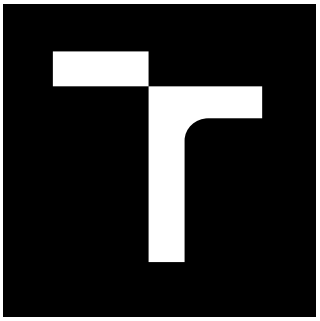


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

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 ŘEČOVÉHO GENERÁTORU PRO EMBEDDED SYSTÉMY

REALIZATION OF A SPEECH GENERATOR LIBRARY FOR EMBEDDED SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Michal Kvasňák**
AUTHOR

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

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Michal Kvasňák ***ID:*** 221305

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Realizace řečového generátoru pro embedded systémy

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout a realizovat software ve formě knihovny zajišťující generování řečového / hlasového výstupu pro zvolený embedded systém.

1. Proveďte rešerši existujících principů generování hlasového výstupu v počítačových systémech.
2. Zdokumentujte existující architektury generování hlasového výstupu v embedded systémech, případně navrhněte vlastní řešení.
3. Zvolte vhodný embedded systém pro realizaci knihovny hlasového výstupu. Definujte důvody pro jeho volbu i jeho případná omezení.
4. Navrhněte blokové schéma řečového generátoru a implementujte jeho základní komponenty.
5. Proveďte ověření funkčnosti jednotlivých bloků.
6. Dokončete programovou implementaci programu pro hlasový výstup v jazyce C a ověřte její funkčnost.
7. Upravte program do formy embedded SW knihovny. Navrhněte demonstrační aplikaci pro prezentaci funkčnosti řešení. Popište blokové uspořádání všech komponent aplikace tak, aby bylo možné realizovaný hlasový generátor vhodně ovládat pomocí PC.
8. Zpracujte podklady pro studentskou laboratorní úlohu využívající hlasový generátor.
9. Zhodnoťte dosažené výsledky, uveďte výhody a nevýhody řešení a navrhněte další možná rozšíření.

DOPORUČENÁ LITERATURA:

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

[2] Yamaha YM2149 software-controlled sound generator, datasheet.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 22.5.2023

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

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

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt

Práce se zaobírá realizací knihovny umožňující generování hlasového výstupu pomocí vývojové desky Arduino Mega2560 rev3, ke které je připojen piezoelektrický měnič. Knihovna pracuje na logice získané pomocí analýzy programu „HLAS“ navrženého pro osobní počítač Sinclair ZX Spectrum. Výstupem práce je funkční knihovna, pro kterou je zároveň navržena výuková laboratorní úloha sloužící k seznámení se s rozhraním sériové komunikace USART.

Klíčová slova

Hlasový generátor, řečový generátor, syntéza řeči, syntetizér řeči, formantová syntéza, vestavěný systém, fonetická transkripce, sériová komunikace, USART

Abstract

The bachelor thesis is about implementation of library which allows to generate voice output using an embedded system Arduino Mega2560 rev3 to which buzzer is connected. The library works on logic obtained by analyzing the „HLAS“ program designed for personal computer Sinclair ZX Spectrum. The output of the work is a working library which is used for lab task designed to learn about USART serial communication interface.

Keywords

Voice generator, speech generator, speech synthesis, speech synthesizer, formant synthesis, embedded system, phonetic transcription, seriál communication, USART

Bibliografická citace

KVASŇÁK, Michal. *Realizace řečového generátoru pro embedded systémy* [online]. Brno, 2023 [cit. 2023-05-21]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/151802. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Petr Petyovský.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	Michal Kvasňák
VUT ID studenta:	221305
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Realizace řečového generátoru pro embedded systémy

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

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: 22.5.2023 ----------

podpis autora

Poděkování

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

V Brně dne: 22.5.2023 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

Bakalářská práce se zabývá realizací generátoru hlasového výstupu za pomoci jednoduchého vestavného systému bez využití externích zvukových obvodů. Cílem práce je realizovat syntetizér, který převádí textový vstup na hlasový výstup pomocí vývojové desky Arduino Mega2560 rev3 s připojeným piezoelektrickým měničem.

Nejprve bude provedeno seznámení s obecným procesem vytváření řeči člověkem, čímž bude získána představa o jednotlivých funkcích, které budou řečovým generátorem nahrazovány.

Následuje představení způsobu přenosu parametrů řeči pomocí fonetických abeced, ze kterých bude zvolena nejvhodnější možnost pro rozebíranou problematiku. Vstupní text bude do zvolené abecedy převáděn pomocí fonetické transkripce, která je řízena pravidly. Ze všech těchto pravidel budou vyselektovány jen ty, které skutečně ovlivňují kvalitu generovaného hlasového výstupu.

Popsáním jednotlivých principů hlasové syntézy jsou získány znalosti pro navazující analýzu programu „HLAS“ představující praktickou realizaci hlasového generátoru. Ten byl publikován roku 1985 společností VoiceSoft pro osobní počítač Sinclair ZX Spectrum.

Pomocí dříve zmíněné analýzy bude vytvořen obecný návrh knihovny pro generování hlasového výstupu s využitím vývojové desky Arduino Mega2560 rev3, na jehož základě bude provedena samotná realizace knihovny pomocí programovacího jazyka C.

Po realizaci je ověřena správnost generovaného hlasového výstupu. Pomocí funkčního řešení bude navržena výuková laboratorní úloha. Její účelem bude seznámení se s rozhraním sériové komunikace USART, které je v rámci knihovny využíváno.

Závěr práce je věnován shrnutí dosažených výsledků a možných rozšíření, které by mohly být v budoucnu implementovány.

Bakalářská práce je členěna do deseti kapitol. První tři kapitoly slouží jako teoretický úvod do problematiky digitální syntézy hlasu. Následně je v kapitole 4 získán princip generování hlasového výstupu. Pomocí něj je v navazujících dvou kapitolách zvolen vhodný vestavný systém, pro který je proveden obecný návrh aplikace generující hlasový výstup. V kapitole 7 je navržená knihovna realizována pomocí jazyka C. Její funkce je ověřena v kapitole 8. Za využití knihovny je v kapitole 9 vytvořena laboratorní úloha pro seznámení s rozhraním sériové komunikace USART. Závěrem je kapitola 10, která shrnuje dosažené výsledky a zároveň jsou v ní diskutovány možné rozšíření pro budoucí implementaci.

1. ČESKÝ JAZYK A JEHO VLASTNOSTI

Kapitola se věnuje obecnému popisu řeči člověka a způsobům jejich zpracování zejména textovou formou z různých úhlů pohledu. Následně jsou zde shrnuty prostředky pro takové zpracování, ze kterých budeme vybírat pro následně realizovanou aplikaci.

1.1 Řeč a její vlastnosti

Řeč je v dnešní době vnímána jako základní dorozumívací prostředek mezi lidmi s čímž přichází i snaha o porozumění jejího vzniku a možností jejího uchování a dalšího zpracování. Zmíněny jsou zde pouze reprezentace řeči za pomoci textu, kdy zaznamenáváme vlastnosti toho, jak byla přednesena a nikoli toho, jaké informace obsahuje.

1.1.1 Řeč vytvářená člověkem

Zdrojem energie pro tvorbu řeči je dechové ústrojí. Pomocí výdechu v hlasovém ústrojí začínají kmitat hlasivky. Důsledkem kmitaní hlasivek produkujeme základní tón o proměnlivé frekvenci základního hlasivkového tónu F_0, který představuje nosič řeči. Tato frekvence se mění v závislosti na věku, pohlaví, ale nikdy není konstantní, jelikož nejsme schopni vyprodukovat ryze periodický průběh tohoto tónu. [1]

Posledním prvkem ve tvorbě řeči je artikulační ústrojí, z něhož je nejpodstatnějším prvkem obecně ústní dutina, jmenovitě jazyk. Díky pohybům jednotlivých částí artikulačního ústrojí, které souhrnně nazýváme artikulací, se proudící základní hlasivkový tón formuje do výsledné podoby, kterou nazýváme řeč a obsahuje tónovou a šumovou složku. [1]

1.1.2 Zpracování mluvené řeči

Od pravěku jsme za základní zpracování řeči považovali sluch, přestože jsme nebyli schopni takto získanou řeč dlouhodobě uchovávat. Následně jsme se dopracovali k ukládání řeči pomocí písma, jenž bylo možné uchovat a následně znovu zpracovávat či upravovat. Písmo však samo o sobě uchovává pouze obsah řeči, nikoliv její vlastnosti a jakým způsobem byla vyslovena. Na základě této skutečnosti se vyvinul obor zvaný lingvistika. [1]

Díky tomuto oboru jsme se na řeč začali dívat i z jiných úrovní. Základní úrovní je akustická, která pracuje s řečí jako signálem s proměnlivou amplitudou a frekvencí závislými na čase. Pomocí toho jsme byli schopni zjistit základní parametry a vlastnosti řeči. Problémem však je náročné zpracování takového signálu a jeho rozbor na jednotlivé složky. [1]

Z hlediska zápisu a zároveň i reprodukce řeči je pro nás relevantnější úroveň fonetická a fonologická. Úroveň fonetická neboli fonetika studuje celkové vytváření řeči, tedy od výdechu až po výstupní signál po projití hlasovým a artikulačním ústrojím. Následně je

rozebírá na jednotlivé složky, jejichž základem je hláska. Z fonetického hlediska je řeč tvořená posloupností hlásek. Soubor všech hlásek tvoří fonetický inventář, díky kterému můžeme řeč zapsat formou její výslovností. Fonologie pak nezkoumá řeč jako takovou, ale zabývá se přímo konkrétním jazykem pomocí jednotky foném. [1]

Foném je považován za nejmenší možnou jednotku, kterou můžeme zapsat jednotlivé prvky řeči. Nereprezentuje ale výslovnost, spíše postavení a pozice jednotlivých prvků pro vytváření řeči. Jelikož má každý jazyk specifické požadavky, tak pro každý používáme ojedinělý fonémový inventář. [1]

1.2 Fonetické abecedy

Fonémový inventář, je vhodným prostředkem pro zpracování řeči, jestliže chceme uchovat její vlastnosti, nikoliv informace v ní obsažené. Aktuálně je vytvořeno několik takových abeced, přičemž každá z nich má ojedinělé vlastnosti a je vhodná pro určitou oblast. Z toho důvodu jsou zde obsaženy nejznámější a zároveň případné varianty, které můžeme využít v nadcházející realizaci aplikace.

1.2.1 Fonetická abeceda IPA

Fonetická abeceda IPA (International Phonetic Alphabet) je mezinárodní standard pro přepis všech jazyků. Její výhodou je právě definování fonetických značek všem jazykům, takže můžeme porovnávat více rozdílných jazyků na základě jejich fonémů. [1]

Jelikož je tato abeceda velmi obsáhlá, tak se v aplikacích zkoumající nebo využívající pouze jeden jazyk používá buďto zredukovaná verze abecedy IPA, popřípadě jiné fonetické abecedy, které jsou navrženy právě pro konkrétní styly jazyků (např. slovanské, germánské apod.) nebo abecedy pro konkrétní jazyk. [1]

Dalším problémem, který by byl pro naši aplikaci zásadnější, je složitost reprezentace jednotlivých znaků v počítači. Tento problém nastává v souvislosti s rozsahem této abecedy, jelikož potřebujeme ojedinělé znaky pro specifické charakteristické vlastnosti konkrétních jazyků. Tento problém se dá vyřešit mnoha způsoby. Nejběžnějším je rozšíření znakové sady o jednotlivé fonetické značky, čímž umožňujeme přímo zobrazit fonetický přepis. Jestliže však nutně nepotřebujeme zobrazit dané znaky, pak je výhodnější využít fonetickou abecedu SAMPA (viz kap. 1.2.2).

1.2.2 Fonetická abeceda SAMPA a X-SAMPA

Abeceda SAMPA (Speech Assessment Methods Phonetic Alphabet) reprezentuje přepis fonetické abecedy IPA do sedmibitových ASCII (American Standard Code for Information Interchange) znaků. [1]

Z toho je patrné, že je její použití na počítačích snazší a výhodnější, jelikož neztrácíme obecnost zápisu, ale ztrácíme nutnost rozšíření znakové sady o speciální fonetické znaky, které abeceda IPA obsahuje.

Nevýhodou této abecedy je skutečnost, že již nejsme schopni obsáhnout tolik jazyků do jedné abecedy. Konkrétně je vždy jednou sadou abecedy SAMPA reprezentován právě jeden světový jazyk. Z toho důvodu vznikla rozšířená verze této abecedy X-SAMPA.

Při tvorbě inventáře X-SAMPA byl kladen důraz o reprezentaci všech znaků abecedy IPA pomocí znaků ASCII. Výsledkem je tedy abeceda, která se dá kompletně reprezentovat sedmibitovými čísly a můžeme tedy snadno přecházet mezi rozdílnými jazyky. [1]

1.2.3 Česká fonetická abeceda ČFA

Jedná se o abecedu vytvořenou právě pro češtinu. Byla vytvořena převážně z toho důvodu, že abeceda IPA je v počítačích složitá pro využití a abeceda SAMPA v době vzniku ČFA (Česká Fonetická Abeceda) stále nebyla vytvořena. [1]

Momentálně se v dokumentech zabývající se českou fonetickou transkripcí (viz kap. 3) přednostně objevuje přepis pomocí této abecedy nebo pomocí zjednodušené české fonetické abecedy.

1.2.4 Shrnutí

Pro naši aplikaci přichází v úvahu pouze varianta využití české verze fonetické abecedy SAMPA nebo použití české fonetické abecedy ČFA. Jejich porovnání se nachází v tabulce níže (viz tab. 2.1).

Jelikož je realizovaná aplikace navržena výhradně pro češtinu a zároveň bude využívána pro výukovou úlohu, nabízí se fonetická abeceda ČFA. Ta však pro reprezentaci některých písmen využívá řetězec dvou znaků, což není vhodné pro zpracování pomocí počítače. Jelikož však neobsahuje velká písmena, pak je možno její zápis upravit (v tabulce 2.1 označen jako mČFA) pro reprezentaci daných fonémů pomocí pouze jednoho znaku. Takovýto způsob by byl optimálním řešením a proto bude využit.

Tabulka 2.1 Fonetická abeceda SAMPA a česká fonetická abeceda ČFA [1]

	SAMPA	ČFA	mČFA	PŘÍKLAD		SAMPA	ČFA	mČFA	PŘÍKLAD
vokály	i	i	I	**li**s	plozivy	p	p	P	**p**ec
	e	e	E	p**e**s		b	b	B	**b**ratr
	a	a	A	s**a**d		t	t	T	**t**uk
	o	o	O	k**o**v		d	d	D	**d**ům
	u	u	U	s**u**kně		c	tj	t	dě**t**i
	i:	ii	i	v**í**no		J\	dj	d	**d**ítě
	e:	ee	e	l**é**k		k	k	K	**k**ost
	a:	aa	a	s**á**l		g	g	G	ty**g**r
	o:	oo	o	k**ó**d	nazály	m	m	M	**m**už
	u:	uu	u	r**ů**že		n	n	N	ví**n**o
diftongy	o_u	ow	OU	b**ou**da		J	nj	n	la**ň**ka
	a_u	aw	AU	**au**to	afrikáty	t_s	c	C	**c**ena
	e_u	ew	EU	**eu**nuch		t_S	ch	c	o**č**i
frikativy	f	f	F	**f**ík		d_z	dz	D	po**d**zim
	v	v	V	**v**ítr		d_Z	dzh	Dz	**dž**bán
	s	s	S	**s**ůl	významné alofony	N	ng		ta**n**go
	z	z	Z	ko**z**a		F	mg		tra**m**vaj
	S	sh	S	**š**kola		G			aby**ch** byl
	Z	zh	z	**ž**ena		Q\	rsh		t**ř**i
	x	x	h	**ch**ata		r=			k**r**k
	h\	h	H	**h**ůl		l=			v**l**k
	l	l	L	v**l**ak		m=			os**m**
	r	r	R	**r**ok		?			„ráz“
	P\	rzh	r	mo**ř**e		@			„šva“
	j	j	J	**j**ev					

2. FONETICKÁ TRANSKRIPCE

Fonetickou transkripcí rozumíme proces, při kterém přepisujeme mluvenou nebo psanou řeč pomocí fonetických abeced za dodržování určených pravidel. Tímto procesem získáme řeč zapsanou na úrovni fonémů, tedy tak, jak ji vyslovujeme. Dle komplexnosti zvolené abecedy získáváme popis s odpovídající přesnosti. [1]

Jelikož by byla manuální fonetická transkripce zdlouhavá a neefektivní, a zároveň by bylo složité docílit rychlé fonetické transkripce z mluvené řeči, začal se tento proces automatizovat. Tento způsob je pak základem všech syntetizátorů. Pro automatickou fonetickou transkripci jsou pro každý jazyk definovaná pravidla. [1]

Pro češtinu je rovněž definován i slovník výjimek tzv. fonetický slovník výjimek. V tomto slovníku jsou zapsána slova, které se v běžné řeči vyskytují, avšak na ně nemůžeme aplikovat klasická pravidla pro fonetickou transkripci. Především jsou v něm uložena slova převzatá z jiných jazyků, jako jsou například názvy měst. [1]

2.1 Pravidla fonetické transkripce

Jak bylo zmíněno, tak pro fonetickou transkripci platí určitá pravidla. Využití a obsáhlost jednotlivých pravidel je ovlivněná složitostí a přesností požadovaného výstupu. Zde si uvedeme tedy pouze podstatná pravidla pro realizovanou aplikaci. Zároveň se mnohdy jedná i o pravidla, které se zaobírají nespisovně mluvenou češtinou, což není předmětem naší aplikace. [1]

Konkrétní pravidla obsahují příklady, jak se dané znaky či skupiny znaků přepisují pomocí abecedy ČFA. V těchto příkladech je uveden zápis do zvolené zjednodušené české fonetické abecedy, aby co nejlépe vystihovala řešenou problematiku. Rovněž je nutno podotknout, že se aplikace zabývá realizací poměrně základního syntetizátoru, proto jsou zde uvedeny pravidla, které reálně ovlivní náš hlasový výstup.

2.1.1 Základní algoritmus

Fonetická transkripce, která se řídí určitými pravidly, probíhá opakovaně v definovaném algoritmu, který lze pro následnou realizaci aplikace zobecnit. Obecný algoritmus může být prováděn v těchto krocích [1]:

1) Vstupní řetězec zpracováváme sekvenčně zleva doprava (nebo zprava doleva).
2) Zjistíme, zda můžeme pro písmeno použít vhodné produkční pravidlo
3) Pokud nemůžeme žádné z pravidel použít, pak písmeno pouze přepíšeme.

Takto zapsaným algoritmem jsme schopni přepsat téměř jakýkoliv česky napsaný řetězec, jenž následně můžeme pomocí syntézy vyslovit. Jediným problémem jsou slova převzatá z jiných jazyků, pro které neplatí česká fonetická transkripce. Tyto výjimky si blíže popíšeme dále (viz kap. 2.1.6).

2.1.2 Základní pravidla

Jedná se o pravidla o určitých znacích, které můžeme automaticky bez jakéhokoliv kontextu přepsat do fonetické podoby. Jedná se zejména o písmena, které tímto přepisem neztrácí své vlastnosti a tím pádem jejich přepisem zjednodušíme zápis a zároveň nezměníme konečný zvuk slova, ve kterém se nachází. Jedná se o tyto pravidla [1]:

ch → x
ů → uu
y → i
ý → ii

2.1.3 Pravidlo spojování samohlásek

Toto pravidlo se zabývá změkčovači hlásek, tedy znaků [i, í] a [ě]. Každé z těchto změkčovačů ovlivňuje předcházející písmeno jinak, proto existují specifická pravidla pro každé z nich. Pravidla pro změkčovače [i, í] můžeme zapsat takto [1]:

di → ď [d‘]
ti → ť [t‘]
ni → ň [n‘]

Vidíme tedy, že znak [i] ovlivňuje pouze hlásky [d], [t] a [n]. Změkčovač zde mění výslovnost daného písmena. V naši aplikaci by byla reprezentace českých znaků s háčky komplikovaná, proto háček nahradíme následovným znakem [‘].

Oproti předcházejícímu, ovlivňuje znak [ě] taktéž výslovnost, avšak způsobem, že při přepisu můžeme využít zjednodušení pomocí přidání znaku [j] mezi změkčovač a předcházející znak. Pravidla pro [ě] vypadají takto [1]:

bě → bje
pě → pje
vě → vje
mě → mňe

Oproti základním pravidlům (viz kap. 2.1.2) je zřejmé, že toto pravidlo nám již mnohem lépe dokáže znázornit, jak je daný řetězec vyslovován. Zároveň již dochází ke značnému zjednodušení daného algoritmu, jelikož je jednoznačné, že pro každou možnou výslovnost nemusíme nastavovat specifickou hodnotu.

2.1.4 Asimilace znělosti

V českém jazyce se vyskytuje velké množství slov, které se zapisují daným způsobem, ale jejich výslovnost je značně odlišná. Tuto odlišnou výslovnost způsobuje souhrn po sobě jdoucích hlásek, které v důsledku postupné výslovnosti vytvoří asimilaci, která zapříčiní vyrovnání znělostí, tedy jinému vyslovení. [1]

Problémem těchto pravidel je, že se poměrně složitě popisují, jelikož při výskytu dvou shodných souhrnů hlásek v rozdílných slovech může docházet jak ke shodné, tak i odlišné výslovnosti. [1]

Obecně lze prohlásit, že v drtivé většině případů jsou oba tyto přepisy správné, avšak existují výjimky, u kterých je spisovný pouze jeden způsob přepisu (např. *shora [zhora], shůry [zhůry]*). Zároveň však existují i výjimky, kde by přepis dle daných pravidel změnil celkový význam daného slova (např. *sházet [scházet], shody [schody]*). [1]

2.1.5 Asimilace artikulační

Obdobně jako u asimilace znalostí se zde postihuje celá skupina sousedních souhlásek, avšak se zde vyrovnává znělost artikulační. [1]

Podstatným rozdílem oproti ostatním pravidlům je, že k vyrovnání artikulací vůbec nemusí docházet, tedy je čistě jen na nás, zda toto pravidlo uplatníme. Využití těchto pravidel by dávalo smysl v případě, kdy by náš syntetizátor mluvil rychle.

2.1.6 Slova přejatá

Slova přejatá tvoří v českém jazyce značnou část slovní zásoby. Mezi takovéto slova patří slova počeštělá, u kterých se již změnila jak výslovnost, tak i jejich zápis, tedy pro naše vypracování nepřináší žádné komplikace (např. *škola, kostel atd.*). Problém vzniká u slov nepočeštělých, kde se výslovnost stále zachovává, tím pádem pro ně neplatí česká fonetická transkripce. [1]

Abychom slova nepočeštělá mohli vyslovovat, je nutné vytvořit fonetickou transkripci pro jejich převod, avšak to je značně neefektivní řešení, jelikož bychom museli rozsah algoritmu rozšířit natolik, aby pokryl všechny světové jazyky, ze kterých slova přechází. [1]

Jednodušším a mnohem využívanějším řešením je vytvoření slovníku, kde budou dané převzatá slova definována přímo jejich výslovností, tím pádem u nich neproběhne fonetická transkripce, ale přímo se vysloví přiřazená skupina znaků. [1]

3. PRINCIPY GENEROVÁNÍ HLASOVÉHO VÝSTUPU

Hlasový výstup se na číslicových počítačích dá generovat odlišnými způsoby, čímž se zaobírá právě tato kapitola, sloužící pro teoretickou rešerši využívanou pro rozpoznání charakteristiky rozebíraného programu (viz kap. 4).

3.1 Artikulační syntéza

Jedná se o jeden ze základních přístupů, avšak oproti formantové (viz kap. 3.3) a konkatenační syntéze (viz kap. 3.2) se přímo zabývá samotným vytvářením řeči. [1]

Celá tato metoda se zakládá na vytváření fyzikálních modelů jednotlivých prvků hlasového ústrojí člověka. Jejich spojením vzniká artikulační model, který se snaží reprezentovat celkovou funkci hlasového ústrojí i s jeho omezeními (např. neschopnost přejít okamžitě z jedné artikulační polohy do druhé). [1]

Obecně lze prohlásit, že syntetizátory vytvořené na principu této syntézy jsou schopny generovat nejvěrohodnější řeč. Její kvalita je vysoce ovlivněna obsáhlostí vytvořeného modelu. V něm jsou často jednotlivé prvky zjednodušovány tak, aby bylo možno navrhovaný generátor vhodně realizovat, čímž je zároveň snižována kvalita generovaného hlasového výstupu. [1]

Jedná se tedy o značně komplikovanou metodu, která ale dokáže replikovat i případné vady řeči nebo jim podobné problematiky.

3.2 Konkatenační syntéza

Tato syntéza, v dnešní době nejvyužívanější, se zakládá na spojování předem uložených řečových jednotek k sobě a tím vytvořit souvislou řeč. Dnes se již objevují syntetizéry, které díky této metodě jsou schopny vytvářet velmi srozumitelnou řeč, mnohdy těžce rozeznatelnou od člověka.

3.2.1 Řečový korpus

Hlavní částí syntetizátorů vytvořených na principu konkatenační syntézy je řečový korpus (databáze řečových jednotek). Ten musí obsahovat minimálně všechny řečové jednotky, které se v daném jazyce vyskytují. Pro dosažení věrohodnější řeči se pak korpus rozšiřuje o ideálně všechny přípustné tvary řečových jednotek, tedy jejich výslovnost v různých situacích. [1]

Vytvoření takového korpusu probíhá vybráním vhodných hlásek, slov, vět nebo textu, který je následně předčítán člověkem, čímž získáme hlasový signál, který poté můžeme rozdělit na jednotlivé segmenty, které jsou následně uloženy do databáze a ty jsou využívány pro následné vytváření řeči. Segmenty reprezentují zvolené řečové jednotky. Na základě jejich zvolení se může řečový korpus stát paměťově náročný. [1]

3.2.2 Vytváření řeči

Předpokládá se, že vstupem takového systému je typ řeči, který již prošel fonetickou transkripcí, tedy jsou k dispozici všechny potřebné informace o tom, co se má vyslovit.

Pomocí informací získaných fonetickou transkripcí pak vybíráme z databáze jednotlivé segmenty odpovídající získaným vlastnostem, ze kterých vytváříme sekvenci řečových jednotek. Na základě požadované kvality výsledné řeči a zároveň v závislosti na rozsahu využitého řečového korpusu se takováto sekvence může, ale nemusí, modifikovat za účelem zlepšení prozaických vlastností (např. intonace na konci vět apod.). Modifikovaná (nebo nemodifikovaná) sekvence se následně řetězí tak, aby mezi jednotlivými segmenty nedocházelo ke skokovým změnám spektrálních charakteristik. Takto získáme hlasový signál, který je přiveden na výstup syntetizátoru. [1]

3.3 Formantová syntéza

Formantová syntéza řeči je založena na teorii zdroje a filtru, která spočívá ve vytváření řeči pomocí dvou nezávislých složek, a to ze zdroje buzení a lineárním akustickým filtrem. [1]

Zdroj buzení je složen ze zdroje znělých zvuků, pod kterým si můžeme představit simulaci periodických vibrací hlasivek. Dalším prvek je generátor neznělých zvuků, který je reprezentován náhodným šumem. Výstup těchto zdrojů, který je určen dle povahy požadovaného zvuku, je přiveden na filtr, který je vytvořen ze sériového nebo paralelního zapojení rezonátorů. Tímto filtrem se snažíme reprezentovat akustické vlastnosti řeči, tedy hlasový trakt člověka. [1]

Při vytváření řeči pomocí této syntézy se řídíme dle pravidel, které jsou pro jednotlivé syntetizéry vytvořeny samostatně dle požadovaného výsledku syntézy. Tato pravidla se vytváří pomocí řečového korpusu, který je vytvořen obdobně jako u konkatenační syntézy (viz kap. 3.2.1). V takto vytvořeném korpusu se následně u všech hlásek oddělí složka buzení od složky hlasového traktu kvůli usnadnění hledání jednotlivých pravidel. Pod těmito pravidly si můžeme představit nalezenou ideální shodu mezi hláskou vyslovovanou v určitých situacích a zapsání parametrů této shody do databáze pravidel. Zapisované parametry jsou frekvence, šířka pásma, doba trvání a další v závislosti na požadované kvalitě výstupu. [1]

Samotná syntéza řeči pak spočívá pouze ve vyhledání daných pravidel a jejich spojením, čímž získáváme řeč ve kvalitě odpovídající podrobnosti získaných pravidel. [1]

4. PROGRAM „HLAS“ PRO SINCLAIR ZX SPECTRUM

Kapitola se zaobírá seznámením s domácím počítačem Sinclair ZX Spectrum a aplikací „HLAS“ (viz příloha A.1), která byla pro tento počítač vytvořena a realizuje funkční hlasový výstup na základě zadaného řetězce.

4.1 Sinclair ZX Spectrum

Jedná se o domácí 8bitový počítač rozšířený v osmdesátých letech dvacátého století. Je tvořen mikroprocesorem Zilog Z80A (viz kap. 4.1.1), 16 kB ROM (Read-Only Memory) pamětí, 16 kB RAM (Random Access Memory) pamětí, čipu ULA pro tvorbu videosignálu obrazu a obnovy dynamických pamětí RAM a televizní modulátor. [3][4][5][6]

Jako monitor může sloužit jakákoliv externí obrazovka s anténním výstupem. Zadávání znaků či příkazů probíhá skrze klávesnici, která je součástí počítače. Obsahem základní desky je reproduktor, který jsme schopni ovládat pomocí jediného bitu v registru. Tento reproduktor je v naši aplikaci využitý jako hlasový výstup počítače.

Hlavním programovacím jazykem byl basic, pro který je klávesnice upravena tak, aby programátor mohl využívat jednotlivé klávesy jako zkratky k daným příkazům, které byly rozděleny do sekcí, kterým odpovídají jednotlivé přepínatelné módy klávesnice. Další možností pro programování je využití instrukční sady procesoru Zilog Z80A na úrovni strojového kódu.

4.1.1 Mikroprocesor Zilog Z80A

Mikroprocesor v tomto počítači pracuje na frekvenci 3,5 MHz. Obsahuje osmnáct 8bitových registrů a čtyři registry 16bitové. Všechny 16bitové a dva 8bitové registry jsou využity pro konkrétní účely, a to jako programový čítač (PC), ukazatel na zásobník (SP), dva indexové registry (IX, IY), vektor přerušení (I) a registr pro čítač obnovy paměti (R). Zbylých šestnáct 8bitových registrů je rozděleno do dvou sad po osmi registrech, kdy jedna sada je označována jako hlavní část registrů a druhá je alternativní částí registrů. [3][6]

V rozebírané aplikaci se aktivně využívají registry hlavní a registr ukazatele na zásobník. Registry hlavní mění svůj účel při chodu programu, tedy nelze obecně říct, že jsou jednotlivě vymezeny pro konkrétní úlohu.

4.1.2 Emulátor ZX Spin

Pro jednodušší zjištění chodu programu byl použit emulátor ZX Spin. Ten umožňuje spuštění původního programu. [9]

Emulátor nabízí možnost sledovat chod programu instrukci po instrukci. Zároveň lze možno chod aplikace zastavit v definovaných momentech (např. na specifické instrukci nebo při dosažení zadané hodnoty v registrech).

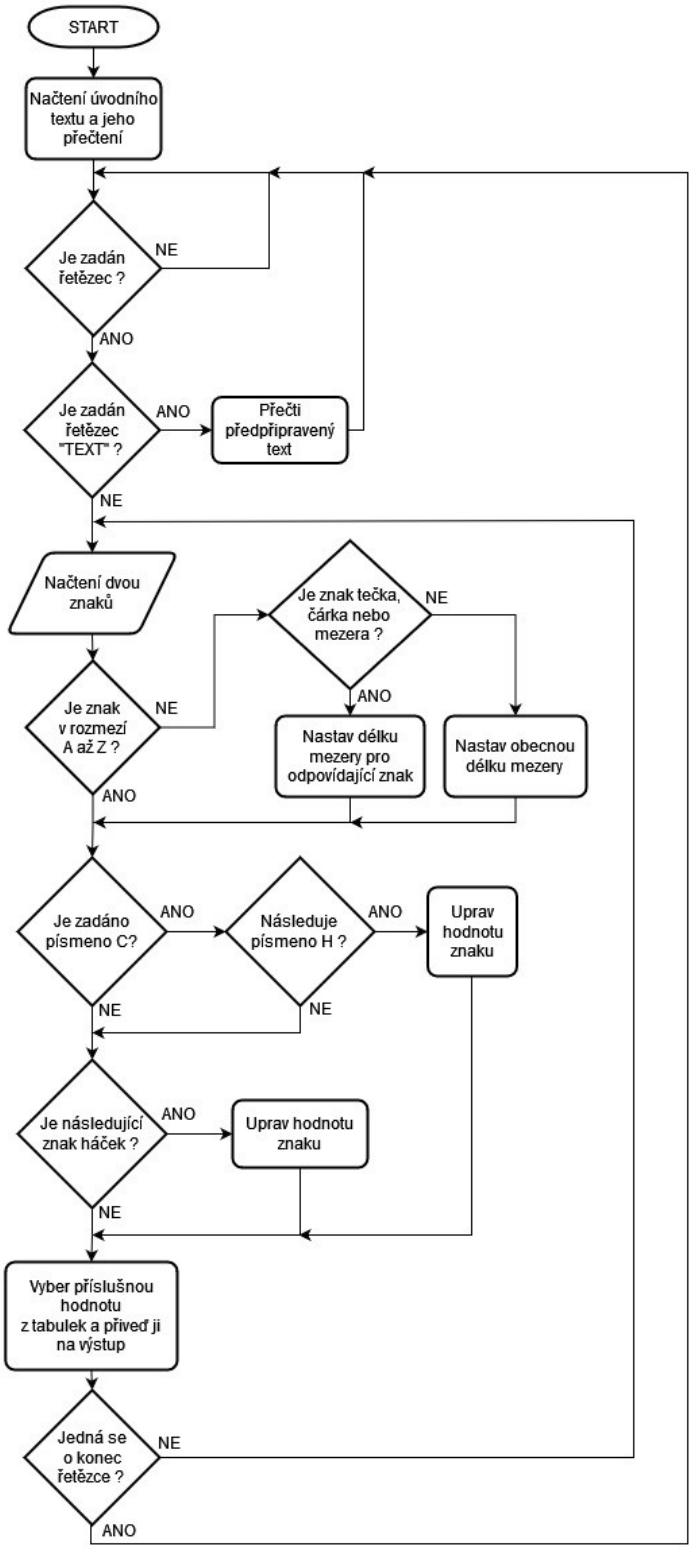
Další využitou funkcí bude možnost vytvoření záznamu hlasového výstupu. Díky této možnosti není potřeba využití externích programů, které by mohly zvukovou stopu dále ovlivnit. Toto nahrávání je vhodné pro zjištění potřebných parametrů k realizaci (viz kap. 7.1), zejména pro délky pauz nebo trvání daného stavu.

4.2 Popis aplikace

Aplikace slouží pro vytváření hlasového výstupu pro počítač Sinclair ZX Spectrum za pomocí reproduktoru zabudovaného na základní desce. Jedná se o syntetizátor, který byl publikován společností VoiceSoft roku 1985 (viz lit. [7]) a hovoříme tedy s největší pravděpodobností o prvním českém syntetizátoru pro danou platformu, který byl následně využit v několika dalších aplikací. Obecné fungování aplikace je graficky znázorněno na vývojovém diagramu (viz obrázek 4.1).

Uživatelské rozhraní aplikace je naprogramováno jazykem basic. V této části se na obrazovku vypíše předpřipravený text, který je následně přečten a představuje tak úvod a návod k použití. Další částí, kterou program v jazyce basic obsahuje, je speciální případ, kdy uživatel zadá řetězec „TEXT“, při kterém program přečte předpřipravený řetězec sloužící jako demonstrace hlasového výstupu.

Hlavní část programu je napsána ve strojovém kódu pomocí instrukční sady mikroprocesoru Zilog Z80A. V této části program nejprve vyčkává na zadání řetězce či jednotlivých znaků. Jelikož na klávesnici tohoto počítače nelze napsat české znaky, tak jsou diakritická znaménka (háček i čárka) reprezentována znakem [‘]. Následně jsou vždy dva po sobě jdoucí znaky přečteny a předloženy zjednodušené fonetické transkripci, která zjišťuje, zda se nejedná o specifický znak s diakritikou, jestli zadaný znak není mezera nebo čárka a případ, kdy načtené znaky reprezentují české písmeno [CH]. Po otestování načtených znaků dochází k vyzvednutí odpovídajících pravidel. Tyto pravidla a hodnoty sloužící k jejich správnému vybrání jsou uloženy a rozděleny do čtyř tabulek o různých rozměrech (viz kap. 4.2.5). Výsledkem zpracování je pak sekvence bitů, podle které se určuje, zda bude reproduktor zapnutý či nikoliv (viz kap. 4.2.6). Zapnutí reproduktoru vygeneruje na jeho výstupu krátký zvukový pulz, z jejichž sledu je tvořena výslovnost daného písmene. Po vyslovení dvou načtených znaků se pokračuje na další pár, dokud se načítání nedostane na konec řetězce.



Obrázek 4.1 Vývojový diagram programu HLAS

4.2.1 Čtení zadaného řetězce

```
.Start
      ld    HL,($EB36)
      di
      call  speech
      ei
      ret
.Speech
      ld    b,$0bb
.LoopChar
      ld    a,b
      ld    c,b
      ld    b,(hl)
      or    a
      ret   z
```

Zdrojový text 4.1 Čtení zadaného řetězce

Načítání znaků probíhá za pomoci šesti 8bitových registrů A, B, C, H a L. Registry H a L se využívají jako společný 16bitový ukazatel na aktuální znak v řetězci.

Nejprve se při začátku programu nastaví registr HL na ukazatel prvního znaku, poté se zakáže přerušení a provede se skok do rutiny *Speech* pro načítání znaků (viz kód). Zde se inicializují registry A, B a C a do registru B se načte ASCII hodnota daného znaku.

Následně je proveden logický součet registru A, která při dalším chodu programu testuje, zda je načtena platná hodnota znaku. Jestliže je načtena neplatná hodnota, pak se program vrací do stavu, kdy čeká na zadání řetězce.

Načtení následujícího znaku probíhá s využitím funkce *NChar*, ve které se pouze inkrementuje ukazatel na aktuální znak v registru HL a vrací se zpět do funkce *LoopChar*.

4.2.2 Testování aktuálně načteného písmene a zjednodušená fonetická transkripce

```
.Test
      ld    a,c
      cp    'A'
      jp    m,SpecialChars
      cp    'Z'
      jp    p,SpecialChars
      cp    'C'
      ld    a,b
      jp    nz,Diacritic
      cp    'H'
      jp    z,LetterCH
.Diacritic
      cp    $27
      ld    a,c
      jp    nz,SaveChar
.LetterCH
      add   a,$1a
      ld    b,$0bb
```

Zdrojový text 4.2 Test aktuálně čteného znaku

Nejprve se testuje, zda je zadaný znak v rozmezí písmen A až Z (viz zdr. text 4.2). Jestliže se písmeno v tomto rozsahu hodnot nenachází, provede se skok do funkce *SpecialChars*, která slouží pro nastavení velikosti nastávající pauzy (viz kap. 4.2.3). V prvním průběhu programu do této funkce skočí vždy, jelikož je v registru A inicializační hodnota, která se mění až po tomto testování.

Jestliže je zapsaný znak součástí české abecedy, začne se provádět zjednodušená fonetická transkripce. Nejdříve se testuje, zda je zadaným znakem písmeno [C], pokud ano, pak se testuje navazující písmeno pro výskyt znaku [H]. Jsou-li obě tyto písmena zadána po sobě, provádí se skok do funkce *LetterCH*, kde se k hodnotě registru A přičítá hexadecimální hodnota $1A_h$, čímž se nastaví reprezentující hodnota pro následné vyzvednutí parametrů pro písmeno [CH].

Pokud bylo předchozí ověření neplatné, pak se testuje výskyt diakritiky nahrazené znakem [‘] (ASCII hodnota 27_h). Jestliže se jedná o výskyt diakritiky, program přičte k hodnotě znaku hodnotu $1A_h$, čímž je písmeno převedeno na reprezentující hodnotu sloužící pro následné vyzvednutí parametrů pro písmeno s diakritikou.

4.2.3 Nastavení pauzy během hlasového výstupu

```
.SpecialChars
      ld    de,$0000
      cp    ',
      jp    z,VoiceDelay
      ld    de,$8000
      cp    '.
      jp    z,VoiceDelay
      ld    de,$4000
      cp    ' '
      jp    nz,NChar
```

Zdrojový text 4.3 Určení délky pauzy

Velikost pauzy během mluvení se nastavuje vždy při prvním načítání znaku a následně vždy při výskytu znaků [mezera], [,] a [.]. Nastavování velikosti této pauzy probíhá za pomocí registrů D a E, které slouží jako jeden 16bitový registr.

Při prvním průchodu programem se vždy registr DE nastaví na hodnotu 4000_h, která je společná i pro zadání znaků abecedy nebo znaku [mezera]. Jedná se o základní hodnotu mezery mezi jednotlivými segmenty vyslovovaného formantu.

Dále se pak testují znaky [.], [,], kdy pro znak [.] se nastaví hodnota 0000_h a pro znak [,] hodnota 8000_h. Nejdelší pauza je nastavena pro znak [.] (při vykonávání pauzy se provádí dekrementace registru DE, čímž se nastaví hodnota $FFFF_h$).

4.2.4 Vykonávání pauzy

```
.VoiceDelay
      dec   de
      ld    a,d
      or    e
      jp    nz,VoiceDelay
      ret
```

Zdrojový text 4.4 Vykonávání pauzy

Vykonávání pauzy je realizováno za pomoci jednoduché cyklu, ve kterém je postupně snižována hodnoty registru DE (viz kap. 4.2.3). Tato dekrementace se opakuje až do nabytí hodnoty 00_h v registru DE, což je ověřováno díky ukládání hodnoty registru D do registru A a následným logickým součtem s registrem E. Registr E je následně testován na hodnotu 00_h.

4.2.5 Výběr odpovídajících pravidel

Po načtení a předložení hodnot fonetické transkripci a po nastavení doby pauzy se ve funkci *SaveChar* uloží na zásobník hodnota registru BC obsahující dva načtené znaky a zároveň se uloží obsah registru HL obsahující ukazatel na poslední přečtený znak. Následuje skok do funkce *FirstTable* (viz zdr. text 4.5).

```
.FirstTable
      ld    bc,$0eaf7
      ld    l,a
      ld    h,$00
      add   hl,bc
.SecondTable
      ld    bc,$0eb6c
      ld    l,(hl)
      ld    h,$00
      add   hl,bc
```

Zdrojový text 4.5 Získání pozice parametru z druhé tabulky

Ve funkci *FirstTable* se registr BC nastaví jako ukazatel na pozici první hodnoty v první tabulce, která obsahuje pozice jednotlivých parametrů. Následně pokračuje výběr odpovídajícího prvku součtem registru BC a registru HL, který obsahuje ASCII hodnotu daného znaku, jenž slouží i jako pozice v této tabulce. Vybraná hodnota, vždy osmibitová, slouží pouze pro výběr hodnot z druhé tabulky, která obsahuje jednotlivé parametry potřebné ke generování řeči.

```
.ReadFromSecondTable
      ld    a,(hl)
      and   $0f
      ld    c,a
      inc   hl
      ld    a,(hl)
      and   $80
      or    c
      ld    c,a
      dec   hl
```

Zdrojový text 4.6 Vyzvednutí hodnoty parametru z druhé tabulky

Z tabulky druhé, která obsahuje jednotlivé parametry pro generování řeči, jsou pomocí funkce ReadFromSecondTable čteny vždy dvě osmibitové hodnoty. Nejprve se z prvního parametru vyzvednou nejnižší čtyři bity, které určují počet opakování daného segmentu. Následně se z druhého parametru vyzvedne nejvýznamnější bit indikující existenci navazujícího segmentu v případě, že je tento bit neplatný.

```
.FourthTable
      ld    a,(hl)
      rlca
      rlca
      rlca
      and   $07
      push  hl
      jp    z,EndOfTalking
      ld    de,$0ed35
      ld    l,a
      ld    h,$00
      add   hl,de
      ld    b,(hl)
      pop   hl
```

Zdrojový text 4.7 Získání pozice délky a hodnoty daného segmentu

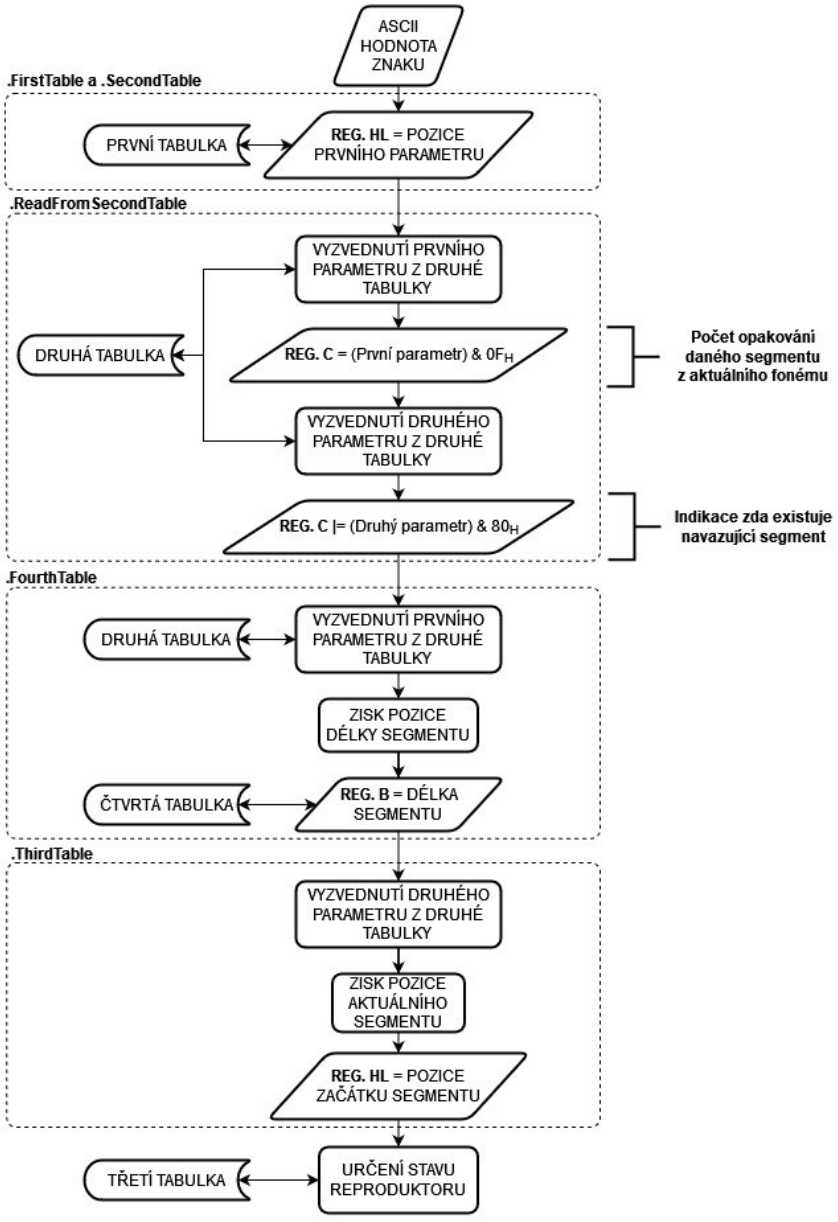
V navazující funkci *FourthTable* znovu načítá první parametr z druhé tabulky. Z něj se vyzvednou nejvyšší 3 bity, které jsou pomocí trojité cirkulační rotace převedeny na pozici nejnižších třech bitů. Po těchto logických operací ukazuje hodnota registru A na pozici délky segmentu ve čtvrté tabulce, která obsahuje jednotlivé délky segmentů.

```
.ThirdTable
      inc   hl
      ld    a,(hl)
      dec   hl
      push  hl
      rla
      ld    de,$0ec2e
      ld    l,a
      ld    h,$00
      add   hl,de
```

Zdrojový text 4.8 Získání pozice začátku aktuálního segmentu

Posledním výběrem je načtení hodnoty z tabulky třetí funkcí *ThirdTable*. V této tabulce se nachází jednotlivé formanty, které jsou většinou složeny z více menších segmentů o různé délce a počtu opakování. Pozice začátku daného segmentu je získávána pomocí druhého parametru z druhé tabulky, který je uložen v registru A. Ten je pomocí bitového posuvu vlevo o jeden bit převeden na ukazatel začátku odpovídajícího segmentu.

Celý výběr odpovídajících pravidel je znázorněn pomocí vývojového diagramu níže (viz obr. 4.2).



Obrázek 4.2 Výběr odpovídajících pravidel

4.2.6 Změna stavu piezoelektrického měniče

```
      ld    a,$80
.LoopSample
      push  af
      and   (hl)
      jp    z,SoundOff

.SoundOn
      ld    a,$10

.SoundOff
      out   ($fe),a
      ld      a,($eb33)
```

Zdrojový text 4.9 Určení stavu piezoelektrického měniče

V této části programu je procházen vyzvednutý segment po jednotlivých bitech od nejvýznamnějšího po nejméně významný. Právě čtený bit je testován pro jeho platnost. Pokud se jedná o platný bit (výsledek logického součinu je log. 1), pak je registr A nastaven na hodnotu 80_h, která je přivedena na výstup a piezoelektrický měnič je v aktivním stavu. Jestliže je však bit neplatný, provede se skok do funkce *SoundOff*, kde je na výstup přivedena logická nula, čímž je piezoelektrický měnič vypnut. Následně je do registru A načtena proměnná z adresy $EB33_h$, tedy hodnota $1C_h$, která je uživateli umožněna měnit a tím ovládat rychlost mluvení.

```
.LoopDelay
        dec     a
        jp      nz,LoopDelay
        pop     af
        dec     b
        jp      z,EndOfTalking
        or      a
        rrca
        jp      nc,LoopSample
        inc     hl
        jp      LoopSample
```

Zdrojový text 4.10 Udržování zvoleného stavu piezoelektrického meniče

Funkce *LoopDelay* udržuje aktuální stav piezoelektrického měniče pomocí dekrementačního cyklu, dokud dříve získaná hodnota registru A není nulová.

Následně je snížena hodnota registru B obsahujícího hodnotu délky segmentu vyzvednuté ze čtvrté tabulky. Pomocí cirkulační rotace vpravo je posunut registr A o jeden bit, čímž se přejde k dalšímu testovacímu vzorku ve funkci *LoopSample*. Takto je registr procházen do doby, než nastane přetečení. V případě přetečení je inkrementována hodnota registru HL a je čtena další 8bitová část segmentu.

Procházení těchto 8bitových částí se děje do doby, než je hodnota registru B nulová. V takovém případě byl přečten celý segment a program pokračuje funkcí *EndOfTalking*.

```
.EndOfTalking
      ld    hl,($eb34)
      ex    de,hl
      xor   a
      call  VoiceDelay
      dec   c
      ld    a,c
      and   $0f
      pop   hl
      jp    nz,ThirdTable
      ld    a,c
      rla
      ret   c
      inc   hl
      inc   hl
      jp    ReadFromSecondTable
```

Zdrojový text 4.11 Pauza mezi segmenty a rozhodnutí o konci čtení

V této funkci je nejprve provedena pauza mezi segmenty (viz kap. 4.2.4). Následně je testováno, zda je v registru A, který v nejnižších čtyřech bitech obsahuje počet opakování daného segmentu, nenulová hodnota. Jestliže je hodnota nenulová, pokračuje se ve zpracování daného segmentu. V opačném případě se testuje přetečení registru A po rotaci vlevo, tedy zda má formant další segment. Pokud nedošlo k přetečení, pak se dvakrát inkrementuje ukazatel na druhou tabulku uložený v registru HL, který poté ukazuje na další parametry aktuálního formantu.

Jestliže došlo k přetečení registru A, pak je vyzvednut ukazatel na poslední čtený znak z řetězce a zároveň i jeho hodnota. Ukazatel je následně inkrementován a pokračuje se funkcí *LoopChar* (viz kap. 4.2.1).

4.3 Zhodnocení

Hlasový výstup generovaný tímto programem je tvořen pomocí bitových řad (segmentů), které jsou, dle vstupního řetězce a vyzvedávaných parametrů, řazeny za sebe. Opakováním a skládáním těchto segmentů za sebe vznikají jednotlivé fonémy. Tyto segmenty nejsou často specifické pro jeden znak. Znaky s podobnou charakteristikou obvykle využívají stejné segmenty nebo dokonce jen jejich části.

Vzhledem k požadavku na 1,75 kB paměti pro výslednou aplikaci se zde nenachází fonémy přizpůsobené pro intonaci či jiné aspekty lidské řeči. Z toho důvodu je generovaný hlas monotónní. Zároveň jsou některá písmena těžko rozeznatelná, avšak vzhledem k jednoduchosti programu a využitých periferií se jedná o adekvátní kvalitu generovaného hlasového výstupu, který je naprosto dostačující pro mnoho aplikací demonstrujících základní principy hlasové syntézy.

5. VÝVOJOVÁ DESKA ARDUINO MEGA2560 REV3

Kapitola se zaměřuje na seznámení se se zvoleným vestavným systémem pro realizaci knihovny generující hlasový výstup pomocí piezoelektrického měniče.

Pro návrh i realizaci aplikace byla zvolena vývojová deska Arduino Mega2560 Rev3 osazena mikroprocesorem ATMega2560 od společnosti MicroChip. Tento vestavný systém byl zvolen především díky jeho dostupnosti nutné k vypracování práce. Zároveň není předpokládáno využití mnoha periferií, čímž bude umožněn snadný převod výsledné knihovny (viz kap. 7) na jiný systém splňující definované požadavky (viz kap. 6).

Pro programování v jazyce C je navrženo prostředí MicroChip studio [12], které umožňuje kompilaci, nahrávání i simulaci samotného vestavného systému.

5.1 Mikroprocesor ATMega 2560

Jedná se o 8bitový mikroprocesor pracující na frekvenci 16 MHz. Pro připojení periferií je zde 86 vstupně/výstupních pinů. [8]

Vestavěná paměť pro uživatele se skládá z maximálně 256 kB programovatelné paměti FLASH, 4 kB paměti EEPROM a 8 kB paměti SRAM. V paměti FLASH je pak vyhrazeno prvních 57 míst pro vektory přerušení. [8]

Mikroprocesor nabízí řadu periferií. Vzhledem k požadavkům a návrhu realizované aplikace (viz kap. 6) je podstatným 16bitový Čítač/Časovač. Ten má dedikovanou předděličku signálu a umožňuje běh v CTC režimu (Clear Time on Compare).

Další důležitou periferií je rozhraní sériové komunikace USART (Universal Synchronous/Asynchronous Receiver and Transmiter), které bude použito pro možnost odesílání generovaného řečového signálu zpět do PC. Tím bude umožněno zobrazit generovaný průběh pro ověření funkce či jeho archivace a opakované přehrávání.

5.2 Periferie vývojové desky

Samotná vývojová deska nemá zabudované externí periferie, avšak pro jejich připojení je zde 86 vstupně/výstupních pinů. Z tohoto rozhraní však budou využity pouze pin GND (Ground) a pin PF0 pro připojení piezoelektrického měniče, který umožní generování hlasového výstupu (viz obr. 6.2).

5.3 Vývojové prostředí MicroChip Studio

Jedná se vývojové prostředí od společnosti Microchip Technology navrženo pro programování jimi vyvíjených mikroprocesorů.

Prostředí nabízí možnost kompilace i nahrávání programu přímo do vývojové desky. Zároveň nabízí možnost simulace procesoru i s periferiemi. To umožňuje sledovat stavy registrů, paměti, průběh zkompilovaného programu ve strojovém kódu apod.

Výhodou je skutečnost, že celé prostředí je navrženo na rozhraní Microsoft Visual Studio, se kterým se studenti setkávají od prvního ročníku studia.

5.4 Shrnutí

Zvolená vývojová deska Arduino Mega2560 Rev3 je pro realizaci aplikace plně dostačující. Volba této desky je opodstatněna dostupností mikrokontroleru pro vývoj výsledné aplikace.

Vzhledem k požadavkům i náročnosti programu by realizace aplikace mohla proběhnout na téměř jakémkoliv vestavném systému, který má dostatečně velkou paměť a obsahuje rozhraní sériové komunikace USART.

6. NÁVRH ŘEČOVÉHO GENERÁTORU

Kapitola je zaměřena na teoretický návrh syntetizéru využívající vývojovou desku Arduino Mega2560 Rev3, ke které je připojen piezoelektrický měnič. Po uvedení přípustných řešení je proveden návrh blokového schématu, jehož funkčnost je následně diskutována.

6.1 Předpokládané chování systému

Pro samotný funkční návrh realizované aplikace je potřeba nejprve diskutovat přípustná řešení a očekávané chování systému.

6.1.1 Očekávaný vstup

Prerekvizitou správné funkčnosti systému je získání řetězce znaků či jednotlivé znaky pomocí sériové komunikace USART. Vývojová deska (viz kap. 5) nám umožňuje variantu využití vstupně/výstupních pinů pro připojení klávesnice, která by sloužila pro zadávání vstupního řetězce. Připojování této periferie by však bylo zdlouhavé a značně nepraktické, jelikož by bylo nutno jednoduchý systém rozšířit o komplikovanou periferii. Zároveň by tím byla omezena kompaktnost celého systému v případě jeho použití mimo laboratorní úlohu (viz kap. 9).

Jelikož vývojová deska umožňuje komunikaci prostřednictvím rozhraní USART, pak je efektivnějším a logičtějším řešením využití sériové komunikace pro zaslání řetězce. Takto zasílaný řetězec pak může být zadáván skrze konzolu počítače, čímž se docílí jednoduchosti při obsluze uživatelem. Zároveň tak aplikace umožní případné jednoduché rozšíření do jiných aplikací, kde je PC součástí systému.

6.1.2 Zpracování vstupního řetězce

Vstupní řetězec či jednotlivé znaky budou zpracovány fonetickou transkripcí převzaté z původního programu (viz kap. 4.2.2). Obsáhlejší fonetická transkripce by dávala smysl v případě, kdybychom se zároveň zabývali i právě přepisem klasicky zapsaného textu. V našem řešení, kdy se jedná o generování hlasového výstupu za pomoci jednoduché syntetizéru by obsáhlejší fonetická transkripce nijak neupravovala hlasový výstup, ani výstup textový, protože zvolená abeceda, která je vhodná pro zpracování textu počítačem, nepřenáší informace námi zanedbanými pravidly fonetické transkripce.

Za pomoci převedeného řetězce se pak vyberou jednotlivá pravidla, po vzoru formantové syntézy (viz kap. 3.3), které bude ovlivňován dvoustavový výstup, čímž bude generován požadovaný hlasový výstup.

6.1.3 Očekáváný výstup

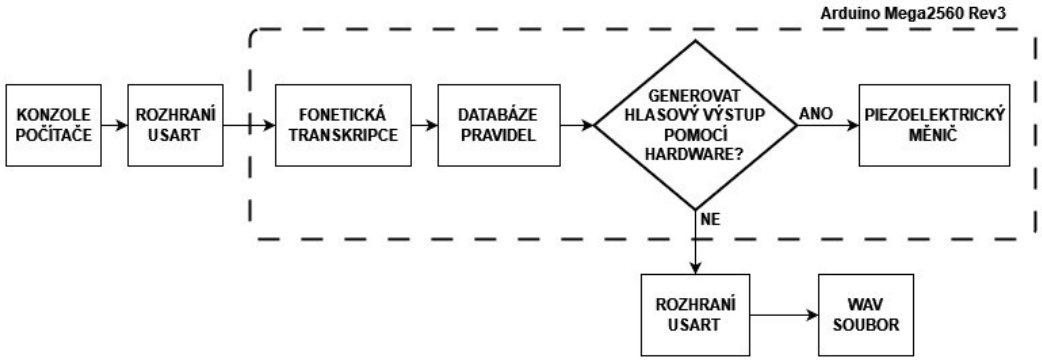
Systém bude generovat hlasový výstup za pomocí vybraných pravidel a připojeného piezoelektrického měniče, obdobně jako u rozebíraného programu „HLAS“ (viz kap. 4).

Hlavním cílem systému je zachovat srozumitelnost hlasového výstupu. Dále rozšířit fonetickou transkripci, zpřehlednit uživatelské rozhraní tak, aby mohlo být ovládáno pomocí osobního počítače a aplikaci modernizovat.

Výstup by mělo být možno odesílat pomocí rozhraní sériové komunikace USART, díky čemuž bude následně umožněno generovaný hlasový výstup zpracovávat prostřednictvím osobního počítače. Tím může být zároveň provedeno otestování správné funkčnosti aplikace za pomoci zasílání odpovídajících hodnot a jejich následnému zpracování v PC.

6.2 Blokové schéma

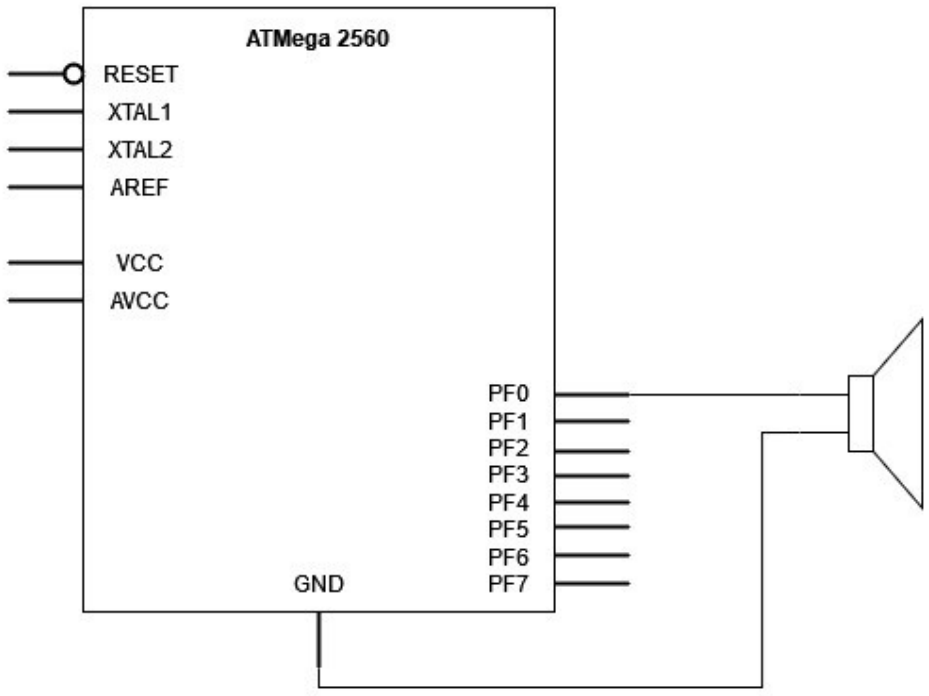
Pro usnadnění realizace knihovny umožňující generovat hlasový výstup pomocí piezoelektrického měniče je potřeba graficky znázornit předpokládané chování celého systému (viz kap. 6.1). To je znázorněno pomocí vývojového diagramu na následujícím obrázku (viz obr. 6.1).



Obrázek 6.1 Blokové schéma aplikace s implementovanými prvky

6.3 Schéma zapojení

Pro realizaci bude využita zmíněný vestavný systém Arduino Mega2560 Rev3 (viz kap. 5). K tomu bude připojen piezoelektrický měnič na vstupně/výstupní pin *PF0*. Díky měniči bude generován hlasový výstup. Schéma zapojení je zobrazeno na následujícím obrázku (viz obr. 6.2).



Obrázek 6.2 Schéma zapojení piezoelektrického měniče k vývojové desce

6.4 Ověření funkčnosti návrhu

Návrh našeho řečového generátoru je založen na znalostech získaných rozborem již fungujícího programu „HLAS“ (viz kap. 4.2). Tento návrh byl přenesen na využívanou vývojovou desku Arduino Mega2560 Rev3 (viz kap. 5) a byl rozšířen o zadávání řetězce a výstup řetězce mimo tuto platformu skrze sériovou linku.

Hlavním změnou je pak programová část, která bude psána v jazyce C a bude rozšířena o podrobnější fonetickou transkripci (viz kap. 2). Podstatným rozdílem je skutečnost, že je nemožné převzít jednotlivé hodnoty pravidel, a to z toho důvodu, že jsou hodnoty pauz a rychlosti mluvení nastavovány klasickými cykly, které jsou optimalizovány pro specifickou frekvenci mikroprocesoru. Z toho důvodu bude aplikace využívat pro generování těchto pauz čítač/časovač. Díky tomu nebude mikroprocesor permanentně blokován a bude zde možnost jednoduché implementace příkazů, které by se měly provádět v případě, že mikroprocesor čeká v pauze.

Na základě těchto informací je možno konstatovat, že navržené blokové schéma je funkčním řešením, jelikož vzhledem k již fungující architektuře programu „HLAS“ budou vytvořeny změny v určitých blocích aplikace, avšak logika vytváření hlasového výstupu nebude pozměněna.

7. REALIZACE KNIHOVNY PRO GENEROVÁNÍ HLASOVÉHO VÝSTUPU

Kapitola se zabývá realizací knihovny pro hlasového výstupu. Knihovna je psána v jazyce C. Provedení programové části je založeno na informacích získaných rozborem původního programu „HLAS“ (viz kap. 4) a následného návrhu (viz kap. 6).

7.1 Zjištění parametrů potřebných k realizaci knihovny

Pro samotnou realizaci je nejprve potřeba zjištění parametrů z původního programu „HLAS“, čímž se zabývá tato podkapitola. Jedná se o délku trvání jednoho stavu, vykonávané pauzy mezi jednotlivými segmenty formantu, vykonávané pauzy mezi jednotlivými písmeny a zároveň o délku pauz v závislosti na zpracovávaném znaku jako je [mezera], [,] a [.]. Tyto pauzy byly realizovány pomocí dekrementačních cyklů (viz kap. 4.2.4), které ale závisí na frekvenci mikroprocesoru. Pro zjištění délek těchto průběhů byl využit emulátor (viz kap. 4.1.2) a jeho funkce nahrávání zvukového výstupu, který byl následně zobrazen prostřednictvím programu Matlab [13].

7.1.1 Zobrazení generovaného hlasového výstupu

K zobrazení zvukové stopy hlasového výstupu byl napsán jednoduchý skript (viz zdr. text 7.1) v prostředí Matlab. V něm je vygenerován průběh, ze kterého je možno zjistit jednotlivé atributy generované řeči.

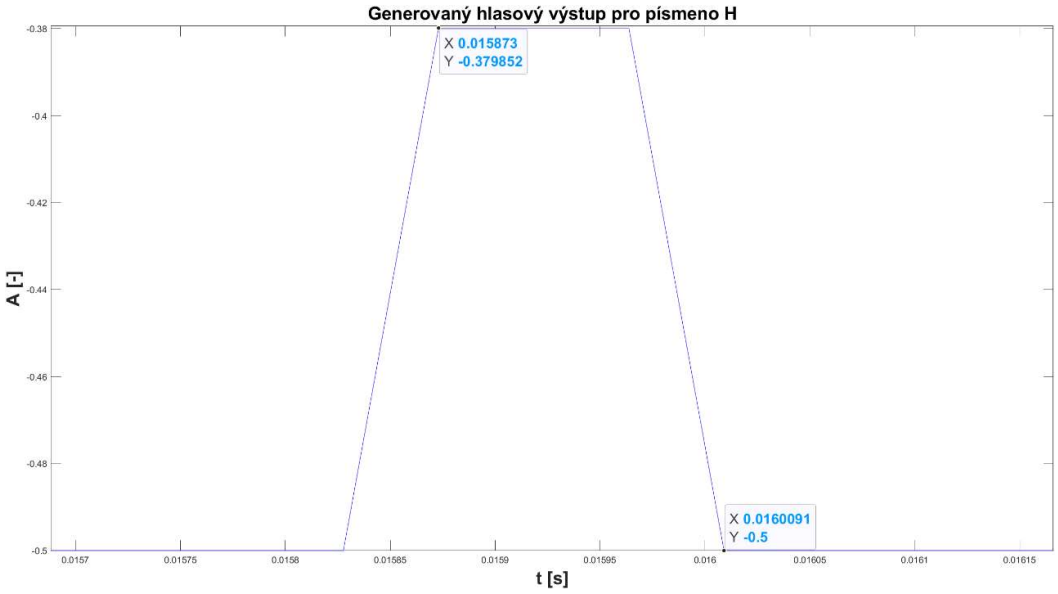
```
[s, f_vzorkovani] = audioread('emulator_pismeno_H.wav');

N = length(s);

T_vzorkovani = 1/f_vzorkovani;

t = 0:T_vzorkovani:T_vzorkovani*(N-1);

figure(1)
plot(t, s, 'b')
xlabel('t')
ylabel('A')
```

Zdrojový text 7.1 Zobrazení průběhu v prostředí Matlab

V tomto skriptu je nejprve otevřena samotná nahraná zvuková stopa z emulátoru, která byla uložena ve do souboru WAV (Waveform Audio File), ze které je vyzvednut průběh *s* a frekvence vzorkování *f_vzorkovani*. Následně je zjištěn počet vzorků *N* a perioda vzorkování *T_vzorkovani*. Z těchto hodnot je vygenerována časová osa o délce *N-1* s krokem *T_vzorkovani*. Poté je hlasový výstup zobrazen pomocí grafu.

7.1.2 Zjištění původní doby trvání jednoho stavu piezoelektrického měniče

Z původního programu bylo zjištěno, že jsou formanty vyslovovány bit po bitu (viz kap. 4.2.6). Piezoelektrický měnič je pak dle hodnoty čteného bitu udržován v daném stavu po určitou dobu za pomoci dekrementačního cyklu. Pro zjištění této doby byl zvolen průběh písmena [H] (viz příloha A.3), jelikož obsahuje v jeho prvním segmentu změnu aktivního a neaktivního stavu, kterou je snadno oddělit díky navazující pauze, která je součástí fonému.



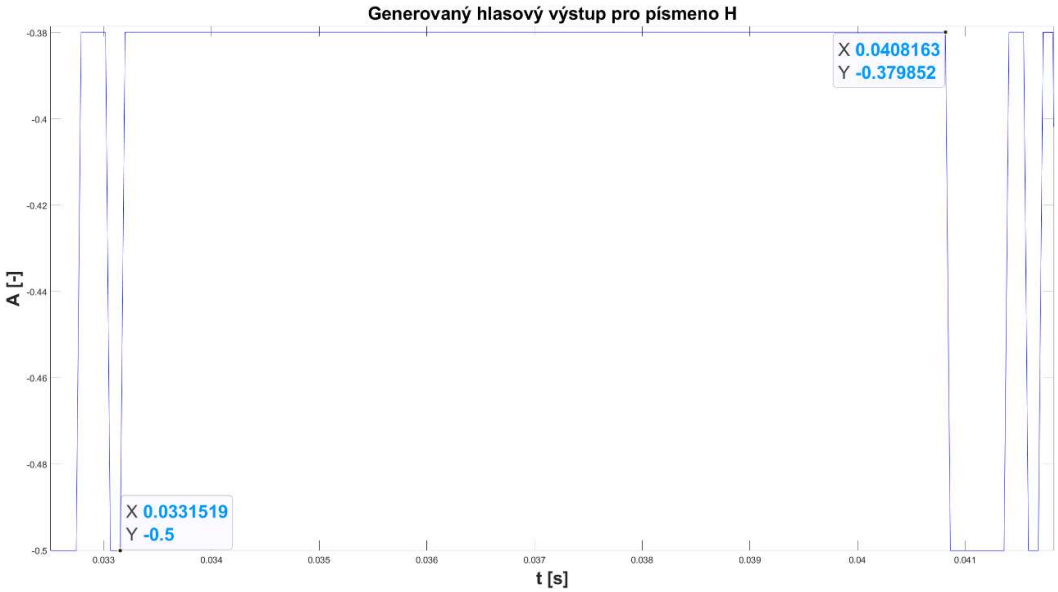
Obrázek 7.1 Průběh nahraného hlasového výstupu písmena H

Z tohoto průběhu je podstatný první segment, který lze zřetelně oddělit od zbytku signálu. Tento segment obsahuje změnu stavu po dobu trvání jednoho bitu (viz obr. 7.1). Z této změny byla zjištěna délka trvání jednoho bitu t_{BIT} dle vzorce níže (7.1):

$$t_{BIT} = t_2 - t_1 = 0{,}0159637 - 0{,}0158277 = 136\ \mu s \tag{7.1}$$

7.1.3 Zjištění původní doby trvání pauzy mezi segmenty a zároveň mezi písmeny

Vzhledem ke skutečnosti, že po každém přečtení jednoho segmentu je provedena pauza (viz kap. 4.2.3), je potřeba zjistit její délku. Pro tento účel je možno opět využít průběh písmena [H], které je složeno z pěti segmentů.



Obrázek 7.2 Průběh nahraného hlasového výstupu písmena H

Z průběhu písmena H (viz obr. 7.2) byla zjištěna doba trvání pauzy t_{SP} dle rovnice níže (7.2).

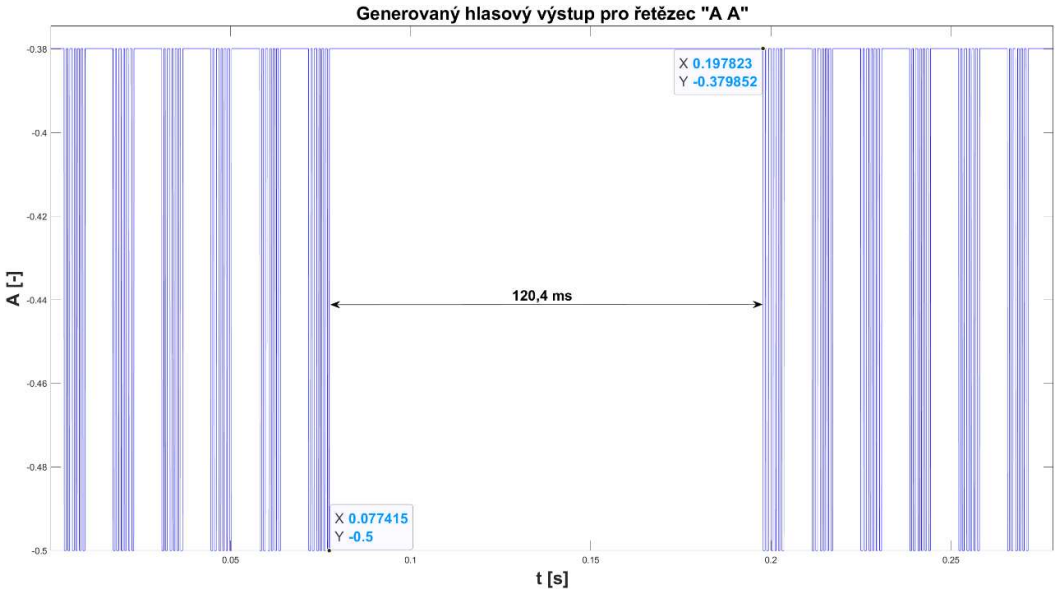
$$t_{SP} = t_2 - t_1 = 0{,}0408163 - 0{,}0331519 = 7{,}664\ ms \tag{7.2}$$

Pomocí rozboru původního programu (viz kap. 4) může být tato délka pauzy využita jako délku mezery mezi písmeny, jelikož se jedná o totožnou hodnotu.

7.1.4 Zjištění původní délky pauzy pro podporované interpunkční znaménka

Pro interpunkční znaménka jsou v původním programu definovány různé délky pauz. Pro zjištění všech těchto délek může být využit řetězec „AA“, kdy mezi písmena bude vkládáno požadované interpunkční znaménko.

Využití prostoru mezi písmeny [A] je vhodné díky průběhu jejich fonému, který začíná i končí hodnotou log. 1. Vzhledem ke skutečnosti, že pauzy představují sekvenci logických nul, je možno tyto sekvence snadno rozeznat.

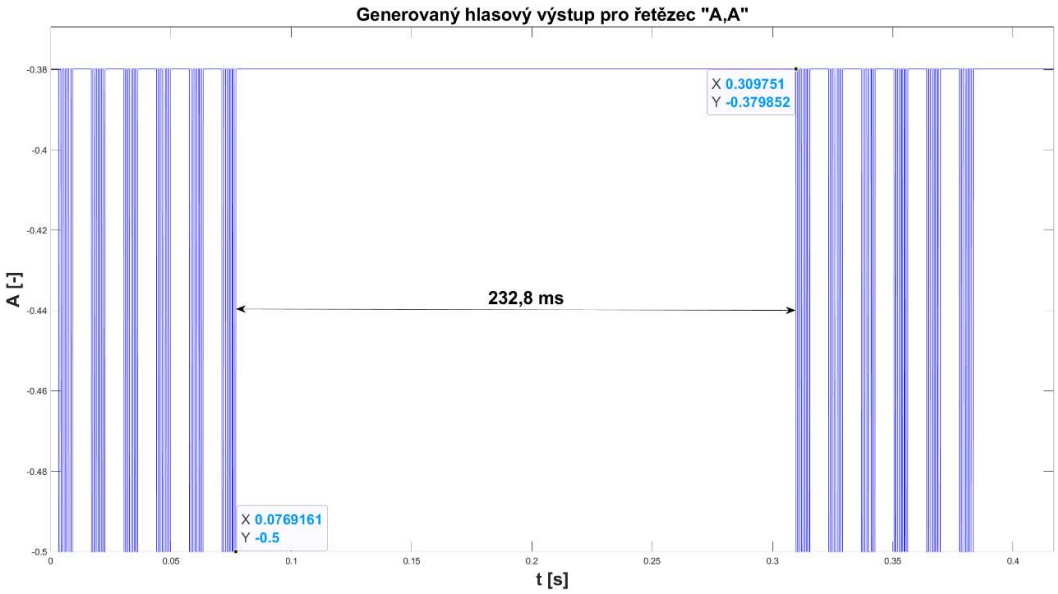


Obrázek 7.3 Průběh nahraného hlasového výstupu pro řetězec „A A“

Ke zjištění délky pauzy interpunkčního znaménka [mezera] byl využit generovaný hlasový výstup pro vstupní řetězec „A A“ (viz obr. 7.3). Z tohoto průběhu byla zjištěna délka mezery t_M pomocí rovnice níže (7.3).

$$t_M = t_2 - t_1 = 0{,}197823 - 0{,}077415 = 120{,}4\ ms \tag{7.3}$$

Pro zjištění trvání délky znaku [,] byl použit řetězec znaků „A,A“.

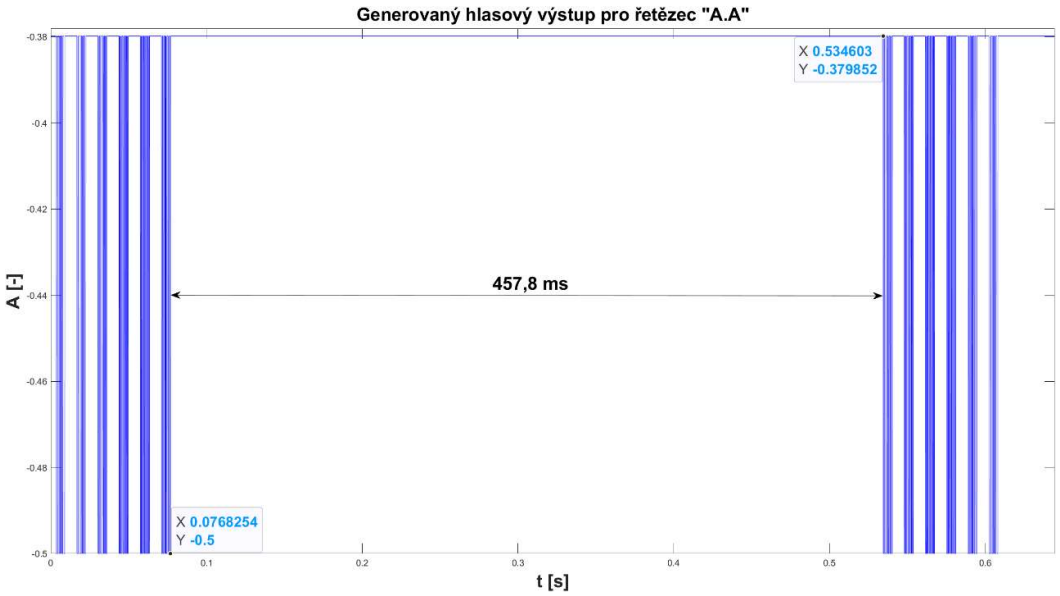


Obrázek 7.4 Průběh nahraného hlasového výstupu pro řetězec „A,A“

Z tohoto průběhu (viz obr. 7.4) byla zjištěna délka čárky t_C dle následující rovnice (7.4).

$$t_C = t_2 - t_1 = 0{,}309751 - 0{,}0769161 = 232{,}8\ ms \tag{7.4}$$

Obdobným způsobem byla zjištěna délka trvání znaku [.] za použití řetězce „A.A“.



Obrázek 7.5 Průběh nahraného hlasového výstupu pro řetězec „A.A“

Ze zobrazeného průběhu (viz obr. 7.5) byla zjištěna délka tečky t_T pomocí rovnice níže (7.5).

$$t_T = t_2 - t_1 = 0{,}534603 - 0{,}0758254 = 457{,}8\ ms \tag{7.5}$$

Jelikož bude realizovaná knihovna rozšířena o zpracování interpunkčních znamének [?] a [!], pak tato délka pauzy bude využita i pro případy zadání těchto znaků.

7.1.5 Převod parametrů

Pauzy o dříve zjištěných délkách budou vykonávány pomocí šestnáctibitového čítače/časovače. Proto je potřeba tyto hodnoty převést do hodnot odpovídající celkovému počtu pulzů, které čítač/časovač musí přijmout pro dosažení požadované délky.

Tento převod je realizován pomocí vzorce

$$TTT = \frac{t}{\frac{p}{f_{CPU}}} \quad [-], \tag{7.6}$$

kde *TTT* (Total Timer Ticks) představuje celkový počet tiků (nejmenší časová jednotka procesoru), f_{CPU} je frekvence hodinového signálu, p je hodnota předděličky hodinového signálu a t je čas pauzy, která má být vykonána.

Frekvence hodinového signálu je konstantní hodnota 16 MHz a zároveň je v této knihovně čítač/časovač spouštěn vždy s hodnotou předděličky 256_D (pokud uživatelem nebyla hodnota změněna), která dělí hodinový signál touto hodnotou. Ta byla zvolena na základě požadavku, aby nenastalo přetečení čítače. Zároveň je tak dosaženo dostatečné rozlišení pro případné ladění doby trvání stavu piezoelektrického měniče.

Převedená doba trvání jednoho stavu piezoelektrického měniče je pak vypočtena jako:

$$TTT_{bit} = \frac{t}{\frac{p}{f_{CPU}}} = \frac{136 \cdot 10^{-6}}{\frac{256}{16 \cdot 10^{6}}} = 8{,}5 \doteq 9 \tag{7.7}$$

Obdobným způsobem byly převedeny i další doby trvání, jejichž hodnoty jsou shrnuty v tabulce níže (viz tab. 7.1).

Tabulka 7.1 Tabulka převedených délek stavů pro čítač/časovač

Typ zpoždění	**TTT [-]**
Hodnota reproduktoru	9
Mezi segmenty a písmeny	479
Znak [mezera]	7581
Znak [,]	14550
Znaky [.], [?] a [!]	28613

7.1.6 Shrnutí

Byly zjištěny délky pauz. Pomocí zjištěných hodnot byly následně vypočteny odpovídající hodnoty pro čítač/časovač dle daného vzorce (viz rov. (7.6). Zjištěné i přepočtené hodnoty jsou shrnuty v tabulce níže (viz tab. 7.2).

Tabulka 7.2 Tabulka obsahující jednotlivé doby trvání stavů a jejich ekvivalent v počtu tiků čítače/časovače

Typ zpoždění	**t [ms]**	**TTT [-]**
Hodnota reproduktoru	$136 \cdot 10^{-3}$	9
Mezi segmenty a písmeny	7,664	479
Znak [mezera]	121,3	7581
Znak [,]	232,8	14550
Znaky [.], [?] a [!]	457,8	28613

Při porovnávání jednotlivých průběhů a fonémového inventáře bylo zjištěno, že změřené délky pauz nejsou přesné. Nahrávání hlasového výstupu nezachycuje některé krátké pulzy, které se však v daném fonému objevují. Při nahrávání pomocí jiných externích programu (např. nahrávací program OBS [11]) tento problém přetrvává nebo je signifikantnější. I přesto jsou hodnoty brány jako platné.

Další možnou metodou pro zjištění jednotlivých délek zpoždění by byl převod pomocí hodinové frekvence původního mikroprocesoru Z80 a počtu taktů nutných

k vykonání jednotlivých instrukcí zpožďovací smyčky. Tento převod by bylo možno provést pomocí následujícího vzorce (viz vzorec (7.8):

$$t_Z = \frac{1}{f_{Z80}} \cdot k \cdot TTT \quad [s], \tag{7.8}$$

kde t_z je délka zpoždění, f_{Z80} představuje hodinovou frekvenci mikroprocesoru Z80, k je celkový počet opakování zpožďovacího cyklu a TTT je celkový počet taktů pro jednu iteraci cyklu.

Grafická metoda měření jednotlivých délek byla zvolena pro její názornější zobrazení jednotlivých typů zpoždění, které je nutno realizovat pro správné generování hlasového výstupu. Jedná se však o méně přesnou metodu.

7.2 Inicializace pro správnou funkci knihovny

Tato podkapitola se zabývá nutnou inicializací, kterou je nutno provést pro správné fungování knihovny.

7.2.1 Inicializace potřebných hodnot a použití knihovny

Pro správnou funkci knihovny je nejprve potřeba provést úvodní inicializaci. Po připojení knihovny do projektu tato inicializace ve funkci *main*, tedy mimo samotnou knihovnu. Nejprve se nastaví pin *PF0* jako výstupní a vynuluje se obsah registru *F* (viz zdr. text 7.2).

```
#include <stdio.h>
#include "hlas.h"

void main( void )
	{
	DDRF = 0x01 // Nastaveni PF0 jako vystupni
	PORTF ^= PORTF; // Vynulovani registru F

	USART_init(UBRR); // Inicializace seriove komunikace

	sei(); // Povoleni preruseni

	while(1)
		{
		if( WholeStringReceived())
			{
			Speak(GetReceivedString(), 'r');
			}
		}
	}
```

Zdrojový text 7.2 Inicializace a volání knihovny

Dále je inicializována sériová komunikace USART pomocí funkce *USART_Init* (viz kap. 7.2.2). Tato inicializace může být provedena kdykoliv před předpokládaným obdržení vstupních dat.

Následně jsou povolena globální přerušení. Ty jsou dále využívány pro příjem dat a generování pauz.

V hlavní smyčce programu je pak čekáno na obdržení řetězce (viz kap. 7.3.2). Pokud je přijat, je spuštěno generování hlasového výstupu dle daného režimu (viz kap. 7.4).

7.2.2 Inicializace sériové komunikace

Sériová komunikace USART je inicializována pomocí funkce *USART_Init* (viz zdr. text 7.3). Ta může být použita kdekoliv během vykonávání programu, čímž je uživateli umožněno měnit její parametry nebo ji zakázat, pokud je potřeba.

```
void USART_Init(const uint16_t ubrr)
      {
      UBRR0H = (uint8_t)ubrr >> 8;
      UBRR0L = (uint8_t)ubrr;

      UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0);

      UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);

      USART_Receive();
      }
```

Zdrojový text 7.3 Inicializace sériové komunikace USART

Tato funkce využívá jako vstupní parametr 16bitovou konstantu *ubrr*. Základní hodnota pro volání této funkce je definována jako makro *UBRR*, které představuje přepočtenou rychlost přenosu pro asynchronní přenos dle rovnice níže (viz mat. vztah (7.9).

$$UBRR = \frac{f_{osc}}{16 \cdot BAUD\_RATE} - 1 \tag{7.9}$$

Z této hodnoty *UBRR* je pak bráno prvních 12 bitů, které jsou vloženy do registru *UBRR0*, čímž se nastaví rychlost přenosu. Hodnota je uživateli umožněna měnit pomocí definování uživatelského makra *BAUD_RATE*, zadávanou v bitech za sekundu, které je nastaveno na hodnotu 57600 bit/s.

Dále je funkcí povolen přenos a odesílání dat spolu s povolením úplného přerušení při přijímání dat.

Jako poslední je zavolána funkce *USART_Receive*, která vyzvedává bytovou hodnotu z registru *UDR0*, čímž je následně vynulován. To zajišťuje, že je přijímací (nebo odesílací) registr *UDR0* po inicializaci prázdný.

Pro správné fungování knihovny není potřeba tuto funkci volat v případě, že uživatel inicializuje sériovou komunikaci jiným způsobem, avšak je potřeba využít modul *USART0*.

7.2.3 Tabulky parametrů a inventář fonémů

Tabulky parametrů a inventář formantů jsou definovány jako globální konstanty. Jejich obsahem jsou pak hodnoty potřebné ke generování hlasového výstupu po vzoru původního programu (viz příloha A.1), ze kterého byly převzaty.

První tabulka *parameterPositions* obsahuje hodnoty, které představují pozici jednotlivých parametrů v tabulce druhé (viz kap. 7.4.2). Hodnoty z této tabulky jsou vyzvedávány na základě ASCII hodnoty daného písmena (viz kap. 7.4.2) v převedeném řetězci (viz kap. 7.3.3).

Druhá tabulka *parametricTable* obsahuje hodnoty jednotlivých parametrů. Ty slouží ke zjištění pozice segmentu čteného formantu, počet opakování a délky tohoto segmentu.

Fonémový inventář je reprezentován tabulkou *phonemeTable*, která obsahuje jednotlivé fonémy. Ty jsou skládány ze segmentů (v ojedinělých případech je foném složen z opakování pouze jednoho segmentu), které jsou vyzvedávány na základě hodnot parametrů z *parametricTable* (viz kap. 7.4.3) tzn. daný foném nemá dedikovanou polohu v tabulce, ale jsou rozmístěny v různých pozicích.

Poslední tabulka *segmentLength* obsahuje délky jednotlivých segmentů daných fonémů. Pozice odpovídajících délek je rovněž získána za pomocí parametrů z *parametricTable* (viz kap. 7.4.2).

7.3 Přijímání a zpracování vstupního řetězce

Vstupní řetězec je potřeba získat a zároveň převést pomocí zjednodušené fonetické transkripce, čímž se zaobírá tato podkapitola. Z důvodu zpracovávání jsou na řetězec kladeny nároky, které usnadňují tento proces. Tyto nároky upravují zápis českých znaků s diakritikou tak, že je jejich zápis rozšířen do dvou znaků místo jednoho.

7.3.1 Požadovaný formát vstupního řetězce

Knihovna podporuje znaky ASCII, která však nepodporují zadávání českých znaků obsahující čárku nebo háček, které jsou součástí znakové sady UTF-8 (Unicode Transformation Format). Z toho důvodu jsou na vstupní řetězec kladeny požadavky při jeho zadávání.

Jestliže zadávané písmeno obsahuje diakritiku, pak je potřeba upravit zápis tohoto znaku. Pokud se jedná o čárku nad písmenem, je nutno zápis změnit přidáním apostrofu (např. písmeno [Á] je zadáno jako [A‘]). Pakliže se jedná o háček, pak je zadávána stříška (např. písmeno [Č] je zadáno jako [C^].

Dalším požadavkem je absence znaků, které nejsou součástí české abecedy a interpunkčních znamének. Toto pravidlo je zavedeno z důvodu zachování jednoduchosti aplikace, které však ovlivňuje její využití pouze v ojedinělých případech.

Mimo dodržení maximální délky řetězce, která může být uživatelem upravena (viz kap. 7.3.1), nejsou kladeny další požadavky.

7.3.2 Přijímání vstupního řetězce pomocí sériové komunikace

Řetězec je přijímán pomocí sériové komunikace pomocí vektoru přerušení *USART0_RX_vect* (viz zdr. text 7.4). Obsluha přerušení je zavolána v případě, že jsou v přijímacím registru *UDR0* nevyzvednutá data. Toto přerušení je povoleno při inicializaci sériové komunikace pomocí funkce *USART_Init* (viz kap. 7.2.2).

```
ISR(USART0_RX_vect)
	{
	uint8_t receivedChar = UDR0; // Vyzvednuti prijateho znaku

	if( receivedChar == '\0' || receivedString_Ptr >= BUFFER_SIZE-1)
		{
		receivedString[receivedString_Ptr] = '\0';
		receivedString_Ptr = 0;
		newString = true;
		}
	else
		receivedString[receivedString_Ptr++] = receivedChar;
	}
```

Zdrojový text 7.4 Obsluha přerušení pro přijímání řetězce

V obsluze přerušení je vyzvednuta 8bitová hodnota z přijímacího registru *UDR0*. Následně je testováno, zda se jedná o konec řetězce a jestli délka přijímaného řetězce *receivedString* nepřesáhla maximální délku pomocí ukazatele na jeho poslední prvek *receivedString_Ptr*. V případě, že je testování platné, přivede se na konec řetězce indikace posledního znaku (tj. '\0'), resetuje se hodnota ukazatele na aktuální znak a nastaví se signalizace příjmu celého řetězce *newString*. Hodnota lze z důvodu snadnější obsluhy vyzvednout pomocí definované funkce *WholeStringReceived*, která vrací její hodnotu a zároveň ji resetuje, pro možnost indikace dalšího řetězce, který může být přijat po sériové lince. Samotný řetězec je možno předat pomocí funkce *GetReceivedString*, která jej vrací.

Jestliže bylo testování neplatné, vloží se vyzvednutý znak na konec řetězce a inkrementuje se hodnota ukazatele na aktuální znak v řetězci.

Maximální délka přijímaného řetězce je nastavována pomocí makra *BUFFER_SIZE*. Základní hodnota je nastavena na 516 znaků. Tuto délku je možné měnit dle potřeby i mimo knihovnu pomocí definice stejnojmenného makra.

Knihovna nevyžaduje, aby řetězec byl přijat pomocí sériové komunikace. Řetězec může být zadán jakýmkoliv způsobem, avšak musí splňovat dané požadavky (viz kap. 7.3.1), aby mohl být zpracován za účelem generování řečového výstupu (viz kap. 7.3.3).

7.3.3 Zjednodušená fonetická transkripce

Řetězec je převáděn dle zjednodušené fonetické transkripce za pomoci funkce *ConvertToSpeech* (viz příloha A.4).

Funkci je předkládán řetězec, který má být zpracován prostřednictvím vstupního parametru *stringToConvert*. Z něj jsou čteny jednotlivá písmena.

Nejprve je znak převeden na verzálky a uložen do výstupního parametru *convertedString*. Následně je předložen posloupnosti podmínek, které testují výskyt diakritiky nebo zda je znak ovlivněn písmeny [I] nebo [Ě]. Posledním testem je přítomnost českého znaku [CH]. Pokud byla jedna z podmínek splněna, pak je vyzvednutý znak převedenn na malé písmeno, případně při výskytu znaku [CH] je proveden převod na znak [h].

Tento převod se vykonává, dokud nebyl převeden celý řetězec. Poté je zakončen indikací ukončení řetězce.

Funkce vrací logickou hodnotu, čímž lze testovat, zda převod proběhl úspěšně či nikoliv. Pokud převod proběhl správně, pak je návratová hodnota rovna *true*. Opačný příklad může nastat v případě, kdy byl zadán neplatný řetězec nebo je jeho součástí nepodporovaný znak (viz kap. 7.3.1).

Výsledkem funkce je převedený řetězec, který je vhodný pro vyzvednutí parametrů potřebných ke generování hlasového výstupu (viz kap. 7.4).

Zjednodušená fonetická transkripce je prováděna před generováním hlasového výstupu přímo ve funkci *Speak* (viz kap. 7.4). Z toho důvodu není potřeba řetězec zpracovávat mimo knihovnu, avšak funkce může být použita i mimo ni.

7.4 Generování hlasového výstupu

Generování hlasového výstupu je hlavní funkcí realizované knihovny, kterou se zabývá tato podkapitola. Celá tvorba řečového výstupu je prováděna za pomoci funkce *Speak*, ze které jsou volány další funkce knihovny potřebné pro správnou funkčnost.

7.4.1 Úvodní inicializace

Při zavolání funkce *Speak* je nejdříve otestována platnost vstupního řetězce, který je předáván skrze parametr *originalString*. Následně je zjištěna jeho délka, která nesmí přesáhnout maximální hodnotu definovanou makrem *BUFFER_SIZE*.

Následně je inicializována proměnná, sloužící pro uložení převedené formy řetězce, který je získán za pomoci funkce *ConvertToSpeech* (viz kap. 7.3.3). Jestliže převod proběhl úspěšně, řetězec je uložen v proměnné *stringForSpeech* a následuje inicializace dalších proměnných, které zajišťují správný běh funkce.

Poté je inicializován šestnáctibitový čítač/časovač za pomoci funkce *TimerInit*. Ta nastaví režim CTC, umožňující zadávat délku pauzy pomocí šestnáctibitové hodnoty, pomocí které se vygeneruje přerušení v případě, že hodnota čítače odpovídá této zadané hodnotě. Zároveň je hodnota čítače/časovače vynulována. Generování tohoto přerušení je v dalším kroku povoleno.

Dále se již pokračuje procházením převedeného řetězce (viz 7.3.3), díky kterému je generován hlasový výstup.

7.4.2 Procházení převedeného řetězce a vyzvednutí parametrů

Převedený řetězec je procházen po jednotlivých znacích. Aktuálně testované písmeno je přivedeno k ověření výskytu interpunkčního znaménka. Jestliže je testování platné, začne se vykonávat pauza o odpovídající délce pomocí funkce *Pause* (viz kap. 7.4.4).

Pokud bylo předchozí testování neplatné, pak se postupně vyzvedávají jednotlivé hodnoty, později sloužící k vyzvednutí parametrů a fonémů.

Nejprve je vyzvednuta pozice parametrů pomocí první tabulky *parameterPositions* (viz kap. 7.2.3). Z té je vyzvednuta hodnota z pozice, která závisí na ASCII hodnotě čteného znaku. V případě znaku, který není ovlivněn diakritickým znaménkem (tzn. je reprezentován velkým písmenem) je od znaku odečítána hodnota 65_D, čímž je získána pozice v první části tabulky pro právě tyto písmena.

V případě výskytu diakritického znaménka (tzn. znak je reprezentován malým písmenem) je ukazatel přesunut do části tabulky pokrývající písmena s diakritikou pomocí odečtení hodnoty 71_D.

Vyzvednutá hodnota je uložena do proměnné *positionOfParameter* (viz zdr. text 7.5).

```
if(stringForSpeech[speechString_Ptr] >= 'A' &&
   stringForSpeech[speechString_Ptr] <= 'Z')
      {
      positionOfParameter =
            parameterPositions[stringForSpeech[speechString_Ptr] - 65];
      }


else
      {
      positionOfParameter =
            parameterPositions[stringForSpeech[speechString_Ptr] - 71];
      }
```

Zdrojový text 7.5 Vyzvednutí pozice parametrů

Díky dříve vyzvednuté pozice jsou získávány hodnoty z tabulky *parametricTable* (viz kap. 7.2.3). Z té jsou brány vždy dva osmibitové parametry. Nejprve jsou z prvního parametru získány nejnižší čtyři bity definující počet opakování segmentu. Následně je z druhého parametru brán nejvýznamnější bit, který slouží jako indikace, zda má znak navazující dvojici parametrů. Tyto vyzvednuté hodnoty jsou sloučeny do osmibitové proměnné *parameterValue* (viz zdr. text 7.6). Ta tedy po těchto operacích obsahuje indikaci dalších parametru na nejvýznamnějším bitu a počet opakování aktuálního segmentu v nejnižších čtyřech bitech.

```
parameterValue = parametricTable[positionOfParameter +
                 nextparameter_pos] & 0x0F;

parameterValue |= paramtericTable[positionOfParameter + 1 +
                 nextParameter_pos] & 0x80;
```

Zdrojový text 7.6 Získání počtu opakování segmentu a indikace navazujícího segmentu

Poslední vyzvednutou hodnotou je pozice délky segmentu uchovávánou v proměnné *segmentLength_pos*. Hodnota je získávána pomocí prvního parametru (viz zdr. text 7.7), ze kterého jsou vyzvednuty nejvyšší tři bity, které jsou posunuty na pozici třech nejnižší, čímž je získána pozice délky segmentu.

```
segmentLength_pos = (parametricTable[positionOfParameter +
                    nextParameter_pos] & 0xE0) >> 5;
```

Zdrojový text 7.7 Získání pozice délky segmentu

7.4.3 Čtení segmentu

Čtení segmentu daného fonému probíhá pomocí cyklu, který je vykonáván, dokud čtyři nejnižší bity proměnné *parameterValue* (viz kap. 7.4.2) nejsou rovny nule.

V tomto cyklu je nejprve vyzvednuta délka segmentu z tabulky *segmentLength*, jejíž délka je na na získané pozici *segmentLength_pos* (viz kap. 7.4.2). Délka segmentu je uložena do proměnné *segmentLength_val* (viz zdr. text 7.8), která může mít nulovou hodnotu. V takovém případě se jedná o pauzu, která je součástí fonému (např. při čtení prvního segmentu písmena [C]).

Následně je vyzvednuta pozice začátku segmentu. Ten je zjištěn za pomoci druhého parametru druhé tabulky (viz zdr. text 7.8), jenž je logicky posunut o jeden bit vlevo. Pozice je uložena do proměnné *phoneme_pos* (viz zdr. text 7.8).

```
segmentLength_val = segmentLength[segmentLength_pos];
phoneme_pos = parametricTable[positionOfParameter + 1 +
                 nextParameter_pos] << 1;
```

Zdrojový text 7.8 Získání hodnoty délky pauzy a pozice fonému

Vykonávání programu pokračuje do složeného cyklu, ve kterém je jedenkrát přečten daný segment. Ten je čten bit po bitu pomocí proměnné *outputMask*, která je v cyklu inicializována na hodnotu 80_h, tedy s aktivním nejvýznamnějším bitem.

Uvnitř cyklů je nejprve testován režim běhu funkce *Speak*, předáváný vstupním parametrem *functionMode*, dle kterého se vykonává buďto část programu generující hlasový výstup pomocí piezoelektrického měniče nebo je výstup odesílán v podobě reprezentačních Bytech pomocí sériové komunikace.

V režimu generování je otestován právě čtený bit segmentu pomocí logického součinu s *outputMask* (viz zdr. text 7.9). Pokud je bit platný, pak je pin *PF0* nastaven do aktivního stavu. V opačném případě je pin nastaven do neaktivního stavu. Poté je provedena pauza pomocí funkce *Pause* (viz kap. 7.4.4), která udržuje pin v daném stavu po dobu 135 µs (viz kap. 7.1.2).

```
if( outputMask & phonemeTable[segment_pos] )
      {
      PORTF ^= (1 << 0);
      }
else
      {
      PORTF ^= PORTF;
      }

Pause(9, functionMode, NULL);
```

Zdrojový text 7.9 Změna stavu piezoelektrického měniče

V případě režimu odesílání je obdobně testován platný bit daného segmentu, avšak pauza je prováděna ihned a funkci je předávána osmibitová hodnota reprezentující stav bitu. V případě platného testování je přivedena hodnota FF_h, v opačném případě hodnota 00_h (viz zdr. text 7.11).

```
if( outputMask & phonemeTable[segment_pos] )
      Pause(9, functionMode, 0xFF);
else
      Pause(9, functionMode, 0x00);
```

Zdrojový text 7.11 Změna reprezentačních hodnot pro piezoelektrický měnič

Na konci cyklů je provedena dekrementace délky segmentu a hodnota masky je posunuta o jeden bit vpravo.

Po přečtení osmi bitů se čtení posune na další část segmentu. Zároveň je opětovně nastavena hodnota masky na hodnotu 80_h.

Po dokončení cyklů, tedy po přečtení celého segmentu, je provedena pauza o délce 7,664 ms (viz kap. 7.1.3), představující pauzu mezi jednotlivými segmenty fonému. Poté je snížena hodnota počtu opakování segmentu.

Jestliže počet opakování dosáhl nulové hodnoty, je otestován výskyt další dvojice parametrů. Pokud existují, pak ukazatel na čtený řetězec zůstává na daném písmenu. Dále je inkrementována pomocná hodnota ukazatele parametrů *nextParameter_pos*, která zajišťuje posun na další pár. To je znázorněno v kódu níže (viz zdr. text 7.12).

Pakliže se nevyskytuje další dvojice parametrů, pak je vynulována pomocná hodnota ukazatele (viz zdr. text 7.12) a při dalším vyhodnocení cyklů se pokračuje dalším písmenem.

```
if( !(parameterValue & 0x0F) ) // Dokonceni opakovani
      {
      if( !(parameterValue & 0x80)) // Vyskyt dalsich parametru
            {
            speechString_ptr--;
            nextParameter_pos += 2;
            }
      else
            {
            nextParameter_pos = 0;
            }
      }
```

Zdrojový text 7.12 Určení konce fonému

7.4.4 Vykonávání pauzy

Pauzy jsou vykonávány pomocí funkce *Pause*. Ta je volána pro daný znak s odpovídající délkou zjištěnou z původního programu (viz kap. 7.1), která byla převedena na počet tiků čítače/časovače (viz kap. 7.1.5). Doba trvání pauzy je předávána parametrem funkce *time* a realizována je pomocí šestnáctibitového čítače/časovače, který je inicializován před generováním hlasového výstupu ve funkci *Speak* (viz kap. 7.4.1). S ní má zároveň společný parametr *functionMode*, který rozhoduje, zda budou během vykonávání pauzy odesílány reprezentační hodnoty (tj. hodnota odpovídající stavu reproduktoru pro zpracování pomocí PC) stavu piezoelektrického měniče. Jejich hodnota je předávána parametrem *byteValue.* Ty jsou v případě odesílacího režimu zasílány po sériové lince (viz zdr. text 7.13).

```
void Pause(const uint16_t time, const char functionMode,
      const uint8_t byteValue)
      {
      timerDone = false; // Reset indikace
      OCR1A = time; // Delka pauzy
      TCNT1 ^= TCNT1; // Vynulovani citace
      uint16_t timer_mem = 0;
      TCCR1B |= (1 << CS12); // Nastaveni preddelicky a povoleni citani

      while(!timerDone)
            {
            if(functionMode == 's')
                  {
                  if(timer_mem != TCNT1)
                        {
                        timer_mem = TCNT1;
                        USART_Transmit(byteValue);
                        }
                  }
            }
      }
```

Zdrojový text 7.13 Funkce vykonávající pauzy

Po zavolání této funkce je nejprve nastavena nulová hodnota indikátoru dokončení pauzy *timerDone*. Následně je nastavena délka pauzy pomocí parametru *time*, který je vložen do registru *OCR1A*, který slouží k porovnávání s registrem *TCNT1* v CTC režimu. Pokud jsou hodnoty v těchto registrech totožné, pak je vyvoláno přerušení (viz kód níže).

Dále je inicializována proměnná *timer_mem*, která později slouží k uchování hodnoty čítače, díky které je zajištěna nezávislost odeslaných dat na přenosové rychlosti. Jako poslední je nastavena předdělička hodinového signálu pomocí registru *TCCR1B*. Ta je nastavena pro dělení frekvence hodnotou 256_D. Tato hodnota byla zvolena na základě převedených hodnot (viz kap. 7.1.6) tak, aby nedocházelo k přetečení čítače, čímž se značně zjednoduší jeho obsluha a zároveň pro dosažení dostatečného rozlišení pro případné citlivé ovládání délky trvání stavu piezoelektrického měniče. Nastavením předděličky je zároveň spuštěno čítání čítače/časovače.

Po začátku čítání je vykonáván cyklus, opakující se dokud není indikován konec čítání pomocí proměnné *timerDone*. V této smyčce je v případě nastaveného režimu odesílání předávána reprezentační hodnota piezoelektrického měniče, získána z funkce *Speak* (viz kap. 7.4.3), funkci *USART_Transmit*, která je odešle po sériové lince.

V případě režimu generování se pouze čeká na dokončení čítání.

Při shodě hodnot registrů *OCR1A* a *TCNT1* je vyvoláno přerušení. V obsluze tohoto přerušení je zastaven chod čítače pomocí vynulování hodnoty předděličky. Následně je nastavena indikace dokončení čítání díky proměnné *timerDone* (viz zdr. text 7.14). Obsluha přerušení automaticky nuluje hodnotu čítače.

```
ISR(TIMER1_COMPA_vect)
      {
      TCCR1B &= ~(1 << CS12); // Zastaveni citani
      timerDone = true; // Nastaveni indikace dokonceni citani
      }
```

Zdrojový text 7.14 Obsluha přerušení při dokončení čítání

7.5 Blokové schéma aplikace

Podkapitola slouží pro znázornění funkce knihovny pomocí blokového schématu. Knihovna byla inicializována a volána pomocí ukázkové aplikaci (viz kap. 7.2.1). Práce této aplikace je zobrazena na následujícím blokovém schématu (viz obr. 7.6).

Ze schématu lze vidět, že nejprve je inicializován pin *PF0*, na kterém je připojen piezoelektrický měnič (viz kap. 7.2.1). Následně je inicializována sériová komunikace (viz kap. 7.2.2).

Poté program vyčkává na obdržení vstupního řetězce. Jakmile je tento řetězec obdržen, je provedena fonetická transkripce (viz kap. 7.3.3) a inicializován šestnáctibitový čítač/časovač (viz kap. 7.4.1).

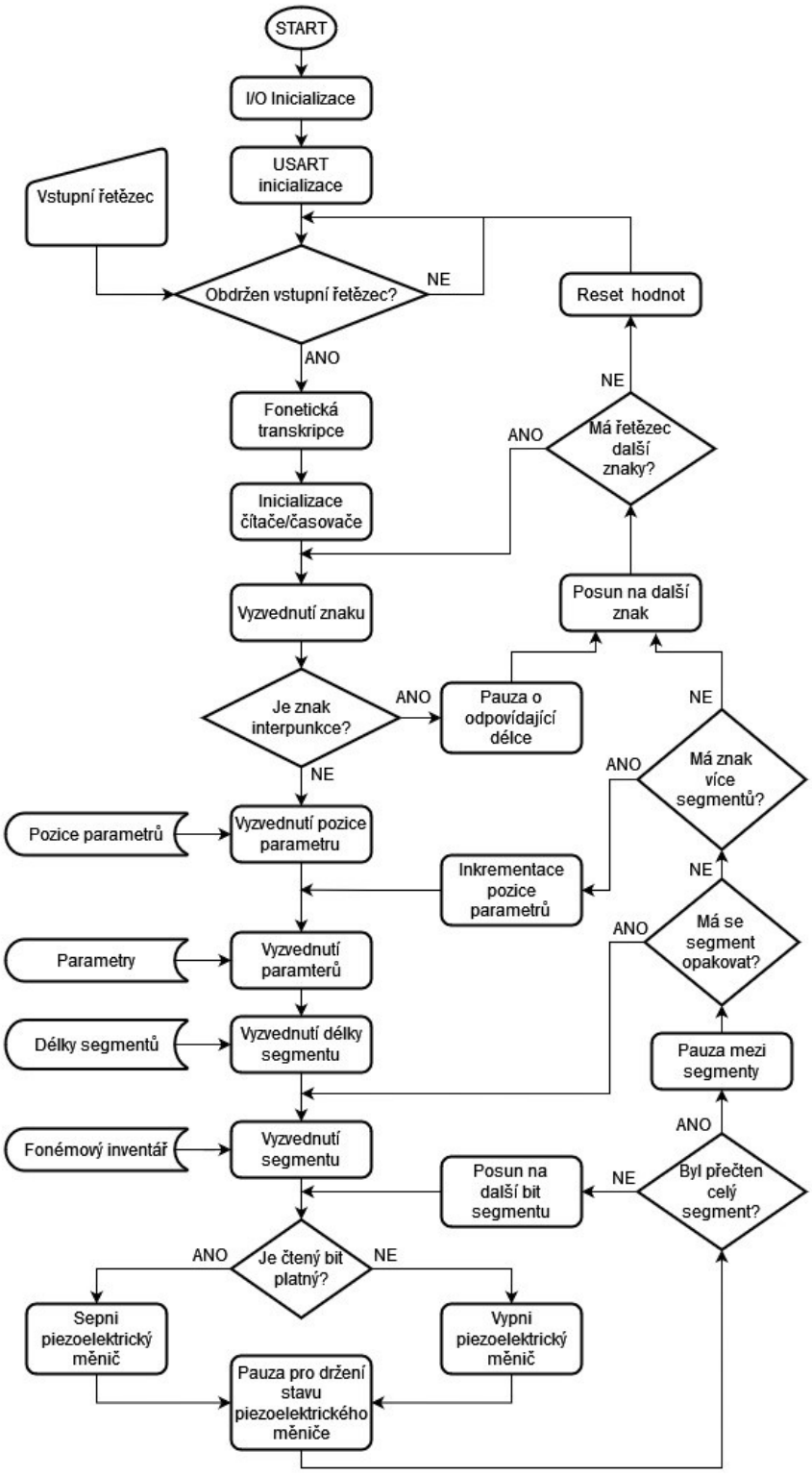
Dále je vyzvednut jeden znak z převedeného řetězce, který je nejdříve testován na výskyt interpunkce. Jestliže se jedná o tento případ interpunkce, pak je vykonána pauza dle daného znaku (viz kap. 7.4.4).

Jestliže znak není interpunkčním znaménkem, jsou pomocí něj vyzvednuty parametry z jednotlivých tabulek (viz kap. 7.4.2). Dle těchto parametrů je vyzvednut jeden segment fonému, jehož jednotlivé bity určují stav piezoelektrického měniče (viz kap. 7.4.3).

Vždy po přečtení celého segmentu je provedena série podmínek, které rozhodují o dalším výkonu programu (viz kap. 7.4.3).

Jakmile byly přečteny všechny segmenty daného znaku, je provedena inkrementace ukazatele na převedený řetězec. Jestliže ukazuje na platný znak, pokračuje se v generování hlasového výstupu.

Jestliže ukazatel není platný, pak je program navrácen do stavu, kdy čeká na příjem nového řetězce.



Obrázek 7.6 Blokové schéma aplikace využívající realizovanou knihovnu

7.6 Shrnutí

Pro správnou funkci knihovny byly z původního programu „HLAS“ (viz kap. 4.2) zjištěny potřebné parametry (viz kap. 7.1) za pomocí emulátoru ZX Spin. Tyto parametry pak byly převedeny na hodnoty šestnáctibitového čítače/časovače (viz kap. 7.1.5), kterým budou tyto pauzy vykonávány v samotné knihovně.

Knihovna byla napsána v jazyce C a umožňuje generovat hlasový výstup za pomoci piezoelektrického měniče, který je připojen na pin *PF0* vývojové desky Arduino Mega2560 Rev3. Zároveň je umožněno místo přímého generování řeči zasílat reprezentační hodnoty, které jsou následně zasílány do počítače za pomoci rozhraní sériové komunikace USART.

Vývojová deska Arudino Mega2560 Rev3 byla zvolena hlavně kvůli dostupnosti pro vypracování knihovny. Jelikož jsou využity pouze dvě periferie (čítač/časovač a rozhraní sériové komunikace USART), je možno knihovnu jednoduše převést i na jiné vestavné systémy.

Funkce knihovny byla znázorněna pomocí blokového schématu, díky kterému je zároveň proveden zjednodušený popis celé logiky knihovny (viz kap. 7.5).

Ověření funkčnosti realizované knihovny je popsáno v navazující kapitole (viz kap. 8)

8. OVĚŘENÍ FUNKČNOSTI KNIHOVNY

Pro ověření funkčnosti knihovny byla vytvořena počítačová konzole, která komunikuje s mikroprocesorem pomocí sériové komunikace USART. Knihovna byla použita obdobně jako při její inicializaci (viz kap. 7.2.1) s tím rozdílem, že je funkce generující hlasový výstup volána v odesílacím režimu, aby bylo možno hlasový výstup porovnat s výstupem emulátoru, čímž se zabývá tato kapitola.

8.1 Aplikace pro ovládání pomocí počítače

Pro podrobné ověření funkčnosti knihovny byla napsána aplikace umožňující sériovou komunikaci s mikroprocesorem prostřednictvím počítačové konzole, čímž se zabývá tato podkapitola.

8.1.1 Uživatelské rozhraní

Pro použití konzole je potřeba pouze spustit aplikaci (viz příloha A.6). Následně je prostřednictvím počítačové konzole možno zadávat potřebné hodnoty pro generování hlasového výstupu.

Nejprve je po uživateli požadováno zadání portu, na kterém je vývojová deska připojena k počítači. Ten zůstává neměnný po celou dobu používání, tedy není možno jej dále měnit.

Po zadání odpovídajícího portu je požadováno zadání názvu zvukové stopy, pod kterým bude uložen soubor WAV (Waveform Audio file format). Název souboru musí být zadán zároveň i příponou „.wav“. V něm bude uložen vygenerovaný hlasový výstup, který může být i přehrán.

Následně je vyžadován řetězec, dle kterého má být hlasový výstup generován. Na ten jsou kladeny již zmíněné požadavky (viz kap. 7.3.1). Tento řetězec je pak odeslán pomocí sériové komunikace USART do mikrokontroleru, kde je zpracován.

V případě, že je v knihovně zvolen režim odesílání generovaného hlasového průběhu se vyčkává na obdržení dat. Poté je vypsán počet odeslaných Bytů a zároveň obdržených Bytů.

Na konec je uživateli umožněno zvolit, zda se má pokračovat novým souborem a řetězcem nebo se má aplikace ukončit.

8.1.2 Nastavení rozhraní sériové komunikace USART

Inicializace sériové komunikace probíhá pomocí funkce *SerialCommSetup* (viz příloha A.6). Nejprve je v ní otevřen soubor, který představuje nastavení portu s připojenou vývojovou deskou. Ten je nastaven pro asynchronní sériovou komunikaci. Název portu je zadáván v uživatelském prostředí (viz kap. 8.1.1).

Následně jsou nastaveny parametry komunikace, které musí být totožné s nastavením na straně mikrokontroleru (viz kap. 7.2.2).

Jako poslední jsou nastaveny časové limity pro odesílání a přijímání dat.

8.1.3 Tvorba souboru s generovaným hlasovým výstupem

Vytváření zvukového souboru ve formátu WAV je zprostředkována funkcí *MakeWav* (viz příloha A.6). Ta byla psána za pomoci dokumentace k tomuto typu souboru.

Nejprve je otevřen samotný soubor s názvem zadaným uživatelem (viz kap. 8.1.1). Poté jsou již definovány a počítány potřebné hodnoty dle struktury formátu WAV.

Jedna z podstatných hodnot je perioda vzorkování, která je definována jako podíl rychlosti přenosu a počtu bitů na vzorek. Tato hodnota pak říká, že každá obdržená osmibitová reprezentační hodnota stavu piezoelektrického měniče, která je mikroprocesorem odesílána (viz kap. 7.4.4), tvoří jeden vzorek výsledného signálu.

Jako poslední jsou do souboru vkládány obdržená data představující hlasový výstup.

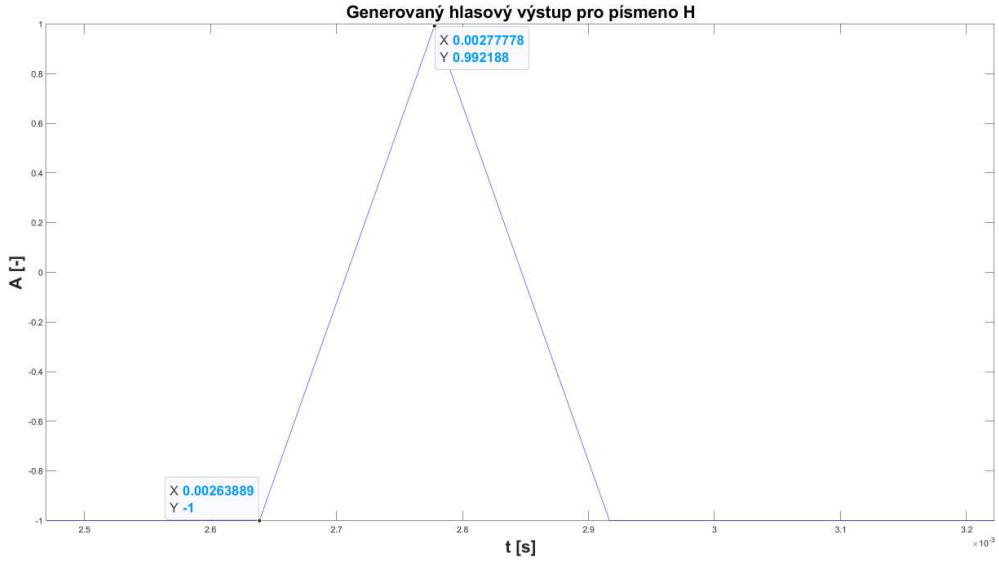
Výsledkem vykonání funkce je vytvořený WAV soubor obsahující generovaný hlasový výstup, který je možno přehrát i zobrazit.

8.2 Porovnání hlasových výstupů

K zobrazení generovaných hlasových výstupů byl znovu využit vytvořený skript pro prostředí Matlab (viz kap. 7.1.1). Porovnáním nahrávaných a generovaných výstupů se zabývá tato podkapitola. Hodnoty jsou porovnávány s již dříve získanými parametry (viz kap. 7.1.6).

8.2.1 Zjištění nové doby trvání jednoho stavu piezoelektrického měniče

Pro zjištění délky trvání aktivního stavu bylo využito písmeno „H“ z dříve zmíněných důvodů (viz kap. 7.1.2). Průběh je zobrazen na obrázku níže (viz obr. 8.1).



Obrázek 8.1 Průběh generovaného hlasového výstupu pro písmeno „H“

Z tohoto průběhu byla dle vzorce níže (8.1) zjištěna délka trvání aktivního stavu t_{BITG}.

$$t_{BITG} = t_2 - t_1 = 0{,}00277778 - 0{,}00263889 = 138{,}89\ \mu s \tag{8.1}$$

Oproti délce t_{BIT} zjištěné z hlasového výstupu emulátoru (viz kap. 7.1.2) se jedná o odchylku vypočtenou dle následující rovnice (8.2):

$$\Delta t_{BIT} = t_{BITG} - t_{BIT} = (136 - 138{,}89) \cdot 10^{-6} = -2{,}89\ \mu s \tag{8.2}$$

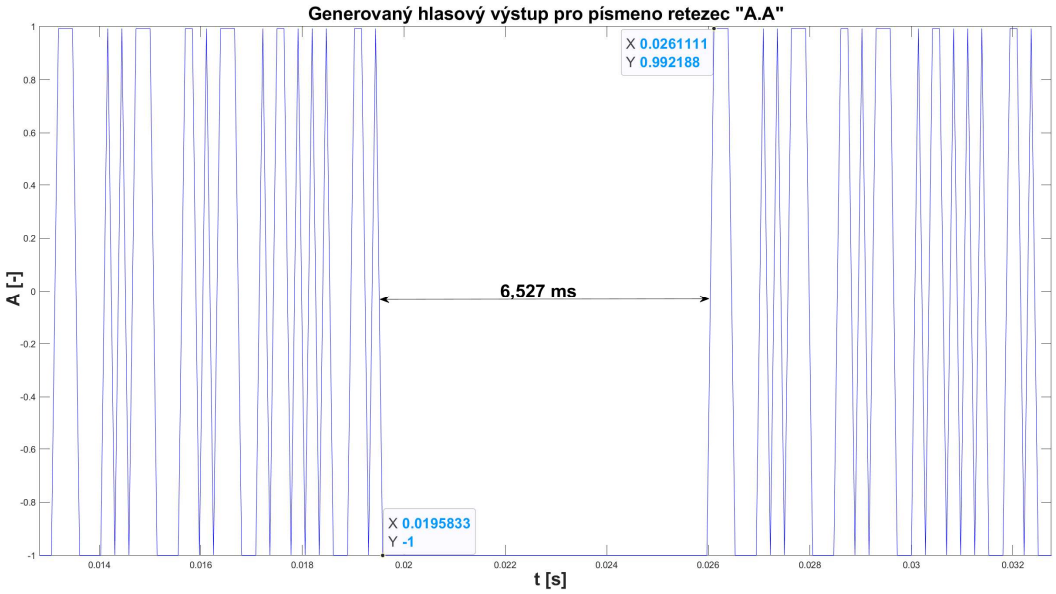
Tato chyba vznikla především díky zaokrouhlování při převodu délky trvání stavu do hodnot pro čítač/časovač (viz kap. 7.1.5).

Oproti výstupu emulátoru je zde i rozdíl v maximálních dosažených hodnotách. Ten je způsoben tím, že v případě generování jsou do souboru přímo vkládány hodnoty 00_H a FF_H, zatímco v případě druhém je zvuk pouze nahráván. Tento rozdíl má vliv pouze na hlasitost při přehrávání daného souboru.

Dalším patrným rozdílem je výskyt více pulzů v případě generování než v případě nahrávání (viz kap. 7.1). To přímo zobrazuje chybu měření popsanou v předchozí kapitole (viz kap. 7.1.6). V tomto případě se jedná o správný průběh generovaného hlasového výstupu s odpovídajícím počtem pulzů. To platí i pro další analyzované průběhy v následujících kapitolách.

8.2.2 Zjištění nové doby trvání pauzy mezi segmenty a zároveň mezi písmeny

Pro otestování délky pauzy mezi segmenty a zároveň i mezi písmeny bylo opět zvoleno písmeno „H“ po vzoru zjištění parametrů (viz kap. 7.1.3). Průběh je zobrazen na obrázku níže (viz obr. 8.2).



Obrázek 8.2 Průběh generovaného hlasového výstupu pro písmeno „H“

Z průběhu byla zjištěna délka pauzy t_{SEGG} dle následujícího vzorce (8.3).

$$t_{SEGG} = t_2 - t_1 = 0{,}0261111 - 0{,}0195833 = 6{,}527\ ms \tag{8.3}$$

Pomocí této hodnoty a hodnoty t_{SEG} (viz kap. 7.1.3) byla vypočtena odchylka (8.4).

$$\Delta t_{SEG} = t_{SEGG} - t_{SEG} = (6{,}527 - 7{,}664) \cdot 10^{-3} = -1{,}137\ ms \tag{8.4}$$

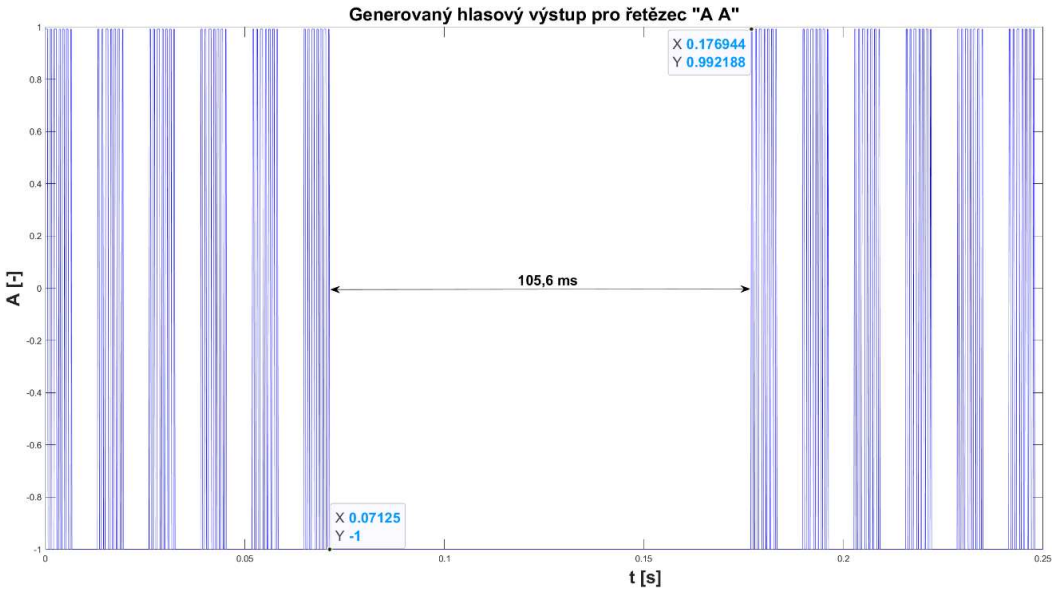
Odchylka pro délku segmentu je velmi značná. To je částečně způsobeno zaokrouhlením při převodu délky pauzy na hodnoty pro čítač/časovač.

Výraznější příčinou odchylky je nespojité odesílání reprezentačních hodnot piezoelektrického měniče. Ty jsou odesílány v cyklu, čímž vzniká zpoždění. Z toho důvodu je obdržen menší objem dat, tedy generovaná řeč je rychlejší.

Dalším faktorem je volba méně přesné metody pro měření původní délky pauzy (viz kap. 7.1.6), jelikož některé krátké pulzy nejsou při nahrávání zaznamenány.

8.2.3 Zjištění nové délky pauzy pro podporované interpunkční znaménka

Pro zjištění délky pauzy v případě interpunkčního znaku mezery byl použit řetězec znaků „A A“, jako v případě zjištění parametrů (viz kap. 7.1.4). Průběh je zobrazen na obrázku níže (viz obr. 8.3).



Obrázek 8.3 Průběh generovaného hlasového výstupu pro řetězec „A A“

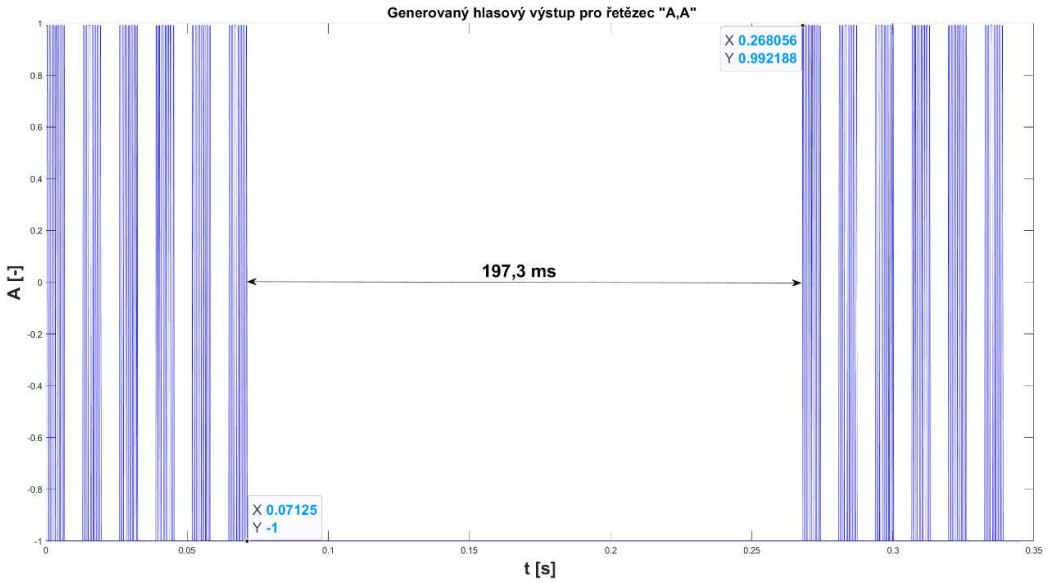
Délka mezery t_{MG} byla vypočtena dle následující rovnice (8.5).

$$t_{MG} = t_2 - t_1 = 0{,}176944 - 0{,}07125 = 105{,}7\ ms \tag{8.5}$$

Pomocí zjištěných délek t_{MG} a t_M byla vypočtena odchylka (8.6).

$$\Delta t_M = t_{MG} - t_M = (105{,}7 - 121{,}3) \cdot 10^{-3} = -15{,}6\ ms \tag{8.6}$$

Pro zjištění doby trvání interpunkčního znaku čárka byl využit řetězec znaků „A,A“. Průběh je zobrazen na následujícím obrázku (viz obr. 8.4).



Obrázek 8.4 Průběh generovaného hlasového výstupu pro řetězec „A,A“

Z průběhu byla zjištěna délka trvání pauzy t_{CG} dle následující rovnice (8.7).

$$t_{CG} = t_2 - t_1 = 0{,}268056 - 0{,}07125 = 197{,}3\ ms \tag{8.7}$$

Pomocí zjištěné délky t_{CG} a původní délky t_C (viz kap. 7.1.4) byla zjištěna odchylka dle následující rovnice (8.8).

$$\Delta t_C = t_{CG} - t_C = (197{,}3 - 232{,}8) \cdot 10^{-3} = -35{,}54\ ms \tag{8.8}$$

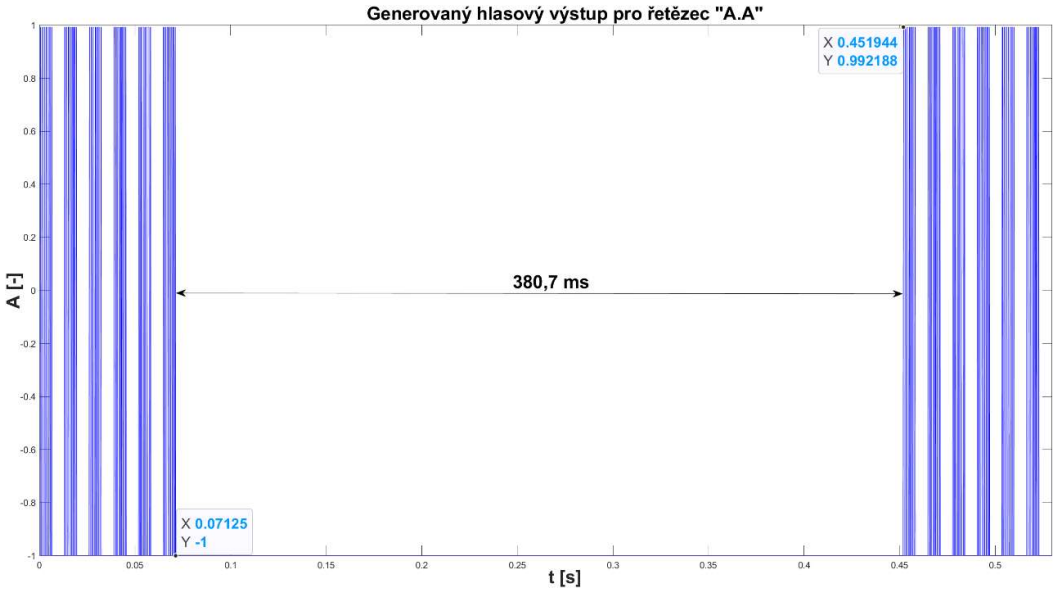
Pro zisk délek zbylých interpunkčních znamének tečka, vykřičník a otazník byl využit řetězec znaků „A.A“. Tento průběh je zobrazen na následujícím obrázku (viz obr. 8.5).

Z průběhu byly odečteny hodnoty a z nich vypočtena délka pauzy t_{TG} dle následující rovnice (8.9).

$$t_{TG} = t_2 - t_1 = 0{,}451944 - 0{,}07125 = 380{,}7\ ms \tag{8.9}$$

Pomocí zjištěné délky t_{TG} a původní délky t_T (viz kap. 7.1.4) byla zjištěna odchylka dle rovnice níže (8.10).

$$\Delta t_T = t_{TG} - t_T = (380{,}7 - 457{,}8) \cdot 10^{-3} = -77{,}1\ ms \tag{8.10}$$



Obrázek 8.5 Průběh generovaného hlasového výstupu pro řetězec „A.A“

Odchylky délek pauz pro interpunkční znaménka jsou zapříčiněny zaokrouhlením převedených hodnot (viz kap. 7.1.5), nepřesným měřením a zároveň nepřesnou reprezentací stavu piezoelektrického měniče (viz kap. 7.1.6).

Délka pauzy není signifikantním parametrem pro správnost generovaného hlasového výstupu, oproti pauze mezi segmenty nebo trvání aktivního stavu piezoelektrického měniče, a proto není potřeba převádět tuto hodnotu.

8.3 Zhodnocení

Za pomoci aplikace pro ovládání knihovny prostřednictvím konzole počítače (viz kap. 8.1.1) byly zjištěny jednotlivé délky stavů, které byly následně porovnány s délkami získanými za pomoci nahrávek hlasového výstupu emulátoru.

Pro délku trvání aktivního stavu piezoelektrického měniče byla zjištěna odchylka $\Delta t_{BIT} = -2{,}89\ \mu s$ pomocí použití hlasového výstupu generovaného pro písmeno „H“. Hodnota ukazuje, že vytvořená knihovna drží aktuální stav piezoelektrického měniče po kratší dobu. Toto odchylka však není výrazná a je způsobena především zaokrouhlením při převodu parametrů (viz kap. 7.1.5).

Odchylka délky pauzy mezi segmenty byla zjištěna jako $\Delta t_{SEG} = -1{,}137\ ms$ (viz kap. 8.2.2) za opětovného použití znaku „H“. Hodnota ukazuje, že tato pauza trvá kratší dobu při použití vytvořené knihovny.

Pro interpunkční znaménko mezery byla zjištěna odchylka $\Delta t_M = -15{,}6\ ms$. Dále pro pauzu v případě výskytu čárky byla vypočtena odchylka $\Delta t_C = -35{,}54\ ms$. Jako poslední pak byla získána odchylka $\Delta t_T = -7{,}1\ ms$ v případě zadání interpunkčních znamének

tečka, vykřičník nebo otazník. Všechny zmíněné hodnoty indikují, že vytvořená knihovna generuje kratší pauzy, tedy generovaný hlasový výstup je rychlejší.

Vypočtené odchylky ukazují výrazný rozdíl mezi výstupem knihovny a emulátoru v případě segmentů a interpunkčních znamének. Tyto odchylky jsou částečně způsobeny zaokrouhlením při převodu parametrů (viz kap. 7.1.5).

Významnější příčinou je skutečnost, že změřené parametry (viz kap. 7.1) jsou poměrně nepřesné. Tato nepřesnost vzniká při samotném nahrávání hlasového výstupu emulátoru, kdy nejsou zaznamenány některé krátké stavy, které však ve fonému mají být obsaženy.

Další nepřesnost vzniká pomocí reprezentace stavu piezoelektrického měniče, která je prováděna zasíláním hodnot při čekání na dokončení pauzy vykonávané pomocí čítače/časovače (viz kap. 7.4.4). Tímto cyklem je vytvořeno určité zpoždění, díky kterému je pak odesíláno méně reprezentačních hodnot, než je ve skutečnosti potřeba pro přesnější simulaci piezoelektrického měniče.

Vzhledem k dříve zmíněným příčinám nepřesnosti generovaného hlasového výstupu se nabízí provedení korekce jednotlivých parametrů na základě vypočtených odchylek, čímž by byly minimalizovány odlišnosti mezi výstupem knihovny a emulátoru v případě generování WAV souboru. To by však ovlivnilo i generování hlasového výstupu pomocí piezoelektrického měniče, což je hlavní funkce knihovny. Z toho důvodu byly hodnoty zachovány. Dalším důvodem zachování daných parametrů je skutečnost, že i přes odlišnosti je hlasový výstup srozumitelný.

Na základě porovnání generovaných výstupů a zároveň sluchového testování je možno konstatovat, že knihovna vytváří srozumitelný hlasový výstup, který je srovnatelný s původním programem „HLAS“ (viz kap. 4).

9. NÁVRH LABORATORNÍ ÚLOHY VYUŽÍVAJÍCÍ HLASOVÝ GENERÁTOR

Kapitola se zabývá návrhem laboratorní úlohy s využitím vytvořené knihovny generující hlasový výstup. Záměrem úlohy je seznámit studenty se základními principy sériové komunikace a prohloubení schopností práce s katalogovým listem.

9.1 Záměr laboratorní úlohy

Při realizaci více prvkových systémů je sériová komunikace běžným prostředkem pro přenášení potřebných dat mezi jednotlivými segmenty těchto systémů. Přiblížení k základním principům využití rozhraní sériové komunikace USART je hlavním záměrem navrhované laboratorní úlohy. Obecným návrhem takové úlohy se zabývá tato podkapitola.

9.1.1 Prerekvizity pro vypracování

Základním předpokladem pro úspěšné vypracování je základní znalost vývojové desky Arduino Mega2560 Rev3 a osazeným mikroprocesorem ATMega2560. Z toho důvodu je k úloze přikládán katalogový list daného mikroprocesoru, aby student mohl potřebné informace dohledat.

Další prerekvizitou je znalost základních principů fungování rozhraní sériové komunikace USART. S ním je student seznámen pomocí teoretického úvodu v rámci zadání úlohy. Zde budou shrnuty především informace potřebné k úspěšnému vypracování úlohy.

Pro realizaci je rovněž předpokládána základní znalost programovacího jazyka C.

9.1.2 Obecný návrh úlohy

Teoretickým úvodem do problematiky bude představení rozhraní sériové komunikace USART. Ten bude sloužit jako prerekvizita pro úspěšné vypracování laboratorní úlohy.

Navazující teorií bude obecné seznámení s vytvořenou knihovnou generující hlasový výstup pomocí piezoelektrického měniče (viz kap. 7). Ta bude sloužit k ověření funkční sériové komunikace s počítačem. S tím souvisí i seznámení s vytvořenou aplikací pro komunikaci mezi vývojovou deskou a počítačem prostřednictvím jeho konzole (viz kap. 8.1).

Následně bude probíhat samostatná příprava pro praktickou realizaci úlohy. Ta bude vyžadovat práci s katalogovým listem mikroprocesoru ATMega2560. Pomocí něj budou zjištěny kontrolní registry rozhraní sériové komunikace USART a hodnoty, na které musí být inicializovány. Prostřednictvím této aktivity si student vytvoří přehledný souhrn jednotlivých prvků, se kterými bude dále pracovat.

Následně bude student naveden do připraveného projektu ve vývojovém prostředí Microchip studio (viz kap. 5.3), ve kterém bude úloha vypracovávána. Zde budou připraveny deklarace a prázdné definice jednotlivých funkcí, které má student v rámci úlohy realizovat.

K ověření správnosti řešení bude sloužit zmíněná aplikace pro komunikaci s mikroprocesorem prostřednictvím počítačové konzole. Skrze ni bude odesílán vstupní řetězec. V případě, že úloha byla vypracována správně, pak bude dle něj generován hlasový výstup.

9.2 Shrnutí

Laboratorní úloha se zabývá praktickým seznámením s rozhraním sériové komunikace USART. Studentům je toto rozhraní přiblíženo prostřednictvím teoretického úvodu (viz Příloha A - Zadání laboratorní úlohy). Dále je představen způsob generování hlasového výstupu pomocí piezoelektrického měniče pomocí knihovny *hlas.h* (kap. 7).

Zadání spolu s předpřipraveným projektem se nachází v příloze (Příloha A - Zadání laboratorní úlohy). Projekt slouží pro vypracování jednotlivých úkolů. S ním jsou rovněž studenti seznámeni prostřednictvím teoretického úvodu, stejně jako s aplikací pro následné ověření funkčnosti. Tato aplikace slouží pouze k zasílání řetězce a není v ní realizován žádný z úkolů.

Výstupem úlohy je kompletní knihovna generující hlasový výstup pomocí piezoelektrického měniče, která je schopna přijímat a odesílat data na základně jejího použití (viz kap. 7).

Mezi znalosti, které studenti získají patří inicializace a obsluha základního režimu asynchronní sériové komunikace pomocí rozhraní USART. V rámci úlohy je zároveň představen základní způsob generování hlasového výstupu. Při vypracování je nutno pracovat s katalogovým listem mikroprocesoru ATMega2560, převážně s částí zabývající se rozhraním sériové komunikace. Tím jsou rozšířeny znalosti o další pojmy a parametry, které mohou být v budoucnu využity.

10. ZÁVĚR

Práce se zabývala návrhem a realizací řečového generátoru ve formě knihovny, psanou v jazyce C, která je určena pro vestavný systém Arduino Mega2560 Rev3. Ta je z části navržena na znalostech získaných úvodní rešerší principů vytváření řeči člověkem, její přepisem do textového formátu a představených způsobů hlasové syntézy. Hlavní principy byly získány rozborem již existujícího hlasového generátoru „HLAS“. Ten byl vytvořen společností VoiceSoft pro osobní počítač Sinclair ZX Spectrum a pro vytváření hlasového výstupu využívá piezoelektrický měnič. Realizovaná knihovna pak byla využita pro výukovou laboratorní úlohu, která má za cíl seznámit studenty s rozhraním sériové komunikace USART.

Nejprve byla provedena úvodní rešerše vytváření řeči člověkem (viz kap. 1.1) a možných způsobů jejího zapsání v textovém formátu pomocí fonetických abeced (viz kap. 1.2). Vzhledem k přepokládané jednoduchosti výsledného syntetizéru byla navržena modifikovaná česká fonetická abeceda mČFA, která bude pro výslednou realizaci dostačující. Následně byl popsán účel a princip fonetické transkripce, ze které byly vybrány pravidla, které skutečně mohou ovlivnit kvalitu hlasového výstupu generovaného pomocí piezoelektrického měniče (viz kap. 2).

Poté byl proveden rozbor již existujícího hlasového generátoru „HLAS“ určeného pro osobní počítač Sinclair ZX Spectrum (viz kap. 4). Tento rozbor sloužil pro získání praktických znalostí, jak může být řeč generována za pomoci piezoelektrického měniče, který je připojen k vestavnému systému.

Na základě získaných požadavků původního programu byl pro realizaci knihovny zvolen vestavný systém Arduino Mega 2560 Rev3 (viz kap. 5), který vyhovuje všem požadavkům až na přítomnost piezoelektrického měniče. Ten musí být dodatečně připojen na jeden ze vstupně/výstupních pinů. V tomto případě byl zvolen pin *PF0*.

Dle dříve zjištěné metody generování hlasového výstupu byl proveden obecný návrh realizované knihovny (viz kap. 6). Na ni byly definovány i potřebné nároky, které by po úspěšné realizaci měla splňovat. Oproti původnímu programu byl návrh rozšířen o sériovou komunikaci prostřednictvím rozhraní USART. To bylo přidáno z toho důvodu, aby zadávání vstupního řetězce bylo možno provádět prostřednictvím PC. Dále byly definován požadavek pro náhradu zpožďovacích cyklů za čítač/časovač. Celý návrh je znázorněn pomocí blokového schématu (viz obr. 6.1).

Pomocí obecného návrhu knihovny a dříve získaných znalostí původního programu „HLAS“, byla provedena její realizace v jazyce C. Byly změřeny potřebné délky zpoždění pomocí prostředí Matlab (viz kap. 7.1), které byly dále přepočteny na hodnoty pro šestnáctibitový čítač/časovač (viz kap. 7.1.5). a převzaty jednotlivé tabulky obsahující parametry, díky kterým je hlasový výstup generován (viz kap. 7.2.3). Následně byla samotná knihovna realizována. Její funkce byla vysvětlena a následně znázorněna pomocí blokového schématu (viz kap. 7.5).

Ověření funkčnosti knihovny (viz kap. 8) bylo provedeno opět pomocí skriptu v prostředí Matlab. Jednotlivé generované průběhy byly získány pomocí odesílacího režimu knihovny. Vstupní řetězec byl zadáván pomocí přiložené aplikace (viz kap. 8.1) spuštěné na osobním počítači. Pomocí té byly zároveň reprezentační hodnoty přijímány a na jejich základě byly generovány jednotlivé soubory formátu WAV, které byly v prostředí Matlab zobrazeny. Pomocí zjištěných průběhů pro jednotlivé znaky bylo zjištěno, že knihovna generuje rychlejší hlasový výstup, než původní program „HLAS“. Na základě zjištěných vlivů, které toto chování způsobují bylo konstatováno, že knihovna generuje odpovídající hlasový výstup. Mezi vlivy způsobující rozdílnou dobu vyslovování jednotlivých řetězců patří například chyba měření v původních průbězích, nepřesná reprezentace samotného piezoelektrického měniče, zaokrouhlení hodnot čítače/časovače apod.

Takto realizovaný řečový generátor má výhodu v jeho jednoduchosti. Pro samotné generování hlasového výstupu je potřeba pouze vestavný systém, který disponuje pamětí pro program alespoň 3,45 kB (v případě, že není použita optimalizace kompilátoru) a pamětí pro data minimálně 1,04 kB (v případě základního nastavení maximální délky vstupního řetězce na 516 znaků). Další podmínkou je připojení piezoelektrického měniče. Další nástavby jako je implementace sériové komunikace nebo vykonávání pauz pomocí čítače/časovače jsou spíš implementovány pro zjednodušení uživatelské obsluhy a optimalizaci knihovny.

Jedná se o vhodnou realizaci v případě, kdy je účelem seznámení se se základním principem syntetizérů či pro využití v aplikacích spíše zábavního charakteru. Generovaný řečový výstup odpovídá jednoduchosti hardwarových požadavků. Z toho důvodu je řeč monotónní a spíše „robotická“, jelikož fonémový inventář neobsahuje vzorky, které by zabezpečovaly jiné charakteristické znaky lidské řeči. Z toho důvodu mohou být některé posloupnosti znaků nesrozumitelné. Proto není vhodné knihovnu využít v případech, kdy je požadavkem dosáhnutí kvality co nejblíže lidské řeči.

Z dříve zmíněných výhod a nevýhod se nabízí několik rozšíření. Mezi ně patří možnost ovládání jednotlivých parametrů z osobního počítače. To by mohlo být realizováno za pomocí rozšíření knihovny o rozpoznávání příkazových řetězců, které by tyto parametry měnily v případě jejich zadání.

Pro rozšíření možnosti využití knihovny by mohly být přidány další fonémové inventáře sloužící pro generování hlasového výstupu v jiných jazycích. To by zahrnovalo vytvoření těchto fonémových inventářů, jejich parametrů a pravděpodobně i návrh nových metod pro výběr odpovídajících hodnot.

Dalším rozšířením by mohla být možnost generovat řeč pomocí piezoelektrického měniče v osobním počítači. Ten je v počítačích používán pro signalizaci některých chyba zároveň jako indikace spuštění PC. Tím by se značně rozšířila možnost použití knihovny.

Za použití realizované knihovny byla navrhnuta laboratorní úloha (viz kap. 9), jejímž cílem je seznámení s rozhraním sériové komunikace USART. V rámci této úlohy byl

vytvořen teoretický úvod, který popisuje rozhraní sériové komunikace, realizovanou knihovnu a zároveň aplikaci pro komunikaci s počítačem. Po tomto úvodu je požadováno zjištění jednotlivých kontrolních a stavových registrů a hodnot, na které musí být pro správnou inicializaci nastaveny. Dále je potřeba zjistit jednotlivé příznakové bity kontrolního registru. Následně se již úloha zabývá návrhem kódu v jazyce C. V rámci něj je potřeba realizovat funkci pro inicializaci sériové komunikace, přijímání dat pomocí obsluhy přerušení, přijímání dat pomocí čekání na příznakový bit a funkci pro příjímání dat pomocí čekání na příznakový bit. Kompletní podoba zadání i s teoretickým úvodem je k práci přiložena (Příloha A - Zadání laboratorní úlohy).

LITERATURA

[1] PSUTKA, Josef. *Mluvíme s počítačem česky*. Praha: Academia, 2006. Česká matice technická (Academia). ISBN 80-200-1309-1.

[2] ZAJÍČEK, Ladislav. *Bity do bytu: základy programování ve strojovém kódu - assembleru Z80* [online]. Praha: Mladá fronta, 1988 [cit. 2023-03-15]. Program Start. ISBN Bity do bytu.

[3] Z80. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-03-15]. Dostupné z: https://cs.wikipedia.org/wiki/Z80

[4] The character set. *World Of Spectrum* [online]. [cit. 2023-03-15]. Dostupné z: https://worldofspectrum.org/ZXBasicManual/zxmanappa.html

[5] *ZX Spectrum Service Manual* [online]. Revision 1.3. [cit. 2023-03-15]. Dostupné z: https://spectrumforeveryone.com/wp-content/uploads/2017/08/ZX-Spectrum-Service-Manual.pdf

[6] *ZX80 PROCESSOR: Technical Manual* [online]. 1979 [cit. 2023-03-15]. Dostupné z: http://www.nascomhomepage.com/pdf/z80-mostek.pdf

[7] VoiceSoft. *HLAS.tap* . [zdrojový text]. 1985. [cit. 2023-03-15]. Dostupné z: https://cs.speccy.cz/?scn=3&new_it=0&srt=41. Požadavky na systém: Emulátor ZX Spectrum, Počítač ZX Spectrum

[8] Atmel Corporation. *ATmega640/1280/1281/2560/2561 datasheet* [online]. Rev. 2549Q-02/2014. 1600 Technology Drive, San Jose, CA 95110 USA, 2014 [cit. 2023-03-15]. Dostupné z: https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

[9] Paul Dunn. *ZX Spin*. [software]. Mark Boyd, Damien Guard. 1. Prosince 2002 [cit. 2023-03-15]. Dostupné z: https://worldofspectrum.org/faq/emulators/windows.htm. Požadavky na systém: Windows 9x, Windows Me, 2000 or XP. Uses MMX extensions where available.

[10] HWUNG, Cecilia. WAV Files: File Structure, Case Analysis and PCM Explained. In: *VideoProc* [online]. 9.2.2023 [cit. 2023-03-15]. Dostupné z: https://www.videoproc.com/resource/wav-file.htm

[11] BAILEY, Hugh. OBS. Open Broadcaster Software, 29.1.1 [software]. 9.5.2023 [cit. 2023-03-15]. Dostupné z: https://obsproject.com/cs/download.

[12] Microchip. Microchi Studio, 7.0.2594 [software]. 20.6.2022 [cit. 2023-03-15]. Dostupné z: https://www.microchip.com/

[13] MathWorks. Matlab, R2023a [software]. 11.5..2023 [cit. 2023-03-15]. Dostupné z: https://www.mathworks.com/

SEZNAM SYMBOLŮ A ZKRATEK

Zkratky:

FEKT	Fakulta elektrotechniky a komunikačních technologií
VUT	Vysoké učení technické v Brně
IPA	International phonetic alphabet
SAMPA	Speech assessment methods phonetic alphabet
ASCII	American standard code for information interchange
ČFA	Česká Fonetická Abeceda
CTC	Clear timer on compare
USART	Universal synchronous/asynchronous receiver and transmiter
GND	Ground
WAV	Wave audio file format
TTT	Total timer ticks
UTF	Unicode transformation format

Symboly:

t_1	Čas začátku stavu	(s)
t_2	Čas konce stavu	(s)
t_{BIT}	Délka stavu piezoelektrického měniče	(s)
t_S	Délka pauzy mezi segmenty nebo písmeny	(s)
t_C	Délka pauzy pro čárku	(s)
t_T	Délka pauzy pro tečku, vykřičník a otazník	(s)
TTT	Celkový počet tiků čítače/časovače	(-)
t	Délka pauzy	(s)
p	Hodnota předděličky signálu	(-)
f_{CPU}	Frekvence mikroprocesoru	(Hz)
t_{BITG}	Generovaná délka stavu piezoelektrického měniče	(s)
t_{SG}	Generovaná délka pauzy mezi segmenty a písmeny	(s)
t_{CG}	Generovaná délka pauzy pro čárku	(s)
t_{TG}	Generovaná délka pauzy pro tečku, vykřičník a otazník	(s)
Δt_{BIT}	Odchylka délek pauz piezoelektrického měniče	(s)
Δt_S	Odchylka délek pauz mezi segmenty a písmeny	(s)
Δt_C	Odchylka délek pauz pro čárku	(s)
Δt_T	Odchylka délek pauz pro tečku, vykřičník a otazník	(s)

SEZNAM PŘÍLOH

Příloha A - Zadání laboratorní úlohy

Laboratorní úloha pro realizaci sériové komunikace pomocí rozhraní USART

1. Cíle úlohy

- Seznámení s rozhraním sériové komunikace USART
- Zdokonalení schopnosti práce s katalogovým listem
- Praktická realizace sériové komunikace mezi vývojovou deskou Arduino Mega2560 rev3
- Ukázka možnosti generování hlasového výstupu za pomoci piezoelektrického měniče

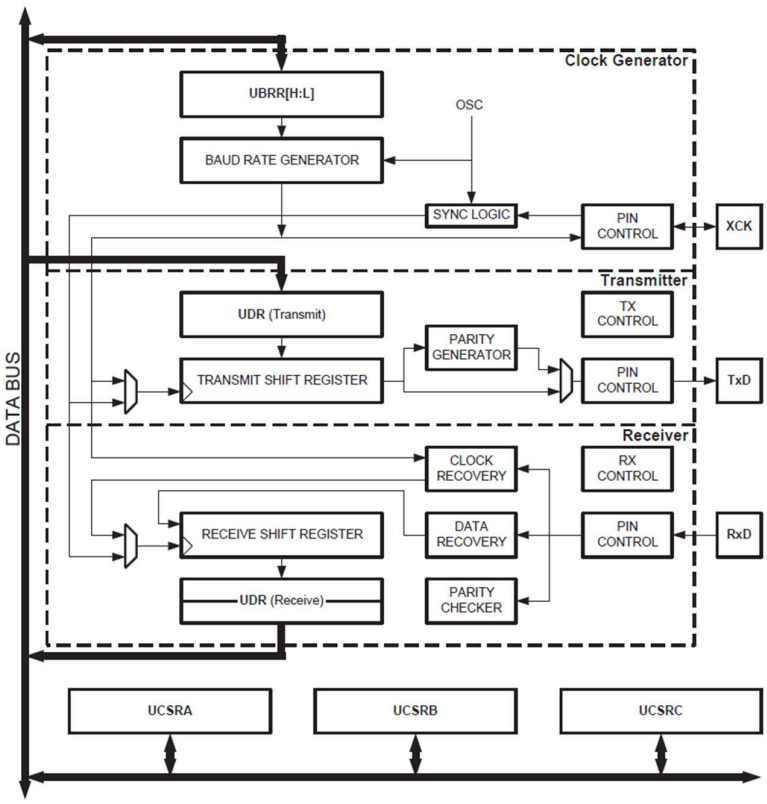
2. Teoretický úvod

2.1 Rozhraní sériové komunikace USART

USART (Universal Synchronous or Asynchronous Receiver and Transmitter) je externí hardwarový prvek představující rozhraní sériové komunikace umožňující synchronní nebo asynchronní obousměrnou komunikaci (full-duplex) mezi periferiemi nebo systémy.

Často je možno se setkat s obdobným rozhraním sériové komunikace s označením UART (Universal Asynchronous Receiver and Transmitter). V takovém případě se jedná o stejný typ komunikace, avšak tento modul neumožňuje její synchronní režim. V případě popisovaného rozhraní USART nás však rovněž zajímá především asynchronní režim přenosu, na který je tento teoretický úvod zaměřen.

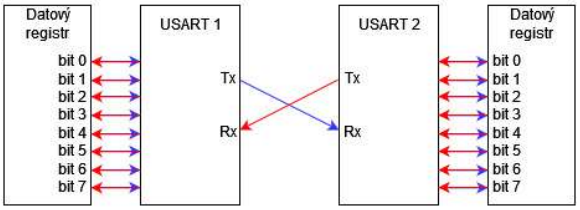
Mikroprocesor ATMega2560 nabízí čtyři moduly tohoto rozhraní. Blokové schéma jednoho z modulů je znázorněno na následujícím obrázku (viz obr. 2.1).



Obrázek 2.1 Blokový diagram jednotky pro sériovou komunikaci USART

2.1.1 Princip asynchronní sériové komunikace

Sériová komunikace probíhá prostřednictvím dvou pinů. Prvním z nich je pin *Rx* (receive) určený pro příjem dat a druhým je pin *Tx* (transmit) pro odesílání dat. Toto je znázorněno na následujícím obrázku (viz obr. 2.2).



Obrázek 2.2 Znázornění funkce sériové komunikace v asynchronním režimu

Nejprve jsou uživatelem nebo přímo vestavným systémem vloženy data do datového registru. V případě mikroprocesoru ATMega2560 se jedná o registr *UDRn* (písmeno „n“ reprezentuje číslo odpovídající použitému modulu USART). Ten je společný pro příjem i odesílání dat.

Při odesílání dat pomocí modulu *USART1* (viz obr. 2.1) jsou hodnoty z registru *UDRn* vloženy do odesílacího posuvného registru (viz obr. 2.1). Následně je generována parita v případě, že je pro ni nastaven požadavek a zároveň jsou data převáděna do nastaveného formátu (viz kap. 2.1.2.1). Poté jsou pomocí kontrolní logiky přidány synchronizační bity (start a stop bit). Dále jsou jednotlivé bity přiváděny s hodinovým signálem, jehož původ i frekvence lze ovládat (viz kap. 2.1.2.1), na odesílací pin *Tx* a jsou předávány druhému modulu *USART1* za pomoci jeho přijímacího pinu *Rx*.

Přijímaná data jsou získávána pomocí modulu „Data recovery" (viz obr. 2.1). Zároveň je ověřována parita, pakliže byla generována. Data jsou následně vloženy do přijímacího posuvného registru, odkud jsou přemístěny do registru *UDRn* odkud mohou být čteny.

Rozhraní má vestavěnou kontrolní logiku, která pomáhá ke zjištění, zda data mohou být odesílána nebo přijímána. Zároveň umožňuje kontrolu v případě, že jedna z těchto operací již byla provedena. K tomu slouží příznakové bity (flagy) (viz kap. 2.1.2.1), které rovněž generují odpovídající přerušení.

2.1.2 Programová inicializace sériové komunikace

Pro správný chod sériové komunikace je nejprve nutno definovat její parametry. Ty jsou nastavovány za pomoci registrů.

2.1.2.1 Kontrolní a stavový registr UCSRnA

Registr *UCSRnA* slouží především jako kontrolní. Jsou v něm bity určené pro indikaci jednotlivých stavů. Dle některých stavů je zároveň generováno odpovídající přerušení, pokud je povoleno v kontrolním registru *UCSRnB* (viz kap. 2.1.2.2).

Nachází se zde indikace dokončení příjmu dat pomocí bitu *RXCn*. Zároveň je zde dedikován bit *TXCn* indikující dokončení přenosu dat.

Dále se zde nachází bity kontrolující správnost obdržených dat a jejich formátu nastaveného pomocí registru *UCSRnC* (viz kap. 2.1.2.3). Mezi indikované poruchy patří chyba formátu dat indikována pomocí bitu *FEn* a zároveň chyba parity pomocí bitu *UPEn*.

Registr nabízí možnost dvojnásobné rychlosti přenosu při asynchronní komunikaci. To je umožněno pomocí bitu *U2Xn*, který sníží hodnotu předděličky rychlosti přenosu z šestnácti na osm.

Jako poslední je zde možnost nastavení komunikace do režimu multi-processor mod. Ten slouží pro systémy kde se vyskytuje více mikroprocesorů. V tomto režimu jsou daným mikroprocesorem přijaty pouze data s odpovídající adresou.

2.1.2.2 Kontrolní a stavový registr UCSRnB

Pro povolení přijímání a odesílání dat slouží kontrolní registr *UCSRnB*. V tomto registru je zároveň možno povolit generování přerušení pro odpovídající stavy. Pokud daná přerušení nejsou povoleny, pak je stavy možno kontrolovat pouze pomocí příznakových bitů v registru *UCSRnA* (viz kap. 2.1.2.1).

Dále se zde nachází pomocný bit pro nastavení formátu odesílaných a přijímaných dat. Jestliže je tento bit při nastavení formátu dat využit, znamená to, že je použito devět datových bitů. Jelikož jsou registry pouze osmi bitové, nachází se zde i rozšiřovací bity *RXB8n* a *TXB8n* pro příjem a odesílání.

2.1.2.3 Kontrolní a stavový registr UCSRnC

V registru *UCSRnC* je nastavován režim sériové komunikace.

Dále je zde umožněno nastavení formátu odesílaných a přijímaných dat. Je možno generovat jak lichou tak sudou paritu. Zároveň je zde definován počet datových bitů, který je možno nastavit v rozmezí pět až devět bitů. S formátem dat souvisí i synchronizační bity, tedy start a stop bit. Počet stop bitů je možno měnit a jsou buď dva nebo je pouze jeden. Start bit je vždy pouze jeden.

Jako poslední je umožněno nastavit polaritu hodinového signálu. V základní konfiguraci je nastavena reakce na nástupnou hranu hodinového signálu, avšak to je možno změnit na hranu sestupnou.

2.1.2.4 Registr rychlosti přenosu UBRRn

Za pomoci tohoto šestnáctibitového registru je nastavována rychlost přenosu sériové komunikace. V katalogovém listu je registr označen jeho osmibitovými částmi *UBRRnL* a *UBRRnH*, avšak v prostředí Microchip Studio je možno využit rovnou celý registr *UBRRn*.

Pro definování rychlosti přenosu je vhodné využít makro, běžně s názvem *BAUD_RATE*, ve kterém bude uchována hodnota rychlosti přenesu v bitech za sekundu. Následně je tuto hodnotu nutno převést pomocí vzorce pro daný typ komunikaci. Ten je možno získat z katalogového listu.

Po převodu je získána dvanáctibitová hodnota sloužící pro nastavení rychlosti přenosu pomocí registru *UBRRn*.

2.1.2.5 Datový registr UDRn

Jedná se o registr sloužící pro zadávání dat, která mají být zaslána a zároveň i pro přijímání dat. Pro zjištění, zda je na danou operaci registr připraven slouží stavový registr *UCSRnA*.

2.1.2.6 Požadavky na inicializaci

Pro správnou funkčnost sériové komunikace je potřeba, aby komunikující strany měly nastaveny stejné hodnoty. Konkrétně se jedná o rychlost přenosu a formát dat.

Jestliže by tyto parametry nebyly totožné na obou stranách, pak by při přenosu došlo k poškození přenášených dat, která by pak nebylo možno použít.

2.1.2 Přijímání dat

Přijímání dat může být realizováno několika způsoby, které jsou dále představeny.

Jedním ze způsobů je nepřetržité vyčkávání ve smyčce, dokud není obdržena nastavená hodnota signalizující konec dat. Tento způsob je jednoduchý pro implementaci, avšak zatěžuje mikroprocesor, jelikož díky čekání ve smyčce není možno vykonávat jinou část kódu.

Podobným způsobem, jako v předchozím případě, je možno vyčkávat na indikaci dokončení příjmu dat pomocí bitu *RXCn* (viz kap. 2.1.2.1).

Dříve popsané metody přijímání dat mají společnou možnost využití, a to v případě, že je pracováno se více systémy, které jsou na sobě závislé. V takovém případě může nastat potřeba zastavit práci jednoho ze systémů pomocí vyčkávání na potřebná data, která jsou k další práci zastaveného systému potřebná. V případě čekání na příznakový bit lze tento způsob využít při inicializaci sériové komunikace pro ujištění, že je datový registr prázdný.

Nejvhodnější metodou je přijímat data pomocí obsluhy přerušení. Pomocí bitu *RXCn* je možno generovat přerušení při obdržení dat. Toto přerušení musí být povoleno v registru *UCSRnB* (viz kap. 2.1.2.2). V tomto případě může být prováděn jiný kód a při obdržení dat se vykonávání programu přemístí do obsluhy přerušení, ve kterém jsou data vyzvednuty. Následně se program vrací zpět do původní části programu. To je výhodné díky skutečnosti, že v případě rozsáhlejší aplikace není nutno kontrolovat, kdy mohou být data přijímány. Zároveň tak lze docílit i menší spotřebě.

2.1.3 Odesílání dat

Odesílání dat je často značně jednodušší problematikou než jejich přijímání. Ve většině aplikací jsou data odesílána najednou po vykonávání určité sekvence příkazů.

Nejvhodnější metodou je vytvoření funkce, které budou předány data k odeslání. Ta následně pomocí cyklu provede odesílání. Je nutno však vždy čekat na registr *UDRn*, který musí být připraven na zápis nových dat.

2.2 Knihovna pro generování hlasového výstupu

Laboratorní úloha, zaměřující se na realizaci sériové komunikace pomocí rozhraní USART, je vypracovávána v prostředí knihovny generující hlasový výstup pomocí piezoelektrického měniče. Po funkčním vypracování úlohy slouží generovaný hlasový výstup jako ověření funkčnosti knihovny.

2.2.1 Popis funkce knihovny

Knihovna umožňuje generovat hlasový výstup pomocí piezoelektrického měniče připojeného na pin *PF0* nebo hlasový výstup zasílat v reprezentačních hodnotách tak, aby z něj mohl být vytvořen WAV (Waveform audio format) soubor za pomoci přiložené aplikace *C_Konzole_pro_komunikaci.*

Hlavní částí knihovny je funkce *Speak*, které je předáván vstupní řetězec. Ta umožňuje generovat hlasový výstup. V ní je nejprve převeden řetězec pomocí zjednodušené fonetické transkripce definované funkcí *ConvertToSpeech*. Následně je nový řetězec procházen po jednotlivých znacích, dle kterých jsou vyzvednuty parametry, vždy dle aktuálního písmena, z tabulek definovaných jako globální proměnné *parameterPositions*, *parametricTable* a *segmentLength*.

Samotné generování hlasového výstupu probíhá za pomoci fonémového inventáře reprezentovaného globální proměnnou *phonemeTable*. V ní jsou uloženy posloupnosti stavů piezoelektrického měniče v segmentech. Segmenty jsou čteny po jednotlivých bitech. Hodnota přečteného bitu je následně vložena na výstupní pin *PF0*, čímž je piezoelektrický měnič nastaven do aktivního nebo neaktivního stavu. Tento stav je následně držen po určitou dobu pomocí funkce *Pause*.

Po průchodu všech bitů segmentu je provedena pauza pomocí funkce *Pause*. Následně je rozhodnuto, zda se má aktuální segment opakovat nebo je pokračováno jiným.

Takto je projit celý řetězec, který byl zadán. Pro podrobnější pochopení generovaní hlasového výstupu je vhodnější využít přímo přiložený projekt *B_Laboratorni_uloha*, ve kterém jsou jednotlivé fáze přímo popsány v konkrétním místě vykonávání. Pro případnou obsluhu knihovny je přiložen soubor *dokumentace*, ve kterém jsou popsány použití a parametry jednolitých funkcí.

2.2.2 Použití knihovny

Knihovna je k projektu připojena pomocí direktivy v souboru *main.c.* Zde je pomocí její funkce *USART_Init*, jejíž vypracování je jeden z bodů zadání, inicializována sériová komunikace.

Následně je volána hlavní funkce *Speak*, které je předáván řetězec, který by měl být obdržen pomocí realizované sériové komunikace. Dále ji je předávána konstantní hodnota určující její režim fungování. Ta je základně nastavena na hodnotu „r“, čímž je definováno, že aplikace bude generovat hlasový výstup pomocí piezoelektrického měniče.

Knihovna je již vhodně zakomponována do celého projektu a proto není potřeba nic měnit mimo dedikována místa.

2.3 Aplikace pro komunikaci mezi počítačem a vývojovou deskou Arduino Mega2560 Rev3

Pro komunikaci s vývojovou deskou Arduino Mega2560 Rev3 byla vytvořena aplikace v prostředí VisualStudio. Ta bude použita pro ověření funkčnosti vypracované laboratorní úlohy.

Po spuštění aplikace je po uživateli požadováno zadání názvu portu, na kterém je vývojová deska připojena. Pokud tato informace není známa, pak je ji možno vyčíst

pomocí aplikace „Správce zařízení“ v operačním systému Windows. Zde je v sekci „Porty“ tento údaj uveden.

Následně je inicializována sériová komunikace na straně počítače a je vypsán návod pro zadání vstupního řetězce. Dále je v případě, že není požadováno vytvoření WAV souboru, což se rozhoduje za pomoci definovaného uživatelského makra *MAKE_WAV*, požadováno zadání řetězce, dle kterého má být hlasový výstup generován.

Na řetězec jsou kladeny požadavky, které jsou uživateli vypsány po inicializaci sériové komunikace. Tyto požadavky je nutno dodržet pro správnou funkci. Jedná se o reprezentaci znaků s diakritikou. Písmena s čárkou musí být převedena do formy zápisu s apostrofem (např. písmeno [Á] musí být zapsáno jako řetězec [A‘]). Pro písmena s háčkem je definováno obdobné pravidlo, avšak háček je reprezentován znakem stříšky (např. písmeno [Č] musí být přepsáno jako řetězec [C^]).

Vstupní řetězec je poté odeslán pomocí sériové komunikace. Uživateli je pak umožněno zadání nového řetězce.

3. Praktická část

Pro realizaci laboratorní úlohy je přiložen projekt *A_Laboratorni_uloha*. Po spuštění projektu jsou v souboru *hlas.c* předdefinovány jednotlivé funkce, které je v rámci úlohy nutno vypracovat. Ty se nachází v horní části souboru. V ostatních přiložených souborech není potřeba nic měnit.

Pro ověření správnosti řešení je přiložen projekt *B_ComConsole*, díky které je možno zasílat vstupní řetězec. V projektu není potřeba nic měnit, stačí jej jen spustit. Pro další spuštění můžete použít již zkompilovanou verzi v automaticky vytvořené složce *Debug*.

3.1 Úkol č.1

Dle katalogového listu vypište do následující tabulky (viz tab. 3.1) jednotlivé názvy kontrolních registrů pro modul USART0. Dále vypište osmibitové hodnoty, na které je nutno nastavit dané registry pro inicializaci sériové komunikace s následujícími parametry:

- Povolení odesílání a přijímání dat
- Je povoleno přerušení pro dokončení přenosu dat
- Přenosová rychlost: 57600 bit/s (nezapomeňte, že tato hodnota musí být pro zadání do registru převedena dle odpovídajícího vzorce)
- Formát dat:
    - Jeden start bit
    - Osm datových bitů
    - Není generována parita
    - Jeden stop bit

Tabulka 3.1 Tabulka pro zápis názvu registrů a jejich hodnot pro odpovídající inicializaci sériové komunikace USART0

Název registru	Požadované hodnoty							

Vypište pomocí katalogového listu odpovídající vzorec pro převod hodnoty rychlosti přenosu pro případ asynchronního přenosu o normální rychlosti, která bude později zapsána do registru …… :

$$UBRRn =$$

Vypište do tabulky níže jednotlivé příznakové bity registru *UCSR0A* a jejich funkci:

Tabulka 3.2 Tabulka pro zápis názvu příznakových bitů a jejich funcí

Název bitu	Funkce příznakového bitu

3.2 Úkol č. 2

V připraveném projektu navrhněte funkci *USART_Init*. Ta bude inicializovat sériovou komunikaci USART0 s určenými parametry.

Před ukončením je volána funkce *USART_Receive*. Tu zde zanechejte. Při inicializaci sériové komunikace se mohou v datovém registru *UDR0* vyskytnout nevyžádaná data, především při resetu způsobeného odpojením napájení. Tato funkce zajišťuje, že po inicializaci bude datový registr vynulován.

Pro správnost zápisu jednotlivých parametrů můžete využít debugger, ve kterém lze vidět stavy jednotlivých bitů v daných registrech.

3.3 Úkol č. 3

V připraveném projektu navrhněte obsluhu přerušení při dokončení příjmu dat, tedy při nastavení odpovídajícího příznakového bitu. Generování je vyvoláno pomocí zjištěného vektoru přerušení.

Tato obsluha přerušení bude přijímat zasílaný řetězec. Jednotlivé znaky jsou vyzvedávány z datového registru. Obdržený Byte je potřeba uložit do pomocné proměnné.

Následně je nutno ověřit, zda se nejedná o indikaci konce řetězce a zároveň, že ukazatel na aktuální písmeno výsledného řetězce *receivedString_Ptr* nepřesáhl maximální hodnotu definovanou uživatelským makrem *BUFFER_SIZE*.

Jestliže se jedná o konec řetězce nebo byla přesažena maximální délka, pak je potřeba vložit do výsledného řetězce *receivedString* indikaci konce řetězce. Zároveň je nutno resetovat ukazatel na aktuální písmeno. Jako poslední je nutno nastavit indikaci na dokončení přijmu řetězce, což je realizováno pomocí proměnné *newString*, která je nastavena na hodnotu *true*.

V opačném případě je potřeba obdržený znak vložit do výsledného řetězce a inkrementovat hodnotu aktuální pozice v tomto řetězci.

3.4 Úkol č. 4

V připraveném projektu navrhněte implementaci definované funkce *USART_Transmit* sloužící k odesílání dat po sériové komunikaci.

Funkce bude čekat na indikaci, že je registr prázdný pomocí odpovídající vlajky. Následně budou do datového registru vloženy data k odeslání pomocí vstupního parametru *dataToSend*.

3.5 Úkol č. 5

V připraveném projektu navrhněte implementaci definované funkce *USART_Receive* sloužící k přijímání dat pomocí čekání na příznakový bit.

Funkce bude čekat, dokud nebudou obdrženy data v datovém registru pomocí odpovídajícího příznakového bitu. Následně bude obdržený Byte pomocí návratové hodnoty předán.

4. Ověření funkčnosti

Pokud byly všechny body zadání správně vypracovány, pak stačí do vývojové desky program nahrát.

Následně pomocí přiložené aplikace pro komunikaci s vývojovou deskou by po zadání řetězce v požadovaném formátu měl být generován hlasový výstup pomocí piezoelektrického měniče.

V případě, že není generován hlasový výstup, pak zde je souhrn základních parametrů, které toto chování mohou způsobovat:

- Piezoelektrický měnič není připojen na pin *PF0*
- Nastavená přenosová rychlost mezi systémy není totožná

- Nastavený formát dat není totožný
- Při inicializaci není volána funkce *USART_Receive*
- Jedna nebo více funkcí není realizována správně

Příloha B - Obsah přiloženého CD

A.1 Text práce

- Na přiloženém CD

A.2 Původní program „HLAS“

- Na přiloženém CD

A.3 Parametry programu „HLAS“

- Na přiloženém CD

A.4 Knihovna pro generování hlasového výstupu

- Na přiloženém CD

A.5 Základní aplikace využívající realizovanou knihovnu

- Na přiloženém CD

A.6 Parametry realizované knihovny

- Na přiloženém CD

A.7 Zadání laboratorní úlohy

- Na přiloženém CD