VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

PROGRAMOVATELNÝ GENERÁTOR SIGNÁLU PŘIPOJITELNÝ PŘES USB
PROGRAMMABLE SIGNAL GENERATOR CONNECTED VIA USB

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Lukáš Patočka
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Ivo Lattenberg, Ph.D.
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Lukáš Patočka ***ID:*** 78811
Ročník: 2 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Programovatelný generátor signálu připojitelný přes USB

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte problematiku generování signálu pomocí D/A převodníku. Realizujte programovatelný generátoru signálu. Použijte mikrokontrolér ATMEL. Zařízení bude napájené z USB portu a bude pracovat i při vypnutí počítače, za předpokladu, že port bude dále napájený. Data zvoleného průběhu budou uloženy do paměti. Bude k dispozici i několik předdefinovaných průběhů (sinusový, obdélníkový, trojúhelníkový či pilový). Při návrhu zohledněte to, že bude možno v ovládacím programu pro PC data průběhu editovat a do generátoru nahrát. Ovládací program bude sloužit také ke konfiguraci generátoru a bude zobrazovat aktuálně zvolené parametry (popis průběhu, kmitočet, amplituda, případně střída).

DOPORUČENÁ LITERATURA:

[1] BRTNÍK, B., MATOUŠEK, D. Mikroprocesorová technika BEN - technická literatura, Praha 2011, 152 stran, ISBN 978-80-7300-406-4.

[2] MATOUŠEK, D. Práce s mikrokontroléry Atmel AVR. Nakladatelství BEN - technická literatura, Praha 2006, 376 stran, ISBN 80-7300-209-4.

[3] VIRIUS, M., C# 2010 Hotová řešení, Computer Press, 2012, 424 s., ISBN 978-80-251-3730-7

Termín zadání: 1.2.2016 ***Termín odevzdání:*** 25.5.2016

Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.
Konzultant diplomové práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

Abstrakt:

Předmětem této práce je navrhnout a vytvořit prototyp generátoru periodického signálu. Generátor bude realizován pomocí mikrokontroléreu XMEGA128A4U firmy Atmel a bude řízen počítačovou aplikací vytvořenou v jazyce C#, zařízení bude s počítačem komunikovat po USB sběrnici.

Práce obsahuje všeobecné řešení problematiky generování signálů, konkrétně zaměřené na generování pomocí DA převodníku- tzv. digitální syntéze a rekonstrukci schodovitého signálu z výstupu DA převodníku na signál spojitý. Dále se v práci zaměřím na realizaci USB vrstvy na straně mikrokontroléru pomocí knihovny LUFA a na straně počítače pomocí knihovny LibUsbDotNet. Celkové zapojení práce bude dále obsahovat elektrický stejnosměrný měnič pro zvětšení maximální možné amplitudy signálu a součtový zesilovač pro přičtení stejnosměrné složky k výstupnímu signálu. Aplikace bude umožňovat generování signálu různých tvarů včetně uživatelem definovaného libovolného signálu. Tyto signály budou v aplikaci zobrazeny a vlastní signál si bude moci uživatel vytvořit pomocí grafického rozhraní, nebo načíst ze souboru.

Klíčová slova:

Generátor, mikrokontrolér, Atmel, AVR, USB, LUFA, LibUsbDotNet, digitálně analogový převodník (DA), operační zesilovač, rekonstrukční filtr

Abstract:

The subject of this thesis is design and construction of a periodic signal generator prototype. The generator will use Atmel XMEGA128A4U microcontroller with computer control managed via USB interface.

The thesis contains general solutions to the problem of generating signals with a special focus on utilization of DA converter – digital synthesis and reconstruction of the signal from DA converter output to continuous signal. The thesis further deals with implementation of the USB layer using two libraries (LUFA library on microcontroller side and LubUsbDotNet library on computer side). The final solution will include DC step-up converter for signal peak amplitude assessment and summing amplifier for adding the DC voltage to the output signal. The application will allow for generating signals of various shapes including user-defined ones. These signals will be displayed in the actual application window. There, users will be allowed to create user-defined signal in easy-to-use GUI or load it directly from a file.

Keywords:

Generator, microcontroller, Atmel, AVR, USB, LUFA, LibUsbDotNet, digital to analog converter (DA), operating amplifier, reconstruction filter

PATOČKA, L. *Programovatelný generátor signálu připojitelný přes USB.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 65 s. Vedoucí semestrální práce doc. Ing. Ivo Lattenberg, Ph.D..

Prohlášení

Prohlašuji, že svou diplomovou práci na téma „Programovatelný generátor signálu připojitelný přes USB“ jsem vypracoval samostatně pod vedením vedoucího semestrální 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 semestrální práce dále prohlašuji, že v souvislosti s vytvořením této semestrální práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona c. 121/2000 Sb., o právu autorském, o právech
souvisejících s právem autorským a o změně některých zákonu (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

podpis autora

Poděkování

Děkuji vedoucímu práce panu doc. Ing. Ivo Lattenbergovi, Ph.D za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne …….. .....

podpis autora

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

Obsah:

Seznam obrázků

Seznam tabulek

1 Úvod

Generování signálů je hojně řešená problematika v oblasti měření a zpracování nízkofrekvenčních i vysokofrekvenčních systémů. V dnešní době se nejčastěji používá systém generování pomocí DA převodníku- digitální syntéze. Tento způsob oproti jiným nabízí podstatně větší možnosti nastavení a širší rozsah parametrů generovaných signálů. Také nastavení těchto generátorů je podstatně snažší díky digitálnímu ovládání, které je možné realizovat i vzdáleně z aplikace, která je spuštěna na připojeném počítači.

Pro realizaci takovéhoto generátoru signálu byl zvolen mikrokontrolér vyrobený firmou Atmel, protože nabízí poměrně velké množství integrovaných periferií, mezi které patří DA převodníky a u některých modelů i USB rozhraní.

USB rozhraní je hojně využíváno už od roku 1998 a i dnes se jeho standardy nadále vyvíjejí, důkazem toho je nedávný příchod standardu USB 3.1. Toto rozhraní je v současnosti součástí každého počítače a také většiny mobilních zařízení.

Cílem této práce je realizovat prototyp softwarového vybavení mikrokontroléru a s ním komunikující aplikace pro PC, ověřit funkčnost USB komunikace a DA převodníku. Dále navrhnout schéma zapojení funkčního generátoru signálů nastavitelného pomocí USB rozhraní.

Výsledkem této práce tedy není funkční generátor signálu, ale jedná se o přípravu realizace, která bude uskutečněna v rámci diplomové práce.

Celá práce bude rozdělena do několika kapitol. Bude obsahovat teoretický rozbor s informacemi o technologiích potřebných pro vlastní realizaci, část věnující se aplikaci mikrokontroléru a část pro aplikaci v počítači, obě tyto části budou obsahovat popis realizace USB rozhraní. Poslední částí práce bude návrh schématu zapojení a desky plošného spoje generátoru signálu.

2 Teoretický rozbor

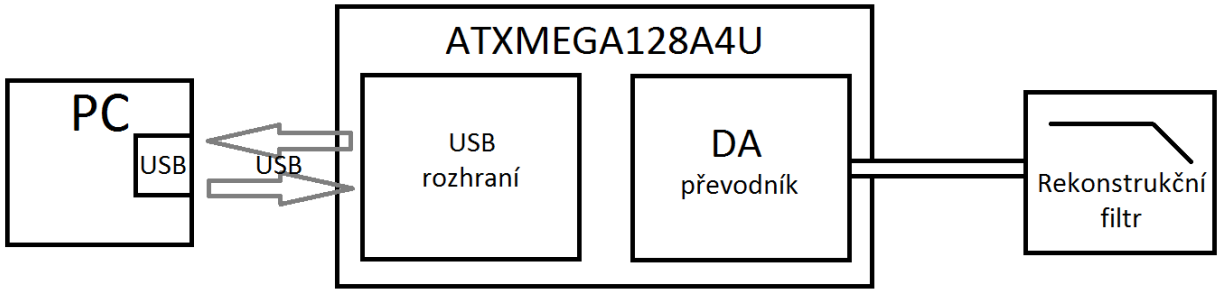
V této části diplomové práce budou rozebrány možnosti realizace laditelného generátoru periodického signálu, volba konkrétního řešení (v tomto případě pomocí DA převodníku mikrokontroléru Atmel), možnosti komunikace mikrokontroléru s počítačem pomocí USB rozhraní a návrh rekonstrukčního filtru DA převodníku.

2.1 Možnosti generování periodického signálu

Analogové oscilátory – základním prvkem těchto zařízení bývá jednoduchý obvod s RC nebo LC článkem. Frekvenci takovýchto oscilátorů je možné měnit změnou kapacity článku buď mechanicky, nebo například pomocí kapacitní diody (varikapu). Toto řešení má ale velmi omezený frekvenční rozsah. Pro jeho zvětšení je třeba využít relativně složité mechanické konstrukce, případně přepínání rozsahů. Oscilátory generují z pravidla sinusový signál. Pro generování signálu jiného tvaru je nutné využít speciální obvod pro úpravu tvaru – tvarovač. S jeho využitím je možné generovat signál obdélníkový nebo pilový.

Digitální generátory – tyto generátory využívají referenční zdroj signálu s konstantní frekvencí a pomocí digitálních úprav generují výstupní signál jako násobek nebo podíl signálu referenčního. Změna frekvence i amplitudy je prováděna skokově, přičemž velikost kroku je dána parametry DA převodníku. Výstupem DA převodníku je signál ve schodovitém nebo impulsním tvaru, který je dále vyhlazen rekonstrukčním filtrem typu dolní propust. Digitální generátory poskytují velmi široký frekvenční rozsah a vysokou frekvenční stabilitu.

2.2 Blokové schéma realizovaného generátoru



Obr. 1 - Blokové schéma generátoru

1. Mikrokontrolér ATXMEGA128A4U
   - USB rozhraní mikrokontroléru
   - integrovaný DA převodník
2. Počítač s USB rozhraním a aplikací k řízení mikrokontroléru a DA převodníku
3. Výstupní rekonstrukční filtr

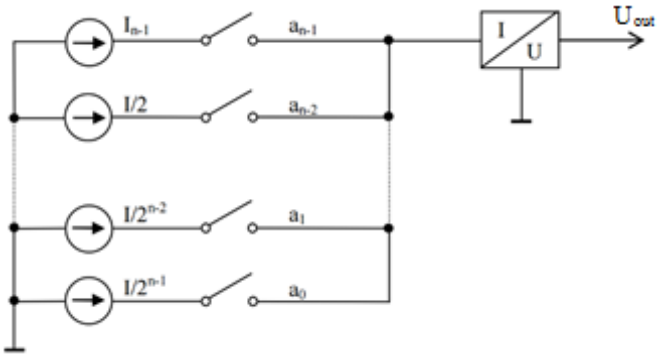
2.3 Principy DA převodníku

[3] [4] Realizace DA převodu spočívá v přiřazení analogové veličiny n-bitovému číslu 2^n. Využívá se rozkladu celého čísla na jednotlivé bity, které mají danou váhu podle svého pořadí a číslo je tak vyjádřeno jejich sumou:

$$x = \sum_{i=0}^{n-1} a_i \cdot 2^i$$

Jednoduchý DA převodník vytvoříme tak, že nahradíme číselnou váhu jednotlivých bitů odpovídajícím proudovým zdrojem doplněným o spínač ovládaný příslušným bitem a proudy sečteme. Takovýto DA převodník ještě bývá obvykle doplněn o převod proudu na napětí. Výpočet takového převodníku je uveden v následující rovnici a princip je zobrazen na obrázku Obr. 2.

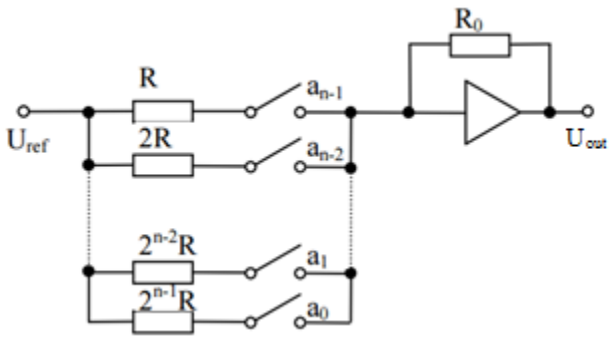
$$I_{out} = \sum_{i=0}^{n-1} a_i \frac{I}{2^{n-(i+1)}}$$



Obr. 2 - Princip DA převodníku

Toto zapojení není pro praktické využití vhodné. DA převodník bývá realizován zdrojem referenčního napětí, soustavou váhových rezistorů se spínači a součtovým zesilovačem. Zapojení je na obrázku Obr. 3. Vzorec výpočtu:

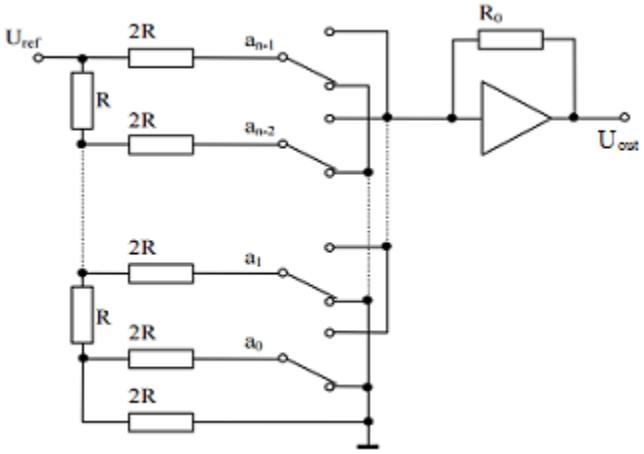
$$U_{out} = -U_{ref} \cdot \frac{R_0}{2^{n-1} \cdot R} \cdot \sum_{i=0}^{n-1} a_i \cdot 2^i$$



Obr. 3 - Realizace DA převodníku

Nevýhodou tohoto zapojení je velký rozsah váhových rezistorů, například pro převodník s bitovou hloubkou 12bitů je poměr 1:4096, v takovémto rozsahu je velmi náročné dodržet požadovanou přesnost použitých součástek. Proto se v praxi nejčastěji používá zapojení DA převodníku s rezistorovou žebříčkovou sítí R-2R (Obr. 4.), vzorec pro výpočet je uveden níže:

$$U_{out} = -U_{ref} \cdot \frac{R_0}{2^n \cdot R} \cdot \sum_{i=0}^{n-1} a_i \cdot 2^i$$



Obr. 4 – Realizace DA převodníku s rezistorovou sítí

2.4 Klíčové vlastnosti DA převodníků

- **Bitová hloubka**
  Udává počet diskrétních úrovní výstupního napětí. Závisí na počtu bitů DA převodníku podle vztahu $N_u = 2^n$, např. 10 bitový převodník nabízí 1024 úrovní výstupního napětí.

- **Rychlost (frekvence) převodu**
  Je to počet převedených binárních slov za jednotku času. Může se také udávat jako maximální frekvence, na které je schopný DA převodník pracovat.

- **Napěťový rozsah**
  Maximální napěťový rozkmit převodníku.

2.5 Volba mikrokontroléru

Pro realizaci diplomové práce byl zvolen mikrokontrolér značky Atmel řady XMEGA. Tyto mikrokontroléry obsahují velké množství integrovaných periferních zařízení, mezi které patří DA a AD převodníky, časovače. Některé typy řady XMEGA mají integrováno i USB rozhraní. Konkrétně jsem si zvolil typ Atmel XMEGA128A4U, níže jsou uvedeny některé parametry tohoto mikrokontroléru:

Flash paměť:	128 KB
EEPROM paměť:	2 KB
SRAM paměť:	8 KB
Pracovní frekvence:	32 MHz
Počet bitů CPU:	8 b
Počet vstupů/výstupů:	34
Typ USB:	Full speed device
Počet kanálů DA převodníku:	2
Bitová hloubka DA převodníku:	12
Maximální rychlost DA převodníku:	1 Msps

Tabulka 1 - Vlastnosti mikrokontroléru XMEGA128A4U

2.6 USB rozhraní

[10] K ovládání a nastavování generátoru je nutné realizovat komunikaci mezi mikrokontrolérem a počítačem pomocí USB rozhraní. USB je standard umožňující rozšíření PC architektury o velké množství periferních zařízení. Vyznačuje se především snadným připojením periferních zařízení (metoda plug and play), vysokou přenosovou rychlostí a možností přenosu obrazu a zvuku v reálném čase.

Verze	USB standardy	Přenosová rychlost
USB 1.1	low speed full speed	1,5 Mb/s 12 Mb/s
USB 2.0	high speed	480 Mb/s
USB 3.0	super speed	4800 Mb/s
USB 3.1		až 10 Gb/s

Tabulka 2 - Přehled USB standardů

Typy USB přenosů:

- **Kontrolní přenosy (control transfers)**
  Tyto přenosy využívá především systémový software pro konfiguraci zařízení při jeho prvotním připojení. Kontrolní přenosy mohou být využity i uživatelským softwarem specifickým způsobem. Spolehlivost předávání dat kontrolními přenosy je zajištěna potvrzováním.
- **Přenosy přerušení (interrupt transfers)**
  Přenosy s garantovaným časem doručení, využívají se hlavně ke zpracování událostí. Přenášená zpráva obsahuje informaci o typu události a malý blok dat. Využívají se například u polohovacích zařízení.

- **Hromadné přenosy (bulk transfers)**
  Přenášená data se obvykle skládají z větších bloků dat, využívají se například pro tiskárny a skenery. Data jsou přenášena sekvenčně, spolehlivost přenosu dat je zajištěna na hardwarové úrovni pomocí detekce chyb a vyvolání opakovaných odeslání.

- **Izochronní přenosy (isochronous transfer)**
  Izochronní data jsou přenášena nepřetržitě, spojitě po celou dobu přenosu. Jsou to data, u kterých je požadována maximální přenosová rychlost, ale není zaručena jejich bezchybnost. Využívají se například pro telefonní hovor, přenos obrazu nebo zvuku. Izochronní přenosy probíhají na vyhrazené části přenosového pásma a díky tomu jsou data přenášena konstantní rychlostí.

Pro realizaci USB vrstvy na straně mikrokontroléru byla zvolena open source knihovna LUFA. Aplikace na straně počítače bude vytvořena v jazyce C# a frameworku .NET, proto byla jako knihovna pro podporu USB rozhraní na straně počítače zvolena knihovna LibUsbDotNet.

2.6.1 Knihovna LUFA (Lightweight USB Framework for AVRs)

[6] Knihovna LUFA je kompletní open-source framework pro podporu 8bitových a 32bitových mikrokontrolérů AVR s USB rozhraním. Je šiřitelná pod licencí MID, tato licence zaručuje její volné využití i šíření. Framework LUFA obsahuje vlastní bootloader pro mikrokontroléry AVR a několik demo projektů. Programy napsané s využitím knihovny LUFA by měly být přenositelné mezi všemi mikrokontroléry Atmel AVR.

2.6.2 Knihovna LibUsbDotNet

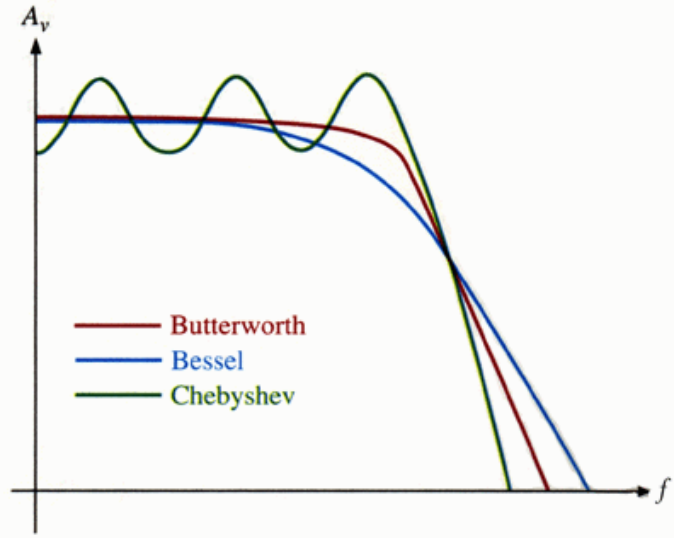
[7] LibUsbDotNet je open-source knihovna jazyka C#, která byla vytvořena jako nadstavba Win32 knihovny WinUsb pro jazyk C#, je šiřitelná pod licencí GNU. Knihovna podporuje všechny základní funkce a typy přenosů USB rozhraní na platformách unixového typu a Windows.

2.7 Rekonstrukční filtr

Na výstupu DA převodníku je generován signál schodovitého tvaru, to znamená, že na výstupu převodníku je konstantní úroveň napětí až do příchodu dalšího vzorku. Tento signál tedy neodpovídá námi požadovanému signálu. [5] Ideální rekonstrukci signálu umožňuje ideální filtr typu dolní propust, pokud je mezní frekvence tohoto filtru menší nebo rovna polovině vzorkovací frekvence a jestliže byl při vzorkování dodržen vzorkovací teorém. V případě signálového generátoru to znamená, že generovaná frekvence nesmí být vyšší než polovina frekvence vzorkovací.

2.7.1 Dělení filtrů

- Podle funkce filtru
  - Dolní propust
  - Horní propust
  - Pásmová propust
  - Pásmová zádrž
- Podle realizace filtru
  - Číslicové filtry
  - Analogové filtry
    - pasivní filtry (LC / RLC)
    - aktivní filtry (RC)
- Podle charakteristiky filtru
  - Butterworthova aproximace
  - Besselova aproximace
  - Chebychevova aproximace
  - …



Obr. 5 – Příklad charakteristik filtrů

Dalším významným parametrem je řád filtru, který ovlivňuje strmost klesání přenosové charakteristiky v nepropustném směru. Filtr prvního řádu má strmost klesání 6 dB/oct (20dB/dec), dále například filtr čtvrtého řádu má strmost 24 dB/oct (80 dB/dec).

2.7.2 Volba rekonstrukčního filtru

Jako rekonstrukční filtr je vhodné použít analogový filtr typu dolní propust (číslicový filtr není možné použít, protože jeho výstupem je opět číslicová posloupnost). Filtr by měl mít pokud možno plochou charakteristiku v propustném pásmu a co nejstrmější klesání v nepropustném pásmu. Dále je výhodnější použít aktivní filtr typu LC, protože má menší útlum v propustném pásmu. Nejpříhodnější se tedy jeví Butterworthova aproximace filtru vyššího řádu, protože má nejpřímější charakteristiku v propustném pásmu a nejstrmější klesání za dělícím frekvencí.

3 Řídící PC aplikace

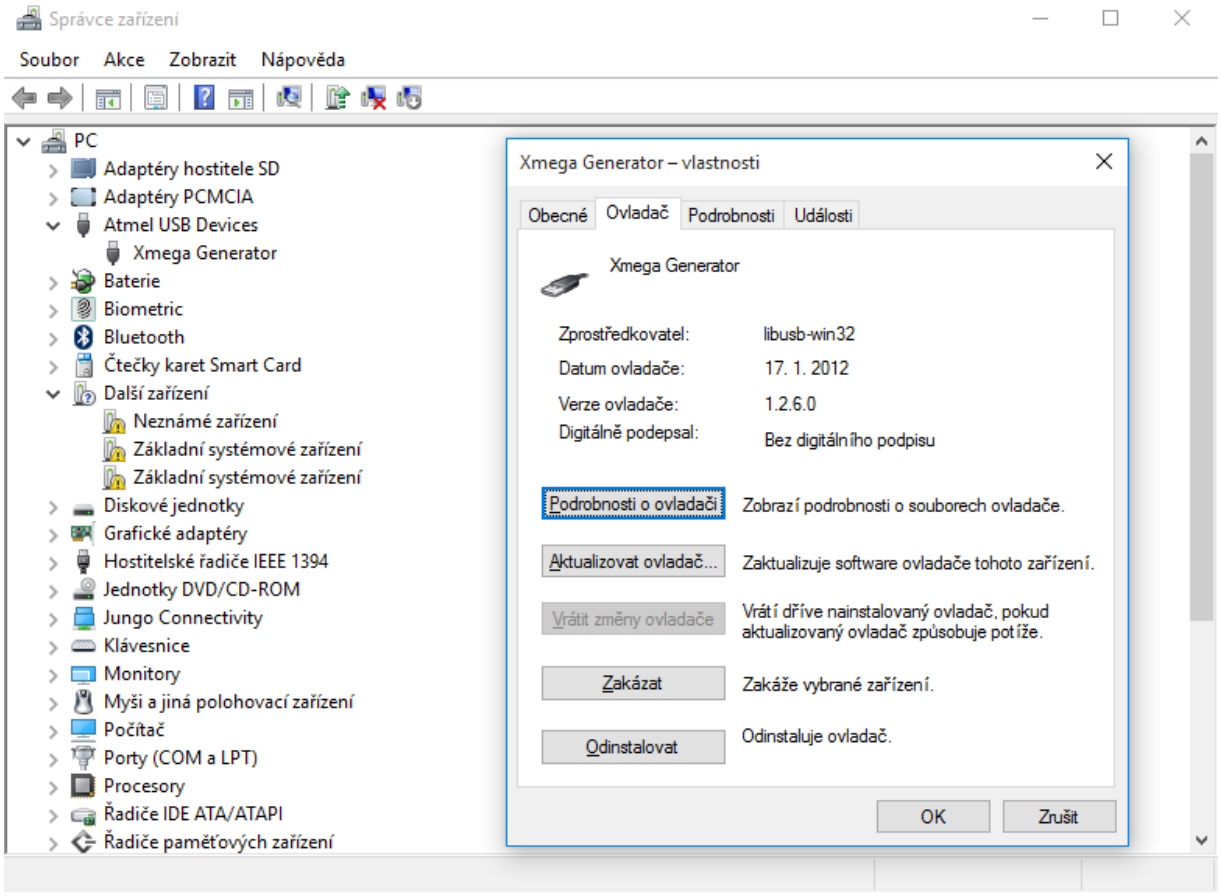
Úlohou PC aplikace bude řízení mikrokontroléru a nahrávání dat pro generování periodických signálů. Po navázání USB spojení budou z mikrokontroléru načteny parametry DA převodníku. Následně aplikace vygeneruje frekvenční kroky podle vzorkovací frekvence. Dále bude možné nastavit parametry průběhu (frekvenci, amplitudu, tvar signálu a stejnosměrnou složku), odeslat je mikrokontroléru a zahájit vlastní generování signálu, aplikace také zobrazuje tvar aktuálně nastaveného signálu, toto zobrazení je pouze ilustrační a nemusí odpovídat skutečně generovanému signálu.

[6][7] Aplikace je vytvořena v prostředí Microsoft Visual studio 2015 v jazyce C# s pomocí frameworku .NET verze 4.5 a s využitím výše uvedené knihovny LibUsbDotNet.

Aplikace je rozdělena do tří celků- vlastního uživatelského rozhraní a dvou knihoven. První knihovna se stará o veškerou komunikaci s USB zařízením pomocí knihovny LibUsbDotNet, druhá provádí výpočty frekvencí a stará se o generování signálů různých tvarů.

3.1 Inicializace a navázání USB komunikace

K zajištění správné funkce každého periferního zařízení je třeba v operačním systému nainstalovat příslušný ovladač. Mikrokontrolér Atmel Xmega128A4U je podporován ovladačem LibUsb-Win32 verze 1.2.6.0. Na rozdíl od nižších řad mikrokontrolérů Atmel nevytváří mikrokontroléry řady Xmega virtuální sériový port, ale hlásí se přímo jako USB zařízení. Díky tomu je možné v aplikaci reagovat na události vzniklé na USB sběrnici a s využitím technologie plug and play zobrazovat aktuálně připojená zařízení. Při připojení nebo odpojení libovolného USB zařízení vyvolává knihovna událost, díky které je možné například korektně ukončit komunikaci s právě používaným zařízením a předejít tak nekorektnímu ukončení aplikace.



Obr. 6 - Ovladač Atmel USB zařízení

3.1.1 Sledování USB událostí a seznam zařízení

[7] Knihovna LibUsbDotNet nabízí třídu *DeviceNotifier*, která umožňuje sledování událostí vzniklých na USB sběrnici, jako je například připojení a odpojení nového zařízení. V okamžik vzniku některé z USB událostí třída oznamuje tuto skutečnost událostí (event) *OnDeviceNotify*. Díky tomu je možné mít stále přehled o aktuálně připojených zařízeních.

```
UsbDeviceNotifier = DeviceNotifier.OpenDeviceNotifier();
UsbDeviceNotifier.OnDeviceNotify += OnUsbChange;
```

Pokud je třeba získat seznam všech USB zařízení, nabízí knihovna LibUsbDotNet ve třídě *UsbDevice* seznam objektů typu *UsbRegistry - AllDevices*. Z *AllDevices* je možné vyčíst veškeré potřebné informace k identifikaci USB zařízení, jako je *Name*, *FullName*, *SymbolicName*, *Vid* a *Pid*, podle kterých je možné určit, jestli je zařízení vhodné ke komunikaci s aplikací. Na základě

těchto informací aplikace rozdělí zařízení do dvou seznamů- známých a neznámých zařízení. Ze seznamu známých zařízení je možné vybrat to, které bude aplikací otevřeno a bude s ním nadále spolupracovat.

```
foreach (UsbRegistry regDev in UsbDevice.AllDevices)
{
	if (regDev.Vid == 0x03EB && (regDev.Pid == 0x2065 || regDev.Pid == 0x2066))
	{
		this.Devices.Add(new USB_Device(regDev));
		this.OtherDevices.Add(regDev.SymbolicName, regDev.FullName);
	}
	else
	{
		this.OtherDevices.Add(regDev.SymbolicName, regDev.FullName);
	}
}
```

3.1.2 Připojení k USB zařízení

Po volbě vhodného zařízení je třeba zařízení otevřít a připojit se k němu. Prvním krokem je otevření zařízení, k tomu slouží metoda *Open* třídy *UsbRegistry* s výstupním parametrem *USBDevice*. Dále je třeba zvolit konfiguraci zařízení metodou *SetConfiguration* a připojit se k rozhraní USB zařízení metodou *ClaimInterface*. Po provedení těchto operací je USB zařízení připraveno realizaci komunikace. Pozn. uvedená ukázka neobsahuje ošetření chyb a ostatní kód.

```
UsbReg.Open(out thisDevice);
IUsbDevice wholeUSBdev = thisDevice as IUsbDevice;
wholeUSBdev.SetConfiguration(1);
wholeUSBdev.ClaimInterface(0);
```

3.2 USB komunikace

Pro funkci mikrokontroléru jako generátoru signálů budou třeba dva druhy USB komunikace a to řídící signály, realizované pomocí kontrolních přenosů s uživatelsky definovaným významem. Řídících signálů nebude mnoho a budou přenášet jen malé objemy dat (jednotky bytů), proto není třeba využívat přenosy přerušení. Dále budou využity hromadné přenosy k transportu signálu formou bufferu vzorků tohoto signálu, buffer může mít objem od jednotek až po stovky bytů.

3.2.1 Řídící přenosy- načítání dat

Pro případ použití více typů mikrokontrolérů ve funkci generátoru signálu jsou parametry DA převodníku aplikací načítány přímo z mikrokontroléru. Jedná se o bitovou hloubku DA převodníku, samplovací frekvenci, maximální dělící poměr kmitočtu a velikost paměťového bufferu v nejbližší vyšší paměťové jednotce (pro 8 bitů je to byte, pro 12 bitů je to word).

Nejprve je třeba vytvořit nastavení přenosu, v knihovně LibUsbDotNet k tomu slouží třída UsbSetupPacket, obsahující následující informace:

- typ přenosu (směr přenosu dat, typ a vlastnosti přenosu)
- identifikátor zprávy
- hodnota
- index
- délka

Po nastavení parametrů přenosu je možné realizovat vlastní přenos voláním metody *ControlTransfer* třídy *UsbDevice*. Jako parametry je třeba zadat výše vytvořený *UsbSetupPacket*, výstupní buffer, velikost bufferu a jako výstupní parametr délka přenesených dat.

```
UsbSetupPacket setup = new UsbSetupPacket((byte)
(REQDIR_DEVICETOHOST | REQTYPE_VENDOR | REQREC_DEVICE), GET_DA_FREQ, 0, 0, 4);

byte[] buffer = new byte[4];
int len = 0;
while(!thisDevice.ControlTransfer(ref setup, buffer, 4, out len));

frequency = BitConverter.ToUInt32(buffer, 0);
```

3.2.2 Řídící přenosy- odesílání dat

Před samotným přenosem bufferu vzorků požadovaného signálu jsou pomocí řídících signálů přeneseny parametry signálu- délka bufferu, dělící poměr vzorkovacího kmitočtu a hodnota stejnosměrné složky signálu.

Přenos řídícího signálu ve směru do kontroléru se provádí voláním stejné metody, pouze s jinými parametry, protože koncový bod pro řídící parametry je na rozdíl od ostatních koncových bodů označován číslem nula pro oba směry. Data určená pro odeslání do mikrokontroléru jsou uložena v proměnné buffer. Mezi jednotlivými přenosy je ponechána prodleva 100ms kvůli zpracování dat na straně mikrokontroléru, tento problém bude popsán v kapitole věnované softwaru mikrokontroléru.

```
public bool SetOffset(UInt16 offset)
{
    UsbSetupPacket setup = new UsbSetupPacket((byte)
    (REQDIR_HOSTTODEVICE | REQTYPE_VENDOR | REQREC_DEVICE), SET_DCOFFSET, 0, 0, 2);
    int len = 0;
    return thisDevice.ControlTransfer(ref setup, offset, 2, out len);
}
```

3.2.3 Bulk přenos- odesílání bloku dat

Pro odeslání bloku dat do mikrokontroléru je využita asynchronní varianta USB bulk přenosu. Asynchronní přenos umožňuje odesílání dat do mikrokontroléru bez pozastavení běhu aplikace, po dobu probíhajícího přenosu. Bulk přenos je uskutečněn v několika krocích. Nejprve je třeba inicializovat takzvaný UsbEndpointWriter, jehož pomocí je přenos realizován. Parametrem jeho konstruktoru je číslo požadovaného endpointu. Následuje zahájení asynchronního přenosu voláním metody SubmitAsyncTransfer. Data určená k odeslání jsou metodě předána jako parametr, společně s časovým limitem pro přenos a kontextem přenosu. Následuje vyčkávání na dokončení přenosu metodou WaitAll třídy WaitHandle a na závěr jsou uvolněny prostředky potřebné k přenosu voláním metody Dispose.

```
UsbTransfer usbwt;
int wtransfered;

UsbEndpointWriter usbw = thisDevice.OpenEndpointWriter(WriteEndpointID.Ep03);

ErrorCode ecw = usbw.SubmitAsyncTransfer(buffer, 0, buffer.Length * sizeof(UInt16),
5000, out usbwt);

WaitHandle.WaitAll(new WaitHandle[] { usbwt.AsyncWaitHandle }, 5000, false);

if (!usbwt.IsCompleted)
   usbwt.Cancel();

ecw = usbwt.Wait(out wtransfered);

usbwt.Dispose();
```

3.3 Výpočet frekvenčních kroků

Po navázání komunikace a vyčtení všech potřebných informací o DA převodníku, bude dalším krokem vytvoření tabulky frekvencí, které je převodník schopen generovat. Aby generátor mohl generovat signál o přesné frekvenci, musí být daná frekvence určitým podílem frekvence samplovací. Dále musí být splněna podmínka pro rekonstrukci signálu, to znamená, že generovaná frekvence musí být minimálně poloviční oproti frekvenci samplovací. Pro každý frekvenční krok je třeba znát tři parametry- kromě vlastní frekvence také počet vzorků na jednu periodu signálu a hodnotu děliče samplovací frekvence. Všechny tyto parametry jsou uloženy v seznamu (Listu) objektů třídy *ToneTable*. Definice třídy je uvedena níže.

```
public class ToneTable
{
    public ToneTable(double freq, uint samples, uint devider, uint multipler)
    {
        this.Freq = freq;
        this.SamplesPerT = samples;
        this.Devider = devider;
        this.Multipler = multipler;
    }
    public double Freq { get; private set; }
    public uint SamplesPerT { get; private set; }
    public uint Devider { get; private set; }
    public uint Multipler { get; private set; }
}
```

Pro výpočet frekvenčních kroků jsem vytvořil metodu, která dělí základní- samplovací kmitočet počtem vzorků na jednu periodu, délka bufferu pro vzorek signálu bude tedy proměnná. Pro splnění podmínky rekonstrukce signálu začíná výpočet od pěti vzorků a zvyšuje se až po maximální velikost bufferu. Tímto způsobem by nejnižší možná frekvence, kterou by bylo možné generovat, byla rovna samplovací frekvenci lomeno maximálním počtem vzorků. V mém případě je to tedy pro samplovací frekvenci 160 kHz a velikost bufferu 512 vzorků by byla nejnižší generovatelná frekvence 312,5 Hz, což je pro praktické využití nepoužitelné. Pro rozšíření šířky pásma na nejnižší frekvenci byla do výpočtu přidána proměnná- dělitel samplovací frekvence. Když výpočet dosáhne maxima bufferu, nastaví se počet vzorků na poloviční hodnotu a dělitel samplovací frekvence na dvojnásobek, takto výpočet pokračuje až do dosažení limitu dělitele samplovací frekvence. Protože ve spodní části tabulky frekvencí byl krok mezi jednotlivými frekvencemi zbytečně malý, bylo do výpočtu přidáno zvětšení kroku na nižších frekvencích. Při dosažení maxima bufferu pokračuje další cyklus s krokem vždy o jedna větším. Příklad výpočtu:

samplovací frekvence = 160 kHz
velikost bufferu = 512 vzoků
max. dělitel vzorkovací frekvence = 256

Pomocí uvedeného výpočtu bylo vytvořeno 981 frekvenčních kroků, frekvence nejnižšího kroku je 1,23 Hz, dělič vzorkovací frekvence 256 a počet vzorků na periodu je 509. Rozdíl mezi prvním a druhým krokem je 0,02 Hz. Frekvence posledního kroku je 40 kHz bez dělení samplovací frekvence a perioda obsahuje 5 vzorků signálu. Poslední frekvenční krok je 8 kHz.

Výrobce mikrokontroléru uvádí až 1 MHz vzorkovací frekvenci DA převodníku, ale při tomto nastavení již byla omezena funkčnost USB rozhraní kvůli vyvolávání přerušení od časovače DA převodníku mikrokontroléru. Aby byla zajištěna bezproblémová funkce mikrokontroléru byla zvolena vzorkovací frekvence 160 kHz.

```
public static List<ToneTable> GenerateTones
(uint sampleRate, uint memoryCount, uint maxfDev)
{
    List<ToneTable> tones = new List<ToneTable>();
    int j = 5;
    int dev = 1;
    for (int f = 1; dev <= maxfDev; f++)
    {
        for (; j <= memoryCount; j += f)
        {
            tones.Insert(0, new ToneTable((double)sampleRate / j, j, dev));
        }
        sampleRate /= 2;
        dev *= 2;
        j /= 2;
    }
return tones;
}
```

3.3.1 Úprava pro zmenšení frekvenčních kroků sinusového signálu

Dle výše uvedeného příkladu výpočtu frekvenčních kroků je vidět, že rozdíl mezi dvěma kroky na nejvyšší frekvenci je poměrně velký- 8 kHz. Pro zmenšení frekvenčních kroků na vyšších frekvencích byl algoritmus upraven. Úprava využívá toho, že u sinusového signál nemusí být délka periody shodná s délkou bufferu. Do výpočtu jsou tedy zařazeny i frekvence signálů, kde je do jednoho bufferu vzorků vloženo více period signálu. S ohledem na dodržení pravidla pro rekonstrukci signálu jsou přidávány pouze frekvence, kde jsou minimálně čtyři vzorky na periodu. Po úpravě vypadá algoritmus výpočtu frekvenčních kroků následovně:

```
for (uint f = 1; dev <= maxfDev; f++) {
  for (; j <= memoryCount; j += f)
  {
    tones.Insert(0, new ToneTable((double)sampleRate / j, j, dev, 1));
    sineTones.Insert(0, new ToneTable((double)sampleRate / j, j, dev, 1));
    if (dev == 1)
    {
      double j2 = j; uint mult = 2;
      while ((j2 / mult) >= 4)
      {
       sineTones.Insert(0, new ToneTable((double)sampleRate / j * mult, j, dev, mult));
       mult++;
      }
    }
  }
  sampleRate /= 2;
  dev *= 2;
  j /= 2;
}
```

Díky výpočtu tímto způsobem bylo vypočteno 33112 frekvenčních kroků při stejné šířce pásma. Jistou nevýhodou tohoto algoritmu je vznik duplicitních frekvencí a to, že nejsou do pole vkládána postupně, jak tomu bylo před úpravou. Před vlastním generováním je tedy ještě nutné hodnoty frekvencí v seznamu seřadit a vyfiltrovat duplicity.

```
sineTones.Sort((x, y) => x.Freq.CompareTo(y.Freq));

for(int i = 1; i < sineTones.Count; i++)
{
    if (Math.Abs(sineTones[i - 1].Freq - sineTones[i].Freq) < 0.05)
    {
        if (sineTones[i - 1].Multipler <= sineTones[i].Multipler)
            sineTones.RemoveAt(i--);
        else
            sineTones.RemoveAt(--i);
    }
}
```

Po odstranění vzorků s rozdílem menším než je 0,05 Hz zůstalo 20419 frekvenčních kroků s nepravidelným krokem. Například mezi posledním a předposledním vzorkem je rozdíl 78,58 Hz, ale rozdíl mezi následujícími kroky je pouze 0,62 Hz. Tento rozdíl je dán tím, že počet period, které je možné vložit do bufferu, je omezen minimální délkou periody na 4 vzorky.

3.4 Výpočet vzorků signálu

Se znalostí amplitudy, vzorkovací frekvence a jednotlivých frekvencí, které je schopen generátor generovat, je možné začít s výpočty konkrétních vzorků. Dle zadání jsou realizovány výpočty pro následující průběhy:

3.4.1 Sinusový

Výpočet jednotlivých vzorků pro sinusový průběh signálu probíhá v cyklu tolikrát, kolik daný frekvenční krok obsahuje vzorků, k výpočtu je využita knihovní funkce *Math.Sin*. Rovnice a výpočtu a ukázka kódu:

$$u = U \cdot \sin\left(\frac{2\pi \cdot i \cdot f \cdot d}{f_s}\right)$$

u – okamžitá hodnota vzorku
U – amplituda
i – pořadí vzorku
f – generovaná frekvence
d – dělitel vzorkovací frekvence
fs – samplovací frekvence

```
public void GenerateSine(int index, int amplitude, int sampleFreq)
{
    Sequnce.Clear();
    for (int i = 0; i < Tones[index].SamplesPerT; i++)
    {
        Sequnce.Add(amplitude * Math.Sin(2 * Math.PI * i * Tones[index].Freq /
        (sampleFreq / Tones[index].Devider)));
    }
}
```

3.4.2 Pilový

Rovnice výpočtu a ukázka kódu:

$$u = \frac{U \cdot 2i}{n} - U$$

u – okamžitá hodnota vzorku
U – amplituda
i – pořadí vzorku
n – počet vzorků na periodu

```
public void GenerateSaw(int index, int amplitude, int sampleFreq)
{
    Sequnce.Clear();
    for (int i = 0; i < Tones[index].SamplesPerT; i++)
    {
        Sequnce.Add(amplitude * i * 2 / Tones[index].SamplesPerT - amplitude);
    }
}
```

3.4.3 Obdélníkový

Rovnice výpočtu a ukázka kódu:

$$u_{0\sim\frac{1}{2}} = U$$

$$u_{\frac{1}{2}\sim 1} = -U$$

u – okamžitá hodnota vzorku
U – amplituda

```
public void GenerateRectangle(int index, int amplitude, int sampleFreq)
{
    Sequnce.Clear();
    for (int i = 0; i < Tones[index].SamplesPerT / 2; i++)
    {
        Sequnce.Add(amplitude);
    }
    for (int i = 0; i < Tones[index].SamplesPerT / 2; i++)
    {
        Sequnce.Add(-amplitude);
    }
}
```

3.4.4 Trojúhelníkový

Rovnice výpočtu a ukázka kódu:

$$u_{0\sim\frac{1}{4}} = \frac{U \cdot 4i}{n}$$

$$u_{\frac{1}{4}\sim\frac{3}{4}} = U - \frac{(U \cdot 4i)}{n}$$

$$u_{\frac{3}{4}\sim 1} = \frac{U \cdot 4i}{n} - U$$

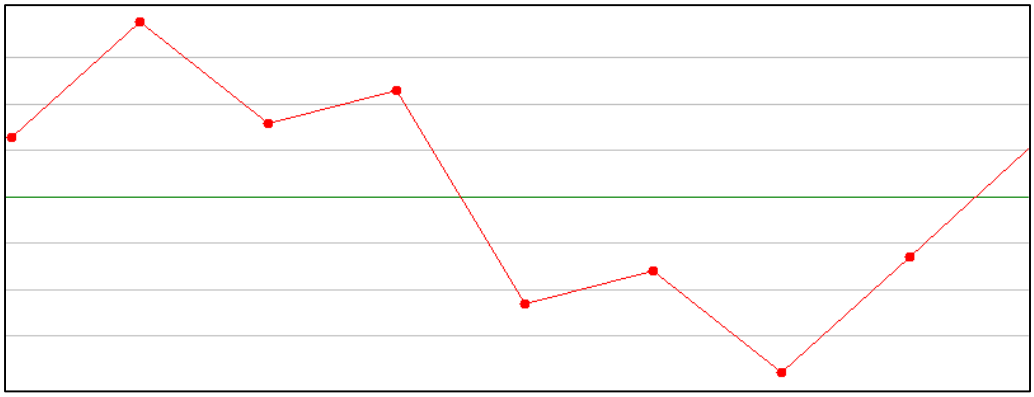
u – okamžitá hodnota vzorku
U – amplituda
i – pořadí vzorku
n – počet vzorků na periodu

```
public void GenerateTriangle(int index, int amplitude, int sampleFreq)
{
    Sequnce.Clear();

    for (int i = 0; i < Tones[index].SamplesPerT / 4; i++)
    {
        Sequnce.Add(amplitude * i * 4 / Tones[index].SamplesPerT);
    }
    for (int i = 0; i < Tones[index].SamplesPerT / 2; i++)
    {
        Sequnce.Add(amplitude - amplitude * i * 4 / Tones[index].SamplesPerT);
    }
    for (int i = 0; i < Tones[index].SamplesPerT / 4; i++)
    {
        Sequnce.Add(amplitude * i * 4 / Tones[index].SamplesPerT - amplitude);
    }
}
```

3.5 Generování signálu libovolného tvaru

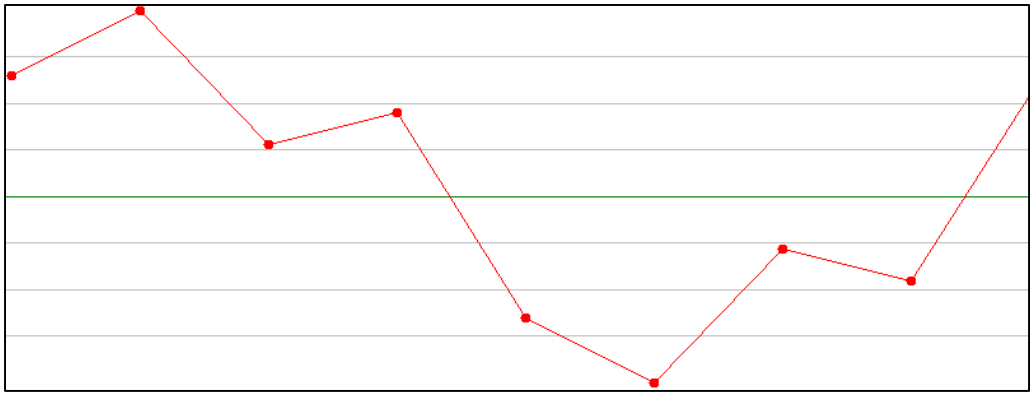
Pro usnadnění definice tvaru uživatelsky definovaného signálu byl vytvořen panel s pohyblivými body, které určují tvar požadovaného signálu. Počet těchto bodů si může uživatel nastavit v rozsahu 4 až 40 bodů. Pro usnadnění tvarování vlastního signálu jsem vytvořil dva režimy zrcadlení signálu- při editaci jedné poloviny signálu se posun daného bodu promítá do druhé poloviny v opačné polaritě. První režim nazvaný „Mirror mode 1“ upravuje body středově symetricky, to znamená že, úprava prvního bodu změní polohu posledního bodu. Druhý režim „Mirror mode 2“ při změně pozice jednoho bodu ovlivňuje polohu bodu od něj o polovinu periody vzdáleného. S využitím těchto režimů je možné vytvořit přesně symetrický signál, napodobující například funkci sinus.

```
if(MirrorMode1)
{
    int index = DPoints.IndexOf(actPoint);
    int indexToChange = DPoints.Count - 1 - index;

    DPoints[indexToChange].Top = Max - DPoints[index].Top;
}
```



Obr. 7 - Ukázka režimu Mirror mode 1

```
if(MirrorMode2)
{
    int index = DPoints.IndexOf(actPoint);
    int indexToChange = ((index > DPoints.Count / 2) ? (index - DPoints.Count / 2) :
                         (index + DPoints.Count / 2)) % DPoints.Count;

    DPoints[indexToChange].Top = Max - DPoints[index].Top;
}
```



Obr. 8 - Ukázka režimu Mirror mode 2

Uživatelský signál je po vytvoření také možné uložit do textového souboru, nebo je možné vzor signálu ze souboru načíst. Vzorky signálu jsou v souboru uloženy po řádcích jako desetinná čísla v rozsahu -1 až 1. Níže následuje ukázka obsahu textového souboru vytvořeného ze vzorku na obrázku Obr. 8:

```
0,650684931506849
1
0,280821917808219
0,452054794520548
-0,650684931506849
-1
-0,280821917808219
-0,452054794520548
```

3.5.1 Výpočet vzorků uživatelsky definovaného signálu

Uživatelsky definovaný signál se zadává v rozsahu 4 až 40 vzorků, k tomu aby bylo možné tento signál generovat s libovolnou frekvencí, je třeba jej převzorkovat, na 4 až 512 vzorků podle požadované frekvence. Pro převzorkování požadovaného signálu byla vytvořena metoda *interpolate*, která na základě podobnosti pravoúhlých trojúhelníků dopočítává potřebné vzorky. Vstupními parametry této funkce jsou pole vstupních bodů rozšířené o poslední bod, který je totožný jako první bod, a počet požadovaných bodů pro generování dané frekvence, výstupem je nová posloupnost bodů požadované délky.

Pro vlastní výpočet je nejprve nutné určit poměry (poměrné délky kroků) vstupního a výstupního signálu k délce celé periody: $D_{in} = 1/L_{in}$ *a* $D_{out} = 1/L_{out}$. Vlastní výpočet probíhá ve dvou vnořených cyklech- první cyklus má tolik kroků, z kolika úseček se skládá vstupní křivka,

v druhém cyklu jsou dopočítávány body, které jsou vytvářeny v rámci aktuální úsečky. Pro určení počtu iterací druhého cyklu bylo nutné vytvořit pomocnou proměnnou *inPart*, která odpovídá délce vstupní úsečky, ale je vyjádřena počtem vzorků výstupní posloupnosti. Počítadlo vnořeného cyklu potom v každém kroku přičítá délku výstupní úsečky, naopak vyjádřenou počtem vzorků vstupní posloupnosti, a probíhá, dokud je tato hodnota menší než proměnná *inPar*t. Toto řešení se může zdát poněkud krkolomné, ale dospěl jsem k němu po několika neúspěšných pokusech s výpočtem s desetinnými čísly. Tento postup se ukázal jako nejpřesnější díky využití pouze celočíselné logiky.

Vlastní výpočet je pak dán následujícím vzorcem:

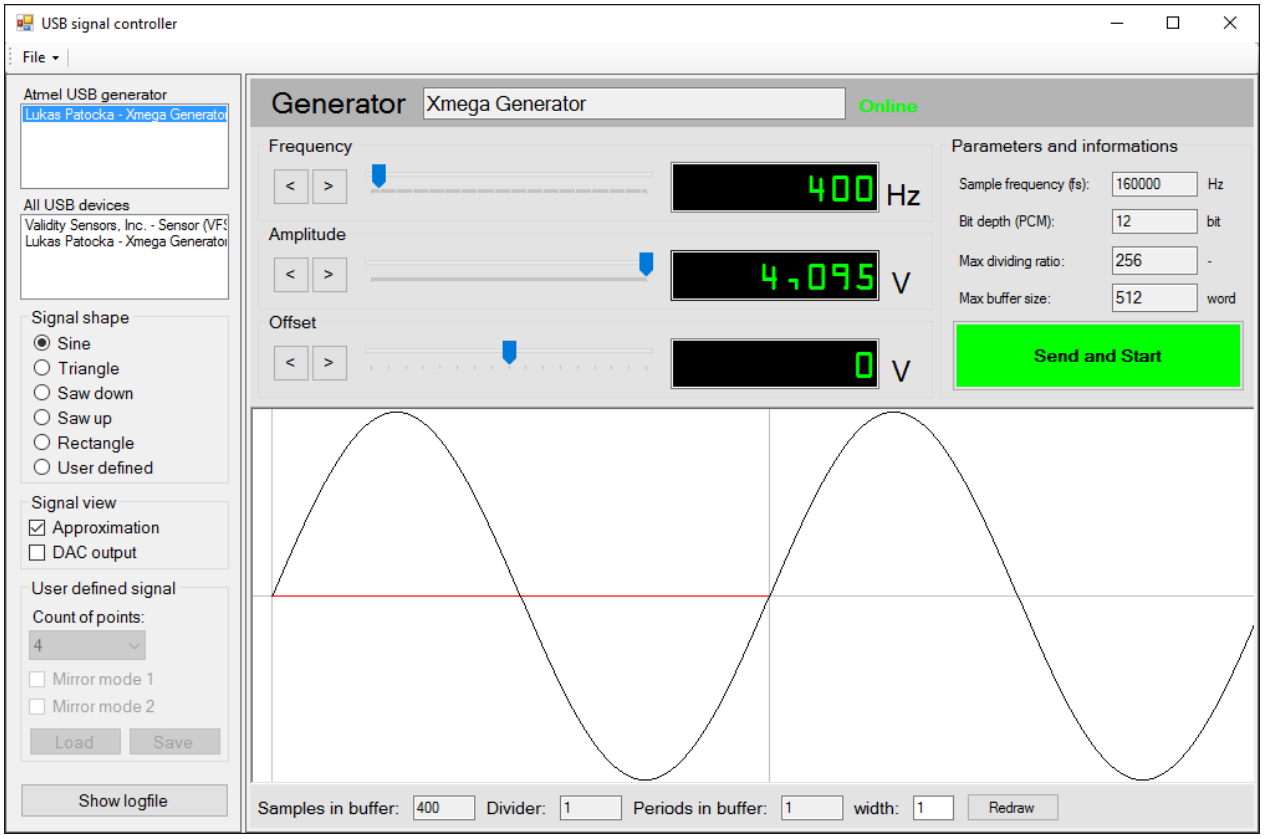
$$P_j = P_i + \left((P_{i+1} - P_i) \cdot \frac{j \cdot D_{out} - i \cdot D_{in}}{D_{in}} \right)$$

Kde *i* je počítadlo pro indexaci vstupních bodů a *j* je počítadlo výstupních bodů.

```
public static double[] interpolate(double[] inPoints, int outCount)
{
    double[] outPoints = new double[outCount];
    double first, second, inPart;
    double d_in = 1.0 / (inPoints.Length - 1);
    double d_out = 1.0 / outCount;

    int _in = outCount;
    int _out = (inPoints.Length - 1);
    int j = 0;

    for (int i = 0; i < inPoints.Length - 1; i++)
    {
        first = inPoints[i];
        second = inPoints[i + 1];
        inPart = (i + 1) * _in;

        while ((_out * j) < inPart)
        {
            if (first == second)
                outPoints[j] = first;
            else
                outPoints[j] = first + ((second - first) *
                               ((j * d_out) - (i * d_in)) / d_in);
            j++;

        }
    }
    return outPoints;
}
```

Tento postup pomocí metody *interpolate* jsem nakonec využil i pro výpočet trojúhelníkového signálu, protože se ukázal jako přesnější. Výše uvedený výpočet by byl vhodný pouze v případě, že by byla délka periody dělitelná čtyřmi, pro libovolnou délku periody vzniká při výpočtu zkreslení. Jako vstupní pole metody *interpolate* byla pro generování trojúhelníkového signálu využita posloupnost hodnot [0, 1, 0, -1, 0].

3.6 Vzhled aplikace



Obr. 9 - Ukázka PC aplikace

V levém panelu aplikace se nacházejí dva seznamy USB zařízení. První se zařízeními, které se identifikovali svými VID a PID jako generátor, v druhém veškerá USB zařízení podporované ovladačem LibUsb-Win32. Poklepáním na položku prvního seznamu se aplikace připojí k mikrokontroléru USB generátoru a následně z něj vyčte parametry zařízení, které jsou zobrazeny v pravé horní části hlavního panelu. Zobrazené parametry jsou: samplovací frekvence a bitová hloubka DA převodníku, maximální dělící poměr samplovací frekvence a velikost bufferu pro

uložení vzorku signálu. Pod těmito parametry se nachází tlačítko pro odeslání bufferu a spuštění vlastního generování signálu DA převodníkem. Pod seznamy v levém panelu se dále nachází přepínače pro nastavení tvaru generovaného signálu (sinusový, trojúhelníkový, pilový klesající a stoupající, obdélníkový a uživatelsky definovaný) a režim zobrazení generovaného signálu. V případě zvolení uživatelsky definovaného signálu se aktivuje poslední část levého panelu, která umožňuje nastavení počtu bodů určených k nastavení tvaru signálu, režim zrcadlení pro usnadnění tvorby signálu a tlačítka pro uložení a načtení vytvořeného signálu.

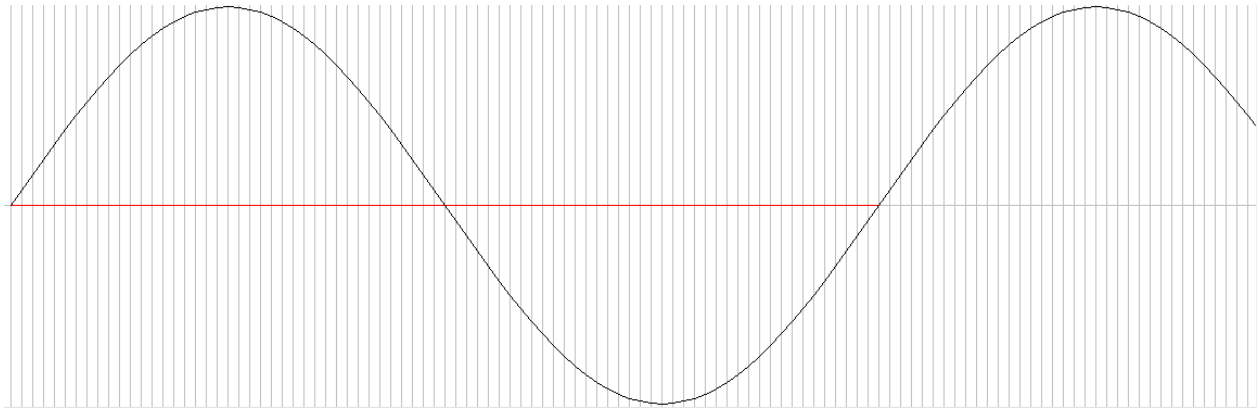
V pravé horní části se nachází ovládání generátoru- tři track bary pro nastavení frekvence, amplitudy a offsetu. Za těmito track bary jsou textová pole pro zobrazení aktuálně nastavené hodnoty. Po kliknutí do textového pole zobrazujícího frekvenci je možné zapsat přímo požadovanou hodnotu frekvence. Vzhledem k tomu, že uživatel nemusí vždy zadat hodnotu, kterou je generátor schopen generovat, je vždy nastavena hodnota nejbližší k hodnotě požadované. Níže je zobrazena ukázka funkce pro vyhledávání nejbližší hodnoty:

```
int i = 1;
while (val > waveform.Tones[i++].Freq);

if (Math.Abs(val - waveform.Tones[i - 1].Freq) <
    Math.Abs(val - waveform.Tones[i - 2].Freq))

    trackBar2.Value = i - 1;
else
    trackBar2.Value = i - 2;

trackBar2_Scroll(null, null);
```

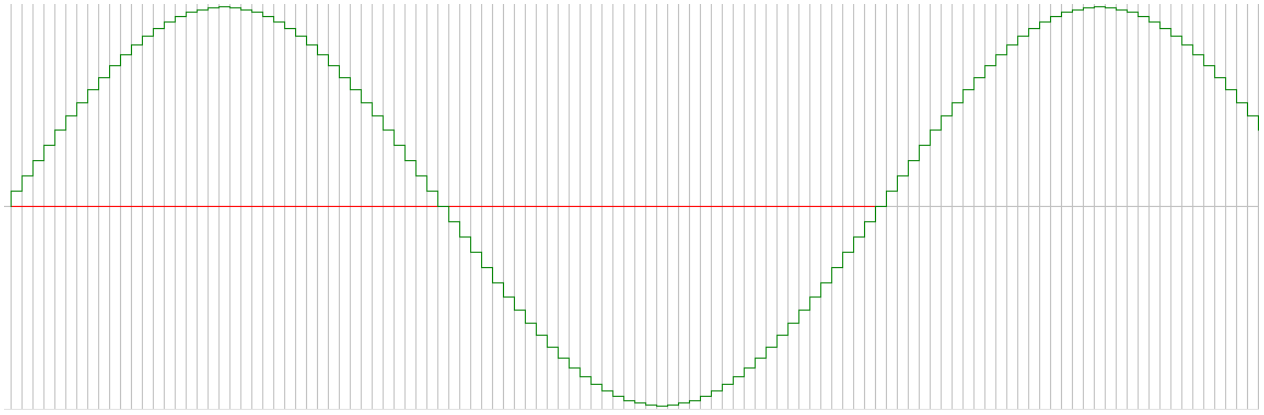
Ve spodní části se nachází panel zobrazující aktuálně nastavený signál, může být zobrazena buď spojitá aproximace signálu (jednotlivé body jsou mezi sebou propojeny přímkami) nebo schodovitý tvar jako je na výstupu DA převodníku. Pod zobrazovacím panelem je možní vyčíst informace o aktuálně nastaveném signálu- počet vzorků na periodu a dělič samplovací frekvence. Políčkem „width“ je možné zobrazený signál „roztáhnout“, aby bylo možné zobrazit vyšší frekvence. V zobrazení signálu je vyznačena první perioda- odesílaný buffer.

Vykreslování signálu se provádí v metodě *panel3_Paint*, která je registrována jako handler události *panel3.Paint*. Níže je uvedena ukázka kódu vykreslení spojitého signálu bez pomocných čar a příklady vykreslovaných signálů.

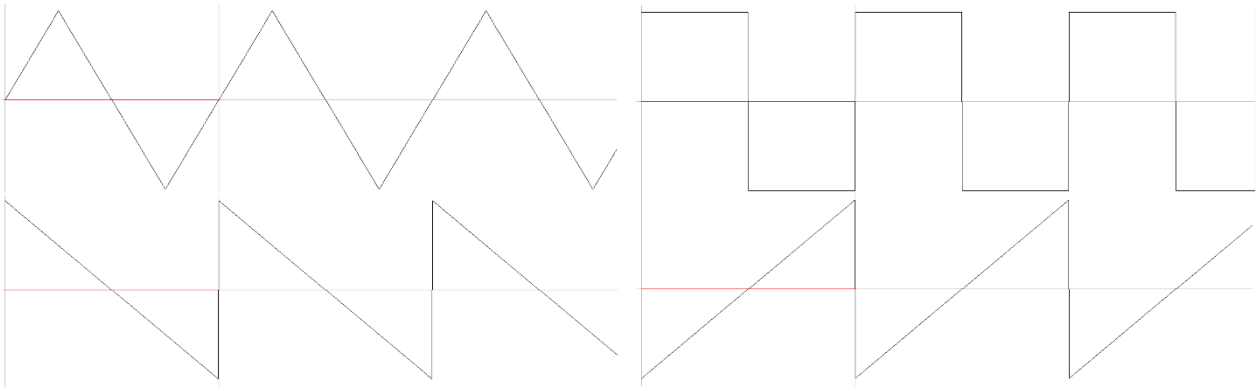
```
for (int i = 0; i < pnt.Length; i++)
{
    pnt[i].X = 15 + (int)(i * waveform.Tones[trackBar2.Value].Devider * mult);
    pnt[i].Y = WinHeight + (int)(-1 * (offset + (waveform.Sequnce[i %
        waveform.Sequnce.Count])) * ratio);
}
if (checkBox1.Checked) g.DrawLines(Pens.Black, pnt);
```



Obr. 10 – Ukázka aproximace sinusového signálu



Obr. 11 – Ukázka schodovitého zobrazení sinusového signálu



Obr. 12 – Ukázky ostatních signálů

Po přepnutí na uživatelsky definovaný signál se mění zobrazení vykreslovacího panelu. Na panelu jsou nyní zobrazeny body, které byly vytvořeny jako potomek třídy Button. V nové třídě byly přepsány její metody upravující zobrazení a chování bodu.

Nejprve byla nahrazena metoda vykreslující tlačítko- *OnPaint*. V těle této metody se vykresluje červený kruh pomocí metody *FillElipse* třídy *Graphics* a upravuje velikost neboli region tvaru tlačítka přiřazením stejně velkého kruhu do parametru *Region*.

```
protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
{
    e.Graphics.FillEllipse(Brushes.Red, 0, 0, ClientSize.Width, ClientSize.Height);
    GraphicsPath grPath = new GraphicsPath();
    grPath.AddEllipse(0, 0, ClientSize.Width, ClientSize.Height);
    this.Region = new System.Drawing.Region(grPath);
}
```

Aby bylo možné s body pohybovat, bylo nutné upravit metody zpracovávající kliknutí na bod a pohyb myši: *OnMouseDown*, *OnMouseUp* a *OnMouseMove*. V metodě *OnMouseDown* je uložena aktuální poloho bodu, v metodě *OnMouseMove* je nastavena vertikální poloha bodu podle pohybu ukazatele myši a v metodě *OnMouseUp* je ošetřen stav, kdy by uživatel bod přesunul mimo rozsah panelu, v takovém případě je bod vrácen do maximální, nebo minimální vertikální polohy. Všechny tyto metody navíc volají metodu *Refresh* nadřazeného panelu, aby se změny v poloze bodu v daný okamžik překreslovaly na uživatelském rozhraní.

```
protected override void OnMouseDown(MouseEventArgs mevent)
{
    if (mevent.Button == System.Windows.Forms.MouseButtons.Left)
    {
        MouseDownLocation = mevent.Location;
        this.Parent.Refresh();
    }
}
protected override void OnMouseMove(MouseEventArgs mevent)
{
    this.Top = mevent.Y + this.Top - MouseDownLocation.Y;
    this.Parent.Refresh();
}
protected override void OnMouseUp(MouseEventArgs mevent)
{
    if (mevent.Button == System.Windows.Forms.MouseButtons.Left)
    {
        if (this.Top < 0) this.Top = 0;
        if (this.Top > MyForm.Max) this.Top = MyForm.Max;
        this.Parent.Refresh();
    }
}
```

4 Aplikace mikrokontroléru

V této kapitole bude rozebrána implementace softwaru pro mikrokontrolér. První část této kapitoly se bude věnovat realizaci USB rozhraní na straně mikrokontroléru za pomoci funkcí knihovny LUFA, bude se jednat zejména o navázání spojení s počítačem (takzvaný proces enumerace) a vlastní přenosy dat. Pro realizaci USB rozhraní bude využita knihovna LUFA verze 151115. Další část se bude věnovat konfiguraci a realizaci DA převodníku jako generátoru periodických signálů.

4.1 Nastavení časování mikrokontroléru Atmel XMEGA

Mikrokontroléry řady XMEGA umožňují použití více zdrojů časování- interní, externí, nebo z interní násobičky. Jádro mikrokontroléru je nastaveno na frekvenci (F_CPU) 32 MHz z interní násobičky frekvence, ta využívá jako zdroj referenční frekvence interní oscilátor 2 MHz.

```
XMEGACLK_StartPLL(CLOCK_SRC_INT_RC2MHZ, 2000000, F_CPU);
XMEGACLK_SetCPUClockSource(CLOCK_SRC_PLL);
```

4.1.1 Nastavení časování pro USB

Jako zdroj frekvence (F_USB) 48 MHz pro USB rozhraní je využita DFLL násobička, která využívá jako zdroj referenční frekvence interní oscilátor 32 MHz.

```
XMEGACLK_StartInternalOscillator(CLOCK_SRC_INT_RC32MHZ);
XMEGACLK_StartDFLL(CLOCK_SRC_INT_RC32MHZ, DFLL_REF_INT_USBSOF, F_USB);
```

4.1.2 Nastavení časování pro DA převodník

Pro nastavení časování DA převodníku se využívá registr *TCC0.CTRLA*, do kterého je možné nastavit frekvenci jádra, případně její podíl až f_{cpu} / 1024, dále je možné nastavit častost vyvolání přerušení počtem taktů časovače, po jejichž uplynutí je přerušení vyvoláno- registr *TCC0.PER*. Pozn.: při nastavení vyšší frekvence blížící se frekvenci jádra přestává fungovat USB komunikace, pravděpodobně protože často vyvolávané přerušení DA převodníku zasahuje přenosu dat po USB a data nejsou přenesena včas. Z tohoto důvodu byla samplovací frekvence DA převodníku nastavena na 160 kHz ($f_s = f_{cpu} / 200$).

```
TCC0.CTRLA = TC_CLKSEL_DIV1_gc;
TCC0.PER = 200;
```

S takto nastaveným časováním a zvolenou nejkratší možnou periodou signálu čtyři vzorky, bude generátor schopen generovat signál o maximální frekvenci 40 000 Hz.

4.2 Navázání USB komunikace

V okamžiku připojení USB zařízení k počítači začíná proces takzvané enumerace, kdy si operační systém vyžádá informace o tomto zařízení. Mikrokontrolér odesílá požadované informace o zařízení, možných konfiguracích a jejich rozhraních pomocí deskriptorů. K tomuto účelu je nutné všechny tyto deskriptory v zařízení definovat. K tomu jsou určeny soubory `Descriptors.h` a `Descriptors.c`.

4.2.1 Deskriptory

- Deskriptor zařízení
    - Přenáší informace o verzi USB, třídě zařízení, VID, PID.
    - Je přenášen jako první po připojení k počítači, na základě informací v tomto deskriptoru rozhoduje operační systém o výběru ovladače tohoto zařízení.

- Konfigurační deskriptor (Configuration descriptor)
    - Obsahuje informace o konfiguraci, deskriptor rozhraní a deskriptory koncových bodů, číslo konfigurace počet rozhraní a napájecí proud. Zařízení může mít více konfigurací.
    - O výběru použité konfigurace rozhoduje aplikace, která se zařízením bude komunikovat.

- Deskriptor rozhraní (Interface descriptor)
    - Obsahuje informace o rozhraní a počtu koncových bodů.

- Deskriptory koncových bodů (Endpoint descriptor)
    - Přenáší informace o konkrétním koncovém bodu- jeho adresu, typu (interrupt, bulk, nebo isochronní), směr přenosu, velikost mezipaměti a časování.

- String descriptor
    - Obsahuje textové popisy zařízení.

4.2.2 Proces enumerace

Pro realizaci potřebných funkcí bude třeba využívat kontrolní přenosy, pro ty není nutná žádná další konfigurace, protože jsou využívány systémem, ale umožňují i uživatelské využití, a hromadné (bulk) přenosy, pro ty je potřeba vytvořit endpoint ve směru z počítače do mikrokontroléru. Pro ověření správnosti probíhající USB bulk komunikace jsem vytvořil i druhý bulk endpoint, díky kterému je možné realizovat komunikaci z mikrokontroléru směrem do počítače. Dále jsem vytvořil dva endpointy pro realizaci isochronní komunikace, které ale nejsou v rámci této práce využity.

Pro odeslání všech dekriptorů musíme vytvořit strukturu deskriptoru zařízení *USB_Descriptor_Configuration_t*, která obsahuje všechny ostatní deskriptory.

```
typedef struct
{
	USB_Descriptor_Configuration_Header_t	Configuration;
	USB_Descriptor_Interface_t		Interface;
	USB_Descriptor_Endpoint_t		IsoEpIn;
	USB_Descriptor_Endpoint_t		IsoEpOut;
	USB_Descriptor_Endpoint_t		BulkEpIn;
	USB_Descriptor_Endpoint_t		BulkEpOut;

}	USB_Descriptor_Configuration_t;
```

K odeslání deskriptorů slouží funkce *CALLBACK_USB_GetDescriptor* knihovny LUFA, ta je volána přímo knihovní rutinou v okamžiku příchozího požadavku ze strany hostitelského PC.

```
uint16_t CALLBACK_USB_GetDescriptor(
    const uint16_t wValue,
    const uint8_t wIndex,
    const void** const DescriptorAddress)
```

Tato funkce na základě parametru *wValue* předává knihovně LUFA ke zpracování adresu požadovaného deskriptoru pomocí výstupního parametru (ukazatele) *DescriptorAddress*. Knihovna následně odesílá tento deskriptor hostitelskému počítači.

Před zahájením komunikace je nutné ještě inicializovat koncové body, to se provádí ve funkci *EVENT_USB_Device_ConfigurationChanged*, která je volána knihovnou LUFA v okamžiku, kdy si hostitelský počítač zvolí požadovanou konfiguraci. Vlastní inicializace se provádí zavoláním funkce *Endpoint_ConfigureEndpoint*.

```
void EVENT_USB_Device_ConfigurationChanged(void)
{
  bool result = true;

  result &= Endpoint_ConfigureEndpoint(EP_ISO_OUT, EP_TYPE_ISOCHRON, EP_ISO_SIZE,2);
  result &= Endpoint_ConfigureEndpoint(EP_ISO_IN, EP_TYPE_ISOCHRON, EP_ISO_SIZE,2);
  result &= Endpoint_ConfigureEndpoint(EP_BULK_OUT, EP_TYPE_BULK, EP_BULK_SIZE,2);
  result &= Endpoint_ConfigureEndpoint(EP_BULK_IN, EP_TYPE_BULK, EP_BULK_SIZE,2);

  if(!result) LedBlinkRed();
  else LedBlinkYellow();
}
```

Po úspěšném dokončení inicializace koncových bodů je zařízení připravené k realizaci vlastních datových přenosů po USB sběrnici.

4.3 Řídící přenosy uživatelských zpráv

Kontrolní přenosy jsou vždy inicializovány ze strany hostitelského zařízení. Po přijetí uživatelsky definované zprávy je vyvolána událost reprezentovaná funkcí *EVENT_USB_Device_ControlRequest* a veškeré informace o zprávě jsou uloženy v globální proměnné USB_ControlRequest, typu *USB_Request_Header_t*. Identifikátor zprávy je uložen v parametru *bRequest*. Seznam využitých uživatelsky definovaných řídících zpráv je uveden v tabulce níže.

Id zprávy	Směr přenosu	Název zprávy	Význam zprávy
0x50	z MC do PC	GET_DA_BITS	Počet bitů DA převodníku
0x51	z MC do PC	GET_DA_FREQ	Samplovací frekvence DA převodníku
0x52	z MC do PC	GET_DA_DEVS	Maximální dělitel sampl. frekvence
0x53	z MC do PC	GET_DA_MEMS	Velikost paměti pro vzorek signálu
0x60	z PC do MC	SET_BF_LENGTH	Nastavení délky vzorku signálu
0x61	z PC do MC	SET_DIVIDER	Dělitel samplovací frekvence
0x62	z PC do MC	SET_DCOFFSET	Velikost stejnosměrné složky

Tabulka 3 - Použité řídící zprávy

Přijetí každé řídící zprávy je nutné potvrdit zavoláním funkce *Endpoint_ClearSETUP*. Po identifikaci příchozí zprávy je třeba na ní reagovat buď vyčtením přijatých dat, nebo odeslat požadovaná data. K odeslání dat z mikrokontroléru využívám funkci *Endpoint_Write_32_LE*, jejímž parametrem je 32 bitové číslo určené k odeslání, písmena LE v názvu funkce značí využití

formátu čísla little endian. Po odeslání dat je třeba opět odeslání potvrdit voláním funkce *Endpoint_ClearIN*. Pro příjem dat je využita funkce *Endpoint_Read_Control_Stream_LE*, která má dva parametry- ukazatel na proměnnou, do které mají být data uložena a její velikost. Přijetí dat musí být opět potvrzeno funkcí *Endpoint_ClearOUT*.

```
void EVENT_USB_Device_ControlRequest()
{
    switch(USB_ControlRequest.bRequest)
    {
    case(GET_DA_BITS):
        if(USB_ControlRequest.bmRequestType == (REQDIR_DEVICETOHOST |
            REQTYPE_VENDOR | REQREC_DEVICE))
        {
            Endpoint_ClearSETUP();
            Endpoint_Write_32_LE(DA_bits);
            Endpoint_ClearIN();
        }
        break;
    case(SET_DCOFFSET):
        if(USB_ControlRequest.bmRequestType == (REQDIR_HOSTTODEVICE |
            REQTYPE_VENDOR | REQREC_DEVICE))
        {
            Endpoint_ClearSETUP();
            Endpoint_Read_Control_Stream_LE(&DCOFFSET, sizeof(DCOFFSET));
            Endpoint_ClearOUT();
        }
        break;
    }
}
```

Před každým přijetím bloku dat obsahující vzorek signálu určený ke generování jsou pomocí kontrolních přenosů nastaveny pomocné parametry, jak již bylo zmiňováno, jedná se o délku vzorku, dělitel samplovací frekvence a stejnosměrná složka signálu. Po přenesení těchto informací může teprve začít vlastní přenos dat.

4.4 Struktura programu mikrokontroléru

Výše uvedená komunikace zatím vždy probíhala v rámci enumerace, inicializace zařízení, nebo jako reakce na rutinu knihovny LUFA volanou formou callback funkce. Proto ještě nyní uvedu stručný popis struktury programu.

Vykonávání vlastního programu začíná, jak je obvyklé v jazyce C nebo C++, funkcí *main*, V její úvodní části se provádí veškeré inicializace mikrokontroléru, v mém případě jsem inicializace vložil do funkce *initDevice*, která je volána jako první ve funkci *main*. Nejprve jsou nastaveny výstupy pro indikační LED diody. Následuje již zmiňované nastavení časování,

inicializace knihovny LUFA a USB rozhraní a nastavení DA převodníku. Po inicializaci následuje povolení vyvolání globálních přerušení mikrokontroléru realizované voláním funkce *sei()* a vlastní nekonečná smyčka programu. Ta obsahuje pouze obsluhu USB zařízení, o kterou se stará knihovna LUFA volaná funkcí *USB_USBTask* a kontrola příchozích dat na USB rozhraní.

```
int main(void)
{
    initDevice();
    sei();

    while (1)
    {
        USB_USBTask();

        Endpoint_SelectEndpoint(EP_BULK_OUT);
        if(Endpoint_IsOUTReceived())
        {
              DACoff();
              _delay_ms(10);
              SetGenerator();
              DACon();
        }
    }
}
```

4.5 Bulk přenosy dat a uložení do EEPROM

Jak jsem již zmiňoval, v hlavní smyčce programu se v každém cyklu provádí kontrola přijatých dat na USB rozhraní. Protože ke komunikaci je v tomto případě využit pouze jeden endpoint typu bulk, provádí se kontrola přijatých dat pouze na něm. Koncový bod je nejprve zvolen funkcí *Endpoint_SelectEndpoint*, která má jako jediný parametr právě číslo zvoleného koncového bodu. Následuje vlastní kontrola přítomnosti přijatých dat na tomto koncovém bodě.

```
Endpoint_SelectEndpoint(EP_BULK_OUT);

if(Endpoint_IsOUTReceived())
{
       DACoff();
       _delay_ms(10);
       SetGenerator();
       DACon();
}
```

Pokud jsou na koncovém bodě přijata data, vrací funkce *Endpoint_IsOUTReceived* hodnotu *true*, v opačném případě *false*. V okamžiku přijetí dat se zastavuje generování signálu DA

převodníkem, aby vyvolávání přerušení nutné pro generování signálu nenarušovalo probíhající USB komunikaci. Funkce pro čtení dat z koncového bodu:

```
uint8_t Endpoint_Read_8 (void)
static uint16_t Endpoint_Read_16_LE (void)
static uint32_t Endpoint_Read_32_LE (void)
uint8_t Endpoint_Read_Stream_LE (void *const Buffer, uint16_t Len, uint16_t *const Count)
uint8_t Endpoint_Read_EStream_LE (void *const Buffer, uint16_t Len, uint16_t *const Count)
```

Všechny uvedené funkce k sobě mají i variantu pro formát big endian. Poslední z uvedených funkcí *Endpoint_Read_EStream_LE* slouží k přímému zápisu do paměti EEPROM. Vzhledem k požadavku na uložení dat v paměti EEPROM, tak aby bylo možné generovat signál i po přivedení pouze napájecího napětí, byla zvolena právě tato funkce.

Vlastní bulk přenos probíhá po částech, jejichž velikost je parametrem koncového bodu. V tomto případě je to 64 bytů. Protože přenášený blok dat může být podstatně větší, provádí se volání funkce cyklicky, ale i přes to jsou data uspořádána v celistvém bloku, protože funkce si ukládá stav svého průběh do posledního parametru *Count*. Jako návratová hodnota funkce je číslo, které může odpovídat následujícím konstantám.

Hodnota	Význam
0	Přenos dokončen bez chyby.
1	Koncový bod byl zavřen.
2	Zařízení bylo odpojeno.
3	Hostitel uzavřel spojení.
4	Vypršel časový limit přenosu.
5	Přenos nebyl dokončen a bude pokračovat.

Tabulka 4 - výsledky bulk přenosu

```
uint8_t result;
uint16_t ready = 0;

while((result = Endpoint_Read_EStream_LE(Data, sizeof(uint16_t) * BSIZE, &ready))
== ENDPOINT_RWSTREAM_IncompleteTransfer);

Endpoint_ClearOUT();

if(result != ENDPOINT_RWSTREAM_NoError)
{
	LedBlinkRed();
}
```

Po vyzkoušení přenosu pomocí funkce knihovny LUFA přímo do paměti EEPROM, se ukázalo, že tento přenos je značně nespolehlivý. Nová data obsahovala velké množství chyb. Po podrobnějším prozkoumání jsem zjistil, že v paměti EEPROM zůstávají uloženy hodnoty předchozích vzorků a novými hodnotami se přepíší jen některé. Provedl jsem krátké měření k ověření chybovosti daného přenosu, nebo zápisu do paměti EEPROM.

Délka bloku dat	Počet chyb	Poměr chyb
503	162	32,2%
511	248	48,5%
440	216	49.1%
511	246	48,1%
490	219	44,7%

Tabulka 5 - Počet chyb přenosu

Z tabulky je patrné, že chybovost přenosu a uložení do EEPROM je příliš vysoká a nebude možné přenos takto realizovat. Proto jsem si ověřil i spolehlivost funkce ukládající data do paměti RAM. Ta se oproti funkci pracující s EEPROM ukázala jako spolehlivá. Je tedy jisté, že k chybám v signálu dochází až při jeho ukládání do paměti EEPROM.

Z tohoto důvodu jsem upravil způsob ukládání tak, že data byla nejprve načtena do paměti RAM a následně opakovaně nahrávána do paměti EEPROM dokud oba bloky paměti nebyly shodné. Pro kopírování bloku dat z paměti RAM do paměti EEPROM byla využita funkce eeprom_write_block, jejímiž parametry jsou ukazatel na vstupní data, ukazatel na cílovou oblast paměti a délka bloku dat.

```
while((result = Endpoint_Read_Stream_LE(tmp_Data, sizeof(uint16_t) * BSIZE, &ready))
== ENDPOINT_RWSTREAM_IncompleteTransfer);

do
{
	eeprom_write_block(tmp_Data, Data, sizeof(uint16_t) * BSIZE);
} while (compare_data() > 0);
```

Pro porovnání bloků dat jsem si vytvořil funkci compare_data, tato funkce nemá žádné parametry, protože oba ukazatele na data jsou uloženy v globálních proměnných, funkce tedy pracuje přímo s nimi. Návratovou hodnotou funkce je index první nalezené neshody v datech. V případě, že jsou data shodná, funkce vrací 0.

```
uint16_t compare_data()
{
    for(size_t i = 0; i < BSIZE; i++)
    {
        if(tmp_Data[i] != eeprom_read_word(&Data[i]))
        {
              return i;
        }
    }
    return 0;
}
```

Tato metoda se sice osvědčila, ale čas kopírování bloku dat do paměti EEPROM se zvýšil na zhruba 35 až 50 sekund. Původní nespolehlivý přenos přímo do paměti EEPROM trval okolo 8 sekund a čas USB přenosu do paměti RAM zabral necelé 3 sekundy. Z tohoto důvodu jsem funkci ještě dále upravil.

Data se přenáší do paměti RAM stejně jako v algoritmu 2, ale kopírování bylo provedeno po 1 wordu pomocí funkce eeprom_write_word, po každém zapsaném wordu do paměti EEPROM je tento word porovnán se zdrojovým wordem v paměti RAM. Tímto způsobem se čas přenosu snížil na asi polovinu, celý přenos tak trvá asi 16 až 22 sekund.

```
for(int i = 0; i < BSIZE; i++)
{
    do
    {
         eeprom_write_word(&Data[i], tmp_Data[i]);
    } while (tmp_Data[i] != eeprom_read_word(&Data[i]));
}
```

4.6 Generování signálu

Po dokončení přenosu bufferu se vzorkem signálu se generování spouští voláním funkce *DACon* jak je vidět v ukázce funkce *main* v hlavní smyčce programu. Tělo této funkce obsahuje nastavení registru DACB.CTRLA, které povoluje spuštění obou kanálů DA převodníku a spouští časovač, který vyvolává přerušení pro obsluhu DA převodníku.

```
void DACon(void)
{
    DACB.CTRLA = DAC_CH0EN_bm | DAC_CH1EN_bm | DAC_ENABLE_bm;
}
```

Vlastní generování signálu DA převodníkem probíhá v obsluze funkce vyvolané přerušením časovače DA převodníku na základě nastavení časování, které bylo popsáno výše. Aktuální úroveň výstupního napětí převodníku je načítána z paměti EEPROM, do které byl vzorek signálu uložen. Načtená aktuální hodnota je vložena do datového registru kanálu A DA převodníku a následně je tato hodnota přenesena na výstup PB2 mikrokontroléru Atmel. Do datového registru kanálu B je vkládána hodnota stejnosměrné složky, ta je pak přenesena na výstup PB3. Níže je vidět obrázek signálu na výstupu mikrokontroléru, DA převodník byl nastaven na sinusový průběh o frekvenci 40 kHz a plné výstupní napětí.

```
ISR( TCC0_OVF_vect )
{
	if (index >= BSIZE) index -= BSIZE;
	DACB.CH0DATA = eeprom_read_word(&Data[index++]);
	DACB.CH1DATA = DCOFFSET;
}
```

Obr. 13 - Signál na výstupu DA převodníku

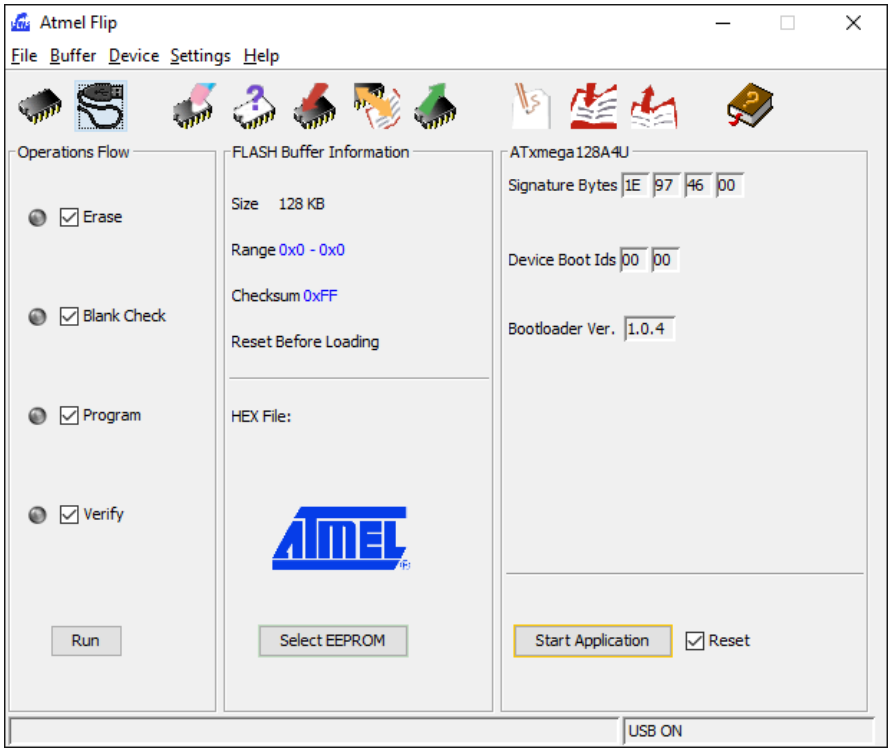
4.7 Programování mikrokontroléru Atmel

Software mikrokontroléru Atmel byl vytvořen v prostředí AtmelStudio verze 6.2, které přímo umožňuje kompilaci hotového projektu, včetně přilinkování funkcí knihovny LUFA. Výstupem překladu projektu je binární soubor typu *hex*. Podle informací z výstupu překladu je využití paměti mikrokontroléru XMEGA128A4U uvedeno v následující tabulce.

Typ paměti	Využití paměti	Poměr využití paměti
Programová paměť	4802B	3,4%
Paměť RAM	1797B	21,9%
Paměť EEPROM	1030B	50,3%

Tabulka 6 - Využití paměti mikrokontroléru

Pro nahrání binárního souboru do paměti mikrokontroléru bylo využito programu Atmel Flip verze 3.4.7. Po stisknutí bootovacího tlačítka mikrokontroléru se přepne do režimu nahrávání programu. V tomto režimu je možné se pomocí programu Atmel Flip připojit k USB rozhraní mikrokontroléru a následně přenést soubor *hex*, poté je po restartu mikrokontroléru spuštěn nahraný program a zařízení je připraveno k použití.

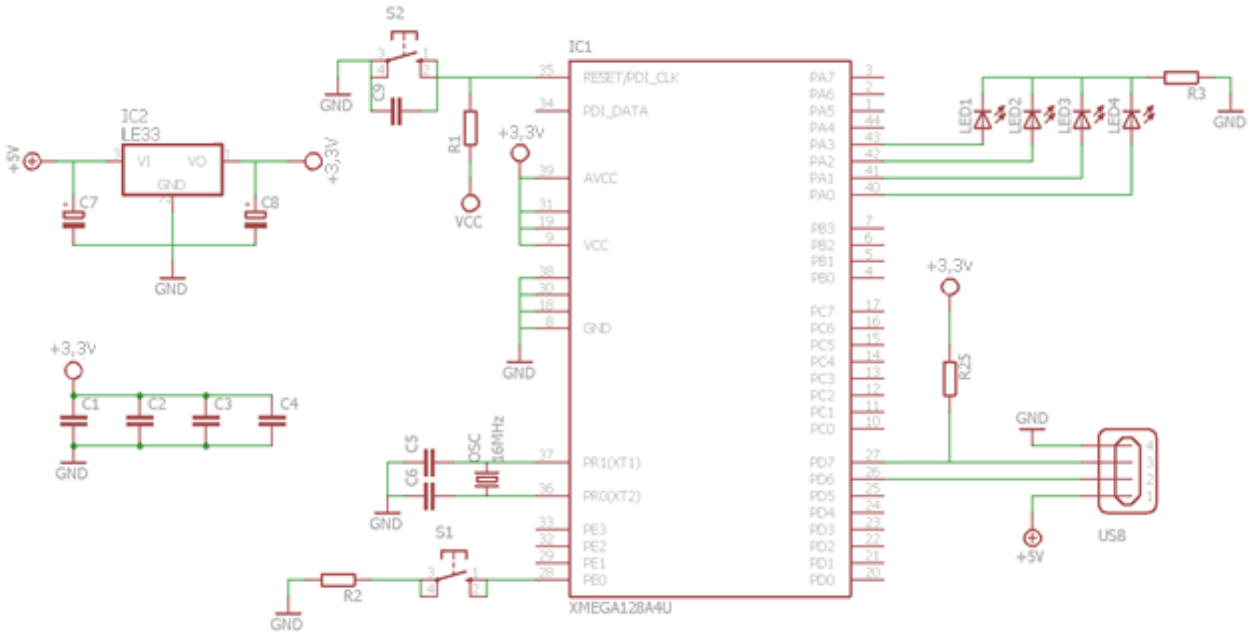


Obr. 14 - Atmel Flip

5 Návrh zapojení mikrokontroléru

Návrh elektrického zapojení se skládá z několika částí, budou v této kapitole popsány. První část se bude věnovat vlastnímu zapojení mikrokontroléru Atmel XMEGA128A4U, součástí nutných k jeho fungování, signalizačních LED diody a USB konektoru.

Pro návrh zapojení mikrokontroléru bylo využito výrobcem doporučené zapojení doplněné o signalizační LED diody na portech PA0 až PA3, tlačítko reset na portu RESET a tlačítko pro spuštění bootloaderu na portu PE0. Nutnou součástí je napájecí zdroj, který převádí napájecí napětí USB sběrnice 5 voltů na 3,3 voltu, tento zdroj je nutný protože napájecí napětí mikrokontroléru je 3,3 voltu. Napájecí napětí je přiváděno na celkem čtyři porty, co nejblíže k těmto portům jsou umístěny filtrační kondenzátory C1 až C4. Podle doporučeného zapojení byl připojen i krystalický oscilátor doplněný kondenzátory C5 a C6. Na porty PD6 a PD7 jsou připojeny datové vodiče USB sběrnice, ke kladnému vodiči je navíc připojen rezistor R25, který jej spojuje s kladnou větví napájení mikrokontroléru. Tento rezistor slouží k informování hostitelského zařízení o tom, že zařízení podporuje přenos ve full speed režimu.



Obr. 15 - Základní zapojení mikrokontroléru

5.1 Napájecí zdroj pro operační zesilovače

Napájení celého obvodu je realizováno pouze z napájecího napětí USB sběrnice, tedy 5 voltů. Toto napětí je pro generátor signálu poměrně malé a pro přidání stejnosměrné složky- offsetu, nedostačující. Rozkmit výstupního napětí z DA převodníku je roven jeho napájecímu napětí, což je 3,3 voltu. Po přičtení napětí z druhého DA převodníku by mohlo celkové výstupní napětí přesáhnout 5 voltů, teoreticky až 6,6 voltu, což už USB sběrnice neumožňuje.

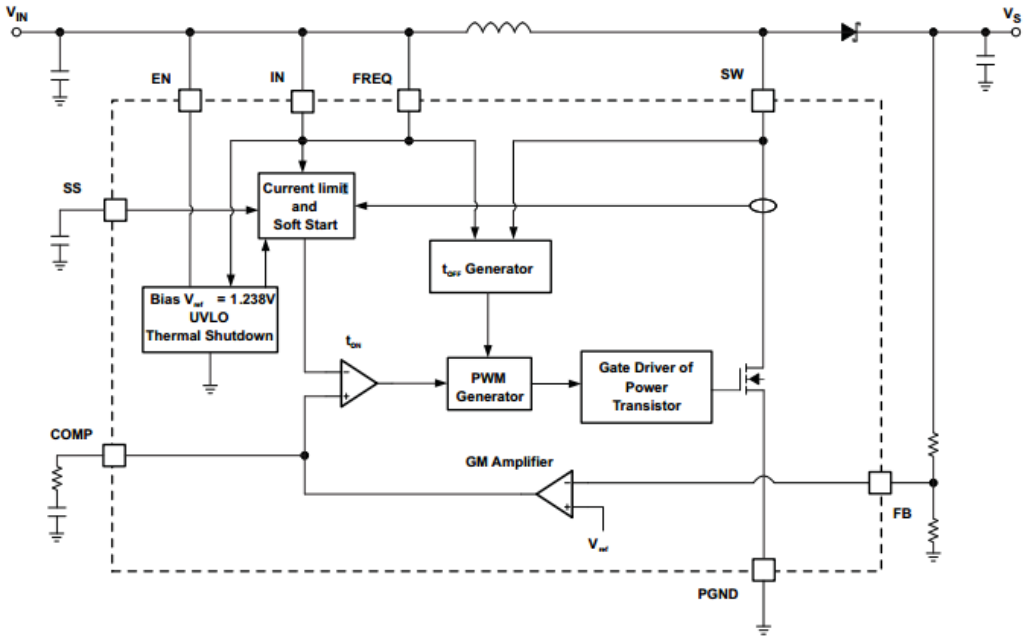
Z tohoto důvodu jsem přistoupil k použití stejnosměrného měniče napětí, takzvaného step-up měniče, který mi umožní zvýšit výstupní napětí generátoru.

Jako step-up měnič jsem si zvolil integrovaný obvod společnosti Texas Instruments s označením TPS61085. Jedná se o vysokofrekvenční DC-DC měnič s vysokou efektivitou. V následující tabulce jsou uvedeny technické parametry tohoto obvodu.

Vstupní napětí:	2,3 V – 6 V
Maximální výstupní napětí:	18,5 V
Maximální výstupní proud:	2 A
Pracovní frekvence:	650 kHz / 1,2 MHz

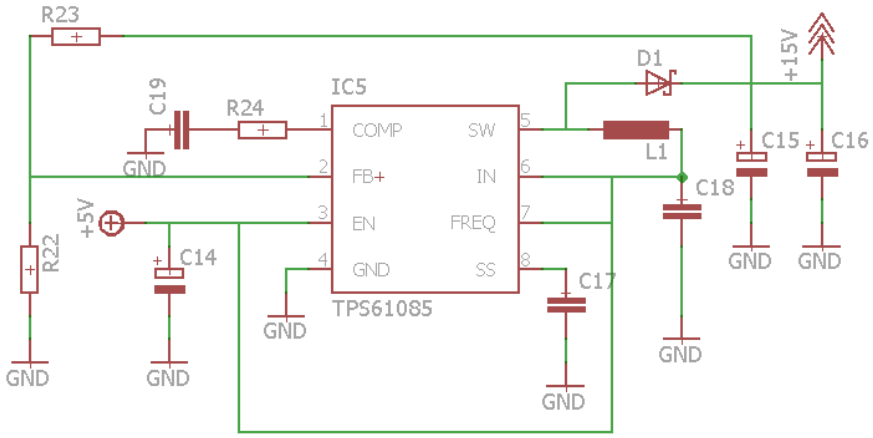
Tabulka 7 - Parametry obvodu TPS16085

Tento integrovaný obvod dále obsahuje tepelnou ochranu a obvod pro nastavitelné zpoždění zapnutí- soft start. Na následujícím obrázku je uvedeno vnitřní blokové schéma obvodu.



Obr. 16 - Vnitřní schéma obvodu TPS61085

K realizaci napájecího zdroje integrovaných obvodů bylo využito výrobcem doporučené zapojení obvodu s drobnými úpravami. Hodnota výstupní napětí byla zvolena na 15 voltů, protože napájecí napětí použitých operačních zesilovačů je maximálně 16 voltů a napětí okolo 15 voltů je dostačující pro výstup generátoru signálu i s přidanou stejnosměrnou složkou. Na následujícím obrázku je uvedeno schéma zapojení obvodu step-up měniče.



Obr. 17 - Schéma zapojení step up měniče

Výstupní napětí měniče se nastavuje pomocí odporového děliče tvořeným rezistory R22 a R23. Dle katalogového listu je hodnota odporu R22 daná a to 18kΩ, dále je pevně daná hodnota vnitřního referenčního napětí U_{FB} = 1,238V. Níže je uveden postup výpočtu hodnoty rezistoru R23.

$$R_{23} = R_{22} \cdot \left(\frac{V_{OUT}}{V_{FB}} - 1\right) \; [k\Omega]$$

$$R_{23} = 18 \cdot \left(\frac{15}{1{,}238} - 1\right) \cong \underline{200\ k\Omega}$$

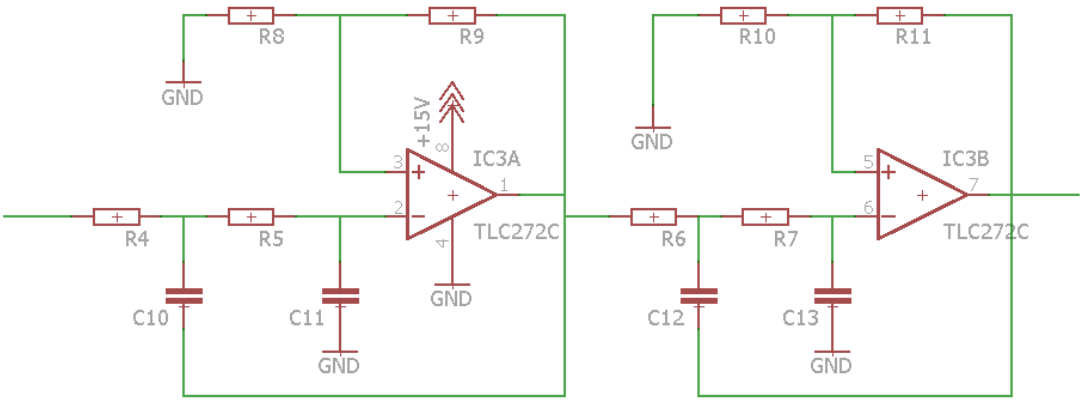
5.2 Návrh rekonstrukčního filtru

Jak bylo zmíněno v kapitole 2.7, jako rekonstrukční filtr byl zvolen aktivní filtr typu dolní propust dle Butterworthovy aproximace 4. řádu realizovaný zapojením Sellen-Key. Mezní frekvence filtru je shodná s maximální frekvencí, kterou je schopen generovat DA převodník, to znamená 40kHz. Vyšší frekvence obsažené ve spektru generovaného signálu budou tedy potlačeny.

Výpočet charakteristiky filtru je daný polynomem Butterworthovy aproximace a jeho řád odpovídá požadovanému řádu filtru.

řád filtru (n)	koeficienty normovaného Butterworthova polynomu $B_N(p)$
1	$(p + 1)$
2	$(p^2 + 1{,}4142p + 1)$
3	$(p^2 + p + 1) \cdot (p + 1)$
4	$(p^2 + 0{,}7654p + 1) \cdot (p^2 + 1{,}8478p + 1)$
5	$(p^2 + 0{,}618p + 1) \cdot (p^2 + 1{,}618p + 1) \cdot (p + 1)$
6	$(p^2 + 0{,}5176p + 1) \cdot (p^2 + 1{,}4142p + 1) \cdot (p^2 + 1{,}9318p + 1)$

Tabulka 8 - koeficienty Butterworthova polynomu



Obr. 18 - Schema zapojení rekonstrukčního filtru

Pro filtr 4. řádu bude tedy využit polynom:

$$B_N(p) = (p^2 + 0{,}7654p + 1) \cdot (p^2 + 1{,}8478p + 1)$$

Podle návrhu Sellen-Key zapojení bude filtr rozdělen na dvě části, kde každá bude obsahovat jeden operační zesilovač a bude odpovídat jedné závorce normovaného polynomu. Napěťové přenosy jednotlivých částí se potom rovnají:

$$A_{U1} = 3 - 0{,}7654 = \underline{2{,}2346}$$
$$A_{U2} = 3 - 1{,}8478 = \underline{1{,}1522}$$

Podle vzorce pro výpočet napěťového zesílení neinvertujícího operačního zesilovače potom určíme poměr rezistorů:

$$A_U = \frac{R_2}{R_1} + 1$$
$$R_2 = R_1(A_U - 1)$$

Hodnota rezistoru R1, tedy konkrétně rezistorů R8 a R10 byla stanovena na 10kΩ. Následuje výpočet rezistorů R9 a R11:

$$R_9 = R_8(A_{U1} - 1)\ [\Omega]$$
$$R_9 = 10(2{,}2346 - 1) = \underline{12{,}35\ k\Omega}$$

$$R_{11} = R_{10}(A_{U1} - 1)\ [\Omega]$$
$$R_{11} = 10(1{,}1522 - 1) = \underline{1{,}525\ k\Omega}$$

Mezní frekvence celého filtru je daná hodnotami rezistorů R4, R5, R6, R7 a kondenzátorů C10, C11, C12, C13 a je dána následujícím vztahem:

$$f = \frac{1}{2\pi RC}$$

Hodnota kondenzátorů C (C10, C11, C12, C13) byla stanovena na 1nF. Z výše uvedeného vzorce dopočítáme hodnotu rezistoru R.

$$R = \frac{1}{2\pi fC}\ [\Omega]$$

$$R_{4,5,6,7} = \frac{1}{2\pi \cdot 40 \cdot 10^3 \cdot 1 \cdot 10^{-9}} = \underline{3978\ \Omega}$$

5.2.1 Měření rekonstrukčního filtru

Funkčnost rekonstrukčního filtru byla ověřena měřením, které bylo provedeno v domácích podmínkách na následujících měřících přístrojích:

Generátor signálu:

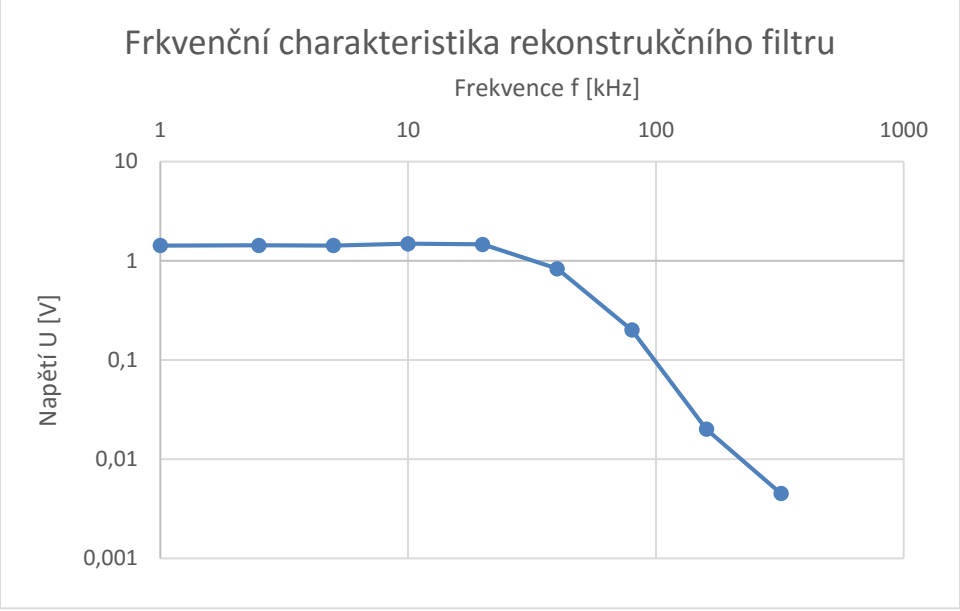
`RC oscilátor Tesla BK 124`

Voltmetr:

`Školní NF milivoltmetr Tesla BK 128`

f [kHz]	U_{out} [V]	U_{out} [dBuV]	ΔU [dB]
1	1,42	123	
2,5	1,43	123	
5	1,42	123	
10	1,48	123	
20	1,46	123	
40	0,93	119	4
80	0,2	106	13
160	0,02	86	20
320	0,0045	73	13

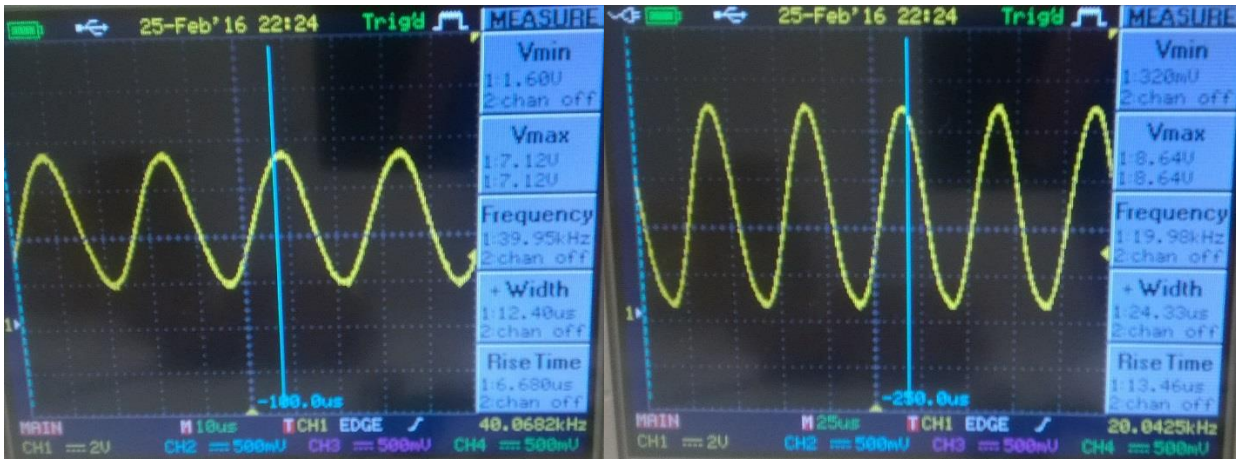
Tabulka 9 - Naměřené hodnoty charakteristiky filtru



Obr. 19 - Frekvenční charakteristika filtru

Jak je patrné z tabulky naměřených hodnot a grafu frekvenční charakteristiky, mezní frekvence filtru odpovídá zhruba hodnotě 40 kHz a pokles na mezní frekvenci je 4 dB. Následující pokles je 13, 20 a 13 dB na oktávu, tato hodnota má být dle řádu filtru rovna 18 dB na oktávu.

Nepřesnost tohoto měření může být způsobena tolerancí použitých součástek, ale pravděpodobnější je nepřesné nastavení frekvence na analogovém generátoru a celková nepřesnost obou použitých měřících přístrojů. Navzdory těmto nepřesnostem tvar grafu odpovídá předpokládané charakteristice filtru. Signál za rekonstrukčním filtrem již není schodovitého tvaru a jeho tvar již odpovídá spojitému sinusovému průběhu, jak je zobrazeno na obrázku níže.

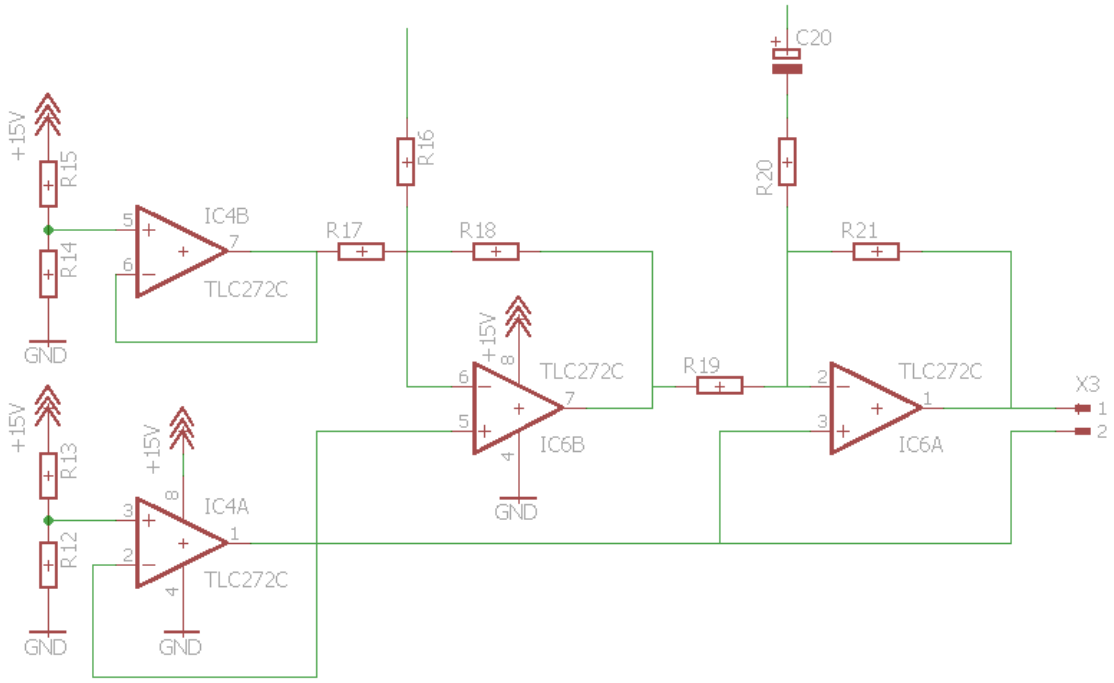


Obr. 20 - Signál za filtrem

5.3 Obvody pro realizaci a přičtení stejnosměrné složky

Poslední a pro mě možná nejnáročnější částí návrhu bylo navrhnout obvod, který na základě druhého výstupu DA převodníku vytvoří stejnosměrnou složku signálu, která bude následně přičtena ke střídavému výstupu generátoru.

Protože napájecí napětí výstupních obvodů je pouze kladné polarity, bylo nejprve nutné vytvořit referenční střed, který bude umožňovat posun stejnosměrné složky signálu jak do kladné tak i do záporné polarity. Referenční napětí je vytvořeno napěťovým sledovačem realizovaným operačním zesilovačem IC4A a odporovým děličem 1:1 připojeným ke kladné větvi napětí a zemi, tento dělič je tvořen rezistory R12 a R13 hodnoty 100 kΩ.



Obr. 21 - Zapojení výstupních obvodů

Druhý výstup DA převodníku pro nastavění stejnosměrné složky (výstup mikrokontroléru PB2) je připojen přes rezistor R16 na součtový zesilovač IC6B, jehož druhým vstupem je referenční napětí rovné polovině maximálního napětí na výstupu DA převodníku. Zdroj referenčního napětí je opět tvořen napěťovým sledovačem IC4B a odporovým děličem tvořeným rezistory R14 a R15. Maximální napětí výstupu DA převodníku je 3,3 voltu, polovina je tedy 1,65 voltu. Hodnota rezistoru R14 byla stanovena na 10kΩ a hodnota rezistoru R15 dopočtena:

$$R_{14} = R_{15}\left(\frac{U - U_R}{U_R}\right)\ [k\Omega]$$

$$R_{14} = 10\left(\frac{15 - 1{,}65}{1{,}65}\right) = 80{,}9\ k\Omega$$

Výstupní napětí součtového zesilovače je definováno vztahem uvedeným níže. Při zvolení rezistorů $R_{zp} = R_1 = R_2$ se výstupní napětí rovná záporné hodnotě součtu vstupních napětí.

$$U_{Out} = -\left(\frac{R_{zp}}{R_1} \cdot U_1 + \frac{R_{zp}}{R_2} \cdot U_2\right) \; [V]$$

$$U_{Out-min} = -\left(\frac{100}{56} \cdot 1{,}65 - \frac{100}{56} \cdot 0\right) = -2{,}95 \; V$$

$$U_{Out-max} = -\left(\frac{100}{56} \cdot 1{,}65 - \frac{100}{56} \cdot 3{,}3\right) = 2{,}95 \; V$$

Opačné polarity výstupního napětí bylo dosaženo tím, že referenční napětí operačního zesilovače IC6B je nastaveno na polovinu napájecího napětí, výstupní napětí DA převodníku se může pohybovat v rozmezí 0 – 3,3 voltu a referenční napětí druhého vstupu je 15 – 1,65 voltu, je tedy vzhledem k referenci 7,5 voltu rovno -1,65 voltu. Výsledná stejnosměrná složka může tedy nabývat hodnoty 7,5 +/- 2,95 voltu.

Toto napětí je následně přivedeno na výstupní součtový zesilovač IC6A, kde je sečteno se střídavou složkou signálu generátoru. Výstup z rekonstrukčního filtru je připojen k oddělovacímu kondenzátoru C20, který má za úkol odfiltrovat stejnosměrnou složku střídavého signálu. Výstup součtového zesilovače IC6A je celkovým výstupem generátoru signálu, který obsahuje jak stejnosměrnou tak střídavou složku.

Zemní potenciál výstupu generátoru není shodný se zemním potenciálem napájení, protože je připojen na referenční napětí 7,5 voltu. Proto toto zařízení nemůže být používáno v kombinaci se zařízeními, která mají společnou zem s napájením USB sběrnice. V takovémto případě není zaručena správná funkčnost zařízení a hrozí jeho poškození.

6 Závěr

V rámci diplomové práce byla vytvořena počítačová aplikace v jazyce C#, která komunikuje s mikrokontrolérem pomocí USB rozhraní. Tato aplikace je schopná na základě informací získaných z mikrokontroléru vypočítat frekvence, na kterých bude generátor pracovat, a vytvořit vzorek signálu zadané amplitudy a frekvence vhodný pro další zpracování mikrokontrolérem. Aplikace zobrazuje veškeré informace o parametrech DA převodníku a aktuálně nastaveném signálu.

Další vytvořenou součástí je software mikrokontroléru, ten umožňuje navázání USB komunikace s výše uvedenou aplikací a odeslání parametrů DA převodníku. Mikrokontrolér ukládá načtený vzorek signálu do paměti EEPROM, aby bylo možné generovat signál bez nutnosti připojení zařízení k počítači. Z paměti jsou cyklicky načítány jednotlivé vzorky signálu a převáděny na výstupní napětí pomocí DA převodníku.

Poslední částí praktického řešení je schéma zapojení a podle něj navržená deska plošného spoje. Zapojení obsahuje obvody potřebné pro funkci mikrokontroléru, zdroj napětí 15 voltů, rekonstrukční filtr a směšovací obvod pro přičtení stejnosměrné složky signálu.

6.1 Měření výstupní frekvence

Pro ověření funkce zařízení bylo provedeno kontrolní měření výstupní frekvence.

Nastavená frekvence	Naměřená frekvence	Odchylka	Odchylka v %
40 kHz	40 902 Hz	902 Hz	2,26%
20 kHz	20 440 Hz	440 Hz	2,2%
10 kHz	10 220 Hz	220 Hz	2,2%
5 kHz	5 110 Hz	110 Hz	2,2%
2,5 kHz	2 554 Hz	54 Hz	2,16%
1 kHz	1 022 Hz	22 Hz	2,2%
500 Hz	510,8 Hz	10,8 Hz	2,16%
250 Hz	256 Hz	6 Hz	2,4%
125 Hz	128,2 Hz	3,2 Hz	2,56%
80 Hz	82 Hz	2 Hz	2,5%
39,92 Hz	40,95 Hz	1,03 Hz	2,58%
19,96 Hz	20,48 Hz	0,52 Hz	2,6%

Tabulka 10 - Měření výstupní frekvence

Vzhledem k tomu, že se odchylka pohybovala v rozmezí 2,16% až 2,6%, jsem upravil časování přerušení DA převodníku. Hodnota registru *TCC0.PER* byla nastavena na 204. Po změně časování bylo provedeno další měření. Odchylka frekvence je pravděpodobně způsobena nepřesností interního oscilátoru.

Nastavená frekvence	Naměřená frekvence	Odchylka	Odchylka v %
40 kHz	40 080 Hz	80 Hz	0,002%
20 kHz	20 040 Hz	40 Hz	0,002%
10 kHz	10 020 Hz	20 Hz	0,002%
5 kHz	5 010 Hz	10 Hz	0,002%
2,5 kHz	2 505 Hz	5 Hz	0,002%
1 kHz	1 002 Hz	2 Hz	0,002%
500 Hz	501,05 Hz	1,05 Hz	0,0021%
250 Hz	251,17 Hz	1,17 Hz	0,009%
125 Hz	125,72 Hz	0,72 Hz	0,0057%
80 Hz	80,5 Hz	0,5 Hz	0,0063%
39,92 Hz	40,17 Hz	0,25 Hz	0,0063%
19,96 Hz	20,09 Hz	0,13 Hz	0,0065%

Tabulka 11 - Měření frekvence s upraveným časováním

Výsledek měření s upraveným časováním se ukázal jako výrazně přesnější, odchylka nepřesáhla 0,01%.

6.2 Shrnutí výsledků práce

V rámci práce bylo vytvořeno kompletní softwarové vybavení pro mikrokontrolér Atmel XMEGA128A4U a PC běžící na platformě Windows, vzájemně komunikující po USB sběrnici. Funkce softwaru byla ověřena na prototipové desce *MOD-20/A.Z* firmy *MODUŁOWO SP. Z O.O.* Při pokusu o oživení vlastní desky generátoru jsem narazil na problém s nahrávátím mikrokontroléru. Výrobce zřejmě nedodává mikrokontrolér s předinstalovaným bootloaderem pro USB rozhraní a ani s využitím programátorů mikrokontroléru Atmel (STK500 a USBASP) se mi nepodařilo mikrokontrolér naprogramovat. Je možné že při manipulaci došlo k jeho poškození. Proto byla veškerá měření provedena s mikrokontrolérem z uvedené prototipové desky.

Seznam použité literatury

[1] Atmel AVR 8-bit and 32-bit Microcontroller [online], http://www.atmel.com/products/microcontrollers/avr/default.aspx

[2] BRTNÍK, B., MATOUŠEK, D. Mikroprocesorová technika BEN - technická literatura, Praha 2011

[3] HAASZ, V., ROZTOČIL, J., NOVÁK, J., Číslicové měřicí systémy. ČVUT, Praha 2000

[4] HAASZ, V. SEDLÁČEK, M., Elektrická měření, Přístroje a metody. ČVUT, Praha 2005

[5] Humlhans, J.: Filtrace a aktivní filtry, časopis KTE č. 5 - 10, 1997

[6] Knihovna LUFA [online], http://www.fourwalledcubicle.com/LUFA.php

[7] LibUsbDotNet [online], http://libusbdotnet.sourceforge.net/V2/Index.html

[8] MATOUŠEK, D. Práce s mikrokontroléry Atmel AVR. Nakladatelství BEN - technická literatura, Praha2006

[9] Punčochář J.: OPERAČNÍ ZESILOVAČE v elektronice, BEN – technická literatura, Praha 1999

[10] USB.org USB 2.0 Specification [Revision 3.1, online] http://www.usb.org/developers/docs/usb_31_052016.zip

[11] VIRIUS, M., C# 2010 Hotová řešení, Computer Press, 2012

Seznam použitých symbolů a zkratek

AVR	-	Označení rodiny mikrokontrolérů firmy Atmel
CPU	-	Central Processing Unit, Centrální procesorová jednotka
DA (DAC)	-	Digital to Analog Converter, Digitálně analogový převodník
EEPROM	-	Electrically Erasable Programmable Read-Only Memory, elektricky mazatelná semipermanentní paměť
GUI	-	Graphical User Interface, grafické uživatelské rozhraní
LC	-	Induktor Capacitor, cívka a kondenzátor
MC	-	Micro controller, mikrokontrolér
PC	-	Personal Computer, počítač
RC	-	Rezistor Capacitor, rezistor a kondenzátor
RLC	-	Rezistor Induktor Capacitor, rezistor, cívka, kondenzátor
USB	-	Universal Serial Bus, univerzální sériová sběrnice

Obsah přiloženého CD

Application MC

Code	- zdrojový projekt
Hex	- binární soubory

Application PC

Code	- zdrojový projekt
Exe	- binární soubory

Library

LUFA	- knihovna LUFA
LibUsb	- knihovna LibUsbDotNet

Drivers

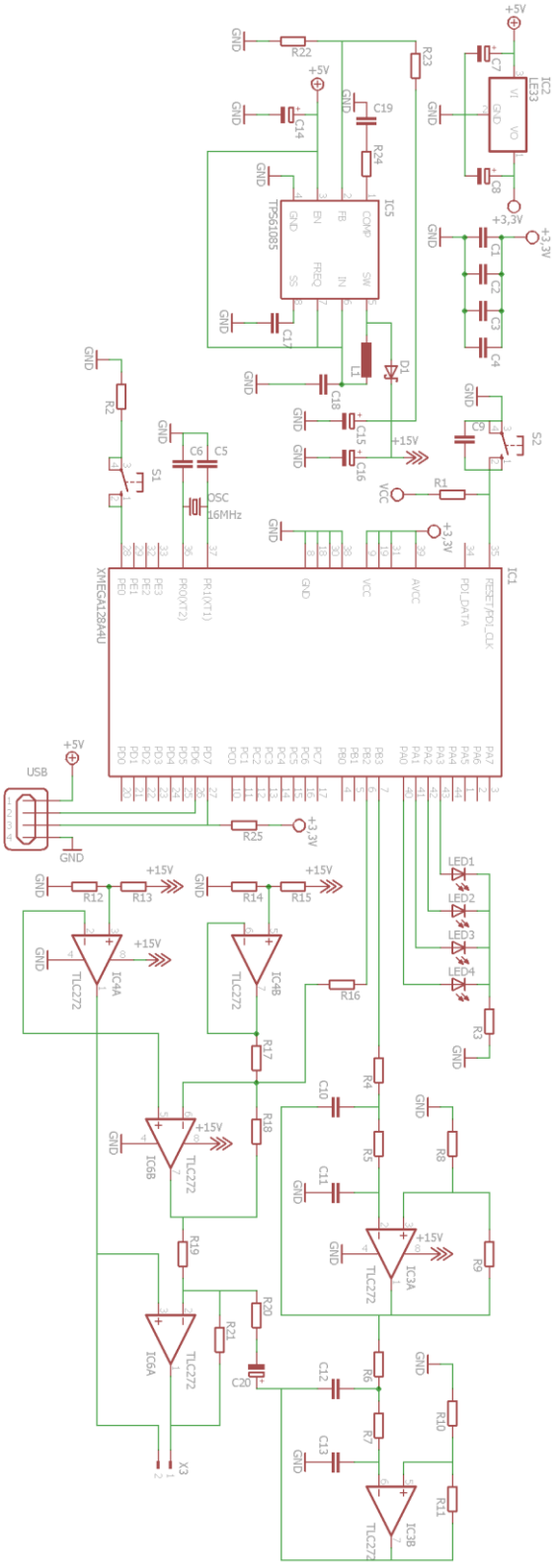
WinUSB	- ovladače LibUSB

Board - schéma a plošný spoj

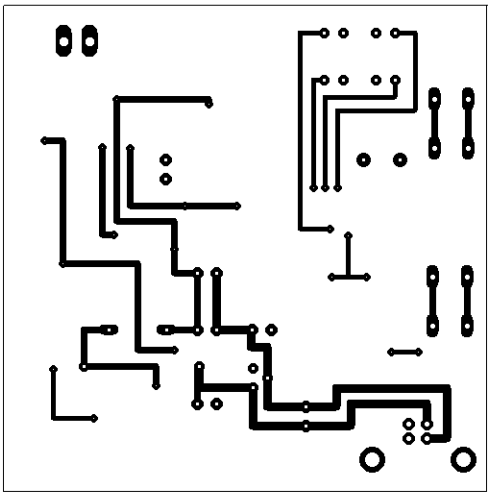
Document - text práce

Seznam příloh:

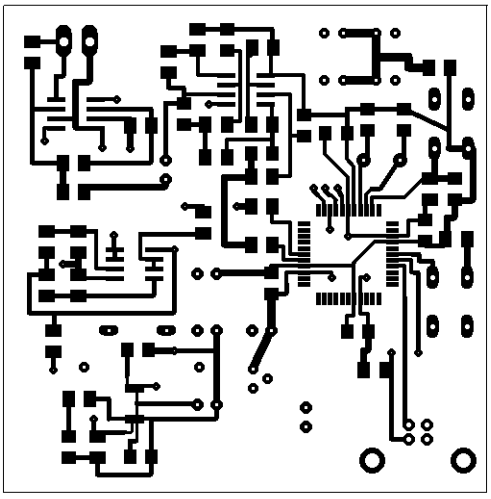
Schéma zapojení:



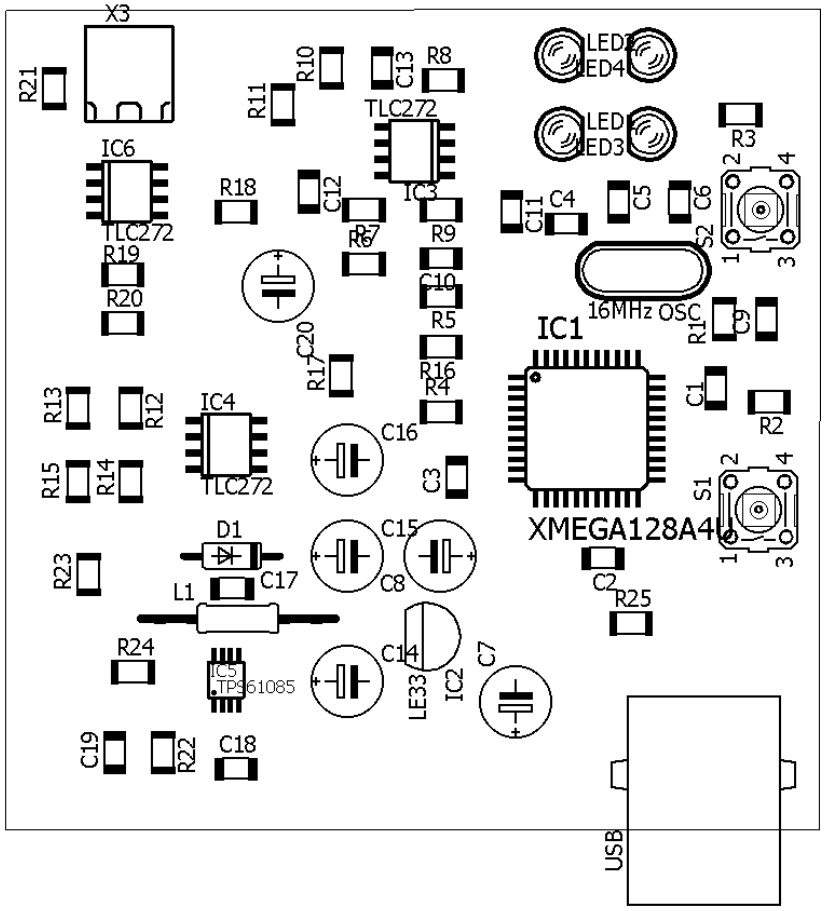
Plošný spoj- pájecí strana:



Plošný spoj- strana součástek:



Plošný spoj- rozložení součástek:



Seznam součástek:

IC1	XMEGA128A4U	L1	3,3uH
IC2	LE33		
IC3	TLC272	D1	3mm LED
IC4	TLC272	D2	3mm LED
IC5	TPS61085	D3	3mm LED
IC6	TLC272	D4	3mm LED
D1	PMEG2010AEH		
R1	10kΩ	C1	100nF
R2	250Ω	C2	100nF
R3	100Ω	C3	100nF
R4	4kΩ	C4	100nF
R5	4kΩ	C5	22pF
R6	4kΩ	C6	22pF
R7	4kΩ	C7	10uF
R8	10kΩ	C8	10uF
R9	12kΩ	C9	100nF
R10	10kΩ	C10	1nF
R11	1,5kΩ	C11	1nF
R12	100kΩ	C12	1nF
R13	100kΩ	C13	1nF
R14	82kΩ	C14	10uF
R15	10kΩ	C15	10uF
R16	56kΩ	C16	10uF
R17	56kΩ	C17	100nF
R18	100kΩ	C18	1uF
R19	56kΩ	C19	1nF
R20	120kΩ	C20	1uF
R21	56kΩ		
R22	18kΩ		
R23	200kΩ		
R24	51kΩ		
R25	1,5kΩ		