] |= (type0vystup1vstup & 1);}
#define UDP_FILL_INPUT_UID(ptr,uid) \
    {_UDP_FILL(ptr, uid, 4, MEMBER_SIZE(BInputStream_t,UID))}
```

```
#define UDP_FILL_OUTPUT_UID(ptr,uid) \
    {_UDP_FILL(ptr ,uid, 6, MEMBER_SIZE(BoutputStream_t,UID))}
#define UDP_FILL_SERIAL_NUMBER(ptr,number) \
    {_UDP_FILL(ptr ,number ,8, MEMBER_SIZE(BSlave_t,SerialNumber))}
#define UDP_FILL_TIME_MARKER(ptr,marker) \
    {_UDP_FILL(ptr, marker,12, TIME_MARKER_SIZE)}

#define UDP_EXTRACT_STREAM_TYPE(ptr,retVal8) {retVal8 = ptr[3] & 0x1;}
#define UDP_EXTRACT_INPUT_UID(ptr,retVal16) \
    {_UDP_EXTRACT(ptr,retVal16,4,MEMBER_SIZE(BInputStream_t,UID))}
#define UDP_EXTRACT_OUTPUT_UID(ptr,retVal16) \
    {_UDP_EXTRACT(ptr,retVal16,6,MEMBER_SIZE(BoutputStream_t,UID))}
#define UDP_EXTRACT_SERIAL_NUMBER(ptr,retVal32) \
    {_UDP_EXTRACT(ptr,retVal32,8,MEMBER_SIZE(BSlave_t,SerialNumber))}
#define UDP_EXTRACT_TIME_MARKER(ptr,retVal32) \
    {_UDP_EXTRACT(ptr,retVal32,12,TIME_MARKER_SIZE)}


#endif // BTYPES_H
```