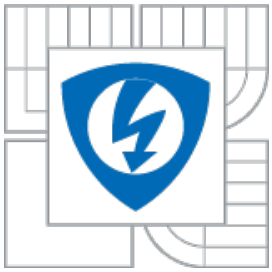
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF RADIO ELECTRONICS

POLYMORFNÍ USB – I²S ROZHRANÍ

POLYMORPHIC USB – I²S INTERFACE

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — BC. MARTIN STEJSKAL
AUTHOR

VEDOUCÍ PRÁCE — ING. MARTIN FRIEDL
SUPERVISOR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student:	Bc. Martin Stejskal	***ID:***	125311
Ročník:	2	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Polymorfní USB – I2S rozhraní

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s 32-bitovými mikrokontroléry firmy ATMEL se synchronním digitálním rozhraním. Prostudujte možnosti USB protokolu a implementaci v jednočipech. Dále navrhněte zařízení s využitím mikrokontroléru a kmitočtového syntezátoru, které se chová jako USB audio zařízení s plně proměnnými parametry (vzorkovací frekvence,rozlišení, formát dat) digitálního výstup ve formátu I2S. Zabezpečte elektrické oddělení vstupně-výstupní části digitálního audia.

Dále navrhněte metodu nastavování parametrů zařízení přes USB, tj. protokol zpráv a jednoduchou řídící aplikaci pro PC. Program pro mikrokontrolér implementujte v programovacím jazyce C s důrazem na plnou kompatibilitu se standardem USB Audio a datovou věrnost výstupu.

DOPORUČENÁ LITERATURA:

[1] AXELSON, J. USB Mass Storage: Designing and Programming Devices and Embedded Hosts (1st ed.). Lakeview Research, 2006.

[2] Universal Serial Bus. USB IMPLEMENTERS FORUM. Universal Serial Bus [online]. 2013 [cit. 2013-05-10]. Dostupné na: http://www.usb.org/

Termín zadání:	10.2.2014	***Termín odevzdání:***	23.5.2014

Vedoucí práce: Ing. Martin Friedl
Konzultanti diplomové práce:

doc. Ing. Tomáš Kratochvíl, Ph.D.
Předseda oborové rady

Abstrakt

Cílem práce bylo seznámit se s 32bitovými MCU od firmy ATMEL, synchronním digitálním rozhraním a implementaci USB protokolu v MCU. Na základě těchto poznatků navrhnout zařízení které se chová jako USB audio s plně proměnnými parametry s digitálním výstupem ve formátu I^2S.

Klíčová slova

AVR32, UC3, USB, I^2S, zpracování dat

Abstrakt

The thesis goal was introduce with 32-bit microcontrollers produced by ATMEL, synchronous digital interface and implementation USB protocol in MCU. On base of this informations design device which have behaviour as USB audio with fully variable parameters and digital output in I^2S format.

Keywords

AVR32, UC3, USB, I^2S, data processing

STEJSKAL, M. Polymorfní USB – I^2S rozhraní. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014. 81 s. Vedoucí diplomové práce: Ing. Martin Friedl.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma Polymorfní USB – I^2S rozhraní 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 nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

Poděkování

Děkuji vedoucímu práce Ing. Martinu Friedlovi a Ing. Josefovi Nevrlému za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne ....

(podpis autora)

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Cílem práce je navrhnout a sestrojit zařízení pro testování a měření systémů používajících sériové sběrnice pro digitální audio (I^2S, PCM). Typickými příklady použití jsou:

- analýza a záznam datového streamu mezi audio kodekem a mikrokontrolérem s I^2S periferií,
- injekce testovacího datového streamu do audio kodeku/digitálního zesilovače,
- simultánní injekce a záznam datového streamu pro test integrovaných obvodů/modulů provádějících audio DSP operace.

V současné době jsou výše zmíněné scénáře plně uskutečnitelné jen pomocí sofistikovaných měřicích systémů, jako jsou audio analyzátory s digitálním modulem, částečně pak s digitálními (MSO) osciloskopy se speciálními dekodéry.

Zařízení má být univerzální, tj. podporující co nejvíce možných nastavení frekvence a formátu dat. Dále má být snadno použitelné s již existujícími PC programy a aplikacemi, tedy chovat se jako běžné USB audio rozhraní, jaké je standardně podporováno všemi běžnými operačními systémy.

U tohoto typu zařízení je vyžadována tzv. bitová věrnost. To znamená, že data na výstupu zařízení musí přesně odpovídat původním zvukovým datům produkovaným audio aplikací běžící na PC. U běžných USB audio rozhraní nastávají problémy v případech, kdy je vzorkovací frekvence záznamu rozdílná s vzorkovací frekvencí převodníku. V takovém případě se běžně aplikuje převzorkování a filtrace signálu. Takovéto transformace dat (změna, odstraňování, přidávání) jsou však z hlediska měření nepřijatelné. Řešením je použití vždy stejné vzorkovací frekvence a dalších parametrů datového audio streamu, jaké jsou nastaveny v použité audio aplikaci na PC. Důležitou částí projektu je proto metoda flexibilního nastavení všech parametrů zařízení přes USB.

Dalším kritickým bodem je schopnost zařízení zabránit „ztrátě“ sebemenšího počtu vzorků kvůli přetečení/podtečení audio bufferu. Vzhledem k charakteru přenosu dat po USB sběrnici je v případě běžných USB audio převodníků podobná ztráta statisticky možná, při malých množstvích vzorků je však různými metodami v digitální i analogové doméně kompenzovatelná. To však opět není akceptovatelné pro případ měřicího zařízení. Problém je řešen adaptivním řízením přenosu audio dat po USB v závislosti na stavu zaplnění bufferu.

Tento dokument je rozdělen do sedmi částí. V následující kapitole je teoretický rozbor I^2S a zjednodušený popis implementace USB protokolu v MCU. V další kapitole je blokové schéma zařízení a popis funkcí jednotlivých komponent. Následuje HW a poté SW řešení celého zařízení. V další kapitole je rozebrán komunikační protokol s PC aplikací. Předposlední kapitola pojednává o naměřených hodnotách. V závěru je pak shrnuta dosavadní práce na tomto projektu a nastínění dalších možností.

1.1. Motivace návrhu

Na trhu je dostupných několik typů zařízení, které by zdánlivě mohly splnit daný účel. Avšak u každého z nich existují nevýhody, které nasazení limitují.

1.1.1. Osciloskop s podporou I^2S

Na trhu jsou běžně dostupné osciloskopy, které umožňují analyzovat I^2S. U většiny osciloskopů je to pouze otázkou volitelné (většinou za příplatek) funkce firmwaru. Jejich výhodou je, že umožňují velice detailně zobrazovat signálové průběhy. Ale už z principu je zde limit v počtu uložených vzorků (ač osciloskopy mívají až několik desítek MB paměti), a tím i oknu, po které je

možné signál analyzovat. Vzorkovací frekvenci 48 kHz s rozlišením 16 bitů odpovídá čistý datový tok 1,536 Mbit/s. Z toho plyne, že lze analyzovat pouze krátké signály. Navíc osciloskopy neumí injektovat I^2S stream, a mohou tak sloužit pouze k analýze. Proto je toto řešení nedostačující.

1.1.2. Audio reference od XMOS

Jako další komerčně dostupný produkt existuje audio reference od firmy XMOS [18]. Zařízení disponuje plně duplexním stereo I^2S rozhraním. Připojení k počítači je možné pomocí USB, takže zpracování dat je ponecháno na libovolné PC aplikaci.

Zařízení podporuje nejpoužívanější bitová rozlišení [17]: 16, 24 a 32 bitů. Podporované vzorkovací frekvence jsou: 44.1, 48, 88.2, 96, 176.4, 192, 358.2 a 384 kHz. Bohužel zařízení nepodporuje nižší vzorkovací frekvence. Konkrétně 8 a 16 kHz, které jsou běžně využívány v digitální telefonii. Jelikož z výše uvedených důvodů není akceptovatelné signál převzorkovat, je toto řešení opět nedostačující.

1.1.3. Audio analyzátor s digitálním modulem

Jedním z méně rozšířených zařízení jsou audio analyzátory. Např. firma Audio Precision nabízí hned několik typů analyzátorů s digitálním modulem [19]. Z hlediska technických parametrů jsou vyhovující (vzorkovací frekvence, bitové rozlišení, bitová věrnost, možnost odposlouchávat a injektovat audio data). Zároveň s hardwarem je dodávána i aplikace pro PC, která umožňuje audio data vyhodnocovat.

Tato zařízení většinou vyžadují instalaci specifického ovladače, takže implementace do jiných aplikací a systémů než těch dodaných výrobcem bývá složitá. Pro množství analytických a dalších funkcí je také obsluha relativně složitá. Největší nevýhodou je však cena pohybující se okolo 20 000 USD za analyzátor s adekvátní konfigurací. Toto řešení je tedy z technického hlediska vyhovující, ale z ekonomického hlediska je problematické (zvláště v případě, kdy je potřeba několik těchto zařízení).

1.1.4. Vývojový kit EVK1105

Společnost Atmel nabízí vývojový kit EVK1105 a příslušný framework se zdrojovými kódy a schématy [10]. Tento vývojový kit obsahuje MCU AT32UC3A0512, který disponuje USB a sériovým digitálním rozhraním (SSC). Při drobné hardwarové modifikaci umožňuje nastavit MCU tak, aby bylo detekováno jako USB zvuková karta, která dokáže přesně reprodukovat data z aplikace běžící na PC. Zpracování dat se opět provádí na straně PC aplikace.

Původní firmware podporoval vzorkovací frekvence 32, 44.1 a 48 kHz se stereo výstupem a mono vstupem. Protože jsou k dispozici zdrojové kódy a schémata, jevila se tato varianta jako vhodná pro modifikaci. Díky tomu, že byl vývojový kit k dispozici, se usnadnilo první testování aplikace. Rozšíření podpory vzorkovacích frekvencí se ukázalo jako možné a funkčnost byla ověřena.

Problém nastal při úpravě vstupu na stereo kanál. Při vzorkovacích frekvencích 44.1 a 48 kHz byl výstupní signál silně narušen. Problém se tedy zprvu jevil jako nedostatek výpočetního výkonu. Až náročná analýza odhalila, že pro přenos audio dat je zapotřebí, aby vyrovnávací USB audio buffery byly zdvojené. Zdvojenost bufferů je nutná u datových přenosů „Isochronous“ (viz. kapitola 2.2). Právě při vzorkovacích frekvencích 44.1 a 48 kHz bylo posíláno již takové množství dat, že hardwarové USB buffery nestačily, takže jejich obsah byl přepsán. To způsobovalo výpadky vzorků, a tudíž degradaci signálu. Kvůli degradaci signálu se toto řešení ukázalo jako nevhodné.

1.1.5. Projekt SDR-Widget

Open source projekt SDR-Widget, který je licencován pod GPL-2.0, si dal za cíl vytvořit zvukovou kartu, která bude v mnohém podobná variantě od společnosti Atmel [13], ale s plnou podporou stereo kanálů v obou směrech. Oba projekty používají 32bitovou architekturu Atmel UC3, jsou bitově věrné, nabízí zdrojové kódy a schémata. Rozdíly jsou především v použitých variantách MCU, dalším použitém hardwaru, datové propustnosti (souvisí s USB periferií a její velikostí RAM), ovládacím softwaru a podpoře USB audio class 2 (viz. níže).

Díky tomu, že bylo použito MCU AT32UC3A3256, bylo možné vyřešit problém stereo kanálů i při vyšších vzorkovacích frekvencích. Tato karta může pracovat v několika módech, přičemž nejuniverzálnější je varianta s podporou USB audio class 1, která je kompatibilní s většinou operačních systémů. V této třídě zařízení (resp. firmware) podporuje pouze dvě vzorkovací frekvence: 44.1 a 48 kHz.

Limitující u USB audio class 1 je ve velikosti paketu odesílaného každou milisekundu [20]. Paket může maximálně obsahovat 1024 Bajtů. Například při vzorkovací frekvenci 176 kHz, bitovém rozlišení 24 bitů stereo vychází velikost paketu 1056 B (176*3*2). Velikost tohoto paketu přesahuje maximum a tuto kombinaci nelze v USB audio class 1 implementovat. Proto byla v roce 2009 [20] vydána nová verze (USB audio class 2), která umožňuje přenést větší objemy dat. Zde je problém s operačními systémy Windows. Nativně standard nepodporují ani nejnovější Windows 8.1 [21]. Projekt SDR-Widget podporuje oba standardy.

Vzhledem k zaměření projektu byla preferována univerzálnost USB audio class 1, i za cenu chybějící podpory dalších vzorkovacích frekvencí. Cenová bilance na výrobu jednoho kusu (DPS, součástky a krabička) se pohybuje okolo 220 USD. Při výrobě deseti kusů je cena jednoho kusu pouze 150 USD. Tento projekt byl proto zvolen jako základ, na kterém je vystavěna tato diplomová práce.

2. Teoretický rozbor

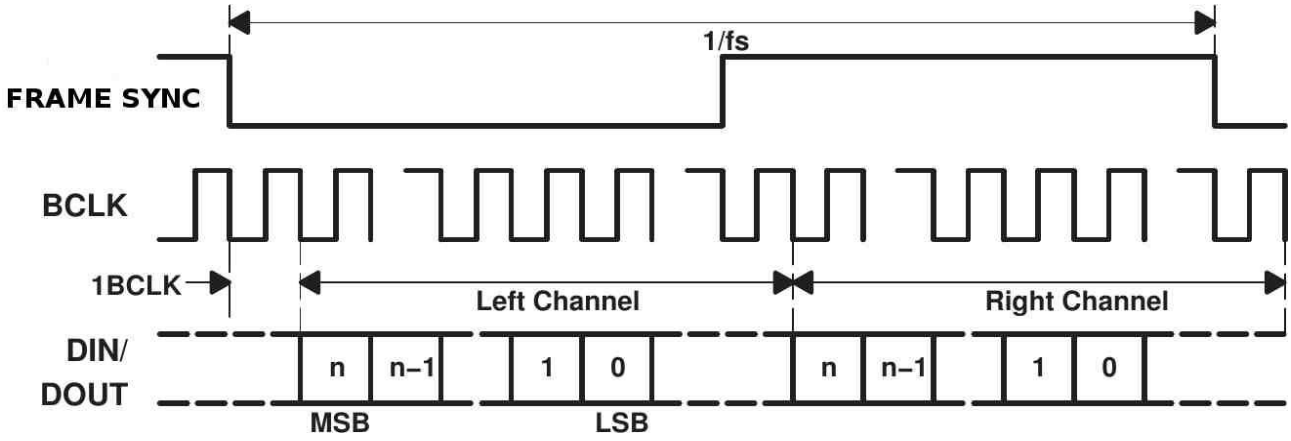
Jak bylo uvedeno v úvodu, pro co nejvěrnější podání nahrávky je potřeba, aby zařízení mělo proměnné parametry jak časové základny (vzorkovací frekvence), tak bitového rozlišení (délka slova, formát). To však naráží na problém s různým časováním různých periferií. Například pro funkci USB je potřeba takt hodin 48 MHz [1], ale přitom potřebujeme, aby I^2S rozhraní běželo na taktu 11,2896 MHz, 12,288 MHz, 8,192 MHz apod. To jsou neceločíselné násobky od 48 MHz, takže obyčejná dělička zde nestačí. Právě MCU UC3A umožňují dvojí hodiny [1], čímž se částečně řeší problém s dvojím taktováním. Takt pro USB je odvozen z 12 MHz krystalu a takt pro I^2S rozhraní je řízen PLL, které je nastavováno přes I^2C sběrnici.

2.1. Protokol I^2S

Přenášet zvuk v digitální formě po synchronní sběrnici jde několika způsoby. V průběhu času se rozšířily především následující standardy [2]:

- Right justified (zarovnání bitů vpravo),
- Left justified (zarovnání bitů vlevo),
- I^2S,
- DSP.

Principiálně jsou si jednotlivé možnosti velice podobné, avšak liší se v detailech. Zařízení komunikující s počítačem často využívají právě I^2S. Datový tok výstupních dat je synchronní se vstupním tokem dat (digitalizovaného zvuku). Časový diagram pro I^2S je na Obr. 1.



Obr. 1: Časový diagram I^2S protokolu (převzato z [2])

Popis jednotlivých signálů:

- FRAME SYNC – synchronizace rámce. Také určuje, zda jsou data určena pro levý, nebo pravý kanál
- BCLK – bit clock. Pro synchronizaci jednotlivých datových bitů. Takt bit clocku se dá vypočítat následovně:

$$f_{BCLK} = f_{SAMPLE} \cdot (sample\ precision) \tag{1.1}$$

kde f_{SAMPLE} je vzorkovací frekvence, *sample precision* je typicky bitové rozlišení datového slova. Sample precision udává počet bitů (datové + nuly) v jedné periodě signálu FRAME SYNC.

- DIN/DOUT – vstupní/výstupní data (bráno z pohledu tohoto zařízení) – zvuk reprezentovaný ve vzorcích.

Teoreticky by tyto 4 signály stačily na plně duplexní stereo přenos. V praxi je ale často přidáván signál MCLK (master clock), který je určen pro kodek nebo další I^2S zařízení. Bývá násobkem BCLK, v praxi jsou nejběžnější násobky 32, 64, 128 a 256 BCLK. Většina kodeků má již integrovanou děličku, takže při změně vzorkovací frekvence není potřeba měnit MCLK. Většinou stačí jen vhodně nastavit kodek. Bývá zvykem, že BCLK, FRAME SYNC a DOUT jsou pro master výstupní signály a DIN je vstupní (z pohledu master). Pokud bude zařízení ve slave módu, pak z jeho pohledu budou BCLK, FRAME SYNC a DIN brány jako vstupní. Naopak DOUT bude brán jako výstupní.

2.2. Základy USB protokolu a jeho implementace v MCU

Protože USB protokol je komplexní a na implementaci náročný, některé mikrokontroléry už jej částečně podporují na hardwarové úrovni. Vzhledem k možnostem USB je nutné nejprve detailně nastavit USB rozhraní a pak provést inicializaci samotné USB sběrnice.

USB zařízení se může chovat jako *USB Device*, *USB Host* nebo jako *On-The-Go* (OTG) [3]. Mód *USB Device* je pro zařízení, která se připojují do USB sběrnice. Tedy například flash disk, myš, klávesnice apod. Mód *USB Host* je přesný opak. Jak už název napovídá, jedná se o zařízení, které „má na starost“ jednotlivé *USB Device* a řídí tok na sběrnici. Z pohledu uživatele se jako „USB Host“ jeví počítač. Poslední mód, „On-The-Go“, je kombinace předchozích dvou. Zařízení se může chovat jako „USB device“, ale i jako „USB Host“. Toho je využíváno například u tiskáren. Pokud je tiskárna připojena k počítači, pak vyžadujeme, aby se tvářila jako „USB device“, ale v případě, že potřebujeme vytisknout nějaký obrázek nebo dokument, například z flash paměti, pak se musí chovat jako „USB Host“. Právě mikrokontroléry UC3 podporují mód „On-The-Go“, takže z hardwarového hlediska není problém používat mikrokontrolér v obou módech. Komponenta zodpovědná za komunikaci v módu „USB Host“ se nazývá host controller, zkráceně host. V textu je použit kratší název: host.[1].

Protokol USB umožňuje komunikovat v několika módech:

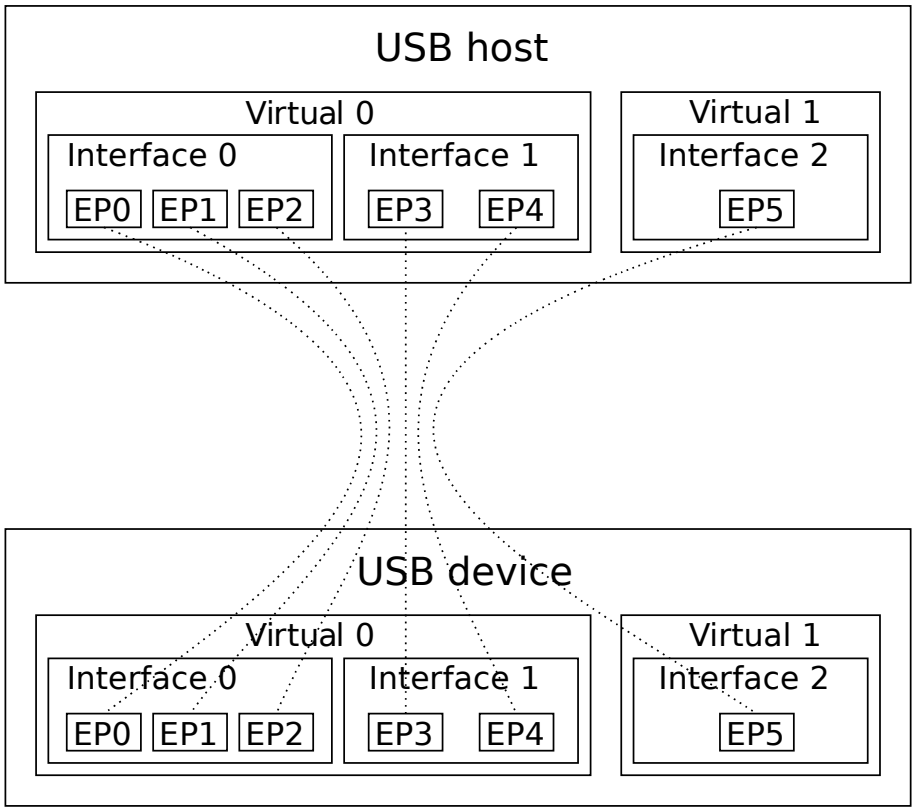
- Control,
- Interrupt,
- Bulk,
- Isochronous.

Mód „Control“ se používá především pro inicializaci. Nic ale nebrání tomu, aby zařízení komunikovalo právě přes tento mód. Velmi jednoduché USB zařízení využívají tento mód. Pro nenáročnou komunikaci se používá „Interrupt“ mód. Tento mód je vhodný pro aplikace, kdy vyžadujeme periodicky posílat malé množství dat. V tomto módu fungují například myši a klávesnice. Pokud je vyžadováno přenášet velké množství dat a kontrolovat při tom, zda všechna data byla úspěšně přenesena, pak je zde „Bulk“ mód. V tomto módu pracují typicky flash disky. Pro případy, kdy je potřeba přenášet data synchronně, ale nevadí větší BER (bitová chybovost), je zde mód „Isochronus“. Ten je často používán pro audio aplikace. Vzhledem k tomu, že u tohoto zařízení je vyžadováno, aby vzorky byly přijaty všechny a spolehlivě, byl by vhodnější „Bulk“ mód. Protože je však požadováno, aby zařízení bylo použitelné na většině operačních systémů bez nutnosti instalaci ovladačů, je nutné používat pro přenos audio dat mód „Isochronus“. Níže bude podrobněji vysvětleno proč.

1 Přestože je termín „host“ anglický a v českém textu může mást, ustálený překlad tohoto pojmu v rámci USB standardu neexistuje.

Pro prvotní komunikaci s USB zařízením je použit mód „Control“. V tomto módu si host obvykle zažádá zařízení o tzv. „Device Descriptor“. Zde je uložena informace VID (Vendor ID) a PID (Product ID). Díky těmto ID je OS schopný nahrát správný driver pro zařízení v případě, že zařízení nepoužívá jeden ze standardních tříd zařízení. Jako další jsou zde „Configuration Descriptors“. Těch může být několik. Každý pak popisuje jednu konfiguraci zařízení. V každé konfiguraci může zařízení vykonávat stejné funkce, ale může se lišit v drobných detailech (např. použitý mód, velikost přenášených dat atd.). V praxi ale zařízení mívá jen jednu konfiguraci, která se nemění. Každá konfigurace obsahuje jeden nebo více interface. Každý interface popisuje, v jakém módu se data budou přenášet, zda se budou data přenášet v některém z předdefinovaných standardů (HID, audio, …), kam se v paměti budou ukládat (do jakého endpointu), jaké množství dat se bude přenášet atd.

Jako příklad si můžeme představit multifunkční tiskárnu, která obsahuje i scanner. Fyzicky se jedná o jedno zařízení s jedním USB portem, ale uživatel v OS vidí dvě zařízení. Tato jednotlivá zařízení OS rozpozná podle deskriptorů interface. Každé toto virtuální zařízení může obsahovat jedno, nebo více interface. Interface určuje svoje endpointy. Do těchto endpointů jsou ukládána surová data, případně se z těchto endpointů načítají data, která mají být odeslána. Endpointy existují jak na straně device (zařízení), tak na straně hosta (počítače). Jednotlivé endpointy jsou „propojeny“ pomocí tzv. „pipes“ (roury), což je abstrakce pro datový kanál. Zjednodušené obecné schéma je na Obr. 2.



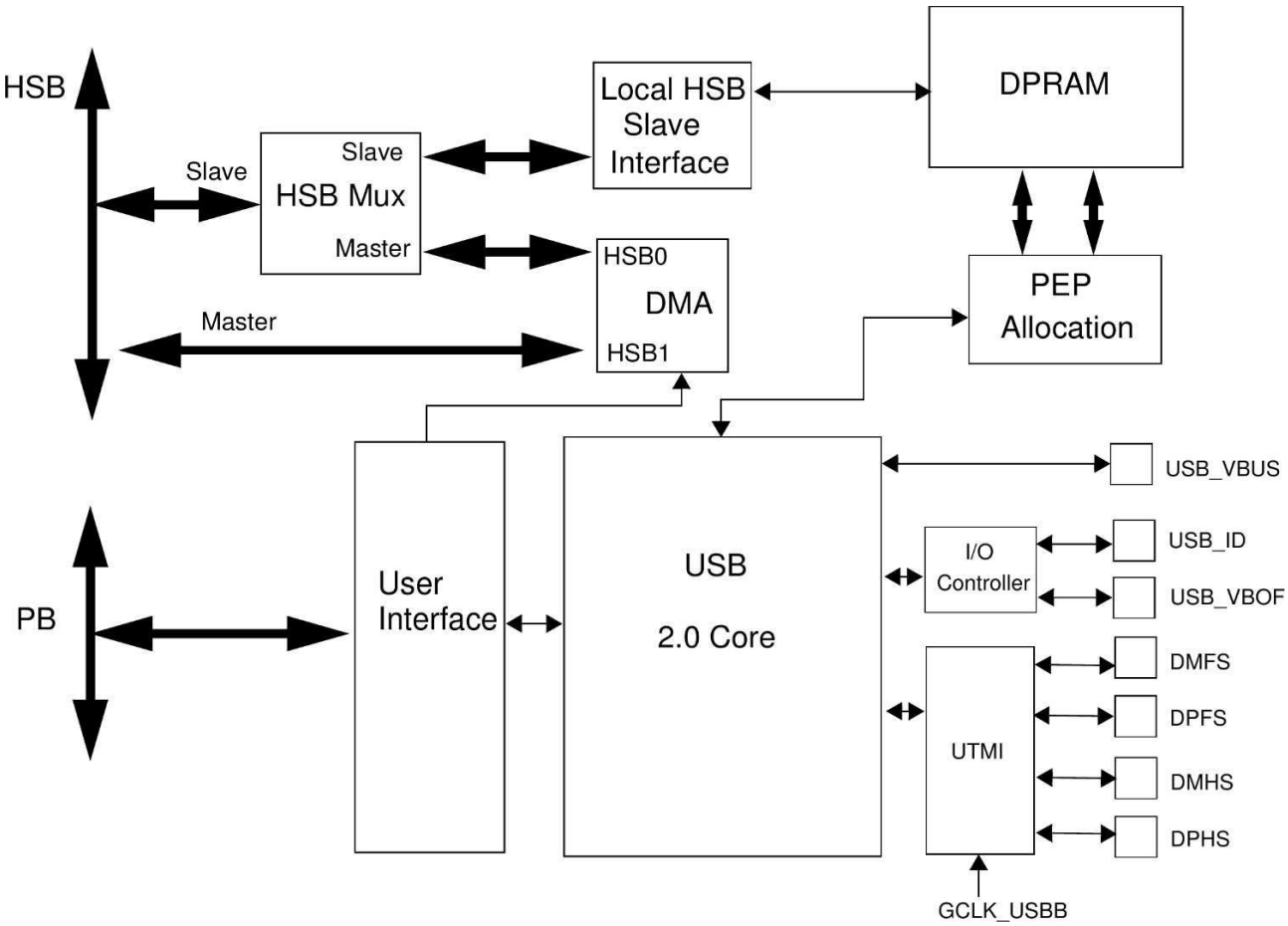
Obr. 2: Zjednodušené schéma USB protokolu

Aby uživatel nemusel pro každé USB zařízení instalovat specifický ovladač (typu myš, klávesnice), byly v USB standardu definovány třídy zařízení („device classes“), které definují, jaké má mít interface nastavení, aby se k němu přistupovalo podle předdefinovaných konfigurací. Do těchto tříd spadají klávesnice, myši, flash disky, síťové karty, bluetooth moduly, zvukové karty, HID, tiskárny apod.

Směr všech USB dat je definován z pohledu hosta. To znamená, že pokud data proudí směrem

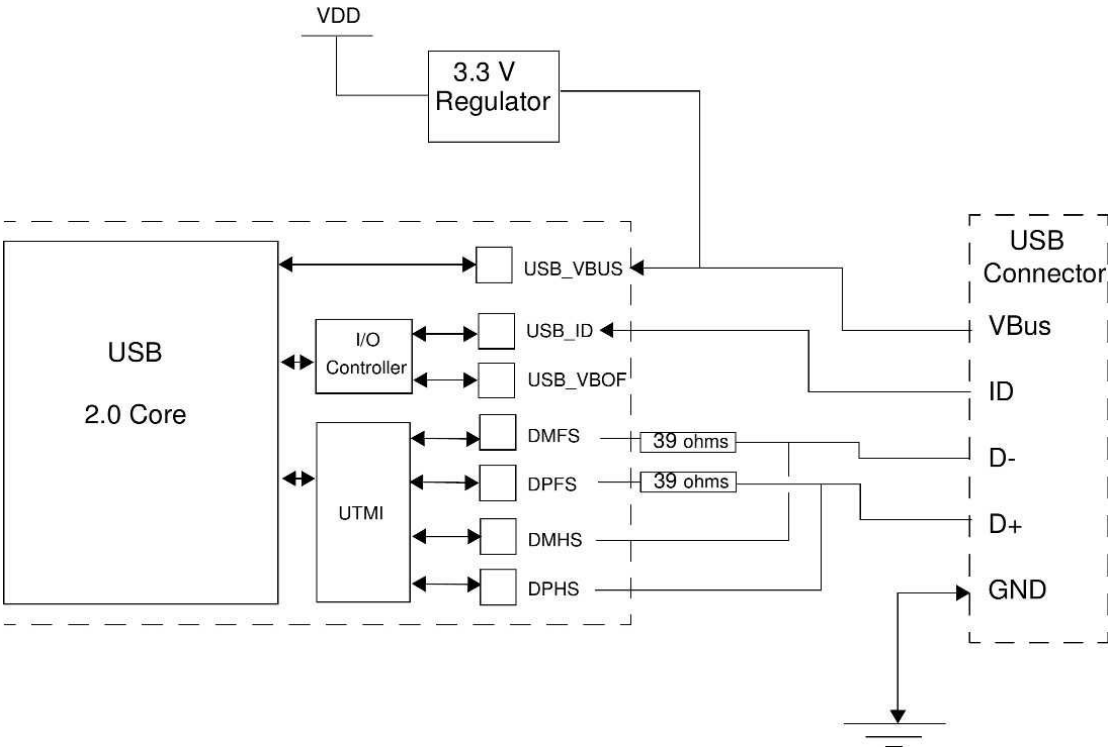
od zařízení k hostu, vždy jsou označena jako vstupní, a to i na straně zařízení! A naopak. Pokud proudí data od hosta směrem k zařízení, jsou vždy označena jako výstupní.

Pro správnou funkci vyžaduje USB modul na UC3A3256 12 MHz zdroj hodin. Ten může být generován pomocí interních PLL (fázových závěsů) a například krystalu (viz. kapitola Power Management - PM). Tento takt je pak použit pro generování 480 MHz (v případě high-speed) bit clocku. Pro obnovu je použit DPLL (není na blokovém schématu). Blokové schéma je na Obr. 3.



Obr. 3: Blokové schéma USB rozhraní (převzato z [1])

Pro napájení USB modulu je zapotřebí pin USB_VBUS připojit na +5V. Pin USB_ID slouží pro identifikaci konektoru, resp. je možné detekovat, jestli se jedná o konektor pro „USB host“, nebo „USB device“. Vzhledem k tomu, že zařízení bude pracovat pouze jako „USB device“, nebude tato funkce rozebírána podrobněji. Diferenční páry pro USB jsou rozděleny na „full-speed“ a „high-speed“, ale podle doporučeného zapojení by měly být propojeny přes externí rezistory. Doporučené zapojení pro USB modul je na Obr. 4.



Obr. 4: Doporučené zapojení USB modulu (převzato z [1])

Celý modul disponuje velkým množstvím možných přerušení a to na několika úrovních. Počínaje přerušením na nejnižší úrovni (zařízení připojeno, odpojeno, počátek rámce, atd.) přes přerušení od pipe (data přijata, data odeslána) až po přerušení od DMA (přímý přístup do paměti).

Dále je na čipu integrovaný obvod „plug-in detection“. Tento obvod sleduje napájecí napětí a napětí na diferenčním páru. Pokud jsou splněny podmínky, pak obvod generuje signál, který informuje vyšší vrstvy o tom, že je USB sběrnice připojena.

Hardwarově je modul připraven až pro 8 endpointů, včetně „control“ endpointu. Pro ukládání zpracovaných dat je zde připravena DPRAM (dvouportová RAM), která umožňuje uložit až 2368 B. Teoreticky však lze nakonfigurovat tak, aby celkový datový tok byl posílán po 3648 B. V takovém případě by obsah DPRAM byl přepsán. Proto je nutné napsat program tak, aby velikost endpointů (EP) nepřekročila danou mez. Ovšem ne každý EP může být konfigurovaný libovolně. Zjednodušený popis parametrů a omezení je v tabulce Tab.1.

Tab. 1: Možnosti nastavení endpointů (převzato z [1])

Endpoint	Podpora DMA	Podporované typy přenosu	Maximální velikost EP [B]
0	Ne	Control	64
1	Ano	Isochronus/Bulk/Interrupt/Control	512
2	Ano	Isochronus/Bulk/Interrupt/Control	512
3	Ano	Isochronus/Bulk/Interrupt	512
4	Ano	Isochronus/Bulk/Interrupt/Control	512
5	Ano	Isochronus/Bulk/Interrupt/Control	512
6	Ano	Isochronus/Bulk/Interrupt/Control	512
7	Ano	Isochronus/Bulk/Interrupt/Control	512

To, jestli bude USB rozhraní pracovat v tzv. „low-speed“, „full-speed“ nebo „high-speed“ módu, rozhoduje pull-up rezistor připojený na diferenční sběrnici. Tyto módy v podstatě určují maximální možnou přenosovou rychlost dat. Pokud bude pull-up připojen na D-, pak bude zařízení pracovat v „low-speed“ módu. Jestliže bude pull-up připojen na D+, pak bude zařízení pracovat v „full-speed“ módu. Pokud má zařízení pracovat jako „high-speed“, musí se nejprve inicializovat stejně jako „full-speed“ zařízení a poté přepne na vyšší přenosovou rychlost [12]. Tyto pull-up rezistory jsou již integrovány na čipu, takže pro jejich aktivaci stačí jen nastavit dané registry a není nutné osazovat DPS dalšími součástkami.

Jak již bylo naznačeno výše, existuje třída USB audio class 1, která definuje, jaký ovladač má být na straně PC použit. Výhodou tohoto standardu je, že je podporován většinou OS. Pokud bude použita tato třída, odpadá tím řešení ovladače na straně PC. Na druhou stranu odesílání a přijímání audio dat ve třídě USB audio je omezeno na mód „Isochronous“. Ten způsobí, že audio data nebudou zabezpečena (host se nestará o to, zda data byla úspěšně přijata). Výhodnější by tedy bylo použít mód „Bulk“. To by ale obnášelo vytvoření ovladačů pro PC. Vzhledem k časovým možnostem tak byla zvolena varianta, ve které je použita třída USB audio. Samotná třída USB audio class 1 pak nabízí tři možnosti nastavení, které definují časování mezi hostem a zařízením. Od toho se pak odvíjí hardwarové zapojení a možná řešení synchronizace. Stručný přehled poskytuje Tab. 2.

Tab. 2: Přehled možností časování mezi hostem a zařízením (převzato z [22])

Metoda	Popis
Synchonous	Periodu udává host. Zařízení musí být schopno zpracovávat data na periodě udávané hostem. Tato metoda není pro měřící účely příliš vhodná, protože počítače nemají příliš přesný a stabilní zdroj hodin.
Asynchonous	Periodu udává zařízení. Host přizpůsobuje množství dat podle zpětné vazby od zařízení. Ideální pro měřicí a Hi-Fi zařízení. Obecně mají zařízení mnohem stabilnější a přesnější zdroj hodin. Tato metoda je použita v navrhovaném zařízení.
Adaptive	Perioda je dána datovým tokem. Tato metoda není vhodná kvůli nestabilním a nepřesným zdrojům zdrojům hodin v počítačích.

2.3. Řešení synchronizace audio streamu

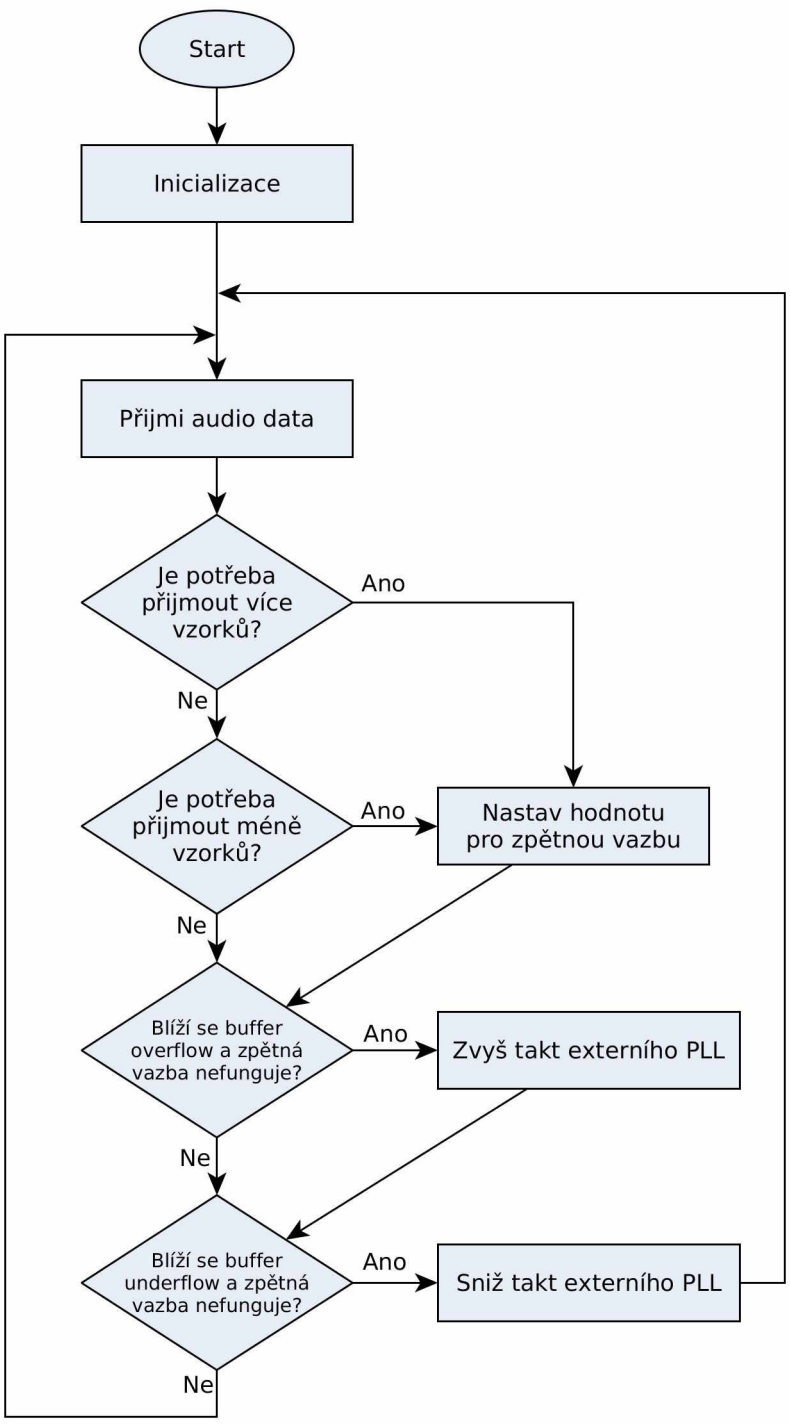
Hodiny pro USB a SSC (sériové synchronní rozhraní) lze považovat za nezávislé už jen kvůli rozdílným časovým základnám. Lze tedy očekávat mezi jednotlivými časovými základnami drift. Ten musí být kompenzován, jinak se začne buffer pro audio vzorky buďto zaplňovat, nebo vyprazdňovat rychleji, než je potřeba. Aby nedošlo k přetečení nebo podtečení, je potřeba tento drift korigovat. V USB standardu jsou definovány 4 možnosti [10]. Shrnutí jejich výhod a nevýhod je v Tab. 3.

Tab. 3: Přehled synchronizačních metod v třídě USB audio (převzato z [10])

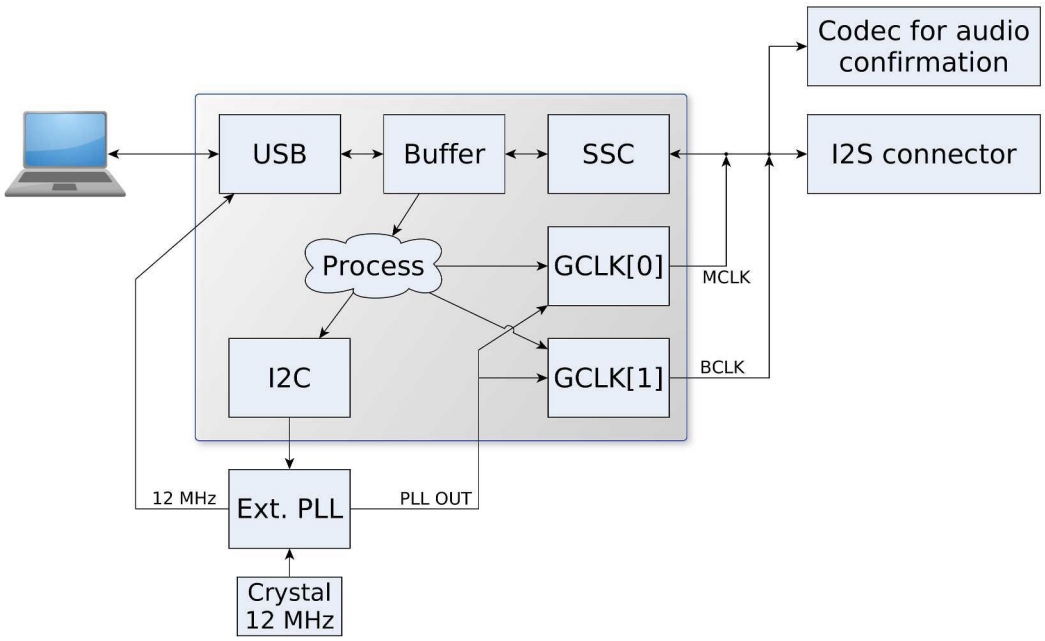
Synchronizační metoda	Výhody	Nevýhody
A) Vkládání/odstraňování vzorků	Velice jednoduché Minimální nároky na HW	Zkreslení signálu v závislosti na rozdílech vzorkovacích frekvencí
B) Použití přesné PLL	Jednoduchá implementace Minimální nároky na HW Bitově přesná kopie 1:1	Nutnost externí přesné PLL Absolutní hodnota vzorkovací frekvence se mírně mění (v řádech 10 ppm)
C) Adaptivní převzorkování	Menší zkreslení než řešení A	Zkreslení signálu kvůli převzorkování Náročné na HW
D) Použití zpětnovazebního EP	Jednoduchá implementace Bitově přesná kopie 1:1	Některé starší OS jej nemusí nativně podporovat (např. Windows XP)

U metody B je perioda dána hostem. Metoda B sice nabízí bitově věrný přenos, ale jak bylo vysvětleno v předchozí podkapitole, časování podle hosta není vhodné pro měřící účely. V důsledku přelaďování PLL se absolutní hodnota vzorkovací frekvence mírně mění. Zdroj hodin se pak pro ostatní zařízení jeví jako časově nestabilní (změny v desítkách ppm). Proto je o něco vhodnější metoda D. U metody D udává periodu zařízení, což je mnohem vhodnější. Při přehrávání program sleduje zaplnění výstupních bufferů a podle aktuálního vytížení pošle zpětnovazebním EP informaci o tom, kolik má host (PC) přenést dat. Naopak pokud zařízení odesílá audio data hostu, tak rozhoduje kolik dat pošle v závislosti na tom, kolik dat má k dispozici (jak je zaplněn vstupní buffer). Díky tomu není potřeba měnit frekvenci externího PLL (od které je odvozen MCLK, BCLK a FRAME_SYNC) a výstupní vzorkovací frekvence bude stabilní jako její zdroj.

Protože metoda D nemusí být podporována ve starších OS, zařízení využívá kombinaci metod D a B. Při inicializaci nahlásí OS používanou metodu D. Pokud je vše v pořádku, pak se tato metoda používá po celou dobu komunikace s hostem. V momentě, kdy metoda D není podporována, výstupní buffery se začnou plnit, nebo podtékat. V tento moment program přenastaví externí PLL podle aktuálních potřeb tak, aby nedošlo k podtečení nebo přetečení bufferu a zároveň aby změna byla co nejmenší. Host o této změně není a nemusí být informován a přesto nedojde (za normálních podmínek) k přetečení nebo podtečení bufferů. Protože digitální vstupní a výstupní audio moduly (SSC) jsou synchronní, neměly by přetéct/podtéct ani vstupní buffery. Zjednodušený vývojový diagram na Obr. 5 popisuje, jak funguje algoritmus.



Obr. 5: Použitý algoritmus pro synchronizaci audio streamu



Obr. 6: Blokové schéma s externím PLL

Do návrhu byl navíc přidán kodek pro kontrolní odposlech I^2S sběrnice. Zjednodušené blokové schéma je na Obr. 6. Data z počítače proudípřes USB rozhraní. Jednotlivé vzorky jsou ukládány do RAM. Díky PDCA je možné nakonfigurovat SSC tak, že data z bufferu budou čtena do SSC bez nutnosti použití CPU. To sice musí průběžně nastavovat PDCA periferii, ale celková režie tohoto procesu je mnohem menší, než kdyby mělo CPU kopírovat data do SSC vzorek po vzorku. Data z SSC pak proudí do konektoru a kodeku pro kontrolní poslech. Mezitím CPU jednou za 1 ms zkontroluje stav bufferu a podle algoritmu na Obr. 5 provede potřebné operace. Kvůli jistým omezením externí PLL nemůže generovat hodiny o nižším taktu než 6 MHz, takže hodiny z externího PLL jsou přivedeny do interních děličeк GCLK[0] a GCLK[1] (součást PM), kde můžou být podle potřeby poděleny celým číslem. To umožňuje generovat MCLK a BCLK přímo. V takovéto konfiguraci modul SSC pro svůj běh nepotřebuje přímo MCLK, ale stačí mu jen BCLK jako zdroj referenčních hodin. Tím se dá dosáhnout vysoké flexibility při nastavování parametrů.

3. Popis použitého mikrokontroléru

3.1. Výběr mikrokontroléru

Jak už bylo naznačeno v úvodu, původní návrh vycházel z řešení přímo od společnosti Atmel. Ten používal MCU AT32UC3A0512. Kvůli hardwarovým omezením bylo nutné použít jiné MCU. Jelikož projekt SDR-Widget využívá MCU, které je už po hardwarové stránce dostačující, bylo pro další verzi použito MCU AT32UC3A3256.

3.2. Základní parametry

UC3A3256 obsahuje 256 kB flash a 64 kB SRAM. Tato velikost je pro požadavky projektu dostatečná. Celé MCU je 32bitové, což je ideální pro přenos audio dat (šířka slova audio vzorku bývá od 16 B do 32 B, ale lze se setkat s 8bitovým rozlišením). Účinnost vykonávání instrukcí je 1,51 DMIPS / MHz [1]. Maximální takt je 84 MHz. Dále obsahuje množství dalších funkčních bloků, které umožní využít hardware na čipu s minimálními nároky na CPU. Dále obsahuje ADC, který je využit pro nastavení hlasitosti kodeku a sledování napětí na straně konektoru s I^2S signály. Užitečné jsou i I^2C a SPI moduly, které usnadňují implementaci ovladačů pro další komponenty, které jsou externě připojeny k MCU.

3.3. Další funkční bloky na čipu

Softwarová emulace komponent je jednak pomalá a jednak značně zatěžuje CPU. Proto výrobci často implementují na čipu i další hardwarové komponenty, které se pak jen řídí zapsáním správné hodnoty do daného registru. Příkladem může být PWM. Pokud ji chceme implementovat softwarově, bude nutné v určitých časových intervalech přecházet do přerušení, kde se bude nastavovat logická úroveň na výstupu. Pokud ale použijeme již implementovaný PWM modul, pak stačí jednou nahrát požadovanou hodnotu do registru a dále již není nutné generování PWM řídit programem. Nejen že uspoříme výpočetní výkon, ale i program může být jednodušší a menší. Navíc většina komponent umožňuje skočit na vektor přerušení při určité události (například pokud jsou všechna data přenesena, nebo pokud nastane chyba během přenosu atd.). Na druhou stranu kód již není 100% přenositelný, protože je více vázán na hardware, který daný čip nabízí. Tento problém se dá částečně řešit abstrakcí hardware v kódu a využíváním preprocesoru v jazyce C. To ale přináší další negativní jev: kód programu je mnohem větší a především přestává být přehledný a dobře čitelný. Proto je důležité zvolit dobrý kompromis mezi přenositelností kódu a přehledností. Zároveň je vhodné oddělit jednotlivé vrstvy tak, aby nebyly vázány na hardware.

UC3 nabízí spoustu komponent, ale ne všechny bude aplikace vyžadovat. Proto v následujících odstavcích budou popsány pouze periferie, které budou nebo by mohly být v budoucnu použity. USB periferie již zde není uváděna, protože je podrobně probrána v předchozí kapitole.

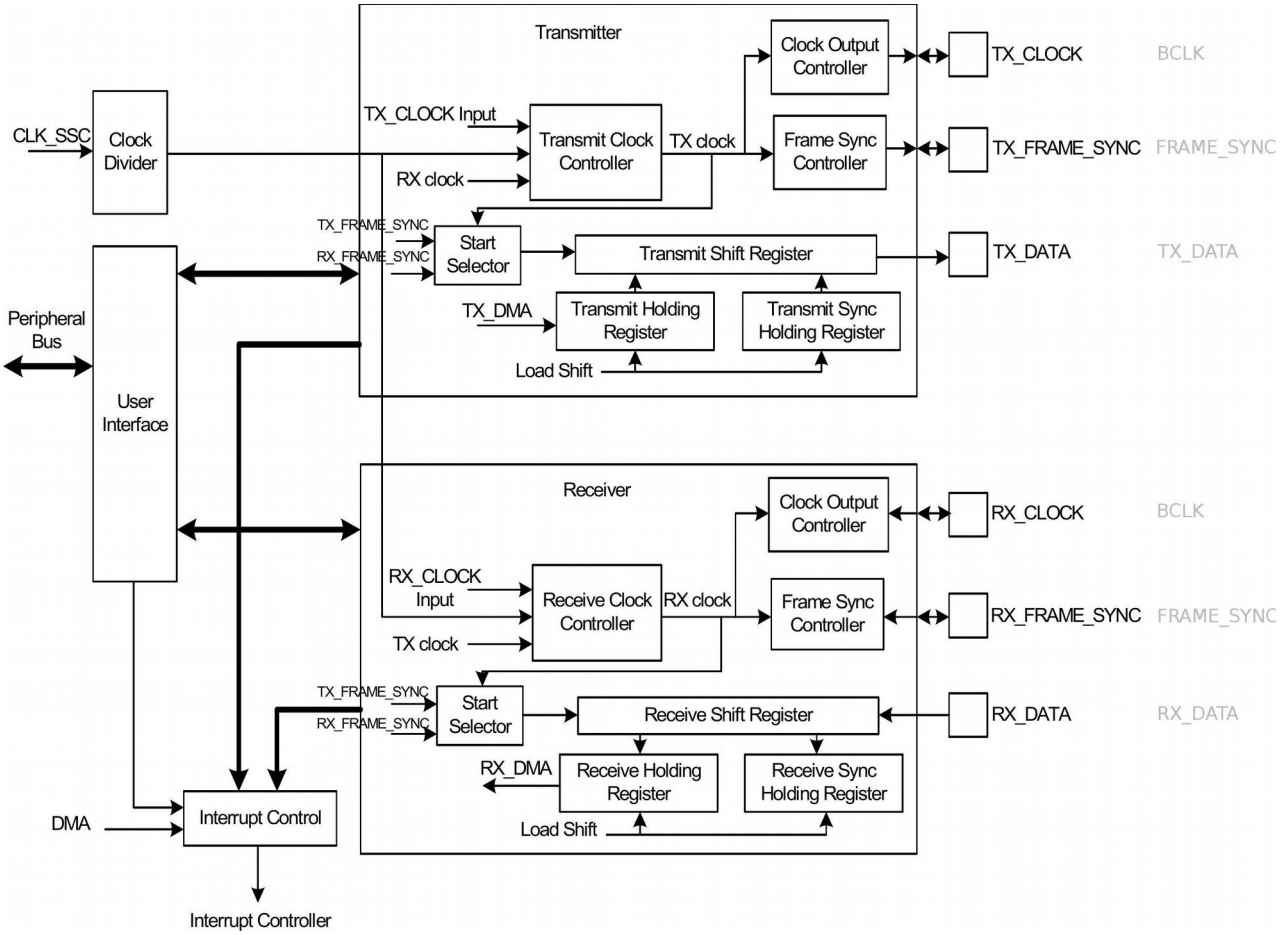
Poznámka: U většiny blokových schémat je blok, který je nazván PM. Jedná se o Power Manager, který je zodpovědný za hodiny směřující do daného modulu. Zároveň je zodpovědný za správné resetování MCU, uspání MCU a nastavení zdrojů hodin.

3.3.1. Synchronous Serial (SSC)

Jednotka, která zpracovává tok digitálního audia ve formátu I^2S. Zařízení je propojené s PDCA, takže výpočetní náročnost na zpracování dat klesne (viz. kapitola 3.3.2). SSC může pracovat jako master, ale i jako slave, což z něj dělá z pohledu I^2S protokolu univerzální zařízení.

Modul má nezávislé RX a TX jednotky. Díky tomu je možné, aby každá pracovala na úplně

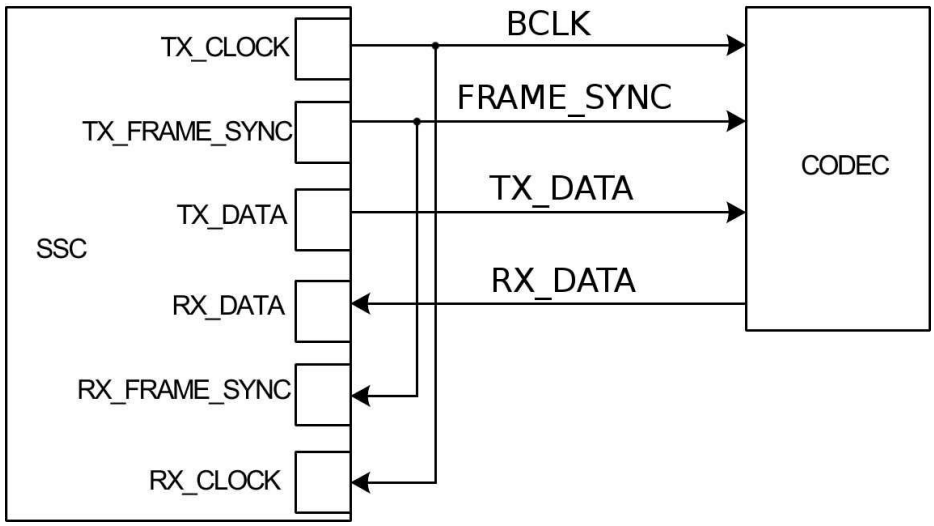
odlišném taktu a s úplně jiným nastavením. Na druhou stranu by se tím celkové zapojení velice zkomplikovalo, takže v aktuální verzi pracují obě jednotky synchronně. Blokové schéma SSC jednotky je na Obr. 7. Šedě jsou popsány piny korespondující s I^2S protokolem.



Obr. 7: Blokové schéma SSC jednotky (převzato z [1])

V této konfiguraci může být zdroj hodin pro každou jednotku pin TX/RX_CLOCK (BCLK) nebo interní hodiny, které mohou být děleny 12bitovou předděličkou. To samé platí pro TX/RX_FRAME_SYNC (FRAME_SYNC). Buď může být použit jako zdroj signálu, nebo může na tento pin signál generovat. Piny TX/RX_CLOCK a TX/RX_FRAME_SYNC jsou záměrně propojeny. V momentě, kdy je RX jednotka nastavena na čistě jako vstupní, změny provedené na TX jednotce se úplně stejně projeví i na RX jednotce. To výrazně zjednodušuje rekonfiguraci na straně driveru. Modul umí mimo jiné „loop mode“, což znamená, že přijímaná data jsou okamžitě vysílána zpět. To je užitečné například při ladění aplikace. Výsledné zapojení je na Obr. 8.

Pokud bude přenos audio dat bez problémů (podtečení, přetečení bufferů, výpadek synchronizace atd.), pak kvalita analogového signálu závisí pouze na použitém kodeku, který může být až na úplném konci řetězce (i fyzicky). Periferie má opět konfigurační registry, díky kterým je možné jemně nastavit potřebné parametry.



Obr. 8: SSC v zapojení pro kodek (převzato z [1])

3.3.2. Peripheral DMA Controller (PDCA)

Pro komunikaci s hardwarem je zde PDCA (Peripheral DMA Controller). PDCA umožňuje, aby data z hardwaru byla nahrávána na předem definovaná místa v SRAM. Tím jednak můžeme odlehčit CPU a jednak se tím vyhneme přeskupování dat z jednoho bufferu do druhého. V podstatě je to jen pole ukazatelů, které ukazují na proměnné, do kterých budou data přijatá hardwarem zapsána, případně čtena. Aby se minimalizovalo zpoždění mezi vysíláním/přijímáním dat z periferií, jsou zde tzv. reload ukazatele. Program tedy definuje dva ukazatele. Poté co např. hardware odešle data, načte následující data z reload ukazatele. Mezitím může být spuštěna nějaká rutina v přerušení, která nastaví nové ukazatele. Samozřejmě, že takto je možné přistupovat hned k několika hardwarovým modulům. Pro každý modul je pak možné nastavit individuálně konfigurační registry.

3.3.3. Periferní sběrnice

Protože ověřit I^2S je pomocí klasických měřicích přístrojů vcelku složité, je efektivnější přidat na DPS kodek, který je paralelně připojen na I^2S sběrnici. Po správném nastavení kodeku dostaneme na výstupu analogovou formu signálu, která se dá jednoduše změřit, případně si lze signál přímo poslechnout. Lidský sluch je velice citlivý na výpadky a degradaci signálu (vlivem přetečení/podtečení bufferů). Poslechem se tak dá provádět velmi efektivní analýza. Pokud do cesty přidáme přepínač, bude možné poslouchat buď DOUT, nebo DIN a ne pouze jeden směr. Právě pro nastavení kodeku je využívána I^2C (nazývaná též TWI) sběrnice.

Nejen kodek, ale i PLL se nastavuje přes I^2C sběrnici. Díky vlastnostem I^2C lze na jednu sběrnici připojit více zařízení, což usnadňuje celkový návrh. Právě externí PLL je potřeba pro generování dvojích hodin pro MCU a jemné donastavení taktu pro SSC modul. Toto zařízení využívá I^2C periferii v master režimu přímo pro ovládání kodeku a externí PLL.

Další, velice rozšířeným rozhraním, je SPI. Často toto rozhraní využívají ADC, DAC, ale i LCD. A právě podpora LCD je v tomto případě klíčová. Celé zařízení totiž obsahuje jednoduchý display, který v konečné verzi informuje uživatele o stavu zařízení. Zároveň během vývoje je možné posílat si na display ladící informace. Modul SPI pracuje také v master módu.

3.3.4. General-Purpose Input/Output Controller (GPIO)

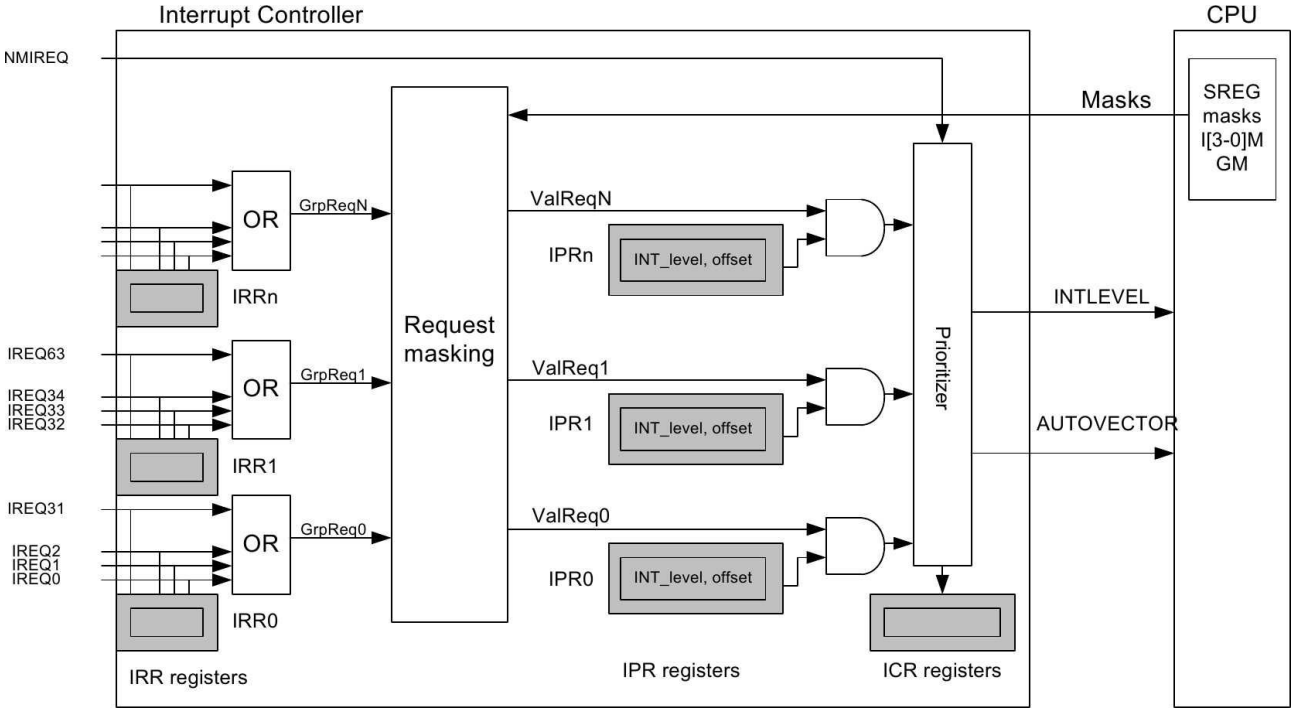
Tato komponenta má na starost veškerou správu vstupně/výstupních pinů. Na rozdíl od 8bitových MCU, kde po povolení dané komponenty hardware sám přepnul funkci pinu, UC3 jsou

více univerzálnější. Například nastavení určitého přerušení může být na 1 z N vstupně/výstupním pinu. Dále je možné, aby dvě různé periferie přistupovaly ke stejným pinům (piny se samozřejmě musí multiplexovat), nastavit na vstupu tzv. glitch filter, který odfiltruje pulzy, které mají délku trvání menší jak jeden hodinový takt. Ale po aplikování filtru mají vstupy zpoždění dva takty a navíc se uplatní pouze pro piny, které jsou využity pro přerušení. Periferie umožňuje pokročilé funkce jako je nastavení pull-up rezistoru nebo výstupu s otevřeným kolektorem..

Přístup k vstupně/výstupním pinům může být prováděn čtyřmi různými způsoby. První je přímý přístup k registrům (paměti). Tento způsob je prakticky stejný, jaký se používá u 8bitových AVR. Další přístup je pomocí „set“. Prakticky to znamená, že je jen nastavována maska, která definuje, které bity budou nastaveny. Jako další je zde „clear“ přístup. Opět se v podstatě jedná o masku, která definuje, které bity budou nastaveny na log. 0. Poslední je „toogle“ přístup. Opět jde o masku, která invertuje jen nastavené bity. Díky těmto metodám je možné efektivněji přistupovat k pinům, protože v praxi je často prováděno maskování softwarově, což stojí čas procesoru, a tím se i snižuje výpočetní výkon.

3.3.5. Interrupt Controller (INTC)

Periferie, která má na starost přerušení. Zatímco např. u populárních 8bitových AVR mikrokontroléů je vektor přerušení dán hardwarově, UC3 umožňuje si tento vektor nadefinovat. Konkrétně UC3A3256 umožňuje přerušení rozdělit do 64 skupin po 32 přerušeních. Prakticky to znamená, že priority jednotlivých přerušení lze definovat v podstatě libovolně. Tato flexibilita je ale vykoupena větší náročností na nastavení vektoru přerušení. Blokové schéma je na Obr. 9.



Obr. 9: Blokové schéma periferie pro přerušení (převzato z [1])

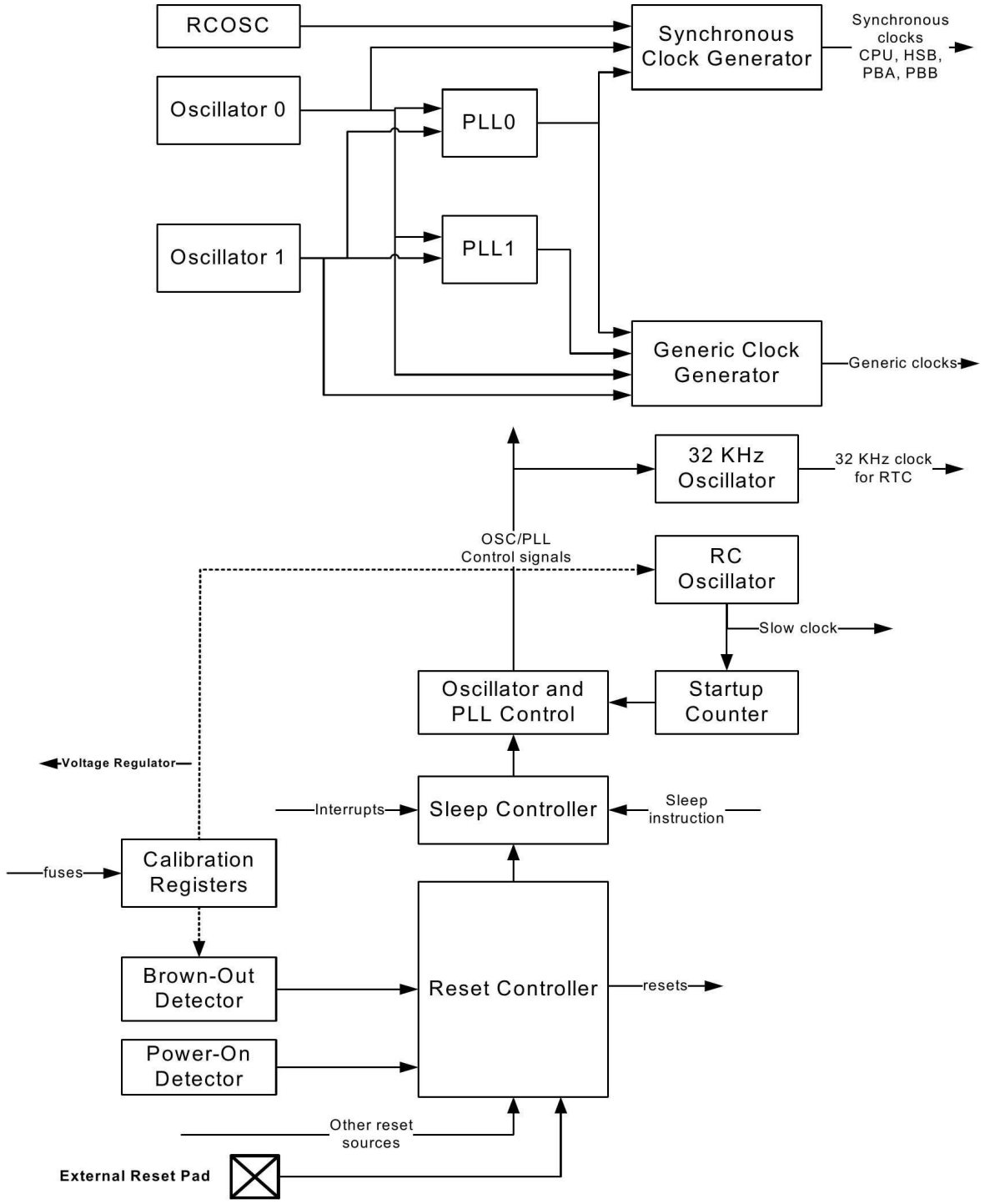
Jednotlivé vektory přerušení jsou označeny IREQn, kde „n“ je číslo vektoru. Jednotlivé vektory přerušení jsou vzorkovány do IRR (Interrupt Request register) příslušných skupin. Poté se provede maskování a výsledný signál přerušení jde do CPU jednotky. Té jsou předány nezbytné informace pro provedení přerušení.

3.3.6. Power Manager (PM)

Power manager má za úkol nastavovat tzv. sleep režimy, takt jednotlivých sběrnic, obsluhovat wake-up eventy ze zdrojů přerušení a resetovat MCU. Obsahuje mj. i PLL a integrovaný RC oscilátor. Další vlastností je generování tzv. hard a soft reset. Blokový diagram je na Obr. 10.

PM má možnost vyvést hodiny „ven“ pomocí GPIO. To je užitečné, když je zapotřebí mít třístavový výstup. Díky tomu je možné jednoduše nastavovat směr hodin (master/slave).

U 8bitových AVR se zdroj hodin nastavoval pomocí pojistek (fuses), což mohlo při špatné konfiguraci způsobit, že AVR již nebylo možné přeprogramovat (protože nastavený zdroj hodin nebyl k dispozici). Tento problém je elegantně vyřešen u UC3. Ihned po restartu UC3 je použit integrovaný oscilátor. Je pak na aplikačním firmwaru, aby správně přenastavil registry, které definují zdroj hodin. Takže i když bude špatně napsaný firmware, bude stále možné UC3 přeprogramovat, protože při restartu (součást programovacího procesu) bude nastaven jako zdroj hodin integrovaný pomalý RC oscilátor.



Obr. 10: Blokový diagram PM (převzato z [1])

Na piny „Oscillator 0“ a „Oscillator 1“ může být připojený externí zdroj hodin nebo krystal. Tento hodinový takt může být pak zvýšen, nebo snížen pomocí integrovaných PLL. V případě, že je zapotřebí tyto piny použít jako vstupně/výstupní, tak je to možné. Samozřejmě tím přijdeme o jeden možný zdroj taktu. Ihned po restartu jsou tyto piny nastaveny jako vstupně/výstupní a až aplikační firmware musí nastavit piny jako zdroj hodin. Aktuální program využívá oba piny pro hodiny.

Jako další periferie je zde zdroj taktu 32 kHz. Zdroje o této frekvenci se často používají pro aplikace RTC (Real Time Clock). Ihned po zapnutí je periferie vypnutá a její použití je volitelné nastavením registrů. Tento zdroj hodin po zapnutí běží i během resetu (výjimkou je reset, který

probíhá při připojení napájení), což je výhodné, pokud je takt vyveden externě z MCU do nějakého dalšího obvodu (hodiny běží stále).

UC3A3256 nabízí hned dvě nezávislé PLL. Díky tomu je možné jako zdroj hodin použít například 12 MHz krystal, a přesto je možné, aby CPU pracovalo například na 66 MHz (12 MHz x11/2). Dále umožňuje, aby CPU spolu s HSB běželo na jiném taktu než PBA a PBB. CPU pak může fungovat na relativně vysokém taktu a přitom vstupně/výstupní piny mohou běžet na mnohem menším taktu. To umožňuje efektivně řídit spotřebu a zároveň je možné v případě potřeby změnit hodiny bez nutnosti pozastavení jakékoliv komponenty.

PM má také na starost tzv. „sleep módy“. Podle toho, jak potřebujeme využívat jednotlivé periferie, je možné nastavit jeden ze sleep módů. Nejméně výrazný sleep mód je „Idle“. Takt do CPU je vypnut, ale veškeré ostatní periferie běží. Naopak nejvíce úsporný sleep mód je „static“, ve kterém jsou veškeré hodiny zastaveny. Jediné možné „probuzení“ je externím přerušením nebo resetem (což je ve své podstatě také externí přerušení).

Jak bylo naznačeno výše, je několik „úrovní“ resetu. Nejvyšší je „Power-on reset“. Ten provede reset všech periferií a pamětí. Další je „External reset“. To je reset, který je proveden pomocí externího pinu. Při tomto resetu se již neresetuje 32 kHz oscilátor, vybrané registry (RTC, kalibrační pro oscilátory). Dále je zde například reset od Watchdogu, BOD atd. Podrobný popis lze vyhledat v datasheetu. Důležité je si uvědomit, že zde existují různé úrovně resetu, a tedy „není reset jako reset“.

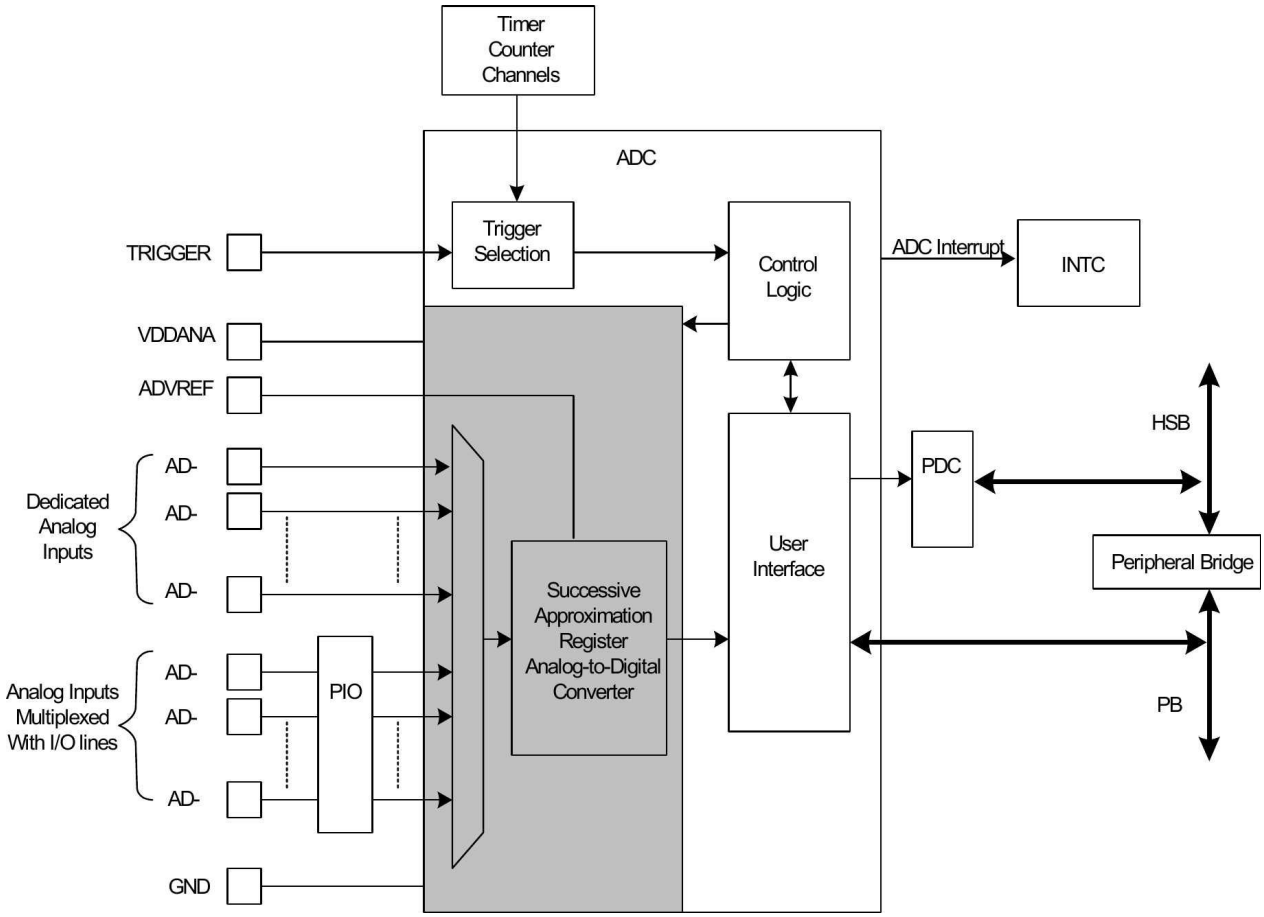
3.3.7. Analog-to-Digital Converter (ADC)

Pro jednodušší ladění I^2S sběrnice je na DPS použit kodek. Je vhodné nějakým způsobem umožnit řízení výstupní hlasitosti na kodeku (pro případ, že signál nepůjde na osciloskop, ale do sluchátek). Na to je vhodný obyčejný potenciometr. Jeho analogová hodnota se převádí na číslo a to pak určí hlasitost na kodeku.

Během testování se ukázalo, že je vhodné detekovat, zda je na straně konektoru napájecí napětí (pro kontrolu validity signálů). Detekce je prováděna pomocí optočlenu s fototranzistorem, takže výstup není čistě digitální. Proto je vhodné tento signál měřit pomocí ADC. Navíc lze (s určitou přesností) odhadovat skutečné napětí na straně konektoru.

Na takový nenáročný úkol zcela postačí integrovaný převodník, který je standardem v každém MCU.

ADC na UC3A3256 podporuje 8bitové a 10bitové rozlišení, PDCA, sleep mód, hardwarový a softwarový trigger. Pro správnou funkci je potřeba nejprve piny namapovat pomocí GPIO a pokud chce programátor využít i přerušení, musí nejprve nastavit komponentu INTC. Samotný ADC má integrovanou děličku, takže pokud je potřeba, tak lze snížit spotřebu tak, že bude snížen taktovací kmitočet. Blokový diagram je na Obr. 11.



Obr. 11: Blokový diagram ADC (převzato z [1])

4. HW řešení

Základní schéma zapojení vychází z projektu SDR-Widget, který je volně dostupný [13] a implementuje USB audio zařízení na shodném mikrokontroléru, avšak s odlišným cílem. Vyžití již existujícího HW základu bylo motivováno následujícími faktory:

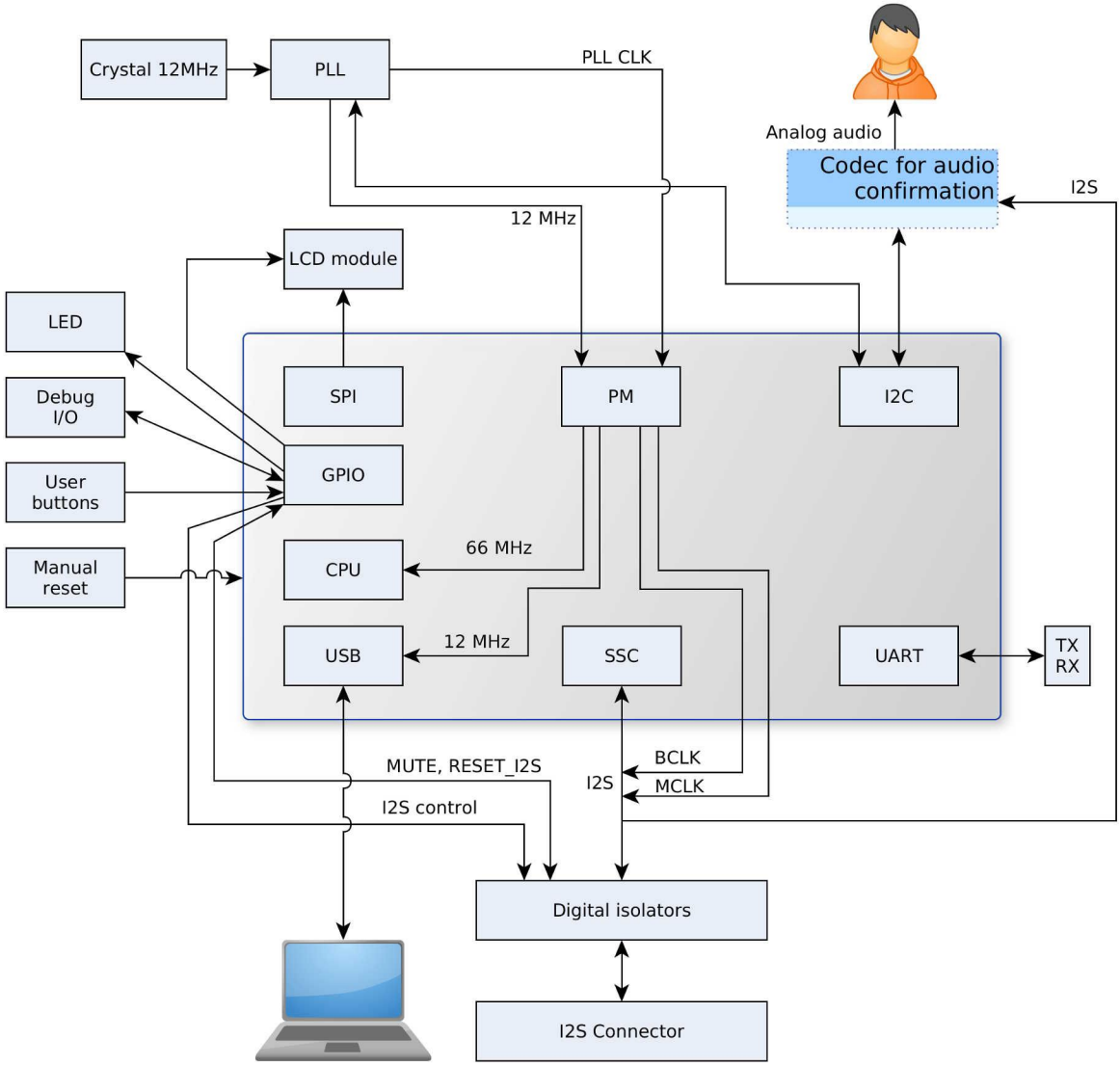
- na trhu existuje jen velmi málo vhodných typů MCU pro daný účel, dostupných pro malosériové ruční prototypování, takže překryv s projektem SDR widget byl prakticky nevyhnutelný,
- existující funkční koncept umožnil rychlejší postup do další fáze implementace.

Z původního zapojení SDR Widget projektu nakonec zbyl v podstatě jen UC3A3256 a čtyři informační LED. Všechny ostatní komponenty byly zaměněny, nebo úplně vynechány.

Celkový odběr zařízení v provozu se pohybuje okolo 230 mA, takže zařízení lze napájet přímo z USB 2.0 sběrnice (limit USB 2.0 je 500 mA). Podle standardu USB je host povinen zařízení dodat proud alespoň 100 mA. Pokud má zařízení větší nároky na napájení, musí hostu tuto skutečnost nahlásit. To umožňuje jeden z deskriptorů, které popisují zařízení jako celek. Vzhledem k relativně malým proudovým odběrům samotných integrovaných obvodů byly při návrhu použity obyčejné lineární stabilizátory. Jejich nevýhodou je sice nižší účinnost než u spínaných zdrojů, ale na druhou stranu jsou mnohem jednodušší, méně náročné na komponenty, a na jejich výstupu je mnohem menší vysokofrekvenční šum, který má negativní dopady na analogové obvody pro kontrolní poslech. Zejména na SNR u kodeku, který je na desce spolu s UC3A3256.

4.1. Blokové schéma hardwaru

Propojení jednotlivých funkčních bloků s MCU je popsáno na Obr. 12.



Obr. 12: Blokové schéma hardwarového návrhu

Externí PLL má dva výstupy. Jeden je fixní a jeho takt je dán přímo zdrojem hodin pro PLL (ve výchozím nastavení). Tento fixní výstup je pak použit jako zdroj hodin pro CPU nebo USB. Druhý výstup je již variabilní (PLL CLK) a takt je dán nastavením registrů v PLL.

SSC potřebuje pro správnou funkci referenční BCLK. Ten může být generován z modulu PM, nebo jako zdroj může posloužit I^2S sběrnice. Přepínání zdroje hodin lze provádět pouze softwarově. V případě, že BCLK má být generován, pak v modulu PM musí být nastaven jeho zdroj (PLL CLK), dělící poměr a přiřazen patřičný výstupní pin. Signál je tedy vyveden „ven“ z pouzdra. To umožňuje jednoduše a rychle změřit jeho frekvenci, takže ladění aplikace se stává jednodušším. Pokud má být zdrojem BLCK I^2S sběrnice, pak stačí pouze nastavit patřičný pin do Hi-Z. Tím dojde k odpojení od PM bloku. Signál MCLK není pro SSC nijak potřebný. Přesto může být nastaven stejně jako BCLK (tzn. může být vstupní i výstupní). Kodek je pak přímo napojen na I^2S sběrnici tak, aby ze sběrnice pouze data přijímal. To znamená že umožňuje sběrnici pouze odposlouchávat.

Zařízení má mít možnost pracovat jako I^2S master, i jako I^2S slave. Je proto nutné, aby digitální izolátory umožňovaly přenos dat oběma směry. Resp. digitální izolátory umožňují nastavit nezávisle směr pro MCLK, BCLK, FRAME_SYNC, MUTE a RESET_I2S, takže zařízení nemusí být čistě master nebo slave. To je opět výhodné při ladění, nebo použití zařízení v nestandardních situacích. Pro řízení směru toku dat jsou použity piny z GPIO modulu. Samotný I^2S konektor obsahuje standardní I^2S signály plus další pomocné pro jednodušší řízení celé aplikace (MUTE, RESET_I2S, MCLK, napájení).

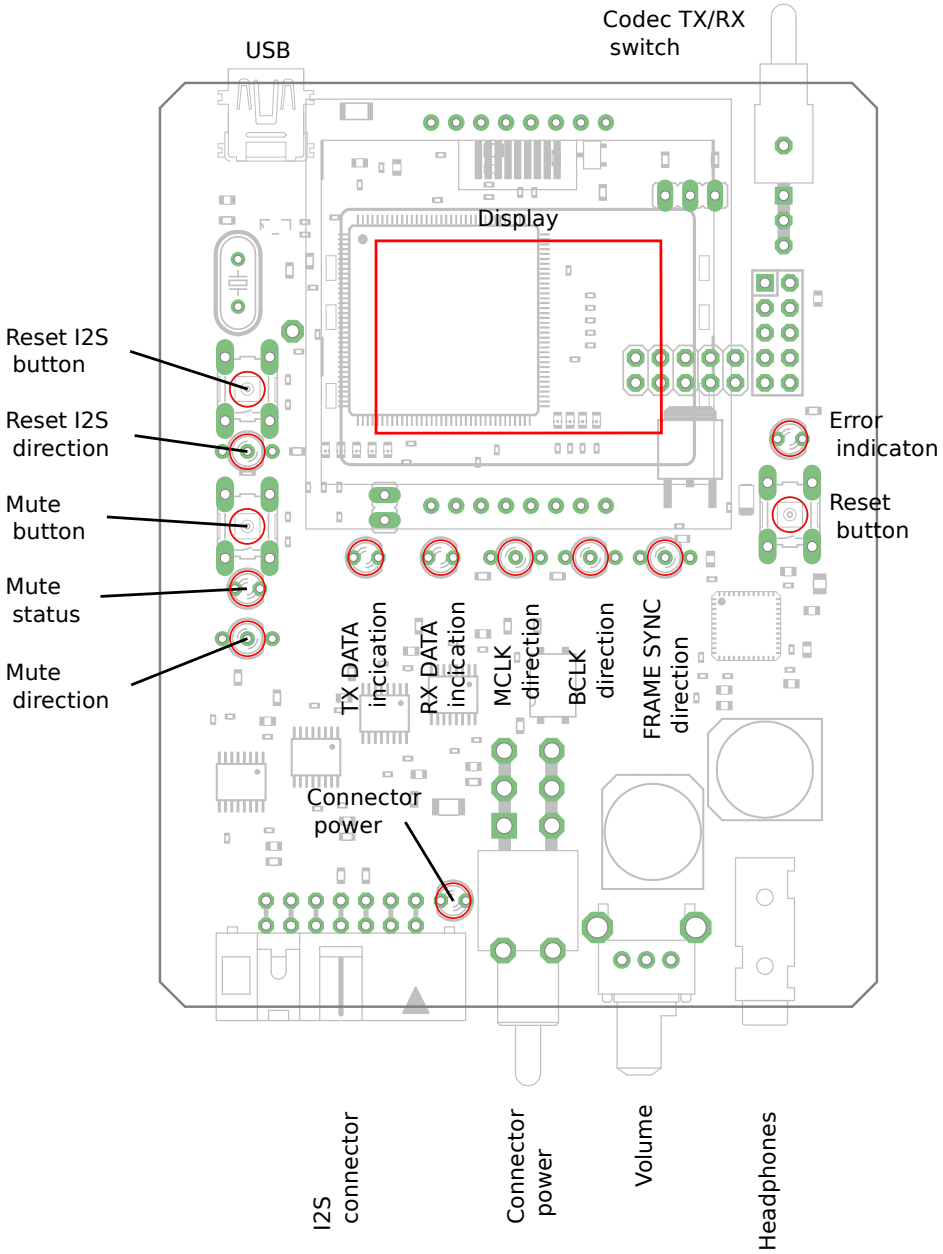
Pro korektní funkčnost LCD modulu je zapotřebí nejen SPI signálů, ale jsou použity další piny z GPIO modulu. LCD modul má za úkol informovat uživatele o stavu zařízení a o výskytu případných chyb.

Pro ladění je na desce ještě vyveden UART, který je na výpis ladících informací mnohem vhodnější než LCD display. UART umožňuje posílat ladící informace pomocí převodníku do počítače, kde je na vývojáři, aby si potřebné informace zpracoval či zobrazil. Signály RX a TX jsou vyvedeny na desku, ale nejsou vyvedeny z krabičky. Běžný uživatel je totiž nebude potřebovat.

Signály MUTE, RESET_I2S a NSRST je možné ovládat tlačítky. Signál NSRST je spojen přímo s MCU a kodekem, takže při jeho stisku se okamžitě vyvolá reset na obou zařízeních. Tlačítka pro MUTE a RESET_I2S nejsou zapojena na I^2S sběrnici, ale na piny MCU. To je pro případ, že by tyto signály byly nastaveny jako výstupní a v případě stisku tlačítka (které by bylo připojeno přímo) by byl výstup na MCU přetěžován. Aktuální zapojení řeší tento problém. Pokud bude pin na MCU nastaven jako výstupní, pak software bude vzorkovat hodnotu na tlačítku a tuto hodnotu nastaví i na daný signál (MUTE/RESET_I2S). V případě, že signálový pin bude nastaven jako vstupní, pak MCU nebude nastavovat úroveň signálu, ale pouze provede dané operace. To znamená že buď ztiší výstup pro kontrolní poslech, nebo resetuje I^2S sběrnici (výchozí nastavení).

Pro rychlý přehled byly na desku přidány signalizační LED, které mají uživateli zpřehlednit aktuální nastavení a stav na některých pinech.

Pro debug byly rezervováno osm vstupně výstupních pinů a 9 LED. Indikační SMD LED nemusí být ve finální podobě osazovány. Jedná se pouze o LED, které mají za úkol usnadnit vývoj (signalizace stavů, přetečení bufferu atd). Vyhrazené vstupně výstupní piny je ale možné v konečné fázi využít libovolně. Například zde budou připojena tlačítka pro uživatele, nebo další signalizační prvky. Rozmístění uživatelských periferií je na Obr. 13.



Obr. 13: Rozmístění uživatelských periferií na desce

4.2. MCU AT32UC3A3256

Protože vstupně/výstupní piny jsou stavěny na rozmezí 3 až 3,6 V [1], je zapotřebí, aby napětí na VDDIO, VDDIN a VDDANA bylo v tomto rozsahu. Proto bylo zvoleno jako napájecí napětí pro MCU a další komponenty 3,3 V. Většina moderních obvodů je schopna pracovat spolehlivě při takto nízkém napětí. Navíc se sníží i proudový odběr ostatních integrovaných obvodů. Pro napájení jádra MCU a interní PLL je sice zapotřebí 1,8 V, ale MCU již má integrovaný stabilizátor, takže není potřeba dalších komponent (vyjma blokovacích kondenzátorů).

Pro naprogramování MCU je zde JTAG rozhraní. Díky JTAG je teoreticky možné ladit program, sledovat obsah registrů, ale problém nastává při komunikaci s periferiemi, které trvale vyžadují odpověď od MCU. Typicky jde o USB rozhraní v Isochronním módu. Pokud se program zastaví na breakpointu, pak USB má za úkol periodicky odesílat a přijímat data, ale MCU nebude odpovídat, což může vyústit v neočekávané chování programu. To samé platí při použití RTOS, kde je ladění celé aplikace velmi problematické.

U každého tlačítka je již připravený RC článek, který sníží počet zákmitů na tlačítku a zjednoduší se i obslužný software.

Jak bylo uvedeno výše, při debugování skrze JTAG není zcela vyloučeno, že nenastane nějaký nečekaný problém. Proto speciálně pro debug bylo vyhrazeno osm vstupně/výstupních pinů. Na tyto piny mohou být připojené například další LED indikující různé stavy, nebo tlačítka, pomocí kterých bude možné řídit běh programu v reálném čase. Tyto signály jsou na schématu označeny jako GPIO_x, kde „x“ značí číslo signálu. Pro debug zprávy lze využít i UART nebo LCD, ale zde je limit v počtu zobrazených znaků.

U většiny výstupních pinů jsou sériově přidány ochranné odpory. Tyto odpory limitují maximální proud jednotlivých výstupů a tím chrání výstupní obvody při zkratu nebo jiném problému. Zároveň je na nich možné měřit úbytek napětí. Tam kde je vysoký úbytek lze očekávat větší proud a tím pádem i nějaký problém. To opět usnadňuje ladění a zvyšuje robustnost celé aplikace.

4.3. Externí PLL

Kvůli potřebě generovat necelistvé násobky vzorkovacích frekvencí byl vybrán programovatelný obvod CS2200-CP od Cirrus Logic. Tento obvod umožňuje nastavit výstupní frekvenci v rozsahu 6 až 75 MHz. Výstupní frekvence se nastavuje především poměrem ve formátu 12.20b. To znamená, že horních 12 bitů je před desetinnou čárkou a dalších 20 bitů je za desetinnou čárkou. Díky tomuto poměru je možné nastavovat výstup s krokem 10 ppm (bráno ze vstupního zdroje hodin). Vstupní rozsah frekvencí pro krystal je od 8 do 50 MHz.

Protože obvod má dva výstupy (AUX_OUT a CLK_OUT), je vhodné je efektivně využít. Jako zdroj signálu je použit 12 MHz krystal (především kvůli USB periferii – viz. předchozí kapitola). To znamená, že na výstupu AUX_OUT je po přivedení napájení takt o 12 MHz. Tento takt je přiveden přímo do MCU, kde je vynásoben pro USB modul, CPU a další periferie. Druhý výstup (CLK_OUT) je po přivedení napájení ve stavu vysoké impedance a je na MCU, aby tento obvod přes I^2C sběrnici správně nakonfigurovalo. Tento výstup je použit pro generování MCLK a BCLK pomocí děliček integrovaných v MCU.

4.4. Galvanické oddělení pro I^2S konektor

Galvanické oddělení je možné několika způsoby. Nejčastěji se používají transformátory, optočleny a vazba RF signálem. Transformátory nejsou pro tuto aplikaci vhodné, protože nemohou přenášet stejnosměrnou složku. To je diskvalifikuje pro signál, který nastavuje mute. Velice často se používají optočleny. Jejich nevýhodou je vyšší napájecí napětí (okolo 5 V) a velice špatná dostupnost pro vysoké přenosové rychlosti. Většinou lze sehnat optočleny pracující do 1 MHz, ale vzhledem k tomu, že MCLK se pohybuje kolem 12 MHz je 1 MHz naprosto nevyhovující. Na druhou stranu umožňují přenášet i stejnosměrnou složku. Spíše neobvyklou metodou oddělování jsou tzv. digitální izolátory. V podstatě jde o podobný princip jako u optočlenů, ale rozdíl je ve frekvenci, na které je přenášena informace. U optočlenů je informace přenášena na několika THz. K příjmu je pak použita fotodioda, nebo fototranzistor a signál je dále zpracován (například hradlem). U digitálních izolátorů je namísto světla použito RF pásmo (stovky MHz až jednotky GHz).

Firma Silicon Labs nabízí právě digitální izolátory, které zvládají přenést až 150 Mbit/s při napájecím napětí od 2,7 V do 5,5 V [7]. Jak už název napovídá, obvody jsou určené čistě pro přenos digitální informace (na rozdíl od optočlenů, které umožňují přenášet i analogový signál). Navíc digitální izolátory od Silicon Labs nabízí třístavový výstup, což je vhodné pro signály, u nichž se může měnit směr (např. MCLK). Dostupnost těchto komponent na maloobchodním trhu je o něco horší než u optočlenů, ale jejich parametry jsou pro danou aplikaci ideální.

Kvůli tomu, že jsou použity aktivní součástky pro galvanické oddělení, je zapotřebí napájet zvlášť i stranu s konektorem. Napájení může být přímo na konektoru. Není to však nezbytnost. Pokud totiž není vyžadováno galvanické oddělení, pak je možné stranu s konektorem napájet přímo ze stabilizátoru, který napájí MCU. To je vhodné například při vývoji, kdy veškerá zařízení jsou připojena k jednomu počítači, a tedy není nutné galvanické oddělení.

4.5. Kodek pro kontrolní poslech I^2S signálu

Jako kodek byl zvolen obvod TLV320AIC33. Tento obvod je vyráběn v pouzdru 48-QFN, takže jeho rozměry jsou velmi kompaktní (7x7 mm) [2]. Jeho hlavní výhodou je možnost velmi jemného nastavení jak analogové, tak digitální části.

Obvod má integrované analogové vstupy: 2x stereo a 1x mono. Každý může pracovat jako diferenční i jako jednoduchý. Analogové výstupy jsou zde 3x stereo a 1x mono. Opět je možné nastavit, zda bude výstup diferenční, nebo ne. Jeden stereo výstup je navržen pro přímé připojení sluchátek, čili odpadá stavba koncového stupně. Vstupy nebudou použity, takže kvůli úspoře místa na DPS nejsou na vstupy připojeny žádné odpory, díky kterým by šly vstupy v budoucnu použít. Výstup bude použit jeden: sluchátkový. Ostatní výstupy nebudou použity a jejich piny nebudou vyvedeny ven. Opět kvůli úspoře místa na DPS.

Obvod potřebuje ke své funkci tři zdroje napětí. Jeden zdroj pro analogovou část kodeku (3,3 V), další pro digitální jádro (1,8 V) a poslední pro digitální vstupně/výstupní piny (3,3 V). Právě kvůli snížení šumu na analogové části je napájení pro analogovou a digitální část odděleno už samotnými regulátory. Pozice kodeku a regulátoru pro analogové obvody je záměrně zvolena co nejdále od ostatních obvodů, aby bylo zamezeno indukci nechtěného rušení od dalších obvodů a sběrnic.

Kodek umožňuje mixování vstupních signálů s výstupními. Vzhledem k tomu, že v tomto případě bude kodek využit pouze jako výstup, nebudou tyto možnosti dále probírány.

Výběr a popis některých parametrů je v Tab. 4.

Tab. 4: Popis vybraných parametrů kodeku (převzato z [2])

Parametr	Hodnota	Popis
DAC SNR	100 dB	Pro ověření signálu na I^2S sběrnici naprosto dostačující parametr. Vhodné i pro méně náročné měření. Zde již záleží na designu DPS a kvalitě jednotlivých komponent (regulátory, blokovací kondenzátory atd.)
Napájení analogové části	2,7 V- 3,6 V	Napájecí napětí bylo zvoleno 3,3 V. Jedná se o standard a je to i doporučeno v datasheetu. Přestože pro napájení analogové části je zde zvlášť stabilizátor, byl na desku ke kodeku přidán LC filtr.
Napájení digitálního jádra	1,65 V – 1,95 V	Toto je jediný pin na desce, který vyžaduje takovéto napětí. Proto je zde další stabilizátor na 1,8 V. Napětí 1,8 V je doporučeno výrobcem.
Napájení digitálních vstupů/výstupů	1,42 V – 3,6 V	Vzhledem k tomu, že použité napětí pro MCU je 3,3 V, tak je výhodné použít toto napětí i pro napájení vstupů/výstupů kodeku.

Parametr	Hodnota	Popis
		Tím pádem nejsou potřeba žádné konvertory mezi MCU a digitálním rozhraním kodeku.
Rozsah vzorkovacích frekvencí	8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz	Protože jednotlivé vzorkovací frekvence si navzájem nejsou celými násobky, je nutné nejen správně nastavit kodek, ale zvolit i správný takt pro MCLK, BCLK a FRAME_SYNC. Z toho plyne přímá návaznosti na externí PLL, SSC a děličku v MCU.
Šířka slova	16,20,24,32 bitů	Nejnižší rozlišení (16 bitů) odpovídá CD standardu [6], Další podporované možnosti jen přidávají kodeku na univerzálnosti a věrnosti původnímu záznamu.
Podporované standardy pro digitální rozhraní	I^2S, MSB justified, LSB justified, DSP, TDM	Mimo požadovaný standard I^2S podporuje i standardy využívané především DSP procesory. Vhodné pro budoucí rozšíření, kdy výstup MCU nebude jen I^2S.
Nastavení rozhraní	I^2C, SPI	Samotný kodek je možné nastavovat skrze I^2C nebo SPI. Vzhledem k tomu, že I^2C kontroluje odeslaná data už na nejnižší úrovni (příjemce musí odpovědět), je jednodušší ovládat kodek právě pomocí I^2C.
Analogový výstup	Line out, headphones	Na čipu je již integrovaný zesilovač pro sluchátka. Díky tomu je možné přímo na piny připojit sluchátka (32 Ω) bez dalších obvodů navíc. Kodek umožňuje softwarové ovládání hlasitosti, takže pokud je potřeba, může být původní signál zeslaben v rozsahu od 0 do -78 dB. Dále je zde line out výstup, který ale není v této situaci zapotřebí a není nikam připojen.

4.6. LCD modul

Jako display byl vybrán grafický modul původně určený pro telefony Nokia 5110. Tento produkt je v současnosti populární volbou v prototypových zapojeních kvůli svým parametrům a dostupnosti. Rozlišení je 84x48 px, což je postačující pro zobrazení základních informací. Display v podstatě používá SPI rozhraní rozšířené o signál, který přepíná mezi datovým slovem, příkazem a resetem. Modul dokáže komunikovat až rychlostí 4 Mbit/s [9], což je na display dostačující. Díky popularitě tohoto modulu existuje několik volně dostupných knihoven s předdefinovanými písmy. Experimentálně bylo vyzkoušeno, že modul funguje i při taktu 12 MHz, takže v případě potřeby je možné modul „přetaktovat“ bez větších potíží, a tím urychlit chod programu. Popis pinů je v Tab 5.

Tab. 5: Přehled pinů na LCD modulu (převzato z [9])

Pin	Popis
1 - RST	Reset. Aktivní v nízké úrovni. Nutný před inicializací.
2 - CE	Chip enable. Povoluje SPI rozhraní. To je užitečné v případě, že SPI sběrnici využívá několik zařízení současně.
3 - DC	Data/command. Přepíná mezi surovými daty pro displej a příkazy pro řadič displeje.
4 - DIN	Data in. Sériová reprezentace dat. Data musí být platná při náběžné hraně signálu CLK.
5 - CLK	Clock. Takt pro řadič. Podle datasheetu může dosahovat až 4 MHz.
6 - VCC	Napájení pro modul. Doporučené napětí je 3,3 V.
7 - LIGHT	Podsvětlení. Aktivní v nízké úrovni, takže vhodné spínat NPN tranzisotrem.
8 - GND	Zem

Inicializace a další komunikace s modulem je podrobně popsaná v datasheetu [9], takže zde nebude dále rozebírána.

5. SW řešení

Vzhledem k tomu, že hardware vychází z projektu SDR-Widget, je výhodné použít již napsaný a vyzkoušený software jako základ a ten dále rozšiřovat a modifikovat. Protože kromě MCU jsou prakticky všechny komponenty na DPS jiné a architektura UC3 není příliš rozšířená, bylo potřeba napsat ovladač pro každou komponentu (PLL, kodek, řídící piny). Zároveň bylo nutné odstranit z původního kódu vazby na komponenty, které ve skutečnosti nebyly připojeny.

Řešení využívá FreeRTOS. Jedné se o volně dostupnou implementaci „real time operating system“ - RTOS [15]. V praxi to znamená, že na MCU běží zdánlivě paralelně několik vzájemně nezávislých úloh (tasků). Je zvykem, že každý task je napsán jako funkce, která běží v nekonečné smyčce. Uvnitř této smyčky se provádí samotné zpracování dat. Díky použití RTOS je možné nastavit priority jednotlivých tasků, a tím jednodušeji implementovat úkoly, které má MCU plnit. Například task pro zpracování audio dat bude mít vysokou prioritu a bude volán mnohem častěji než task, který má na starost tlačítka, které je potřeba zkontrolovat jednou za několik desítek ms. Na druhou stranu RTOS přináší i některé nevýhody:

- režie při přepínání jednotlivých tasků,
- nutná definice stacku (v některých případech je složité správně odhadnout využití RAM),
- nutná signalizace při používání společného hardwaru (používají se tzv. mutexy),
- nutnost vyhradit alespoň jeden hardwarový časovač pro RTOS.

I přes výčet nevýhod je pro tento projekt zdánlivě paralelní zpracování zvláště vhodné a umožňuje efektivní a přehlednou implementaci v kódu.

5.1. USB Deskriptory

Deskriptory jsou souborem konstant, které popisují jednotlivé vrstvy USB zařízení (zařízení jako celek, jednotlivá interface, endpointy atd.). Díky deskriptorům je host schopný detekovat, o jaké zařízení se jedná, jakým způsobem komunikuje a jaké jsou jeho možnosti. Umožňují definovat povahu přenášené informace (surová data, zpětná vazba, atd.). Zároveň jsou některé deskriptory prezentované jako textová pole, takže i uživatel může snadno zjistit název zařízení, výrobce a nebo sériové číslo. Ve zdrojových kódech jsou deskriptory definovány v souborech, které mají v názvu „usb_descriptors“. Stručný přehled vybraných deskriptorů pro celé zařízení je v Tab. 6.

Tab. 6: Stručný přehled deskriptorů zařízení

Deskriptor	Hodnota	Popis
bcdUSB	2.00	Verze USB, kterou zařízení využívá. V tomto případě nepodporuje starší USB 1.1
idVendor	0x16D0	Složené audio zařízení [8]
idProduct	0x0763	Product ID. Pouze číslo pro odlišení od SDR-Widget, který používá 0x0761 a 0x0762. Nemá vliv na instalaci ovladače v OS.
iManufacturer	ALCZ	Výrobce
iProduct	Sonochan mkII	Jméno produktu
iSerial	0.0.0.0.0.0.1	Sériové číslo
bNumConfigurations	1	Počet konfigurací
bNumInterfaces	4	Počet interface
bmAttributes	0xC0	Napájeno z USB
MaxPower	130	Maximální proudový odběr v mA podělený dvěma. Zařízení hlásí, že požaduje zdroj, který dodá alespoň 260 mA.

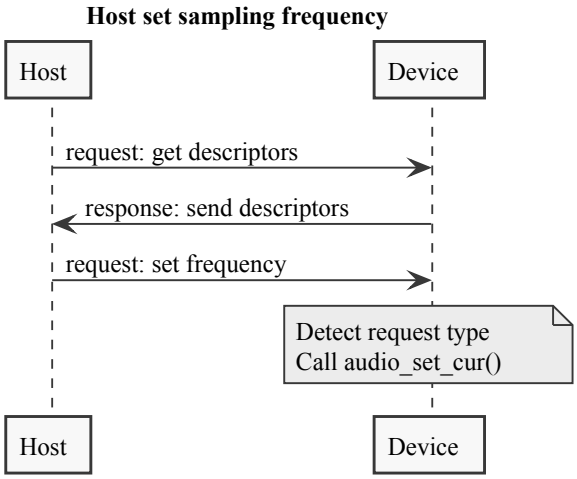
Každý interface pak obsahuje další deskriptory, které jej popisují. Vlastnosti jednotlivých interface jsou shrnuty v Tab. 7. Vzhledem ke komplexnosti USB protokolu jsou shrnuty vlastnosti jen některých deskriptorů.

Tab. 7: Popis jednotlivých interface

Interface 0	HID – pro konfiguraci zařízení pomocí počítače. Obsahuje dva EP: vstupní a výstupní pro plně duplexní komunikaci. Pakety mají velikost 8 B a mohou být odesílány v intervalu 5 ms (mód „Interrupt“).
Interface 1	Popisuje ovládací prvky (mute, volume), počet kanálů na interface 2 a 3.
Interface 2	Výstupní audio stream ve formátu PCM s rozlišením 24 bitů. Mód přenosu: „Isochronous“. Synchronizace: „Asynchonous“. Maximální velikost paketu: 308 B. Podporované vzorkovací frekvence: 8000, 11025, 16000, 22050, 24000, 32000, 44100 a 48000 Hz.
Interface 3	Vstupní audio stream ve formátu PCM s rozlišením 24 bitů. Mód přenosu: „Isochronous“. Synchronizace pomocí zpětné vazby. Maximální velikost paketu: 308 B. Podporované vzorkovací frekvence: 8000, 11025, 16000, 22050, 24000, 32000, 44100 a 48000 Hz.

5.2. Princip nastavení vzorkovací frekvence

Při USB inicializaci host získá všechny potřebné parametry, včetně seznamu podporovaných frekvencí. Před samotným přehráváním (resp. odesíláním audio dat) odešle host požadavek, ve kterém je zahrnuta informace o vzorkovací frekvenci. V tomto momentě je na zařízení, aby zvolilo platnou konfiguraci. Host poté začne odesílat audio data. Celá komunikace je zjednodušeně znázorněna na Obr. 14.

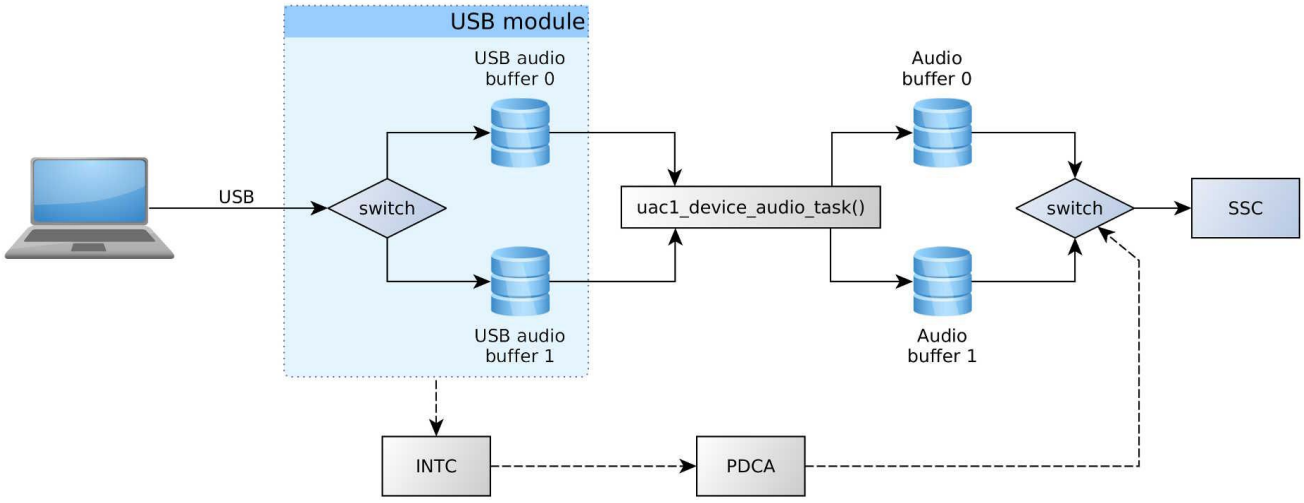


Obr. 14: Nastavení vzorkovací frekvence

Funkce `audio_set_cur()` má za úkol nastavit externí PLL a děličky GCLK[0] a GCLK[1] tak, aby FRAME_SYNC, BCLK a MCLK odpovídaly požadavkům uživatele. V případě ladění aplikace může rozsvítit danou kombinaci LED, poslat zprávu přes UART nebo LCD.

5.3. Tok audio dat z počítače do sériového synchronního rozhraní

Už z principu je velikost hardwarových USB bufferů menší než velikost bufferů pro audio stream. Proto je nutné řešit způsob ukládání dat. Na Obr. 15 je znázorněný výstupní audio stream. Diagram pro vstupní audio stream by byl prakticky stejný, ale směr šipek pro audio data by byl opačný. Plné čáry reprezentují audio stream, přerušované pak řídící signály.

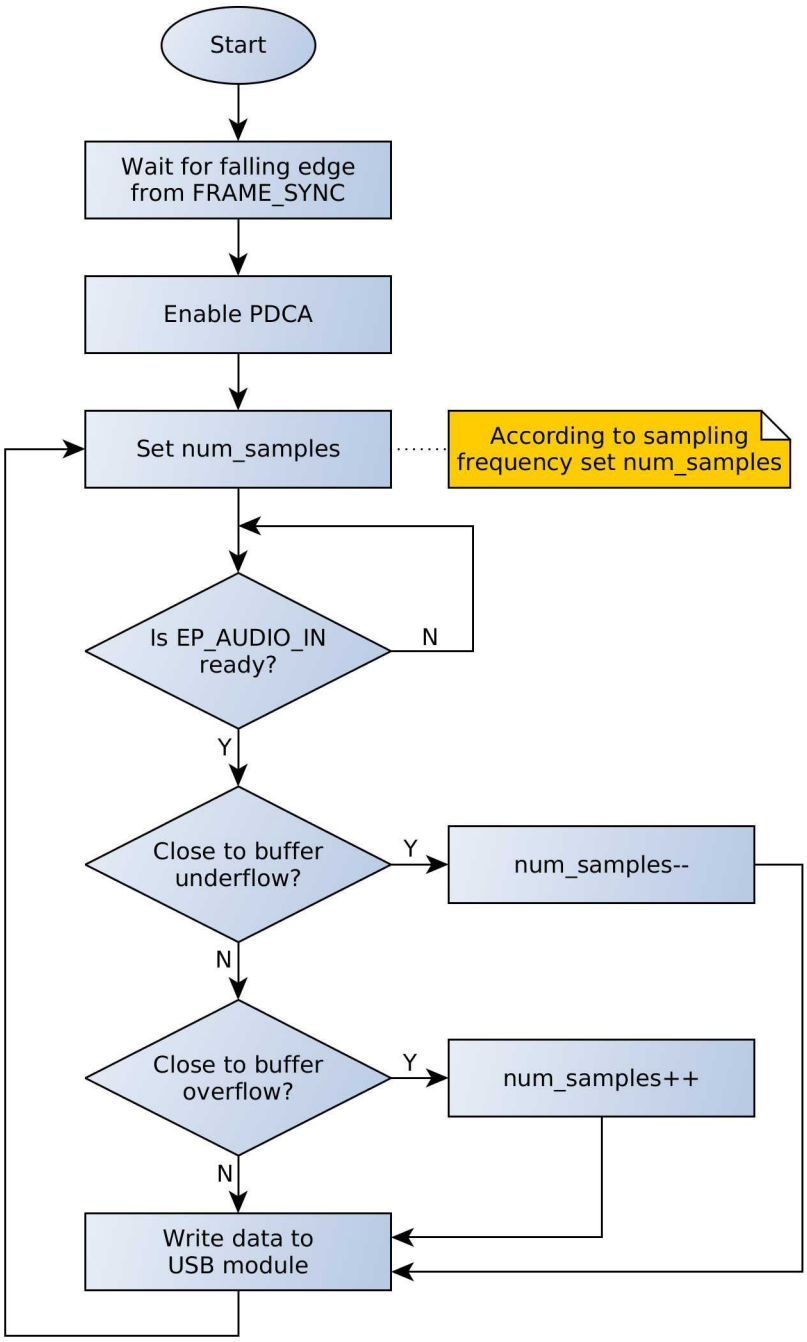


Obr. 15: Zjednodušený diagram pro výstupní audio stream

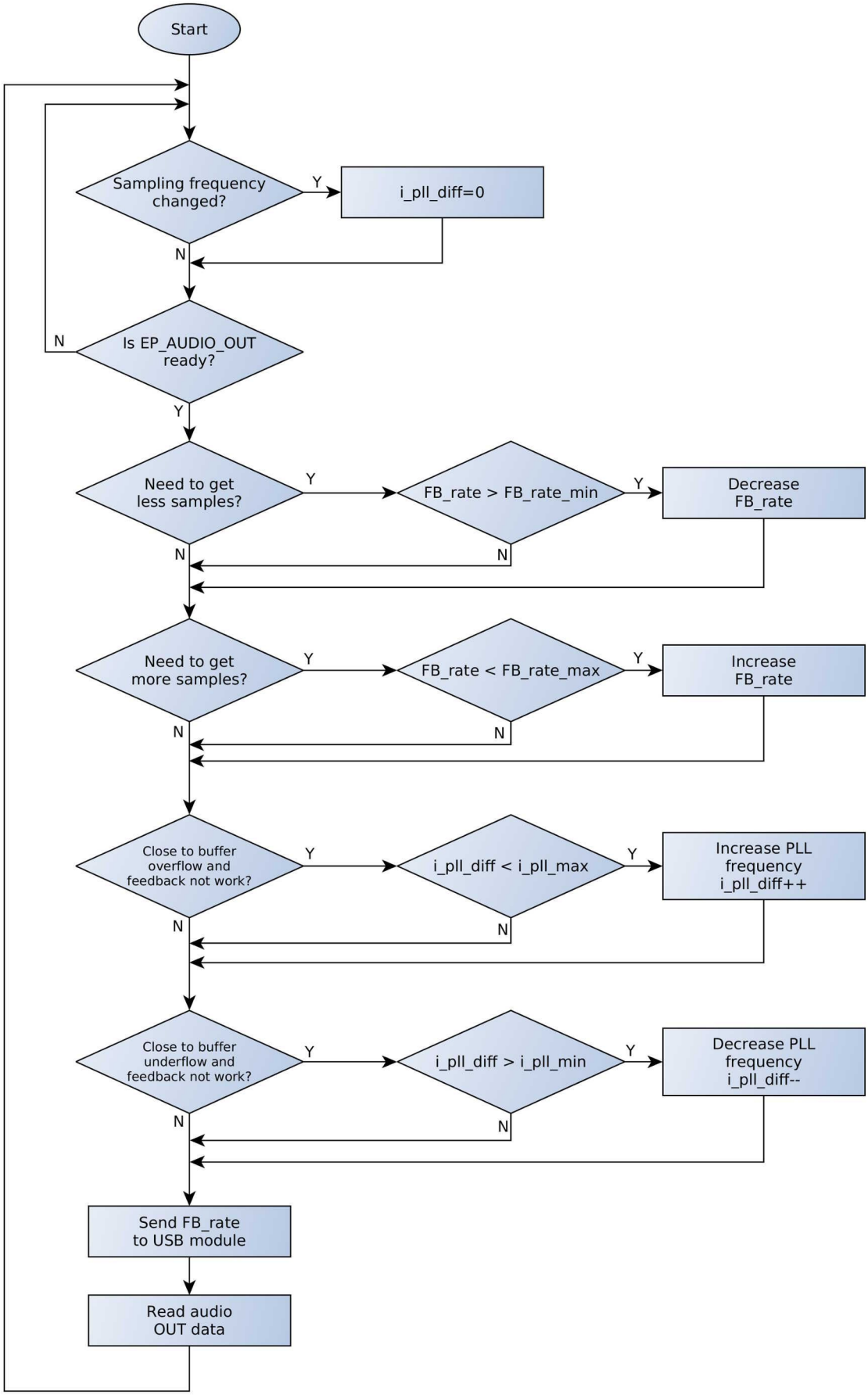
Funkce `uac1_device_audio_task()` běží jako nezávislý proces a v případě příchozích dat z USB modulu kopíruje audio data do patřičných audio bufferů. Zároveň sleduje zaplnění bufferů a provádí patřičné kroky, aby nedošlo k podtečení, nebo přetečení bufferů (problematika synchronizace je probrána podrobněji v kapitole 5.4). Při příjmu dat se vyvolá přerušení (blok INTC) a rutina nastaví PDCA periferii. Ta nastaví virtuální přepínač (switch). Respektive PDCA nastaví ukazatel na audio data, který pak využívá SSC jednotka. Z programátorského pohledu pracují periferie INTC a PDCA na pozadí.

5.4. Řešení synchronizace audio dat

Synchronizace audio dat je klíčová vlastnost požadovaného zařízení. Metody synchronizace jsou podrobně popsány v kapitole 2.3. Tato kapitola se zabývá konkrétním softwarovým řešením. Synchronizaci obou audio streamů (jak vstupních, tak výstupních) provádí funkce `uac1_devaice_audio_task()`. Pro přehlednost jsou zde dva vývojové diagramy: jeden pro vstupní data (Obr. 16) a druhý pro výstupní data (Obr. 17). Princip nastavení vzorkovací frekvence a dalších hodin je popsán v kapitole 5.3.



Obr. 16: Řešení synchronizace pro vstupní audio stream



Obr. 17: Řešení synchronizace pro výstupní audio stream

Synchronizace vstupních dat je jednodušší než u výstupních. Podle aktuální vzorkovací frekvence je nastaven odhad počtu vzorků a čeká se, dokud není endpoint EP_AUDIO_IN připraven. V momentě, kdy je EP připraven, se zkontroluje stav bufferů, zda se neblíží podtečení, nebo přetečení. Pokud ano, pak je upraven počet vzorků, který se bude odesílat. Samozřejmě že program může být rozšířen o sadu podmínek, které mohou počet vzorků měnit více dle potřeby, ale princip zůstává stejný.

Synchronizace výstupních dat byla v rámci vývoje prioritizována, a proto je její implementace komplexnější. Prvně se testuje, zda byla změněna vzorkovací frekvence. Následuje kontrola dostupnosti endpointu EP_AUDIO_OUT. Pokud je EP připraven, pak se testuje zaplnění bufferů a funkčnost zpětné vazby. Pokud zpětná vazba funguje, pak se pouze upraví hodnota, která říká, kolik dat má host příště přenést. Pokud je detekována nefunkčnost zpětné vazby (například kvůli špatné podpoře v OS), pak je v určitých limitech měněna frekvence PLL, od které se odvíjí MCLK, BCLK a FRAME_SYNC. Krok je přibližně 10 ppm.

Modul SSC potřebuje ke své funkci v podstatě jen BCLK. Signál FRAME_SYNC si je schopen generovat podle nastavení v registrech a MCLK nepotřebuje vůbec. Protože SSC jednotka se skládá z RX a TX modulu, bylo zapotřebí některé piny hardwarově spojit. To se týká RX_FRAME_SYNC, TX_FRAME_SYNC (společný FRAME_SYNC) a dvojice RX_CLOCK a TX_CLOCK (společný BCLK). Softwarově je TX modul SSC nastaven tak, že přijímá BCLK. Pokud je v roli master, tak generuje FRAME_SYNC, jinak signál přijímá a odesílá data na pin TX_DATA. Modul RX pouze přijímá BCLK, FRAME_SYNC a RX_DATA. Kvůli flexibilnosti celé aplikace bylo zařízení navrženo tak, že směr (I^2S konektor → MCU ; MCU → I^2S konektor ; Hi-Z) signálů MCLK, BCLK, FRAME_SYNC, MUTE, RESET_I2S může být nastaven na libovolnou kombinaci. To je opět výhodné pro různé případy, kdy zařízení nepracuje čistě jako master, nebo slave.

Přehled jednotlivých scénářů je v Tab. 8. Pro větší přehlednost byl vstupní směr označen jako „I^2S konektor → MCU“ a výstupní pak „MCU → I^2S konektor“. Kvůli možné kombinaci jsou v popisu vždy všechny situace, které mohou nastat a případně uvedeny reference na dané signály. Signály MUTE a RESET_I2S slouží především pro ovládání připojených I^2S zařízení a jsou volitelné.

Tab. 8: Možnosti nastavení směru signálů a jejich vlastnosti

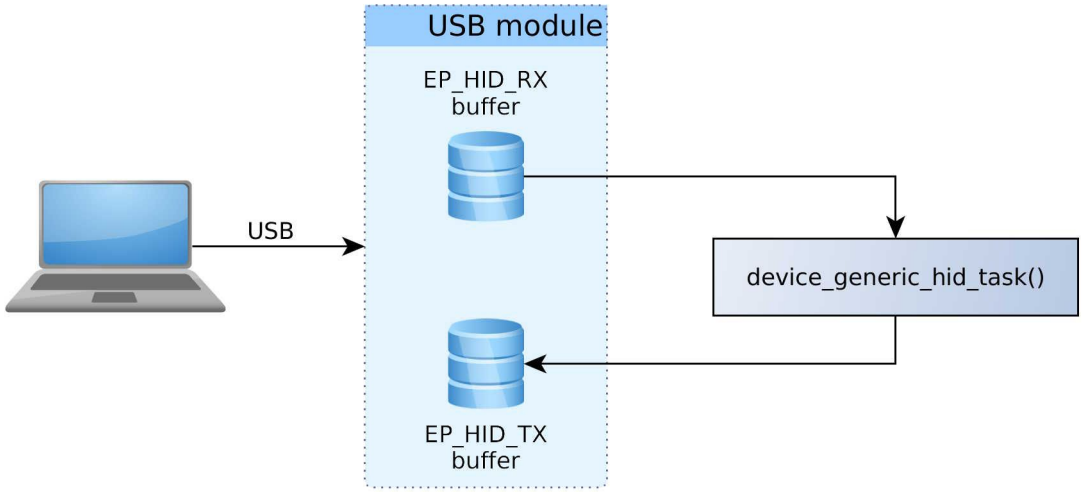
Signál	Směr	Popis
MCLK	I^2S konektor → MCU	Pouze kodek pro kontrolní odposlech vyžaduje korektní MCLK. Pokud MCLK není k dispozici, nebo nebude korektní, pak to neovlivní komunikaci mezi počítačem a zařízením.
	MCU → I^2S konektor	Signál je generován pomocí externí PLL a děličky GCLK[0]. V případě, že nefunguje zpětná vazba pro nastavení počtu vzorků pro hosta, je frekvence automaticky měněna.
	Hi-Z	Signál je generován, takže kodek pro kontrolní odposlech funguje, ale na straně I^2S konektoru je nastavena vysoká impedance. V případě, že nefunguje zpětná vazba pro nastavení počtu vzorků pro hosta, je frekvence automaticky měněna.
BCLK	I^2S konektor → MCU	Signál z konektoru musí mít správnou frekvenci. Pokud je TX modul v SSC nastaven jako master, pak generuje FRAME_SYNC z BCLK. Zařízení je schopné

Signál	Směr	Popis
		v určitých mezích řešit nepřesnost tohoto signálu. U vstupního audio streamu zařízení provádí korekce tak, že mění počet vzorků, které budou odeslány hostu. Pokud u výstupního streamu nefunguje zpětná vazba, pak jediná možná korekce od zařízení je změna MCLK za předpokladu, že MCLK je nastaven jako výstupní. Pokud MCLK není nastaven jako výstupní a zpětná vazba není funkční, může po určitém časovém intervalu dojít k podtečení, nebo přetečení bufferů.
	MCU → I²S konektor	Signál je generován pomocí externí PLL a děličky GCLK[1]. V případě, že nefunguje zpětná vazba pro nastavení počtu vzorků pro hosta, je frekvence automaticky měněna.
	Hi-Z	Signál je generován, takže kodek pro kontrolní odposlech funguje, ale na straně I²S konektoru je nastavena vysoká impedance. V případě, že nefunguje zpětná vazba pro nastavení počtu vzorků pro hosta, je frekvence automaticky měněna.
FRAME_SYNC	I²S konektor → MCU	Signál z konektoru musí mít správnou frekvenci. Zařízení je schopné v určitých mezích řešit nepřesnost tohoto signálu. U vstupního audio streamu zařízení dle potřeby upraví počet odesílaných vzorků směrem k hostu. Pokud u výstupního streamu nefunguje zpětná vazba, pak jediná možná korekce od zařízení je změna MCLK a BCLK za předpokladu, že MCLK, nebo BCLK je nastaven jako výstupní. Pokud MCLK nebo BCLK není nastaven jako výstupní a zpětná vazba není funkční, může po určitém časovém intervalu dojít k podtečení, nebo přetečení bufferů.
	MCU → I²S konektor	Signál je generován z BCLK.
	Hi-Z	Signál je generován z BCLK, takže kodek pro kontrolní odposlech funguje (za předpokladu že BCLK a MCLK je korektní), ale na straně I²S konektoru je nastavena vysoká impedance.
MUTE	I²S konektor → MCU	Pokud je v nízké úrovni, funkce mute je vypnuta, jinak je funkce mute zapnuta. Čtení z pinu MUTE je aktivní pouze v případě, že I²S konektor je napájen. V případě, že je stisknuto tlačítko MUTE, funkce mute je zapnuta.
	MCU → I²S konektor	Signál je řízen úrovní z tlačítka MUTE.
	Hi-Z	Signál MUTE je na straně I²S konektoru ve vysoké impedanci. V případě, že je stisknuto tlačítko MUTE, funkce mute je zapnuta.
RESET_I2S	I²S konektor → MCU	Pokud je ve vysoké úrovni, provede se restart I²S konektoru (tzn. výchozí nastavení).

Signál	Směr	Popis
	MCU → I²S konektor	Signál je řízen úrovní z tlačítka RESET_I2S.
	Hi-Z	Signál RESET_I2S je na straně I²S konektoru ve vysoké impedanci. V případě, že je dlouze stisknuto tlačítko RESET_I2S, provede se restart I²S konektoru (tzn. výchozí nastavení).

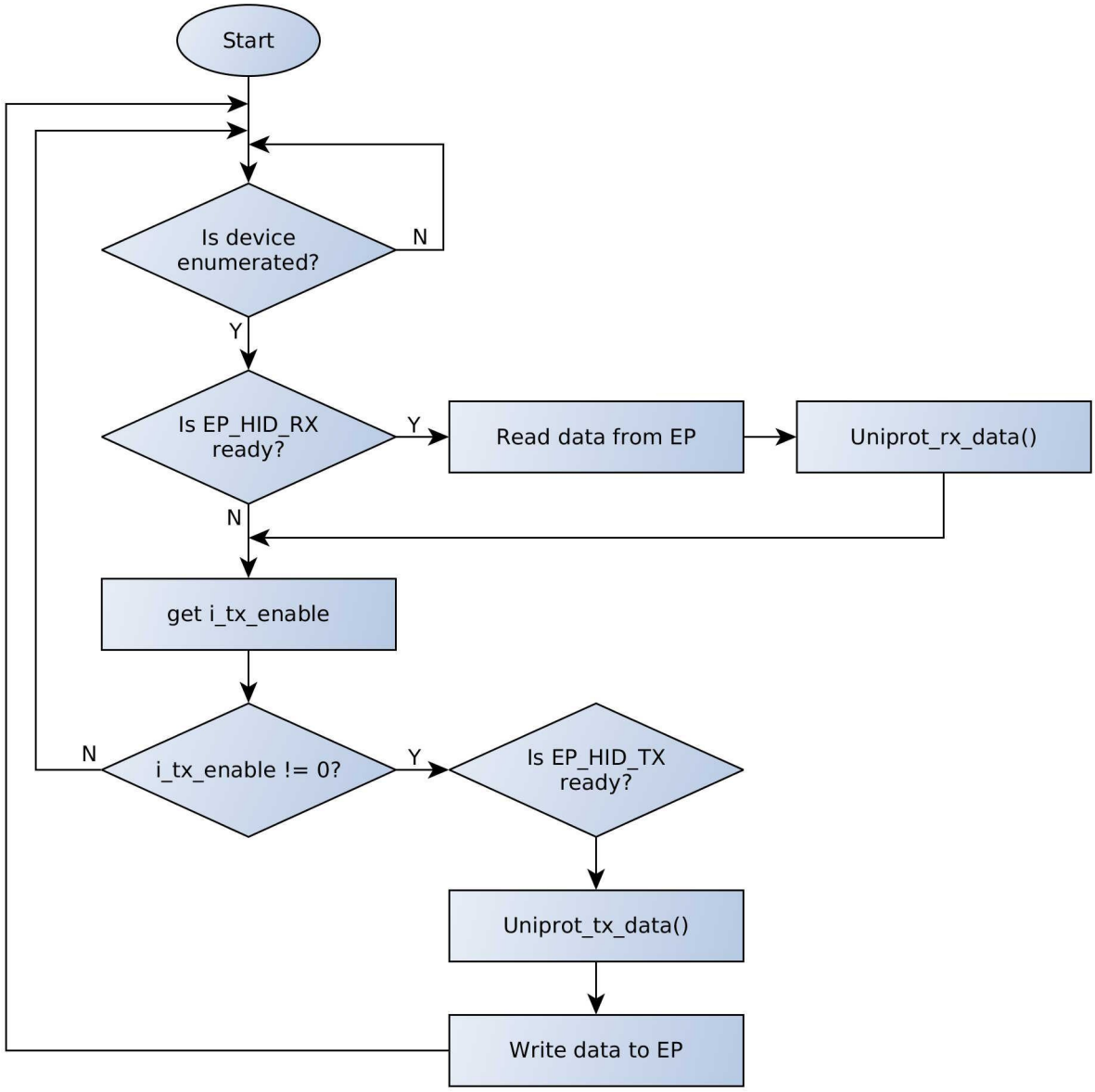
5.5. Tok HID dat

Třída HID je relativně jednoduchá jak na implementaci, tak na obsluhu a zároveň je standardem prakticky ve všech OS. Objemy přenesených dat jsou mnohem menší ve srovnání s audio daty, takže i samotné zpracování může být čistě na softwarové úrovni. Zjednodušený diagram je na Obr. 18.



Obr. 18: Diagram pro přenos USB HID dat

Samotná funkce `device_generic_hid_task()` je velmi jednoduchá. Vývojový diagram je uveden na Obr. 19.



Obr. 19: Vývojový diagram funkce `device_generic_hid_task()`

Nejprve se kontroluje, zda je zařízení, resp. USB modul, enumerován. Pokud ano, testuje se, zda nebyla přenesena nějaká data od hosta. Pokud ano, pak jsou vyčteny a předány funkci `Uniprot_rx_data()`, která data zpracuje. Následuje získání hodnoty i_tx_enable, která poté rozhoduje, zda se budou odesílat data směrem k hostu, nebo ne. Pokud se mají odeslat data, musí se zkontrolovat, zda je EP připraven. Pokud je i tato podmínka splněna, pak je volána funkce `Uniprot_tx_data()`, které má za úkol připravit data k odeslání. Následně jsou data zapsána do EP. Funkce `Uniprot_rx_data()` a `Uniprot_tx_data()` jsou součástí jedné z vrstev komunikačního protokolu, který je popsán v kapitole 6.2.

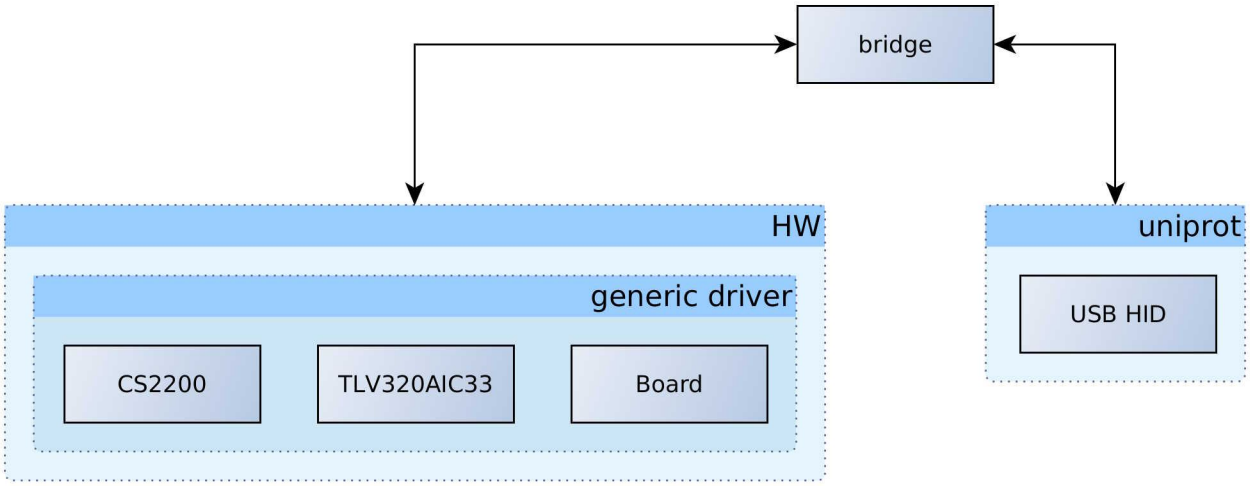
6. Vrstvy protokolu pro nastavení zařízení

Řídící a konfigurační protokol pro tento projekt byl navržen a z velké části implementován zcela obecně. Cílem bylo vytvořit opakovatelně použitelný systém, tj. protokol a knihovny pro zařízení s MCU a pro PC aplikaci. Tento systém má sloužit pro nastavování jakýchkoliv zařízení přes běžně používaná rozhraní. Tedy nejenom přes logické spojení v rámci USB HID, ale i např. přes UART apod.

Obecné řešení je inspirováno ISO/OSI komunikačním modelem a realizováno pomocí vrstev. Výsledkem je flexibilní software, který umožňuje velmi rychlé přidávání nových konfiguračních možností do zařízení s MCU, aniž by bylo potřeba jakkoliv modifikovat konfigurační PC aplikaci.

Praktičnost konceptu byla ověřena portací protokolu do jiného projektu mimo rámec této práce, kdy se s minimální námahou podařilo implementovat konfigurační rozhraní pro jiný projekt s jinými parametry a na jiné MCU architektuře. Pro konfiguraci bylo přitom možno používat stejnou PC aplikaci.

Samozřejmě toto řešení má i své nevýhody. Mezi největší nevýhody patří zejména potřeba vyšších výpočetního výkonu. Jak bylo naznačeno v předchozí kapitole, tento problém může efektivně řešit RTOS kvůli vhodnému nastavení priorit. Schématické rozvrstvení celého systému je na Obr. 20.



Obr. 20: Systém komunikace jednotlivých vrstev

Nejnižší vrstva jsou ovladače k samotnému hardwaru, který má MCU řídit. Jako příklad jsou zde ovladače k PLL cs2200, kodeku TLV320AIC33 a ovladač pro desku jako takovou (LED, tlačítka atd.). Každý ovladač, který má využívat vrstva „generic driver“, musí tuto vrstvu podporovat.

Samotná vrstva „generic driver“ umí kvůli metadatům uložených v ovladači poskytnout vyšším vrstvám potřebné informace o funkcích, které ovladač nabízí. Tato vrstva však požaduje jako jeden z argumentů ukazatel na adresu, kde se metadata daného ovladače nacházejí.

To řeší vrstva „HW“, která je jakýsi virtuální správce hardwaru. Tato vrstva zná adresy k metadatům jednotlivých ovladačů a umožňuje vyšším vrstvám přistupovat k jednotlivým ovladačům pomocí indexu. To je výhodné, pokud je zapotřebí využívat více zařízení, která jsou připojena k MCU.

Vrstva „uniprot“ má za úkol bezpečně přijímat a odesílat data. Tato vrstva umožňuje pracovat

s několika datovými toky, ale v této aplikaci je využit pouze jeden a to USB HID. V principu je jedno, jestli jsou data přenášena přes USB, UART, I^2C nebo SPI. Vrstva „uniprot“ předává vrstvám již celé pakety dat.

Propojení mezi vrstvami „HW“ a „uniprot“ řeší vrstva „bridge“. Tato vrstva udává, jak budou datové pakety vypadat. Je totiž potřeba definovat, jakým způsobem bude identifikován konkrétní požadavek a konkrétní odpověď. Tato vrstva abstrahuje „HW“ a posílá surová data pomocí vrstvy „uniprot“.

Všechny vrstvy byly napsány tak, aby podporovaly jak architekturu AVR8, tak AVR32. Teoreticky by kód měl být funkční i na dalších architekturách, ale z časových důvodů nebylo možné vyzkoušet další architektury.

6.1. Generic driver a HW

Soubor metod, které umožňují přistupovat k libovolnému ovladači (který podporuje „generic driver“) předdefinovaným způsobem. Výhodou je, že ačkoliv ovladač může obsluhovat různá zařízení, přístup k nim je vždy stejný.

Každý ovladač, který má podporovat „generic driver“, musí obsahovat tabulku podporovaných příkazů a metadata k této tabulce. Tabulka podporovaných příkazů je jen pole struktur. Jako příklad je zde část kódu z ovladače pro externí PLL:

```
const gd_config_struct CS2200_config_table[]
  {
    {
      0,                                          // Command ID
      "Initialize CS2200 hardware",               // Name
      "Initialize I/O and TWI module (if used)",  // Descriptor
      void_type,                                  // Input data type
      {.data_uint32 = 0},                         // Minimum input value
      {.data_uint32 = 0},                         // Maximum input value
      void_type,                                  // Output data type
      {.data_uint32 = 0},                         // Minimum output value
      {.data_uint32 = 0},                         // Maximum output value
      (GD_DATA_VALUE*)&gd_void_value,             // Output value
      cs2200_init                                 // Function, that should be called
    },
    {
      1,
      "Set PLL frequency",
      "Frequency format: unsigned 32 bit in Hz",
      uint32_type,
      {.data_uint32 =  6000000},        // Min 6 MHz
      {.data_uint32 = 75000000},        // Max 75 MHz
      uint32_type,
      {.data_uint32 =  6000000},
      {.data_uint32 = 75000000},
      (GD_DATA_VALUE*)&s_virtual_reg_img.i_real_freq.i_32bit,
      cs2200_set_PLL_freq
    },
    ...
```

První položka je číslo příkazu. Ve své podstatě je to zároveň i index v poli, takže se jeho informace může jevit jako redundantní. Pokud je vše v pořádku, pak se jedná skutečně o redundantní informaci. Ale v momentě, kdy je některý ukazatel špatně nastaven, za pomocí této informaci je možné velice rychle najít chybu. Je to obdoba CRC součtu. Jedná se o redundantní informaci, dokud je vše v pořádku.

Následuje název funkce. Jedná se pole znaků, jehož maximální délka je definována v souboru *generic_driver.h* jako *GD_MAX_STRING_SIZE*. Jeho hodnotu lze měnit v závislostech na požadavcích aplikace, ale doporučená hodnota je 80. To znamená 79 viditelných znaků a znak null.

To je pro většinu případů dostatečná rezerva a zároveň je to dobrý kompromis mezi popisem funkce a náročnosti na RAM MCU.

Třetí položka je popis funkce. Délka popisu je opět definována jako *GD_MAX_STRING_SIZE*. Zde je vhodné popsat, co funkce dělá, v jakých jednotkách jsou vstupní hodnoty, nebo jakých hodnot může nabývat (např: „0 – input ; 1 – output“).

Jako další v pořadí je datový typ pro vstupní parametr funkce. Přehled datových typů je v Tab. 9.

Tab. 9: Přehled datových typů používaných "generic driverem"

Datový typ	Popis
void_type	Prázdný datový typ. Volaná funkce nezpracovává žádný vstupní argument.
char_type	Jeden 8bitový ASCII znak.
int_type	Znaménkové celé číslo. U většiny architektur definováno jako 32bitové, ale například u AVR8 je 16bitové. Z tohoto důvodu není doporučeno tento datový typ používat. Na straně počítače jej aplikace zpracovává vždy jako 32bitové!
int8_type	Znaménkové 8bitové číslo.
int16_type	Znaménkové 16bitové číslo.
int32_type	Znaménkové 32bitové číslo.
uint_type	Neznaménkové celé číslo. U většiny architektur definováno jako 32bitové, ale opět není podmínkou. Proto není doporučeno tento datový typ používat. Na straně počítače jej aplikace zpracovává vždy jako 32bitové.
uint8_type	Neznaménkové 8bitové číslo.
uint16_type	Neznaménkové 16bitové číslo.
uint32_type	Neznaménkové 32bitové číslo.
float_type	Třiceti dvou bitové číslo s plovoucí desetinnou čárkou.
group_type	Definuje zvláštní příkaz, který je množinou několika funkcí, a uživatel pak vybírá jednou z nich.

Následuje minimální a maximální hodnota pro argument funkce. Pokud funkce nepotřebuje žádný argument, je vhodné nastavit obě hodnoty jako 32bitové nuly.

Jako další je zde datový typ pro výstupní hodnotu. Zde jsou možnosti stejné jako pro argument funkce – viz. Tab. 9.

I výstupní hodnota by měla mít definované své limity, takže další dvě hodnoty udávají minimální a maximální přípustnou hodnotu výstupní proměnné. Pokud funkce neposkytuje žádnou výstupní hodnotu, je vhodné nastavit oba limity jako 32bitové nuly.

Jako předposlední je zde ukazatel (adresa) na proměnnou, kam je uložena výsledná výstupní hodnota funkce. Ukazatel proto, aby celá tabulka mohla být jen výčet konstant, a tím pádem mohla být uložená v paměti programu a ne v RAM, které bývá v MCU nedostatek.

Poslední hodnota je ukazatel na funkci, která má být volána.

Každá tabulka musí mít definovaná metadata, prostřednictvím nich je možné tabulku

jednoznačně identifikovat. Metadata nejsou nic jiného než struktura, která obsahuje následující konstanty:

- počet příkazů v tabulce,
- textový popis ovladače (maximální délka definována *GD_MAX_STRING_SIZE*),
- ukazatel na první prvek tabulky,
- sériové číslo (8bitové).

Díky znalosti celkovému počtu příkazů v tabulce je možné provádět scan zařízení (tzn. získat seznam funkcí, které nabízí). Textový popis je pro uživatele, aby věděl, který ovladač právě používá. Protože struktura tabulky je předem daná a její délka je také známa (je definován celkový počet příkazů), stačí znát adresu prvního prvku v tabulce a další se dají velice jednoduše dopočítat. Sériové číslo je tu například pro rozeznání dvou hardwarově stejných desek, nebo pro odlišení jednotlivých hardwarových revizí. Příklad metadat je na následujících řádcích:

```
const gd_metadata CS2200_metadata =
{
        CS2200_MAX_CMD_ID,                          // Max CMD ID
        "Fractional PLL CS2200-CP v0.7",            // Description
        (gd_config_struct*)&CS2200_config_table[0], // Pointer to table
        0x21                                        // Serial number (0~255)
};
```

Samotný „generic driver“ nabízí pouze dvě funkce: `gd_get_setting()` a `gd_set_setting()`. Funkce `gd_get_setting()` zjistí veškeré parametry funkce (datové typy, rozsah hodnot, ukazatel na výstupní hodnotu, ukazatel na funkci) na základě ukazatele na metadata a ID příkazu. Funkce `gd_set_setting()` zavolá funkci. K tomu potřebuje platný ukazatel na metadata, ID příkazu a hodnotu argumentu.

Jak bylo popsáno výše, vrstva „HW“ je jen abstrakcí vrstvy „generic driver“. Zároveň je jakýmsi virtuálním správcem zařízení, resp. jejich ovladačů. Protože ne každé zařízení musí být nutně přístupné z pohledu konfigurace (některé zařízení je vhodné zpřístupnit např. jen pro debug fázi projektu), je na uživateli, aby v souboru *HW.h* definoval, které ovladače bude tato vrstva spravovat. Definice je velmi snadná:

```
/// \brief Board driver
#define DEVICE0         BRD_DRV_metadata
#include "brd_driver_hw_03.h"

/// \brief TLV320AIC33 codec
#include "tlv320aic33.h"
#define DEVICE1         TLV320AIC33_metadata

/// \brief Fractional PLL CS2200
#include "cs2200.h"
#define DEVICE2         CS2200_metadata
```

Nejprve musí být definováno symbolické jméno pro zařízení. Je nutné, aby symbolické jméno bylo ve formátu „DEVICEx“, kde „x“ je index použitého ovladače, které pak používá tato a všechny vyšší vrstvy. Následuje jméno proměnné, ve které jsou uložena metadata. Pak už jen zbývá přidat include s patřičným souborem.. Analogicky se definují další zařízení. Aktuální verze umožňuje takto „připojit“ až deset zařízení a rozšíření je jen otázkou drobné modifikace v hlavičkovém souboru.

Aby byl celkový koncept dostatečně flexibilní, byla vytvořena funkce, která vrátí index posledního zařízení – `hw_get_max_device_index()`. To umožní aplikaci na počítači proskenovat všechna zařízení obsažená v „HW“. Protože jednotlivá zařízení jsou rozlišována indexem, je vhodné dát vyšším vrstvám možnost získat metadata. To je možné pomocí funkce

`hw_get_device_metadata()`, která jako argument používá index zařízení. Funkce `hw_get_setting()` a `hw_set_setting()` jsou obdobou `gd_get_setting()` a `gd_set_setting()`, které jsou vysvětleny výše. Jediný rozdíl je ve vstupních parametrech. Funkce z vrstvy „HW“ používají pro identifikování zařízení indexy, zatímco funkce z „generic driver“ používají přímo metadata.

6.2. Universal protocol

Tato vrstva má za úkol zabezpečit a přenést data pro vyšší vrstvu. Aktuální verze umožňuje, aby „universal protocol“ obsluhoval až 7 nezávislých datových streamů z různých periferií. Rozšíření počtu je jen otázkou několika drobných modifikací v kódu a hlavičkovém souboru. Každý datový stream používá virtuální kanál, který je pak značen PIPE0 až PIPE6 (pro 7 zařízení). V rámci vrstveného modelu tento protokol sám o sobě neobsahuje žádné ovladače k periferiím pro datovou komunikaci, pouze nabízí obslužné funkce. Ostatně ani není cílem, aby podporoval různé periferie nativně, protože tím by se kód stal nepřehledným a zároveň by byl těžko přenositelný. Namísto toho musí být upraven ovladač pro konkrétní periferii, ale modifikace je jednoduchá, takže ve většině případů to neznamená, že by ovladač musel být přepsán od začátku. Níže je naznačen postup s příkladem pro PIPE0 (tzn. je použit pouze jeden kanál):

1) Úprava hlavičkového souboru *uniprot.h.* Je nutné includovat hlavičkový soubor ovladače.

   ```
   #include „device_generic_hid.h“
   ```

2) Pokud ovladač nepodporuje „universal protocol“, pak v místě, kde ovladač přijímá data, je potřeba vložit funkci `Uniprot_rx_data()`, které jsou data předána. V místě, kde dochází k odesílání dat, je situace komplikovanější. Nejprve musí být volána funkce `Uniprot_Is_TX_transmission_enable()`. Pokud tato funkce vrátí 1, pak je možné zavolat `Uniprot_tx_data()`, která naplní požadovaný buffer daty. Poté může ovladač začít vysílat.

3) Pokud ovladač podporuje „universal protocol“, tak v jeho hlavičkovém souboru by měly být definovány UNI_PIPE0_FRAME_SIZE (udává kolik Bajtů ovladač přenese najednou), UNI_PIPE0_FUNC_INIT (definuje jméno inicializační funkci pro ovladač) a UNI_PIPE0_FUNC_TASK (definuje jméno tasku pro ovladač). Pokud v jeho hlavičkovém souboru nejsou, pak je možné je dodefinovat v souboru *uniprot.h*.

   ```
   /// Define frame size (number of bytes per one transmission)
   #define UNI_PIPE0_FRAME_SIZE    8
   /// Initialize function which will be called in initialization process
   #define UNI_PIPE0_FUNC_INIT     \
     device_generic_HID_init(UAC1_EP_HID_RX, UAC1_EP_HID_TX)
   /// Driver task
   #define UNI_PIPE0_FUNC_TASK     \
     device_gneric_hid_task()
   ```

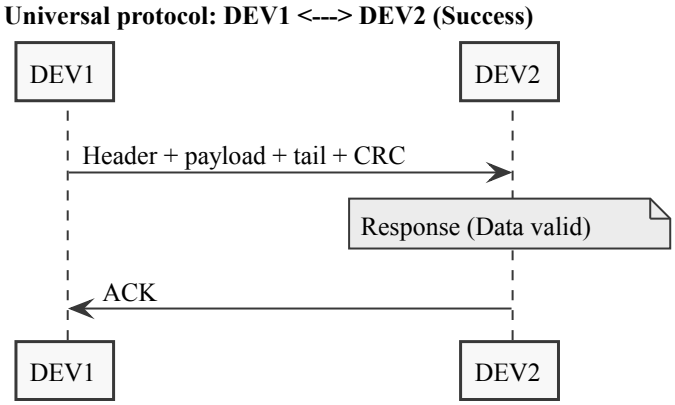
V případě čtení vyšší vrstva musí nejprve nakonfigurovat příchozí paket. To se provede pomocí `Uniprot_config_RX_packet()`. Pak jen čeká až funkce `Uniprot_Is_RX_pipe_ready()` vrátí 1. V ten moment jsou data v bufferu platná.

Odesílání dat je velice podobné. Nejprve se musí naplnit buffer daty a nakonfigurovat odchozí paket pomocí `Uniprot_config_TX_packet()`. Pak už jen zbývá čekat, až funkce `Uniprot_Is_TX_pipe_ready()` vrátí 1.

6.2.1. Základní scénáře při komunikaci

Za ideálních podmínek by se žádná data neztrácela a celý „universal protokol“ by pouze parsoval data. V reálném prostředí je ale ztráta dat vždy možná. Ať už je to hardwarem (špatný

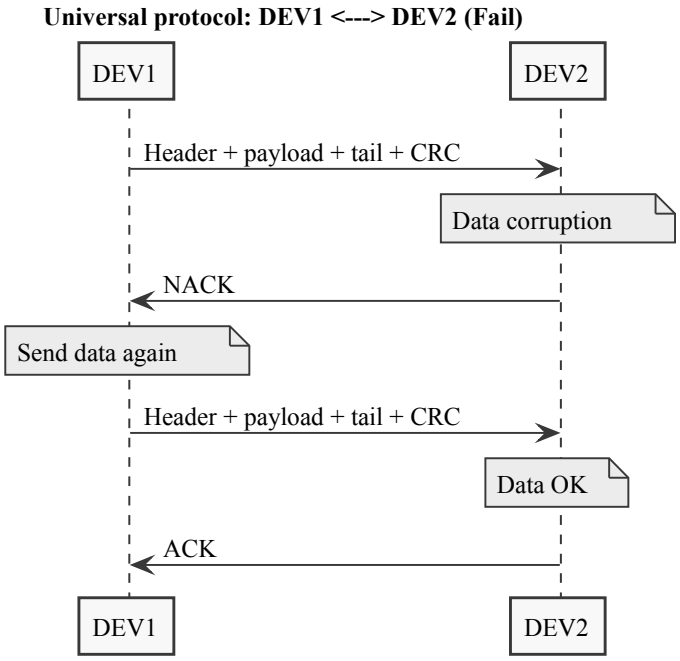
kontakt), nebo na straně OS (přetečení bufferu kvůli pomalému multitaskingu apod.). Proto se musí řešit různé výjimky, které mohou nastat. Na Obr. 21 je zobrazen přenos dat bez potíží. Veškeré další diagramy platí pro obecné zařízení DEV1 a DEV2. Není tedy pevně dáno, které zařízení je master a které slave. Tato vrstva řeší pouze přenos dat.



Obr. 21: Úspěšný přenos dat

DEV1 odešle datový paket (o určité struktuře) a DEV2 přijme data. Proběhne kontrola data (CRC). V tomto případě je vše v pořádku a DEV2 odešle příkaz „ACK“ a informuje vyšší vrstvu o úspěšném přijetí dat. Poté co DEV1 přijme „ACK“ je vyšší vrstva informována o úspěšné přenosu dat.

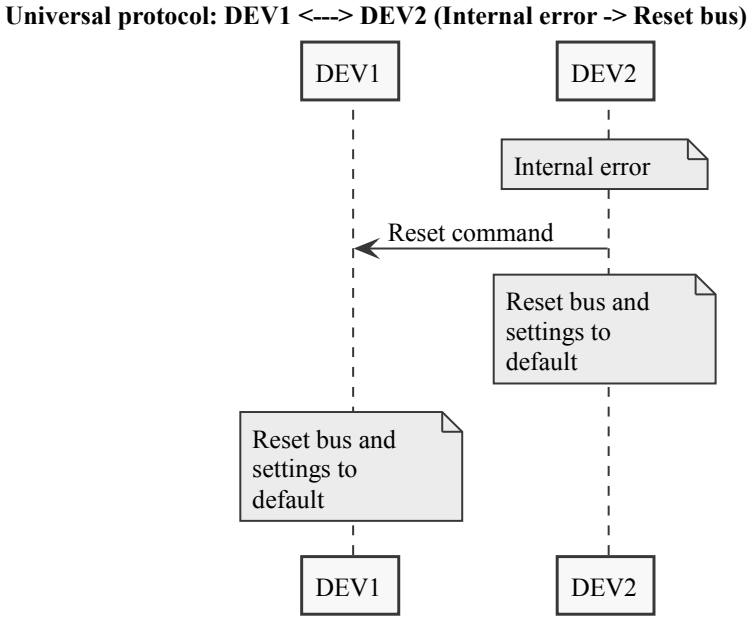
Na Obr. 22 je případ, kdy jsou data nějakým způsobem poškozena.



Obr. 22: Řešení poškození dat

Po příjmu dat bylo pomocí CRC detekováno jejich poškození. DEV2 vyšle příkaz „NACK“. DEV1 přijme příkaz a vyšle stejná data znovu. Podruhé jsou již data v pořádku a DEV1 odesílá „ACK“. Vyšší vrstvy na obou stranách jsou informovány o úspěšné operaci. V případě, že ani podruhé se data nepřenesou v pořádku, opakuje se celý proces, dokud se nedosáhne definovaného

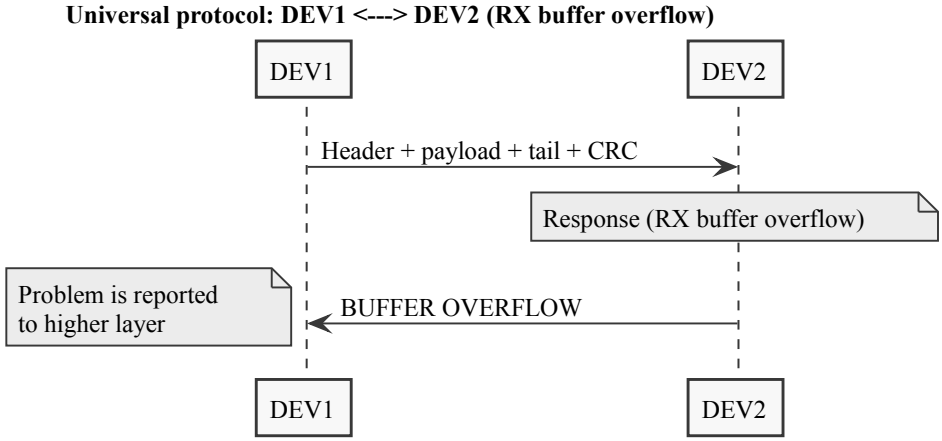
počtu pokusů. Tento počet pokusů není pevně nastavený a je na programátorovi, aby zvolil vhodné číslo v závislost na tom, po jaké sběrnici jsou data odesílána. Pokud je tento počet překročen, vyvolá se interní chyba, která způsobí reinicializaci ovladačů a vrstvy „universal protokolu“. Tento proces je na Obr. 23.



Obr. 23: Resetování ovladačů a "universal protokolu"

Pokud nastane tato situace, pak to značí, že chybovost přenosového kanálu je vysoká. Jako poslední možnost zotavení se pak jeví reinicializovat ovladače a „universal protokol“ samotný. Pokud je indikována interní chyba, pak zařízení, které chybu detekuje, odešle příkaz „RESET“. Poté se obě zařízení reinicializují. Pokud by příkaz „RESET“ nebyl doručen DEV1, pak by při další komunikaci nastala chyba na straně DEV1 a to by odeslalo příkaz „RESET“ a samo by se reinicializovalo. Poté by měly být obě strany připraveny pro další komunikaci.

Dále se může objevit problém s příchozími daty. Problém je nastíněn na Obr. 24.



Obr. 24: Signalizace přetečení vstupního bufferu

V tomto scénáři poslal DEV1 více dat, než na kolik byl DEV2 připraven. Aby nedošlo

k přetečení vstupních bufferů (a přepisu náhodných dat v RAM), jsou veškerá nadbytečná data ihned zahazována. Poté DEV2 odešle odpověď „BUFFER OVERFLOW“. Co se týče protokolu, tak zde žádná chyba není. Chyba je na straně vyšší vrstvy, která buďto špatně nakonfigurovala vstupní paket (DEV2), nebo nerespektuje hardwarová omezení druhé strany (v tom případě je chyba na vyšší vrstvě u DEV1).

Chyb může v reálné situaci nastat více. Většinu z nich je výhodné řešit na úrovni aplikace pro PC, kde lze pracovat ve vyšším programovacím jazyce a kde se prakticky nemusí řešit nároky na paměť, nebo výpočetní výkon. Zde uvedené případy ale pokrývají drtivou většinu všech v praxi se vyskytujících problémů.

6.2.2. Struktura paketu

Protože data mohou být odesílána ve shlucích různých velikostí, je nutné používat předdefinovanou strukturu paketů. Jak bylo naznačeno výše, protokol odesílá dva typy paketů: datové a příkazové. Struktura datového paketu je na Obr. 25.

Pořadí Bajtu	0	1	2	3	-	N+4	N+5	N+6	
	H	Num. of Bytes (H)	Num. of Bytes (L)	Data begin	Payload	Tail	CRC16 (H)	CRC16 (L)	Dummy data
	„H“ (0x48)	0xXX	0xXX	„D“ (0x44)	0xYYyy ...	„T“ (0x54)	0xZZ	0xZZ	0xFF

Obr. 25: Data paket

Jako první Bajt je symbol „H“ (ASCII hodnota 0x48), který symbolizuje počátek hlavičky. Následují dva Bajty, které definují délku payloadu, užitečných dat, jako počet Bajtů. Jako další je přenesen znak „D“ (ASCII hodnota 0x44), který říká, že následují užitečná data o délce N Bajtů. Protože je tato informace na předem definovaném místě za symbolem „H“, je možné velice rychle ověřit (s určitou pravděpodobností), zda se jedná o počátek paketu, či nikoliv. Díky symbolu „T“ je možné velice snadno odhalit, zda byla všechna data přijata, nebo ne. Pokud byla data přijata, může se ověřit CRC součet (2 Bajty). Pokud se odesílají data po více než jednom Bajtu, může se stát, že ne všechen paměťový prostor by byl využit. Proto je vhodné (ale není podmínkou), aby data v tomto prostoru byla definována.

Struktura příkazového paketu je mnohem jednodušší. Viz. Obr. 26.

Pořadí bajtů	0	1	2	
	CMD	CRC16 (H)	CRC16 (L)	Dummy data
	0xXX	0xZZ	0xZZ	0x00

Obr. 26: Příkazový paket

Jako první Bajt je tzv. příkazový symbol. Ten určuje o jaký příkaz, resp. zprávu jde. Seznam těchto symbolů je v Tab. 10. Následuje zabezpečení CRC. Opět je možné, že některé Bajty by mohly být nedefinované. Zde opět není nutností tato data definovat, ale je to vhodné.

Tab. 10: Seznam příkazových symbolů

ASCII symbol	Význam
A	ACK – potvrzení přijetí dat. Vše v pořádku.
N	NACK – data byla přijata, ale jsou poškozena. Poslat data znovu.
R	RESET – je nutné reinicializovat ovladač a protokol samotný.
O	BUFFER OVERFLOW – data nemohla být přijata. Příliš mnoho dat.

6.3. Bridge

Virtuální most mezi vrstvou „HW“ a „universal protocol“. Tato vrstva má za úkol obsluhovat veškerý hardware pomocí předdefinovaných příkazů. Aktuální verze podporuje 4 základní příkazy, které se ale ukázaly jako naprosto dostačující. Celá komunikace funguje tak, že aplikace na počítači pošle jeden z příkazů a AVR pošle zpět odpověď. Seznam příkazů a jejich popis je v Tab. 11.

Tab. 11: Popis příkazů

Příkaz	Popis
Get setting	Získá všechny potřebné informace o vybrané funkci
Set setting	Zavolá vybranou funkci s parametrem (pokud jej funkce zpracovává)
Get metadata	Získá metadata vybraného zařízení
Get number of devices	Navrátí index posledního zařízení

Pro zjednodušení popisu jednotlivých příkazů jsou v Tab. 12 vysvětleny zkratky, které jsou použity níže.

Tab. 12: Seznam zkratek pro vrstvu „bridge“

Zkratka	Minimální hodnota	Maximální hodnota	Velikost v Bajtech	Popis
DID	0	0xFF	1	ID zařízení. Tato hodnota odpovídá indexu používaném vrstvou „HW“.
BCMD	0	0xFF	1	Příkaz vrstvy „bridge“.
CMD ID	0	0xFFFE	2	ID příkazu pro zvolené zařízení.
RES CODE	0	0xFF	1	Návratová hodnota. Pokud je vše v pořádku, pak vrátí 0.
IN TYPE	0	10	1	Datový typ pro argument funkce. Definován vrstvou „generic driver“.
IN MIN	0x00000000	0xFFFFFFFF	4	Minimální hodnota pro argument funkce.
IN MAX	0x00000000	0xFFFFFFFF	4	Maximální hodnota pro argument funkce.
IN VALUE	IN MIN	IN MAX	4	Argument pro funkci, které bude volána.
OUT TYPE	0	10	1	Datový typ pro výsledek zpracovaný funkcí. Definován vrstvou „generic driver“.
OUT MIN	0x00000000	0xFFFFFFFF	4	Minimální hodnota výsledku.
OUT MAX	0x00000000	0xFFFFFFFF	4	Maximální hodnota výsledku.
OUT VALUE	OUT MIN	OUT MAX	4	Výstupní hodnota z volané funkce.
NAME	Nedefinováno	Nedefinováno	Nedefinováno	Jméno funkce jako pole symbolů. Musí být zakončeno hodnotou null (0x00).

Zkratka	Minimální hodnota	Maximální hodnota	Velikost v Bajtech	Popis
DESC	Nedefinováno	Nedefinováno	Nedefinováno	Popis funkce jako pole symbolů. Musí být zakončeno hodnotou null (0x00).
SERIAL NUM	0	0xFF	1	Sériové číslo používaného ovladače.
NUM OF DEVICES	0	0xFF	1	Index posledního zařízení (resp. ovladače) které má „HW“ obsluhovat.

V následujících podkapitolách jsou uvedeny formáty příkazů a jejich odpovědí. Data jsou odesílána a přijímána od MSB po LSB. MSB je vždy uváděno jako první.

6.3.1. Get setting

Formát příkazu je na Obr. 27. Hodnota BCMD je fixní: 0x01. Jak bylo uvedeno výše, hodnota BCMD určuje, o jaký příkaz vrstvy „bridge“ se jedná. Formát odpovědi je pak na Obr. 28. Pokud je hodnota RES CODE nenulová, pak veškerá další data by měla být ignorována, protože jsou pravděpodobně neplatná.

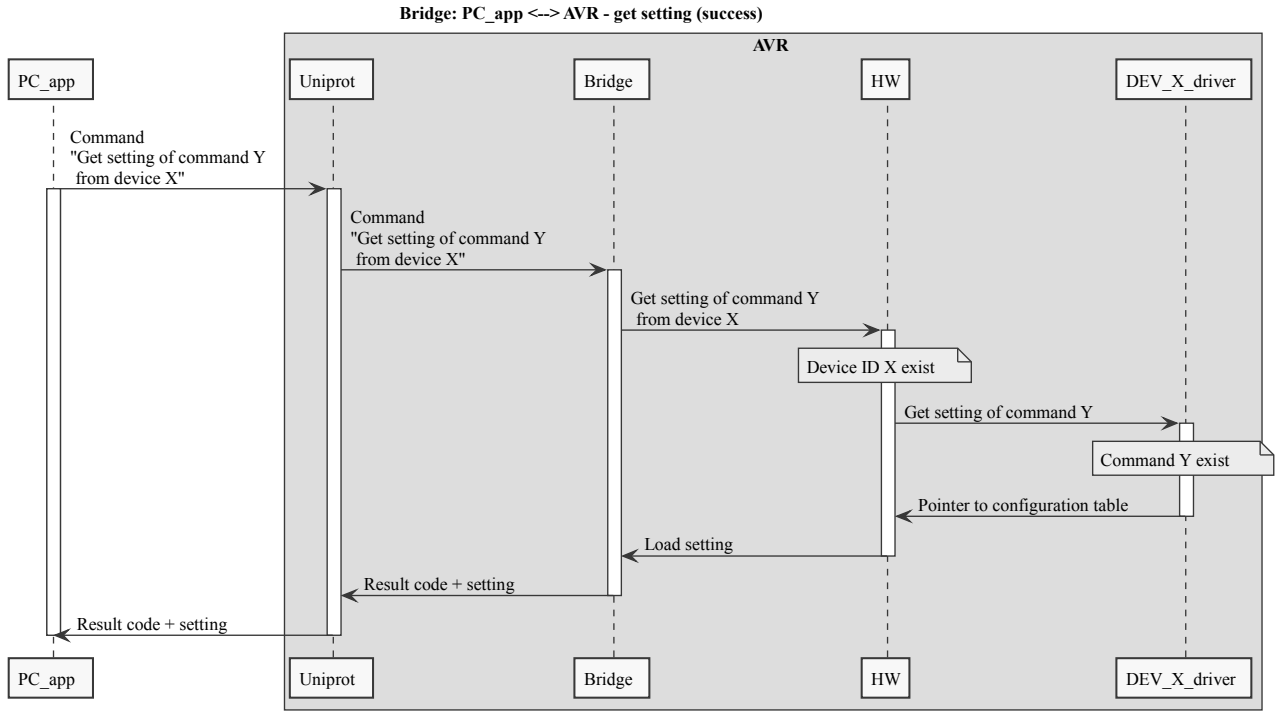
DID	BCMD (0x01)	CMD ID

Obr. 27: Formát příkazu "get setting"

DID	RES CODE	CMD ID	IN TYPE	IN MIN	IN MAX	OUT MIN	OUT MAX	OUT VALUE	NAME	DESC

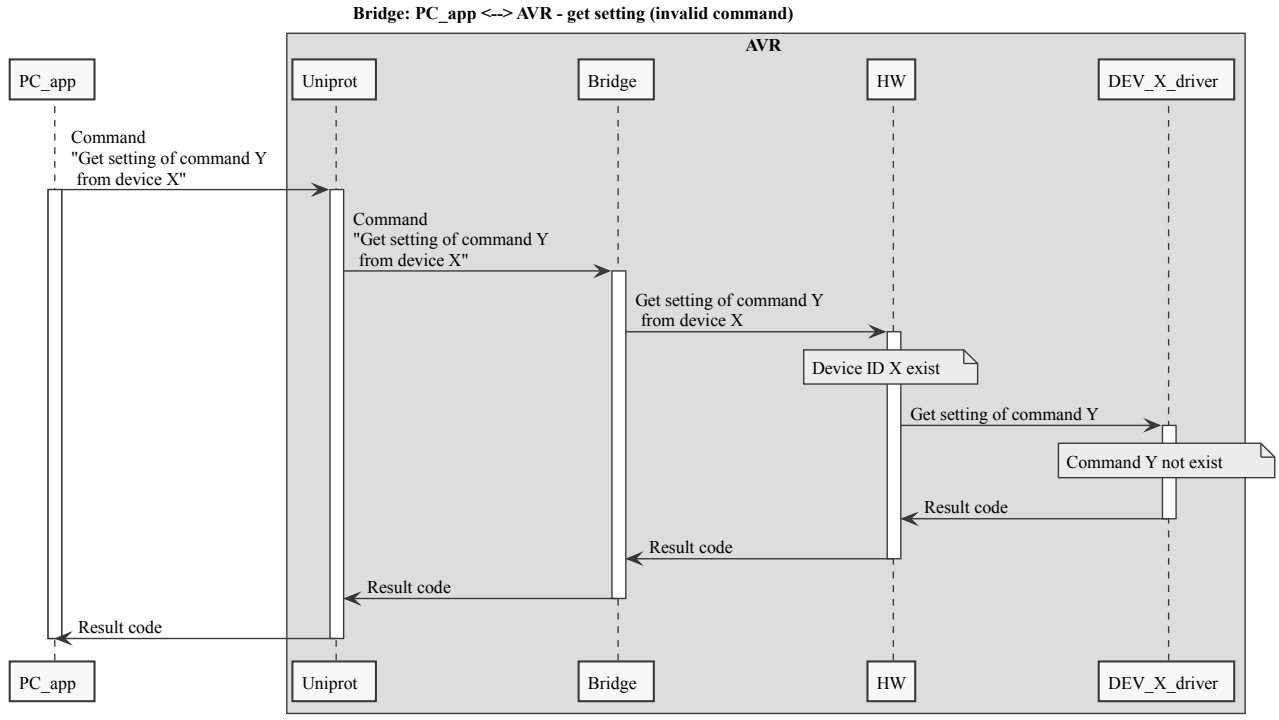
Obr. 28: Formát odpovědi na "get setting"

Komunikace přes jednotlivé vrstvy je na Obr. 29. Protože vrstva „generic driver“ je z tohoto pohledu prakticky transparentní, byla zde a v následujících diagramech záměrně vynechána.



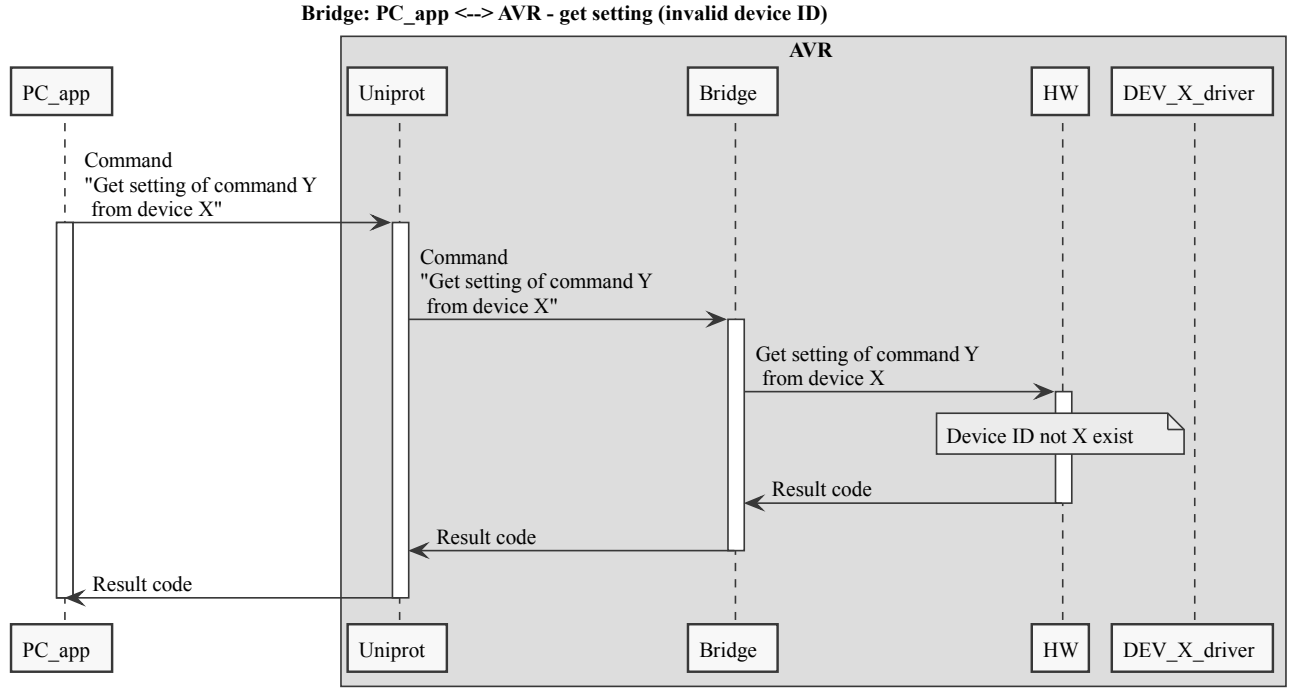
Obr. 29: Get setting - vše v pořádku

Může nastat situace, kdy je CMD ID neplatné. Pak bude komunikace probíhat podle diagramu na Obr. 30.



Obr. 30: Get setting - neplatné CMD ID

Případně se může stát, že DID (Device ID) nebude validní. Tento scénář je popsán na Obr. 31.



Obr. 31: Get setting - neplatné DID

6.3.2. Set setting

Formát příkazu je na Obr. 32. Hodnota BCMD je 0x02. Formát odpovědi je na Obr. 33.

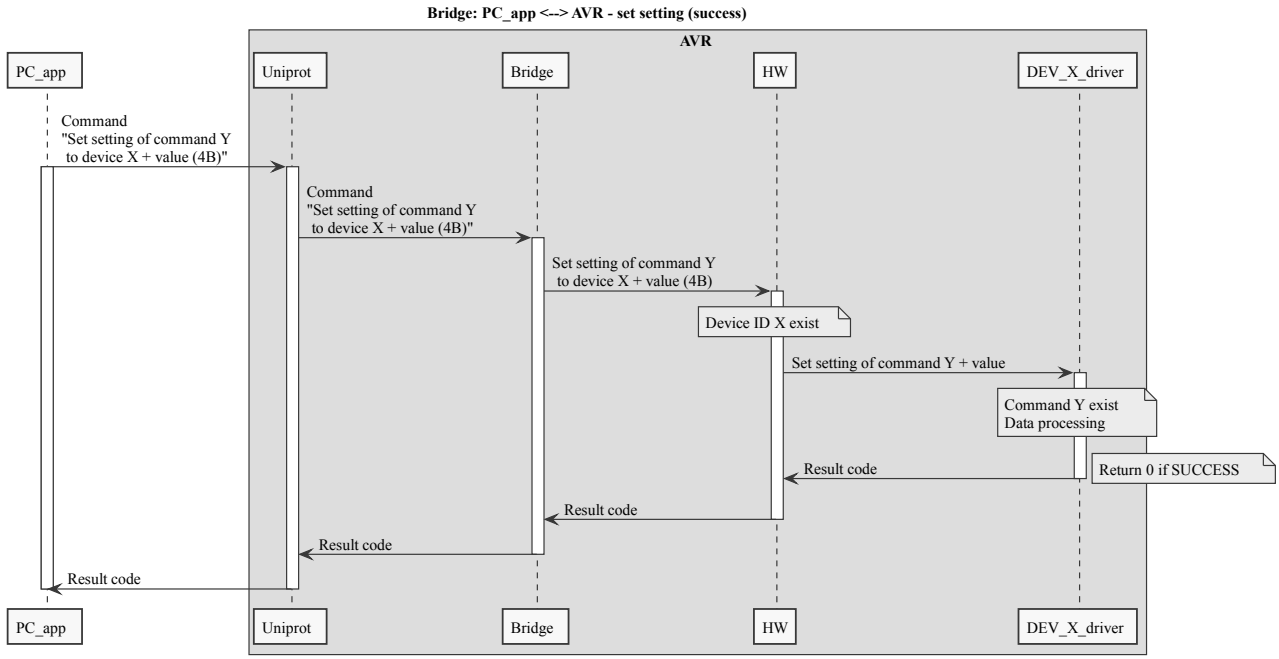
DID	BCMD	CMD ID	IN VALUE

Obr. 32: Formát příkazu "set setting"

DID	RES CODE

Obr. 33: Formát odpovědi na "set setting"

Komunikace přes jednotlivé vrstvy je na Obr. 34.



Obr. 34: Set setting - vše v pořádku

V případě problémů s DID nebo CMD ID bude postup naprosto stejný jako v případě „get_setting“.

6.3.3. Get metadata

Formát příkazu je na Obr. 35. Hodnota BCMD je 0x03. Formát odpovědi je na Obr. 36. Pokud je hodnota RES CODE nenulová, pak veškerá data v paketu by měla být ignorována.

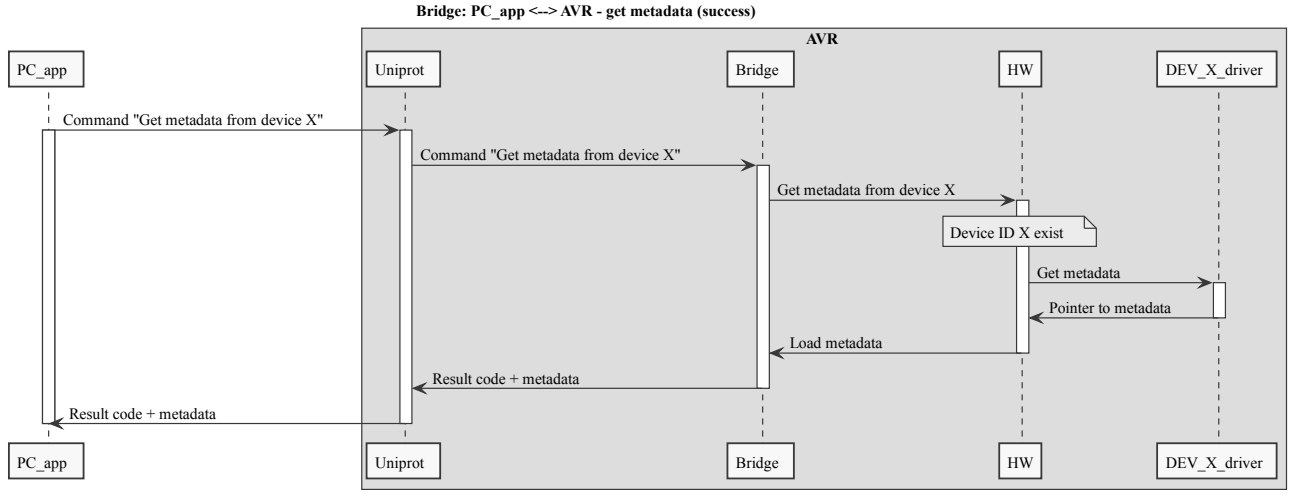
DID	BCMD

Obr. 35: Formát příkazu "get metadata"

DID	RES CODE	MAX CMD ID	SERIAL NUM	NAME	DESC

Obr. 36: Formát odpovědi na "get metadata"

Komunikace přes jednotlivé vrstvy je na Obr. 37.



Obr. 37: Get metadata - vše v pořádku

V případě, že by DID nebylo validní, pak by vrstva „HW“ navrátila jako RES CODE nenulovou hodnotu.

6.3.4. Get number of devices

Formát příkazu je na Obr. 38. Je nutné, aby hodnota DID byla rovna nule, jinak může zařízení příkaz odmítnout. Hodnota BCMD je 0x04. Formát odpovědi je na Obr. 39. I v odpovědi by měla hodnota DID být nastavena na 0, jinak může být odpověď odmítnuta. Pokud je hodnota RES CODE nenulová, pak by NUM OF DEVICES mělo být ignorováno.

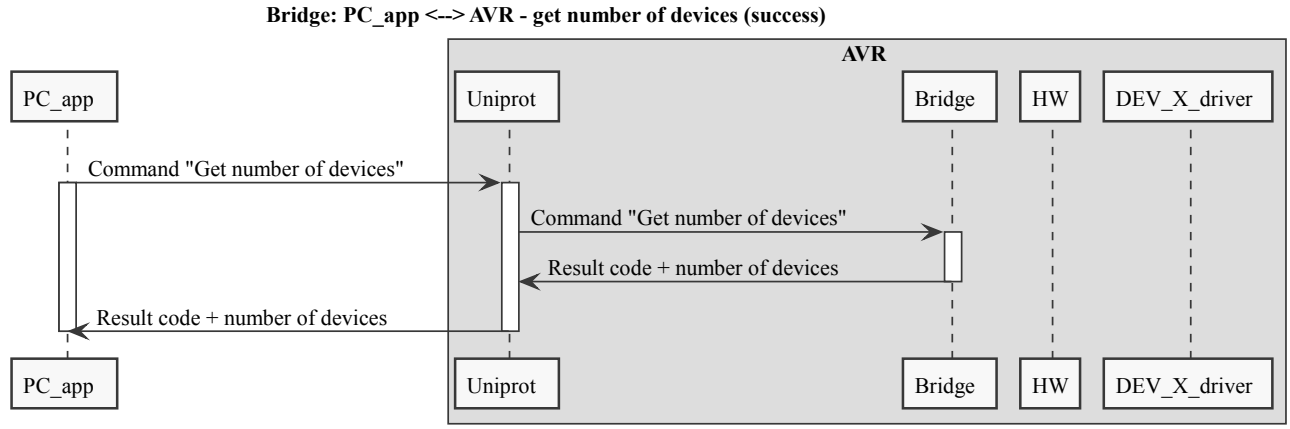
DID	BCMD

Obr. 38: Formát příkazu "get number of devices"

DID	RES CODE	NUM OF DEVICES

Obr. 39: Formát odpovědi na "get number of devices"

Komunikace je naznačena na Obr. 40.



Obr. 40: Get number of devices - průběh komunikace

6.4. Aplikace pro PC

Samotná aplikace má zjednodušit konfiguraci zařízení. Základní návrh uvažuje aplikaci pro příkazovou řádku. Ta v prvním běhu vygeneruje konfigurační soubor, který bude snadno čitelný a upravitelný. Uživatel pak provede úpravu tohoto konfiguračního souboru a aktuální nastavení pošle zpět do zařízení v dalším spuštění aplikace. Taková aplikace nemá příliš vysoké požadavky na hardware nebo softwarové vybavení.

Pro implementaci aplikace byl zvolen jazyk Python. Mezi jeho výhody oproti ostatním programovacím jazykům patří zejména jednoduchost a efektivita práce [23], dále pak také jednoduchá přenositelnost mezi platformami.

6.4.1. Python 2.7 a 3.x

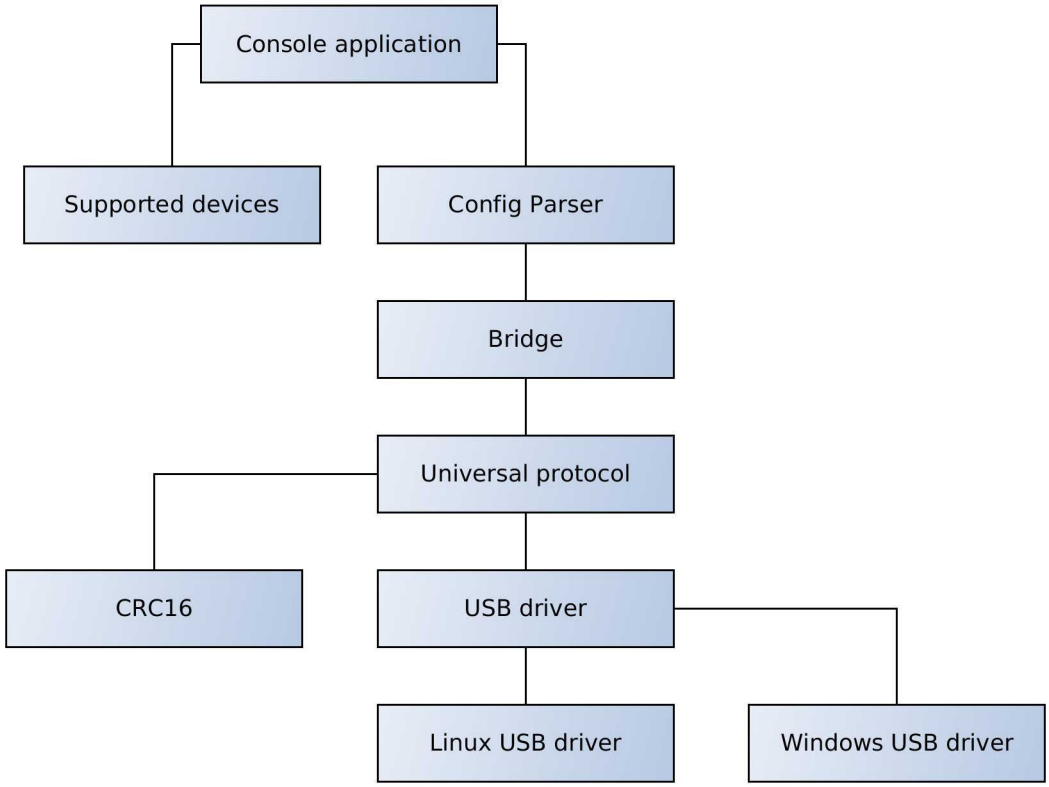
Jako každý programovací jazyk prošel i Python svým vývojem. Během vývoje se ukázaly drobné nedostatky, a proto byla představena verze 3, která je více striktní než verze 2. Přestože jsou rozdíly minimální, může se stát, že aplikace napsaná pro starší verzi nemusí fungovat. Protože mnoho aplikací používá dodnes Python 2.7, je tato verze stále jakýsi standard. Na druhou stranu by bylo neefektivní psát novou aplikaci optimalizovanou pouze pro starou verzi tohoto programovacího jazyka. Naštěstí Python jako takový umožňuje detekovat v programu (resp. skriptu), pod jakou verzí byl spuštěn. Podle toho je možné problematické části kódu rozdělit na část pro verzi 2.7 a verzi 3. Řešení detekce verze Pythonu je velmi jednoduché a elegantní. Ukázka detekce verze 2, 3 a nespecifikované verze:

```
import sys
if(sys.version_info[0] == 2):
  print(„Version 2“)
elif(sys.version_info[0] == 3):
  print(„Version 3“)
else:
  print(„Unknown version“)
```

Aktuální verze programu byla testována na verzích 2.7 (Linux, Windows 7, Windows XP), 3.2, 3.3 a 3.4 (Linux).

6.4.2. Vrstvy aplikace

Stejně jako na straně AVR, tak i na straně počítače je vhodné celé řešení rozdělit do několika vrstev. Kód je pak modulárnější a čitelnější. Sice jemnější rozvrstvení znamená větší režii programu, ale pro aplikaci daného typu výkonové faktory prakticky nehrají roli. Schématické znázornění komponent/vrstev je na Obr. 41. Jedna komponenta na schématu není: „logger“. To je vrstva, která umožňuje záznam informačních zpráv, varování a chyb do konzole i souboru. Protože ji využívá většina vrstev, nebyla do schématu pro větší přehlednost přidána.



Obr. 41: Rozvrstvení aplikace pro počítač

Vrstva „USB driver“ zprostředkovává abstrakci platformy. Podle aktuálních informací se rozhodne, jaký USB ovladač se použije. Z pohledu vyšších vrstev je to úplně jedno. Vrstva „USB driver“ nabízí stejné funkce a pracuje se stejnými daty. Funkčnost byla ověřena jak na OS Linux (x86_64, ARM), tak na OS Windwos XP (x86_64) a Windows 7 (x86_64). Úspěšné ověření funkčnosti na několika rozdílných HW i SW platformách potvrzuje dobrou přenositelnost programu a vhodnost volby jazyka Python pro danou aplikaci.

Vrstvy „universal protocol“ je v podstatě totožná s implementací použitou v AVR. Viz předchozí kapitola „universal protocol“. Pro tuto vrstvu byla vytvořena funkce, které implementuje stejný CRC algoritmus jako MCU.

Vrstva „bridge“ se mírně liší. Ihned při inicializaci totiž oskenuje připojené zařízení a stáhne si všechny informace o všech funkcích, které zařízení jako celek nabízí (tj. stáhne veškeré informace o každém ovladači, který je „připojen“ k vrstvě „HW“ na straně MCU). Tyto informace nechá uložené v RAM, takže pro další operace již není nutné přenášet informace znovu. Až na tuto výjimku nabízí vrstva stejné možnosti jako u implementace pro MCU.

Vrstva „config parser“ má za úkol vytvořit, nebo vyčíst data z konfiguračního souboru. Proto si ještě při inicializaci stáhne veškerá data o zařízení do RAM (pomocí vrstvy „Bridge“). Pokud je zavolána funkce `write_setting_to_cfg_file()`, pak je konfigurační soubor vygenerován. Vygenerování konfiguračního souboru je relativně komplexní záležitost a vyžaduje podrobné znalosti především vrstvy „bridge“. Vzhledem k možnostem zde nebude uveden vývojový diagram. Níže je uveden pouze demonstrační příklad toho, jak může vypadat konfigurační soubor:

```
[Board driver for Sonochan mkII v0.1]
; device id (did): 0
; serial number: 15

[Initialize board hardware]
; initialize i/o and prepare codec
; set call_function to non zero if you want call this function
call_function = 0

[Audio codec TLV320AIC33 v0.1a]
; device id (did): 1
; serial number: 102

[Digital interface data mode]
; 0  - i2s ; 1 - dsp ; 2 - right justified ; 3 - left justified
; type: uint8 < 0 : 3 > | current value: 0
value = 0

[Word length]
; options: 16, 20, 24, 32
; type: uint8 < 16 : 32 > | current value: 32
value = 32

[Fractional PLL CS2200-CP v0.7]
; device id (did): 2
; serial number: 33

[Set PLL frequency]
; frequency format: unsigned 32 bit in hz
; type: uint32 < 6000000 : 75000000 > | current value: 12287990
value = 12287990

[Increase PLL frequency by 1]
; increase ratio number by 1
; set call_function to non zero if you want call this function
call_function = 0

[Set output divider/multiplier ratio]
; input values: 1 2 4 8 0.5 0.25 0.125 0.0625
; type: float < 0.0625 : 8.0 > | current value: 1.0
value = 1.0
```

Funkce jednotlivých ovladačů jsou řazeny podle jejich příkazového čísla (CMD ID). Před funkcemi samotného ovladače je nejprve uvedeno jméno modulu (případně i verze), jeho DID a sériové číslo. Pak následuje výčet funkcí pro daný ovladač. Řádky započaté středníkem jsou komentáře a mají uživateli zjednodušit orientaci v souboru. Uživatel mění pouze hodnoty u řádků nezačínající středníkem, nebo hranatou závorkou.

Naopak funkce `read_setting_from_file()` vyčítá data z konfiguračního souboru. Vyčítání ze souboru je opět relativně komplexní, proto zde není uveden vývojový diagram, ale pouze stručný popis, co funkce dělá. Aby konfigurace zařízení trvala co nejkratší dobu, při čtení z konfiguračního souboru jsou registrovány položky, které byly změněny, a pouze tyto změny jsou poslány do zařízení. Tím se lze vyhnout nechtěnému volání některých funkcí zařízení.

Vrstva „console application“ je již vrstva určená uživateli. Má za úkol sledovat vstupní parametry, generuje nápovědu v případě v nekorektních parametrů a celkově zjednodušuje uživateli práci. Popis uživatelského rozhraní je v následující kapitole. Tato vrstva zároveň využívá vrstvu „supported devices“, která generuje z textového souboru seznam zařízení, která jsou podporována. Každé zařízení musí mít přiřazené jméno, VID a PID (pro identifikaci na USB).

6.4.3. Základní ovládání aplikace

Přestože se aplikace spouští v konzoli, její ovládání je jednoduché a ve výchozím nastavení

potřebuje minimum parametrů. Výpis nápovědy je možný s parametrem „--help“ (bez uvozovek).

```
python UniversalControlApp_console.py –help
```

Aplikace vypíše stručný přehled všech parametrů, popis samotné aplikace, copyright a podrobnější popis k parametrům. Ve výchozím nastavení (tj. bez parametru) je seznam podporovaných zařízení hledán v *config/device_list.cfg*, konfigurační soubor pro logování je hledán v *config/logging_global.cfg* a je vygenerován konfigurační soubor *device.cfg*. Pomocí parametru „-r“ je možné přečíst konfigurační soubor pro zařízení a aplikovat změny, které uživatel provedl v konfiguračním souboru *device.cfg*.

```
python UniversalControlApp_console.py –r
```

Jméno souboru pro ukládání, nebo načítaní nastavení zařízení může být explicitně určeno parametrem „-d“.

```
python UniversalControlApp_console.py –d Sonochan_mkII.cfg
```

Seznam podporovaných zařízení může být také explicitně definován parametrem „-L“.

```
python UniversalControlApp_console.py –L config/supported_devices.list
```

Stejně tak může být explicitně definována cesta ke konfiguračnímu souboru pro logování. K tomu slouží parametr „-l“.

```
python UniversalControlApp_console.py –l config/logging.cfg
```

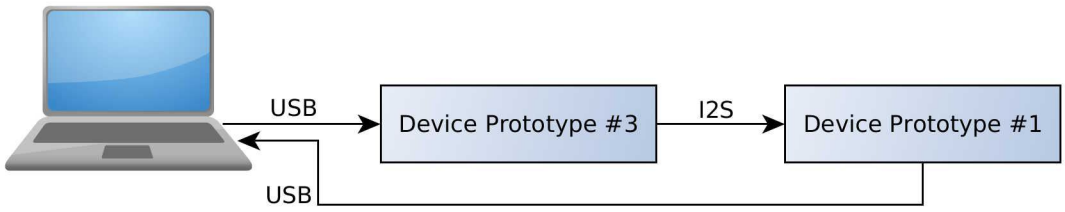
Pro ladění programu je zde ještě parametr „-f“. Pokud je k počítači připojeno několik různých zařízení (musí mít různá alespoň PID) a tento parametr je nastaven na nenulovou hodnotu, pak je automaticky zvoleno zařízení, které je nalezeno v seznamu podporovaných zařízení jako první. Tato možnost není zaměřena na uživatele, ale spíše na vývojáře v průběhu testování. Ve výchozím nastavení je tato volba vypnuta. V momentě, kdy je tato volba vypnuta a je k počítači připojeno několik různých zařízení (která jsou aplikací podporována), pak aplikace uživateli zobrazí názvy jednotlivých zařízení. V tento moment je na uživateli, aby si zvolil zařízení, které chce nastavit. V případě, kdy je k počítači připojeno jenom jedno z podporovaných zařízení, není výběr nutný a aplikace automaticky vybere připojené zařízení.

Aplikace je multiplatformní a podporuje jak Python 2.7, tak nové verze 3.x. Rozvrstvenost umožňuje snadno rozšiřovat a ladit kód. Zároveň je kód dobře čitelný. Jako plánované rozšíření je GUI, které bude generováno podobným způsobem jako konfigurační soubor pro zařízení.

7. Naměřené údaje

7.1. Ověření funkčnosti

Pro ověření funkčnosti byla použita první a poslední verze testovacího zařízení. Schématické zapojení je na Obr. 42.



Obr. 42: Schématické zapojení pro ověřování funkčnosti

Zařízení „Device Prototype #3“ je aktuální verze testovacího zařízení, zatímco „Device Prototype #1“ je první verze. Směr audio dat je naznačen šipkami. Pro ověření byl použit testovací signál, který záměrně není reprodukovatelný analogovými obvody (především kvůli oddělovacím kondenzátorům), ale je možné jej přenést přes čistě digitální rozhraní. Testovací signál je uložen v přílohách jako *BPT_Test_16000.wav*. Parametry měření jsou shrnuty v Tab. 13.

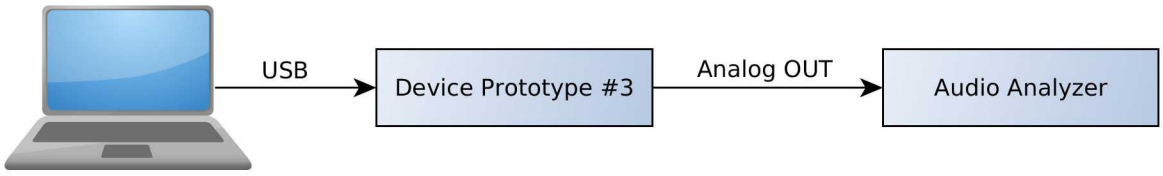
Tab. 13: Přehled parametrů při ověřování funkčnosti

Parametr	Hodnota
Vzorkovací frekvence	16 000 Hz
Bitové rozlišení souboru	16 bitů
Bitové rozlišení v přehrávači	24 bitů
Násobek MCLK	256x
Násobek BCLK	32x
Testovací soubor	BPT_Test_16000.wav

Měřením byla ověřena funkčnost obou zařízení.

7.2. Sluchátkový výstup pro kontrolní poslech

Schématické zapojení pro měření vlastností sluchátkového výstupu je na Obr. 43.



Obr. 43: Schématické zapojení pro měření vlastností sluchátkového výstupu

Jako audio analyzátor byl použit KENWOOD Audio Analyzer VA-2230A. Přehled parametrů měření je v Tab. 14. Přehled naměřených hodnot je v Tab. 15.

Tab. 14: Přehled parametrů při měření vlastností sluchátkového výstupu

Parametr	Hodnota
Vzorkovací frekvence	48 000 kHz
Bitové rozlišení	24 bitů
Násobek MCLK	256x
Násobek BCLK	32x
Frekvence testovacího signálu	1 000 Hz sin
Úroveň testovacího signálu	0 dBFS
Úroveň hlasitosti nastavená na kodeku	0 dB

Tab. 15: Naměřené hodnoty na výstupu

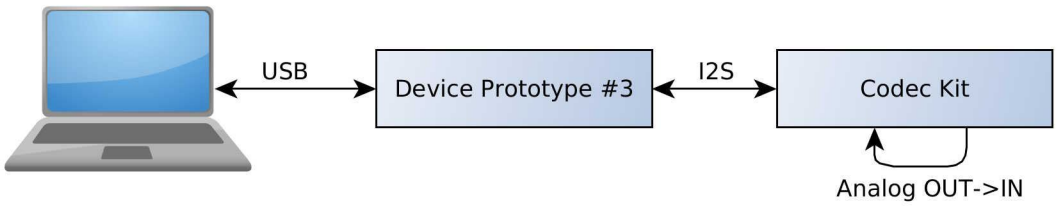
Parametr	Hodnota
THD	0,1 %
THD+N	0,2 %
SNR	50 dB

Nízká hodnota SNR je s nejvyšší pravděpodobností způsobena samotným výkonovým stupněm v kodeku, protože i při vypnutém DAC na kodeku hodnota nepřesáhla 75 dB. Zároveň byla zjištěna závislost mezi nastavení výstupní úrovně a SNR a to i v případě, že všechny vstupy byly uzemněny. Tím pádem byla vyloučena možnost rušení od vstupních pinů.

Přestože naměřené parametry nejsou vyhovující pro profesionální audio aplikace, je sluchátkový výstup stále vhodný pro kontrolní odposlech.

7.3. Propojení s Codec Kitem

Codec Kit je zařízení, které je určeno pro velmi přesný převod mezi digitálním I^2S a analogovým signálem. Kompatibilita s tímto zařízením je prakticky nezbytná. Schématické zapojení propojení zařízení je na Obr. 44.



Obr. 44: Schématické propojení testovacího zařízení a Codec Kitu

Audio stream proudil z PC do testovacího zařízení, kde byl převeden do formátu I^2S. Odtud stream směřoval do Codec Kitu. Zde byl signál převeden z digitální podoby na analogovou a veden zpět. Proběhl převod do digitální podoby a byl poslán zpět do testovacího zařízení, které jej odeslalo do PC. V počítači bylo pak provedeno porovnání.

Pro toto měření nemohl být použit testovací signál *BPT_Test_16000.wav* kvůli analogovým obvodům. Proto byl jako testovací signál použit sinusový průběh s frekvencí 1 kHz. Kvůli převodům nebylo možné zajistit bitovou věrnost přenosu, ale bylo ověřeno, že signál byl úspěšně přenesen. Nastavení parametrů při měření je v Tab. 16.

Tab. 16: Nastavení parametrů při propojení s Codec Kitem

Parametr	Hodnota
Vzorkovací frekvence	48 000 kHz
Bitové rozlišení	24 bitů
Násobek MCLK	256x
Násobek BCLK	32x
Frekvence testovacího signálu	1 000 Hz sin
Úroveň testovacího signálu	0 dBFS

7.4. Použité aplikace

Pro zpracování audio streamů na straně PC byl použit program *Audacity 2.0.5* na OS Linux. Veškerá USB zařízení uvedená na schématech byla připojena přímo k USB. To znamená, že nebyly použité žádné USB rozbočovače.

Nastavení testovacího zařízení a Codec Kitu bylo provedeno pomocí aplikace *UniversalControlApp_console v0.4*, která je součástí přílohy.

Pro získání USB deskriptorů byl použit program *lsusb*.

Následující příkaz byl použit pro získání seznamu podporovaných vzorkovacích frekvencí, informace o synchronizační metodě a stavu testovacího zařízení:

```
watch -n 0,1 'cat /proc/asound/card1/stream0'
```

8. Závěr

V rámci práce se podařilo realizovat funkční zařízení a pomocí měření potvrdit, že splňuje požadavky zadání.

V průběhu projektu došlo k několika zásadním změnám v HW i SW. Prvotní verze s AT32UC3A0512 se ukázala jako nedostačující z důvodů HW omezení. Proto byl použit výkonnější MCU AT32UC3A3256, který daná omezení řešil. Na druhou stranu to znamenalo přepracovat projekt od základů. V té fázi byla zvolena adaptace projektu SDR-Widget. Díky tomu bylo možné vyjít ze základů implementace USB Audio pro zvolený MCU a na té vystavět zbývající části projektu. V aktuální verzi SW tvoří původní program projektu SDR-Widget méně než 20% kódu (bez započtení modifikovaného frameworku firmy Atmel).

Navržené zařízení aktuálně umožňuje obousměrný stereo přenos audio dat s 24bitovým rozlišením. Podporované vzorkovací frekvence jsou: 8000, 11025, 16000, 22050, 24000, 32000, 44100 a 48000 Hz. Celé zařízení nepotřebuje žádné ovladače třetích stran z důvodu použití předdefinovaných standardů, takže vše potřebné má většina OS již implementováno. Jak funkčnost audia, tak ovládací aplikace byla vyzkoušena na Windows XP, Windows 7 i Linuxu. Galvanické oddělení je řešeno pomocí moderních digitálních izolátorů, které mají dostatečnou šířku pásma a zároveň jsou schopny pracovat v požadovaném napěťovém rozsahu.

HW projekt, stejně jako programový kód FW i obslužné aplikace byl důkladně zdokumentován a umožňuje, aby na práci navázali kromě autora i další vývojáři.

V průběhu realizace byly zjištěny další možnosti, jak úpravou FW funkčnost zařízení dále rozšířit a ještě více tak využít potenciál platformy.

Seznam použité literatury

[1] AT32UC3A3/A4 Series Complete [online], Poslední úpravy 10.2012 [citace 5.5.2014]. Dostupné na www: http://www.atmel.com/images/icon_pdf.gif

[2] TLV320AIC33 [online], Poslední úpravy 12.2008 [citace 6.5.2014]. Dostupné na www: www.ti.com/cn/lit/gpn/tlv320aic33

[3] Embedded USB – a brief tutorial [online], [citace 7.10.2013]. Dostupné na www: http://www.computer-solutions.co.uk/info/Embedded_tutorials/usb_tutorial.htm

[4] 32-bit AVR UC3 Microcontrollers [online], [citace 9.10.2013]. Dostupné na www: http://www.atmel.com/products/microcontrollers/avr/32-bitavruc3.aspx

[5] EVK1105 Schematics and BOM [online], [citace 27.10.2013], Dostupné na www: http://www.atmel.com/Images/evk1105-schematics_bom.zip

[6] Compact Disk Digital Audio [online], [citace 27.10.2023], Dostupné na www: http://en.wikipedia.org/wiki/Compact_Disc_Digital_Audio

[7] Si864x DataSheet [online], Poslední úpravy 9.2013 [citace 29.10.2013]. Dostupné na www: http://www.silabs.com/Support%20Documents/TechnicalDocs/Si864x.pdf

[8] List of USB ID's [online], Poslední úpravy 5.5.2014 [citace 10.5.2014], Dostupné na www: http://www.linux-usb.org/usb.ids

[9] PCD8544 [online], Poslední úpravy 03.1999 [citace 29.10.2013]. Dostupné na www: http://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf

[10] Atmel AVR32716: AVR UC3 USB Audio Class [online], [citace 30.10.2013]. Dostupné na www: http://www.atmel.com/Images/doc32139.pdf

[11] Universal Serial Bus Device Class Definition for Audio Devices [online], [citace 9.11.2013]. Dostupné na www: http://www.usb.org/developers/devclass_docs/audio10.pdf

[12] USB in NutShell – Chapter 2 – Hardware [online], [citace 3.5.2014]. Dostupné na www: http://www.beyondlogic.org/usbnutshell/usb2.shtml

[13] SDR-Widget [online], [citace 3.5.2014]. Dostupné na www: https://code.google.com/p/sdr-widget/

[14] AB-1.1_20110818.zip [online], Poslední úpravy 18.8.2011 [citace 4.5.2014]. Dostupné na www: http://sdr-widget.googlecode.com/files/AB-1.1_20110818.zip

[15] Why RTOS and What is RTOS? [online], Poslední úpravy 1.2014 [citace 10.5.2014]. Dostupné na www: http://www.freertos.org/about-RTOS.html

[16] Oscilloscope [online], Poslední úpravy 2.3.2014 [citace 9.5.2014]. Dostupné na www: http://en.wikipedia.org/wiki/Oscilloscope

[17] USB Audio 2.0 Driver for Windows Overview [online], Poslední úpravy 6.2.2014 [citace 9.5.2014]. Dostupné na www: https://www.xmos.com/support/documentation/xkits/Reference%20Designs?product=17498&component=17013

[18] Multi-Function Audio Platform (MFi): product brief [online], Poslední úpravy 25.3.2014 [citace 9.5.2014]. Dostupné na www: https://www.xmos.com/support/documentation/xkits/Reference%20Designs?product=17498&component=17623

[19] APx525 582 585 Families of audio analyzers by Audio Precision [online], Poslední úpravy 10.2013 [citace 9.5.2014]. Dostupné na www: http://www.ap.com/download/file/658

[20] The Well-Tempered Computer (An introdution to computer audio) [online], Poslední úpravy 3.2011 [citace 9.5.2014]. Dostupné na www: http://www.thewelltemperedcomputer.com/KB/USB.html

[21] Amplizone: USB Audio Class 2.0. Windows and Computer HiFi [online], Poslední úpravy 24.2.2014

[citace 10.5.2014]. Dostupné na www: http://amplioaudio.blogspot.cz/2014/02/usb-audio-class-20-windows-and-computer.html

[22] Fundamentals os USB Audio | EDN [online], Poslední úpravy 5.2012 [citace 10.5.2014]. Dostupné na www: http://www.edn.com/design/consumer/4376143/2/Fundamentals-of-USB-Audio

[23] Comparing Python to Other Languages [online], Poslední úpravy 1997 [citace 13.5.2014]. Dostupné na www: https://www.python.org/doc/essays/comparisons/

Abecední seznam zkratek

ADC – Analog to digital convertor – převodník analogového signálu na digitální

BER – Bit error rate – bitová chybovost

DAC – Digital to analog convertor – převodník digitálního signálu na analogový

DMA – Direct memry access – přímý přístup do paměti

DPLL – Digitální PLL

DSP – Digital signal processor – signálový procesor

EP – Endpoint

GPIO – General purpose input-output – obecné vstupně/výstupní piny

GUI – Graphical User Interface – grafické rozhraní pro uživatele

HID – Human interface device – zařízení pro komunikaci s uživatelem

ID – Identifikační (číslo)

INTC – Interrupt controller – kontrolér přerušení

LCD – Displej z tekutých krystalů

MCU – Mikrokontrolér

MSO – Mixed signal oscilloscope – osciloskop mající jak analogové vstupy, tak digitální, které je možné použít pro logickou analýzu

PDCA – Peripheral DMA Controller – kontrolér pro obsluhu periferií využívajících DMA

PID – Product ID – identifikátor produktu

PLL – Fázový závěs

PM – Power manager – správce napájení

SNR – Signal to noise ratio – poměr signál, šum

SOF – Start of frame – počátek rámce

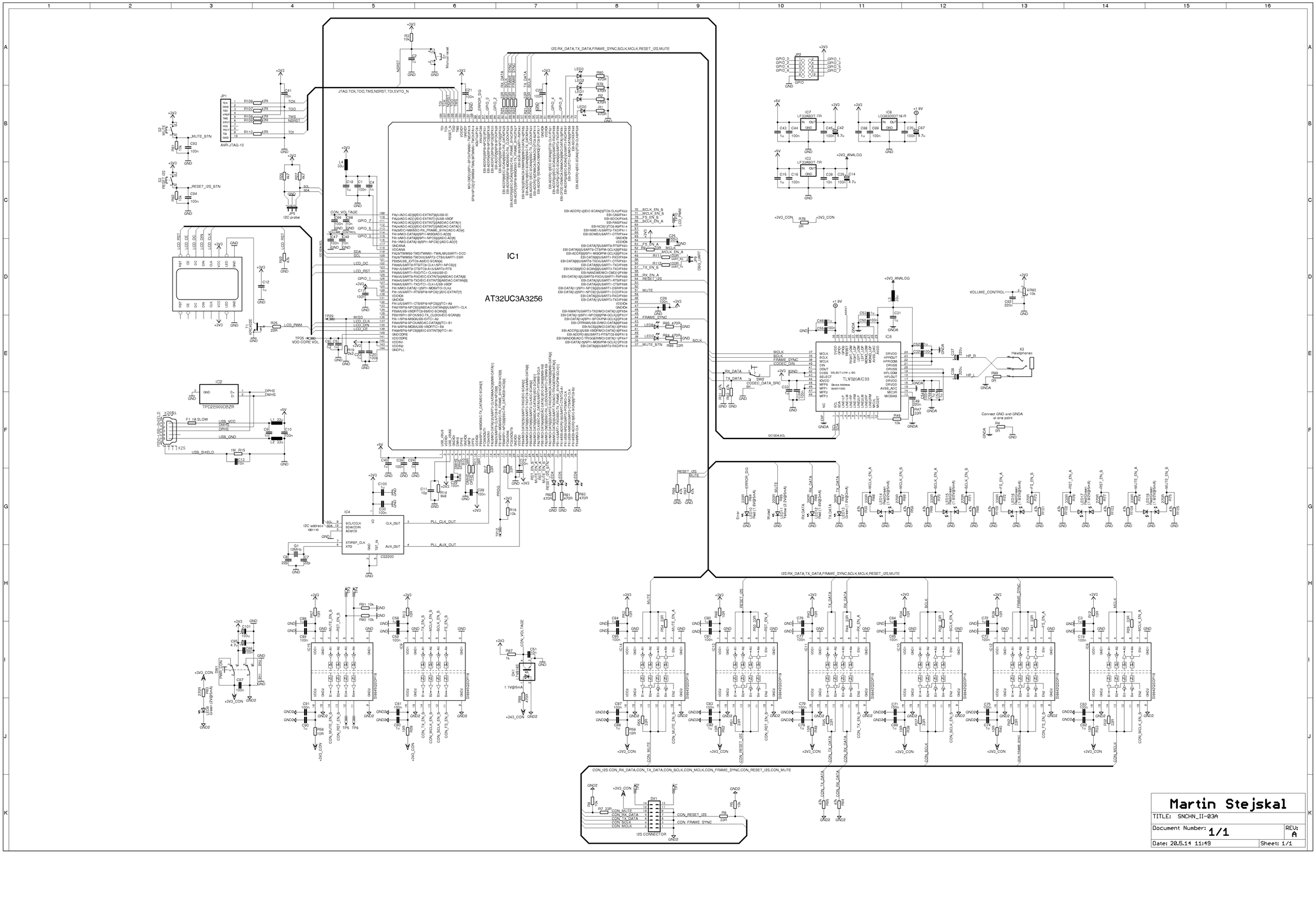
SSC – Sériové synchronní rozhranní

VID – Vendor ID – identifikátor výrobce

Seznam příloh

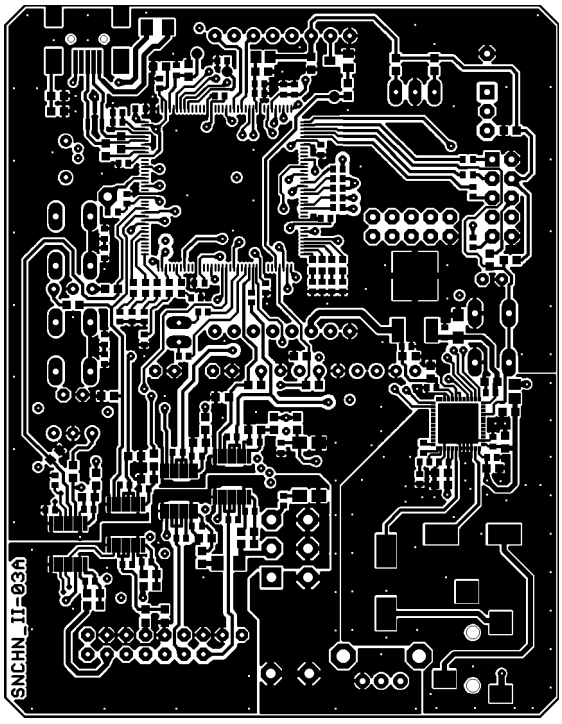
A.Návrh zařízení

A.1. Celkové schéma



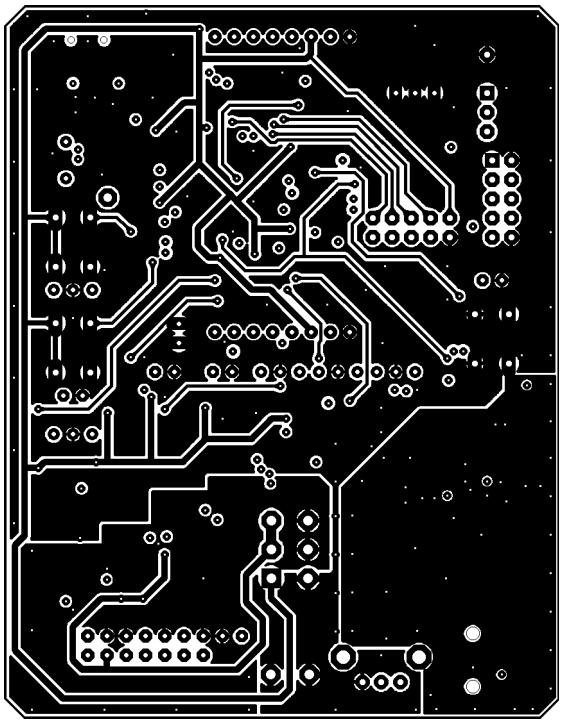
A.2. Deska plošného spoje - top (strana součástek)

Rozměr desky: 73x94 [mm], měřítko M1:1



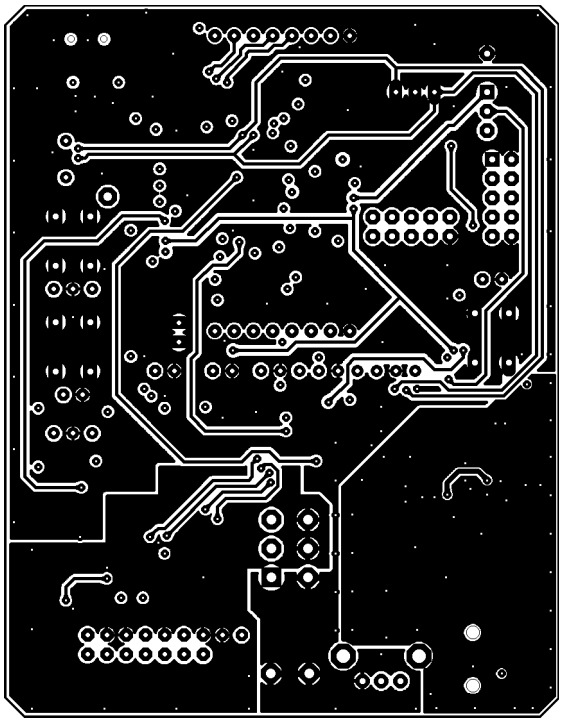
A.3. Deska plošného spoje – vrstva 2

Rozměr desky: 73x94 [mm], měřítko M1:1



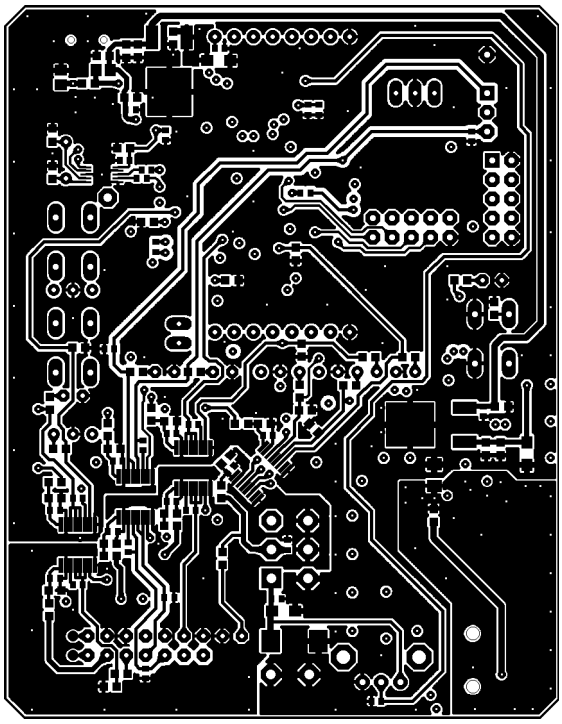
A.4. Deska plošného spoje – vrstva 15

Rozměr desky: 73x94 [mm], měřítko M1:1



A.5. Deska plošného spoje – bottom (strana spojů)

Rozměr desky: 73x94 [mm], měřítko M1:1



B.Seznam součástek

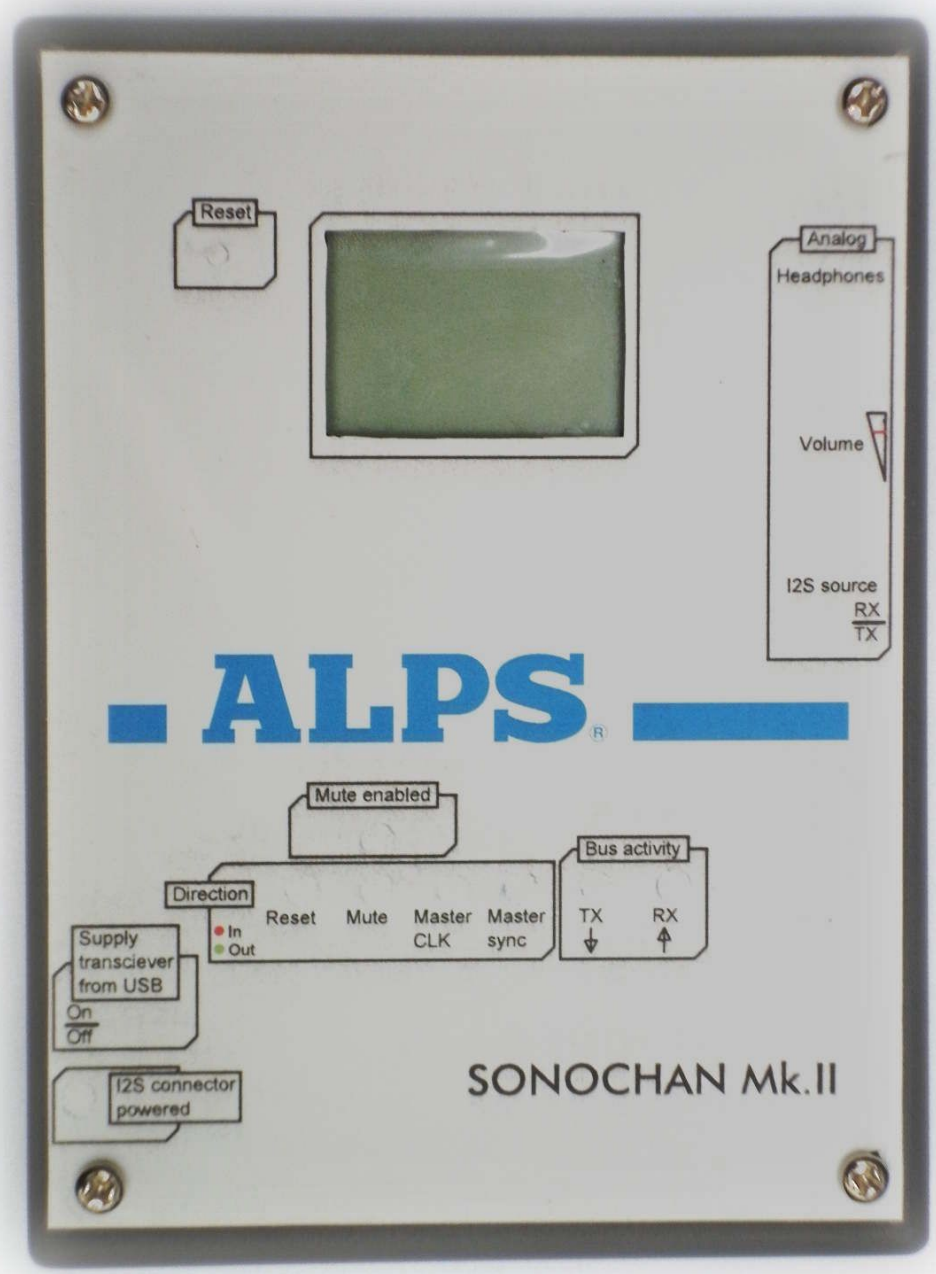
Množství	Hodnota	Pouzdro	Komponenty
9		0603	LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED8
1	0R	0603	R89
2	0R	1206	R4, R78
1	1A SLOW	1206	F1
1	1M	0603	R15
1	1k	0603	R87
3	1n	0402	C4, C5, C24
1	1n	0603	C9
30	1u	0603	C2, C3, C6, C12, C19, C23, C31, C33, C40, C46, C52, C53, C56, C57, C58, C60, C62, C64, C66, C72, C74, C76, C78, C80, C82, C84, C86, C88, C90, C100
3	1u	C0603R	C15, C43, C68
1	2k2	0402	R19
4	4.7u	SMCA	C14, C42, C67, C95
2	4k7	0603	R26, R27
1	6k8	0402	R14
16	10R	0603	R12, R13, R28, R33, R34, R38, R39, R40, R43, R48, R49, R52, R53, R56, R57, R58
6	10k	0402	R16, R46, R61, R62, R90, R91
3	10k	0603	R3, R5, R6
1	10k	RK09K1110AK4	R60
3	10n	0402	C49, C51, C98
3	10n	0603	C13, C36, C41
1	10p	0402	C11
1	12MHz	HC49U-V	Q1
1	22R	0603	R35
2	22p	0603	C7, C8
23	33R	0402	R9, R17, R18, R20, R21,

Množství	Hodnota	Pouzdro	Komponenty
			R22, R23, R24, R25, R29, R32, R36, R37, R41, R42, R44, R45, R47, R50, R51, R54, R55, R88
2	33R	0603	R7, R8
4	33u	L0805	L1, L2, L3, L4
2	39R	0603	R10, R11
4	47R	0402	R106, R107, R108, R110
1	47R	0603	R109
5	47k	0402	R31, R92, R93, R94, R95
12	47k	0603	R30, R59, R96, R97, R98, R99, R100, R101, R102, R103, R104, R105
2	100R	0402	R111, R112
42	100n	0402	C1, C16, C17, C18, C20, C21, C22, C25, C26, C27, C28, C29, C30, C32, C34, C39, C47, C50, C54, C55, C59, C61, C63, C65, C69, C70, C71, C73, C75, C77, C79, C81, C83, C85, C87, C89, C91, C93, C94, C96, C97, C99
4	100n	0603	C10, C35, C44, C45
1	100u	SMCD	C101
1	220R	0603	R65
2	220n	0402	C48, C92
2	220u	140CLH-1010	C37, C38
6	270R	0603	R63, R69, R71, R73, R75, R77
6	330R	0603	R64, R68, R70, R72, R74, R76
2	390R	0603	R66, R67
10	470R	0402	R1, R2, R79, R80, R81, R82, R83, R84, R85, R86
1	AT32UC3A3256	LQFP144	IC1
1	AVR-JTAG-10	10x 2,54 PINHEADER	JP1
1	CODEC_DATA_SRC	2AS1T2A1M7RE	SW2
1	CS2200	MSOP10	IC4

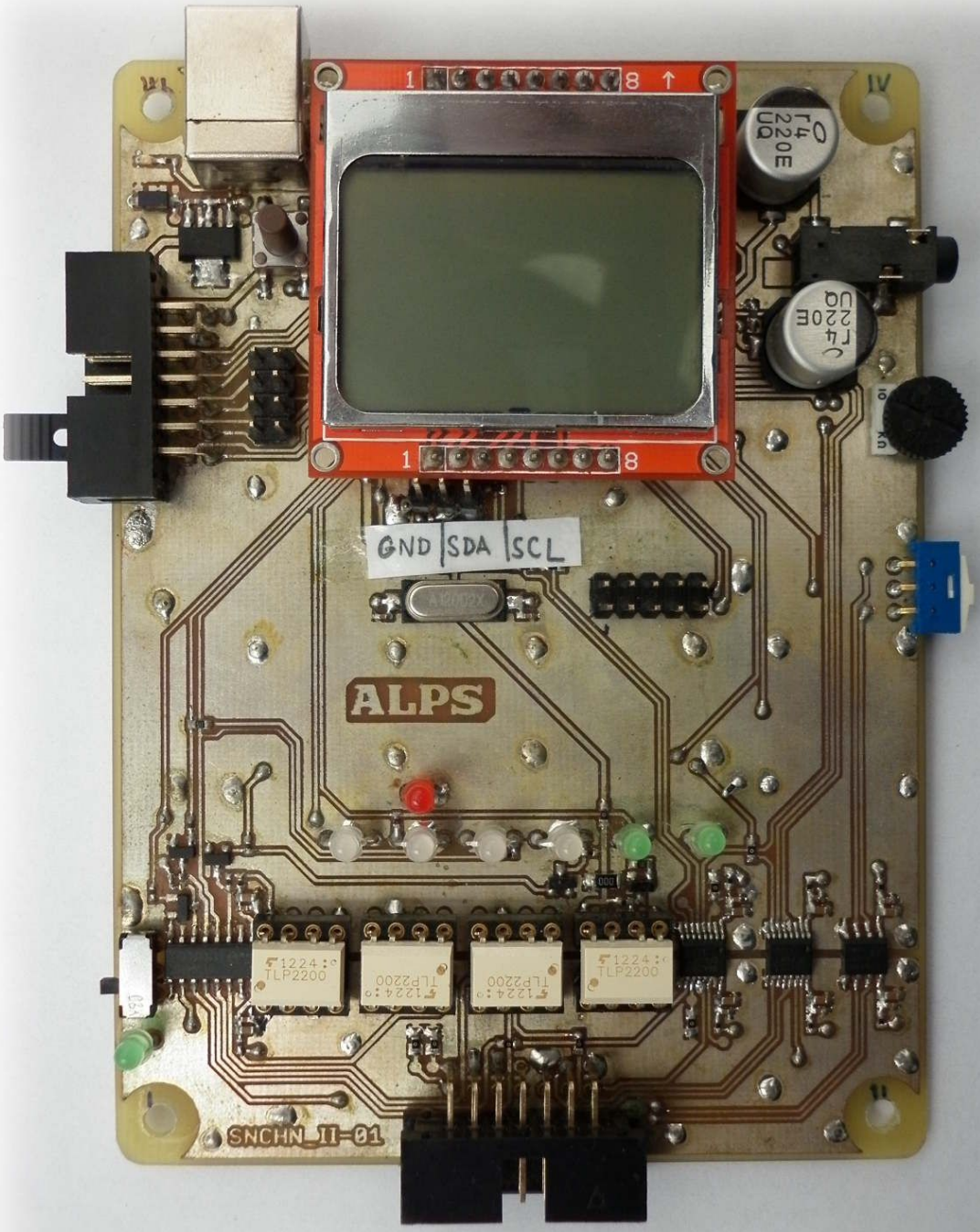
Množství	Hodnota	Pouzdro	Komponenty
1	DBG_UART	2x 2,54 PINHEADER	JP9
1	GPIO	10x 2,54 PINHEADER	JP2
1	Green	LED3MM	LED13
1	Green	LED3MM	LED9
1	Headphones	JACK_3.5MM_3PIN	X3
1	I2C probe	3x 2,54 PINHEADER	JP5
1	I2S CONNECTOR	ML14L	SV1
1	KRC402E	SC59-BEC	T1
1	LCD_NOKIA_5510	NOKIA_5510_MODULE	DIS1
1	LD39300DT18-R	DPACK	IC8
2	LF33ABDT-TR	DPACK	IC3, IC7
1	LTV816S	SMD4	OK1
1	MINI-USB-SHIELD	32005-201	X2
1	MUTE	STANDARD_MICROSWITCH	S2
1	Manual reset	STANDARD_MICROSWITCH	S1
5	PTR1B1,27	PTR1B1,27	TP1, TP2, TP10, TP29, TP35
1	PTR1PAD1-13	PTR1PAD1-13	TP12
4	PTR1TP06R	PTR1TP06R	TP3, TP4, TP5, TP6
1	PWR_CON	1MD1T2B4M7XX	SW1
1	RESET I2S	STANDARD_MICROSWITCH	S3
1	Red	LED3MM	LED12
1	Red	LED3MM	LED10
5	Red/Green	DUO_LED_C3MM	LED14, LED15, LED16, LED17, LED18
6	SI8642QSOP16	QSOP16	IC6, IC10, IC11, IC12, IC13, IC14
2	SI8645QSOP16	QSOP16	IC9, IC15
1	TLV320AIC33	QFN48	IC5
1	TPD2E009DBZR	SOT95	IC2
1	Yellow	LED3MM	LED11

C.Fotografie

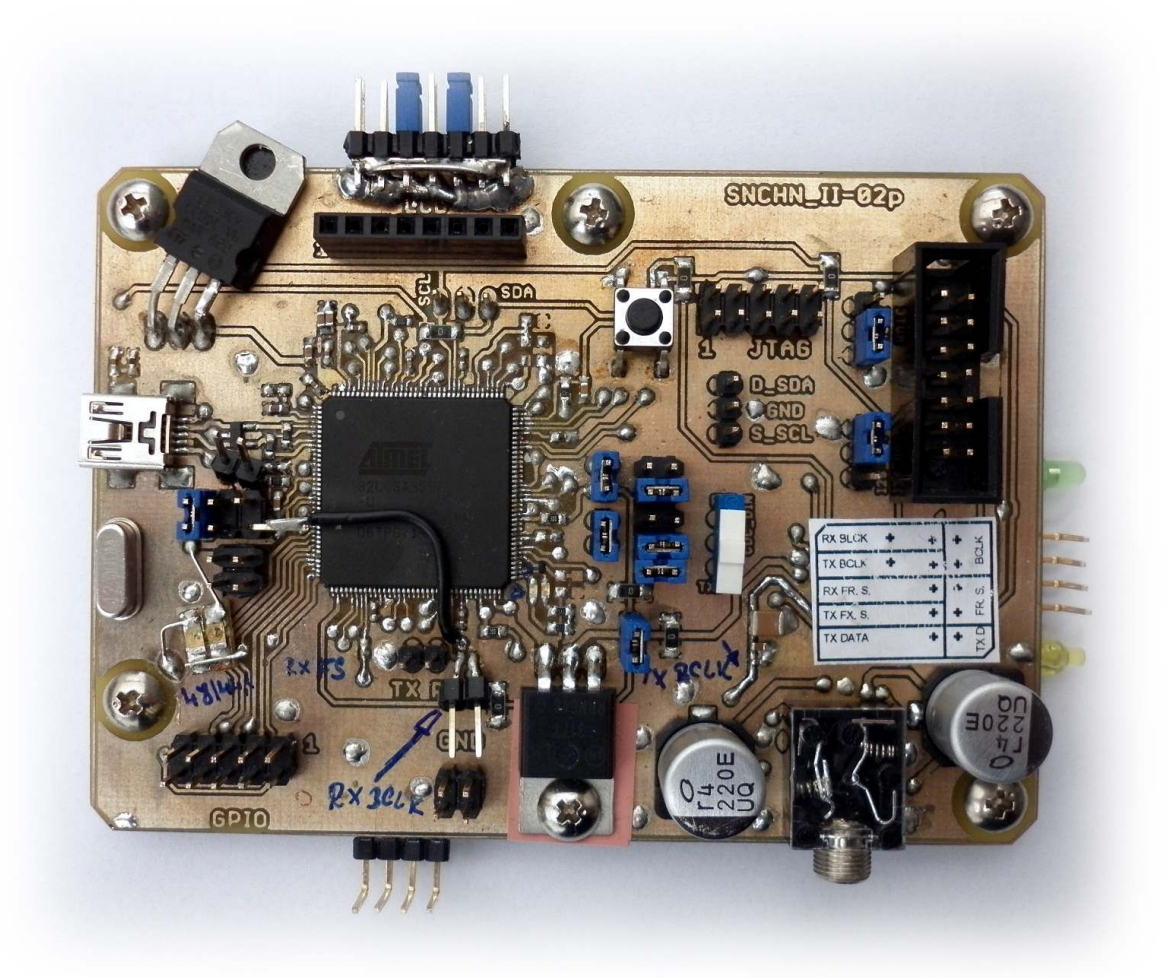
C.1. První verze zařízení v krabičce



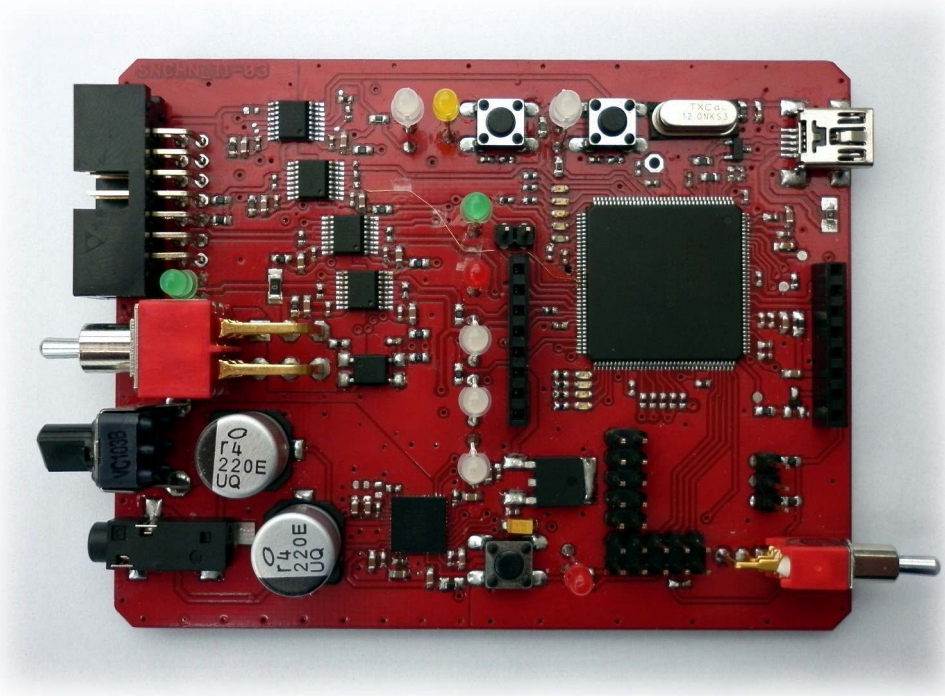
C.2. DPS první verze zařízení



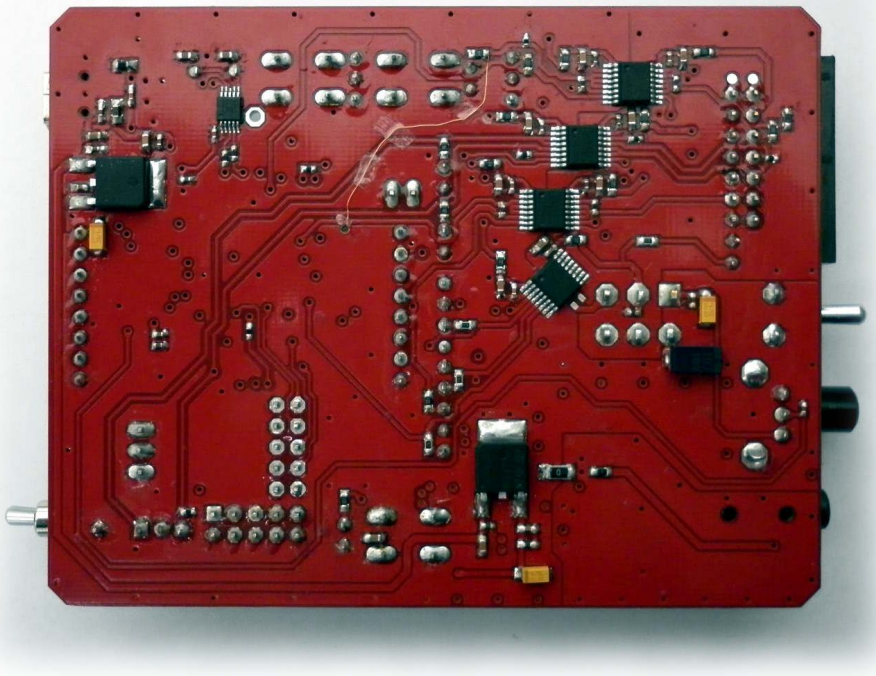
C.3. Prototypová deska pro vývoj druhé verze



C.4. DPS ze strany součástek



C.5. DPS ze strany spojů



C.6. Celé zařízení v krabičce