VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Eliška Homzová

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

REALIZACE VÝUKOVÉHO VÍCEKANÁLOVÉHO ZVUKOVÉHO OBVODU
REALIZATION OF EDUCATIONAL MULTICHANNEL SOUND GENERATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Eliška Homzová

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Petr Petyovský, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Studentka: Eliška Homzová ***ID:*** 211147

Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Realizace výukového vícekanálového zvukového obvodu

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout a realizovat programovatelný vícekanálový zvukový obvod (PSG) umožňující využití výsledné realizace jako výukové úlohy do kurzu: Logické obvody a systémy.

1. Proveďte rešerši existujících principů generování vícekanálového zvuku v počítačových systémech. Zvolte vhodnou existující architekturu programovatelného zvukového generátoru (PSG), případně navrhněte vlastní.
2. Navrhněte vnitřní blokové schéma generátoru a implementujte jej do hradlového pole. Proveďte ověření funkčnosti jednotlivých bloků pomocí simulace.
3. Navrhněte blokové uspořádání všech komponent systému tak, aby bylo možné realizovaný zvukový obvod vhodně ovládat pomocí programu v PC.
4. Realizujte vhodné komunikační rozhraní mezi zvukovým generátorem a PC.
5. Navrhněte a realizujte vhodné softwarové řešení v PC, umožňující přehrávání vícekanálového hudebního doprovodu pomocí realizovaného zvukového generátoru.
6. Na vhodných hudebních datech demonstrujte funkčnost řešení. Zpracujte podklady pro studentskou laboratorní úlohu.
7. Zhodnoťte dosažené výsledky, uveďte výhody a nevýhody jednotlivých řešení a navrhněte další rozšíření.

DOPORUČENÁ LITERATURA:

[1] PINKER, J. POUPA, M: Číslicové systémy a jazyk VHDL. 2006, ISBN 80-7300-198-5.

[2] VIRIUS, Miroslav. Jazyky C a C++: kompletní průvodce. 2., aktualiz. vyd. Praha: Grada, 2011.

ISBN 9788024739175.

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Ing. Petr Petyovský, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Tato práce se zabývá realizací výukového vícekanálového zvukového obvodu, který bude následně použit jako laboratorní úloha pro výuku v kurzu „Logické obvody a systémy“. Práce zahrnuje přehled principů generování vícekanálového zvuku, výběr vhodné architektury programovatelného zvukového generátoru a její implementaci do hradlového pole. Součástí práce je také návrh komunikace mezi vývojovými deskami a PC.

KLÍČOVÁ SLOVA

Programovatelný vícekanálový zvukový obvod, FPGA, VHDL

ABSTRACT

The main goal of this bachelor thesis is an implementation of an educational multichannel sound circuit, which will be used as a laboratory exercise in the course called "Logical circuits and systems". This paper includes an overview of principles of multichannel sound generation, selection of a suitable architecture of programmable sound generator and its implementation into the FPGA. Part of the work is also a design of communication between development boards and PC.

KEYWORDS

Programmable multichannel audio circuit, FPGA, VHDL

HOMZOVÁ, Eliška. *Realizace výukového vícekanálového zvukového obvodu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2021, 57 s. Bakalářská práce. Vedoucí práce: Ing. Petr Petyovský, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Eliška Homzová
VUT ID autora:	211147
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Realizace výukového vícekanálového zvukového obvodu

Prohlašuji, že svou závěrečnou práci jsem vypracovala samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno 24. 5. 2021 .....

podpis autorky*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Ráda bych poděkovala vedoucímu bakalářské práce panu Ing. Petru Petyovskému, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych chtěla poděkovat své rodině a přátelům za jejich podporu.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tématem této bakalářské práce jsou programovatelné zvukové generátory a jejich implementace do hradlového pole. Programovatelné zvukové generátory jsou softwarově řízené zvukové obvody, jejichž zástupcem je například zvukový obvod 6581 SID firmy Commodore používaný v domácích počítačích nebo v nízkonákladových hudebních nástrojích. Více informací k programovatelným zvukovým generátorům je možné nalézt v kapitole 1.

Cílem bakalářské práce je realizovat programovatelný vícekanálový zvukový generátor, jehož výsledná realizace bude použita jako studentská laboratorní úloha v kurzu „Logické obvody a systémy“.

Úvodní část práce se věnuje principům generování vícekanálového zvuku a výběru vhodné architektury programovatelného zvukového generátoru. Následně je vybraná architektura implementována do hradlového pole a funkčnost jednotlivých bloků ověřena pomocí simulace.

V další části práce jsou vytvořena dvě bloková uspořádání systému pro komunikaci s PC. První blokové uspořádání je realizováno s reálným zvukovým obvodem a druhé pomocí navrženého programovatelného zvukového generátoru. Dvě blokové uspořádání systému jsou realizována kvůli následnému srovnání zvukových výstupů.

Následně je vytvořena komunikace s PC, která umožní přehrávání delších skladeb.

V poslední části jsou připraveny podklady pro studentskou laboratorní úlohu, ve které budou mít studenti za úkol navrhnout vybrané funkční bloky realizovaného programovatelného zvukového generátoru.

Nakonec je funkčnost realizovaných řešení demonstrována na třech vybraných skladbách.

1 Principy generování zvuku a architektury programovatelného zvukového generátoru

V rámci této kapitoly jsou popsány vybrané způsoby generování vícekanálového zvuku a následně popsány tři existující architektury programovatelného zvukového generátoru.

1.1 Principy generování vícekanálového zvuku

V podkapitole jsou popsány vybrané metody generování vícekanálového zvuku.

1.1.1 Počítač Manchester Mark I / Ferranti Mark I

Prvním počítačem schopným generovat zvuk byl počítač Manchester Mark I, který byl postaven v roce 1949 v laboratořích Manchesterské univerzity. Na stavbě se podíleli Freddie Wiliams, Tom Kilburn a Alan Turing [3].

Počítač Ferranti Mark I vychází z konceptu počítače Manchester Mark I. Počítač Ferranti Mark I, tedy i počítač Manchester Mark I, obsluhoval reproduktor, který informoval uživatele o splnění úkolu [3].

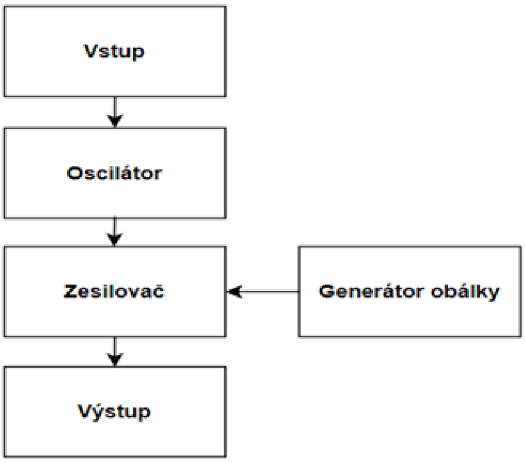
Princip fungování byl takový, že Ferranti Mark I posílal signál „zapnout“ a „vypnout“ do reproduktoru. Délka časového intervalu mezi instrukcemi „vypnout/zapnout“ byla požadovaná frekvence tónu. Toto celé bylo synchronizováno pomocí hodin procesoru s frekvencí okolo 4 kHz. Turing nikdy neuvažoval o použití Ferranti Mark I ke generování hudby. S tímto nápadem přišel až Christopher Strachey v létě 1951. První píseň zahraná počítačem byla britská hymna „God save the King“ [3].

1.1.2 Použití FM syntézy pro generování zvuku

Frekvenční modulace byla objevena v roce 1967 Johnem Chowningem na Stanfordské univerzitě. Chowning prováděl experiment, při kterém používal komplexní průběh signálů pro změny výšek tónu jednoduchých sinusových signálů [4].

FM syntéza funguje na principu frekvenční modulace, při které se pomocí modulačního signálu mění nosný signál. Základem FM syntézy jsou bloky prvků zvané operátory, které se skládají ze vstupu, oscilátoru, zesilovače a výstupu (obrázek 1.1) [4].

Jednotlivé operátory se řadí do větších bloků a mohou se navzájem ovlivňovat. Je možné připojit výstup jednoho operátoru na vstup druhého operátoru. Jeden operátor generuje modulační signál, který je připojen na vstup druhého operátoru, který generuje nosný signál [4].



Obr. 1.1: Operátor FM syntézy (upraveno z [4])

Zvukové obvody Yamaha OPL

Zkratkou OPL je označována rodina zvukových obvodů od firmy Yamaha, které využívají princip FM syntézy pro generovaní téměř realistického zvuku. Například na zvukovém obvodu YM3812 je možné současně generovat 9 zvuků, 6 melodických tónů a 5 rytmických zvuků. Možnost řízení softwarem umožňovala využití tohoto obvodu jako generátoru zvuku pro první herní počítače [5].

1.1.3 Programovatelný zvukový generátor

Programovatelný zvukový generátor, také známý pod anglickou zkratkou PSG, je softwarově řízený obvod schopný generovat tóny. Pro svoji velkou univerzálnost a relativní jednoduchost programování byl používán ve velkém množství aplikací. Typickými zástupci jsou zvukové obvody AY-3-8910 od General Instruments, ze kterých následně vychází i obvody společnosti Yamaha [6].

Na rozdíl od prvních počítačů, kde generování zvuku zajišťovalo CPU, programovatelné zvukové generátory generují zvuk sami a CPU pouze aktualizuje pole vstupně/výstupních registrů, čímž došlo k snížení zatížení CPU [6].

Programovatelné zvukové generátory se skládají ze zvukových kanálů, které generují obdélníkový signál proměnné frekvence a amplitudy. Tímto je možné precizně definovat obálkovou charakteristiku. Je možné skládat signály z více kanálů dohromady. Se správným nastavením registrů je možné generovat široký rozsah tónů a různé zvukové sekvence [6].

PSG bude podrobněji popsán v podkapitole 1.2 na architekturách vybraných zvukových obvodů.

1.1.4 Použití pulzně kódové modulace pro generování zvuku

Při pulzně kódové modulaci je vstupní analogový signál převeden na digitální, v pravidelných časových intervalech je vzorkován, kvantován a kódován do binárního kódu. Při zpětném převodu je následně z binárního kódu opět vytvořen požadovaný analogový signál [7].

1.1.5 Použití pulzně šířkové modulace pro generování zvuku

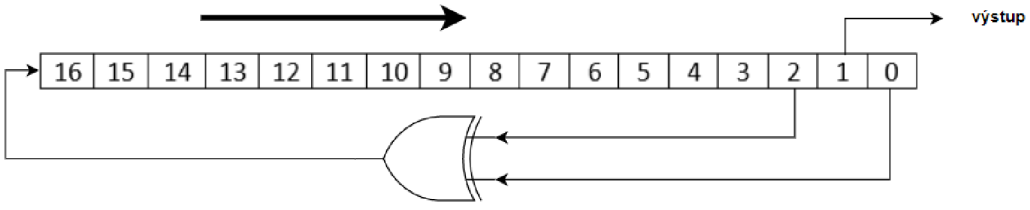
Pulzně šířková modulace je způsob modulování signálu, kde šířka impulsu výstupního signálu je funkcí amplitudy vstupního signálu. Výstupní signál nabývá logických hodnot 0 a 1. Střední hodnota výstupního modulovaného signálu představuje úroveň vstupního signálu. Střední hodnota výstupního signálu je dána poměrem periody modulace a střídy signálu [8].

Výstupní modulovaný signál je generován tak, že vstupní signál je porovnáván s referenčním pilovitým signálem. Když je úroveň vstupního signálu větší než úroveň referenčního signálu, je výstupní signál v logické 1, jinak je v logické 0. Frekvence výstupního signálu je rovna frekvenci referenčního signálu [8].

PWM lze realizovat i pomocí digitálních obvodů, kde je místo generátoru pilovitého signálu použit volně běžící čítač. PWM lze použít například v D/A převodu a výkonových audio zesilovačích [8].

1.1.6 Použití lineárního zpětnovazebního čítače pro generování šumu

Lineární zpětnovazební čítač je druh čítače, jehož zpětnou vazbu tvoří logické funkce XOR, které jsou generátorem liché parity a posloupnost maximální délky je $2^n - 1$. Jediná kombinace, která nikdy nenastane, jsou samé nuly na výstupu, proto je nutné nastavit počáteční stav na libovolnou nenulovou hodnotu [1].



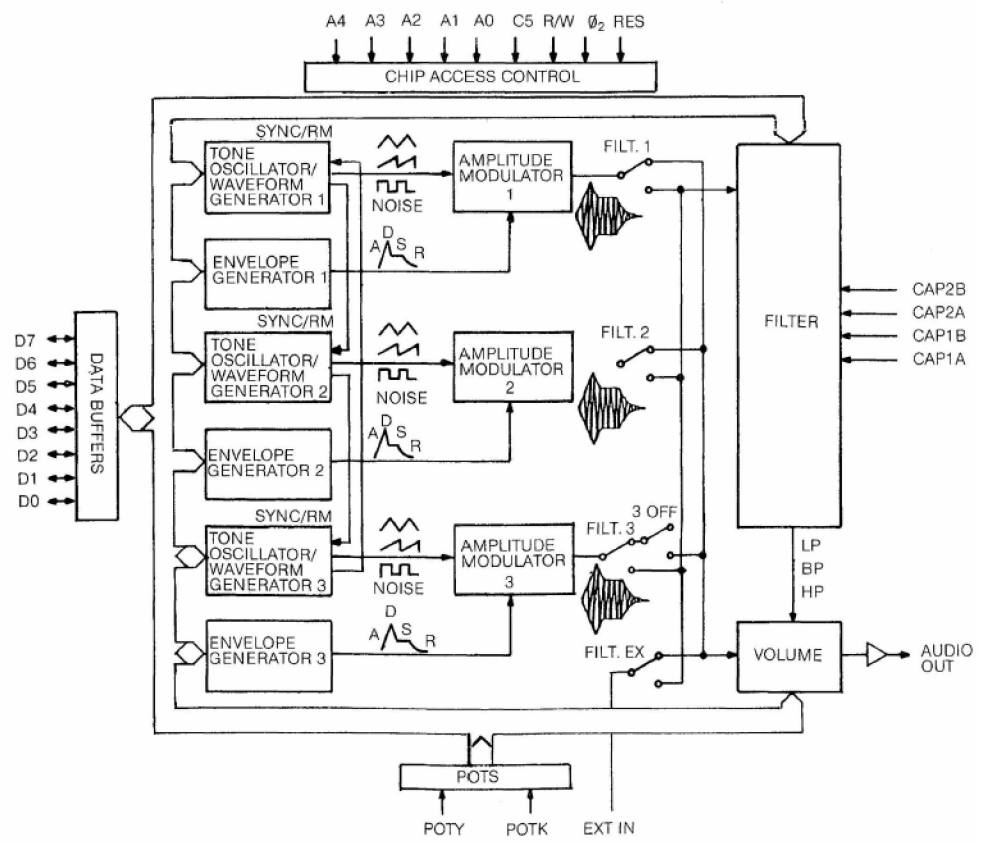
Obr. 1.2: Blokové schéma 17bitového lineárního zpětnovazebního čítače

1.2 Architektury programovatelného zvukového generátoru

Popsanými architekturami PSG jsou architektury zvukových obvodů 6581 SID firmy Commodore, POKEY C012294 firmy Atari a rodina zvukových obvodů AY-3-81X firmy General Instrument.

1.2.1 Zvukový obvod 6581 SID

Obvod 6581 SID (celým názvem 6581 Sound Interface Device) je tříkanálový programovatelný zvukový generátor od firmy Commodore. Používal se v domácích videohrách a pro nízkonákladové hudební nástroje [9].



Obr. 1.3: Blokové schéma zvukového obvodu 6581 SID [9]

Obvod 6581 SID se skládá ze tří zvukových kanálů. Každý zvukový kanál obsahuje svůj vlastní generátor signálu, generátor obálky a amplitudový modulátor. Obvod 6581 SID je schopen zpracovávat externí zvukové signály, což umožňuje řetězení nebo kombinování více obvodů 6581 SID v komplexních polyfonních systémech [9].

Řídicí registry

Řídicí registry tvoří 29 8bitových registrů, které slouží buď jen pro zápis ($R0_{16}$ až $R18_{16}$) nebo jen pro čtení ($R19_{16}$ až $R1C_{16}$). K výběru registru slouží piny A0-A4, které jsou připojeny na odpovídající adresové řádky mikroprocesoru, proto obvod 6581 SID může být adresován stejným způsobem jako paměť. Data do/z registrů se předávají pomocí obousměrných vyrovnávacích registrů [9].

Registry pro zápis jsou rozděleny do 4 skupin, kdy první 3 skupiny po 7 registrech slouží pro ovládání jednotlivých kanálů a poslední skupina 5 registrů nastavuje filtr a hlasitost výstupu [9].

Generátor signálu

Generátor signálu vytváří periodický signál s možností nastavení jednoho ze tří předdefinovaných průběhů signálu. Průběh signálu může být obdélníkový s nastavitelnou střídou, trojúhelníkový nebo pilovitý. Dále je možné pomocí generátoru signálu generovat šum [9].

Frekvence generátoru signálu se získá násobením 16bitové hodnoty dvou frekvenčních registrů ($R00_{16}$ a $R01_{16}$, $R07_{16}$ a $R08_{16}$, $R0E_{16}$ a $R0F_{16}$) hodnotou 0,0596 Hz. Standardní vstupní hodinový signál má frekvenci 1 MHz [9].

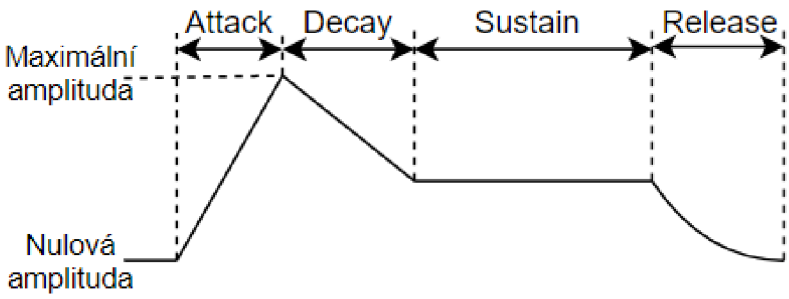
Pomocí dvou generátorů signálu lze vytvářet kruhovou modulaci, kdy výstup jednoho generátoru signálu může být modulován výstupem druhého generátoru signálu. Možnosti kruhové modulace generátorů signálu jsou následující: výstup generátoru signálu 3 moduluje výstup generátoru signálu 1, výstup generátoru signálu 2 moduluje výstup generátoru signálu 3 a výstup generátoru signálu 1 moduluje výstup generátoru signálu 2 [9].

Generátor obálky

Generátor obálky vytváří pomalu proměnnou amplitudu s nastavitelnými rychlostmi zvyšování a snižování amplitudy. Obálka amplitudy se skládá ze čtyř parametrů (Attack, Decay, Sustain a Release). Vhodným nastavením parametrů lze napodobit řadu hudebních nástrojů i mnoho unikátních zvuků [9].

Parametr Attack udává rychlost náběhu první hrany, Decay rychlost sestupu druhé hrany. Pomocí Sustain se nastavuje stabilní úroveň obálky v rozsahu 0 až 15 a Release udává rychlost klesání poslední hrany [9].

Běh generátoru obálky se zahájí zápisem logické 1 do bitu 0 kontrolního registru ($R04_{16}$, $R0B_{16}$, $R12_{16}$). Po provedení operace Attack a Decay se generátor obálky zastaví na parametru Sustain, dokud se do bitu 0 nezapíše logická 0. Následně se provede parametr Release a generátor obálky se ustálí na hodnotě 0 [9].



Obr. 1.4: Parametry generátoru obálky zvukového obvodu 6581 SID (upraveno z [9])

Modulátor amplitudy

Modulátor amplitudy vytváří amplitudu podle výstupu generátoru obálky a kombinuje ji s výstupem generátoru signálu [9].

Filtr

Mezní frekvence filtru se nastavuje pomocí dvou registrů $R15_{16}$ a $R16_{16}$. Frekvenční rozsah filtru je možno podle potřeby nastavit změnou hodnot připojených kondenzátorů CAP1, CAP2. Bity 7 až 4 registru $R17_{16}$ řídí rezonanční frekvenci filtru a rezonanci špičkového efektu, který zdůrazňuje frekvenční složky při mezní frekvenci filtru, což způsobuje ostřejší zvuk. Lze nastavit 16 úrovní rezonanční frekvence. Dolní 4 bity registru $R17_{16}$ určují, které signály budou vstupovat do filtru [9].

Lze nastavit 3 režimy. Prvním režimem je „dolní propust“, která zeslabuje frekvenční složky nad mezní frekvencí o 12 dB/oktávu. Režim „horní propust“ zeslabuje frekvenční složky pod mezní frekvencí o 12 dB/oktávu a režim „pásmová propust“ zeslabuje všechny frekvenční složky nad i pod mezní frekvencí o 6 dB/oktávu [9].

Ovládání hlasitosti

Ovládání hlasitosti nastavuje hlasitost výstupu filtru, tří zvukových kanálů a externího vstupu. Výstupní hlasitost lze nastavit na 16 úrovní podle hodnoty nastavené v dolních 4 bitech registru $R18_{16}$ [9].

1.2.2 Zvukový obvod POKEY C012294

POKEY C012294 (celým názvem Pot Keyboard Integrated Circuit C012294) je programovatelný zvukový generátor používaný v domácích počítačích ATARI 400 a 800 [10].

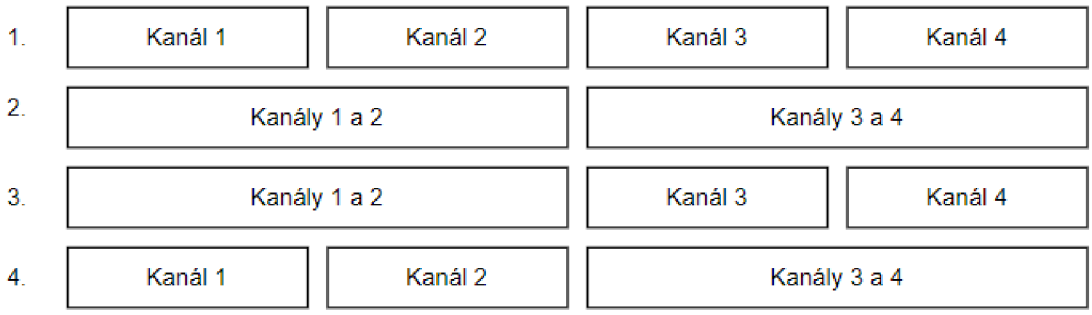
POKEY C012294 obsahuje čtyři nezávislé zvukové kanály s vlastním ovládáním frekvence, šumu a hlasitosti. Podle nastavení frekvenčních děličů lze upravit množství kanálů a jejich rozlišení. Obsahuje také generátor náhodných čísel a sériový vstupně/výstupní port [10].

Řídicí registry

POKEY C012294 obsahuje celkem 9 8bitových registrů pro řízení zvukových kanálů. Čtyři řídicí registry AUDC1-4 slouží pro nastavení šumu a hlasitosti každého ze čtyř zvukových kanálů. Frekvence jednotlivých zvukových kanálů se nastavuje pomocí čtyř frekvenčních řídicích registrů AUDF1-4. Pro nastavení vstupního hodinového signálu, filtru horní propust, generátoru šumu a sloučení kanálů slouží registr AUDCTL[10].

Generátor tónu

Ke generování tónů slouží čtyři děličky frekvence. Všechny lze taktovat frekvencí 64 kHz nebo 15 kHz, což se nastavuje pomocí bitu 0 registru AUDCTL. Děličky frekvence kanálů 1 a 3 lze taktovat i frekvencí 1,79 MHz nastavením bitů 5 a 6 registru AUDCTL. Nastavením bitů 4 a 3 registru AUDCTL lze sloučit kanály 4 a 3 a kanály 2 a 1. To umožňuje následující možnosti: 4 kanály s 8bitovým rozlišením, 2 kanály s 16bitovým rozlišením nebo 1 kanál s 16bitovým rozlišením a 2 kanály s 8bitovým rozlišením [10].



Obr. 1.5: Možnosti sloučení zvukových kanálů obvodu POKEY C012294 [10]

Generátor šumu

Ke generování šumu slouží tři lineární zpětnovazební čítače (17bitový, 5bitový, 4bitový). 17bitový lineární zpětnovazební čítač lze snížit na 9bitový pomocí bitu 7 registru AUDCTIL. Tyto čítače jsou taktovány na frekvenci 1,79 MHz.

Jejich výstupy však mohou být nezávisle vzorkovány čtyřmi zvukovými kanály rychlostí určenou děličkou frekvence každého kanálu. Lineární zpětnovazební čítače jsou řízeny bity 7 až 5 registrů AUDC1-4 [10].

Filtr horní propust

Filtr horní propust lze nastavit pro kanál 1 a 2 pomocí bitů 1 a 2 registru AUDCTL. Při bitu 1 v logické 1 bude na kanál 1 filtr propouštět pouze frekvence vyšší než je frekvence nastavená na kanálu 3. Při bitu 2 v logické 1 bude na kanál 2 propouštět pouze frekvence vyšší než je frekvence nastavená na kanálu 4 [10].

Ovládání hlasitosti

Ovládání hlasitosti je umístěno na výstupu každého kanálu. Jedná se o 4bitový D/A převodník, který umožňuje výběr z 16 možných úrovní hlasitosti. Výběr hlasitosti je řízen dolními čtyřmi bity registrů AUDC1-4. Zvukový výstup libovolného kanálu lze zcela vypnout nulovým zápisem do čtyř dolních bitů registrů AUDC1-4. Režim „Pouze ovládání hlasitosti“ lze ovládat bitem 5 registrů AUDC1-4. V tomto režimu je zvuk vytvářen pomocí pulzů posílaných mikroprocesorem [10].

1.2.3 Rodina zvukových obvodů AY-3-891X

Rodinu AY-3-891X tvoří zvukové obvody AY-3-8910, AY-3-8912 a AY-3-8913. Jednotlivé zvukové obvody se od sebe liší množstvím vstupně/výstupních pinů.

Rodina AY-3-891X jsou tříkanálové paměťově orientované programovatelné zvukové generátory. Používaly se v aplikacích jako je hudební syntéza, generování zvukových efektů, alarmů a také v osobních počítačích [11].

Řídicí příkazy jsou vkládány do programovatelného zvukového generátoru zápisem do 16 registrů. Je opatřen třemi samostatně řiditelnými analogovými výstupními kanály, které jsou opatřeny 4bitovými logaritmickými D/A převodníky, což vylepšuje dynamický rozsah produkovaného zvuku [11].

Řídicí registry

Obsah 16 registrů se nastavuje pomocí 10 adresových bitů (8 bitů z běžné datové/adresové sběrnice a 2 oddělené adresové bity) [11].

Čtyři adresové bity nižšího řádu vybírají jeden z 16 registrů ($R0_8$-$R17_8$). Šest adresových bitů vyššího řádu funguje pro výběr obvodu pro řízení třístavových obousměrných vyrovnávacích pamětí.

Výběr mezi zápisem, čtením a nastavením adresy registru zajišťují řídicí vstupy BC1, BC2, BDIR. Uzamčená adresa zůstane platná, dokud neobdrží novou adresu, což umožní více čtení a zápisů stejného obsahu registru bez nutnosti redundantního opětovného adresování [11].

Generátor tónu

Generátory tónu vytváří základní obdélníkové signály tónových frekvencí pro každý kanál. Frekvence každého obdélníkového signálu generovaného třemi generátory tónu je získávána nejprve vydělením vstupních hodin šestnácti a následným dělením výsledku 12bitovou hodnotou periody tónu. Každá 12bitová hodnota periody tónu je získána kombinováním obsahu příslušných hrubých (dolní 4 bity registrů $R1_8$, $R3_8$, $R4_8$) a jemných (8 bitů registrů $R0_8$, $R2_8$, $R4_8$) tónových registrů [11].

Generátor šumu

Generátor šumu vytváří pseudonáhodný obdélníkový signál, který je tvořen 17bitovým lineárním zpětnovazebním čítačem, jehož zpětnou vazbu tvoří exkluzivní logický součet bitů 2 a 0 čítače [12]. Vstupní frekvence pro generátor šumu je získána nejprve dělením vstupního hodinového signálu šestnácti a následným dělením výsledku 5bitovou hodnotou periody šumu. Tato 5bitová hodnota se skládá z dolních 5 bitů registru $R6_8$ [11].

Směšovače

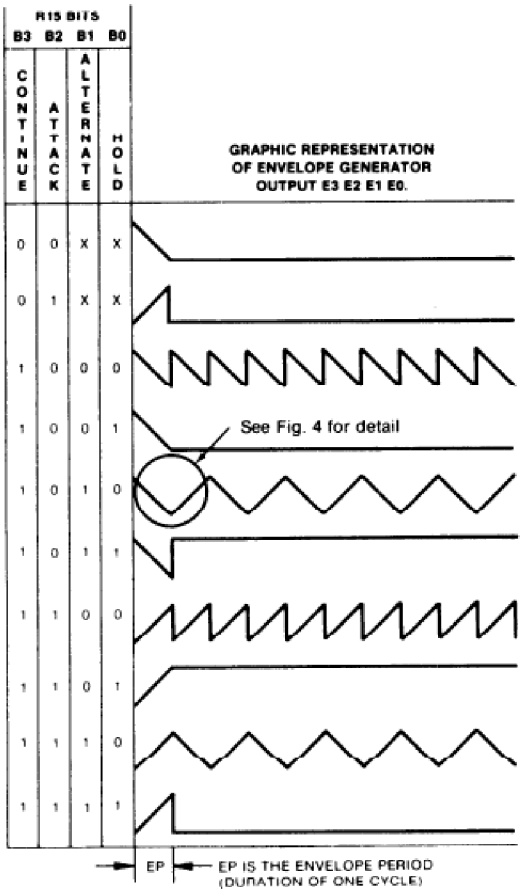
Směšovače kombinují výstupy jednotlivých generátorů tónu s generátorem šumu. Určení kombinace šumových a tónových frekvencí na každém kanálu je určena podle dolních 5 bitů registru $R7_8$. Bity 2 až 0 slouží pro povolení výstupů jednotlivých generátorů tón a bity 5 až 3 slouží pro povolení generátoru šumu. Daný výstup je povolen pokud je na příslušné pozici v registru logická 0 [11].

Generátor obálky

Generátor obálky tvoří obálkový vzor, který může být použit pro amplitudovou modulaci výstupu každého směšovače. Pomocí registrů $R13_8$ a $R14_8$ je možné měnit frekvenci obálky a registr $R15_8$ slouží pro změnu vzoru cyklu obálky [11].

Frekvence obálky se získá nejprve dělením vstupního hodinového signálu číslem 256 a následným dělením výsledku 16bitovou hodnotou, která se získá kombinací obsahu hrubých (registr $R13_8$) a jemných (registr $R14_8$) obálkových registrů [11].

Frekvence obálky se dále dělí šestnácti, čímž vytváří obálkový vzor šestnácti stavů na cyklus, který je definován výstupem 4bitového čítače. Jednotlivé vzory cyklu obálky jsou zobrazeny na obrázku 1.6 [11].



Obr. 1.6: Vzory cyklu obálky generátoru obálky rodiny AY-3-891X [11]

Ovládání hlasitosti

Ovládání hlasitosti poskytuje D/A převodníkům pevnou nebo proměnnou amplitudu. Pevná amplituda je určena dolními 3 bity registrů $R10_8$, $R11_8$, $R12_8$. Bit 4 registrů $R10_8$, $R11_8$, $R12_8$ slouží pro výběr mezi pevnou a proměnnou amplitudou z generátoru obálky [11].

D/A převodníky

Každý ze tří D/A převodníků vytváří šestnáctiúrovňový analogový výstupní signál, který je určen blokem ovládání hlasitosti. Analogový výstupní signál vytváří spolu s výstupem směšovačů požadovaný zvuk [11].

Hodnoty napětí na výstupu D/A převodníku pro 16 úrovní hlasitosti jsou uvedeny v tabulce 1.1.

Tab. 1.1: Hodnoty výstupu D/A převodníku rodiny obvodů AY-3-891X

Hlasitost kanálu	Napětí na výstupu
15	1
14	0,707
13	0,500
12	0,354
11	0,250
10	0,177
9	0,125
8	0,088
7	0,063
6	0,044
5	0,031
4	0,022
3	0,016
2	0,011
1	0,008
0	0,006

Hodnoty výstupu D/A převodníku pro jednotlivé hlasitosti byly vypočítány postupným dělením hodnoty předchozího řádku ze sloupce napětí na výstupu hodnotou $\sqrt{2}$.

1.2.4 Srovnání architektur programovatelného zvukového generátoru

V tabulce 1.2 jsou porovnány některé ze základních parametrů architektur zvukových obvodů 6581 SID, POKEY C012294, rodiny AY-3-891X.

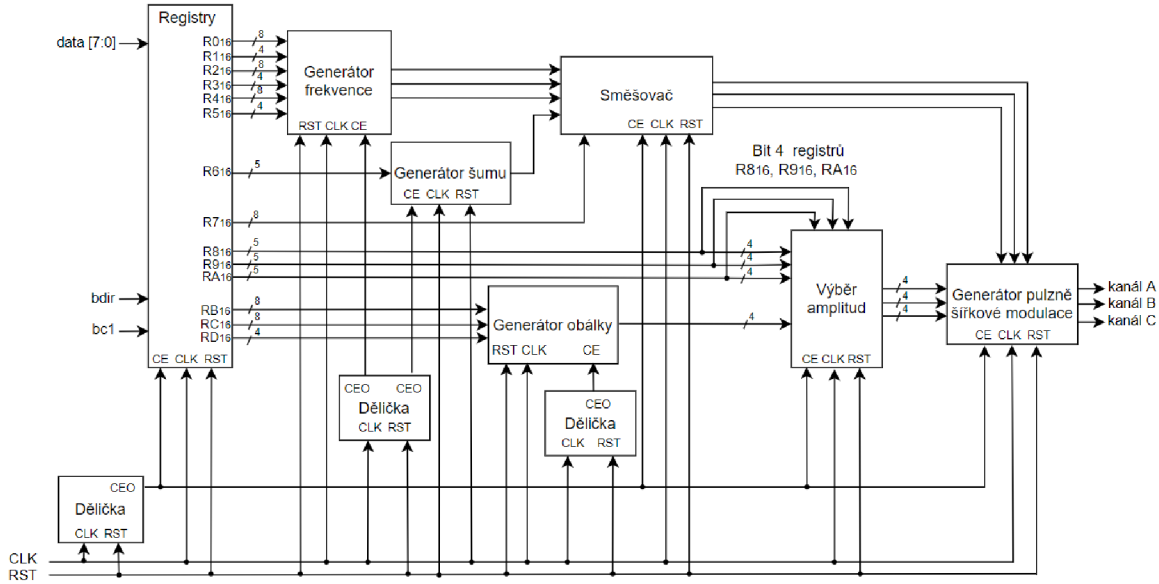
Tab. 1.2: Přehled základních parametrů architektur zvukových obvodu 6581 SID, POKEY C012294, rodiny AY-3-891X [9][10][11]

Architektura	6581 SID	POKEY C012294	AY-3-891X
Hodinový signál	1MHz	15kHz, 64kHz, 1,79MHz	1 až 2 MHz
Počet kanálů	3	2 až 4	3
Počet generátorů obálky	3	0	1
Průběh signálu	obdélníkový, trojúhelníkový, pilovitý	obdélníkový	obdélníkový
Filtry	horní, dolní, pásmová propust	horní propust	žádný

Jako vhodná architektura PSG pro implementaci do hradlového pole byla zvolena architektura rodiny zvukových obvodů AY-3-891X, protože zvuk produkovaný zvukovým obvodem POKEY C012294 je náchylný na ztrátu hlasitosti a špatnou kvalitou zvuku [10]. Argumentem proti použití zvukového obvodu 6581 SID je, že dokumentace zcela neodpovídá tomu jak je zvukový obvod realizován [13]. Dalším důvodem pro použití architektury zvukových obvodů AY-3-891X je vlastnictví hudebních dat.

2 Vnitřní blokové schéma PSG a jeho implementace do hradlového pole

Na obrázku 2.1 je zobrazeno blokové schéma PSG pro implementaci do hradlového pole. Předlohou pro hradlové pole byla architektura rodiny zvukových obvodů AY-3-891X [11].



Obr. 2.1: Blokové schéma PSG pro implementaci do hradlového pole

Realizovaný PSG je navržen tak, aby byl plně synchronní se vstupním hodinovým signálem. Všechny bloky mají připojený asynchronní reset a řídicí signál CE, který řídí provádění procesů jednotlivých bloků s požadovanou frekvencí.

Vstup *data* je 8bitový vektor, který slouží pro nastavení adresy registru a příjem dat k uložení do registrů. Zápis hodnot do registrů je řízen pomocí vstupních signálů *bdir* a *bc*1.

Blok výběru amplitud slouží pro volbu zdrojové amplitudy a to přesněji zda bude pro daný kanál použita pevná amplituda z registrů $R8_{16}$ až RA_{16} nebo proměnná z generátoru obálky. Pokud je bit 5 registrů $R8_{16}$ až RA_{16} nastaven do logické 1, vstupuje do pulzně šířkové modulace výstup generátoru obálky. Jinak do generátoru pulzně šířkové modulace vstupují pevné amplitudy.

Vstupními signály pro směšovač jsou výstupy generátoru frekvence a generátoru šumu. Směšovač má za úkol zkombinovat výstupy generátoru frekvence s výstupem generátoru šumu. Směšovač je v hradlovém poli implementován pomocí logického hradla AND, kterým se kombinují logické součty výstupu generátoru frekvence s výstupem generátoru šumu. Jednotlivé kombinace pro směšovač se zapisují do registru $R7_{16}$.

Zbylé bloky blokového schématu včetně simulace budou popsány níže v následující podkapitole.

2.1 Implementace PSG do hradlového pole

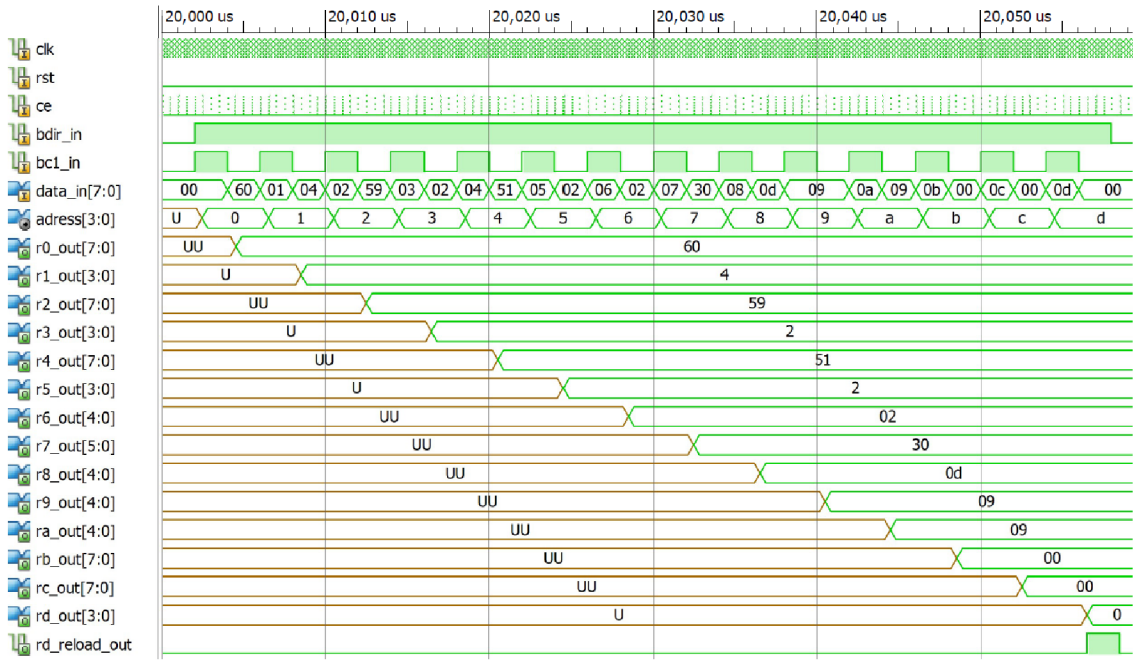
V následujícím textu jsou popsány jednotlivé bloky blokového schématu z obrázku 2.1, který je předlohou pro realizované hradlové pole. Hradlové pole bylo realizováno pomocí jazyka VHDL v návrhovém prostředí Xilinx ISE WebPack. Pro větší názornost jsou přiloženy simulace jednotlivých bloků.

2.1.1 Blok registrů PSG

Blok registrů slouží pro zápis hodnot do registrů. Pro výběr registru a zápis dat slouží jeden vstupní 8bitový vektor *data*. Jeho funkce je řízena pomocí vstupních signálů $bc1$ a $bdir$. Pokud jsou hodnoty $bdir = 1$ a $bc1 = 0$ slouží vektor *data* pro zápis hodnot do vybraného registru a při $bdir = 1$ a $bc1 = 1$ slouží pro zápis adresy registru. Pokud je nastavení řídicích vstupů jiné, bude pouze vynulována hodnota *RD_reload*. Adresa registru je vložena do vnitřní proměnné, která následně při zápisu slouží jako podmínka pro výběr registru. Jednotlivé vstupní signály a vnitřní proměnné jsou synchronizovány pomocí vstupního hodinového signálu a řídicího vstupu CE s frekvencí 2 MHz. Pokud dojde k přepisu registru RD_{16}, nastaví se hodnota výstupu *RD_reload* do logické 1, která je po ukončení zápisu do registrů opět nastavena do logické 0. Výstup *RD_reload* řídí chod generátoru obálky. Na obrázku 2.2 je zobrazena simulace jednoho zápisu do registrů.

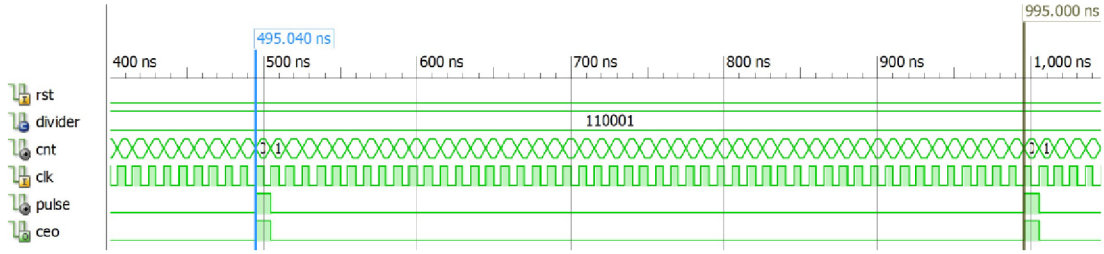
2.1.2 Dělička vstupního hodinového signálu

Dělička vstupního hodinového signálu generuje řídicí signál CEO o frekvenci dle požadavků jednotlivých bloků. Dělička je implementována synchronním čítačem pracujícím v intervalu $< 0; divider - 1 >$. Po dočítání čítače se na výstup děličky pošle kladný puls a čítač začne opět pracovat od začátku. Vstupní signál rst vynuluje čítač a nastaví výstupní signál na 0.



Obr. 2.2: Simulace zápisu do bloku registrů

Na obrázku 2.3 je znázorněna simulace děličky vstupního hodinového signálu, která generuje řídicí signál s frekvencí 2 MHz.

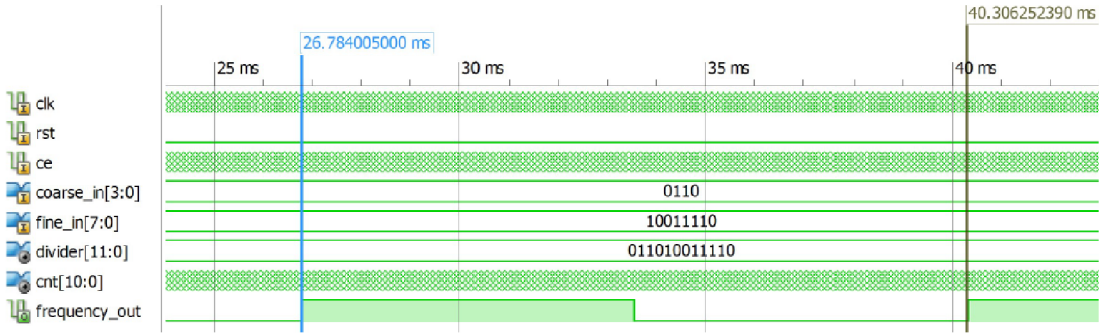


Obr. 2.3: Simulace děličky vstupního hodinového signálu na 2 MHz

2.1.3 Generátor frekvence

Generátor vytváří frekvenci tónu pro kanály A, B a C. Ze vstupních vektorů *coarse_in* (registry $R1_{16}$, $R3_{16}$, $R5_{16}$) a *fine_in* (registry $R0_{16}$,$R2_{16}$, $R4_{16}$), které slouží pro hrubé a jemné nastavení tónu, ze kterých vznikne dělitel vstupního hodinového signálu. Vektor *fine_in* tvoří 8 dolních bitů dělitele a vektor *coarse_in* tvoří horní 4 bity.

Generátor frekvence je další děličkou vstupního hodinového signálu implementovanou jako synchronní čítač. Čítač čítá s frekvencí určenou řídicím signálem CE v intervalu <0;(divider-1)/2>.

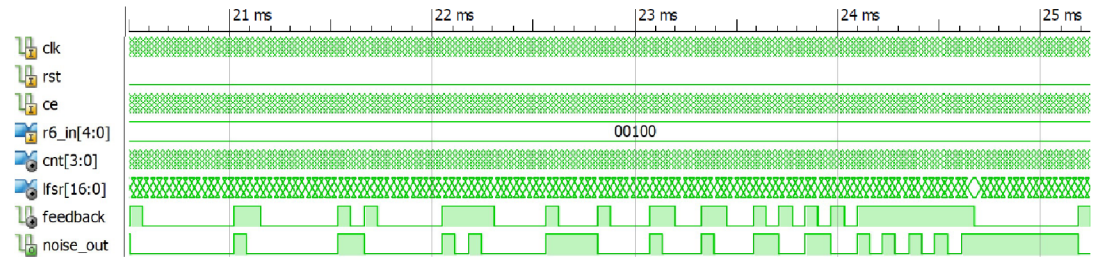


Obr. 2.4: Simulace generátoru frekvence pro frekvenci 73,8 Hz

2.1.4 Generátor šumu

Generátor šumu je implementován pomocí 17bitového lineárního zpětnovazebního čítače. Frekvence čítání je vytvořena děličkou frekvence jejíž dělitelem je obsah registru $R6_{16}$. Čítač děličky frekvence čítá s frekvencí určenou řídicím signálem CE. Počáteční hodnota lineárního zpětnovazebního čítač je $1FFF_{16}$.

Zpětnou vazbu tvoří exkluzivní logický součet bitů 2 a 0 lineárního zpětnovazebního čítače. Zpětná vazba se ukládá do bitu 16 a výstupem generátoru šumu je hodnota bitu 1.



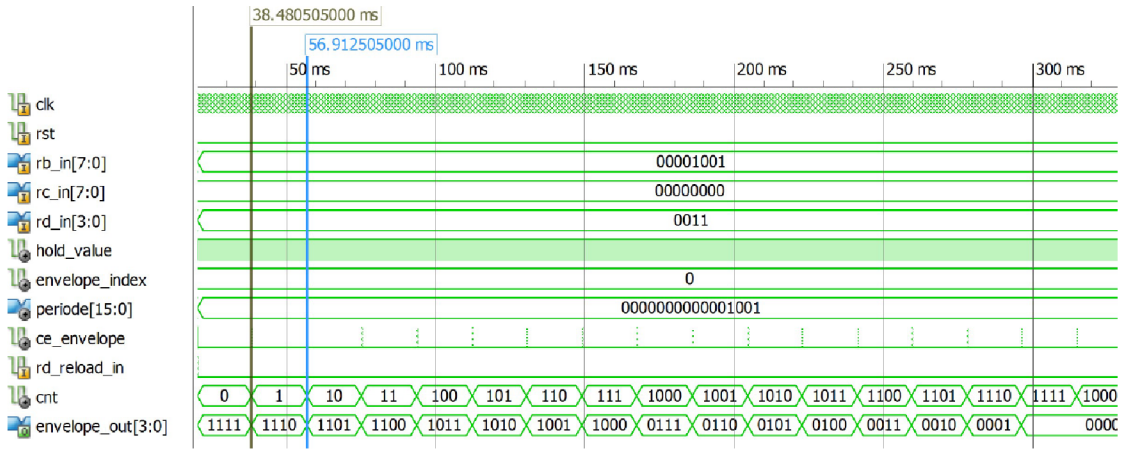
Obr. 2.5: Simulace periody generátoru šumu

2.1.5 Generátor obálky

Generátor obálky vytváří proměnnou amplitudu pro výstup směšovače. Frekvenci změny amplitudy tvoří dělička vstupního hodinového signálu. Dělitelem vstupního hodinového signálu je násobek čísla 204 800 s 16bitovou hodnotou tvořenou registry RB_{16} (horních 8 bitů) a RC_{16} (dolních 8 bitů). Číslo 204 800 vzniklo násobením čísel 50, 256 a 16. Dělením vstupního hodinového signálu násobkem čísel 50 a 256 by se vytvořila frekvence, se kterou pracuje generátor obálky. Číslem 16 se vytváří obálkový vzor šestnácti stavů na cyklus.

Tvary cyklu amplitudy jsou uloženy jako konstanty v 2D poli, jehož sloupce tvoří dvě periody vzoru cyklu amplitudy a každý řádek odpovídá jednomu nastavení kontrolního registru RD_{16}. Pro tvar obálky odpovídá nastavení kontrolního registru RD_{16} nastavení registru $R15_8$ rodiny zvukových obvodů AY-3-891X podle obrázku 1.6.

Při *RD_reload* v logické 1 se nastaví odpovídající řádek tabulky, čítač se nastaví do 0 a pokud se nemá obálka opakovat, tak se nastaví proměnná *hold_value* do logické 1. Přepínání mezi sloupci tabulky zajišťuje synchronní čítač čítající s frekvencí změny amplitudy v intervalu <0;31>. Pokud je proměnná *hold_value* v logické 1, tak se čítač zastaví na hodnotě 31, kde zůstane do dalšího přepisu registru RD_{16}.



Obr. 2.6: Simulace generátoru obálky pro hodnotu registru RD_{16} = ”0011”

2.1.6 Generátor pulzně šířkové modulace

V generátoru pulzně šířkové modulace je převedena hodnota amplitudy z rozsahu 16 úrovní hlasitosti do rozsahu 256 úrovní hlasitosti. Odpovídající hodnoty jsou v převodní tabulace 2.1.

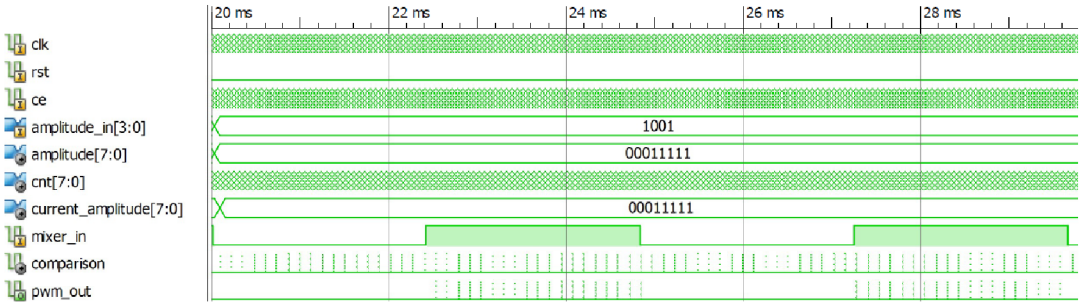
Tab. 2.1: Hodnoty převodu amplitudy výstupního signálu

Hlasitost kanálu	Napětí na výstupu	Hodnota pro PWM
15	1	255
14	0,707	180
13	0,500	127
12	0,354	90
11	0,250	63
10	0,177	44
9	0,125	31
8	0,088	22
7	0,063	15
6	0,044	10
5	0,031	7
4	0,022	5
3	0,016	3
2	0,011	2
1	0,008	1
0	0,006	0

Hodnoty pro pulzně šířkovou modulaci byly vypočítány postupným dělením hodnoty předchozího řádku ze sloupce napětí na výstupu hodnotou $\sqrt{2}$ a následným násobením zaokrouhlené hodnoty napětí na výstupu hodnotou 256.

Převedená hodnota amplitudy je uložena do proměnné *current_amplitude* při zahájení dalšího čítání čítače pracujícího v intervalu <0; 255>.

Pokud je hodnota *current_amplitude* menší nebo rovna hodnotě čítače, tak je hodnota *comparison* logická 0 a jinak logická 1. Výstup pulzně šířkové modulace je tvořen logickým součinem hodnoty *comparison* a výstupu směšovače *mixer_in*.



Obr. 2.7: Simulace generátoru pulzně šířkové modulace

3 Realizace blokového uspořádání systému pro ovládání pomocí aplikace v PC

V rámci práce byla realizována dvě bloková uspořádání systému, aby bylo možné porovnání zvukových výstupů reálného zvukového obvodu AY-3-8912A s navrženým PSG. Prvním blokovým uspořádáním je zapojení s vývojovou deskou Arduino Uno a skutečným zvukovým obvodem AY-3-8912A. Druhým je zapojení realizované pomocí vývojové desky Nexys 3 rozšířené modulem Pmod AMP2.

3.1 Blokové uspořádání systému se zvukovým obvodem AY-3-8912A

V následujícím textu je popsána realizace blokového uspořádání systému se zvukovým obvodem AY-3-8912A.

3.1.1 Vývojová deska Arduino Uno

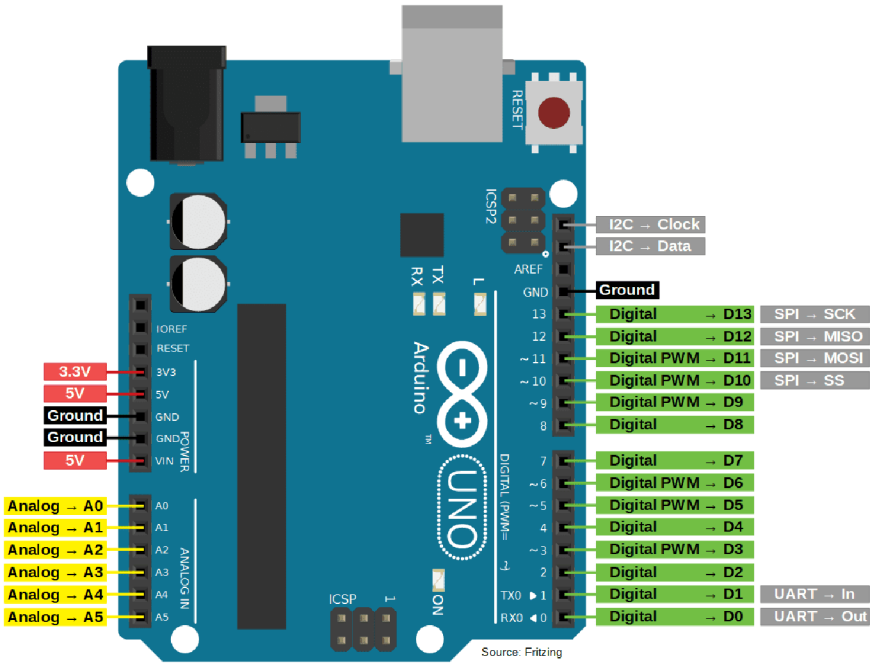
Arduino Uno je vývojová deska založena na mikroprocesoru ATmega328P a pracuje na frekvenci 16 MHz. Obsahuje 14 digitálních vstupně/výstupních pinů a 6 analogových vstupních pinů, které lze použít i jako digitální vstupně/výstupní piny. Ze 14 digitálních vstupně/výstupních pinů lze 6 použít jako PWM výstup [14]. Všech 20 pinů je rozděleno do tří vstupně/výstupních registrů, které umožňují okamžitý paralelní zápis nebo čtení [15].

K dispozici je 32 KB flash paměti, 2 KB SRAM paměti a 1 KB paměti EEPROM. Deska má provozní napětí +5 V a lze ji napájet pomocí externího napájení nebo přes USB kabel [14].

Mikroprocesor ATmega328P poskytuje sériovou komunikaci UART TTL, která je vyvedena na digitální piny 0 a 1. Jejich propojení s USB konektorem poskytuje mikroprocesor ATmega16U2, díky němuž se softwaru počítače vývojová deska jeví jako virtuální port [14].

Dále mikroprocesor ATmega328P obsahuje dva 8bitové a jeden 16bitový časovač, které lze použít například na tvorbu rychlé PWM a funkce přerušení [14].

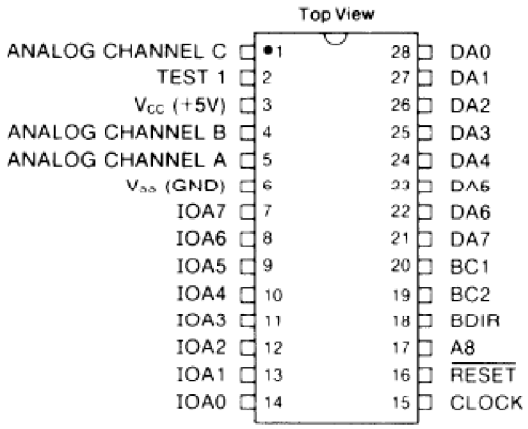
Vývojová deska Arduino Uno byla vybrána z důvodu nízké ceny a rozšiřitelnosti hardwaru.



Obr. 3.1: Rozložení pinů Arduino Uno [16]

3.1.2 Rozložení pinů zvukového obvodu AY-3-8912A

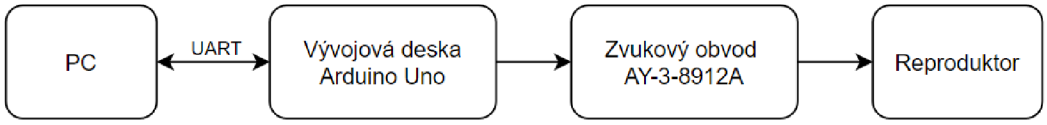
Na obrázku 3.7. lze vidět rozložení pinů zvukového obvodu AY-3-8912A.



Obr. 3.2: Rozložení pinů zvukového obvodu AY-3-8912A [11]

3.1.3 Realizace blokového uspořádání systému se zvukovým obvodem AY-3-8912A

Realizované blokové uspořádání se skládá ze dvou hlavních bloků. Z vývojové desky Arduino UNO, která zprostředkovává komunikaci mezi programem v počítači a zvukovým obvodem a ze zvukového obvodu AY-3-8912A, který generuje zvuk. Napájení a komunikaci s PC zajišťuje USB kabel.

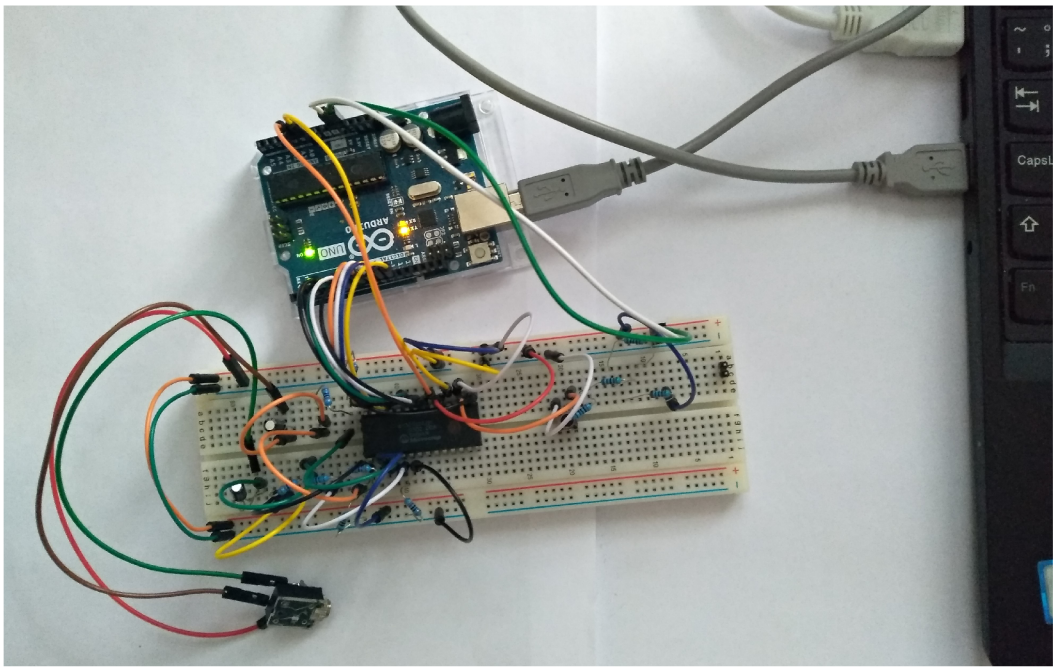


Obr. 3.3: Blokové schéma uspořádání systému se zvukovým obvodem AY-3-8912A

Vstupní hodinový signál pro zvukový obvod je generován 8bitovým časovačem mikroprocesoru ATmega328P. Pro resetování zvukového obvodu bylo k pinu 16 připojeno tlačítko, které umožňuje okamžitě resetovat zvukový obvod bez závislosti na prováděném programu. Frekvence posílání dat do zvukového obvodu je generována pomocí 16bitového časovače mikroprocesoru ATmega328P, po jehož uplynutí je zavolána funkce přerušení. Funkce přerušení je volána z frekvencí 50 Hz, což je požadovaná frekvence aktualizace registrů zvukového obvodu pro vybraná hudební data.

Data pro aktualizaci registrů zvukového obvodu jsou do vývojové desky posílána přes asynchronní sériovou komunikaci, kde jsou zpracována a následně poslána do zvukového obvodu. Data jsou posílána z vývojové desky do zvukového obvodu pomocí tří vstupně/výstupních registrů a výstupy tří kanálů zvukového obvodu jsou zapojeny tak, aby vytvářeli stereo zvuk. Schéma zapojení je zobrazeno na obrázku A.1.

Funkčnost realizovaného blokové uspořádání systému byla ověřena na třech skladbách a je funkční.



Obr. 3.4: Fotografie zapojení vývojové desky Arduino Uno se zvukovým obvodem AY-3-8912A

3.2 Blokové uspořádání systému s vývojovou deskou Nexys 3

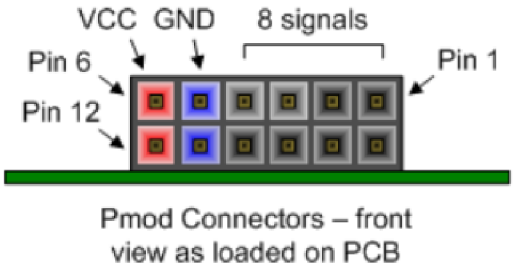
V následujícím textu je popsána realizace blokového uspořádání systému s vývojovou deskou Nexys 3.

3.2.1 Vývojová deska Nexys 3

Nexys 3 je vývojová deska založena na obvodu FPGA Spartan 6. Jako vstupní hodinový signál pro obvod FPGA Spartan 6 slouží CMOS oscilátor s frekvencí 100 MHz.

Vývojová deska obsahuje 8 přepínačů a LED diod, 4 číselné sedmi-segmentové displeje a 5 tlačítek. Dále je deska osazena VGA konektorem, 10/100 Ethernet konektorem a třemi USB konektory. První USB konektor slouží například pro připojení myši nebo klávesnice. Druhý je využíván pro nahrávání programu a třetí slouží jako sériový port (UART) [17].

Vývojovou desku lze rozšířit pomocí 68pinového VHDC konektoru nebo 5 Pmod konektory. Pmod konektor se skládá z 12 pinů z nichž jsou 2 napájecí, 2 zemnící a 8 datových [17].



Obr. 3.5: Rozložení pinů Pmod konektoru

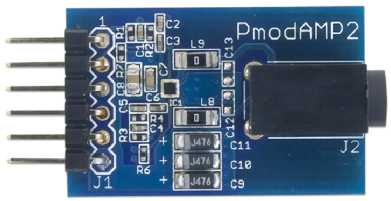
3.2.2 Modul Pmod AMP2

Modul Pmod AMP2 využívá audio zesilovač SSM2377 třídy D, který řídí monofonní výstup pomocí analogového nebo digitálního vstupu. Modul Pmod AMP2 komunikuje s vývojovou deskou pomocí GPIO protokolu. Na výstupu lze nastavit zisk 6 dB (logická 1) nebo 12 dB (logická 0). Dále lze pomocí vstupu shutdown nastavit režim nízké spotřeby energie (logická 0) [18].

Modul Pmod AMP2 byl vybrán, protože je kompatibilní s vývojovou deskou Nexys 3.

Tab. 3.1: Rozložení pinů modulu Pmod AMP2 [18]

Pin	Signál	Popis signálu
1	AIN	Audio vstup
2	GAIN	Výběr zisku
3	NC	Nepřipojen
4	~ SHUTDOWN	Aktivace režimu nízké spotřeby energie
5	GND	Zem
6	VCC	Napětí



Obr. 3.6: Modul Pmod AMP2 [18]

3.2.3 Realizace blokového uspořádání systému s vývojovou deskou Nexys 3

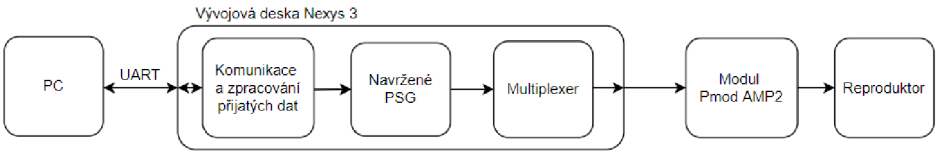
Blokové uspořádání se skládá z vývojové desky Nexys 3 a modulu Pmod AMP2, který je k vývojové desce připojen pomocí Pmod konektoru. Pro napájení a nahrání programu do vývojové desky slouží jeden USB kabel. Druhý USB kabel slouží ke komunikaci s počítačem.

Zvuk je generován pomocí PSG navrženého v kapitole 2. Výstupy PSG jsou připojeny na multiplexer, který posílá na výstup modulu Pmod AMP2 postupně jednotlivé výstupy.

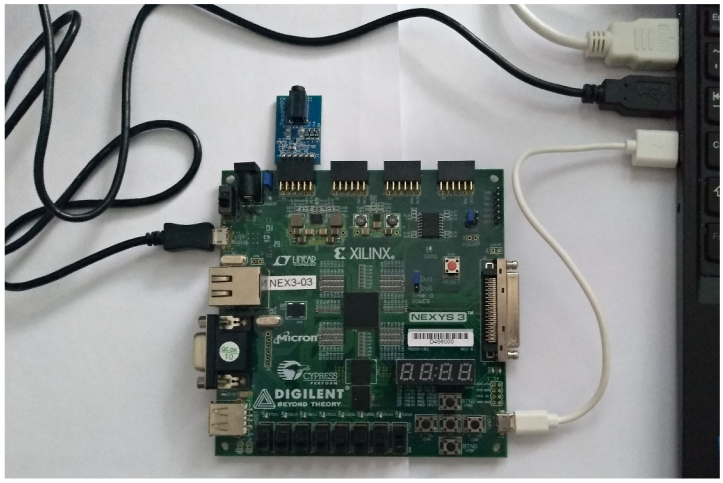
Řídicí vstupy multiplexoru jsou přepínány s taktovací frekvencí 66 kHz, což je frekvence vyšší než je lidský sluch schopen vnímat, díky čemuž můžeme slyšet všechny tři kanály najednou.

Funkčnost realizovaného blokového uspořádání bez komunikace s PC, byla ověřena na třech skladbách a je funkční. Zvukový výstup je totožný se zvukovým výstupem s blokovým uspořádáním systému se zvukovým obvodem AY-3-8912A.

Návrh komunikace s počítačem je vysvětlen v kapitole 4.5.



Obr. 3.7: Blokové schéma uspořádání systému s vývojovou deskou Nexys 3



Obr. 3.8: Fotografie zapojení vývojové desky Nexys 3 s modulem Pmod AMP2

4 Komunikační rozhraní mezi zvukovým generátorem a PC

Jako vhodné komunikační rozhraní byla zvolena asynchronní sériová komunikace, protože obě vývojové desky podporují UART komunikaci přes USB port.

Sériová komunikace mezi vývojovou deskou Nexys 3 a PC nebyla z časových důvodu realizována a proto byla nahrazena uložením hudebních dat do knihovních balíků.

Přes sériovou komunikaci jsou posílány hodnoty pro jednu aktualizaci registrů zvukového obvodu, které jsou rozšířeny o hlavičku a kontrolní součet kvůli kontrole správnosti dat přijatých vývojovou deskou. Opětovné odeslání hodnot a odeslání hodnot následujících je řízeno pomocí potvrzovacích zpráv odeslaných vývojovou deskou.

Rychlost přenosu dat sériové komunikace byla zvolena tak, aby nenastal problém při aktualizaci registrů s frekvencí 50 Hz.

Nastavení asynchronní sériové komunikace:

- Rychlost přenosu dat: 19 200 bps
- Délka rámce: 8 bitů
- Počet stop bitů: 1
- Paritní bit: žádný

4.1 Konstrukce dat odesílaných programem v PC

Hodnoty 14 registrů jsou před odesláním rozšířeny o hlavičku a kontrolní součet. Hlavičku tvoří hodnoty 81_{16}, 82_{16}, 83_{16}, které byly zvoleny tak, aby se nemohly v daném pořadí vyskytnout v prvních třech registrech PSG. Kontrolní součet je realizován exkluzivním logickým součtem odeslaných hodnot.

Tab. 4.1: Konstrukce dat odesílaných programem v PC

81_{16}	82_{16}	83_{16}	$R0_{16}$ až RD_{16}	Kontrolní součet

4.2 Konstrukce potvrzovacích zpráv vývojové desky

Potvrzovací zprávy vývojové desky určují jestli má program v PC poslat stejná data znovu nebo poslat následující data. Zprávy jsou rozšířeny o hlavičku a kontrolní součet. Hlavička je tvořena hodnotou 83_{16} a kontrolní součet je tvořen exkluzivním logickým součtem dvou hodnot tvořících odpověď. Při žádosti o opětovné zaslání dat se posílají hodnoty 41_{16}, 47_{16} a při žádosti o zaslání následujících dat se posílají hodnoty 52_{16} a 45_{16}. Posílané hodnoty byli vybrány tak, aby se nikde neopakovali.

Tab. 4.2: Konstrukce potvrzovacích zpráv vývojové desky

83_{16}	potvrzovací zpráva	Kontrolní součet

4.3 Realizace komunikace na straně PC

Program realizující komunikaci na straně PC je vytvořen v jazyce C++ a využívá hlavičková soubor knihovny $< windows.h >$.

Po spuštění programu se zavolá konstruktor třídy Serial, která obsahuje funkce pro obsluhu virtuálního sériového portu a metody pro odesílání a příjímání sériové komunikace. Následně se ve funkci *main*() deklaruje proměnná *No_Finale*, do které se následně vloží vypočítané množství přepisů registrů. Dalšími deklarovanými proměnnými jsou *No_Actual* pro uložení čísla aktuálního přepisu registrů, *message_sent* pro kontrolu odeslání dat, *checksum* pro výpočet kontrolního součtu a *send_again* pro uložení potvrzovacích zpráv vývojové desky.

Jako poslední se vytvoří dvě pole datového typu *uint*8_*t*. První pole tvoří 18 prvků a slouží pro uložení odesílaných dat včetně hlavičky a kontrolního součtu. Druhé pole tvoří 14 prvků a prvky tohoto pole jsou ukazateli na 14 polí s hodnotami k přepisu registrů.

Hlavní část programu tvoří cyklus *while*, který se provádí dokud je hodnota *No_Actual* menší než hodnota *No_Finale*. Operace uvnitř cyklu *while* jsou rozděleny do dvou částí pomocí konečného stavového automatu vytvořeného funkcí *switch* a proměnnou *Com* výčtového typu *enum class*. Stavový automat má stavy *Send* a *Answer*.

Ve stavu *Send* jsou ukládány aktuální hodnoty registrů do pole *message*, vypočítán kontrolní součet a volána metoda *Write*(*message*), která odešle připravené data. Po úspěšném odeslání dat přejde stavový automat do stavu *Answer*, ve kterém přijme odpověď od vývojové desky. Příjem potvrzovací zprávy provádí metoda *Read*(), která zpracuje přijatá data a uloží odpověď do proměnné *send_again*.

Pokud je proměnná *send_again* rovna 1, tak se vrátí stavový automat zpět do stavu *Send* bez navýšení hodnoty *No_Actual*, takže se budou znovu posílat stejná data. Pokud je proměnná *send_again* rovna 0, tak se před návratem do stavu *Send* navýší hodnota *No_Actual* a budou se odesílat data následující.

Pokud dojde k chybě při odesílání dat nebo při přijímaní potvrzovací zprávy, tak se do proměnné *No_Actual* uloží hodnota *No_Finale* a tím dojde k ukončení cyklu *while*.

Po ukončení cyklu *while* se zavolá destruktor třídy Serial, který uzavře otevřený virtuální sériový port a ukončí se program.

Algoritmus komunikace na straně PC je zobrazen na obrázku 4.1.

4.3.1 Implementace třídy Serial

Ve třídě Serial jsou použity funkce $CreateFile()$, $CloseHandle()$, $SetCommState()$, $WriteFile()$ a $ReadFile()$ hlavičkového souboru knihovny $< windows.h >$.

Konstruktor a destruktor třídy Serial

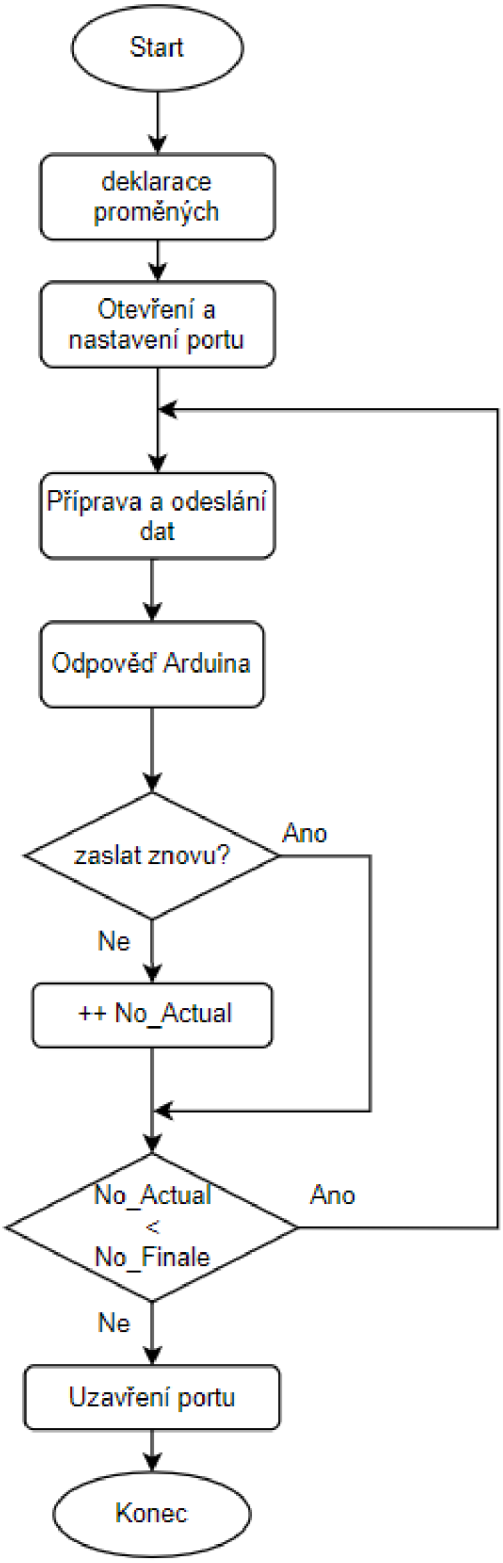
Konstruktoru třídy Serial otevírá pomocí funkce $CreateFile()$ virtuální sériové porty počítače dokud nenajde virtuální port, ke kterému je připojena vývojová deska.

Následně se nastaví parametry sériové komunikace pomocí funkce *SetCommState()*. Nastavení virtuálního portu je uloženo v proměnné *port* datového typu $HANDLE$.

Destruktor třídy Serial uzavře virtuální port pomocí funkce $CloseHandle()$.

Metoda Write()

Metoda $Write(message)$ má návratovou hodnotu typu *bool* a vstupním parametrem je ukazatel na pole *message*. V metodě jsou deklarovány proměnné *BytesToWriteOut* typu $DWORD$ určující kolik bude odesláno bytů, *message_sent* typu *bool* do níž se ukládá výsledek volané funkce $WriteFile()$, která odesílá data. Pokud je proměnná *message_sent* rovna $false$, tak došlo k chybě při odesílání dat a návratovou hodnotou metody $Write(message)$ bude hodnota $false$. Při úspěšném odeslání dat je návratovou hodnotou metody $Write(message)$ hodnota $true$.



Obr. 4.1: Vývojový diagram komunikace na straně PC

Metoda Read()

Metoda *Read*() má návratovou hodnotu typu *size_t*. První jsou deklarovány proměnné *message* typu *uint*8*_t* pro uložení přijaté hodnoty, *ByteWriteIn* typu *DWORD* určující kolik bytů bude přijato, *read* typu *size_t*, *while_en* typu *bool*, *checksum* a *checksum_received* datového typu *site_t*.

Hlavní část metody *Read*() tvoří cyklus *while*, který je prováděn dokud není změněna hodnota proměnné *while_en* na 1. Uvnitř funkce *while* je konečný stavový automat tvořený funkcí *switch* a proměnou *Received* výčtového typu *enumclass*. Stavový automat má stavy *Start* a *Data*.

Ve stavu *Start* se pomocí funkce *ReadFile*() přijímají data ze sériové komunikace dokud se přijatá hodnota nerovná hlavičce potvrzovací zprávy vývojové desky. Následně přejde automat do stavu *Data*, ve kterém je postupně přijata odpověď vývojové desky a je z ní vypočítán kontrolní součet *checksum*. Následně je do proměnné *checksum_received* uložena hodnota přijatého kontrolního součtu a ukončen cyklus *while* změnou proměnné *while_en*.

Pokud se proměnné *checksum* a *checksum_received* rovnají a *checksum* má hodnotu 23, tak je výstupním parametrem metody hodnota 0, jinak je výstupním parametrem metody hodnota 1. Pokud došlo k chybě při přijímaní potvrzovací zprávy, tak je návratovou hodnotou 2. Hodnota 23 je získána exkluzivním logickým součtem hodnot 52_{16} a 45_{16}, které tvoří potvrzovací zprávu pro odeslání následujících dat.

4.4 Realizace komunikace na straně vývojové desky Arduino Uno

Program pro komunikaci na straně vývojové desky Arduino Uno byl vytvořen ve vývojovém prostředí Arduino IDE. Algoritmus komunikace je na obrázku 4.3 a algoritmus provádění funkce přerušení na obrázku 4.2.

Po nahrání programu do vývojové desky se jako první deklarují globální proměnné *checksum*, *checksum_received*, *serial_read* typu *uint*8*_t* a proměnná *while_en* typu *size_t* . Následně se deklaruje jednorozměrné pole *data* typu *uint*8*_t* o 14 prvcích. Nakonec se deklaruje proměnná *states* výčtového typu *enum*.

Ve funkci *setup*() se nastaví sériová komunikace, vstupně/výstupní registry a zavolají se funkce *setup_clk*() a *interrupt*(). Funkce *setup_up*() nastavuje 8bitový časovač, který generuje vstupní hodinový signál pro zvukový obvod o frekvenci 2 MHz. Funkce *interrupt*() nastavuje 16bitový časovač, který volá funkci přerušení s frekvencí 50 Hz.

Následně se začne opakovaně vykonávat funkce *loop*(). Ve funkci *loop*() se při splnění podmínky, že vývojová deska přijala 18 bytů, začne provádět cyklus *while*, který se vykonává dokud je proměnná *while_en* rovna 0.

Uvnitř cyklu *while* je vytvořen konečný stavový automat pomocí funkce *switch*, jehož stavy jsou *Start*1, *Start*2, *Start*3, *Data* a *Check*.

Tab. 4.3: Přehled stavů konečného stavového automatu komunikace na straně vývojové desky Arduino Uno

Stav	Funkce
*Start*1	Nalezení první hodnoty hlavičky přijatých dat (hodnota 81_{16})
*Start*2	Kontrola druhé hodnoty hlavičky přijatých dat (hodnota 82_{16})
*Start*3	Kontrola třetí hodnoty hlavičky přijatých dat (hodnota 83_{16})
Data	Přijetí a uložení hodnot registrů
Check	Kontrola kontrolního součtu přijatých dat

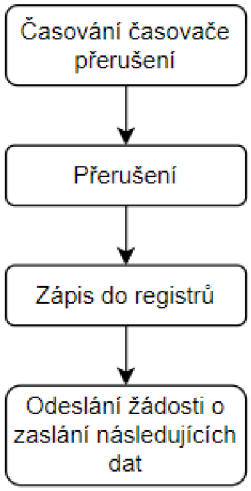
Prvním prováděným stavem je stav *Start*1, který se provádí dokud není hodnota uložená v proměnné *serial_read* rovna hodnotě 81_{16}. Po splnění podmínky pro přechod do dalšího stavu přejde stavový automat do stavu *Start*2, kde pokud je následující přečtená hodnota 82_{16}, přejde do stavu *Start*3. Jinak se automat vrátí zpátky do stavu *Start*1. Ve stavu *Start*3 se přečtená hodnota porovná s hodnotou 83_{16} a pokud se rovná, tak automat přejde do stavu *Data*. Jinak se vrátí zpátky do stavu *Start*1.

Ve stavu *Data* se uloží hodnoty registrů do pole *data* a do proměnné *checksum_received* se uloží přijatá hodnota kontrolního součtu. Po stavu *Data* následuje stav *Check*, ve kterém se zavolá funkce *Checksum*(*data*).

Ve funkci *Checksum*(*data*) se vypočítá z přijatých dat kontrolní součet *checksum*. Dále jsou ve stavu *Check* porovnány proměnné *checksum* a *checksum_received*.

Pokud se nerovnají, tak se zavolá funkce *send*(2), která pošle žádost o opětovné zaslání dat. Následně se do proměnné *while_en* uloží 1, čímž se ukončí provádění cyklu *while*.

Žádost o následující data je posílána ve funkci přerušení, která se volá po dočasování 16bitového časovače. Ve funkci přerušení se jako první zapíší aktuální hodnoty do registrů pomocí opakovaného volání funkcí *adress_write*(*číslo registru*) a *data_write*(*odpovídající prvek pole*) a následně se zavolá funkce *send*(1). Po zavolání funkce *send*(1), se nastaví proměnná *while_en* zpět do 0.



Obr. 4.2: Vývojový diagram funkce přerušení komunikace na straně vývojové desky Arduino Uno

4.5 Návrh komunikace na straně vývojové desky Nexys 3

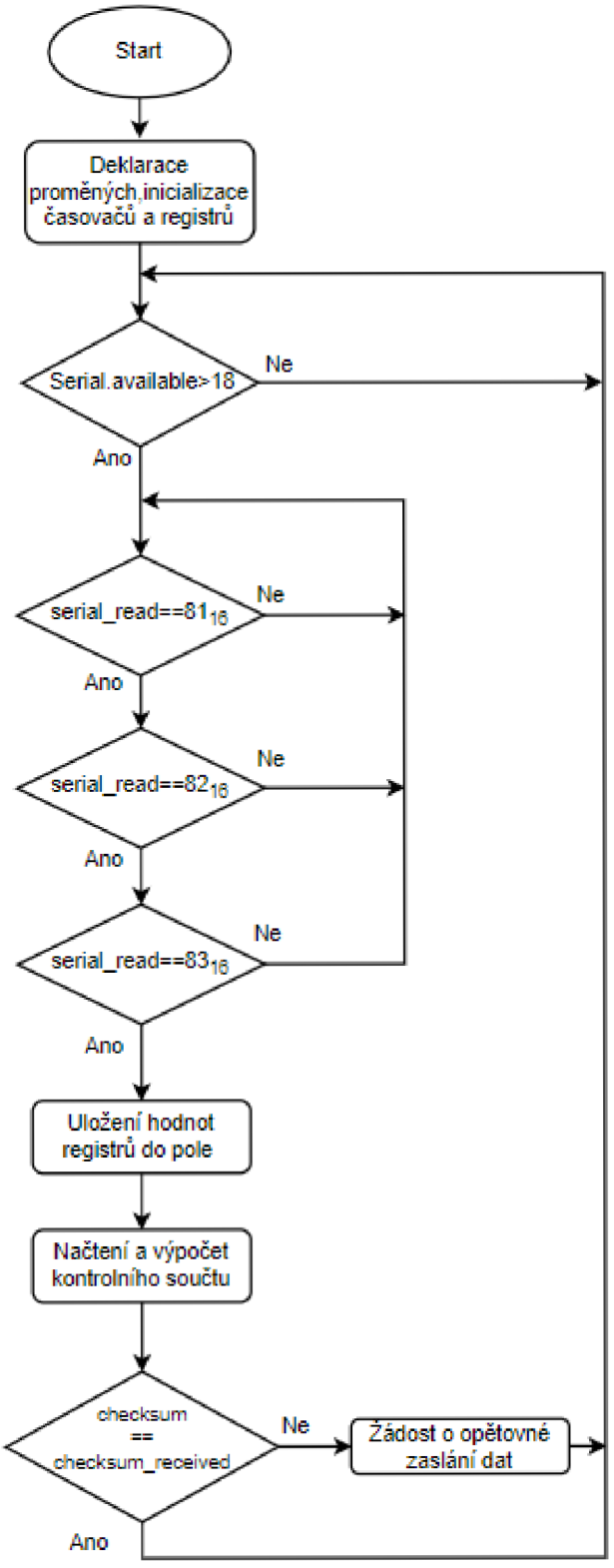
Sériová komunikace na straně vývojové desky Nexys 3 nebyla z časových důvodů realizována, proto je zde popsán pouze její návrh.

Navržená komunikace ve VHDL by obsahovala 3 procesy: jeden pro příjem hodnot, druhý pro odesílání potvrzovací zprávy a třetí pro posílání dat do registrů. Dále by zde byli tři děličky frekvence generující řídicí signál s frekvencí 50 Hz, 500 kH, a pro sériovou komunikaci s frekvencí odpovídající rychlosti přenosu dat 19 200 bps.

Procesy pro odeslání a příjem hodnoty by obsahovaly konečný stavový automat se stavy *Start_bit*, *Bity* a *Stop_bit*.

Proces přijímání hodnot by po přijetí start bitu přešel do stavu *Bity*, ve kterém by bylo postupně přijato 8 bitů. Po přijetí 8 bitů by stavový automat přešel do stavu *Stop_bit*, ve kterém by přijal stop bit. Nakonec by se stavový automat vrátil zpět do stavu *Start_bit*, kde by čekal na příchod další hodnoty. 8 přijatých bitů by tvořilo jednu hodnotu, která by byla uložena do příslušného prvku pole tvořeného 18 8bitovými vektory.

Proces pro posílání dat do registrů by obsahoval konečný stavový automat se stavy *Start*1, *Start*2, *Start*3, *Check* a *Data*. Po přijetí dat by se v procesu pro posílání dat do registrů zahájil chod stavového automatu. Jako první by se ve stavech *Start*1 až *Start*3 zkontrolovala hlavička posílaných dat a potom by se ve stavu *Check* zkontrolovaly kontrolní součty.



Obr. 4.3: Vývojový diagram komunikace na straně vývojové desky Arduino Uno

Pokud by se hlavička i kontrolní součet shodovaly přešel by automat do stavu *Data*. Jinak by přešel zpět do stavu *Start*1, ve kterém by počkal na přijetí dalších dat a proces odesílání potvrzovací zprávy by odeslal žádost o opětovné poslání hodnot.

Přechod do stavu *Data* by byl řízen řídicím signálem s frekvencí 50 Hz. Ve stavu *Data* by byli poslány hodnoty do registrů a přepínání řídicích signálů *bdir* a *bc*1 by bylo řízeno s frekvencí 500 kHz. Po ukončení zápisu hodnot do registrů by se pomocí procesu odesílání potvrzovací zprávy odeslala žádost o poslání následujících dat.

Proces odesílání potvrzovací zprávy by fungoval stejně jako proces příjmu hodnot pouze by se neukládaly přijaté hodnoty do pole, ale odesílaly by se hodnoty z konstantního pole hodnot.

5 Hudební data a návrh studentské laboratorní úlohy

Funkčnost realizovaných blokových uspořádání byla ověřena na třech skladbách s frekvencí přepisu registrů 50 Hz. Pro blokové uspořádání se zvukovým obvodem AY-3-8912A byla data posílána přes sériovou komunikaci z PC, což umožnilo přehrání celých skladeb.

Pro blokové uspořádání s vývojovou deskou Nexys 3 byly hodnoty 14 registrů uloženy jako 14 konstantních polí hodnot v knihovních balících, proto byla délka skladeb zkrácena na cca 30 s kvůli tomu,že jsou uloženy v obvodu FPGA. Využití sériové komunikace by umožnilo přehrání celých skladeb, ale ta nebyla z časových důvodů realizována.

Pro ověření funkčnosti navržené laboratorní úlohy je jednodušší číst data z paměti vývojové desky než pomocí dalšího USB kabelu připojovat vývojovou desku k PC a následně spustit další program.

5.1 Návrh studentské laboratorní úlohy

Z realizovaného PSG byla vytvořena laboratorní úloha, ve které budou mít studenti za úkol vytvořit chybějící bloky v připraveném projektu. Jako vhodné bloky pro výuku byly zvoleny bloky generátorů frekvence a šumu.

V příloze B je přiloženo zadání laboratorní úlohy, jehož součástí je i krátký teoretický úvod. Zdrojové soubory pro realizaci laboratorní úlohy jsou přiloženy v elektronické příloze a vedoucímu práce byla předána kompletní verze zdrojových souborů.

Závěr

Cílem práce bylo navrhnout a realizovat programovatelný zvukový generátor, jehož výslednou realizaci bude možné použít jako studentskou laboratorní úlohu v kurzu „Logické obvody a systémy“.

V první kapitole byla provedena rešerše existujících principů generování vícekanálového zvuku a dále byly popsány a porovnány tři architektury programovatelných zvukových generátorů. Porovnány byly architektury zvukových obvodů 6581 SID, POKEY C012294 a rodina zvukových obvodů AY-3-891X.

Jako vhodná architektura pro implementaci do hradlového pole byla zvolena architektura rodiny zvukových obvodů AY-3-891X, která generuje tříkanálový zvuk s možností generování proměnné amplitudy pomocí jednoho generátoru obálky.

Argumentem proti použití zvukového obvodu POKEY C012294 je jeho náchylnost na ztrátu hlasitosti a špatná kvalita zvuku. U zvukového obvodu 6581 SID je problém s dostupností odpovídající dokumentace.

Po výběru vhodné architektury bylo vytvořeno blokové schéma pro implementaci do hradlového pole, které je zobrazeno na obrázku 2.1. Realizovaný PSG je navržen tak, aby byl plně synchronní se vstupním hodinovým signálem a všechny bloky mají připojený asynchronní reset. Jednotlivé bloky byli realizovány a jejich funkčnost byla ověřena pomocí simulace. Realizace PSG byla časově náročná a zdrojové soubory mají přes 400 kB.

V kapitole 3 byla realizována dvě bloková uspořádání systému pro komunikaci s PC, která slouží k porovnání zvukového výstupu reálného zvukového obvodu AY-3-8912A a realizovaného PSG. Prvním blokovým uspořádáním systému bylo zapojení s vývojovou deskou Arduino Uno a zvukovým obvodem AY-3-8912A, kde pro napájení vývojové desky a komunikaci s PC byl použit jeden USB kabel (obrázek 3.4).

Druhým blokovým uspořádáním bylo zapojení s vývojovou deskou Nexys 3 rozšířenou modulem Pmod AMP2 (obrázek 3.8), do kterého byl nahrán realizovaný PSG. K vývojové desce byly připojeny dva USB kabely, přičemž jeden sloužil pro napájení a nahrání programu a druhý pro komunikaci s PC.

V kapitole 4 byla realizována komunikace mezi vývojovou deskou a PC pomocí asynchronní sériové komunikace, která byla vybrána, protože obě vývojové desky podporují UART komunikaci přes USB kabel.

Komunikace mezi vývojovou deskou a PC funguje tak, že program v PC pošle data pro jeden zápis do registrů PSG a následně čeká na potvrzovací zprávu od vývojové desky, zda má poslat nová data nebo poslat opět stejná. Pro kontrolu správnosti doručených dat byla odesílaná data rozšířena o hlavičku a kontrolní součet. Kontrolní součet je realizován exkluzivním logickým součtem odeslaných hodnot.

Sériová komunikace na straně vývojové desky Nexys 3 nebyla z časových důvodů realizována a byla nahrazena čtením dat z knihovních balíků, jejichž nevýhodou je omezení délky skladby z důvodu uložení v obvodu FPGA. Délka skladby je stále dostačující k demonstraci funkčnosti řešení a má délku cca 30 sekund.

Pro realizovaný PSG bylo vytvořeno zadání studentské laboratorní úlohy, které je přiloženo v příloze B. V této úloze budou mít studenti za úkol realizovat generátory frekvence a šumu. Vybrané generátory budou navrhovat v připraveném projektu, který je součástí elektronické přílohy a jehož kompletní verze se vzorovým řešením generátorů byla předána vedoucímu práce.

Funkčnost obou blokových uspořádání byla demonstrována na třech skladbách, jejichž nahrávky jsou součástí elektronické přílohy. Obě bloková uspořádání jsou funkční a zvukové výstupy jsou totožné.

V pokračování práce je možné rozšířit komunikaci vývojové desky s PC o odesílání většího množství dat najednou a obě bloková uspořádání systému by mohla být rozšířena o nastavení rychlosti přepisu registrů, což by umožnilo větší variabilitu při výběru hudebních dat.

Literatura

[1] PINKER, J. POUPA, M. *Číslicové systémy a jazyk VHDL.* 2006, ISBN 80-7300-198-5.

[2] VIRIUS, Miroslav. *Jazyky C a C++: kompletní průvodce.* 2., aktualiz. vyd. Praha: Grada, 2011. ISBN 9788024739175.

[3] COPELAND, Jack a Jason LONG. *Alan Turing: How his universal machine became a musical instrument.* IEEE Spectrum [online]. 26. 10. 2017 [cit. 2021-01-02]. Dostupné z: `https://spectrum.ieee.org/tech-history/silicon-revolution/alan-turing-how-his-universal-machine-became-a-musical-instrument?fbclid=IwAR0lNabGt0KBnKJaAkHKnRQFeymcYCCnq3_6y9q4solNGPYXP0rUk5RLo80`

[4] *Sound Design Basics: FM Synthesis* [online]. Cymatics.fm [cit. 2021-01-02]. Dostupné z:`https://cymatics.fm/blogs/production/fm-synthesis`

[5] YAMAHA. *YM3812: FM Operator Type-LII (OPLII).* 1992. Datasheet: LSI-2138123.

[6] POLICH, John, Andrew FISCHER a Arnold STARR. *Instrumentation & Techniques: A programmable multiple-tone generator.* Behavior Research Methods & Instrumentation. 1983(15(1)), 39-41.

[7] *Pulse Code Modulation*[online]. Tutorialspoint.com [cit. 2021-01-02]. Dostupné z:`https://www.tutorialspoint.com/digital_communication/digital_communication_pulse_code_modulation.htm`

[8] PETYOVSKÝ, Petr. *Předmět Logické obvody a systémy: Úloha č. 7 : Návrh n-bitového pulsně šířkového modulátoru, návrh sekvenčního obovodu, pulsně šířková modulace.* Rev. 4. Brno.

[9] A COMMODORE COMPANY. *Commodore mos technology NMOS: 6581 Sound interface device (SID).* Datasheet.

[10] *De Re Atari: A guide to effective programming 400/800 Home Computer*[online]. Atari Inc., 1982, 244 s. [cit. 2021-5-17]. Dostupné z: `https://archive.org/details/ataribooks-de-re-atari/page/n1/mode/1up`

[11] GENERAL INSTRUMENT. *AY-3-8910, AY-3-8912, AY-3-8913: Programmable Sound Generator.* Datasheet.

[12] MICROCHIP TECHNOLOGY. *AY8930: Enhanced Programmable Sound Generator.* 1990. Datasheet.

[13] CASS, Stephen. *Chip Hall of Fame: MOS Technology 6581.* IEEE Spectrum[online]. 15. 7. 2019 [cit. 2021-5-10]. Dostupné z: `https://spectrum.ieee.org/tech-history/silicon-revolution/chip-hall-of-fame-mos-technology-6581`

[14] *Arduino Uno Rev3.* Arduino[online]. [cit. 2021-5-10]. Dostupné z: `https://store.arduino.cc/arduino-uno-rev3`

[15] *Port Registers.* Arduino[online]. [cit. 2021-5-10]. Dostupné z: `https://www.arduino.cc/en/Reference/PortManipulation`

[16] *Arduino Uno Tutorial [Pinout].* DIYI0T [online]. [cit. 2021-5-10]. Dostupné z: `https://diyi0t.com/arduino-uno-tutorial/`

[17] *Nexys 3 Reference Manual.* Digilent [online]. [cit. 2021-5-10]. Dostupné z: `https://reference.digilentinc.com/reference/programmable-logic/nexys-3/reference-manual`

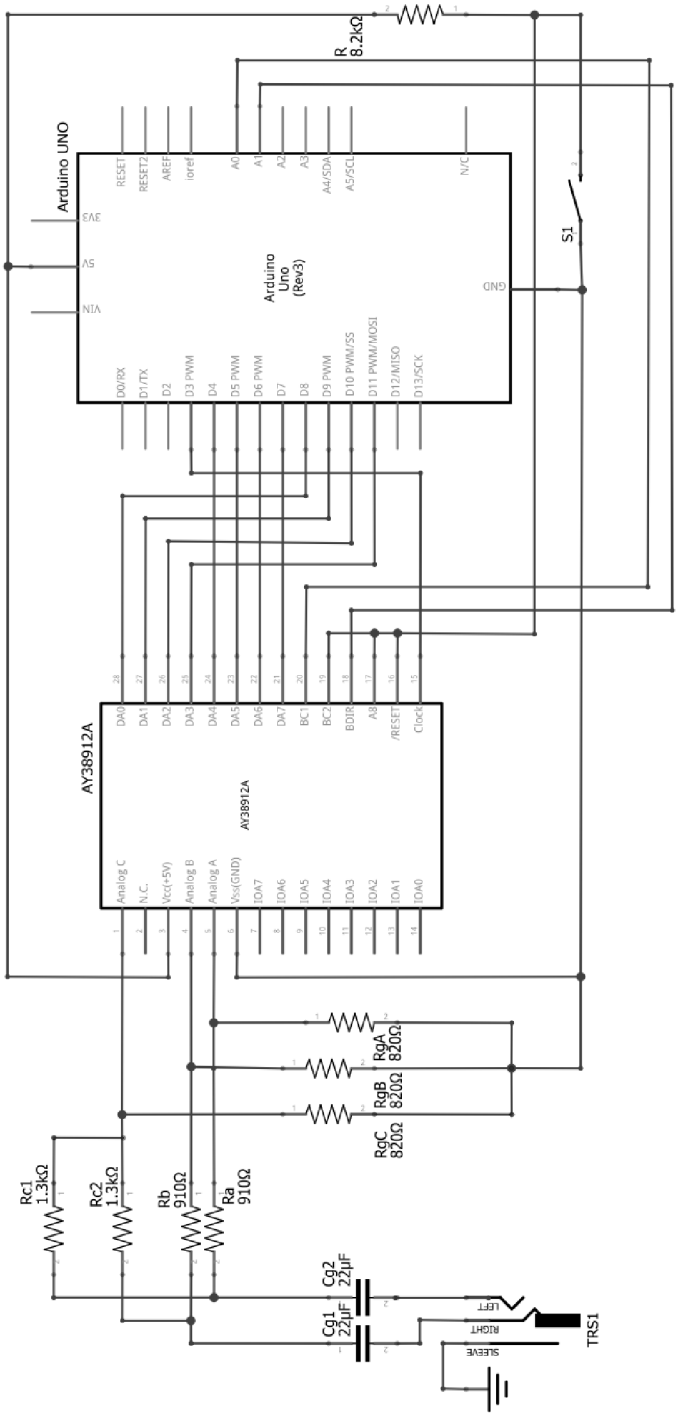
[18] *Pmod AMP2 Reference Manual.* Digilent [online]. [cit. 2021-5-10]. Dostupné z: `https://reference.digilentinc.com/pmod/pmodamp2/reference-manual`

Seznam symbolů a zkratek

CPU	Centrální procesorová jednotka
EEPROM	Elektricky vymazatelná paměť pouze pro čtení
FM	Frekvenční modulace
FPGA	Programovatelné hradlové pole
GPIO	Univerzální vstupně/výstupní
LED	Luminiscenční dioda
OPL	Rodina zvukových obvodů firmy Yamaha
PC	Osobní počítač
Pmod	Periferní modul
POKEY	Pot Keyboard Integrated Circuit
PSG	Programovatelný zvukový generátor
PWM	Pulzně šířková modulace
SID	Sound Interface Device
SRAM	Statická paměť pro čtení i zápis
UART	Univerzální asynchronní přijímač/vysílač
USB	Univerzální sériová sběrnice
VGA	Počítačový standard pro zobrazovací techniku (Video Graphics Array)

Seznam příloh

A Schéma zapojení vývojové desky Arduino Uno a zvukového obvodu AY-3-8912A



Obr. A.1: Schéma zapojení Arduino Uno a AY-3-8912A

B Zadání studentské laboratorní úlohy

Cíle

- Návrh generátoru frekvence pomocí synchronního binárního čítače.
- Návrh generátoru šumu pomocí lineárního zpětnovazebního čítače.
- Zprovoznění programovatelného zvukového generátoru.

Teoretický úvod

Čítače

Čítače jsou zařízení počítající kolikrát proběhl daný jev. Z hlediska digitální techniky je čítač obvykle chápán jako sekvenční logický obvod, který zaznamenává počet změn vstupního hodinového signálu. S každou změnou hodinového signálu dochází ke změně hodnoty výstupu. Dle distribuce hodinového signálu uvnitř čítače se dělí na synchronní a asynchronní. U asynchronního čítače je vstupní hodinový signál přiveden pouze na první klopný obvod a u synchronního na všechny klopné obvody.

V technické praxi se často používají binární čítače čítající v intervalu $<0;(2^n - 1)>$, kde n je šířka výstupního slova čítače. Čítače jsou často rozšířeny o asynchronní nebo synchronní reset a o možnost přednastavení počáteční hodnoty.

Pomocí čítačů lze realizovat děličku vstupního hodinového signálu.

Lineární zpětnovazební čítače

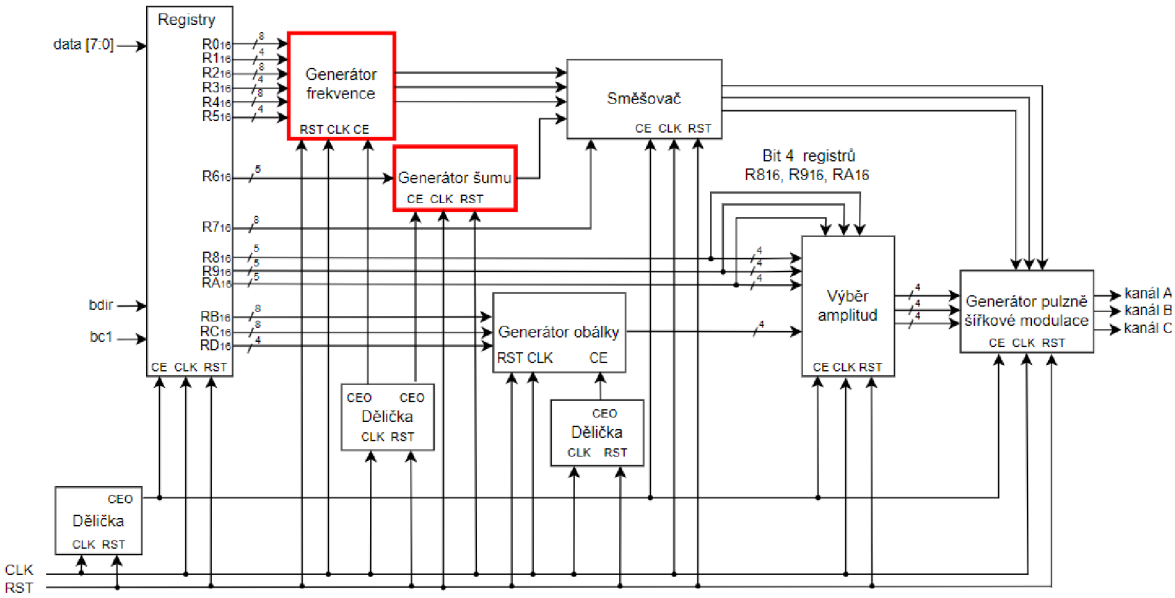
Lineární zpětnovazební čítač je druh zpětnovazebního čítače, jehož zpětnou vazbu tvoří funkce XOR, které jsou generátorem liché parity a posloupnost maximální délky je $2^n - 1$. Jediná kombinace, která nikdy nenastane, jsou samé nuly na výstupu, proto je nutné nastavit počáteční stav na libovolnou nenulovou hodnotu. Lineární zpětnovazební čítače se v praxi používají v kryptografii nebo jako generátory šumu v programovatelných zvukových generátorech.

Programovatelný zvukový generátor

Programovatelný zvukový generátor (PSG) je softwarově řízený zvukový obvod generující tóny. Hlavním prvkem PSG je pole vstupně/výstupních registrů, ve kterém se nastavují vstupní parametry pro jednotlivé kanály.

Architektury PSG se od sebe liší množstvím a druhem funkčních bloků generujících zvuk, ale většina obsahuje tyto funkční bloky: generátor frekvence, generátor šumu, generátor obálky a nastavení hlasitosti. Některé architektury mohou obsahovat i různé filtry nebo mít nastavitelný počet výstupních kanálů.

Na obrázku je zobrazeno blokové schéma tříkanálového PSG realizovaného v jazyce VHDL. Červeně zvýrazněné funkční bloky jsou úlohou této laboratorní úlohy.



Generátor frekvence je vytvořen jako dělička vstupního hodinového signálu a generátor šumu jako lineárně zpětnovazební čítač. Generátor obálky generuje proměnou amplitudu bez potřeby opakovaného přepisování registrů hlasitosti a výstupní zvuk PSG je tvořen pomocí pulzně šířkové modulace.

Vypracování laboratorní úlohy

Úkol č. 1

V návrhovém prostředí Xilinx ISE WebPack navrhněte generátor frekvence, který bude realizován synchronním binárním čítačem. Generátor frekvence realizujte v entitě *frequency_generator_vhdl* projektu *PSG_vhdl*.

12bitový vektor *divider* bude tvořen vstupním vektorem *coarse_in* (bity 11 až 8) a *fine_in* (bity 7 až 0). Čítání čítače bude povolováno vstupem *CE* a čítač bude pracovat v intervalu $<0;(divider/2) - 1>$. Dále bude komponenta obsahovat asynchronní reset *rst* a po dočítání čítače se bude měnit výstup *frequency_out*.

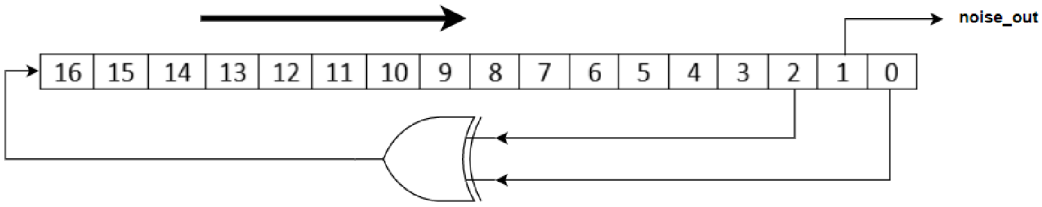
```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity frequency_generator_vhdl is
        port
           (
           clk: in  STD_LOGIC;                          --- vstupni hodinovy signal
           rst: in STD_LOGIC;                           --- asynchronni reset
           CE: in STD_LOGIC;                            --- vstup povolujici citani
           coarse_in: in STD_LOGIC_VECTOR(3 downto 0);  --- vstupni vektor (bity 11 az 8 divider)
           fine_in: in STD_LOGIC_VECTOR(7 downto 0);    --- vstupni vektor (bity 7 az 0 divider)
           frequency_out: out  STD_LOGIC                --- vystup
           );
end frequency_generator_vhdl;

architecture Behavioral of frequency_generator_vhdl is

begin

end Behavioral;
```

Úkol č. 2

V návrhovém prostředí Xilinx ISE WebPack navrhněte generátor šumu, který bude tvořen synchronním binárním čítačem a 17bitovým lineárním zpětnovazebním čítačem fibonacciho typu. Lineární zpětnovazební čítač realizujte v entitě *noise_generator_vhdl* entity *PSG_vhdl*.

Komponenta bude obsahovat asynchronní reset *rst*. Čítání synchronního čítače bude řízeno vstupem pro povolení čítání *CE* a pracovat bude v rozsahu <0; *R6_in* − 1>.

Výstup synchronního čítače bude povolovat čítání lineárního zpětnovazebního čítače, který bude realizovaný podle následujícího obrázku. Počátečními vnitřními hodnotami lineárního zpětnovazebního čítače budou hodnoty: X"1FFFF". Výstupem generátoru šumu je bit 1 lineárního zpětnovazebního čítače.




```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity noise_generator_vhdl is
        port
            (
               clk: in STD_LOGIC;                          --- vstupni hodinovy signal
               rst: in STD_LOGIC;                          --- asynchronni reset
               CE: in STD_LOGIC;                           --- vstup povolujici citani
               R6_in: in STD_LOGIC_VECTOR(4 downto 0);     --- vstupni vektor
               noise_out: out STD_LOGIC                    --- vystup
             );
end noise_generator_vhdl;

architecture Behavioral of noise_generator_vhdl is

begin

end Behavioral;
```

C Obsah elektronické přílohy

- `/` ..... kořenový adresář elektronické přílohy
  - `Arduino` ..... adresář se zdrojovým souborem pro vývojovou desku Arduino Uno
    - `Arduino.ino`
  - `PSG_vhdl` ..... adresář se zdrojovými soubory pro realizované PSG
    - `amplitude_convertion_vhdl.vhd`
    - `Amplitude_vhdl.vhd`
    - `clk_enable_vhdl.vhd`
    - `comparison_vhdl.vhd`
    - `counter_vhdl.vhd`
    - `data_send_vhdl.vhd`
    - `envelope_en_vhdl.vhd`
    - `envelope_generator_vhdl.vhd`
    - `frequency_generator_vhdl.vhd`
    - `mixer_vhdl.vhd`
    - `noise_generator_vhdl.vhd`
    - `output_mix_vhdl.vhd`
    - `pattern_vhdl.vhd`
    - `PSG_vhdl.vhd`
    - `pwm_vhdl.vhd`
    - `registers_vhdl.vhd`
    - `registr_t.vhd`
    - `PSG_testbench.vhd`
    - `BEVRLYCP.rb0.vhd - rbd.vhd` ..... zdrojové soubory registrů pro 1. skladbu
    - `GHOSTBST.rg0.vhd - rgd.vhd` ..... zdrojové soubory registrů pro 2. skladbu
    - `STARWARS.rs0.vhd - rsd.vhd` ..... zdrojové soubory registrů pro 3. skladbu
  - `Serial` ..... adresář se zdrojovými soubory pro komunikaci na straně PC
    - `main.cpp`
    - `Serial.cpp`
    - `Serial.h`
    - `Hudebni_data` ..... adresář se zdrojovými soubory registrů
      - `BEVRLYCP.rb0.c - rbd.c` ..... zdrojové soubory registrů pro 1. skladbu
      - `GHOSTBST.rg0.c - rgd.c` ..... zdrojové soubory registrů pro 2. skladbu
      - `STARWARS.rs0.c - rsd.c` ..... zdrojové soubory registrů pro 3. skladbu
  - `videa` ..... adresář s demostračními nahrávkami
    - `BEVRLYCP_AY.mp4` ..... nahrávka 1. skladby zvukového obvodu AY-3-8912A
    - `GHOSTBST_AY.mp4` ..... nahrávka 2. skladby zvukového obvodu AY-3-8912A
    - `STARWARS_AY.mp4` ..... nahrávka 3. skladby zvukového obvodu AY-3-8912A
    - `BEVRLYCP_VHDL.mp4` ..... nahrávka 1. skladby realizovaného PSG
    - `GHOSTBST_VHDL.mp4` ..... nahrávka 2. skladby realizovaného PSG
    - `STARWARS_VHDL.mp4` ..... nahrávka 3. skladby realizovaného PSG
  - `thesis.pdf` ..... elektronická verze bakalářské práce