UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Selektivní voltmetr pro HDO

Václav Trpišovský

Bakalářská práce
2025

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Václav Trpišovský**
Osobní číslo:	**I21098**
Studijní program:	**B0714P060001 Aplikovaná elektrotechnika**
Téma práce:	**Selektivní voltmetr pro HDO**
Zadávající katedra:	**Katedra elektrotechniky**

Zásady pro vypracování

Má se navrhnout a realizovat selektivní voltmetr pro měření signálu HDO na principu číslicového filtru. Rozsahy: 1V , 3V a 10 V, frekvence HDO a frekvence 217 Hz. Indikace displejem a ručkovým měřidlem.

Rozsah pracovní zprávy: **40**
Rozsah grafických prací: **5**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Brtník, B , Matoušek, D. : Algoritmy číslicového zpracování signálů. BEN Praha, 2011, ISBN 978-80--7300-400-2.
Brtník, B.: Základní elektronické obvody. BEN Praha, 201, ISBN 978-80-7300-408-8.
Brtník, B.: Číslicové systémy. BEN Praha, 201, ISBN 978-80-7300-407-1.
Brtník, B , Matoušek, D. : Digital signal processing algorithms: Implementation of digital filters in C using microcontroller ATxmega16. Lambert Academic Publishing, Saarbrucken, 2012. ISBN 978--3847344117.

Vedoucí bakalářské práce: **Ing. Bohumil Brtník, Dr.**
Katedra elektrotechniky

Datum zadání bakalářské práce: **15. prosince 2023**
Termín odevzdání bakalářské práce: **10. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

doc. Ing. Jan Pidanič, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 24. ledna 2024

Prohlášení autora

Prohlašuji:

Práci s názvem Selektivní voltmetr pro HDO jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 16. 5. 2025

Václav Trpišovský

Poděkování

Vážím si podpory vedoucího práce dr. Brtníka, rovněž děkuji rodině za pomoc při fyzické realizaci zařízení a za zázemí, bez něhož by práce nemohla vzniknout. Rovněž děkuji dr. Pidaničovi za motivaci a pozitivní přístup.

Anotace

Obsahem této práce je popsání teoretického principu selektivního voltmetru pro signál HDO a implementace tohoto zařízení. Systém řízení spotřeby v elektrické síti vyžaduje dostatečné šíření nízkofrekvenčního signálu, jež je pomocí tohoto přístroje možné ověřit. Voltmetr pro izolaci nosné frekvence používá úzkopásmový číslicový filtr, konkrétně inverzní Čebyševův. Navržená deska plošných spojů „ATMEGA DSP KIT“, založená na oblíbeném mikrokontroléru AVR, může sloužit i pro jiné začátečnické experimenty v oblasti číslicového zpracování signálu.

Klíčová slova

selektivní voltmetr, číslicové zpracování signálu, číslicový filtr, AVR, hromadné dálkové ovládání

Title

Selective Voltmeter for RCS

Annotation

This thesis covers the theoretical workings of a digital-filter selective voltmeter for HDO (ripple control system) signals and describes a practical implementation of the device. This system for controlling loads on the electric grid relies on sufficient propagation of tone-frequency signals across the network, which this device can be used to measure. The voltmeter uses a narrow-band digital filter, specifically the Chebyshev-II type. Its printed circuit board design, “ATMEGA DSP KIT”, based on a popular AVR microcontroller, allows for various introductory experiments into the field of digital signal processing.

Keywords

selective voltmeter, digital signal processing, digital filter, AVR, ripple control system

Obsah

Seznam zkratek

HDO	Hromadné dálkové ovládání
RCS	Ripple Control System
LC	Kombinace cívky a kondenzátoru
RC	Kombinace rezistoru a kondenzátoru
DV	Dlouhé vlny (elektromagnetické pásmo)
SV	Střední vlny (elektromagnetické pásmo)
KV	Krátké vlny (elektromagnetické pásmo)
NPN	Dvojpřechodový polovodič s oblastmi v pořadí N, P, N (typ BJT)
PNP	Dvojpřechodový polovodič s oblastmi v pořadí P, N, P (typ BJT)
FET	Field Effect Transistor
MOSFET	Metal-Oxide-Semiconductor Field Effect Transistor
BJT	Bipolar Junction Transistor
RF	Rádiová frekvence / radiofrekvenční
MCU	Mikrokontrolér (Microcontroller Unit)
AVR	obchodní známka Atmel/Microchip pro architekturu MCU (není zkratka)
LO	Local Oscillator (součást heterodynních systémů)
A/D	Analogový/digitální
ADC	Analogově-digitální převodník
D/A	Digitální/analogový
DAC	Digitálně-analogový převodník
FM	Frekvenční modulace
AM	Amplitudová modulace
PCM	Pulse Code Modulation
LUT	Look-Up Table
S/s	Samples per second (vzorky za sekundu, přidávají se metrické předpony)
PWM	Pulse Width Modulation
FPGA	Field-Programmable Gate Array
ČR	Česká republika
ČEZ	ČEZ, akciová společnost (dříve České energetické závody, koncern)
EG.D	EG.D, akciová společnost (Electricity & Gas Distribution, dříve E.ON)
PRE	Pražská energetika, akciová společnost
ČSN	označení české technické normy (dříve Československá státní norma)
DIN	označení německé technické normy (Deutscher Institut für Normung)
EN	Evropská norma
VA	voltampér (fyzikální jednotka zdánlivého výkonu)
V-A	volt-ampérový (zobrazující vztah napětí a proudu; zkratka zvolena namísto často používaného „VA“ pro rozlišení od voltampéru)
ČSSR	Československá socialistická republika
KEP	krajský energetický podnik

RCGE	Regional Group Continental Europe – – organizace spravující síť Continental Europe Synchronous Area
IoT	Internet of Things
NB-IoT	Narrowband Internet of Things
LTE-M	Long-Term [Mobile Network] Evolution – Machine [Communication]
LoRa	Long-Range (IoT komunikační protokol)
GSM	Global System Mobile
LoRa	Long Range
ČK CIRED	Český komitet Congres International des Reseaux Electriques de Distribution
JE	jaderné elektrárny
PE	parní elektrárny
PPE	paroplynové elektrárny
PSE	plynové a spalovací elektrárny
VE	vodní elektrárny
PVE	přečerpávací vodní elektrárny
FVE	fotovoltaické elektrárny
VTE	větrné elektrárny
vvn	velmi vysoké napětí
vn	vysoké napětí
nn	nízké napětí
LAN	Local Area Network
OOK	on/off keying
OZ	operační zesilovač
RAM	Random Access Memory
px	pixel
STN	Super Twisted Nematic
CoG	Chip on Glass
TXC0	Transmission Complete for USART Unit 0
U(S)ART	Universal (Synchronous/)Asynchronous Serial Receiver/Transmitter
UCSR0A	USART 0 Control & Status Register A
STC	Serial Transfer Complete
F-F	female-female (propojovací prvek s oběma konektory typu zásuvka)
LSB	Least Significant Bit; bit nejnižšího řádu, zpravidla 8. v bajtu
MSB	Most Significant Bit; bit nejvyššího řádu, zpravidla 1. v bajtu
XOR/EOR	Exclusive OR; bitová operace exkluzivní disjunkce
DDRC	Data Direction Register for Port C
AREF	Analog Reference Voltage
gcc	GNU C Compiler
DIP	dual in-line package
RMS	root-mean-square
GUI	graphical user interface

Seznam obrázků

Seznam tabulek

Úvod

Spolehlivost a efektivní využití elektrické sítě je jedním z předpokladů fungování moderní společnosti. Klíčové je v tomto ohledu neustálé vyrovnávání výroby a spotřeby energie a jedním z nástrojů pro udržení této rovnováhy je systém hromadného dálkového ovládání. Již více než půl století se za tímto účelem používá komunikace po elektrickém vedení, kdy jsou impulsy modulované na nosné frekvenci v řádu stovek Hz superimponovány na síťovém napětí. Přijímače tyto tónové signály detekují a dekódují tak povely, dle kterých pak spínají stykače a připojují k síti různé spotřebiče.

Signál HDO v elektrické síti podléhá rušení a útlumu, pro testování jeho šíření je proto nutné použít měřicí zařízení citlivé pouze na konkrétní frekvenci. Jedná se o selektivní voltmetr, jež pomocí úzkopásmového filtru tento kmitočet izoluje ze spektra a naměří jemu odpovídající amplitudu. Zpočátku se pro tyto účely využívaly analogové filtry ze základních pasivních součástek, později byly realizovány pomocí zpožďovacích členů a zesilovačů se zesílením voleným dle polynomických řad. Takový systém je nyní možné simulovat za pomoci digitálních obvodů, jedná se pak o takzvané číslicové zpracování signálu.

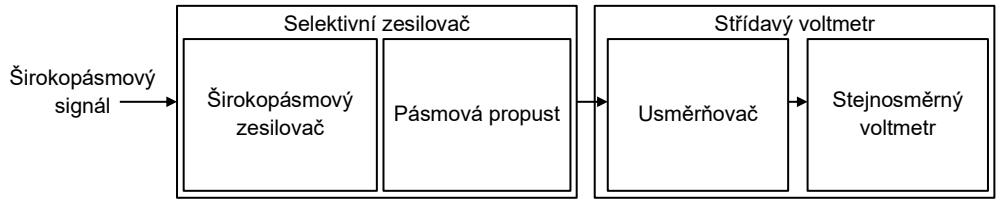
Ve frekvenčním pásmu využívaném HDO je přitom možné selektivní voltmetr uskutečnit bez specializovaných integrovaných obvodů. Po vzoru profesora Matouška proto byl použit osmibitový mikrokontrolér architektury AVR. V praktické části je popsán návrh hardwarového i softwarového řešení voltmetru HDO. U hardwarového řešení se dbá na přívětivost pro začátečníky, použitý mikrokontrolér ATmega328P patří mezi nejoblíbenější a nejlépe zdokumentované, rovněž konstrukce počítá s možností využít velmi dostupné technologie jednostranné desky plošných spojů s THT součástkami. Název navrženého modulu, „ATMEGA DSP KIT“, odpovídá jeho určení pro experimentální činnost, podobně jako univerzálnější desky Arduino pak přes malé rozměry umožňuje využití v různých projektech, často bez externích součástek.

Výsledné zařízení je vybaveno grafickým displejem a přívětivým uživatelským rozhraním. Kromě zadaných výstupů, kterými jsou displej a ručkový ukazatel, pak obsahuje navíc datový výstup pro možnost analýzy dat a ladění vyvíjeného programu. V rámci firmwaru byly rovněž vytvořeny knihovny pro komunikaci procesoru s vybranými periferiemi, které lze například použít při případné přestavbě zařízení na jiné.

1 Teoretická část

1.1 Princip selektivního voltmetru

Selektivní voltmetr je měřicí přístroj pro střídavé napětí pouze v části kmitočtového spektra. Tyto voltmetry dělíme podle šířky měřeného pásma na úzkopásmové a širokopásmové, a podle jeho nastavitelnosti na pevné a laditelné. Všechny tyto typy se zpravidla skládají z tentýchž hlavních článků: selektivního zesilovače a střídavého voltmetru.



Obrázek 1 – Blokové schéma selektivního voltmetru

Úloha selektivního zesilovače spočívá v omezení šířky pásma vstupního signálu a úpravy jeho amplitudy na úroveň odpovídající vlastnostem následujících článků, obvykle v řádu jednotek voltů. Vzhledem k povaze vstupního signálu, kterým může být například síťové napětí nebo anténa, může koeficient zesílení u různých voltmetrů dosahovat hodnot v rozsahu mnoha řádů, lze se setkat i s hodnotami pod 1, kdy článek pracuje jako zeslabovač. Je-li požadován přepínatelný rozsah měření, je realizován nastavitelným zesílením tohoto článku. Výstupem selektivního zesilovače je střídavé napětí, jehož amplituda odpovídá amplitudě signálu v měřeném pásmu. V závislosti na topologii se jako tento článek používá přímý zesilovač nebo heterodyn; jeho výstup, přímo úměrný laděné složce vstupního signálu, tedy může mít stejnou frekvenci jako měřený signál nebo může podléhat zdvihu. Tento signál následně putuje do usměrňovače a měřícího přístroje: diodový detektor obálky a stejnosměrný voltmetr, v případě digitálního DSP systému, pak algoritmus, jež dokáže vypočítat efektivní hodnotu algoritmem RMS.

1.1.1 Přijímač s přímým zesílením

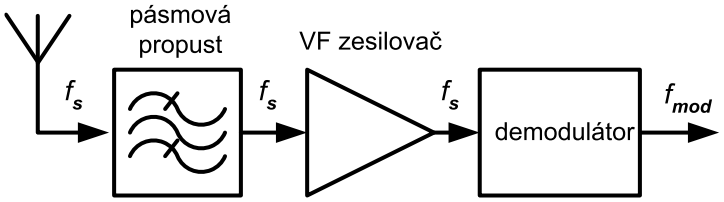
Jednou z topologií je systém přímého zesílení, používaný v radiopřijímačích v první polovině 20. století. Laditelný LC obvod je buzen signálem z antény a nejvyšší propustnost má v rezonanční frekvenci. Výsledný signál tedy obsahuje především složku odpovídající naladěné frekvenci, a je přiváděn na vstup zesilovače schopného pracovat v pásmu, ve kterém se zvolená nosná vlna může vyskytovat. Pro tento účel se tedy volí širokopásmový zesilovač s propustí v šíři celého RF pásma (např. DV, SV, KV).

Nevýhodou tohoto postupu je malá selektivita, ze které vychází nutnost zařadit několik takovýchto článků kaskádně[1]. V takovém případě má každý z nich vlastní LC obvod, a ty je pro správnou funkci přijímače nutno naladit na stejnou frekvenci. Rozladění jednotlivých stupňů mělo za následek jiné než konstrukční zesílení a nižší citlivost. U selektivního voltmetru je ovšem kritické konzistentní zesílení absolutní úrovně

vstupního signálu, aby při přelaďování nedošlo k tzv. chybě zesílení. Z tohoto důvodu lze analogový přímý zesilovač použít pouze pro selektivní voltmetr bez laditelné frekvence.

Pro využití v selektivním voltmetru HDO je ovšem tato nevýhoda nepodstatná, protože signál na vstupu má již vysokou amplitudu a je proto potřeba použít zeslabovač. Nosná frekvence HDO signálu je i tak v nízkofrekvenčním pásmu, její zesílení je tedy možné i s nejjednoduššími zesilovači. V souvislosti s číslicovým zpracováním signálu je nízká frekvence nosné vlny rovněž výhodnou, dá se totiž přímo převést na číslicový průběh vzorkováním na frekvencích v řádu jednotek kHz, dosažitelných i s pomocí jednoduchých MCU, například těch s 8bitovou architekturou AVR, jež byly využity v praktické části této práce.

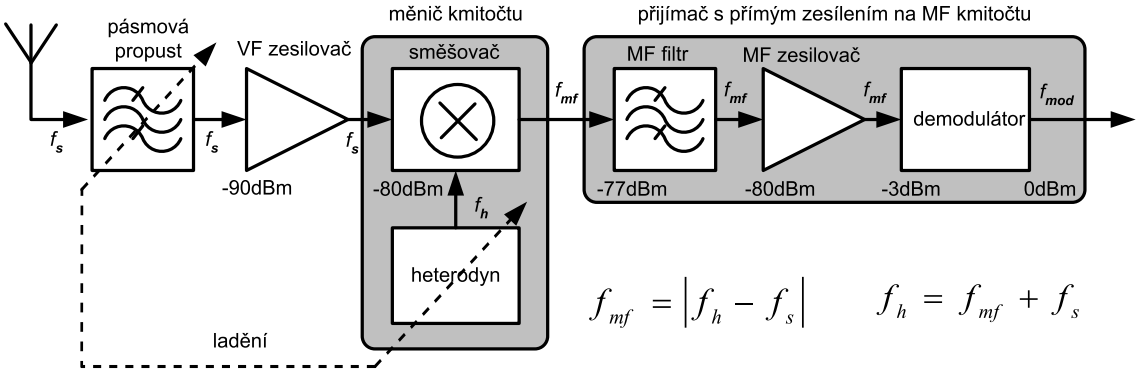
Dalším důležitým parametrem je selektivita vybraného filtru, aby nebyly demodulovány i ostatní frekvence v rozvodné síti. U úzkopásmových selektivních voltmetrů se jako pásmová propust používá několik rezonančních LC článků, případně vysoce selektivní číslicové filtry.



Obrázek 2 – Blokové schéma přímozesilujícího AM demodulátoru [2]

1.1.2 Superheterodyn

Superheterodyn je obvod založený na mísení signálů různých frekvencí – operaci založení na nelineárním členu, při které vzniká signál na součtové a rozdílové frekvenci. Rovnice ideálního směšovače (násobiče) signálů vychází z trigonometrické identity $\cos\omega_1\cos\omega_2 = \frac{1}{2}\cos(\omega_1 - \omega_2) + \frac{1}{2}\cos(\omega_1 + \omega_2)$, popisující vznik stejně velkých složek součtové i rozdílové frekvence z navzájem násobených signálů. [3] Takový směšovač se v blokových schématech značí symbolem ⊗ a v praxi se řeší cyklickým zapojením čtyř diod, tzv. kruhovým modulátorem. Vznik rozdílové frekvence při mísení umožňuje zdvih frekvence modulačního signálu do nižšího pásma na tzv. mezifrekvenci neboli IF (*intermediate frequency*). Ladění se provádí změnou frekvence místního oscilátoru tak, aby se obraz laděné stanice přesunul na mezifrekvenci. Tento signál je filtrován kaskádou filtrů pro pevnou frekvenci, odpadá tedy nutnost ladit je samostatně; návrh demodulátoru pak zjednodušuje větší relativní šířka pásma přijímaného signálu. [4] Tímto se superheterodyn stal nejběžnější topologií přijímačů rádiových signálů ve 20. století.



Obrázek 3 – Blokové schéma superheterodynového AM přijímače [2]

1.2 Číslicový filtr

1.2.1 Úvodem k číslicovým filtrům

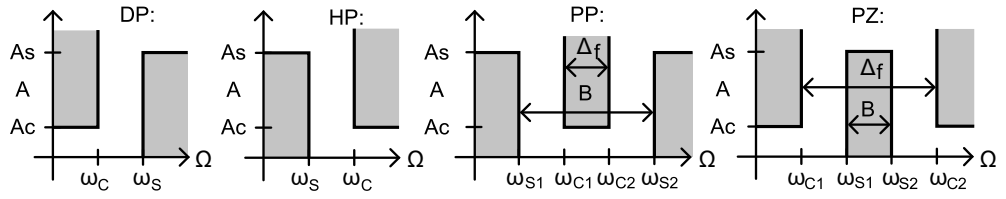
Selektivní voltmetr popsaný v této práci pracuje na bázi číslicového filtru. Číslicové neboli digitální filtry jsou základním prvkem systémů DSP (*digital signal processing* – číslicového zpracování signálu) a jsou zcela realizovány výpočetním algoritmem, zpravidla implementovaném v dedikovaném čipu nebo programovatelném procesoru.

Číslicové filtry fungují se vzorkovaným a kvantovaným signálem, pro práci se spojitými veličinami je tedy na jeho vstupu a/nebo výstupu nutné využít A/D, resp. D/A převodník. Nevýhodou je tak přidaná složitost oproti analogovým filtrům jako RC nebo LC, s vysokou integrovaností (spolu s ADC i DAC[5], případně i procesoru a ostatních částí MCU[6]) a odpovídajícím poklesem velikosti a ceny digitálních obvodů v posledních desetiletích se ovšem počet jednotlivých elementárních součástek stává nerelevantním kritériem. U aplikací s nižšími požadavky na výkon je možné DSP implementovat i bez specializovaného hardwaru, například v MCU nebo skriptu MATLAB. Konečná vzorkovací frekvence ovšem omezuje pásmo zpracovatelných signálů dle Nyquistovy frekvence a omezená bitová hloubka pak určuje dosažitelný práh šumu. Zatímco nízkofrekvenční DSP (např. v audio zařízeních) je běžnou a levnou záležitostí, vzorkování signálu na rádiových frekvencích bylo proto až donedávna praktické jen ve specializovaných zařízeních. Ze vzorkovacího teorému rovněž vyplývá, že signál je před A/D převodem nutné analogovým filtrem pásmově omezit pro vyhnutí se aliasingu, není-li cílem úmyslné podvzorkování a zachycení překrývajících se frekvenčních pásem zrcadlených kolem nuly a Nyquistovy frekvence. S podobnými problémy se setkáváme u D/A převodníků, kde je ze signálu *sample-and-hold* nebo PWM nutné odstranit nežádoucí vysokofrekvenční složky.

Naopak nastavitelnost parametrů filtru uvnitř softwaru umožňuje, aby jedna hardwarová realizace, zpravidla jako integrovaný obvod s minimálním počtem podpůrných součástek, pokryla potřeby různých zákazníků a aplikací, a tím snížila cenu zařízení díky masové výrobě. Citlivé obvody je možné v rámci výrobních tolerancí odladit automaticky, není třeba k tomuto účelu využívat fyzické trimry nebo varikapy. Při návrhu číslicových filtrů lze použít přesné softwarové modely na počítači (pomocí vlastních skriptů např.

v Pythonu, případně hotových DSP nástrojů obsažených v programech MATLAB, Micro-Cap nebo MPLAB), jejichž simulace funkce filtru naprosto odpovídá výpočtům prováděným ve finálním hardwaru, čehož v případě analogových systémů s širokými tolerancemi součástek, parazitními vlastnostmi, tepelnou závislostí a rušivými vlivy není možné přesně docílit. Číslicové filtry tedy dosahují vysoké flexibility a analogové filtry často předčí v ceně i velikosti, a s rozvojem polovodičové techniky jsou využívány při stále vyšších frekvencích.

Návrh filtru vždy začíná definicí kýžených parametrů. Požadavky na analogový i číslicový filtr lze nejlépe znázornit pomocí grafů frekvenční závislosti jejich přenosu případně jako na obrázku 4 přijatelnými hodnotami útlumu (A, *attenuation*) dle úhlové frekvence signálu ω. Podle posloupnosti, šířky a vzdáleností propustných a nepropustných pásem volíme typ filtru – dolní propust, horní propust, pásmová propust nebo pásmová zádrž. Kmitočtem f_C (*corner frequency*, též *passband frequency*), vyjádřeným jako úhlová frekvence ω_C, označujeme hranici propustného pásma, ve kterém očekáváme nulový či nepatrný útlum, úhlový kmitočet ω_S pak hranici nepropustného pásma (*stopband*), ve kterém se očekává téměř či zcela úplný útlum signálu. Úrovně A_C, resp. A_S označují maximální a minimální útlum, který od filtru vyžadujeme v propustném, resp. nepropustném pásmu. Od navrženého filtru tedy očekáváme, že se kmitočtová charakteristika vyhne stínovaným oblastem a bude ležet v bílém, tzv. tolerančním poli filtru. Její skutečný průběh pak závisí na vybrané aproximaci, volbě koeficientů a přesnosti, se kterou pracuje použitý hardware.



Obrázek 4 – Základní typy frekvenčních filtrů [7]

1.2.2 Laplaceova a Z-transformace

Pro efektivní číslicové zpracování signálu bylo potřeba řady matematických poznatků, principů a abstrakcí. Při návrhu analogového filtru je bráno v potaz frekvenční spektrum signálu a přenos filtru ve frekvenční oblasti, u číslicového filtru by převod vzorkovaného signálu do frekvenční oblasti a zpět byl ovšem výpočetně náročný a často tedy není prakticky využitelný. V návrhu systému DSP je tedy nutno využít matematické nástroje, kterými se vyřeší požadavky na frekvenční přenos filtru pomocí relativně jednoduchých rovnic v oboru $n \in \mathbb{Z}$, tedy oblasti diskrétních vzorků.

Laplaceova transformace, dána integrálem

$$F(p) = \mathcal{L}\{f(t)\} = \int_0^\infty f(t)e^{-pt}dt$$

převádí funkci $f(t)$ z oblasti spojitého času na funkci $F(p)$ v oblasti komplexní frekvence.

Podobně z-transformace, daná sumou

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]z^{-n}\,; n \in \mathbb{Z}$$

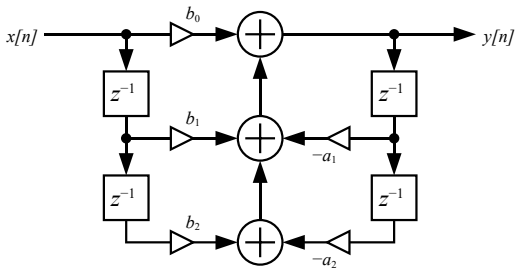
převádí funkci $x(n)$ z oblasti diskrétního času (vzorků) na funkci $X(z)$ v oblasti komplexní frekvence.

Jako článek z^{-1} ve schématech číslicových filtrů (setkáváme se též se symbolem T) označujeme paměťový prvek, který vrací hodnotu opožděnou o vzorkovací periodu, jeho kaskádovým zapojením se tedy realizuje posloupnost předchozích hodnot v registru. Například pro vstupní registr $x(n)$ získává zpožďovací kaskáda o délce m hodnoty $x(n), x(n-1), \dots x(n-m)$. Ty je možné sečíst po vynásobení váhovými koeficienty $a_0, a_1, \dots a_m$[1] a vytvořit tak filtr FIR (*finite impulse response*), provádějící konvoluci signálu. Mimoto je možné zpožďovací kaskádu realizovat i z výstupní hodnoty $y(n), y(n-1), \dots y(n-m)$ a vytvořit tak článek, pro jehož přenos platí:

$$F(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{a_0 + a_1 z^{-1} + \cdots + a_m z^{-m}}$$

Koeficienty se zpravidla normují tak, aby $a_0 = 1$.

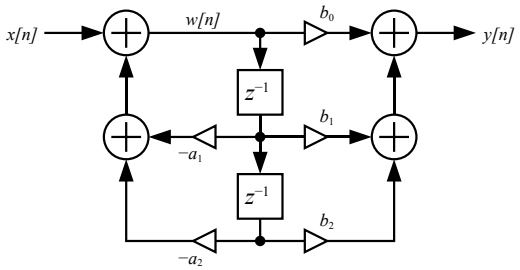
Filtr 2. řádu, tzv. bikvad, tedy lze znázornit následujícím diagramem:



Obrázek 5 – Schéma číslicového filtru IIR, přímá forma I [8]

[1] Označení váhových koeficientů zpětnovazebních článků filtru písmenem $a_0 \dots a_m$ je normou v české [12] [7] i zahraniční literatuře. [13] Jako alternativní zápis byla nalezena v kódu doktora Matouška [7] řada proměnných `kzv` nebo `kyp`, jež jsou pro zápis rovnice nevhodné a dávám tedy zde i v diagramech přednost standardnímu zápisu. Některé prameny [7] ovšem stejným způsobem ($a_0 \dots a_n$) označují normalizované koeficienty Butterworthových polynomů $B_n(s)$, ze kterých se pro výpočet koeficientů zpětnovazebních filtrů vychází – ty ovšem platí v doméně Laplaceova s-prostoru, nikoli z-prostoru diskrétních signálů. Pro jednoznačnost jsem se rozhodl následovat praktiky Wikipedie [36] a při uvádění Butterworthových polynomů se vyhnout jejich označování písmeny.

Transpozicí na přímou formu II můžeme snížit náklady na paměť, jelikož je nyní potřeba jen jedna řada zpožďovacích členů:



Obrázek 6 – Schéma číslicového filtru IIR, přímá forma II [9]

Filtry pracující se vzorky skutečných veličin zpracovávají tzv. pravostranný signál – ten začíná v určitém bodě (zpravidla $n = 1$ v matematických modelech nebo $n = 0$ v praktických implementacích), a hodnoty $x(n)$ a $y(n)$ před tímto bodem nejsou definované, obvykle se pro ně uvažuje nulová hodnota. Signál po tomto bodě může dosahovat teoreticky nekonečné délky ($n \to \infty$) a nabývá konečných hodnot $x(n) \leq x_{\max}$. Od filtru očekáváme, že jeho výstup bude rovněž konečný, tedy pro libovolně velké $n \in \mathbb{N}$ nikdy nepřekročí mez $y_{\max}$ – lze zpracovat jen konečně velká čísla, než dojde k saturaci aritmetiky.[10]

Zatímco filtry typu FIR postrádají zpětnou vazbu a jsou tak proti růstu do nekonečna, tedy nestabilitě, odolné, v číslicovém filtru typu IIR, který dosahuje vyšší efektivity při stejném řádu, dochází ke zpětné vazbě a ta může mít za následek růst výstupu nade všechny meze.[11] Stabilitu filtru můžeme ověřit jednoduchou simulací v časové doméně tím, že na jeho vstup přivedeme elementární jednotku diskrétního signálu, tzv. jednotkový (Heavisideův) skok $1(t)$ vycházející ze vztahu $1(t) = \begin{cases} 0 \,\forall\, t < 0 \\ 1 \,\forall\, t > 0 \end{cases}$; uvažovaná hodnota v bodě $t = 0$ závisí na interpretaci; nebo jeho derivaci, jednotkový (Diracův) impuls $\delta(t)$. Nejprve si představíme základní vztahy pro práci s těmito funkcemi.

Diracův impuls $\delta(t) = \frac{\mathrm{d}}{\mathrm{d}t} 1(t)$ dosahuje nulové hodnoty pro všechna $t \neq 0$ a jeho hodnota v $t = 0$ nelze vyčíslit, jedná se totiž o interval nulové délky s určitým integrálem rovným jedné. V oblasti vzorkovaného signálu, kde může čas dosahovat pouze celočíselných hodnot n (násobků vzorkovací periody), ovšem neexistuje toto omezení a Diracův impuls v oblasti n lze definovat jako $\delta(n) = \begin{cases} 1; & n = c \\ 0 \,\forall\, n \neq c \end{cases}$ (čas skoku c zpravidla volíme 0 nebo 1 dle aritmetických konvencí). Signál $\delta(n)$ můžeme uvažovat na vstupu číslicového filtru a simulací získat impulsní charakteristiku, případně na jeho vstup zavést funkci $x(n) = 1(n)$ a získat tak přechodovou charakteristiku, tj. odezvu na jednotkový skok. Oba tyto průběhy lze použít jako elementární jednotky signálu. Sumací spočetného množství kopií Diracovy i Heavisideovy funkce posunutých o celý počet vzorků a vynásobených reálnými koeficienty odpovídajícími aktuální, resp. diferenciální

hodnotě libovolné diskrétní funkce $x(n)$ lze totiž dosáhnout přesné reprezentace signálu $x(n)$[2]:

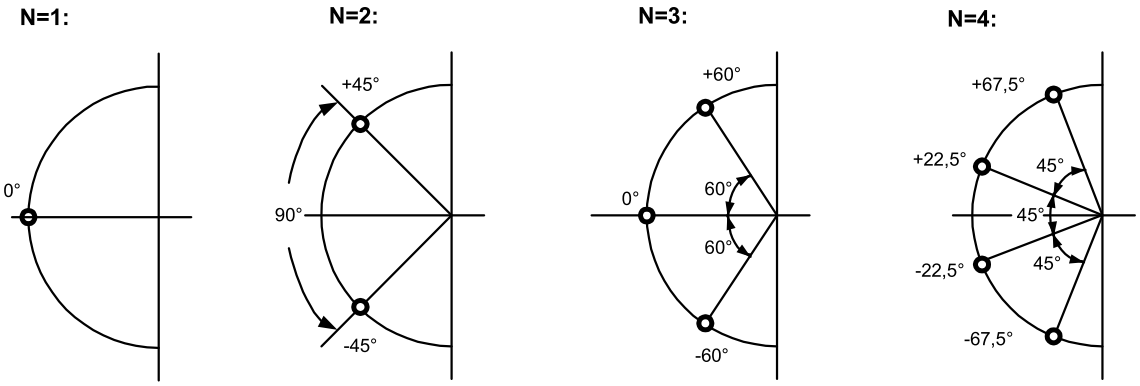
$$x(n) = \sum_{i=0}^{\infty} x(i)\delta(n-i) = \sum_{i=0}^{\infty} [x(i) - x(i-1)]1(n-i) = \sum_{i=0}^{\infty} \Delta x(i)1(n-i)$$

Z tohoto vztahu a faktu, že součet konečného počtu omezeně velkých sčítanců je vždy konečný, vyplývá, že číslicový filtr je stabilní, tedy absolutní hodnota jeho výstupu nikdy nepřekročí konečnou mez, právě tehdy, pokud se jeho odezva na Diracův impuls i Heavisideův skok po určité době ustálí a nebude růst nad všechny meze. Tím jsme dokázali, že využitím snadno generovatelného signálu v simulačním softwaru (za předpokladu, že jeho aritmetická omezení odpovídají reálné implementaci a amplituda vstupu je omezena na úroveň, jež předejde přetečení) lze otestovat, zda přenos IIR filtru náleží do tzv. tolerančního pole – oblasti, která garantuje jeho stabilitu.

1.2.3 Filtrační aproximace, vliv kvantování

Existuje několik matematických nástrojů, kterými můžeme určit polohu pólů, jež nejlépe odpovídají požadavkům určených tolerančním polem filtru nebo napodobením analogového prototypu.

Pravděpodobně nejpoužívanější aproximací je Butterworthova, jejíž předností je plochá charakteristika v propustném pásmu, zatímco v nepropustném pásmu klesá asymptoticky s přímkou o sklonu -20 dB/dekádu na řád filtru. Jedná se o filtr nejlépe aproximující analogové články, z dostupných možností nabízí nejplošší charakteristiku v propustném pásmu a je striktně monotónní s nepříliš strmým, ovšem předvídatelným klesáním přenosu v nepropustném pásmu.



Obrázek 7 – Umístění pólů Butterworthova filtru řádů 1~4 [12]

Jeho póly, jejichž počet je dán řádem filtru, leží na levé půlkružnici se středem v počátku komplexní frekvenční roviny a úhlově jsou vzdálené o $\frac{180°}{n}$. Díky tomu, že jsou jejich pozice souměrné podle reálné osy a tvoří tedy komplexně sdružené páry, mají koeficienty

[2] Pro zjednodušení uvádím pouze případ pravostranné funkce, sumací se zleva neomezenými a/nebo zprava omezenými hodnotami i lze docílit rekonstrukce signálu konečné délky, levostranného i oboustranného.

Butterworthových polynomů reálné hodnoty a lze je vypočítat trigonometrickými funkcemi

$$B_n(s) = \left.\begin{matrix}\text{pro sudá } n\text{: } 1 \\ \text{pro lichá } n\text{: } (s+1)\end{matrix}\right\} \cdot \prod_{k=1}^{\left\lfloor \frac{n}{2} \right\rfloor} [s^2 + 2\cos(\psi_k)\, s + 1]\,,$$

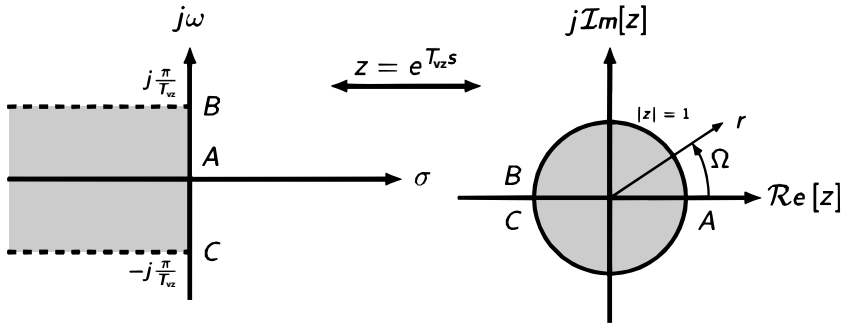
kde ψ_k je úhel odpovídajících pólů, například pro filtr 2. řádu $\psi_1 = \pm 45°$, polynom tedy vychází $B_2(s) = s^2 + \sqrt{2}s + 1$. Alternativou je využít tabelovaných hodnot v přesném nebo desetinném tvaru, uvádím tabulku polynomů pro prvních 8 řádů:

Tabulka 1 – Polynomy Butterworthovy aproximace s vyčíslenými koeficienty [13]

n	$B_n(s)$
1	$(1 + s)$
2	$(1 + 1{,}414s + s^2)$
3	$(1 + s)(1 + s + s^2)$
4	$(1 + 0{,}765s + s^2)(1 + 1{,}848s + s^2)$
5	$(1 + s)(1 + 0{,}618s + s^2)(1 + 1{,}618s + s^2)$
6	$(1 + 0{,}518s + s^2)(1 + 1{,}414s + s^2)(1 + 1{,}932s + s^2)$
7	$(1 + s)(1 + 0{,}445s + s^2)(1 + 1{,}247s + s^2)(1 + 1{,}802s + s^2)$
8	$(1 + 0{,}390s + s^2)(1 + 1{,}111s + s^2)(1 + 1{,}663s + s^2)(1 + 1{,}962s + s^2)$

Po vybrání koeficientů se rovnice převede do prostoru diskrétního času z (obr. 8), čímž vznikne již známý lomený výraz

$$F(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_n z^{-n}}{a_0 + a_1 z^{-1} + \cdots + a_n z^{-n}}.$$



Obrázek 8 – Převod Laplaceova *s*-prostoru do diskrétního *z*-prostoru; vyznačena oblast stability. [14]

Koeficienty $a_0, \ldots a_n$ a $b_0, \ldots b_n$ se tzv. *normují* – zlomek se vykrátí, aby platilo $a_0 = 1$, a hodnoty se využijí pro nastavení vah sčítanců IIR filtru.[15]

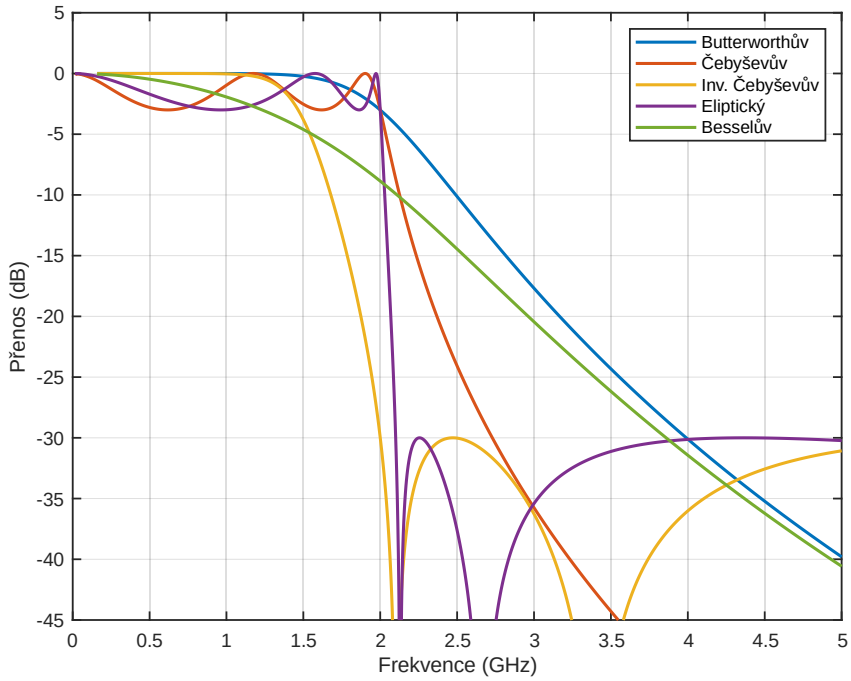
Zvýšením řádu filtru se znásobí sklon -20 dB/dekádu a dosáhne se tím ostřejší charakteristiky – jak si ovšem později ukážeme, roste u filtru vyššího řádu dopravní zpoždění, rovněž i paměťová a výpočetní náročnost implementace, hrozí i nestabilita obvodu, není-li rozložen na filtry nižších řádů, například bikvady.

Pro dosažení strmější charakteristiky u stejného řádu se používá tzv. Čebyševův filtr, využívající polynomy odvozené ruským matematikem Pafnutijem L. Čebyševem. Póly Čebyševova filtru leží na elipse se středem v bodě $0 + 0i$. Cenou za ostřejší útlum je ovšem zvlnění přenosu v propustném pásmu. Hloubka tohoto zvlnění v dB je nastavitelná při výpočtu koeficientů. Volba ploššího přenosu v propustném pásmu rozšiřuje elipsu pólů blíže kruhu odpovídajícímu Butterworthovu filtru a tím snižuje strmost frekvenční charakteristiky v nepropustném pásmu.

Definice Čebyševových polynomů je složitější a pro návrh filtru se proto v dnešní době používá specializovaný software, například funkce `[b, a] = cheby1(n,Rp,Wp)`, která je součástí *Signal Processing Toolbox* v MATLABu. Zadáním počtu řádů, akceptovatelného zvlnění v propustném pásmu (*passband ripple*) (`Rp` [dB] $= A_{\mathrm{C}}$) a normalizované krajní frekvence (`Wp` $= f_{\mathrm{Cn}} = \frac{2f_{\mathrm{C}}}{f_{\mathrm{VZ}}}$) přímo získáváme matici přenosových koeficientů filtru $[b;\ a]$. Dalšími parametry můžeme vytvořit jiné ze čtyř typů filtru, tedy horní propust či pásmovou propust a zádrž, viz oficiální dokumentace funkce.[16]

Inverzní Čebyševův filtr, v cizojazyčné literatuře označovaný jako *Chebyshev Type-II filter*, má přenos opačný oproti běžnému (*Type-I*) filtru. Pro jeho návrh lze použít příkaz `[b, a] = cheby2(n,Rs,Ws)`, parametrem `Rs` (*stopband attenuation/ripple*) se nastavuje maximální hodnota přenosu v nepropustném pásmu ($-F_{\mathrm{S}}$ [dB]).[17]

Čtveřici nejpoužívanějších filtrů pak uzavírají filtr eliptický, jež dosahuje ještě strmější frekvenční charakteristiky za cenu zvlnění přenosu v propustném i nepropustném pásmu, a filtr Besselův, u kterého je frekvenční charakteristika pozvolnější, ovšem jeho odezva na jednotkový skok odpovídá kriticky tlumenému obvodu, tedy bez překmitů. Pro selektivní voltmetr, kde očekáváme co nejplošší přenos v propustném pásmu a strmý útlum v nepropustném pásmu, jsou však méně výhodné než oba dříve zmíněné filtry.



Obrázek 9 – Srovnání filtrů 5. řádu dle vzorového příkladu pro MATLAB.[3]

1.2.4 Kritérium stability filtru

Uvažováním okrajů tolerančního pole můžeme odvodit vztah pro výpočet koeficientů filtru, a ten použít jako metodu návrhu IIR filtru požadovaných vlastností.

Nejprve definujeme množinu, kam musejí náležet póly filtru pro zachování Nyquistova kritéria stability. Vycházíme z obecného vzorce násobení komplexních čísel vyjádřených goniometrickým tvarem:

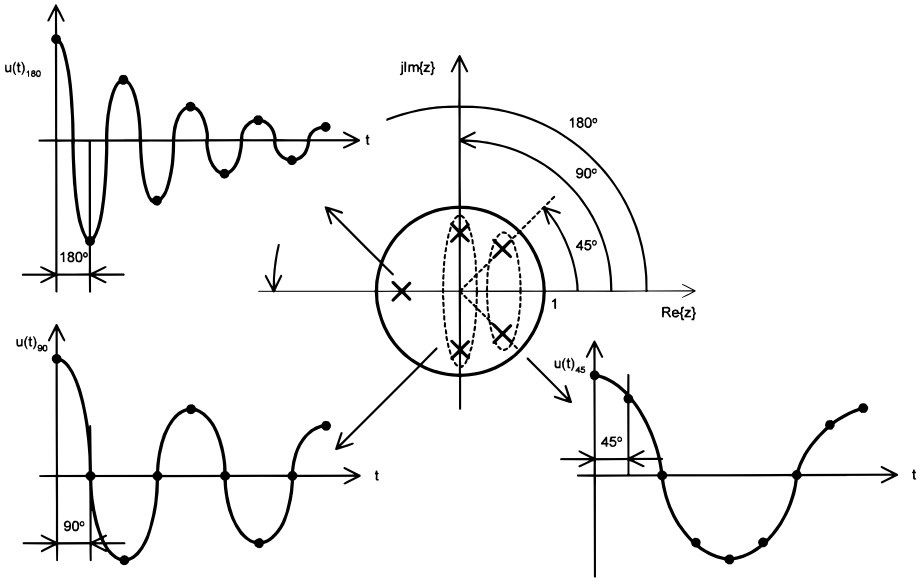
pro $z_1 = r_1(\cos\theta_1 + i\sin\theta_1)$ a $z_2 = r_2(\cos\theta_2 + i\sin\theta_2)$ platí

$$z_1 z_2 = r_1 r_2[\cos(\theta_1 + \theta_2) + i\sin(\theta_1 + \theta_2)]\,.$$

Diskrétní kroky, při kterých jakýkoli signál putuje iteracemi zpětné vazby filtru, odpovídají opakovanému násobení komplexní frekvencí odpovídající jedním z pólů filtru z_P, tedy násobení stále se zvyšující mocninou. Body, kterými bude procházet, tvoří logaritmickou spirálu se středem v počátku souřadnicového systému, neboť při každém kroku dosahuje absolutní hodnota následujícího členu geometrické posloupnosti s koeficientem $|z_P|$, zatímco se úhel zvětšuje o $\arg(z_P)$. Reálná složka impulsní charakteristiky filtru pak tvoří kmity, jejichž úhlová frekvence násobená T_{VZ} je rovna $\arg(z_P) \in \langle -\pi;\ +\pi\rangle$ rad, a útlum, jímž se jejich amplituda násobí každou T_{VZ}, je roven $|z_P|$. Její absolutní hodnota při dostatečně velkém n libovolnou mez překročí právě tehdy, pokud pól filtru, normalizovaný jeho zesílením, leží mimo jednotkovou kružnici. Pro pól ležící na jednotkové kružnici dochází k nestabilitě jen za určitých okolností, jelikož absolutní hodnota signálu se s iteracemi nemění, pouze mění fázi a kumuluje se s dalšími vzorky. Signál se stejnosměrnou složkou, kterým je například Heavisideův skok, tudíž po

[3] Jedná se o simulaci analogového provedení filtrů, tj. ve spojitém čase, s ideálními zpožďovacími členy a bez kvantování. Kód pro vykreslení grafu viz příklad *Comparison of Analog IIR Lowpass Filters*.[37]

dostatečném čase, ačkoli po lineárním namísto geometrického nárůstu, způsobí přesáhnutí libovolné meze. Takový filtr označujeme jako krajně nestabilní.



Obrázek 10 Vliv umístění pólů na přenos impulsu filtrem [12]

1.2.5 Výpočet koeficientů Čebyševova filtru

Při návrhu Čebyševova filtru se volí následující parametry:

- α_P [dB]: zvlnění, tj. maximální útlum v propustném pásmu, obvykle 3 dB;
- ω_0: krajní úhlová frekvence propustného pásma, pro zjednodušení výpočtů se zpravidla normuje na 1: pak se počítá s relativní frekvencí $\Omega = \frac{\omega}{\omega_0}$;
- n: řád filtru.

Ze zvlnění α_P [dB] se vypočítá koeficient zvlnění v propustném pásmu ϵ; $\epsilon = \sqrt{10^{\frac{\alpha_P}{20}} - 1}$, naopak zvlnění lze vyjádřit jako $\delta\,[\mathrm{dB}] = 20\log_{10}(1+\epsilon^2)$.

Řád filtru vychází z potřebné strmosti frekvenční odezvy, a odpovídá mu počet nerovností v propustném pásmu. Minimální přenos v propustném pásmu je $G_{\min} = \frac{1}{\sqrt{1+\epsilon^2}}$, a to na frekvencích $\Omega_{G\min} = \cos\left(\frac{2k-2}{2n}\pi\right)$; maximální je $G_{\max} = 1$, a to na frekvencích $\Omega_{G\max} = \cos\left(\frac{2k-1}{2n}\pi\right)$, kde $k \in \mathbb{N}; k \leq \frac{n+1}{2}$. Přenos na frekvenci $\Omega = 0$ proto závisí podle parity řádu filtru: filtr lichého řádu dosahuje přenosu $G_n(0) = 1$, filtr sudého řádu pak $G_n(0) = \frac{1}{\sqrt{1+\epsilon^2}}$. Na okraji propustného pásma $\Omega = 1$ pak vždy dochází k přenosu $G_n(1) = \frac{1}{\sqrt{1+\epsilon^2}}$ Vyšší řád filtru umožní dosáhnout strmějšího útlumu v nepropustném pásmu při stejné úrovni zvlnění v propustném pásmu, mimo vyšší počet minim a maxim mu však rovněž odpovídá vyšší fázové zpoždění (90° za každý řád), rovněž stoupá výpočetní náročnost. Skupinové zpoždění filtru není monotónní, počet lokálních maxim

a minim odpovídá řádu filtru; v případě signálů impulsního charakteru se může jednat o nežádoucí vlastnost z důvodu vzniku překmitů.

Potřebný řád filtru je možné odvodit z hranice propustného a nepropustného pásma, viz obrázek 4. Napěťový přenos na kraji nepropustného pásma musí dosahovat $G(\omega_S) \leq 10^{\frac{-A_S}{10}} = \frac{1}{\sqrt{1+\lambda^2}}$, tedy odvodíme vztah pro útlumový činitel λ: $\lambda \leq \sqrt{\frac{1-G(\omega_S)}{G(\omega_S)}}$. Obdobně platí pro kraj propustného pásma $G(\omega_C) \geq 10^{\frac{-A_S}{10}} = \frac{1}{\sqrt{1+\epsilon^2}}$, platí tedy $\epsilon \geq \sqrt{\frac{1-G(\omega_C)}{G(\omega_C)}}$. Řád filtru volíme dle následující nerovnice:

$$n \geq \frac{\operatorname{arccosh}\frac{\lambda}{\epsilon}}{\operatorname{arccosh}\frac{\omega_S}{\omega_C}}$$

Amplituda přenosu Čebyševova filtru odpovídá rovnici

$$G_n(\Omega) = \frac{1}{\sqrt{1+\epsilon^2 T_n^2 \Omega}}$$

Pro přenos výkonu tedy platí

$$H(j\Omega)^2 = H(j\Omega) \cdot H(-j\Omega) = \frac{1}{1+\epsilon^2 \cdot T_n^2 \Omega}$$

Záměnou $s = j\Omega$; $\Omega = \frac{s}{j}$ získáváme

$$H(s)^2 = H(s) \cdot H(-s) = \frac{1}{1+\epsilon^2 \cdot T_n^2\left(\frac{s}{j}\right)}$$

Člen $T_n(\Omega)$ (charakteristický polynom filtru $\varphi(\Omega)$) je Čebyševův polynom n-tého řádu, odpovídající definici

$$T_n(x) = \begin{cases} \cos(n \arccos x), & |x| \leq 1 \\ \cosh(n \operatorname{arccosh} x), & x > 1 \end{cases},$$

případně je možné koeficienty získat rekurzivní rovnicí:

$$T_{n+1}(x) = 2x\big(T_n(x)\big) - T_{n-1}(x), \;\; n \in \mathbb{N}, \;\; T_0(x) = 1, \;\; T_1(x) = x.$$

Pro póly platí, že jmenovatel zlomku je roven 0, tedy

$$1 + \epsilon^2 \cdot T_n^2\left(\frac{s}{j}\right) = 0.$$

Proměnná Čebyševových polynomů je $\frac{s}{j}$ a vzhledem k tomu, že se póly nacházejí v přenosovém pásmu ($\Omega < 1$), získáváme

$$T_n\left(\frac{s}{j}\right) = \cos\left(n \arccos\frac{s}{j}\right) = \pm\frac{j}{\epsilon}.$$

Vyjádřením $\arccos\frac{s}{j} = u + jv$ jako součet reálné u a imaginární složky jv získáváme

$$v = \frac{1}{n} \cdot \arg\sinh\frac{1}{\epsilon}$$

Následně platí pro póly $m \in \{1, 2, \dots, n\}$:

$$\Sigma_m = -\sinh v \cdot \sin\left[(2m-1)\frac{\pi}{2n}\right];\ \Omega_m = -\cosh v \cdot \cos\left[(2m-1)\frac{\pi}{2n}\right]$$

$$\frac{\Sigma_m^2}{\sinh^2 v} = \sin^2\left[(2m-1)\frac{\pi}{2n}\right]; \frac{\Omega_m^2}{\cosh^2 v} = \cos^2\left[(2m-1)\frac{\pi}{2n}\right]$$

$$\frac{\Sigma_m^2}{\sinh^2 v} + \frac{\Omega_m^2}{\cosh^2 v} = \sin^2\left[(2m-1)\frac{\pi}{2n}\right] + \cos^2\left[(2m-1)\frac{\pi}{2n}\right] = 1$$

Kořeny polynomů s_m jsou rozmístěny na elipse se středem v počátku souřadnicového systému; tvoří páry s opačným znaménkem reálné části, jsou tedy zrcadlově symetricky uspořádané podle osy y; zároveň jsou dvojice pólů (vyjma středního u filtru lichého řádu) komplexně sdružené a tedy uspořádané symetricky podle osy x.

Pro sudá n platí

$$H(s) = \frac{1}{\sqrt{1+\epsilon^2}} \cdot \prod_{m=1}^{\frac{n}{2}} \frac{\Sigma_m^2 + \Omega_m^2}{s^2 - 2s\Sigma_m + \Sigma_m^2 + \Omega_m^2}$$

Pro lichá n platí

$$H(s) = \frac{\sinh v}{s + \sinh v} \cdot \prod_{m=1}^{\frac{n+1}{2}} \frac{\Sigma_m^2 + \Omega_m^2}{s^2 - 2s\Sigma_m + \Sigma_m^2 + \Omega_m^2}$$

Poloha pólů odpovídá Butterworthovu filtru, jehož póly jsou rovnoměrně rozmístěny na levé polovině jednotkové kružnice na úhlech $\phi_k = \frac{\pi}{2}\frac{2k-1}{n}$, avšak jejich souřadnice jsou škálovány koeficienty r_x, r_y a nacházejí se tak na elipse, jejíž poloosy odpovídají těmto koeficientům.

Pro výpočet koeficientů se získá pomocná proměnná β z útlumového činitele ϵ:

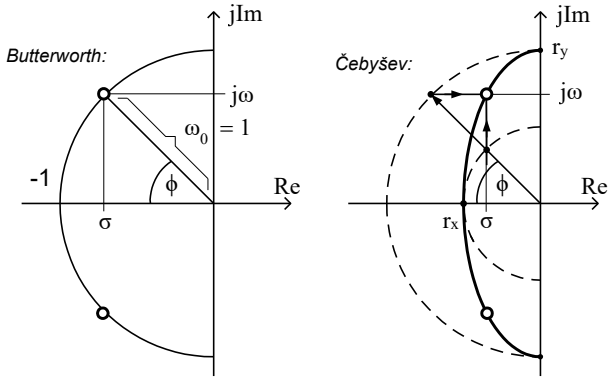
$$\beta = \sqrt[n]{\frac{\sqrt{1+\epsilon^2}+1}{\epsilon}}$$

Hlavní (svislou) poloosu r_y elipsy i vedlejší (vodorovnou) poloosu r_x pak určují koeficienty

$$r_y = \Omega_C \frac{\beta^2+1}{2\beta};\ r_x = \Omega_C \frac{\beta^2-1}{2\beta}$$

Zvýšení faktoru zvlnění má za následek snížení proměnné β, a tím i zvýšení rozdílu poloos r_x, r_y; tomu odpovídá větší výstřednost elipsy a větší rozdíl od Butterworthova filtru. Pro zvlnění blížící se nule se póly limitně blíží poloze na kružnici (obě poloosy mají délku 1), filtr je pak totožný s Butterworthovým filtrem téhož řádu.

Geometricky lze polohu bodu na elipse získat pomocí kružnic poloměrů r_x, r_y, kde z bodu v úhlu ϕ na každé z nich je vytyčena kolmice příslušné souřadnicové osy:



Obrázek 11 – Převod polohy pólů Butterworth → Čebyšev geometrickou cestou (B. Brtník)

Z toho vycházejí rovnice pro souřadnice (x_k, y_k) pólů filtru v s-prostoru:

$$(x_k, y_k) = \frac{\beta^2+1}{2\beta}\cos\left(\frac{\pi}{2}\frac{2k-1}{n}\right) + \frac{\beta^2-1}{2\beta} j \sin\left(\frac{\pi}{2}\frac{2k-1}{n}\right)$$

1.3 Hromadné dálkové ovládání

1.3.1 Cíl a princip regulace elektrické rozvodné sítě

Vlastností síťového rozvodu elektrické energie je neustálá rovnost výrobního výkonu a spotřebního příkonu – samotná rozvodná zařízení nemají možnost ukládání energie.

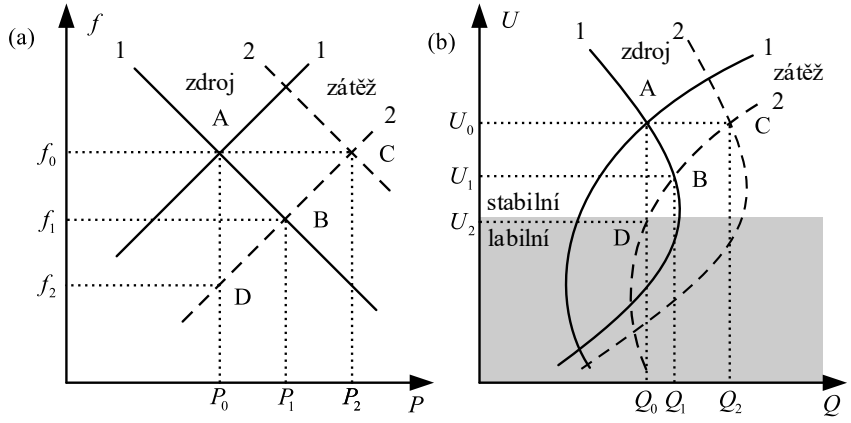
V praxi se ovšem setkáváme s pravidelnými i mimořádnými výkyvy výroby i spotřeby. Nezávisle na vůli dodavatele dochází k předvídatelným změnám výrobní kapacity dle denní doby a počasí, i nepředvídatelným (nehody, přírodní katastrofy, chyby zařízení [18]). Rovněž na straně poptávky, tedy admitanci zátěže kladené na elektrickou síť

spotřebiteli, dochází k výkyvům, jež dodavatel nemůže ovlivnit, především závislých na provozní době podniků a změnách osobní spotřeby v závislosti na denní době.

Vlivem záporné závislosti napětí a jalového výkonu (viz obr. 12b) synchronních generátorů i transformátorů dochází v závislosti na reaktivní složce poptávky ke kolísání síťového napětí. Na straně zátěže se pro regulaci napětí v sítích vn a nn využívá automatické přepínání odboček transformátorů. Aby se pracovní bod sítě (viz graf) nedostal do labilní části U-Q charakteristiky (poloha D), a nedošlo pak k lavinovému poklesu napětí [19], je nutno využít umělé reaktanční zátěže pro kompenzaci jalového výkonu, případně změnit režim elektronických invertorů fotovoltaických elektráren. [20]

Větším problémem pro distributory elektřiny je sledování frekvenčně-výkonové závislosti sítě. Pokles síťové frekvence působí saturaci jader transformátorů mezi vvn-vn-nn podsítěmi a tepelné ztráty, může dojít i k jejich trvalému poškození. Je tedy nutné dodržet co nejkratší odchylky; v případě extrémní chyby 5 % (47,5 Hz) ukončují elektrárny provoz okamžitě a vyžadují manuální restart. [18] V Evropě se dlouhodobě úspěšně daří dodržovat toleranční pásmo 50 Hz ±0,2 Hz.

Elektrárny se synchronními generátory využívají mechanický moment setrvačnosti rotorů turbín a generátorů, případně přídavných setrvačníků, pro krátkodobé vyrovnávání okamžitých změn síťového napětí. [21] Tuto samovolnou regulaci popisuje statická napěťově-frekvenční charakteristika generátorů a zátěží, viz obr. 12a:



Obrázek 12 – Statická charakteristika elektrické sítě[4]

Vycházejme z ustáleného bodu A (P_0, f_0) Síť pracuje na cílovém kmitočtu f_0 a spotřeba i výroba energie (normovaná účinností distribuční sítě) se rovnají P_0. Dojde-li zvýšením poptávky k posunu zatěžovací přímky z polohy označené 1 do polohy 2 a zdroj nemá schopnost regulace, zůstává výkon zdroje nezměněn a pracovní bod (D) odpovídá nižší

[4] Charakteristiky jsou obvykle znázorněny inverzně; tento graf (Li, 2022) [38] byl invertován, aby f a U byly coby závislé proměnné na svislé ose; zjednodušená U-Q charakteristika byla doplněna o zakřivení a nestabilní část (Beran, 1988), u f-P charakteristiky je nelinearita v praxi zanedbatelná.

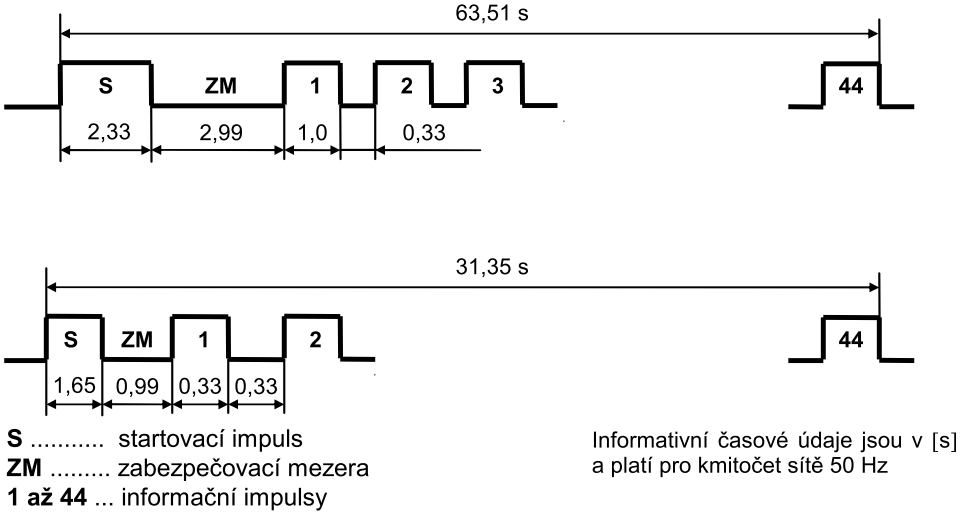
frekvenci f_2. Zdroj s primární (setrvačnou) regulací samovolně zvýší svůj výkon na bod P_1, avšak stále za snížení síťové frekvence.

Pro navrácení frekvence směrem k cílové hodnotě tak nastává nasazení tzv. sekundární regulace s odezvou v řádu sekund, zprostředkovanou zejména vodními elektrárnami, případně bateriovým úložištěm. To umožňuje dočasně (na několik desítek minut či jednotek hodin) dodáním rozdílového výkonu $P_2 - P_1$ přesunout pracovní bod do bodu C. Dle doporučení RCGE má tvořit sekundární regulace alespoň 2 % výrobní kapacity. Než dojde k vyčerpání energie vyhrazené pro sekundární regulaci, musí být dlouhodobá nerovnováha pokryta systémem terciární regulace neboli minutovou zálohou – zpravidla změnou výkonu kotle tepelných elektráren. To definitivně posouvá frekvenčně-výkonovou charakteristiku zdroje na bod C a vyrovná tak odchylky poptávky trvající desítky minut a výše. Nejpomalejší průběh charakteristiky pak mohou vyrovnávat i jaderné elektrárny, avšak zásahy mohou trvat i více než den.

Vzhledem k nákladům na instalaci záložní výrobní kapacity, zejména u obnovitelných zdrojů, se jako ekonomicky výhodnější řešení pro distributory elektrické energie (v ČR ČEZ, EG.D, PRE) jeví rovněž regulovat poptávku. Bez smlouvy s odběratelem se nabízí pouze radikální metoda tzv. „odlehčení zatížení“ (*load shedding*), spočívající v odpojení části odběratelů, a tedy snížení spolehlivosti služeb. Tato varianta je ze zřejmých důvodů mezi zákazníky nepopulární a působí ekonomické ztráty, proto bylo rozhodnuto o vybudování systému pro dálkové řízení spotřeby.

1.3.2 Historie a protokoly HDO

Experimenty se systémy dálkového řízení spotřeby energie u nás započaly ve 30. letech 20. století. Později byly pro ovládání spotřeby energie používány spínací hodiny. Tento systém jednotné dvoutarifové sazby, tehdy nazývané *noční proud*, měl zřejmé problémy: ve spínací čas docházelo k prudkým změnám spotřeby, nebylo možné centrálně měnit čas spínání ani provádět akutní zásahy pro regulaci sítě. Zřejmé výhody komunikačního systému v elektrické síti, včetně rozšířených možností jako ovládání veřejného osvětlení nebo komunikaci mezi elektrárnami, vedly k výstavbě sítě HDO od roku 1962. Pro rozdělení sítí krajských energetických podniků i v rámci experimentálního chodu bylo rozhodnuto o různé nosné frekvenci i časování datagramů – krátké datagramy se dodnes využívají v bývalém Severočeském kraji. Šlo nejprve o 44 bitů s modulací OOK a kódováním „impuls–mezera“, kdy přítomnost impulsu v jedné ze 44 pozic dávala pokyn k sepnutí stykače mezera pak jeho vypnutí. [22]



Obrázek 13 – Časování dlouhého, resp. krátkého datagramu typu impuls–mezera Informační obsah je u obou typů shodný. [23]

Pro rozšíření adresovacího prostoru vznikl dodnes využívaný systém „impuls–impuls“ (též zvaný sériový povelový kód), kde má každý z přijímačů adresu tvořenou jednou ze čtyř poloh ve skupině A a jednou z osmi poloh ve skupině B. Datagram nejprve adresuje skupinu přijímačů kódováním „1 z N“ ve čtyřbitové adrese A a kombinací jedné až tří jedniček z osmi ve skupině B. Následuje povelová část (P) s 16 kanály, pro každý z nichž se vysílá dvojice bitů „zapnout-vypnout“, kde přítomnost impulsu v příslušné pozici znamená odpovídající povel. Změna stavu stykače je podmíněna právě jedním impulsem z každé dvojice. Kombinací adres a kanálů může distributor nezávisle ovládat až 512 skupin spotřebičů, zpravidla označených na přijímači kódem ve formátu A#B#P##. [24]

V současné době mají díky vysokému počtu kódů HDO domácnosti i firemní odběratelé možnost využít mnoha tarifů pro různé účely. [25] Pro každou sazbu pak je zákazníkovi přidělen jeden z kódů HDO odpovídající skupiny, a tak se dociluje možnosti plynuleji ovládat změny spotřeby např. na přelomu hodiny. Pro ještě širší využití (dálkové odpojení neplatičů, potřeby soukromých podniků, vysílání nouzových zpráv v případě ohrožení státu) byl vyvinut patentovaný komunikační protokol VERSACOM, část jehož specifikací udává DIN 43861 [26]. Cílem tohoto systému je možnost ovládat jak skupiny zařízení, tak jednotlivá odběrná místa. Přijímače jsou vybaveny hodinami reálného času a mohou dle časovaných povelů spínat samostatně bez aktivní spolupráce centrály, zatímco je přenosový kanál volný pro povely typu ZPA.

Anglický pojem pro činnost vedoucí k vyrovnání výchylek výroby a spotřeby je *ripple control*, od toho se odvíjí název pro systémy HDO: RCS – *ripple control system*.

Zatímco domácnosti mají zpravidla dlouhodobě stálé spínací časy, odběratelé elektřiny v průmyslu mají možnost zařízení zařadit do jedné z mnoha tarifních kategorií, a s dodavatelem uzavřít smlouvu o dodávce, jež definuje cenu energie a pevné časy sepnutí, případně přípustný rozsah mimořádných spínacích událostí a kompenzace za výpadky. Zákazníkem správně využívaný HDO k ovládání vhodných zařízení zásadně

neovlivní jeho pohodlí a činnost. Nejčastěji jsou ovládány spotřebiče, jejichž práce má akumulační charakter, [27] tedy například

- akumulační ohřívače vody v domácnostech (tzv. boilery), rekreaci, průmyslu,
- přímotopné vytápění, tepelná čerpadla, zejména s akumulačními jednotkami,
- čerpací zařízení vodáren,
- automatické výrobní procesy (např. elektrické pece, výpočetně složité úkony),
- nabíjení baterií, zejména elektrovozů.

Budoucnost systému hromadného dálkového ovládání je v tzv. *smart grid*, konceptu provázaném s IoT (*Internet of Things*). Počítá se se stále větším rozšířením elektroměrů se zabudovaným modulem oboustranné bezdrátové komunikace využívající technologii GSM, NB-IoT nebo LTE-M s možností dálkového řízení i odečtu, jsou však náchylnější na umístění elektroměru. Ve světě se nyní experimentuje s možností čtvrthodinové sazby pro domácnosti, kdy zákazník nakupuje elektřinu za cenu odpovídající aktuální tržní situaci. [28]

1.3.3 Technické parametry šíření

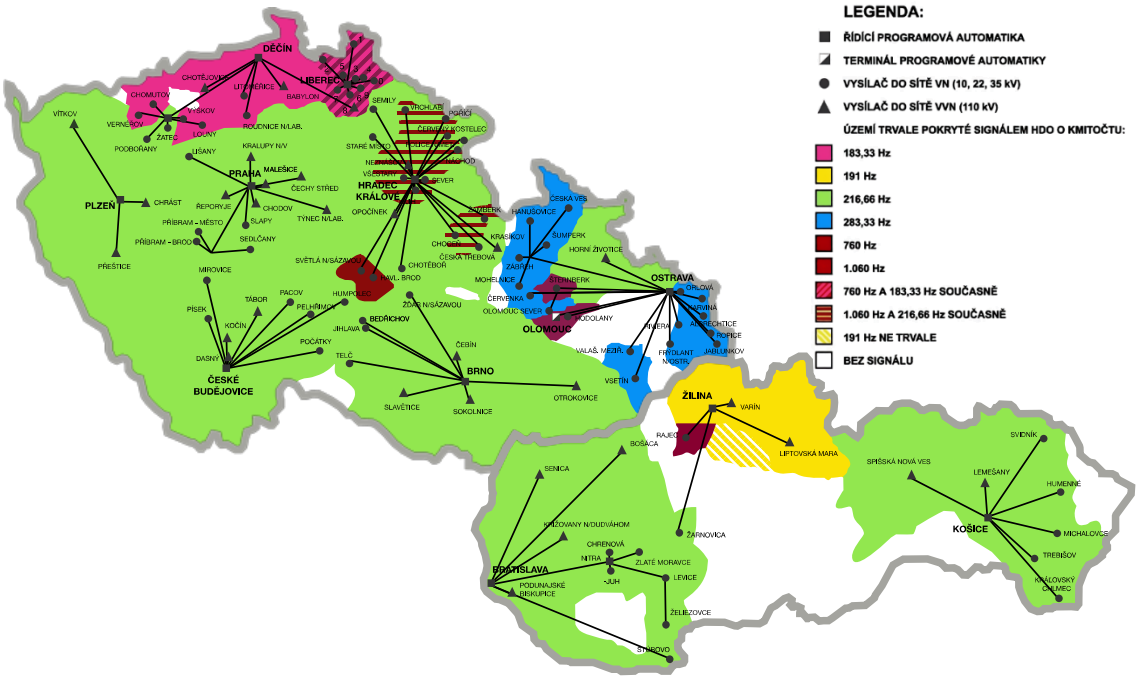
Pro systémy hromadného dálkového ovládání byly dle normy ČSN EN 61000-2-2 určeny následující frekvence:

Tabulka 2 – Přehled frekvencí doporučených pro HDO[27]

Jmenovitý ovládací kmitočet [Hz]	Doporučené maximum vysílací úrovně [%U_n]	Maximální úroveň ovládacího napětí [%U_n]
110	1,75	6
$133\frac{1}{3}$	1,75	6
167	1,75	6
$183\frac{1}{3}$	3,00	6
$216\frac{2}{3}$	1,80–2,00	6
232	1,75	6
267	1,75	6
$283\frac{1}{3}$	3,00	6
$316\frac{2}{3}$	3,00	6
$383\frac{1}{3}$	3,00	6
425	3,00	6
500	5,00	6
600	5,00	6
760	5,00	6
1060	4,00	6

Frekvence 267 Hz je vyhrazena pro potřeby Českých drah. Norma doporučuje s ohledem na přeslechy používat co nejnižší vysílací úroveň.

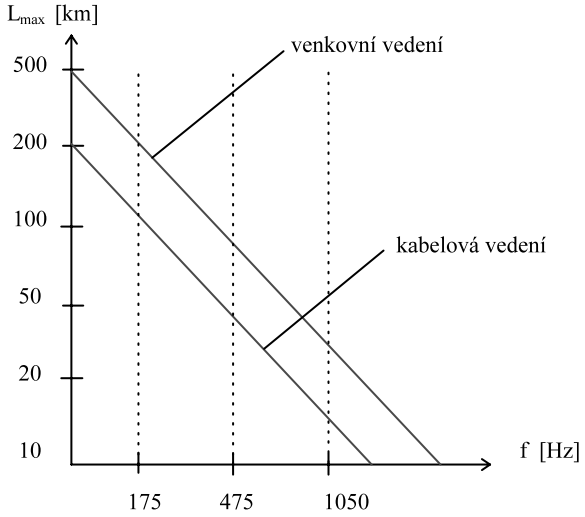
Skutečně používané frekvence mají různé pokrytí, vycházející z bývalého rozdělení sítě ČSSR na krajské energetické podniky, viz obrázek 14.



Obrázek 14 – Pokrytí území ČR a SR signálem HDO, stav k červnu 2000.[5]

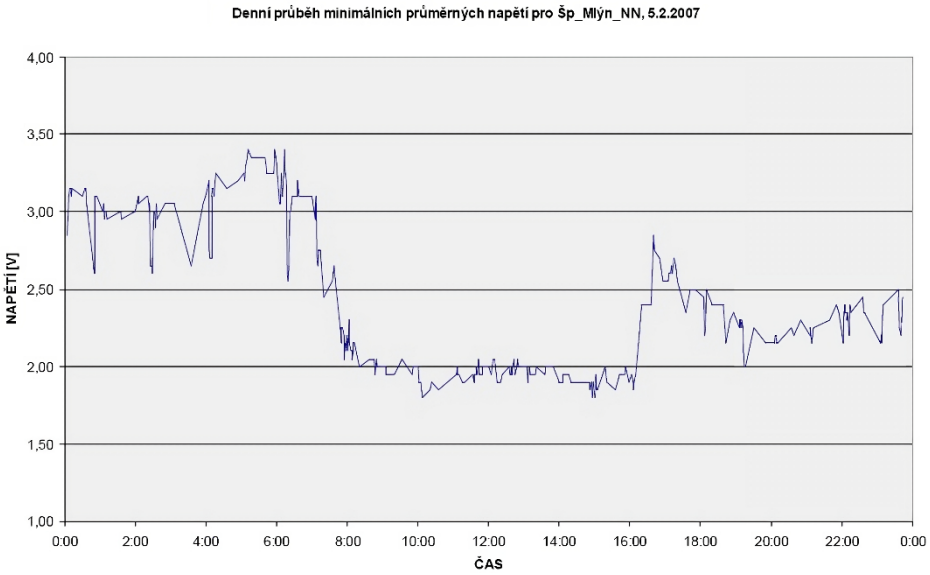
Volba signální frekvence je ovlivněna především rozsahem a impedanční charakteristikou sítě. Z mapy je patrné, že signál vyšších frekvencí (červeně a fialově vyznačené oblasti) zůstává využitelný jen v okruhu nižších desítek kilometrů. [27]

[5] Jedná se o vektorové překreslení obrázku nízké kvality. Informaci o tehdejší frekvenci vysílače Rajec (760 Hz nebo 1060 Hz) nebylo možné dohledat, nyní se používá 191 Hz. Obrázek se vyskytuje v několika akademických dokumentech a závěrečných pracích[32][39][21], žádný z nich necituje novější data.



Obrázek 15 – Dosah signálu HDO v závislosti na signální frekvenci

V poslední době se například upouští od využívání frekvence 1060 Hz na severu Královéhradeckého kraje. Jedním z důvodů je provozování lanových drah v oblasti Krkonoš, jejichž synchronní motory tvoří indukční zátěž tlumící amplitudu signálu HDO.



Obrázek 16 – Denní průběh napětí HDO ve Špindlerově Mlýně, 5. 2. 2007[6].

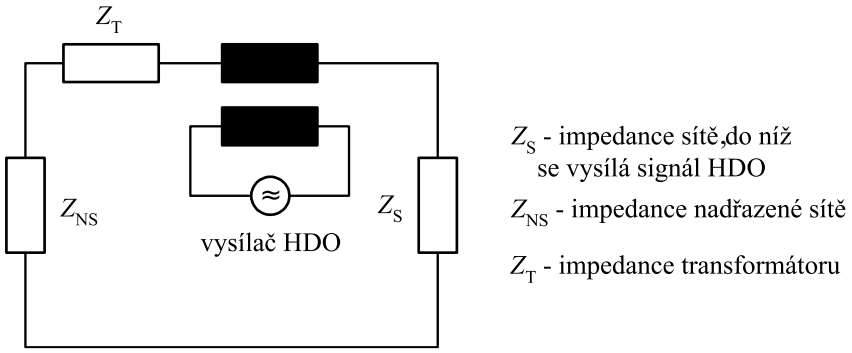
HDO není jediný komunikační protokol v elektrické síti, technologii PLC využívaly například i elektrárny a důlní čety k hlasovým hovorům. Pásma A–D pro tyto účely vymezila organizace CENELEC mezi 95 kHz až 150 kHz. PLC rovněž nachází využití pro sítě LAN, kde se širokopásmový signál (broadband, šíře až 30 MHz) po síťových rozvodech šíří pouze v řádu stovek metrů. Výzkum na VUT Brno [29] ukázal závislost mezi délkou nn vedení a rychlostí přenosu dat systému Power Line Networking.

[6] Nepravidelné rozmístění bodů odpovídá vysílání datagramů. Obrázek uvedl Jan Hlaváček v prezentaci *Vliv lanovek a lyžařských vleků na signál HDO* na kongresu ČK CIRED 2009. [26]

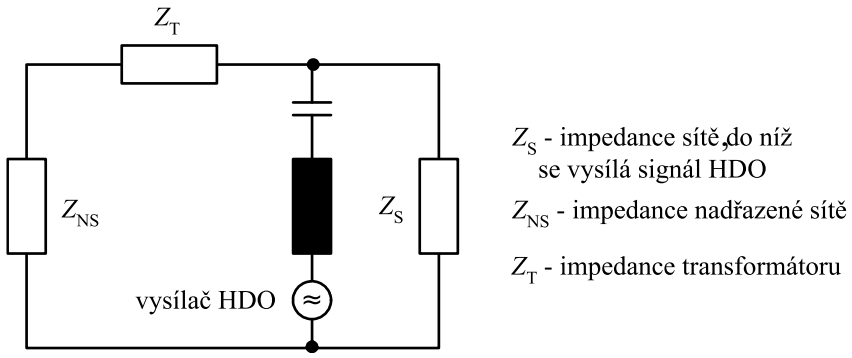
1.3.4 Vysílání signálu

Vysílače signálu HDO se rozlišují podle určení do sítí nn, vn a vvn, a podle vazby – sériové nebo paralelní. Sériová vazba se používá v místech, kde platí $Z_{\mathrm{T}} + Z_{\mathrm{NS}} \geq Z_{\mathrm{S}}$ (viz obr. 17, paralelní pak pro opačný případ. Paralelní vazba má výhodu, že revize nebo porucha vysílače HDO nepřeruší funkci rozvodné stanice, navíc umožňuje využít výstupní transformátor pro nižší proud.

V sítích nn je možné vysílač zapojit proti střednímu vodiči. Ten tak vysílá současně do všech fází, signál se navíc nešíří transformátorem do nadřazené sítě. [30]



Obrázek 17 – Schéma zapojení vysílače HDO se sériovou vazbou [30]



Obrázek 18 – Schéma zapojení vysílače HDO s paralelní vazbou [30]

Pro omezení šíření signálu HDO do částí sítě, kde by mohlo vyvolávat rušení citlivých zařízení nebo vysílání HDO pro jiný kraj, se používají hradící členy (*signal traps*) realizované rezonančními obvody LC.

1.3.5 Příjem signálu

Příjem signálu HDO je ztížen existencí mnoha frekvencí v rozvodu síťového napětí. Hlavní složka 50 Hz má amplitudu řádově vyšší než přijímaný signál, její frekvence je ovšem výrazně nižší. Dalšími významnými rušivými frekvencemi jsou liché harmonické (150 Hz, 250 Hz atd.), jelikož zařízení s můstkovým usměrňovačem a filtrací, například stále běžnější spotřebiče napájené spínaným zdrojem, nabíjejí filtrační kondenzátor blízko kladného i záporného maxima síťového napětí, tedy na dvojnásobné frekvenci, za vzniku harmonického rušení na násobcích 100 Hz. Násobením s 50 Hz sinusoidou, jež odpovídá změnám polarity, dojde k inverzi každého druhého pulsu a podobně jako ve

směšovači heterodynu k posunutí frekvencí o ±50 Hz, tedy ze sudé na každou lichou harmonickou frekvenci střídavého napětí.

Spotřebiče na vstupu zpravidla obsahují filtr typu dolní propust nebo navíc člen pro korekci účiníku, vyšší harmonické frekvence jsou tedy tlumeny. Přesto se v praktickém měření síťového napětí ukázala existence složky 150 Hz o napětí srovnatelném se signálem HDO. Pro měření signálu HDO okolo 200 Hz, jehož amplituda dosahuje pouhých několika voltů, je tedy při výběru filtrů třeba volit ty s vysokou selektivitou. Z tohoto důvodu byly stanoveny jednotlivé frekvence signálu HDO mimo hodnoty harmonických násobků rozvodné sítě, a přijímače využívají vysoce selektivních filtrů pro jediný kmitočet, zpravidla tvořených rezonančními LC články odladěnými na cílovou frekvenci. Příkladem takového zařízení je selektivní voltmetr státního podniku ZPA Trutnov, model AU 03 (obr. 19), jehož LC články mají $f_0 = 217$ Hz.

Obrázek 19 – Selektivní voltampérmetr pro frekvenci 217 Hz, ZPA Trutnov. Foto B. Brtník

2 Praktická část

2.1 Návrh hardwarového provedení

2.1.1 Inspirace

Hlavním vodítkem provedení číslicového filtru byla učebnice *Algoritmy číslicového zpracování signálů* [12]. Zaujala mě koncepce zařízení, jež se zde pro nízkofrekvenční DSP používá: jedná se o modulární vývojový kit, ze kterého je možné sestavit zařízení pro různé aplikace, například měřič RLCG nebo wattmetr pro obvody se střídavým proudem. Základním prvkem tohoto systému je přípravek COM644KIT disponující jedním z nejvýkonnějších 8bitových MCU řady AVR, ATmega644. Jedná se o jednostrannou desku plošných spojů o rozměrech 87x117 mm, navrženou pro usazení do běžné krabičky KP60, a obsahující zejména:

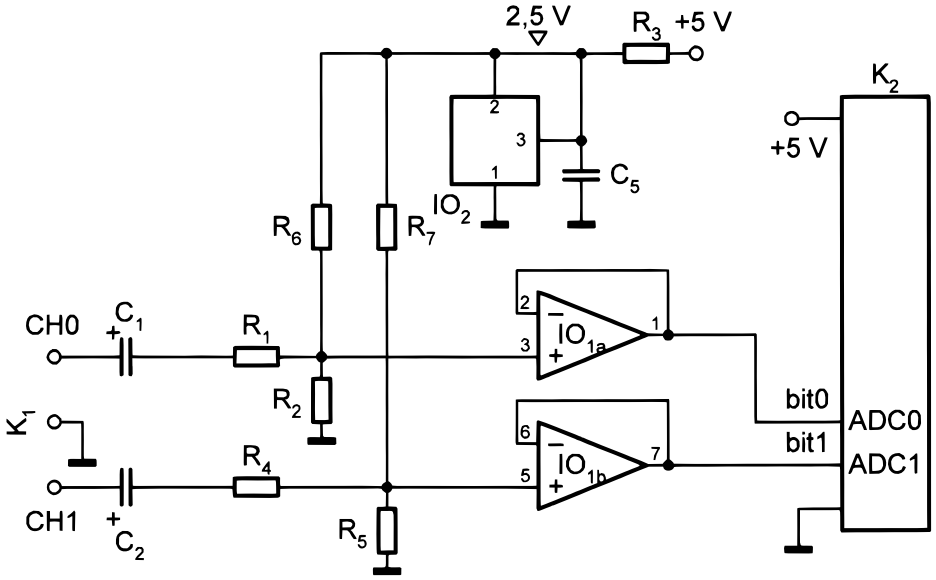
- podpůrné součástky zajišťující chod MCU – filtrace napájení, krystal 20 MHz,
- mikrokontrolér AT89C2051 umožňující nahrání programu po sériové sběrnici,
- sériový port DE-9 a integrovaný obvod MAX232 pro převod napěťových úrovní,
- napájecí souosý konektor (zásuvka 5,5/2,1 mm) pro externí zdroj DC 5V,
- vyvedení všech I/O portů procesoru (PORTA~PORTD) spolu s napájením pomocí 10pinových konektorů DC3-10P.

Dalším přípravkem, označeným „PANEL“, je pak ovládací panel realizovaný na DPS stejné velikosti, jež lze umístit do krabičky ve vrstvě nad COM644KIT, a přidává uživatelské rozhraní pomocí řady tlačítek a standardního podsvíceného dvouřádkového znakového displeje s paralelním řízením kompatibilním s HD44780.

Je rovněž popsán přípravek umožňující analogový výstup, označený jako EDAC, jež obsahuje integrovaný obvod TLC7528, osmibitový dvoukanálový DAC převodník, jež s hlavní deskou komunikuje po sběrnici SPI.

Součástí MCU ATmega644 je 12bitový A/D převodník, jeho vstupní rozsah je ovšem omezen mezi úrovně na vývodech AGND a AREF. Samotný COM644KIT tedy neumožňuje zpracování signálů střídavého napětí. Pro úpravu napěťových úrovní slouží přípravek „EADC“, který upravuje stejnosměrnou složku signálu na úroveň nastavenou pomocí napěťového děliče ze zabudované napěťové reference 2,5 V (TL431 zapojeném jako Zenerova dioda). Operační zesilovač v zapojení opakovače snižuje impedanci vstupního napětí. Všechny signálové prvky jsou zdvojené, čímž se využívá obou operačních zesilovačů v integrovaném obvodu MCP602 a je proto možné zpracovat dva nezávislé analogové signály najednou. MCU sice obsahuje pouze jeden ADC s 8 kanály, mezi dvěma z nich ovšem lze během obsluhy přerušení přepínat; výrazně pak dojde k poklesu dosažitelné vzorkovací frekvence [31].

Rozměry přípravku odpovídají krabičce KM22, deska plošných spojů měří 8,5×6,1 cm.



Obrázek 20 – Schéma přípravku EADC [12]

Na vstupu má přípravek sériový RC člen (10 kΩ, 1 μF) tvořící horní propust s mezní frekvencí okolo 1 Hz, oddělující stejnosměrnou složku vstupního signálu od vstupu operačního zesilovače. Na neinvertující vstup OZ, U_+, je přivedeno napětí z odporového děliče tak, aby stejnosměrná složka jeho napětí byla polovinou referenčního napětí, a amplituda střídavé složky oproti vstupu třetinová.

Jako druhotná inspirace sloužila hardwarová koncepce řady Arduino, například Arduino Uno, pravděpodobně nejběžnější vývojové desky začínajících kutilů v oblasti elektroniky. Tento vývojový kit je založen na MCU ATmega328P a obsahuje zejména

- podpůrné součástky pro chod hlavního MCU na frekvenci 16 MHz,
- vývody napájení (`5V` a `GND`) a všech GPIO pinů MCU (`PB0~PD7`) pomocí dutinkových lišt 2,54 mm,
- port USB-B a mikrokontrolér ATMEGA8U2 s hardwarovou implementací USB, umožňující sériovou komunikaci a programování hlavního MCU,
- 4x LED pro signalizaci zapnutí, indikaci sériové komunikace, libovolnou funkci,
- regulátor napětí 5 V a konektor umožňující napájení z adaptéru 9-15 V,
- regulátor napětí 3,3 V pro napájení externích zařízení.

2.1.2 Zvážení cílů hardwarového řešení

Pro návrh hardwarového řešení jsem stanovil následující cíle:

1. bezpečnost uživatele vzhledem k zapojení do sítě nn,
2. vnější vzhled připomínající profesionální výrobek,
3. možnost rychlého přeprogramování a testování,
4. proveditelnost v amatérských podmínkách,
5. možnost přestavby na zařízení jiné funkce,
6. pokud možno malé rozměry a vzhledné provedení,
7. levné provedení s využitím součástek z dosavadní kolekce autora.

Bod 5 byl vytyčen z toho důvodu, že zvolená služba výroby desek plošných spojů PCBWay nerozlišuje cenu mezi jednou až desíti deskami standardního provedeni, je tedy vhodné vytvořit univerzální design, jež bude možné přestavět na jiné, podobné zařízení. Jako zajímavý příklad lze uvést wattmetr pro střídavý proud, jež lze sestavit z COM644KIT a několika externích vstupních a výstupních modulů.[7] V této práci bude proto vytvořen „ATMEGA DSP KIT“, jež bude umožňovat experimentování se zpracováním signálu na procesoru ATmega nejen pro realizaci selektivního voltmetru HDO, ale rovněž osobní projekty autora či čtenářů této práce.

Zásady pro vypracování bakalářské práce zmiňují indikaci „displejem a ručkovým měřidlem“. Původně se podle interpretace vedoucího práce počítalo pouze s využitím displeje s tím, že bude k dispozici analogový výstup, k němuž bude možné připojit ručkový měřící přístroj externě. Později však bylo rozhodnuto přístroj umístit do větší, plechové skříně, více připomínající analogový měřící přístroj AU 031, viz obrázek 19. Toto provedení umožňuje dovnitř umístit ručkový ukazatel se 3 LED indikujícími rozsah, větší displej a přepínač mezi měřením HDO na napájecím síťovém napětí a ze zdířek vstupu externího signálu.

Jako jedna z možností pro měření v síti nn bylo uvažováno snížit vstupní napětí transformátorem a měřit střídavé napětí na jeho výstupu. U transformátorů ovšem dochází k nelineárnímu frekvenčnímu přenosu, při saturace jádra a vznik nežádoucích harmonických složek. Proto bylo rozhodnuto snížit vstupní napětí pomocí děliče a dbát na izolaci částí vztažených k síťovému napětí. V poslední verzi zařízení však byla možnost měření v síti nn vynechána, na bezpečnost je ovšem stále třeba dbát vzhledem k napájení pomocí vestavěného zdroje stejnosměrného napětí.

Dalším požadavkem je možnost přepínání rozsahů mezi 1 V, 3 V a 10 V. Ačkoli by bylo možné tuto funkci realizovat přepínatelným analogovým děličem napětí na vstupu, došlo by ke ztrátě přesnosti kvůli realizaci pomocí rezistorů s výrobní tolerancí, již by nebylo možné kompenzovat jediným koeficientem, zvýšila by se tedy náročnost kalibrace přístroje. Vstupní napětí A/D převodníku, tvarované ze vstupu nn, by navíc při použití děliče pro vyšší rozsah dosahovalo pouze zlomku jeho rozsahu, bylo by tedy reprezentováno menším počtem kvantovacích kroků a více by trpělo na náhodný šum. Přepínač děliče by musel být navíc nejméně dvoupólový, aby mohl procesor volbu rozsahu kompenzovat pro zobrazení správné hodnoty na displeji. Jako jednodušší a přesnější se tedy jeví digitální provedení, kde je hodnota ihned po filtraci složky 50 Hz znásobena koeficientem odpovídajícím rozsahu. Tímto nedochází ke ztrátě přesnosti a rozsah 1 V může mít stejně jako vyšší rozsahy 4 platné číslice, tedy delitev=0,1 mV namísto 1 mV. U rozsahu 3 V je delitev 0,3 mV, avšak pro zobrazení je hodnota zaokrouhlena na celé milivolty Rozsah je vybírán tlačítky pomocí interaktivní nabídky na displeji a zároveň signalizován pomocí indikátorů odpovídajících faktoru, jímž musí být násoben údaj na ručkovém měřidle pro získání skutečné hodnoty. Ručkový přístroj slouží jen jako zobrazovací prvek řízený z PWM výstupu mikrokontroléru.

2.1.3 Volba a zapojení mikrokontroléru

Pro projekt byla vybrána architektura procesoru AVR, jež je velmi dobře dokumentovaná a existuje pro ni řada softwarových nástrojů, zejména kompilátor *avr-gcc* a zprostředkovatel komunikace *avrdude*. Kolegové na Katedře elektrotechniky FEI UPCE mají rovněž s tímto typem procesoru zkušenosti: opírá se o něj mj. předmět Aplikace mikroprocesorů 1 a je na něm založen ATMEGA644KIT doktora Matouška.

Z důvodu minimalizace ceny a rozměrů bylo zvoleno fyzické provedení MCU s co nejmenším počtem vývodů nad odhadovaným minimem, zároveň byl pro možnost zasazení do snadno dostupné patice a jednoduché umístění THT součástky na jednovrstvou desku preferován standard DIP. Výsledkem je volba řady MCU ATmegaXX8-P o 28 vývodech. V rámci řady se pak procesory dělí dle provozního napětí, rychlosti a velikosti pamětí Flash, RAM a EEPROM. Pro realizaci byl zvolen nejprve nejrozšířenější a nejschopnější ATmega328PU s tím, že v závislosti na skutečném využití pamětí bude možné díky pinové kompatibilitě nahradit za menší.

Byl zvolen krystal 16 MHz, při zapojení s kondenzátory 22 pF jej řídí procesor přímo. GPIO periferie ATmega umožňuje využít interní pull-up rezistor, a přímo připojit tlačítka vůči GND. Všechny nevyužité GPIO vývody byly vyvedeny do konektorů pro pinové lišty o rozteči 2,54 mm, aby je bylo možné využít při případné stavbě zařízení jiného účelu na stejné desce plošných spojů.

2.1.4 Vstup střídavého napětí

Pro realizaci A/D převodníku byla použita zabudovaná jednotka procesoru, jež disponuje 10bitovou přesností na rozsahu 0 V až AREF. Pro přidání nutné stejnosměrné složky a snížení impedance slouží následující analogový tvarovač podle osvědčeného schématu přípravku EADC. První verze zařízení, jež počítala se vstupem síťového napětí, snížení vstupního napětí mezi svorkami L (fáze) a GND (nulový vodič) zajišťoval dělič napětí s odpory v poměru 1:99. Protože se ale ukázal vestavěný A/D převodník jako nedostatečně citlivý, aby naměřil hodnotu dělenou 300, bylo od možnosti přímého měření v síti nn upuštěno a poměr děliče nastaven na 25. Ten umožňuje vstup napětí o amplitudě až 30 V, což odpovídá plnému rozsahu přístroje (10 V RMS = $10\sqrt{2}$ V_{max} < 15 V_{max}) plus stejné amplitudě signálu jiné frekvence, například 50 Hz. Toto opatření umožní měřit přístrojem síťové napětí za předpokladu, že bude před vstupem složka 50 Hz utlumena o –28 dB, například dolní propustí RC 3. a vyššího řádu.

Jelikož dělič napětí pro neinvertující vstup operačního zesilovače (uzel označen `A1mix`) způsobuje admitanci pro střídavé napětí na děliči síťového napětí, konkrétně $\frac{1}{15\ \mathrm{k\Omega}}$, je praktická hodnota dolního rezistoru děliče nikoli 1 kΩ, ale 937,5 Ω. Pro dělení v poměru $\frac{25}{3}$ potřebujeme na horní větvi odpor $\frac{22}{3}$× větší, tedy 6875 Ω. Tato hodnota přesně odpovídá paralelnímu zapojení rezistorů 10 kΩ a 22 kΩ z řady E6.

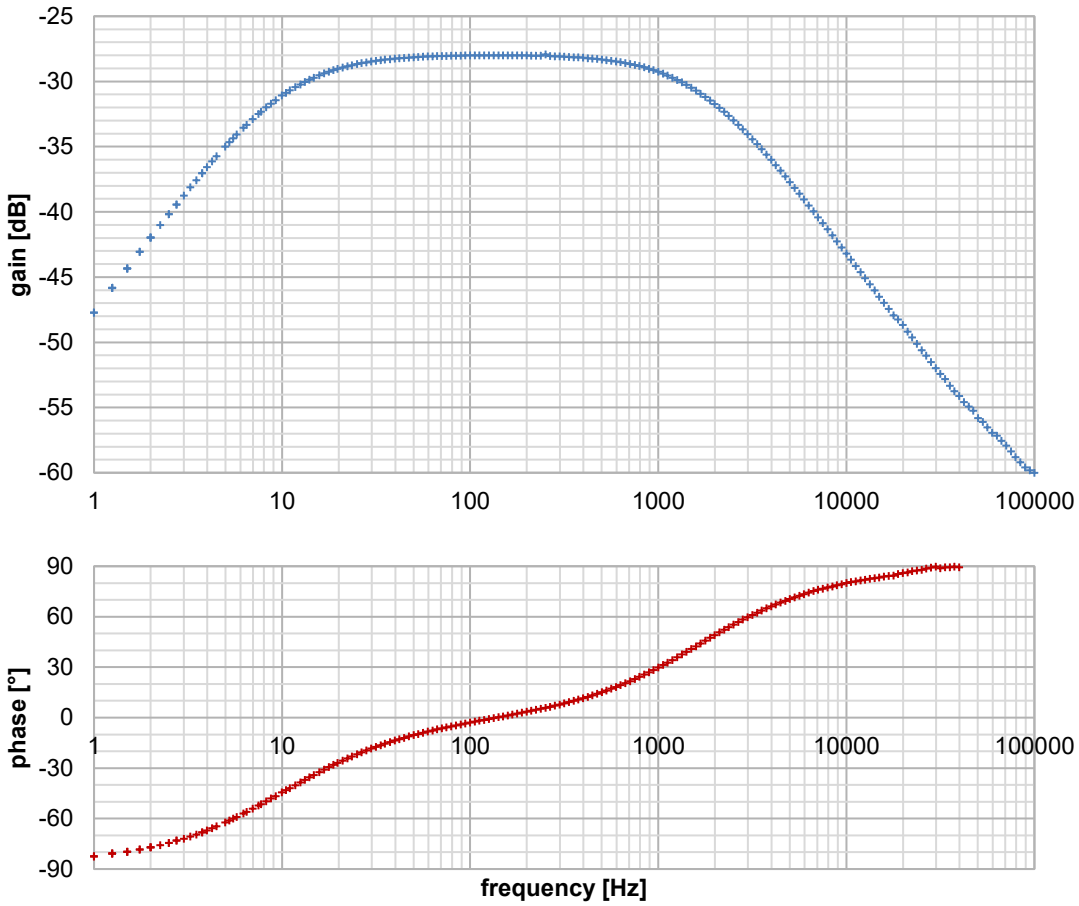
Pro zdroj referenčního napětí 2,5 V je použit klasický regulátor TL431ACZ zapojený jako Zenerova dioda. K němu i k OZ byl přidán blokovací kondenzátor, navíc je referenční napětí stabilizováno elektrolytickým kondenzátorem 220 μF.

Kromě vstupního děliče má na rozdíl od přípravku EADC tento tvarovač navíc dolní propust RC na výstupu (propustné pásmo do 1100 Hz, nepropustné pásmo zhruba od 3 kHz) tlumící signály nad Nyquistovou frekvencí, jež kvůli vzorkování nemůže číslicový obvod odlišit od zrcadlových obrazů v základním pásmu, docházelo by tedy ke zkreslení výsledku. Zpětná vazba OZ byla doplněna o dělič a tedy i možnost přestavění ze sledovače na zesilovač pro měření i velmi malých napětí, například na proudovém bočníku, pro voltmetr HDO ale nebyla využita.

Výběr operačního zesilovače je kritický, TLC272CP je jedním z nejlevnějších, které jsou pro použití v tomto zapojení vhodné. Nesouměrné napájecí napětí 5 V, které by neumožnilo pracovat mnoha starším OZ, leží s velkou rezervou v doporučeném intervalu od 3 V do 16 V. Při tomto napětí je možné na vstupy přivést napětí mezi -0,3 V až +3 V, zatímco praktický interval vstupního napětí je pouze +0,13 V až +2,37 V.

Dalším důležitým parametrem operačního zesilovače je offsetové napětí U_0. TLC272CP patří do nejnižší třídy zesilovačů TLC27x, jeho U_0 může dosahovat ±10 mV. Vzhledem k tomu, že digitálně odečítáme stejnosměrnou složku, není takto vysoká absolutní hodnota U_0 na závadu.

Analogový tvarovač má následující frekvenční charakteristiku:



Obrázek 21 – Bodeho charakteristika analogového tvarovače.

2.1.5 Výstup pro ručkový přístroj

Sada vývojových kitů doktora Matouška umožňuje analogový výstup na pásmové frekvenci signálu vysokorychlostním externím SPI D/A převodníkem. Zde ovšem vyžadujeme pouze ručkovou indikaci a vzhledem k mechanickým možnostem není třeba vyhodnocovat výslednou hodnotu více než 10x za sekundu, postačí tedy možnosti hardwarového PWM MCU ATmega. Většina vývodů s funkcí Output Compare již má využívanou jinou funkci a je možné využít pouze výstupů OC1A a OC2B. Byly proto realizovány 2 D/A kanály. Protože má ručkový stejnosměrný voltmetr FLASH STAR MU-45-DC1V odpor 40 Ω a při plné výchylce (1 V) vyžaduje proud 25 mA (nad doporučenou mezí 20 mA u ATmega), byly vytvořeny polomůstkové zesilovače pomocí běžných bipolárních tranzistorů BC846, BC856. Schéma jednoho z nich je následující:

Polomůstek rovněž umožňuje, aby měl DAC referenční napětí odlišné od MCU, bylo proto na desce vytvořeno místo pro osazení lineárního regulátoru v pouzdře TO-92.

V případě voltmetru HDO bylo na vývod PWM_REF1 namísto toho propojovacím drátem zavedeno referenční napětí 2,5 V analogového vstupu.

Pro ochranu tranzistorů během resetu procesoru, kdy by docházelo k částečnému sepnutí obou z nich, byl pin PWM_0C1A doplněn pull-up rezistorem 680 Ω proti svorce +5 V, během resetovacího stavu tedy dochází k sepnutí NPN tranzistoru a rozepnutí PNP tranzistoru, na výstupu DAC je pak napětí 0 V. K této změně došlo po výrobě desky, rezistor je proto osazen mimo pájecí plošky.

2.1.6 Datový výstup

Pro komunikaci mikrokontrolérů AVR s počítačem za běhu programu, aby bylo možné program ladit a/nebo sbírat výstupní data, je použit vestavěný sériový výstup pomocí vývodu TX. Podobně jako v diplomové práci V. Sídka s podobným zaměřením [32] byl použit optočlen pro galvanické oddělení MCU od výstupu sériového rozhraní. Běžný sériový port RS-232 nemá napájecí vývod a je zastaralý, byl proto použit tzv. převodník USB-TTL, konkrétně modul s portem USB-C a čipem CH340E. Napájí se pomocí USB a na konektor o rozteči 2,54 mm přivádí napájecí svorky 5 V a vývody sériového rozhraní na logické úrovni TTL. Při připojení optočlenu v první verzi desky došlo k inverzi polarity signálu, byla tedy nutná manuální oprava.

2.1.7 Port pro nahrání softwaru, displej

Pro nahrání softwaru do mikrokontroléru instalovaném uvnitř zařízení (ISP – *in-system programming*) se nejčastěji používá protokol SPI. Jeho vývody, spolu s napájením, jsou u AVR zpravidla uspořádány v konektoru 3x2 nebo 5x2 pinů o rozteči 2,54 mm s plastovým rámem s klíčovacím výřezem, aby se předešlo otočení o 180°. Vývod $\overline{\text{RST}}$ se využívá pro přepnutí procesoru do programovacího režimu nahrání programu, pro možnost manuálního resetu je na něj připojeno tlačítko a externí pull-up rezistor.

V původním kompaktním provedení zařízení se počítalo s využitím znakového LCD L303646-g7A145 s řadičem CoG. Tomu odpovídá pořadí vývodů konektoru displeje [33]. SPI rozhraní je připojeno přímo na odpovídající piny mikrokontroléru, pin CON pak slouží k nastavení kontrastu pomocí potenciometru.

Finální verze zařízení byla vybavena grafickým displejem známým jako „Nokia 5110 LCD“ podle jeho nasazení v mobilních telefonech této značky na přelomu tisíciletí. Jedná se o pasivní STN matici o rozlišení 84×48 px řízenou vestavěným (CoG) čipem PCD8544 disponujícím jednotkou pro sériovou komunikaci, bitmapovou RAM, oscilátorem pro řádkový multiplex a nastavitelnou napěťovou pumpou jako zdroj napětí pro řízení displeje. Stejně jako původně zamýšlený znakový displej je tento displej možné řídit asynchronně pomocí hardwarové periferie SPI v MCU.

Ačkoli se jedná o velmi běžný displej pro začátečnické projekty s mikrokontroléry, bylo pro minimalizaci využití RAM, Flash a CPU nutné jeho obsluhu řešit autorskou knihovnou vycházející z dokumentace ovladače displeje.

Displej je dodáván na DPS 45×45 mm, jež obsahuje na horní i dolní straně otvory pro konektor 8×1. Samotný displej s COG řadičem je uchycen v kovovém rámu spolu s bílou plastovou destičkou pro rozptyl světla podsvícení ze 4 SMD LED. Tyto bílé LED, v sérii s rezistorem 150 Ω pro proud 2 mA při 3,3 V, jsou zapojeny mezi vývody VCC a LIGHT, aby je bylo možné nezávisle na chodu displeje zapnout logickou 0 na samostatném GPIO vývodu procesoru.

Doporučovaný rozsah napájecího napětí řadiče je 2,7~3,3 V; napětí na logických vývodech nesmí přesáhnout napájecí napětí o více než 0,3 V. Displej je proto nutné opatřit externím regulátorem napětí, pro tuto funkci byl vybrán lineární regulátor XC6206P33 v provedení SOT-23. Vzhledem k jednostranné komunikaci ke snížení napěťové úrovně logických signálů na přibližně 3,3 V postačuje dělič z rezistorů 3,3 kΩ a 6,8 kΩ připájených na desku displeje. Spojením vývodů označených JP (LIGHT-GND) bylo podsvícení nastaveno jako vždy zapnuté. Je proto možné napájet a řídit displej pomocí pinů 2~7, odpovídajícím 6 vývodům dostupných na konektoru displeje; podsvícení displeje takto plní i funkci indikátoru napájení logických obvodů přístroje.

Pro zjednodušení návrhu desky je používán stejný vývod $\overline{\text{CS}}$ pro programování MCU a komunikaci s displejem. Pro prevenci nedefinovaného chování displeje během programování je nutné spojit vývody $\overline{\text{RST}}$ displeje s $\overline{\text{RST}}$ ATmega, jež je během programování i ručního resetu na úrovni logické 0, tedy aktivní. Pro tento účel byly obě desky doplněny jednoduchým drátovým vývodem, jež lze spojit běžnou drátovou propojkou F-F. Ve finálním zařízení se počítá s nevyužitím této propojky, pro prevenci nahodilého resetování byla proto deska displeje osazena pull-up rezistorem 10 kΩ vývodu $\overline{\text{RST}}$. Pro nutnost držet řadič ve stavu resetu po nejméně 100 ns krátce po zapnutí napájení byl vytvořen RC článek přidáním kondenzátoru 10 pF; jeho vybití je řešeno rezistorem mezi GND a VCC. Navíc při každém startu procesoru dochází k inicializaci všech používaných registrů řadiče na nominální hodnoty.

2.1.8 Návrh desky plošných spojů

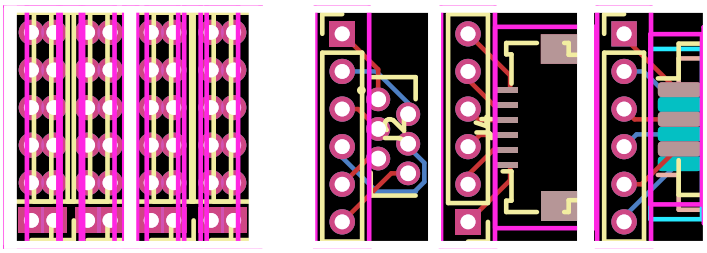
Stejně jako pro kreslení schématu byl pro návrh DPS použit open source software KiCAD. Pro zachování možnosti amatérské výroby byla deska koncipována jako jednostranná, při oboustranné výrobě pak tisk na horní straně slouží k dokumentaci polohy součástek, velké vylité plochy připojené ke kontaktu GND pak k odrušení. Velikost desky 63×57 mm odpovídá plastové krabičce Z55 ABS black (KP20A), ačkoli bylo později rozhodnuto o použití větší plechové skříně.

Jako první byl umístěn analogový vstup. Rezistory s osovými vývody DIN0207 byly zvoleny pro snadnou dostupnost s tolerancí 1 %; v horní části děliče síťového napětí, kde se počítá se ztrátovým výkonem 0,3 W, pak rezistory DIN0309. Vzhledem k dvoukanálovému provedení byla vstupní analogová část řešena symetricky – OZ, napěťová reference se zátěžovým rezistorem i jejich blokovací kondenzátory jsou umístěny v ose, pravá i levá část pak obsahuje rezistory odpovídající jednotlivým

kanálům. Pro vstup síťového napětí se používá třípólového šroubového konektoru o rozteči 5 mm, v případě využití jednoho kanálu pouze dvoupólového.

Tvorba desky pokračovala vytvořením symbolu pro modul USB-TTL a jeho umístěním do zadního rohu společně s optočlenem a odrušovacím kondenzátorem nejvyšší bezpečnostní třídy Y1. Následně bylo zvoleno umístění pro procesor takové, jež minimalizuje vzdálenost mezi výstupy analogové části a ADC vstupy MCU. Spolu s ním byl umístěn i krystal s kompenzačními kondenzátory a jeho okolí bylo prolito mědí.

Vývody nativního protokolu SPI leží na pravé horní straně procesoru, podle čehož bylo vybráno umístění konektoru displeje. Pořadí vývodů odpovídá LCD C7G045, avšak tato volba nebyla během návrhu desky finalizována: proto byla společně s ATMEGA DSP KIT na DPS skupina adaptérů s univerzálním konektorem 1×6 2,54 mm na různé způsoby montáže 1mm FFC/FPC konektoru, případně pro zprostředkování změny pořadí vývodů křížených drátovými propojkami.

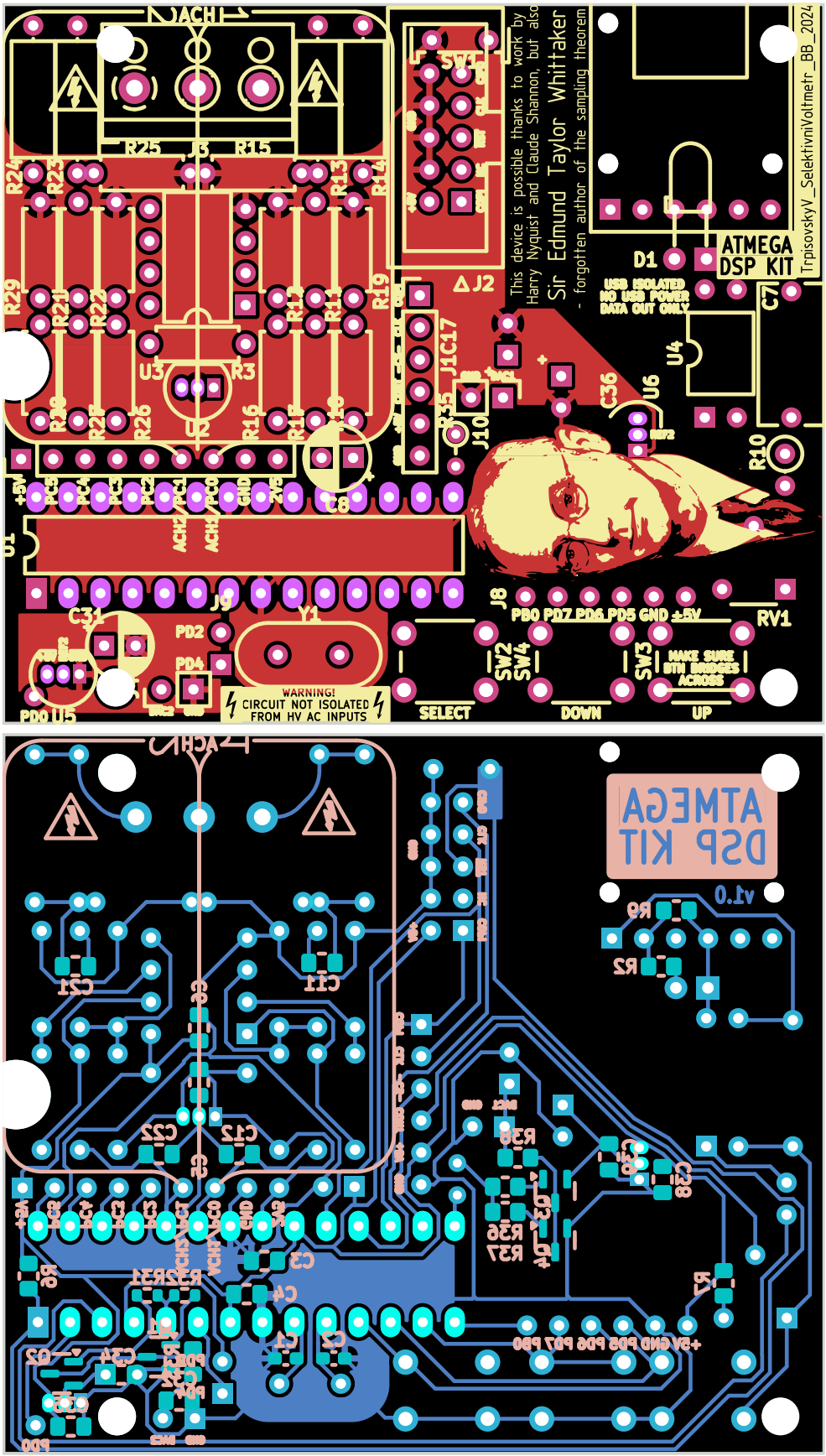


Obrázek 22 – Adaptéry pro různé SPI periferie s plochým kabelem (velikost 200 %)

Zleva doprava: 2x blok pro změnu pořadí vývodů propojkami; adaptér pro FFC konektor displeje: varianta s THT montáží, vodorovná varianta SMD, varianta s THT montáží obkročmo.

Aby se nepřekrývaly kabely programovacího portu a displejem, bylo nutné port umístit do zadní části. Musel tam proto být veden signál $\overline{\text{RST}}$ z opačné strany procesoru, podél samého okraje desky a galvanicky oddělené části. Díky manuálně vyrobenému zářezu v desce je zde umístěno resetovací tlačítko. Umístění mikrospínačů tlačítek čelního panelu bylo pro původní plastovou krabičku uvažováno přímo na DPS, a to v dolní části. Umístění výstupních analogových zesilovačů bylo zvoleno poblíže odpovídajících PWM vývodů.

Kromě označení verze a varování k síťovému napětí se autor rozhodl na zbývající ploše uctít britského fyzika Sira E. T. Whittakera, který se na začátku 20. století věnoval interpolační teorii. Z jeho práce přímo vyplývá vzorkovací teorém, jež tvoří základ digitálního zpracování signálu. Jeho portrét (Arthur Trevor Haddon, 1933) byl převeden do třístupňového vektorového obrázku a vyhotoven na vrstvě mědi a sítotisku.



Obrázek 23 – Návrh desky plošných spojů. Vrchní; spodní vrstva (zrcadlově). Měřítko 200 %.

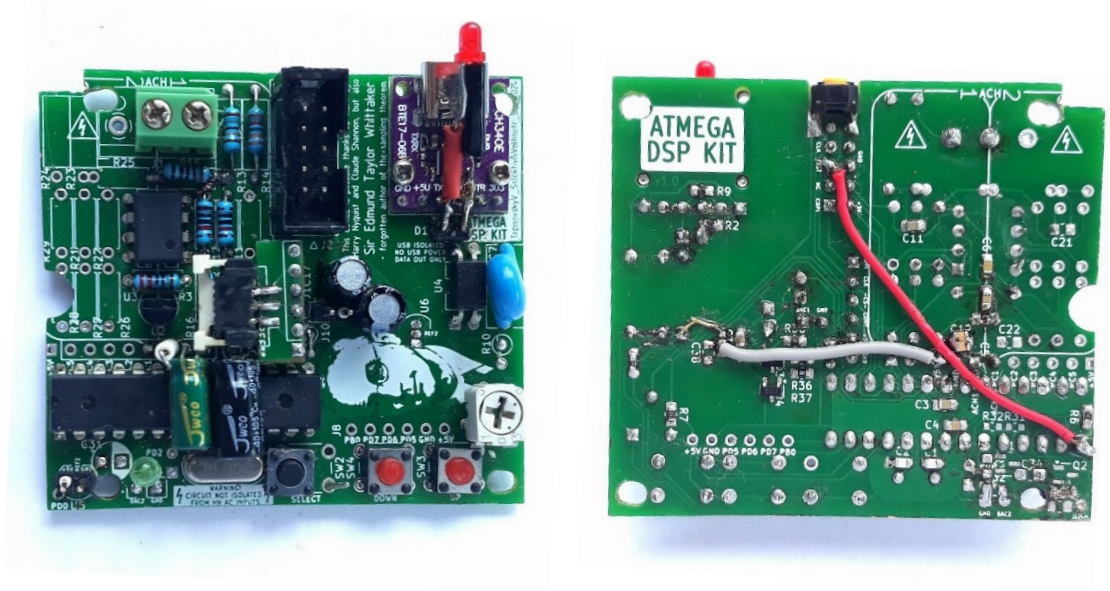
Pro napájení byl použit spínaný zdroj 5 V, 500 mA z nabíječky značky Optus. Namísto neosazeného regulátoru analogového výstupu TO-92 inline byl umístěn konektor 2×1 o rozteči 2,54 mm, kabelem s odpovídající zástrčkou byla doplněna deska plošných spojů spínaného zdroje.

Zatímco původní plastová krabička neumožňovala umístit konektor pro externí měření, prostor kovové skříně vybízí k doplnění externího vstupu. Pro tento účel byly využity zdířky pro klasické 4mm propojovací kabely s panelovou montáží a pájitelným vývodem. Aby nebylo síťové napětí přivedeno na konektor externího vstupu, bylo pro přepínání oběma vstupy naplánováno použít dvoupólový kolébkový přepínač umístěný na zadním panelu.

Pro vstup síťového napětí je využit klasický konektor C14 s panelovou montáží. Jeho zemní vývod je vodivě spojen se skříní. Na fázový vývod byla umístěna pojistka 5×20 mm s kabelovou montáží, jištěný přívod je následně veden do vypínače na čelní straně. Spínatelná fáze i nulový vodič poté pokračují do zdroje 5 V a před jeho vynecháním (viz kapitola 2.1.9) přepínače vstupů.

2.1.9 Výroba

Spolu s výše zmíněnými adaptéry a návrhy pro několik předchozích zařízení autora byla deska umístěna na standardní plochu 10×10 cm a vyrobena v Číně společností PCBWay.



Obrázek 24 – Fyzické provedení desky plošných spojů (zhruba skutečná velikost)

Na zadní straně desky jsou patrné úpravy spojů, způsobené dodatečnými změnami ve schématu.

Pro zaskříňování byla použita kovovo-plastová skříňka vnějších rozměrů 170×130×80mm. Aby nebyly z čelní části viditelné šrouby kromě rohových, byla na zadní stranu čelního plastového rámu umístěna plastová deska pro upevnění součástek panelu se šroubovou montáží: displej, deska s tlačítky, deska s LED, analogový voltmetr.

Kolébkový vypínač byl umístěn přímo do čelního panelu panelovou montáží, stejně tak vstupy a přepínač vstupů do zadního. Po odebrání možnosti přímého měření síťového napětí byl otvor pro přepínač vstupů zaslepen.

Pro vyvedení tlačítek k přednímu panelu byl použit výřez z jedné z neúspěšně vyhotovených desek plošných spojů ATMEGA DSP KIT. Obdobně byla jako univerzální DPS 2×5 bodů s roztečí 2,54 mm pro zapojení LED signalizujících rozsah použita část desky určená původně jako programovací konektor SPI.

Aby bylo možné připojit desku s třemi LED signalizujícími rozsah pouze trojžilným kabelem, byla společná katoda prvních dvou LED pro 3V a 10V rozsah zapojena přes rezistor nikoli ke svorce GND nebo VCC, ale dalšímu GPIO vývodu MCU. K jedné z nich pak byla připojena druhá LED antiparalelně a obrácením polarity napětí přiváděného na příslušný pár diod vybrat, která se rozsvítí, zatímco zbylý vývod je ve stavu vysoké impedance (High-Z), docílený ovládáním registrů DDRC. Hodnoty obou registrů pro rozsvícení příslušné LED jsou uložené v bajtovém poli a zavolány funkcí `showLED()`. Vzhledem k tomu, že vyžadujeme rozsvítit vždy jen jednu kontrolku, je možné za využití rezistoru poloviční odporu u každého vývodu a antiparalelním párem LED mezi každou jejich dvojicí ovládat až 6 různých diod, tato metoda se nazývá *charlieplexing*.

2.2 Výpočet koeficientů filtrů

2.2.1 Definice problému

Od digitálního filtru je nejprve vyžadováno odstranění nízkofrekvenčních složek, zejména 50 Hz. Jako nejvhodnější se v tomto případě jeví inverzní Čebyševův filtr 2. řádu, jež obsahuje právě jednu nulu a jeho mezní frekvenci lze nastavit tak, aby se tato nula vyskytovala právě na frekvenci 50 Hz. Nevýhodou tohoto typu filtru je omezený útlum stejnosměrné složky, ale důležitější vlastností pro selektivní voltmetr je plochá charakteristika v propustném pásmu a strmý útlum v nepropustném pásmu. Při volbě parametrů filtru jde o kompromis mezi strmostí útlumu a přenosu v nepropustném pásmu; hodnota útlumu byla zvolena 20 dB.

Pro každý z kmitočtů HDO používaných v ČR poté vytvoříme úzkopásmovou propust, aby byl tento kmitočet co nejlépe izolován od rušivých frekvencí, zejména lichých harmonických síťového napětí, jež dosahují amplitudy srovnatelné se signálem HDO. Ačkoli je i zde žádoucí dosáhnout co nejstrmějšího útlumu v nepropustném pásmu a ploché charakteristiky v propustném pásmu, nebyly první pokusy s Čebyševovým filtrem úspěšné, a bylo tedy přistoupeno na manuální návrh Butterworthova filtru.

Praktickým měřením bylo určeno, že na AVR procesoru taktovaném na 16 MHz výpočet trojice bikvadů s přesností 16 bitů (tedy násobením 16bitových hodnot 16bitovými koeficienty pro součin šířky 32 bitů) trvá zhruba 180 µs. Jako vzorkovací frekvence byla proto zvolena f_s = 5000 Hz, odpovídající periodě 200 µs.

2.2.2 Výpočet koeficientů pomocí MATLABu

Nejjednodušší způsob návrhu Čebyševova filtru je použití speciálního programu. Standardním řešením je použití programu MATLAB; open source alternativou je například Python knihovna `scipy.signal`.

Podobně jako v algebraických modelech i v MATLABu existuje několik modelů číslicového filtru v závislosti na tom, zda pracujeme v prostoru času *t*, Laplaceově proměnné *s* nebo časově diskrétní proměnné *z*. Jak napovídá název programu MATLAB (Matrix Laboratory), jsou reprezentovány maticemi (příp. jednořádkovými, tj. vektory):

1. `[b, a]`: model přenosové funkce (*transfer function model, tf model*)

   Tento model reprezentuje přenosovou funkci IIR filtru pomocí vektorů *a*, *b* o (*n*+1) rozměrech, kde *n* je počet řádů filtru. Například pro filtr 3. řádu tedy nalezneme koeficienty $a = [a_0; a_1; a_2; a_3]$ a $b = [b_0; b_1; b_2; b_3]$. Vzhledem k indexování od 1 platí a_i = `a(i+1)`; b_i = `b(i+1)`.

2. `[z, p, k]`: model nul, pólů a zesílení (*zero-pole-gain model, zpk model*)

   Tento model reprezentuje IIR filtr pomocí polohy nul a pólů, kde *z* a *p* jsou vektory obsahující komplexní čísla odpovídající souřadnicím nul a pólů v Laplaceově *s*-prostoru. V kapitole 1.2.4 *Kritérium stability filtru* je ukázáno, jaký vliv má poloha pólů na stabilitu filtru. Naopak nuly, jež se vyskytují například u inverzních Čebyševových filtrů, odpovídají frekvencím, u nichž filtr dosahuje nulového přenosu. Poloha pólů a nul ovšem rozhoduje pouze o relativním přenosu filtru, ten musí být pro dosažení konkrétní absolutní hodnoty specifikován reálným číslem *k* – koeficientem zesílení. Některé funkce jej nevyžadují, v případě vynechání se předpokládá *k*=1. Při práci s hodnotami citlivými na přesnost, například při návrhu filtrů řádů 15 a výše, je doporučeno využít tento model a až po dokončení potřebných operací převést např. na přenosovou funkci, neboť dosahuje druhé nejvyšší numerické stability (po stavovém popisu systému)[17].

3. `sos`: model článků 2. řádu (*second-order-section model, sos model*)

   Tento model reprezentuje IIR filtr pomocí polohy nul a pólů, kde *z* a *p* jsou vektory obsahující komplexní čísla odpovídající souřadnicím nul a pólů v Laplaceově *s*-prostoru. V kapitole 1.2.4 *Kritérium stability filtru* je ukázáno, jaký vliv má poloha pólů na stabilitu filtru. Naopak nuly, jež se vyskytují například u inverzních Čebyševových filtrů, odpovídají frekvencím, u nichž filtr dosahuje nulového přenosu. Poloha pólů a nul ovšem rozhoduje pouze o relativním přenosu filtru, ten musí být pro dosažení konkrétní absolutní hodnoty specifikován reálným číslem *k* – koeficientem zesílení.

4. `[A, B, C, D]`: stavový popis systému (*state-space representation, ss model*)

Tento model reprezentuje IIR filtr jako čtyři matice:

- $\boldsymbol{A}$ – čtvercová matice vnitřních vazeb systému (tzv. systémová),
- $\boldsymbol{B}$ – matice zpětných vazeb systému na vstup (tzv. vstupní),
- $\boldsymbol{C}$ – matice vazeb výstupu na systém (tzv. výstupní),
- $\boldsymbol{D}$ – matice přímých vazeb vstupu na výstup (tzv. přímá).

Používá se v systémovém řízení a automatizaci. Pro návrh jednoduchých číslicových filtrů není používání stavové reprezentace podstatné.

Mezi reprezentacemi filtrů je možné provádět převody pomocí funkcí jako `[b, a] = zpk2tf(z, p, k)`, nazvaných podle vstupu a výstupu převodu, v tomto případě se jedná o zkratku *zero-pole-gain to transfer function*.

MATLAB nabízí funkci `cheby2`, a pomocí ní navrhujeme filtr podle požadavků. Nejprve pomocí metody půlení určíme mezní frekvenci takovou, aby se nula inverzního Čebyševova filtru nacházela na frekvenci 50 Hz. Zanedbáme-li výstřednost elipsy, kde leží póly Čebyševova filtru (tedy uvažujme kružnici jako u Butterworthova filtru), jedná se o frekvenci $\frac{50\text{ Hz}}{\sin 45°} = 50\sqrt{2}\text{ Hz} \doteq 70{,}711\text{ Hz}$.

Nejprve vzorkovací frekvenci označíme f_s:

```
>> fs = 5000; % sampling frequency constant
```

Nyní definujeme funkci `f(x)` procházející nulou na frekvenci `x` takové, kde se nula filtru nachází na frekvenci 50 Hz:

```
fs = 5000
function [y] = f(x)
    [z,p,k] = cheby2(2,20,x/(fs/2),'high');
    y = fs*angle(z(1))/(2*pi) - 50;
end
```

Implementace metody půlení vycházející z materiálů k předmětu BAPMA (J. Zahrádka):

```
function [result] = halving(a,b,tolerance)
    i=0;
    while abs(f(x))>tolerance
        x = (a+b) / 2;
       if f(a)*f(x)<0
            b = x;
       else
            a = x;
       end
        i=i+1;
        if i>1000
            break;
        end
    end
    result = x
end
```

Pomocí této funkce byla určena mezní frekvence filtru: 70,6874 Hz. Vypočítaný filtr si necháme vizualizovat a převést na formu koeficientů bikvadu:

```
function [] = plotfreqz(left, right, bottom, top)
    close all;
    freqz(sos,fs*10,'whole',fs);
    ax=findall(gcf,'Type','axes');
    set(ax,'XLim',[left right]); set(ax(1),'YLim',[bottom top]);
end

>> [z,p,k] = cheby2(2,20,70.6874/(fs/2),'high');
>> sos = zp2sos(z,p,k)

sos =
          b0          b1          b2          a0          a1          a2
  0.87559659  -1.7477376  0.87559659           1  -1.7321875  0.76674327

>> plotfreqz(0 0.2 -80 10);
```

Nyní realizujeme filtry typu pásmová propust. První pokus o využití inverzního Čebyševova filtru nebyl úspěšný, bylo tedy přistoupeno na využití Butterworthova filtru.

Prvním krokem je definice tolerančních polí filtru. Očekáváme pokles o $A_{\mathrm{C}} = 3$ dB v propustném pásmu $f_{\mathrm{C1}} = 170$ Hz až $f_{\mathrm{C2}} = 1200$ Hz, v nepropustném pásmu pro kmitočty $f_{\mathrm{S1}} = 130$ Hz a $f_{\mathrm{S2}} = 3000$ Hz pak útlum alespoň: $A_{\mathrm{S}} = 20$ dB.

V případě filtrů propouštějících konkrétní frekvenci HDO se jedná o pásmové propusti. Definujeme si pro každou z nich dolní i horní hranici tak, aby se nejbližší harmonický kmitočet nominální síťové frekvence nacházel v nepropustném pásmu, a frekvence předkreslíme na kmitočty Ω v oblasti *z* pomocí rovnice $\widetilde{\Omega} = \frac{2}{T_{\mathrm{VZ}}} \cdot \mathrm{tg}\frac{2\pi f \cdot T_{\mathrm{VZ}}}{2}$:

Frekvence HDO [Hz]	Označení mezního kmitočtu	Hodnota f [Hz]	Označení mezního kmitočtu	Hodnota $\widetilde{\Omega}$
$183\frac{1}{3}$	$f_{183\mathrm{S1}}$	150	$\widetilde{\Omega}_{183\mathrm{S1}}$	945,28
	$f_{183\mathrm{C1}}$	$173,\overline{3}$	$\widetilde{\Omega}_{183\mathrm{C1}}$	1093,41
	$f_{183\mathrm{C2}}$	$193,\overline{3}$	$\widetilde{\Omega}_{183\mathrm{C2}}$	1220,76
	$f_{183\mathrm{S2}}$	250	$\widetilde{\Omega}_{183\mathrm{S2}}$	1583,84
$216\frac{2}{3}$	$f_{217\mathrm{S1}}$	150	$\widetilde{\Omega}_{217\mathrm{S1}}$	945,28
	$f_{217\mathrm{C1}}$	$206,\overline{6}$	$\widetilde{\Omega}_{217\mathrm{C1}}$	1305,87
	$f_{217\mathrm{C2}}$	$226,\overline{6}$	$\widetilde{\Omega}_{217\mathrm{C2}}$	1433,90
	$f_{217\mathrm{S2}}$	250	$\widetilde{\Omega}_{217\mathrm{S2}}$	1583,84
$283\frac{1}{3}$	$f_{283\mathrm{S1}}$	250	$\widetilde{\Omega}_{283\mathrm{S1}}$	1583,84
	$f_{283\mathrm{C1}}$	$273,\overline{3}$	$\widetilde{\Omega}_{283\mathrm{C1}}$	1734,49
	$f_{283\mathrm{C2}}$	$293,\overline{3}$	$\widetilde{\Omega}_{283\mathrm{C2}}$	1864,22
	$f_{283\mathrm{S2}}$	350	$\widetilde{\Omega}_{283\mathrm{S2}}$	2235,26
760	$f_{760\mathrm{S1}}$	650	$\widetilde{\Omega}_{760\mathrm{S1}}$	4327,39
	$f_{760\mathrm{C1}}$	720	$\widetilde{\Omega}_{760\mathrm{C1}}$	4860,05

	f_{760C2}	780	$\widetilde{\Omega}_{760C2}$	5335,02
	f_{760S2}	850	$\widetilde{\Omega}_{760S2}$	5913,98
1060	f_{1060S1}	950	$\widetilde{\Omega}_{1080S1}$	6795,99
	f_{1060C1}	1020	$\widetilde{\Omega}_{1080C1}$	7459,20
	f_{1060C2}	1080	$\widetilde{\Omega}_{1080C2}$	8063,22
	f_{1060S2}	1150	$\widetilde{\Omega}_{1080S2}$	8816,19

Tabulka 3 – Zvolené frekvence okrajů pásem filtru a jejich převod do Ω

2.2.3 Řešení rovnice

Činitel širokopásmovosti filtrů $\frac{f_{C2}}{f_{C1}}$ je pro jednotlivé frekvence:

f_{HDO} [Hz]	$183\frac{1}{3}$	$216\frac{2}{3}$	$283\frac{1}{3}$	760	1060
$\frac{f_{C2}}{f_{C1}}$	1,022	1,019	1,014	1,013	1,009

Tabulka 4 – Činitel širokopásmovosti filtrů

Ve všech případech je činitel širokopásmovosti menší než 2, bude se tedy filtr realizovat jako pásmová propust.

U každého z filtrů vypočítáme středovou frekvenci, šířku pásma a normované kmitočty potlačení:

f_{HDO} [Hz]	$183\frac{1}{3}$	$216\frac{2}{3}$	$283\frac{1}{3}$	760	1060
středový kmitočet $\Omega_O = \sqrt{\widetilde{\Omega}_{C1} \cdot \widetilde{\Omega}_{C2}}$	1155,33	1368,39	1798,19	5092,00	7755,33
šířka pásma $\Delta\Omega = \widetilde{\Omega}_{C2} - \widetilde{\Omega}_{C1}$	127,35	128,02	129,73	474,96	604,02
Dolní normovaný kmitočet potlačení $\Omega_{S1} = \frac{\lvert\widetilde{\Omega}_{S1}^2 - \widetilde{\Omega}_O^2\rvert}{\widetilde{\Omega}_{S1} \cdot \Delta\Omega}$	3,665	8,089	3,528	3,504	3,401
Horní normovaný kmitočet potlačení $\Omega_{S2} = \frac{\lvert\widetilde{\Omega}_{S2}^2 - \widetilde{\Omega}_O^2\rvert}{\widetilde{\Omega}_{S2} \cdot \Delta\Omega}$	5,819	3,137	6,079	3,221	3,301
Činitel selektivity k (menší z Ω_{S1}, Ω_{S2})	3,665	3,137	3,528	3,221	3,301

Tabulka 5 – Parametry filtrů v prostoru *z*

Pro každý filtr volíme stejný útlum $A_C = 3$ dB v okrajích propustného pásma a $A_S = 20$ dB pro zvolené frekvence, vychází tedy tentýž útlumový činitel

$$d = \frac{10^{\frac{A_S}{10}} - 1}{10^{\frac{A_C}{10}} - 1} = \frac{10^{\frac{20}{10}} - 1}{10^{\frac{3}{10}} - 1} \doteq \frac{99}{0{,}9953} \doteq 99{,}471$$

Řád filtru normované dolní propusti n, jež splňuje požadavky návrhu, získáme dosazením:

$$n \geq \frac{\log d}{2 \log k}$$

f_{HDO} [Hz]	$183\frac{1}{3}$	$216\frac{2}{3}$	$283\frac{1}{3}$	760	1060
$n \geq \frac{\log d}{2 \log k}$	1,77	2,01	1,82	1,97	1,93

Tabulka 6 – Potřebný řád filtru pro dosažení stanoveného útlumu

Během realizace bylo zjištěno, že ačkoli je pro dosažení útlumu –20 dB někdy nedostačující, pro použití zvoleného procesoru musíme slevit na filtr 2. řádu. Vypočítáme, s jakým útlumovým činitelem můžeme pro n=2 počítat pro dolní i horní část pásmové propusti, a jakému bude odpovídat útlumu A_S za zanechání A_C = 3 dB.

Vzorce pro zpětný výpočet dosažitelného útlumu z řádu filtru:

$$\log d \leq 2n \log k$$

$$d = k^{2n}$$

$$d = \frac{10^{\frac{A_S}{10}} - 1}{10^{\frac{A_C}{10}} - 1}$$

$$10^{\frac{A_S}{10}} - 1 = d\left(10^{\frac{A_C}{10}} - 1\right)$$

$$10^{\frac{A_S}{10}} = 10^{\frac{A_C}{10}} d - d + 1$$

$$A_S = 10 \log\left(10^{\frac{A_C}{10}} d - d + 1\right)$$

f_{HDO} [Hz]	$183\frac{1}{3}$	$216\frac{2}{3}$	$283\frac{1}{3}$	760	1060
d_1	180,5	4281,7	154,9	150,8	133,7
d_2	1146,9	96,8	1365,9	107,6	118,8
A_{S1}	22,57	36,30	21,91	21,79	21,27
A_{S2}	30,58	19,88	31,34	20,34	20,76

Tabulka 7 – Dosažitelný útlum s filtrem realistického řádu

V nejhorším případě budeme muset počítat s útlumem dosahujícím pouhých 35,82 dB v nepropustném pásmu.

Nyní vypočítáme koeficienty jednotlivých filtrů v oblasti z pomocí bilineární transformace z oblasti s. Pro řád filtru $n = 2$ získáme z tabulky koeficienty $a_0 = 1; a_1 = \sqrt{2}; a_2 = 1$. Filtrační funkce bude tedy odpovídat rovnici

$$F(s) = \frac{1}{\sum_{i=0}^{n} a_i s^i} = \frac{1}{a_0 + a_1 s + a_2 s^2} = \frac{1}{1 + \sqrt{2}s + s^2}$$

Pro převod dolní propusti na pásmovou využijeme bilineární transformaci

$s \to \beta \frac{z^2 - 2\alpha z + 1}{z^2 - 1}$, kde $\alpha = \frac{4 - \Omega_O^2 T^2}{4 + \Omega_O^2 T^2}; \beta = \frac{4 + \Omega_O^2 T}{2\Delta\Omega T}$

Jako příklad je uveden výpočet filtrační funkce pro $f_{\mathrm{HDO}} = 216\frac{2}{3}$ Hz, pro ostatní frekvence pak pouze koeficienty v tabulce 8.

$$\alpha = \frac{4 - \Omega_O^2 T^2}{4 + \Omega_O^2 T^2} = \frac{4 - \frac{1368{,}39^2}{5000^2}}{4 + \frac{1368{,}39^2}{5000^2}} = 0{,}96324$$

$$\beta = \frac{4 + \Omega_O^2 T}{2\Delta\Omega T} = \frac{4 + \frac{1368{,}39^2}{5000}}{\frac{2 \cdot 128{,}02}{5000}} = 79{,}573$$

Transformací do oblasti z dostáváme lomený kvartický polynom:

$$F(s) = \frac{1}{1 + \sqrt{2}s + s^2} \to F(z) = \frac{1}{1 + \sqrt{2}\beta \frac{z^2 - 2\alpha z + 1}{z^2 - 1} + \left(\beta \frac{z^2 - 2\alpha z + 1}{z^2 - 1}\right)^2} =$$

$$= \frac{z^4 - 2z^2 + 1}{(z^4 - 2z^2 + 1) + \left((\sqrt{2}\beta)z^4 + (-2\sqrt{2}\alpha\beta)z^3 + (2\sqrt{2}\alpha\beta)z + (-\sqrt{2}\beta)\right) + \\ +((\beta^2)z^4 + (-4\alpha\beta^2)z^3 + (4\alpha^2\beta^2 + 2\beta^2)z^2 + (-4\alpha\beta^2)z + (\beta^2))} =$$

$$= \frac{z^4 - 2z^2 + 1}{(\beta^2 + \sqrt{2}\beta + 1)z^4 + (-4\alpha\beta^2 - 2\sqrt{2}\alpha\beta)z^3 + (4\alpha^2\beta^2 + 2\beta^2 - 2)z^2 + \\ +(-4\alpha\beta^2 + 2\sqrt{2}\alpha\beta)z + (\beta^2 - \sqrt{2}\beta + 1)}$$

Koeficienty polynomu v čitateli označíme k_0 až k_4 a vyčíslíme pro $f_{\mathrm{HDO}} = 216\frac{2}{3}$ Hz:

$$\frac{z^4 - 2z^2 + 1}{a'_0 z^4 + a'_1 z^3 + a'_2 z^2 + a'_3 z + a'_4}$$

$$a'_0 = \beta^2 + \sqrt{2}\beta + 1 = 6445{,}4$$

$$a'_1 = -4\alpha\beta^2 - 2\sqrt{2}\alpha\beta = -24613{,}3$$

$$a'_2 = 4\alpha^2\beta^2 + 2\beta^2 - 2 = 36161{,}5$$

$$a'_3 = -4\alpha\beta^2 + 2\sqrt{2}\alpha\beta = -24179{,}8$$

$$a'_4 = \beta^2 - \sqrt{2}\beta + 1 = 6220{,}4$$

Pro ostatní frekvence jsou jejich hodnoty uvedeny v tabulce 8.

f_{HDO} [Hz]	α	β	a'_0	a'_1	a'_2	a'_3	a'_4
$183\frac{1}{3}$	0,97366	79,573	6445,4	–24879,5	36672,5	–24441,3	6220,4
$216\frac{2}{3}$	0,96324	79,573	6445,4	–24613,3	36161,5	–24179,8	6220,4
$283\frac{1}{3}$	0,93736	79,573	6445,4	–23952,0	34915,6	–23530,0	6220,4
760	0,58820	26,513	741,45	–1698,03	2376,74	–1609,81	666,46
1060	0,24887	26,513	741,45	–718,43	1578,05	–681,10	666,46

Tabulka 8 – Nenormované koeficienty kvartických polynomů pásmové propusti 2.+2. řádu

Nyní vydělíme koeficienty hodnotou a'_0 v čitateli i jmenovateli tak, aby vznikly koeficienty a_0 až a_4 ve jmenovateli a b_0 až b_4 v čitateli normované na $a_0 = 1$. Zároveň snížíme mocniny proměnné z na nekladné hodnoty odpovídající předchozím vzorkům:

$$\frac{z^4 - 2z^2 + 1}{a'_0 z^4 + a'_1 z^3 + a'_2 z^2 + a'_3 z + a'_4} \cdot \frac{\frac{1}{a'_0} z^{-4}}{\frac{1}{a'_0} z^{-4}} =$$

$$= \frac{b_0 + b_2 z^{-2} + b_4 z^{-4}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}} = \frac{Y(z)}{X(z)}$$

Pro převod na kanonickou formu s jedinou zpožďovací linkou p až p_4 zavedeme pomocnou proměnnou $P(z)$ a rozšíříme rovnici

$$\frac{Y(z)}{P(z)} \cdot \frac{P(z)}{X(z)} = (b_0 + b_2 z^{-2} + b_4 z^{-4}) \frac{1}{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}}$$

Tato lze rozdělit na

$$\frac{Y(z)}{P(z)} = b_0 + b_2 z^{-2} + b_4 z^{-4};\ \frac{P(z)}{X(z)} = \frac{1}{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}}$$

Pro každou frekvenci HDO uvádím výslednou tabulku normovaných koeficientů; nulové hodnoty $b_1 = b_3 = 0$ jsou vynechány:

f_{HDO} [Hz]	$b_{04}[10^{-4}]$	$b_2\ [10^{-4}]$	a_0	a_1	a_2	a_3	a_4
$183\frac{1}{3}$	1,5515	3,1030	1	–3,86002	5,68969	–3,79202	0,96508
$216\frac{2}{3}$	1,5515	3,1030	1	–3,81872	5,61040	–3,75145	0,96508

283 $\frac{1}{3}$	1,5515	3,1030	1	–3,71611	5,41710	–3,65065	0,96508
760	13,4871	26,9742	1	–2,29015	3,20554	–2,17117	0,89886
1060	13,4871	26,9742	1	–0,96895	2,12834	–0,91861	0,89886

Tabulka 9 – Normované koeficienty kvartického polynomu pásmové propusti 2.+2. řádu

Z definice *z*-prostoru vyplývá, že z^{-n} je vzorek z *n*-té předcházející periody. Přenosovou funkci lze tedy implementovat v kanonické formě se zpožďovací řadou o 4 členech. Pro dosažení vyšší stability filtru použijeme funkci tf2sos v SW balíčku pro zpracování signálu, jako SciPy nebo MATLAB, která převádí přenosovou funkci (transfer function, tf) na bikvady (second-order sections, `sos`).

Kód MATLABu pro získání koeficientů je následující:

```
format("longG");
f = 283 + 1/3; bw = 20;
[b, a] = butter(2,[f-bw/2 f+bw/2]/fs*2,"bandpass"); sos = tf2sos(b, a)
```

Pro každou frekvenci získáme matici `sos` o velikosti 6×2, s koeficienty v pořadí:

b_0	b_1	b_2	a_0	$-a_1$	$-a_2$
`khw`	`khw1`	`khw2`	`kwz`	`-kww1`	`-kww2`
`kyp`	`kyp1`	`kyp2`	`kph`	`-kpp1`	`-kpp2`

Koeficienty bikvadu 1 jsou pro jednotlivé frekvence HDO následující:

f_{HDO} [Hz]	b_0 [10^{-4}]	b_1 [10^{-4}]	b_2 [10^{-4}]	a_0	a_1	a_2
183 $\frac{1}{3}$	1,55148	–3,10297	1,55148	1	–1,92534	0,98172
216 $\frac{2}{3}$	1,55148	–3,10297	1,55148	1	–1,90409	0,98183
283 $\frac{1}{3}$	1,55148	–3,10297	1,55148	1	–1,85152	0,98197
760	6,09855	–12,19709	6,09855	1	–1,10622	0,96465
1060	6,09855	–12,19709	6,09855	1	–0,43077	0,96493

Tabulka 10 – Koeficienty 1. bikvadu pásmové propusti pro jednotlivé frekvence HDO

Koeficienty bikvadu 2 jsou pro jednotlivé frekvence HDO následující:

f_{HDO} [Hz]	b_0 [10^{-4}]	b_1 [10^{-4}]	b_2 [10^{-4}]	a_0	a_1	a_2
183 $\frac{1}{3}$	1	2	1	1	–1,93468	0,98305
216 $\frac{2}{3}$	1	2	1	1	–1,91463	0,98294
283 $\frac{1}{3}$	1	2	1	1	–1,86459	0,98280
760	1	2	1	1	–1,16375	0,96551
1060	1	2	1	1	–0,49872	0,96523

Tabulka 11 – Koeficienty 2. bikvadu pásmové propusti pro jednotlivé frekvence HDO

Koeficienty v čitateli (b) jsou přímo úměrné zesílení bikvadu, jež se v jejich kaskádě násobí. Software automaticky volí zesílení prvního bikvadu tak, aby všechny ostatní měly v čitateli standardní posloupnost začínající jedničkou. Pro zvýšení citlivosti filtru a nižší náchylnost na nasycení aritmetiky ovšem byly koeficienty b druhého bikvadu 256x, tedy o 8 binárních řádů sníženy, a odpovídajícím způsobem byly zvýšen přenos prvního bikvadu pro dosažení 0 dB v propustném pásmu. Získáváme tedy následující tabulku koeficientů:

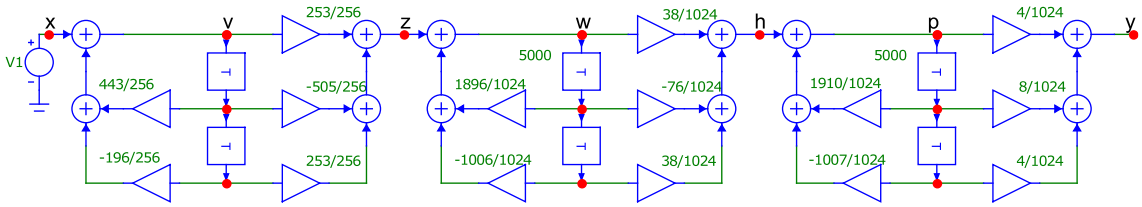
f_{HDO} [Hz]	b_0 (khw)	b_1 (khw1)	b_2 (khw2)	a_0 (kwz)	a_1 (-kww1)	a_2 (-kww2)
$183\frac{1}{3}$	40	-80	40	1024	-1972	1005
$216\frac{2}{3}$	40	-80	40	1024	-1950	1005
$283\frac{1}{3}$	38	-76	38	1024	-1896	1006
760	141	-282	141	1024	-1133	988
1060	141	-282	141	1024	-441	988

Tabulka 12 – Kvantované koeficienty 1. bikvadu PP pro jednotlivé frekvence HDO

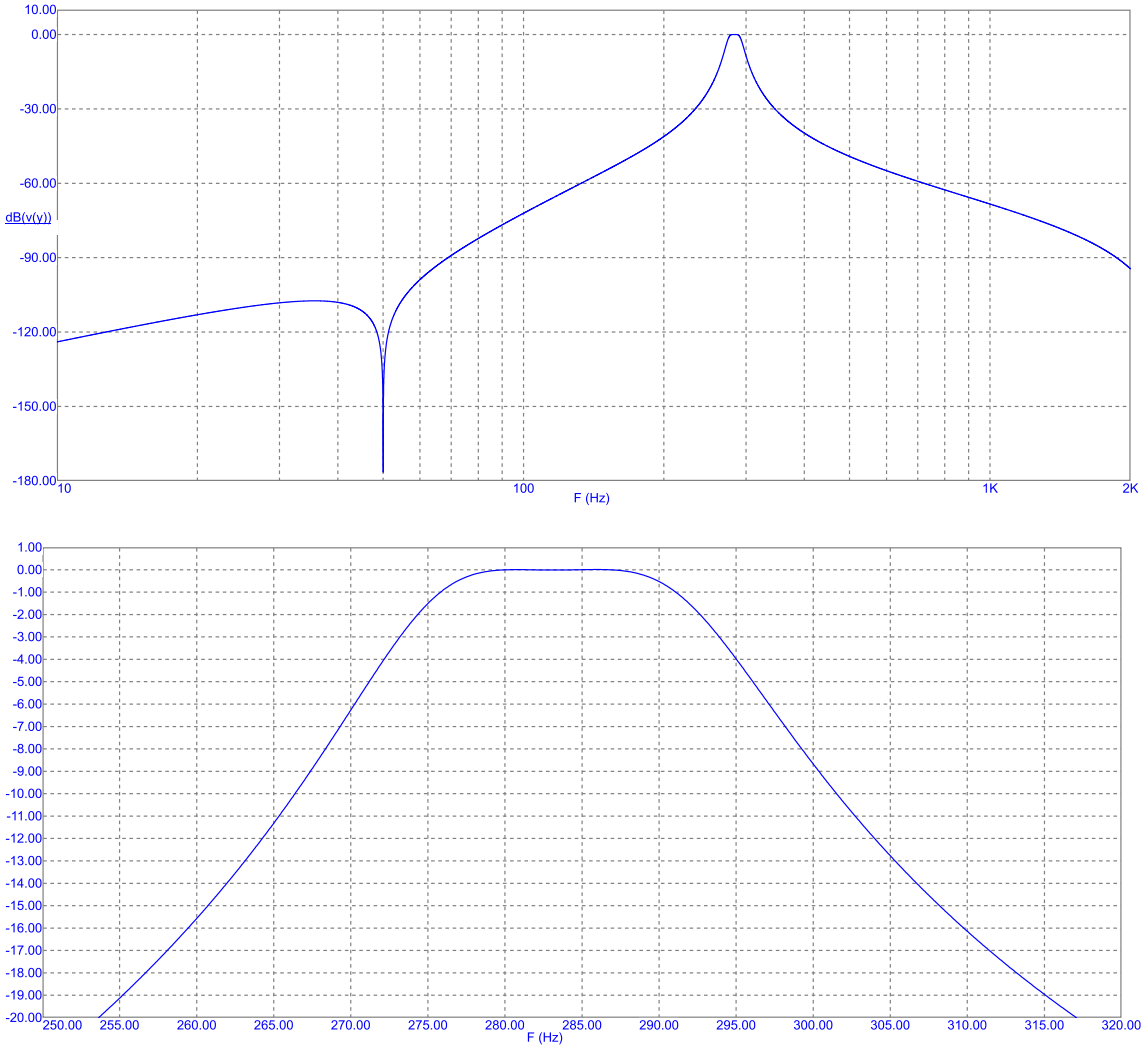
f_{HDO} [Hz]	b_0 (kyp)	b_1 (kyp1)	b_2 (kyp2)	a_0 (kph)	a_1 (-kpp1)	a_2 (-kpp2)
$183\frac{1}{3}$	4	8	4	1024	-1981	1007
$216\frac{2}{3}$	4	8	4	1024	-1960	1006
$283\frac{1}{3}$	4	8	4	1024	-1910	1007
760	4	8	4	1024	-1192	989
1060	4	8	4	1024	-510	988

Tabulka 13 – Kvantované koeficienty 2. bikvadu PP pro jednotlivé frekvence HDO

Filtr je následně simulován v programu MicroCAP:



Obrázek 25 – Schéma filtru 283⅓ Hz pro simulaci v programu MicroCAP



Obrázek 26 – Výsledek simulace filtru 283⅓ Hz; detail propustného pásma

Při praktickém měření došlo k chybě zesílení, tedy výsledku úměrnému skutečné hodnotě s koeficientem odlišným od 1. Do firmwaru byly tedy přidány kalibrační parametry; při kalibraci je možné zvolit binární řád zesílení obou bikvadů tak, aby nedocházelo k saturaci aritmetiky, a zároveň chyba zesílení spadala mezi 1 a 2. To umožňuje přesnější nastavení přenosu osmibitovým kompenzační, koeficientem, jenž zároveň opravuje neplochou charakteristiku analogové předděličky, s hodnotu mezi 128 a 255.

Pro proměnné, se kterými se ve filtru pracuje, volím pro přehlednost a soulad s učebnicí jednopísmenné zkratky. Filtr IIR popsaný v kapitole 5 (Matoušek, 2014)[7] je tvořen kaskádním zapojením bikvadů (horní propust, dolní propust) a používá následující písmena:

- `x`: vstupní proměnná filtru, tedy prvního bikvadu – horní propusti,
- `v`: mezivýsledek bikvadu horní propusti,
- `z`: výstup bikvadu horní propusti; vstup bikvadu dolní propusti,
- `p`: mezivýsledek bikvadu dolní propusti,

- `y`: výstup bikvadu dolní propusti, a tedy i filtru.

Filtry IIR v II. přímé formě používají jednu zpožďovací řadu mezivýsledků `v`, resp. `p`. V číslicovém filtru pracující v diskrétním čase se pro tento účel využívají řady proměnných. Namísto bufferu FIFO pevné velikosti se ovšem vyplatí využít techniky „rozpletení smyček“ a namísto polí využít jednotlivé proměnné. Zkompilovaný program bude mírně delší, procesor ovšem nebude muset provádět aritmetiku pointerů a časově náročné podmíněné skoky. Pro dvě předchozí hodnoty mezivýsledků tedy zavádíme proměnné `v1`, `v2`, resp. `p1`, `p2`, jimiž se s každým novým vzorkem propaguje hodnota `v`, resp. `p`. V našem případě ovšem kaskádně zapojujeme tři bikvady, dva pro horní propust a poté jeden pro dolní propust. Abychom udrželi jednopísmennou konvenci, dostanou proměnné druhého bikvadu (`v`, `z`) abecedně následující názvy (`w`, `ž`; resp. pro reprezentaci v ASCII `w`, `h`). Názvy koeficientů v programu se stejně jako v příkladném programu FRACTIONAL.C ve výše zmíněné knize odvozují následujícím způsobem:

- `k` jako „koeficient“,
- název proměnné, do které bude součet vážených sčítanců uložen (1 písmeno),
- název proměnné, jež se koeficientem násobí (1 písmeno, u opožděných i číslice).

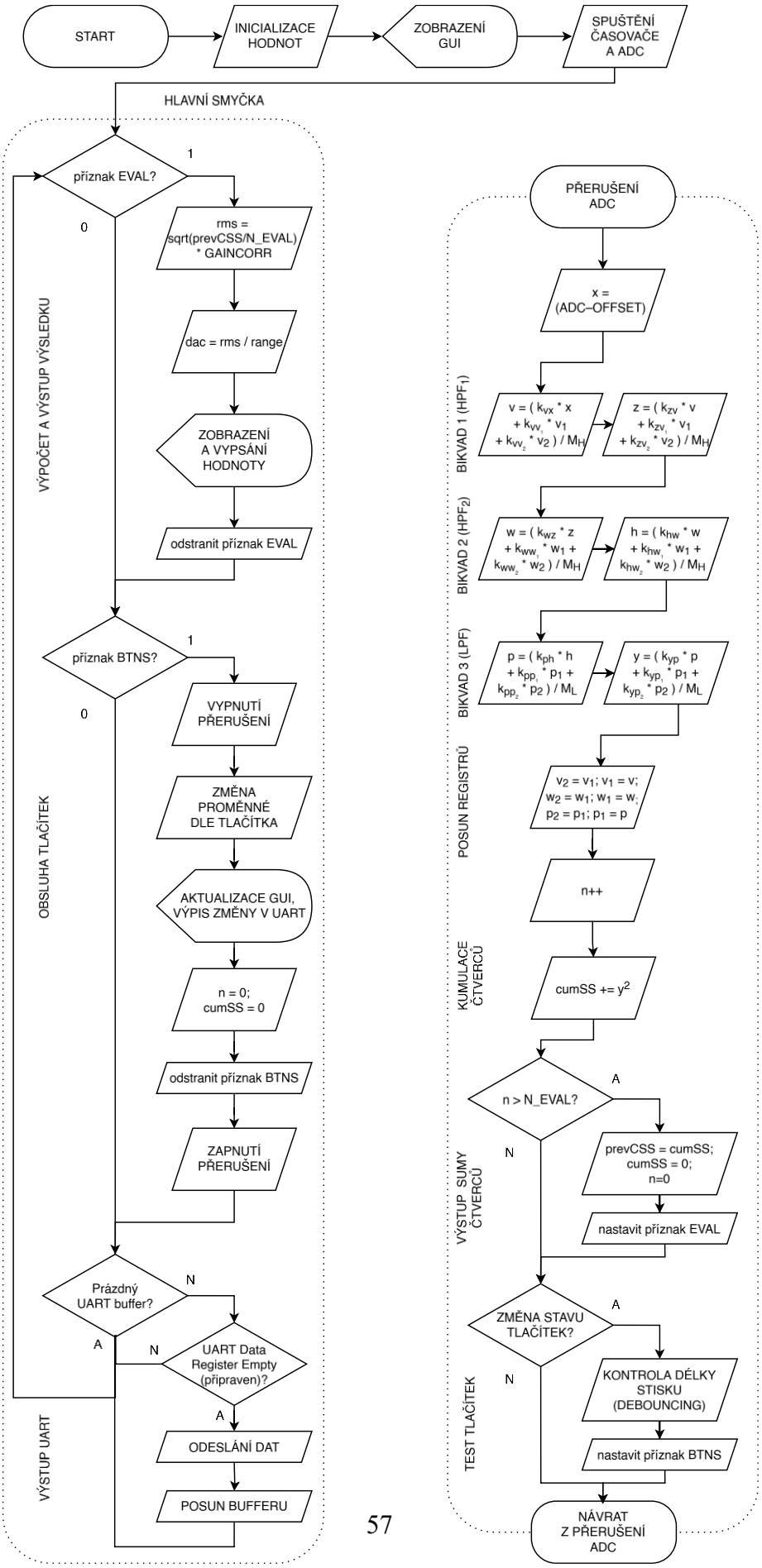
Například koeficient odpovídající dopřednému násobícímu členu b_1 v prvním bikvadu násobí předchozí hodnotu proměnné `v`, nazvanou `v1`, a výsledek součtu bude uložen do proměnné `p`, dostane tedy označení `kpv1`. Jednopísmenné názvy proměnných jsou proto nezbytné pro jednoznačné a srozumitelné značení koeficientů.

2.3 Softwarové řešení

2.3.1 Provedení číslicového filtru

Hlavním vodítkem provedení číslicového filtru byla učebnice *Aplikace algoritmů číslicového zpracování signálů 1* [7], konkrétně program FRACTIONAL.C, podle které jsou implementovány přenosové funkce bikvadů. Od tohoto programu se liší mírnými optimalizacemi a především přidáním výpočtu efektivní hodnoty (RMS) namísto analogového výstupu jednotlivých zpracovaných vzorků.

Program lze shrnout vývojovým diagramem:



Obrázek 27 – Vývojový diagram programu

2.3.2 Časování funkcí programu

Software byl navržen tak, aby prováděl měření v co nejpřesnějších intervalech, nezávisle na využití CPU. Tohoto je dosaženo spouštěním periferie ADC přímo při přerušení z čítače-časovače bez zásahu CPU [34]. Procesor je volán pouze pro zpracování dat získaných každým dokončením A/D převodu. Během obsluhy tohoto přerušení, jež trvá zhruba 160 µs z 200 µs každého cyklu, je procesoru implicitně zakázáno obsluhovat další přerušení. Po dokončení zpracovávání dat pak 40 µs probíhají ostatní operace, případně prázdná smyčka. Aby nedošlo k vynechání vzorku, je nutné zkrátit trvání obsluhy ostatních přerušení na co nejkratší dobu. Mezi operace prováděné nezávisle na zpracování vzorků patří zejména:

- operace druhá odmocnina algoritmem postupné aproximace jako součást výpočtu RMS při vyhodnocování výsledku,
- odeslání bajtu periferií UART (9600 bitů/s, odesílaných pomocí 8bitových bajtů spolu se start- a stop-bitem, tedy zhruba 1050 µs/bajt) – zdroj přerušení `TXC0` nebo `UDRE0`,
- odeslání bajtu na displej periferií SPI (4 000 000 bitů/s tvořících 8bitové bajty, tedy 2 µs/bajt) – zdroj přerušení `SPI_STC`,
- vyhodnocování stisku tlačítek – zdroj přerušení `PCINT2`.

Implementace obsluhy periferie SPI (displeje) pomocí přerušení byla označena za nepraktickou vzhledem k nutné manipulaci se zásobníkem při obsluze přerušení a vyžadování relativně velkého výstupního bufferu, je tedy řešena synchronně v hlavním vlákně programu. Naopak komunikace periferií UART počítá s odesláním zhruba 5 bajtů každých 50 ms a jeden přenos trvá více než 1 ms, není během ní proto praktické blokovat hlavní vlákno programu. Řízení procesu pomocí přerušení se ukázalo jako nespolehlivé, docházelo k vynechávání znaků. Do smyčky programu tedy byla začleněna funkce kontroly registru UCSR0A periferie UART, kde bit 5 (UART Data Register Empty) signalizuje připravenost periferie pro další výstup. Využívání právě jednoho zdroje přerušení výrazně zjednodušuje tok programu a snižuje pravděpodobnost nečekaného chování.

Tlačítka jsou používána pro ovládání jednoduchého grafického uživatelského rozhraní, kde tlačítko SELECT vybírá veličinu k nastavení a UP/DOWN mění její hodnotu v rámci rozsahu nebo seznamu. Kontrolu stavu tlačítek v pravidelných intervalech zajišťuje běh této rutiny uvnitř smyčky vzorkování. Vzhledem k použitým mikrospínačům je nutný jejich tzv. debouncing, tedy odstranění zákmitů při změně polohy plynoucích z mechanických vlastností ohebného kupolovitého disku. Metoda využívající externí RC filtr se považuje za zastaralou, dnes je běžnou praxí ošetřit odstranění sledu změn logické úrovně vstupních pinů měřením času poslední události v softwaru. Pro tento účel se pro každé tlačítko používá osmibitová proměnná jako buffer FIFO: při každém vyhodnocení se obsah posune doleva a stav tlačítka (0=rozepnuto, 1=sepnuto) je zapsán do nejnižšího bitu. Pouze v případě, že je horních 7 bitů stejných, je zapsán příznak, že došlo ke

stisknutí nebo uvolnění tlačítka. Současně existují proměnné počítající délku stisku, aby mohlo dojít k opakované akci při držení tlačítka. Tlačítko SELECT při dlouhém stisku zruší zvolení položky v menu, tlačítka UP/DOWN při dlouhém stisku začnou opakovaně provádět svoji funkci.

2.3.3 Autorská knihovna pro řízení displeje

Rozhraní displeje částečně odpovídá standardu SPI využívajícímu vývody $\overline{\text{CS}}$, SCK a COPI, kde čtení dat probíhá při náběžné hraně, počínaje MSB, o frekvenci nejvýše 4 MHz. Pro dosažení maximální přenosové rychlosti dat se výrobce rozhodl pro mimopásmovou komunikaci pro signalizaci režimu komunikace, a to vývodem D/$\overline{\text{C}}$ (Data/~Command). Logický stav na tomto vývodu při čtení posledního odeslaného bitu (LSB) z každého bajtu přijatého na sběrnici odpovídá, zda má být bajt interpretovaný jako data (1) nebo příkaz (0). Pro tento účel byla poupravena standardní knihovna pro SPI.

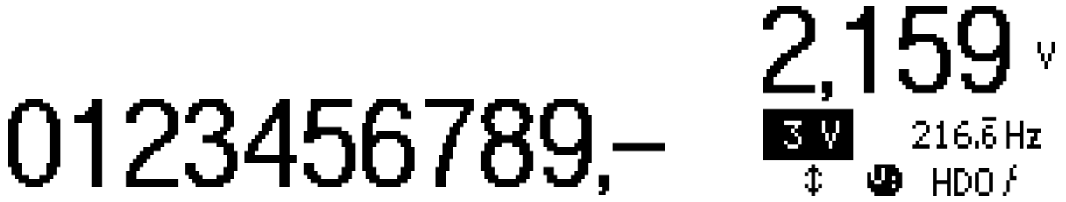
Odeslání bajtu dat i příkazu trvá stejnou dobu; data zapisují hodnoty do postupných adres RAM řadiče, zatímco příkaz může vybrat konkrétní řádek či sloupec pro zápis, změnit napětí nábojové pumpy či invertovat displej. Nejvyšší příkaz nastavuje základní registry displeje a přepíná mezi základní a rozšířenou sadou příkazů; rozšířená sada se uplatňuje pouze při inicializaci pro nastavení kontrastu displeje, po zbytek běhu programu je aktivní základní sada, aby nastavení řádku/sloupce vyžadovalo odeslání pouhých 2 bajtů.

Bajty v RAM displeje odpovídají svislým obdélníkům 1×8, kde spodní pixel odpovídá nejvyššímu bitu. Skupiny 84 bajtů pak tvoří 6 vodorovných pruhů 84×8 zleva doprava, shora dolů. Situace je ilustrována v příloze této práce. Chceme-li se vyhnout nutnosti implementovat framebuffer v RAM MCU a bitových operací pro každý zápis textu, je nutné polohu údajů na displeji a výšku písma navrhnout, aby odpovídala hranicím pruhů.

Žádné z dostupných softwarových řešení pro komunikaci s displejem neodpovídalo požadavkům: minimální využití RAM procesoru (tedy absence framebufferu), nezávislost na frameworku Arduino a C++, byl proto vytvořen jednoduchý systém funkcí pro vykreslování údajů na displej nikoli operacemi s jednotlivými pixely, ovšem ve výše zmíněných pruzích.

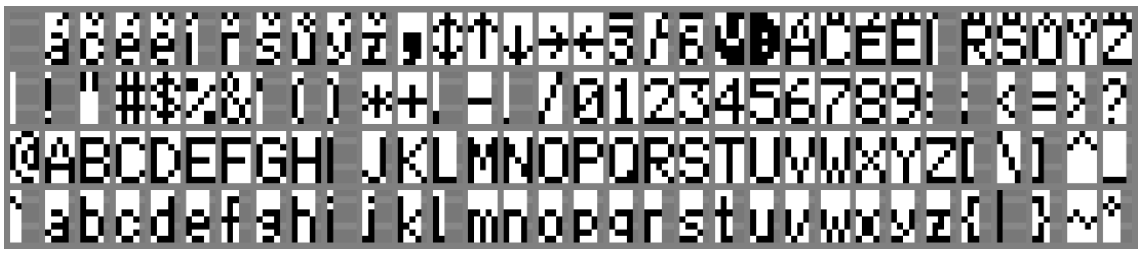
Pro nejdůležitější údaj na displeji – naměřené napětí – byly vyhrazeny první tři pruhy s výškou písma 22 bodů; pro nastavený rozsah zobrazení a frekvenci HDO pak další 2 pruhy s výškou písma 7 bodů uvnitř případného rámečku o výšce 13 bodů; pro označení veličin pak poslední pruh s výškou písma 7 bodů.

Číslice 15×22 bodů jsou ručně kreslené a jsou inspirované rodinou Helvetica. Funkce pro kreslení textu je navržena tak, aby přebírala tentýž řetězec, jako periferie UART: vykreslují se postupně číslice řetězce, při desetinné tečce se „kurzor“ posouvá o její šířku (není ji třeba překreslovat), v případě pomlčky (neplatného výsledku) je vykreslen namísto číslice vodorovný pruh.



Obrázek 28 – Písmo pro zobrazení výsledku; snímek uživatelského rozhraní

Písmo 7 px bylo založeno na vzhledu znaků u klasických znakových displejů, jako například původně zamýšlený LCD s řadičem vycházejícím z klasického HD44780. První verze softwaru rovněž používala konstantní šířku 5 sloupců na znak, avšak pro dosažení většího počtu znaků na řádek (např. pro text „SelektivniVoltmetr_“) bylo rozhodnuto využít proporcionální písmo. Vzhledem k tomu, že velká část znaků (velká písmena, číslice) využívá celou šířku 5 pixelů a adresování v poli 511 použitých bajtů by zabíralo více než 1 bajt na každý ze 128 znaků, bylo rozhodnuto omezit šířku znaků na nejvýše 5 pixelů a konec grafiky užšího znaku signalizovat doplněním bajty 0xdd, jež se v písmu nevyskytují ani s tolerancí překlopení jednoho bitu při případných grafických úpravách. Tímto bylo zjednodušeno zpracování grafiky písma, jež bylo překresleno v jednom bitmapovém obrázku z původně zamýšleného neproporcionálního písma pomocí grafického editoru, aniž by bylo nutné ošetřit adresování jednotlivých znaků. Nástrojem image2cpp [35] pak byl obrázek převeden na 640 bajtů kódujících jednotlivé 8pixelové pruhy grafiky po sloupcích, odpovídaje pořadí vyžadovanému displejem.



Obrázek 29 – Znaková mapa pro grafický displej
Bajty odpovídající konci znaku (`0xdd`) jsou znázorněny nízkým kontrastem;
pro přehlednost byly přidány šedé okraje buněk.

Kromě defaultní mezery (ASCII znak 0x20) o šířce 1 pixelu (plus 1px okraj znaku zleva i zprava) se ukázalo vhodné implementovat mezery nastavitelné šířky pro snadné kreslení pruhů nebo zarovnání předdefinovaných číselných hodnot doprava. Pro tento účel byly vyhrazeny bajty 0xF0 (mezera šířky 0 px, neviditelný znak) až 0xFF (mezera šířky 15 pixelů). Kontrola, že 4 horní bity mají hodnotu 1, je řešena bitovými operacemi tak, aby postačovala jediná konstanta (bitová maska) určující nejvyšší šířku mezery, nastavitelná direktivou `#define FONT_7PT_SPECIFIC_SPACES 0x0F` na libovolnou hodnotu odpovídající $2^n - 1; n \in \mathbb{N}; n < 8$. Aby nebyla nutná další proměnná pro šířku, je namísto inkrementovaného počitadla sloupců použito počitadlo zbývajících sloupců (*remaining column count*, `rcc`) s defaultní hodnotou 6, případně přenastavenou na šířku mezery a dekrementovanou při vykreslení každého sloupce.

Kreslení písma pouze v pruzích vymezených řádky se ukázalo jako příliš omezující, proto byl funkci `draw7ptString()` přidán parametr `shift` ve formátu bajtu se znaménkem (`int8_t`), jenž nastavuje posun textu v řádku o konkrétní počet pixelů dolů, záporné číslo pak značí posunutí nahoru. Zavoláním funkce dvakrát, nejprve pro horní řádek s posunem n, pak pro dolní řádek s posunem $n - 8$, lze tedy dosáhnout umístění textu na libovolných souřadnicích. Stále ale platí omezení vycházející z absence snímkového bufferu – je přepsáno všech 16 řádků v dotčených pruzích. Pro rozšíření možností byl přidán ještě jeden bajtový (`uint8_t`) parametr `xormask`. Jak název napovídá, označuje masku, jež je aplikována operací XOR ke každému odeslanému bajtu pro selektivní inverzi řádků, a umožňuje tak kreslit vodorovný pruh pro efekt inverze, rámečku nebo podtržení. Pro vybranou položku v menu byl zvolen efekt inverze s rámečkem sahajícím 3 px nad i pod znaky běžné výšky 7 px.

Vzhledem k tomu, že odesílání grafických dat displeji je synchronní a omezeno rychlostí periferie SPI na 4 Mb/s, tedy nejméně 32 cykly procesoru na bajt, nemá tento sled jednoduchých bitových operací ve většině případů žádný dopad na rychlost vykreslování textu na displej (nejdelší trvání má logický bitový posun: 1 cyklus za každý bit posunu). Nejpomalejší část procesu vykreslování, komunikaci s displejem, lze optimalizovat omezením opakovaného odesílání bajtů, tedy například namísto nakreslení rámečku zadané délky a následného doplnění textu je vhodné nakreslit levý okraj, text a následně pravý okraj. Tato možnost byla rovněž přidána do funkce `draw7ptString()`: parametr `leftpad` určuje počet prázdných sloupců vlevo, `boxwidth` pak minimální počet celkových sloupců na vykreslení (při příliš krátkém textu se doplní tiskem `xormask`).

2.3.4 Uživatelské rozhraní

Počítá se s nastavením parametrů zařízení pomocí grafického menu. Aktuálně se jedná pouze o rozsah a frekvenci HDO, přesto je program připraven pro až 255 položek menu. Jednotlivé položky jsou implementovány jako statická datová struktura obsahující následující parametry:

- název veličiny pro zobrazení legendy ve spodním řádku i pro výstup UART,
- název jednotky pro zobrazení vpravo v obdélníku i pro výstup UART,
- pole hodnot pro zobrazení vlevo v obdélníku i pro výstup UART,
- počet hodnot,
- pointer do RAM na index nastavené hodnoty (nutný u statické struktury).

Mimoto je pro každou položku menu uložena sada rozměrů, které určují, kde má být nakreslena na displeji. Všechny se nacházejí na témže řádku, jde tedy vždy o vodorovnou vzdálenost v px:

- levý okraj obdélníku,
- pozice textu hodnoty,
- pozice textu jednotky,

- šířka obdélníku,
- pozice textu veličiny.

Menu se rovněž dělí na stránky (v aktuální implementaci je využívána pouze jedna). Pole jednotlivých stránek `menuPages[]` obsahuje index první položky na každé stránce a nakonec i celkový počet položek (`NUMBER_OF_SETTINGS`), jelikož se při vykreslení *n*-té stránky menu iteruje mezi položkami nastavení `menuPages[n]` až `menuPages[n+1]-1`.

Pro podporu veličin řešených jako aritmetická posloupnost i jako posloupnost konkrétních hodnot bylo rozhodnuto v obsluze nastavení příslušných proměnných `applySettings()` řešit každou veličinu zvlášť. Nastavení tak rovněž může způsobit například nulování mezivýsledků. Pravděpodobně by bylo možné ušetřit paměť Flash a zjednodušit přidávání položek použitím pole pointerů na jednotlivé funkce namísto struktury `switch – case`, jednalo by se však o kód s větší náchylností na paměťové chyby. Ze stejného důvodu bylo opuštěno od plánu řešit položky menu jako instance struktury menuItem.

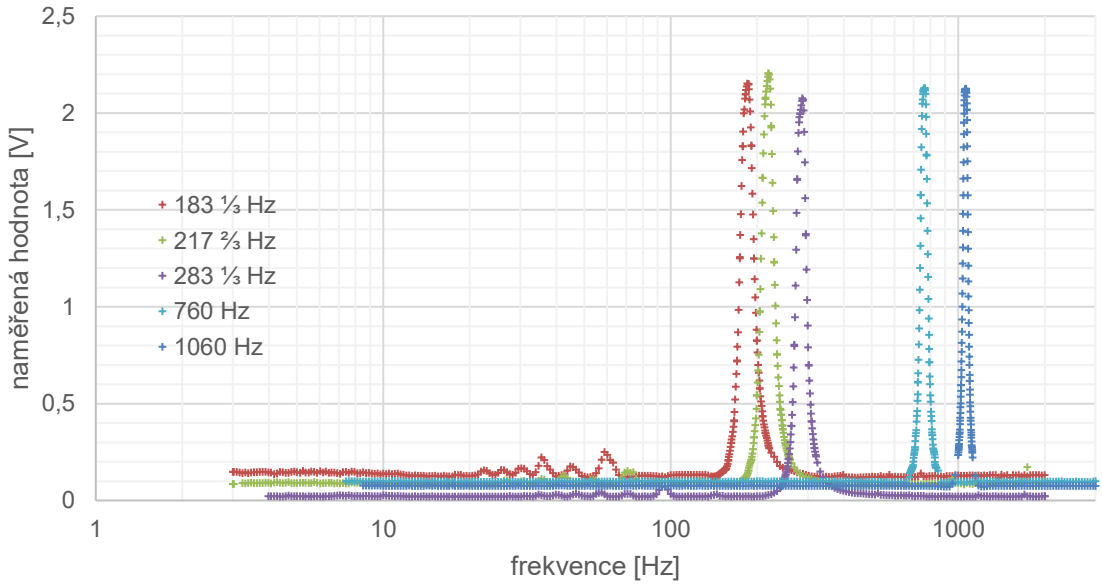
Důvodem rozdělení textových položek na verze pro LCD a UART jsou rozdílné možnosti obou rozhraní: definice vlastního písma umožňuje zobrazit zvláštní znaky namísto netisknutelných ovládacích znaků ASCII 0~31 a 127 (např. symbol periodického desetinného čísla $\overline{3}$, kurzivní symbol frekvence f), naopak podpora ovládacích znaků ASCII v rozhraní UART umožňuje použít například tabulátor pro zarovnání hodnot různé délky.

2.3.5 Výsledky

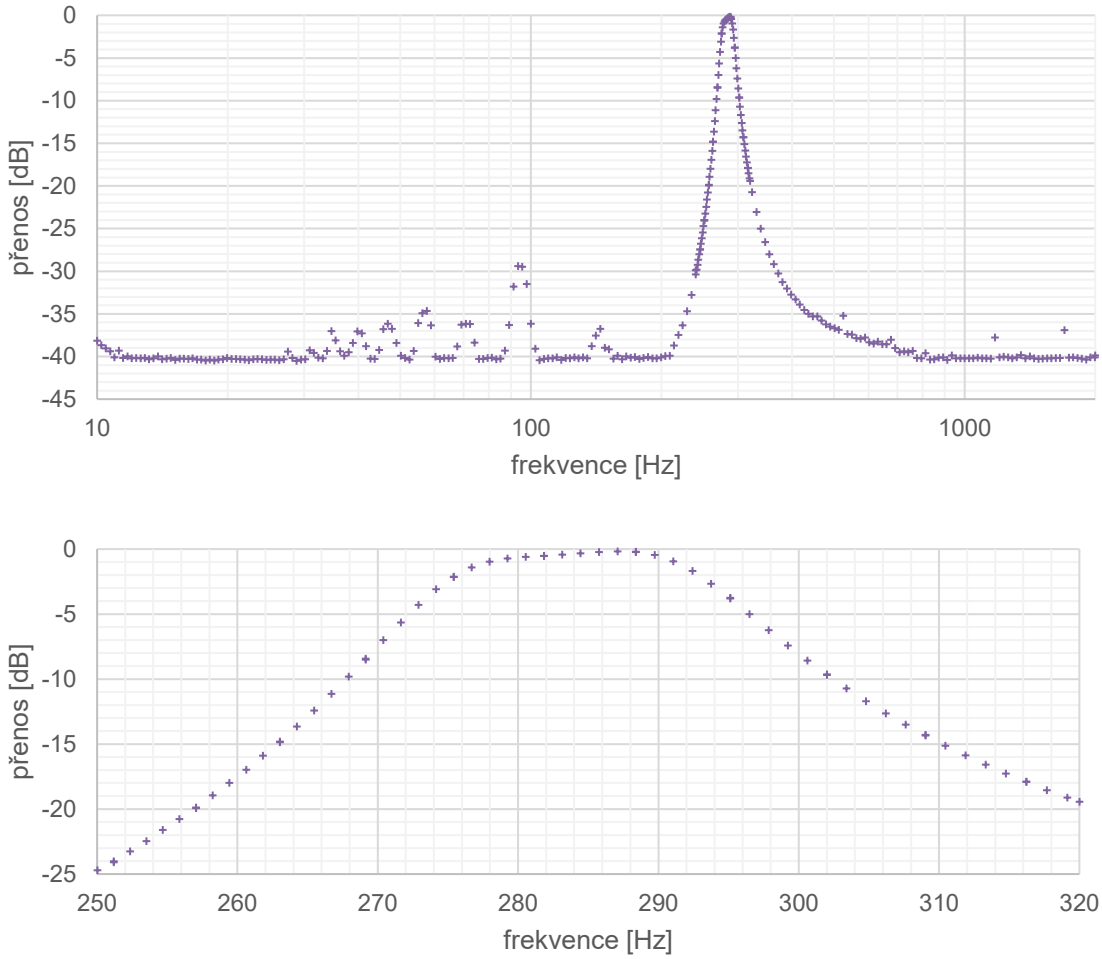
Testování digitálního vstupu probíhalo pomocí generátoru funkcí, kdy byl jeho signál přiveden na analogový vstup MCU a data byla sbírána na počítači pomocí softwarového sériového loggeru. Pro řízení generátoru funkcí byl vytvořen skript v Pythonu, jenž v krocích měnil frekvenci sinusoidy o amplitudě 3 V podle geometrické řady, tedy v konstantních krocích na logaritmické stupnici.

Je patrné, že z důvodu aliasingu dochází k mírnému přenosu i na frekvencích, jež jsou celočíselným podílem nominálního kmitočtu filtru. Přenos v nepropustném pásmu je ovšem i tak velmi nízký, nikdy nedosahuje nad –28 dB, a je na většině pásma srovnatelný s šumem. Nevýhodou, jež brání využití tohoto filtru v přesném voltmetru s více než dvěma platnými číslicemi, je kvantováním koeficientů způsobená neplochost frekvenční charakteristiky v propustném pásmu: bod nejvyššího přenosu (287 Hz) má přenos o 0,26 dB vyšší, než má nominální frekvence filtru, což způsobuje odchylku o 3 %.

Výslednou kmitočtovou charakteristiku voltmetru znázorňuje následující graf:



Obrázek 30 – Kmitočtová charakteristika voltmetru pro signál amplitudy 3V



Obrázek 31 – Kmitočtová charakteristika filtru pro frekvenci 283⅓ Hz; detail propustného pásma

Závěr

Výroba selektivního voltmetru byla zajímavým cvičením v oblasti aplikace číslicového zpracování signálu, návrhu a realizace desky plošných spojů. Během všech těchto kroků nastaly potíže srovnatelné s problémy, s nimiž se běžně potýkají elektrotechnici, a jejich řešení tedy pomohlo autorovi získat předpoklady pro budoucí činnost v tomto oboru.

Jedním z výrazných omezení byla volba mikrokontroléru. Zdrojový kód musel projít opakovanou optimalizací, aby bylo na osmibitovém procesoru na 16 MHz dosaženo výpočtu DSP v reálném čase. Odložení testování zařízení navíc způsobilo, že byla nedostatečná přesnost A/D převodníku pro zpracování síťového napětí odhalena až na poslední chvíli, nebylo tedy možné do práce zahrnout například návrh zmíněného analogového vstupního filtru 3. řádu, jenž by tento nedostatek řešil. Také nebyla provedena důkladná kalibrace zařízení, během které by bylo možné dosáhnout podobně vysokého útlumu, jako u filtru pro 283 $\frac{1}{3}$ Hz, i u ostatních frekvencí HDO. Ačkoli má tento filtr uspokojivé vlastnosti, pro budoucí návrh číslicových filtrů autor doporučuje volit platformu s dostatečnou rezervou výpočetního výkonu a schopností hardwarových periferií.

Zvolená omezení při návrhu desky plošných spojů umožnila navrhnout zařízení s několika možnými způsoby využití a přívětivostí k začátečníkům, ovšem výrazně zvýšila počet iterací a prodloužila tak vývoj zařízení. Následně nebylo možné zařízení před objednávkou desek plošných spojů řádně otestovat, došlo tedy k nutnosti několika ručních oprav.

Podobný případ, kdy bylo dbáno na nepříliš důležité detaily, spíše než na včasné dokončení a možnost dlouhodobějšího testování, byl firmware zařízení. Namísto vyladění přenosu jednotlivých filtrů, aby bylo dosaženo co nejlepších parametrů filtru při omezeném výkonu procesoru, byl čas věnován vývoji vzhled uživatelského rozhraní. Klíčové části práce tak byly dokončovány narychlo a nebyly vhodně otestované.

Literatura

[1] Tuned Radio Frequency Receiver: TRF. In: *Electronics Notes* [online]. [cit. 2024-05-20]. Dostupné z: https://www.electronics-notes.com/articles/radio/radio-receivers/tuned-radio-frequency-trf-basics.php

[2] PROKEŠ, prof. Ing. Aleš, Ph.D. Koncepce přijímačů a vysílačů: Kurz operátorů. In: *Radioklub OK2KOJ při VUT v Brně* [online]. 2016 [cit. 2024-05-20]. Dostupné z: https://www.radio.feec.vutbr.cz/ok2koj/kurz/05_prijimace_vysilace.pdf

[3] BREED, Gary. The Mathematics of Mixers: Basic Principles. *High Frequency Electronics* [online]. 2011, **7**(1), 34-36 [cit. 2024-05-18]. Dostupné z: https://www.highfrequencyelectronics.com/Jan11/HFE0111_Tutorial.pdf

[4] SCHOTTKY, Walter H. On the Origin of the Super-Heterodyne Method. *Proceedings of the I.R.E*. 1926, **14**(10), 695-698.

[5] TLV320AIC3204: Ultra Low Power Stereo Audio Codec. In: *Texas Instruments* [online]. 2008-09, 2019-09 [cit. 2024-05-18]. Dostupné z: https://www.ti.com/lit/ds/symlink/tlv320aic3204.pdf

[6] NXP 32-bit Digital Signal Controllers (DSC). In: *NXP* [online]. 2015, 2022 [cit. 2024-05-18]. Dostupné z: https://www.nxp.com/docs/en/supporting-information/DSCMICROLNCD.pdf

[7] MATOUŠEK, David a Bohumil BRTNÍK. *Aplikace algoritmů číslicového zpracování signálů 1. díl*. Praha: BEN - technická literatura, 2014. ISBN 978-80-7300-478-1.

[8] AKILAA. Flowchart of a digital biquad filter, direct form I. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2010, 2010-04-11, 2011-10-17 [cit. 2024-05-20]. Dostupné z: https://commons.wikimedia.org/wiki/File:Biquad_filter_DF-I.svg

[9] AKILAA. Flowchart of a digital biquad filter, direct form II. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2010, 2010-04-11, 2011-10-17 [cit. 2024-05-20]. Dostupné z: https://commons.wikimedia.org/wiki/File:Biquad_filter_DF-IIx.svg

[10] MAUTNER, Ing. Pavel, PhD. Analýza a zpracování signálů: 5. Z-transformace. In: *Katedra informatiky a výpočetní techniky ZČU* [online]. [cit. 2024-05-18]. Dostupné z: https://www.kiv.zcu.cz/~mautner/Azs/Azs6_Z-transformace.pdf

[11] SARPAL, Dr. Sanjeev. Difference between IIR and FIR filters: a practical design guide. In: *Advanced Solutions Nederland* [online]. 2020, 2020-04-28 [cit. 2024-05-18]. Dostupné z: https://www.advsolned.com/difference-between-iir-and-fir-filters-a-practical-design-guide/

[12] BRTNÍK, Bohumil a David MATOUŠEK. *Algoritmy číslicového zpracování signálů: řešené příklady*. Praha: BEN - technická literatura, 2011. ISBN 978-80-7300-400-2.

[13] MILLMAN, Jacob a Christos C. HALKIAS. Normalized Butterworth polynomials. In: *Integrated electronics: Analog Digital Circuits and Systems*. McGraw-Hill, 1972, s. 550. ISBN 9780070423176.

[14] DE BRUYNE, Franky. Mapping the s-plane into the z-plane. In: *Digital control theory: video 5 From s domain to z domain (part 1)* [online]. Belgie: YouTube/@frankydebruyne9842, 2020, 2020-10-19, 13:03 [cit. 2024-05-17]. Dostupné z: https://www.youtube.com/watch?v=0oHTxcflcSA&t=13m3s

[15] SELESNICK, I. W. a C. S. BURRUS. Generalized digital Butterworth filter design. *IEEE Transactions on Signal Processing* [online]. 1998, **46**(6), 1688-1694 [cit. 2024-05-18]. ISSN 1053587X. Dostupné z: doi:10.1109/78.678493

[16] Cheby1: Chebyshev Type I filter design. In: MATHWORKS. *MathWorks Help Center* [online]. 2024 [cit. 2024-05-18]. Dostupné z: https://www.mathworks.com/help/signal/ref/cheby1.html

[17] Cheby2: Chebyshev Type II filter design. In: MATHWORKS. *MathWorks Help Center* [online]. 2024 [cit. 2024-05-18]. Dostupné z: https://www.mathworks.com/help/signal/ref/cheby2.html

[18] SPERLING, Christian. Who is disrupting the utility frequency? Grid frequency dip of 10 January 2019: Causes and Theories. In: *Next Kraftwerke* [online]. 2019, 2019-02-15 [cit. 2024-05-18]. Dostupné z: https://www.next-kraftwerke.com/energy-blog/who-is-disrupting-the-utility-frequency

[19] IBLER, Prof. Ing. Zbyněk, DrSc. a Doc. Ing. Miloš BERAN. Napěťové statické charakteristiky. In: *Elektrárny II.* [online]. Plzeň: Ediční středisko VŠSE, 1982, s. 245-247 [cit. 2024-05-21]. Dostupné z: http://home.zcu.cz/~nohac/E2/Skripta_Elektrarny_II.PDF#page=255

[20] MÁSLO, Karel. *Řízení a stabilita elektrizační soustavy*. [Praha]: Asociace energetických manažerů, 2013. ISBN 978-80-260-4461-1.

[21] VOGL, Tomáš. *Budoucnost Smart Grid a hromadného dálkového ovládání* [online]. Plzeň, 2015 [cit. 2024-05-18]. Dostupné z: https://otik.zcu.cz/bitstream/11025/32188/1/Bakalarska%20prace%20-%20Budoucnost%20Smart%20Grid%20a%20hromadneho%20dalkoveho%20ovladani.pdf. Autoreferát disertační práce. Fakulta elektrotechnická ZČU v Plzni. Vedoucí práce Ing. Lenka Raková, Ph.D.

[22] Energetická účinnost v českých zemích za posledních 100 let. In: *Ministerstvo průmyslu a obchodu* [online]. 2019 [cit. 2024-05-19]. Dostupné z: https://www.mpo.gov.cz/assets/cz/energetika/energeticka-ucinnost/2019/12/100_let_energeticke_ucinnosti_v_CR_publikace.pdf

[23] HDO spínač - uživatelský manuál. In: IQTRONIC TECHNOLOGIES EUROPE S.R.O. *Mikrovlny s.r.o.* [online]. 2006 [cit. 2024-05-18]. Dostupné z: http://www.mikrovlny.cz/content/file/manualy/hdomanual.pdf. Viz též: http://www.mikrovlny.cz/cz/produkt/15

[24] POHORSKÝ, Jiří. *HDO - hromadné dálkové ovládání* [PDF]. Praha: BEN - technická literatura, 2002 [cit. 2024-05-17]. ISBN 978-80-7300-347-0. Dostupné z: http://shop.ben.cz/121097

[25] Přehled distribučních sazeb elektřiny. In: *Skupina ČEZ* [online]. 2024 [cit. 2024-05-19]. Dostupné z: https://www.cez.cz/cs/podpora/prehled-distribucnich-sazeb-elektriny

[26] SÝKORA, Ing. Tomáš, PhD. Komunikace po silových vedeních: Signál hromadného dálkového ovládání. In: *ČVUT FEL Courseware Wiki* [online]. 2011 [cit. 2024-05-19]. Dostupné z: https://cw.fel.cvut.cz/old/_media/courses/b1m15dee/prednasky/prednaska_06b.pdf. Viz též https://cw.fel.cvut.cz/old/courses/b1m15dee/prednasky/start

[27] POHORSKÝ, Jiří. *HDO - hromadné dálkové ovládání* [PDF]. Praha: BEN - technická literatura, 2002 [cit. 2024-05-17]. ISBN 978-80-7300-347-0. Dostupné z: http://shop.ben.cz/121097

[28] Smart Thermostats, Automation, and Time-Varying Prices: Working Paper 21-20. In: *Resources for the Future* [online]. 2021-06 [cit. 2024-05-19]. Dostupné z: https://media.rff.org/documents/WP_21-20_w28Jfrz.pdf

[29] MLÝNEK, Petr, Jiří MIŠUREC, Pavel ŠILHAVÝ, Radek FUJDIAK, Ján SLÁČIK a Zeynep HASIRCI. Simulation of Achievable Data Rates of Broadband Power Line Communication for Smart Metering. *Applied Sciences* [online]. 2019, **9**(8), 1-22 [cit. 2024-05-20]. ISSN 2076-3417. Dostupné z: doi:10.3390/app9081527

[30] SÝKORA, Ing. Tomáš, PhD. Komunikace po silových vedeních: Signál hromadného dálkového ovládání. In: *ČVUT FEL Courseware Wiki* [online]. 2011 [cit. 2024-05-19]. Dostupné z: https://cw.fel.cvut.cz/old/_media/courses/b1m15dee/prednasky/prednaska_06b.pdf. Viz též https://cw.fel.cvut.cz/old/courses/b1m15dee/prednasky/start

[31] MATOUŠEK, David. *Aplikace mikrokontrolérů ATmega644*. Praha: BEN - technická literatura, 2013. ISBN 978-80-7300-492-7.

[32] SÍDEK, Bc. Vojtěch. *Univerzální přijímač hromadného dálkového ovládání* [online]. Praha, 2012 [cit. 2024-05-18]. Dostupné z: https://wiki.control.fel.cvut.cz/mediawiki/images/8/87/Dp_2012_sidek_vojtech.pdf. Diplomová. České vysoké učení technické v Praze, Fakulta elektrotechnická. Vedoucí práce Ing. Pavel Troller, CSc.

[33] LCD L303646-g7A145: Electronics Projects. In: *Electronics Documentation Wiki* [online]. 2019, 2019-11-27 [cit. 2024-05-24]. Dostupné z: http://www.projects.scorchingbay.nz/dokuwiki/electronic/lcd/g7a145

[34] ATmega48/V / 88/V / 168/V: Complete Datasheet. In: *Microchip Technology* [online]. 2016 [cit. 2024-05-24]. Dostupné z: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2545-8-bit-AVR-Microcontroller-ATmega48-88-168_Datasheet.pdf

[35] VAN LOENEN. *Image2cpp* [online]. 2016, 2024-02-23 [cit. 2024-08-22]. Dostupné z: https://javl.github.io/image2cpp/

[36] Butterworth filter. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 2024-05-09 [cit. 2024-05-17]. Dostupné z: https://en.wikipedia.org/wiki/Butterworth_filter#Normalized_Butterworth_polynomials

[37] Comparison of Analog IIR Lowpass Filters. In: MATHWORKS. *MathWorks Help Center* [online]. 2024 [cit. 2024-05-18]. Dostupné z: https://www.mathworks.com/help/signal/ug/comparison-of-analog-iir-lowpass-filters.html

[38] LI, Yinghui, Jianan CHEN, Xiaohuan WANG, Xudong ZHANG a Xiaojun ZHAO. Dynamic Stability Study of Grid-Connected Inverter Based on Virtual Synchronizer under Weak Grid. *Energies* [online]. 2022, **15**(19), 3 [cit. 2024-05-18]. ISSN 1996-1073. Dostupné z: doi:10.3390/en15197091

[39] KRUTINA, Ing. Aleš. *Optimalizace a bilance řízení toků energie* [online]. Plzeň, 2015 [cit. 2024-05-18]. Dostupné z: https://dspace5.zcu.cz/bitstream/11025/23031/1/TEXT%20Disertace.pdf. Autoreferát disertační práce. Fakulta elektrotechnická ZČU v Plzni. Vedoucí práce Prof. Ing. Zdeněk Vostracký, DrSc., dr.h.c.

Příloha A – Zdrojový kód

./src/lib/spi.h

```
1  // edited standard library code by DavidDiPaola (CC0)
2  // https://github.com/DavidDiPaola/avr_libs/blob/master/spi/spi.h
3  #ifndef __SPI_H
4  #define __SPI_H
5
6  #include <inttypes.h>
7
8  const uint8_t __SPI_MOSI = 3;
9  const uint8_t __SPI_MISO = 9;
10 const uint8_t __SPI_SCK = 5;
11 const uint8_t __SPI_SS = 2;
12 const uint8_t __SPI_LSBFIRST_MASK = 0b00000001;
13 const uint8_t __SPI_MASTER_MASK = 0b00000001;
14 const uint8_t __SPI_MODE_MASK = 0b00000011;
15 const uint8_t __SPI_SPEED_MASK = 0b00000011;
16 const uint8_t __SPI_DBLCLK_MASK = 0b00000001;
17
18 #endif
```

./src/lib/spi.c

```
1  // edited standard library code by DavidDiPaola (CC0)
2  // https://github.com/DavidDiPaola/avr_libs/blob/master/spi/spi.c
3  #include <inttypes.h>
4  #include <avr/io.h>
5  #include "spi.h"
6  #ifndef __SPI_C
7  #define __SPI_C
8
9  #define __SPI_PORT PORTB
10 #define __SPI_DDR DDRB
11 #define __SPI_PIN PB0
12
13 // makes for more readable init code
14 #define SPI_MSBFIRST 0
15 #define SPI_LSBFIRST 1
16 #define SPI_SLAVE 0
17 #define SPI_MASTER 1
18 #define SPI_CLK_SETFALL_READRISE_IDLELOW  0b00
19 #define SPI_CLK_SETRISE_READFALL_IDLELOW  0b01
20 #define SPI_CLK_SETRISE_READFALL_IDLEHIGH 0b10
21 #define SPI_CLK_SETFALL_READRISE_IDLEHIGH 0b11
22 // simplify user selection of speeds to single register
23 //    CLKDIV REGISTERS → 01 /↓ DOUBLE SPEED REGISTER
24 #define SPI_CLKDIV_2   0b00, 1
25 #define SPI_CLKDIV_4   0b00, 0
26 #define SPI_CLKDIV_8   0b01, 1
27 #define SPI_CLKDIV_16  0b01, 0
28 #define SPI_CLKDIV_32  0b10, 1
29 #define SPI_CLKDIV_64  0b10, 0
30 //for same result, one can use #define SPI_CLKDIV_64  0b11, 1
31 #define SPI_CLKDIV_128 0b11, 0
32
33 //initialize the SPI bus
34 //  uint8_t lsbfirst - if 0: most significant bit is transmitted first
35 //  uint8_t master - if 1: use master mode, if 0: slave mode is used
36 //  uint8_t mode - sets the transfer mode:
37 //                 mode   leading clock edge   trailing clock edge
38 //                 ------------------------------------------------
39 //                 0      sample (rising)      setup  (falling)
40 //                 1      setup  (rising)      sample (falling)
41 //                 2      sample (falling)     setup  (rising)
42 //                 3      setup  (falling)     sample (rising)
43 //  uint8_t clkrate - spi bus clock rate, valid speeds are 0-3
44 //                    rate   speed
45 //                    ------------
46 //                    0      CPUCLK/4
47 //                    1      CPUCLK/16
48 //                    2      CPUCLK/64
49 //                    3      CPUCLK/128
50 //  uint8_t dblclk - if 1: doubles the SPI clock rate in master mode
51 //  EXAMPLE: spi_init(0, 1, 0, 3, 0)
52 void spi_init(uint8_t lsbfirst,
53               uint8_t master,
54               uint8_t mode,
55               uint8_t clkrate,
```

```
56                uint8_t dblclk){
57    //set outputs
58    __SPI_DDR |= ((1<<__SPI_MOSI) | (1<<__SPI_SCK));
59    //set inputs
60    if (__SPI_MISO < 9) { //miso enabled
61      __SPI_DDR &= ~(1<<__SPI_MISO);
62      __SPI_PORT |= (1<<__SPI_MISO); //turn on pull-up resistor
63    }
64    //set SPI control register
65    SPCR = (
66              (1<<SPE) | //enable SPI
67              ((lsbfirst & __SPI_LSBFIRST_MASK)<<DORD) | //set msb/lsb ordering
68              ((master & __SPI_MASTER_MASK)<<MSTR) | //set master/slave mode
69              ((mode & __SPI_MODE_MASK)<<CPHA) | //set mode
70              (clkrate & __SPI_SPEED_MASK<<SPR0) //set speed
71            );
72    //set double speed bit
73    SPSR = ((dblclk&__SPI_DBLCLK_MASK)<<SPI2X);
74  }
75
76  //shifts out 8 bits of data
77  void spi_send(uint8_t value){
78    while(!(SPSR & (1<<SPIF)));
79    SPDR = value;
80  }
81
82  void spi_sendbyteseq(uint8_t *s, uint8_t len){
83    uint8_t offset = 0;
84    while (offset < len) {
85      spi_send(s[offset]);
86      offset++;
87    }
88  }
89  #endif
```

./src/lib/serial.h

```
1  #ifndef SERIAL_H
2  #define SERIAL_H
3
4  #include "serial.c"
5  //#define USART_BAUDRATE 9600
6  //#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16))) - 1)
7
8  void serial_init();
9  void serial_dispatch();
10 void serial_send(uint8_t data);
11 void showLED(uint8_t num);
12 void serial_transmit(uint8_t data);
13 //uint8_t serial_recv();
14
15 #endif /* __SERIAL_H */
```

./src/lib/serial.c

```
1  #ifndef __SERIAL_C
2  #define __SERIAL_C
3  #include <avr/common.h>
4  #include <avr/io.h>
5  #include "common.h"
6  #include "serial.h"
7  #include <avr/interrupt.h>
8  #include "leds.h"
9
10 #define SERIAL_BUFFER_SIZE 32 //must be power of 2, see next line
11 #define SERIAL_BUFFER_SIZE_MASK ((uint8_t)SERIAL_BUFFER_SIZE - 1) //returns mask (0x00011111) for
easy modulo operations
12 char serial_buffer[SERIAL_BUFFER_SIZE];
13 uint8_t serial_buffer_head_index = 0; // last buffer index written to
14 uint8_t serial_buffer_tail_index = SERIAL_BUFFER_SIZE_MASK; // last buffer index read from
15 uint8_t sendbusy = 0;
16
17 void serial_init() {
18   UCSR0B = (0 << RXEN0) | (1 << TXEN0)  //UART transmit; receive not enabled, RX pin is free
19                         | (1 << UDRIE0); //enable TXC0 interrupt used for serial_dispatch()
20   // UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00); //not required, bits are 1 by default
21   UBRR0H = 0; UBRR0L = 103;  // 9600 Bd @ 16 MHz
22   sei(); //interrupts globally enabled to ensure buffer flush
23 }
24
25 void serial_transmit(uint8_t data) { //blocking (synchronous) send function
```

```
  26    while ((UCSR0A & (1 << UDRE0)) == 0) {}; //loop until SERIAL READY bit = 1
  27    UDR0 = data;
  28  }
  29
  30  void serial_dispatch(){ //push next byte in buffer to serial peripheral; called when byte
transmission is complete
  31    UCSR0B = (0 << RXEN0) | (1 << TXEN0)  //UART transmit; receive not enabled, RX pin is free
  32                          | (1 << UDRIE0); //enable TXC0 interrupt used for serial_dispatch()
  33    if ((UCSR0A & (1 << UDRE0)) == 0) return; //nothing to do, serial is busy
  34    UCSR0B = (0 << RXEN0) | (1 << TXEN0)  //UART transmit; receive not enabled, RX pin is free
  35                          | (0 << UDRIE0); //enable TXC0 interrupt used for serial_dispatch()
  36    if /*(!fifo_usage)*/(serial_buffer_head_index == serial_buffer_tail_index) {
  37    UCSR0B = (0 << RXEN0) | (1 << TXEN0)  //UART transmit; receive not enabled, RX pin is free
  38                          | (0 << UDRIE0); //NOT enable TXC0 interrupt used for serial_dispatch()
  39    return;} //nothing to do, buffer is empty, don't wake me up again
  40    uint8_t byte_to_send = serial_buffer[serial_buffer_tail_index];
  41    if(byte_to_send) { //not null-terminator
  42      UDR0 = serial_buffer[serial_buffer_tail_index];
  43    }
  44    serial_buffer[serial_buffer_tail_index] = 0;
  45    serial_buffer_tail_index = (serial_buffer_tail_index + 1) & SERIAL_BUFFER_SIZE_MASK;
  46    UCSR0B = (0 << RXEN0) | (1 << TXEN0)  //UART transmit; receive not enabled, RX pin is free
  47                          | (1 << UDRIE0); //enable TXC0 interrupt used for serial_dispatch()
  48  }
  49
  50  void serial_buffer_push(uint8_t data) { //add data to serial buffer
  51    while (serial_buffer[(serial_buffer_head_index + 1) & SERIAL_BUFFER_SIZE_MASK]) //as long as
buffer is full (head just behind tail)...
  52      { cli(); serial_dispatch(); sei(); _delay_loop_1(200);} //... block main thread trying to TX
  53    cli();
  54    serial_buffer[serial_buffer_head_index] = data;
  55    serial_buffer_head_index = (serial_buffer_head_index + 1) & SERIAL_BUFFER_SIZE_MASK;
  56    sei();
  57    serial_dispatch(); //attempt to transmit; necessary to start from empty buffer
  58  }
  59
  60  ISR(USART_UDRE_vect) { //when byte transmission is complete
  61    serial_dispatch(); //send next byte from buffer, if any
  62  }
  63
  64  void printchar(char myChar) {
  65    if (myChar != '\0') serial_buffer_push(myChar);
  66  }
  67
  68  void printstring(char *s){
  69    uint8_t offset = 0;
  70    char thisChar = 0;
  71    do {
  72      thisChar = s[offset];
  73      printchar(thisChar);
  74      offset++;
  75    } while (thisChar != '\0');
  76  }
  77
  78  #endif
```

./src/lib/PCD8544_HWSPI_C.h

```
   1  #ifndef __PCD8544_HWSPI_H
   2  #define __PCD8544_HWSPI_H
   3  #include <avr/io.h>
   4  #include <avr/interrupt.h>
   5  #include "spi.c"
   6  #include <util/delay.h>
   7  #include "serial.h"
   8  #include <avr/pgmspace.h>
   9
  10  // DECLARATIONS
  11    void lcd_sendCommand(uint8_t mybyte);
  12    void lcd_sendData(uint8_t mybyte);
  13    void lcd_putCursor(uint8_t row, uint8_t col);
  14    void lcd_splashscreen();
  15
  16    const uint8_t font22pt[450] PROGMEM;
  17    const uint8_t font7pt[656] PROGMEM;
  18    const uint8_t ke_fei_logo[252] PROGMEM;
  19
  20  #include "PCD8544_HWSPI.c"
  21
  22  // RAW GRAPHICS DATA
  23    const unsigned char * ke_fei_logo_pointer = ke_fei_logo;
  24
  25    #define UPCE_1 "\x14"
  26    #define UPCE_2 "\x15"
```

```
27
28  #define STOP FONT_7PT_STOPBYTE
29  const uint8_t font7pt[656] PROGMEM = {
30    STOP, STOP, STOP, STOP, STOP,  //0x00 (blank)
31    0x40, 0xa8, 0xaa, 0xf1, STOP,  //0x01 á
32    0x70, 0x89, 0x8a, 0x51, STOP,  //0x02 č
33    0x70, 0xa8, 0xaa, 0x31, STOP,  //0x03 é
34    0x70, 0xa9, 0xaa, 0x31, STOP,  //0x04 ě
35    0xfa, 0x01, STOP, STOP, STOP,  //0x05 í
36    0xf9, 0x0a, 0x09, STOP, STOP,  //0x06 ř
37    0x90, 0xa9, 0xaa, 0x49, STOP,  //0x07 š
38    0x78, 0x82, 0x85, 0xfa, STOP,  //0x08 ů
39    0x98, 0xa0, 0x42, 0x39, STOP,  //0x09 ý
40    0x88, 0xe9, 0xba, 0x89, STOP,  //0x0A ž
41    0x00, 0xb8, 0xf8, 0x78, 0x00,  // big comma, or  0x00, 0x38, 0x38, 0x38, 0x00, / /big dp
42    0x24, 0x42, 0xff, 0x42, 0x24,  //0x0C ↕
43    0x04, 0x02, 0x7f, 0x02, 0x04,  //0x0D ↑
44    0x20, 0x40, 0xfc, 0x40, 0x20,  //0x0E ↓
45    0x10, 0x10, 0x54, 0x38, 0x10,  //0x0F →
46    0x10, 0x38, 0x54, 0x10, 0x10,  //0x10 ←
47    0x45, 0x95, 0x95, 0x6d, STOP,  //0x11 ¯3 (periodic 3)
48    0x80, 0x70, 0x0e, 0x09, STOP,  //0x12 ƒ
49    0x79, 0x95, 0x95, 0x61, STOP,  //0x13 ¯6 (periodic 6)
50    0x3c, 0x70, 0xef, 0xef, 0xf1,  //0x14 (U (UPCE1)
51    0xff, 0xff, 0xaf, 0x7e, 0x3c,  //0x15 :) (UPCE2)
52    0xf0, 0x2c, 0x22, 0x2d, 0xf0,  //0x16 Á
53    0x7c, 0x83, 0x82, 0x83, 0x44,  //0x17 Č
54    0xfc, 0x94, 0x96, 0x95, 0x84,  //0x18 É
55    0xfe, 0x93, 0x92, 0x93, 0x82,  //0x19 Ě
56    0xfe, 0x01, STOP, STOP, STOP,  //0x1A Í
57    0xfe, 0x13, 0x32, 0x53, 0x8c,  //0x1B Ř
58    0x4c, 0x93, 0x92, 0x93, 0x64,  //0x1C Š
59    0x7c, 0x82, 0x85, 0x82, 0x7c,  //0x1D Ů
60    0x06, 0x08, 0xf2, 0x09, 0x06,  //0x1E Ý
61    0xc2, 0xa3, 0x92, 0x8b, 0x86,  //0x1F Ž
62    0x00, STOP, STOP, STOP, STOP,  //0x20
63    0xbe, STOP, STOP, STOP, STOP,  //0x21 !
64    0x07, 0x00, 0x07, STOP, STOP,  //0x22 "
65    0x24, 0xff, 0x24, 0xff, 0x24,  //0x23 #
66    0x24, 0x4a, 0xff, 0x52, 0x24,  //0x24 $
67    0x46, 0x26, 0x10, 0xc8, 0xc4,  //0x25 %
68    0x76, 0x89, 0x96, 0x60, 0x90,  //0x26 &
69    0x06, STOP, STOP, STOP, STOP,  //0x27 '
70    0x7c, 0x82, STOP, STOP, STOP,  //0x28 (
71    0x82, 0x7c, STOP, STOP, STOP,  //0x29 )
72    0x28, 0x10, 0x7c, 0x10, 0x28,  //0x2A *
73    0x10, 0x10, 0x7c, 0x10, 0x10,  //0x2B +
74    0xc0, STOP, STOP, STOP, STOP,  //0x2C ,
75    0x10, 0x10, 0x10, 0x10, STOP,  //0x2D -
76    0x80, STOP, STOP, STOP, STOP,  //0x2E .
77    0xc0, 0x30, 0x0c, 0x03, STOP,  //0x2F /
78    0x7c, 0xa2, 0x92, 0x8a, 0x7c,  //0x30 0
79    0x00, 0x84, 0xfe, 0x80, 0x00,  //0x31 1
80    0x84, 0xc2, 0xa2, 0x92, 0x8c,  //0x32 2
81    0x42, 0x82, 0x8a, 0x96, 0x62,  //0x33 3
82    0x30, 0x28, 0x24, 0xfe, 0x20,  //0x34 4
83    0x4e, 0x8a, 0x8a, 0x8a, 0x72,  //0x35 5
84    0x78, 0x94, 0x92, 0x92, 0x60,  //0x36 6
85    0x02, 0xe2, 0x12, 0x0a, 0x06,  //0x37 7
86    0x6c, 0x92, 0x92, 0x92, 0x6c,  //0x38 8
87    0x0c, 0x92, 0x92, 0x52, 0x3c,  //0x39 9
88    0x48, STOP, STOP, STOP, STOP,  //0x3A :
89    0xc8, STOP, STOP, STOP, STOP,  //0x3B ;
90    0x10, 0x28, 0x44, STOP, STOP,  //0x3C <
91    0x28, 0x28, 0x28, 0x28, STOP,  //0x3D =
92    0x44, 0x28, 0x10, STOP, STOP,  //0x3E >
93    0x04, 0xa2, 0x12, 0x0c, STOP,  //0x3F ?
94    0x3c, 0x42, 0x99, 0xa5, 0x3e,  //0x40 @
95    0xf0, 0x2c, 0x22, 0x2c, 0xf0,  //0x41 A
96    0xfe, 0x92, 0x92, 0x92, 0x6c,  //0x42 B
97    0x7c, 0x82, 0x82, 0x82, 0x44,  //0x43 C
98    0xfe, 0x82, 0x82, 0x44, 0x38,  //0x44 D
99    0xfe, 0x92, 0x92, 0x92, 0x82,  //0x45 E
100   0xfe, 0x12, 0x12, 0x12, 0x02,  //0x46 F
101   0x7c, 0x82, 0x92, 0x92, 0x74,  //0x47 G
102   0xfe, 0x10, 0x10, 0x10, 0xfe,  //0x48 H
103   0xfe, STOP, STOP, STOP, STOP,  //0x49 I
104   0x40, 0x80, 0x80, 0x7e, STOP,  //0x4A J
105   0xfe, 0x10, 0x28, 0x44, 0x82,  //0x4B K
106   0xfe, 0x80, 0x80, 0x80, STOP,  //0x4C L
```

```
107      0xfe, 0x0c, 0x30, 0x0c, 0xfe,  //0x4D M
108      0xfe, 0x0c, 0x10, 0x60, 0xfe,  //0x4E N
109      0x7c, 0x82, 0x82, 0x82, 0x7c,  //0x4F O
110      0xfe, 0x12, 0x12, 0x12, 0x0c,  //0x50 P
111      0x7c, 0x82, 0x82, 0xc2, 0xfc,  //0x51 Q
112      0xfe, 0x12, 0x32, 0x52, 0x8c,  //0x52 R
113      0x4c, 0x92, 0x92, 0x92, 0x64,  //0x53 S
114      0x02, 0x02, 0xfe, 0x02, 0x02,  //0x54 T
115      0x7e, 0x80, 0x80, 0x80, 0x7e,  //0x55 U
116      0x0e, 0x30, 0xc0, 0x30, 0x0e,  //0x56 V
117      0x3e, 0xc0, 0x30, 0xc0, 0x3e,  //0x57 W
118      0xc6, 0x28, 0x10, 0x28, 0xc6,  //0x58 X
119      0x06, 0x08, 0xf0, 0x08, 0x06,  //0x59 Y
120      0xc2, 0xa2, 0x92, 0x8a, 0x86,  //0x5A Z
121      0xfe, 0x82, STOP, STOP, STOP,  //0x5B [
122      0x03, 0x0c, 0x30, 0xc0, STOP,  /*0x5C \ */
123      0x82, 0xfe, STOP, STOP, STOP,  //0x5D ]
124      0x04, 0x02, 0x01, 0x02, 0x04,  //0x5E ^
125      0x80, 0x80, 0x80, 0x80, STOP,  //0x5F _
126      0x02, 0x04, STOP, STOP, STOP,  //0x60 `
127      0x40, 0xa8, 0xa8, 0xf0, STOP,  //0x61 a
128      0xfe, 0x88, 0x88, 0x70, STOP,  //0x62 b
129      0x70, 0x88, 0x88, 0x50, STOP,  //0x63 c
130      0x70, 0x88, 0x88, 0xfe, STOP,  //0x64 d
131      0x70, 0xa8, 0xa8, 0x30, STOP,  //0x65 e
132      0x10, 0xfe, 0x12, STOP, STOP,  //0x66 f
133      0x10, 0xa8, 0xa8, 0x78, STOP,  //0x67 g
134      0xfe, 0x08, 0x08, 0xf0, STOP,  //0x68 h
135      0xfa, STOP, STOP, STOP, STOP,  //0x69 i
136      0x80, 0xfa, STOP, STOP, STOP,  //0x6A j
137      0xfe, 0x20, 0x50, 0x88, STOP,  //0x6B k
138      0xfe, 0x80, STOP, STOP, STOP,  //0x6C l
139      0xf8, 0x08, 0xf0, 0x08, 0xf0,  //0x6D m
140      0xf8, 0x08, 0x08, 0xf0, STOP,  //0x6E n
141      0x70, 0x88, 0x88, 0x70, STOP,  //0x6F o
142      0xf8, 0x48, 0x48, 0x30, STOP,  //0x70 p
143      0x30, 0x48, 0x48, 0xf8, STOP,  //0x71 q
144      0xf8, 0x08, 0x08, STOP, STOP,  //0x72 r
145      0x90, 0xa8, 0xa8, 0x48, STOP,  //0x73 s
146      0x08, 0xfe, 0x88, STOP, STOP,  //0x74 t
147      0x78, 0x80, 0x80, 0xf8, STOP,  //0x75 u
148      0x78, 0x80, 0x60, 0x18, STOP,  //0x76 v
149      0x38, 0xc0, 0x30, 0xc0, 0x38,  //0x77 w
150      0x88, 0x70, 0x70, 0x88, STOP,  //0x78 x
151      0x98, 0xa0, 0x40, 0x38, STOP,  //0x79 y
152      0x88, 0xe8, 0xb8, 0x88, STOP,  //0x7A z
153      0x10, 0xee, 0x82, STOP, STOP,  //0x7B {
154      0xff, STOP, STOP, STOP, STOP,  //0x7C |
155      0x82, 0xee, 0x10, STOP, STOP,  //0x7D }
156      0x10, 0x08, 0x10, 0x20, 0x10,  //0x7E ~
157      0x04, 0x0a, 0x04, STOP, STOP   //0x7F °
158      //0xF0-0xFF are spaces of widths 0-15 + 1px padding of prev char, implemented in
lcd_draw7ptString()
159    };
160    #undef STOP
161
162    const uint8_t font22pt[450] PROGMEM = {
163      0xe0,0xf8,0xfc,0x3e,0x06,0x07,0x03,0x03,0x03,0x07,0x06,0x3e,0xfc,0xf8,0xe0, //0T
164      0x00,0x00,0x30,0x30,0x30,0x38,0x3c,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00, //1T
165      0x38,0x3c,0x3e,0x0e,0x07,0x03,0x03,0x03,0x03,0x03,0x07,0x0e,0xfe,0xfc,0xf8, //2T
166      0x30,0x3c,0x3e,0x0e,0x07,0x03,0x03,0x03,0x03,0x07,0x8f,0xfe,0xfc,0x78,0x00, //3T
167      0x00,0x00,0x00,0x00,0x00,0xc0,0xe0,0xf0,0x3c,0x1e,0xff,0xff,0xff,0x00,0x00, //4T
168      0x00,0xf8,0xff,0xff,0x87,0x83,0x83,0x83,0x83,0x83,0x83,0x03,0x03,0x03,0x00, //5T
169      0xc0,0xf8,0xfc,0x3e,0x0f,0x07,0x03,0x03,0x03,0x03,0x07,0x1e,0x1e,0x18,0x00, //6T
170      0x00,0x03,0x03,0x03,0x03,0x03,0x03,0x03,0x03,0x83,0xc3,0xe3,0x7b,0x3f,0x0f,0x07, //7T
171      0x30,0xfc,0xfe,0xde,0x07,0x07,0x03,0x03,0x03,0x07,0x07,0xde,0xfe,0xfc,0x30, //8T
172      0xf0,0xfc,0xfe,0x1e,0x07,0x03,0x03,0x03,0x03,0x03,0x07,0x1e,0xfe,0xfc,0xf0, //9T
173      0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff, //0M
174      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00, //1M
175      0x00,0x00,0x80,0xc0,0xe0,0xf0,0x70,0x38,0x1c,0x1c,0x0e,0x07,0x07,0x03,0x01, //2M
176      0x00,0x00,0x00,0x00,0x00,0x06,0x06,0x06,0x07,0x0f,0x0f,0x3d,0xf8,0xf8,0xe0, //3M
177      0xe0,0xf8,0xfc,0xde,0xc7,0xc3,0xc1,0xc0,0xc0,0xc0,0xff,0xff,0xff,0xc0,0xc0, //4M
178      0x07,0x07,0x07,0x03,0x03,0x01,0x01,0x01,0x01,0x01,0x03,0x0f,0xff,0xfe,0xf8, //5M
179      0xff,0xff,0xff,0x0e,0x07,0x03,0x03,0x03,0x03,0x03,0x07,0x0e,0xfe,0xfc,0xf0, //6M
180      0x00,0x00,0x00,0x00,0x00,0xc0,0xf8,0xfe,0x7f,0x0f,0x01,0x00,0x00,0x00,0x00, //7M
181      0xe0,0xf8,0xfd,0x3f,0x0f,0x0f,0x06,0x06,0x06,0x0f,0x0f,0x3f,0xfd,0xf8,0xe0, //8M
182      0x03,0x0f,0x1f,0x1c,0x38,0x30,0x30,0x30,0x30,0x30,0x38,0x9c,0xff,0xff,0xff, //9M
183      0x01,0x07,0x0f,0x1f,0x18,0x38,0x30,0x30,0x30,0x38,0x18,0x1f,0x0f,0x07,0x01, //0B
184      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3f,0x3f,0x3f,0x00,0x00,0x00,0x00,0x00, //1B
185      0x3c,0x3f,0x3f,0x33,0x31,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30, //2B
```

```
186     0x07,0x0f,0x1f,0x3c,0x38,0x30,0x30,0x30,0x30,0x38,0x38,0x1e,0x1f,0x0f,0x03, //3B
187     0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3f,0x3f,0x3f,0x00,0x00, //4B
188     0x07,0x0f,0x1f,0x1c,0x38,0x30,0x30,0x30,0x30,0x30,0x38,0x1e,0x1f,0x0f,0x03, //5B
189     0x03,0x0f,0x1f,0x1e,0x38,0x30,0x30,0x30,0x30,0x30,0x38,0x1e,0x1f,0x0f,0x03, //6B
190     0x00,0x00,0x00,0x00,0x3e,0x3f,0x3f,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00, //7B
191     0x03,0x0f,0x1f,0x1e,0x38,0x30,0x30,0x30,0x30,0x30,0x38,0x1e,0x1f,0x0f,0x03, //8B
192     0x00,0x06,0x1e,0x1e,0x38,0x30,0x30,0x30,0x30,0x38,0x3c,0x1f,0x0f,0x07,0x00, //9B
193   };
194   const uint8_t ke_fei_logo[252] PROGMEM = {
195     0x00, 0x00, 0x00, 0x88, 0x4c, 0x1e, 0xbc, 0x38, 0x70, 0xe4, 0xd4, 0x92, 0x2a, 0xaa,
196     0xaa, 0xff, 0xaa, 0xaa, 0x2a, 0x52, 0x54, 0x14, 0x28, 0x88, 0x10, 0x00, 0x80, 0x00,
197     0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80,
198     0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00,
199     0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00,
200     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00,
201     0x08, 0x08, 0x7e, 0x89, 0x3c, 0xcb, 0x18, 0x6e, 0x89, 0x3c, 0x4a, 0x89, 0x8b, 0x0a,
202     0x0c, 0xff, 0x08, 0x08, 0x81, 0x00, 0x00, 0x00, 0x00, 0xef, 0xa2, 0xa5, 0x28, 0xe5,
203     0xab, 0xae, 0x21, 0x0f, 0xe9, 0x00, 0x08, 0x00, 0xe0, 0x06, 0x0d, 0xea, 0x00, 0xef,
204     0xa0, 0xa6, 0x4d, 0xca, 0x20, 0x2f, 0x46, 0x09, 0xe1, 0xaf, 0xa9, 0x20, 0x0f, 0x01,
205     0x06, 0x09, 0x06, 0x01, 0x0f, 0x09, 0x06, 0x0d, 0x0a, 0x06, 0x09, 0x09, 0x0f, 0x01,
206     0x0e, 0x00, 0x0f, 0x01, 0x0e, 0x80, 0x8e, 0x80, 0x8f, 0x86, 0x89, 0x13, 0x0c, 0x07,
207     0x00, 0x00, 0x00, 0x01, 0x02, 0x04, 0x09, 0x12, 0x14, 0x29, 0x2a, 0x4a, 0x54, 0x55,
208     0x55, 0xff, 0x55, 0x55, 0x54, 0x4a, 0x2a, 0x28, 0x14, 0x13, 0x08, 0x00, 0x00, 0x03,
209     0x02, 0x02, 0x02, 0x00, 0x03, 0x00, 0x00, 0x00, 0x01, 0x02, 0x02, 0x01, 0x00, 0x03,
210     0x00, 0x00, 0x00, 0x01, 0x02, 0x02, 0x01, 0x00, 0x03, 0x02, 0x02, 0x02, 0x00, 0x00,
211     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
212     0x00, 0x00, 0x00, 0x1e, 0x38, 0x77, 0x77, 0x78, 0x7f, 0x7f, 0x57, 0x3f, 0x1e, 0x00
213   };
214
215   #endif
```

./src/lib/buttons.h

```
 1  #include <avr/io.h>
 2
 3  #define BUTTON_PORT PORTD
 4  #define BUTTON_DDR DDRD
 5  #define BUTTON_PIN PIND
 6  #define BTN_UP_MASK (1 << PD7)
 7  #define BTN_DN_MASK (1 << PD6)
 8  #define BTN_SL_MASK (1 << PD5)
 9  #define BUTTON_MASK (BTN_UP_MASK | BTN_DN_MASK | BTN_SL_MASK)
10
11  #define BUTTON_PCICR (1 << PCIE2) | (0 << PCIE1) | (0 << PCIE0) //only interrupt on pins in
PCMSK2 enabled
12  #define BUTTON_PCMSK0_B 0 //no pin change interrupts on PORTB
13  #define BUTTON_PCMSK1_C 0 //no pin change interrupts on PORTC
14  #define BUTTON_PCMSK2_D (1 << PCINT21) | (1 << PCINT22) | (1 << PCINT23) //pin change
interrupts on PD5-7
15  #define BUTTON_INT_VECT PCINT2_vect
16
17  #define BUTTON_LONGPRESS_THRESHOLD 2500
18  #define BUTTON_REPEAT_PERIOD 500
```

./src/lib/common.h

```
 1  #ifndef COMMON_H
 2  #define COMMON_H
 3
 4  #include <avr/io.h>
 5
 6  void delay_s(uint8_t delay);
 7  void delay_ms(uint8_t delay);
 8  void delay_us(uint8_t delay);
 9
10  #endif /* COMMON_H */
```

./src/lib/common.c

```
 1  #include "common.h"
 2
 3  void delay_ms(uint8_t ms) {
 4      uint16_t delay_count = F_CPU / 17500;
 5      volatile uint16_t i;
 6
 7      while (ms != 0) {
 8          for (i=0; i != delay_count; i++);
 9          ms--;
10      }
11  }
12
13  void delay_us(uint8_t us) {
```

```
    while (us != 0) {
        us--;
        us++;
        us--;
        us++;
        us--;
        us++;
        us--;
    }
}

void delay_s(uint8_t s) {
  while(s != 0) {
    delay_ms(250);
    delay_ms(250);
    delay_ms(250);
    delay_ms(250);
    s--;
  }
}

```

./lib/dsp_kit_comms.h

```
#ifndef __DSP_KIT_COMMS_H
#define __DSP_KIT_COMMS_H
#define DECIMAL_POINT_ASCII '.' //output symbol in UART data; can be changed to comma

void printchar(char myChar);
void printstring(char *s);

#include "../lib/serial.h"
//#include "../lib/PCD8544_HWSPI_C.h"
//#include "../lib/dsp_kit_buttons.h"
#include <stdlib.h>
uint8_t range_dp[] = {3, 1, 1}; // decimal point locations in result string per range

char numstring[8] = "\0\0\0\0\0\0\0"; //used as buffer by i2a to convert
char *resultstring = "-----"; //contains computed result; will be filled with numbers and dp

//int to decimal string, courtesy of https://stackoverflow.com/questions/16282047/convert-int-to-string-with-decimals
int i2a(char *s, int n){
  div_t qr;
  int pos;

  if(n == 0) return 0;

  qr = div(n, 10);
  pos = i2a(s, qr.quot);
  s[pos] = qr.rem + '0';
  return pos + 1;
}
int ui2a(char *s, uint16_t n){
  ldiv_t qr;
  int pos;

  if(n == 0) return 0;
  long nn = n;
  if(nn<0) nn+=65536;
  qr = ldiv(nn, 10);
  pos = ui2a(s, qr.quot);
  s[pos] = qr.rem + '0';
  return pos + 1;
}
char* my_itoa(char *output_buff, int num){
  char *p = output_buff;
  if(num < 0){
    *p++ = '-';
    num *= -1;
  } else if(num == 0)
    *p++ = '0';
  p[i2a(p, num)]='\0';
  return output_buff;
}
char* my_uitoa(char *output_buff, uint16_t num){
  char *p = output_buff;
  if(num == 0)
    *p++ = '0';
  p[i2a(p, num)]='\0';
```

```
  return output_buff;
}
//end of int to decimal string converter
void int2dec4digit(char *output_buff, uint16_t num) {
  output_buff = "----\0";
  div_t qr;
  uint8_t pos = 4;
  uint16_t n = num;
  if(num < 10000)
  {
    while(pos){
      pos--;
      qr = div(n, 10);
      n = qr.quot;
      output_buff[pos] = qr.rem + '0';
    }
  }
}

void int2dec4digitpoint(char *output_buff, uint16_t num, uint8_t pointlocation) { //converts integer including decimal point
  output_buff = "-----";
  output_buff[pointlocation] = DECIMAL_POINT_ASCII;
  div_t qr;
  uint8_t pos = 6;
  uint16_t n = num;
  if(num < 10000)
  {
    while(pos){
      pos--;
      if(pos == pointlocation) pos--; //do not overwrite decimal point
      qr = div(n, 10);
      n = qr.quot;
      output_buff[pos] = qr.rem + '0';
    }
  }
}

char* int2dec4digitpointArg(uint16_t num, uint8_t pointlocation) { //converts integer including decimal point
  char* output_buff = "-----";
  output_buff[pointlocation] = DECIMAL_POINT_ASCII;
  div_t qr;
  uint8_t pos = 5;
  output_buff[pos] = 0;
  uint16_t n = num;
  if(num < 10000)
  {
    while(pos){
      pos--;
      if(pos == pointlocation) pos--; //do not overwrite decimal point
      qr = div(n, 10);
      n = qr.quot;
      output_buff[pos] = qr.rem + '0';
    }
  } else {
    for (pos = 0; pos < 5; pos++)
    {
      if(pos != pointlocation) output_buff[pos] = '-';
    }
  }
  return output_buff;
}

void insertDecimalPointByPointer(char *inputString, uint8_t pointLocation)  //in-place insert of decimal point into string
{
  // char readchar; unused variable ?!
  uint8_t pos = 0;
  while (inputString[pos++]) {}; //seek forward to null-terminator, pos ends up one later
  while(pos) //we can't read from zero pos and write to negative pos
  {
    pos--;
    if (pos == pointLocation) {
      inputString[pos] = DECIMAL_POINT_ASCII;
      break;
    }
    inputString[pos+1] = inputString[pos];  //shift string right by one char
  }
}
```

```
133
134 char* insertDecimalPointByArgument(char *inputString, uint8_t pointLocation)  //inserts
decimal point into string at specified location
135 {
136   char *outputString = "\0\0\0\0\0\0";
137   uint8_t inPos = 0;
138   uint8_t outPos = 0;
139   char readchar;
140   do {
141     if(outPos == pointLocation){
142       outputString[outPos] = '.';
143       outPos++;
144     }
145     readchar = inputString[inPos];
146     outputString[outPos] = readchar;
147     inPos++; outPos++;
148   } while (readchar); //null-termination
149   return outputString;
150 }
151
152 char* insertDecimalPointRangeBased(char *inputString, uint8_t rangeindex)
153 {
154   return insertDecimalPointByArgument(inputString, range_dp[rangeindex]);
155 }
156
157 uint16_t sqrtlong(uint32_t num) //square root of 32-bit number into 16 bits, takes 82.4 us
158 {
159   uint16_t approx = 0;
160   uint16_t newapprox = 0;
161   for(uint8_t shift = 15; shift < 16; shift--)
162   {
163     newapprox = approx | (1 << (shift));
164     if((((uint32_t)newapprox) * ((uint32_t)newapprox)) <= num) approx = newapprox;
165   }
166   return approx;
167 }
168
169 #endif
```

./lib/dsp_kit_comms.c

```
  1 #define DECIMAL_POINT_ASCII '.' //output symbol in UART data; can be changed to comma
  2
  3 void printchar(char myChar);
  4 void printstring(char *s);
  5
  6 #include "../lib/serial.c"
  7 #include <stdlib.h>
  8
  9 char numstring[8] = "\0\0\0\0\0\0\0"; //used as buffer by i2a to convert
 10 char *resultstring = "-----"; //contains computed result; will be filled with numbers
 11
 12 //int to decimal string, courtesy of https://stackoverflow.com/questions/16282047/convert-
int-to-string-with-decimals
 13 int i2a(char *s, int n){
 14   div_t qr;
 15   int pos;
 16
 17   if(n == 0) return 0;
 18
 19   qr = div(n, 10);
 20   pos = i2a(s, qr.quot);
 21   s[pos] = qr.rem + '0';
 22   return pos + 1;
 23 }
 24 int ui2a(char *s, uint16_t n){
 25   ldiv_t qr;
 26   int pos;
 27
 28   if(n == 0) return 0;
 29   long nn = n;
 30   if(nn<0) nn+=65536;
 31   qr = ldiv(nn, 10);
 32   pos = ui2a(s, qr.quot);
 33   s[pos] = qr.rem + '0';
 34   return pos + 1;
 35 }
 36 char* my_itoa(char *output_buff, int num){
 37   char *p = output_buff;
 38   if(num < 0){
```

```
 39      *p++ = '-';
 40      num *= -1;
 41    } else if(num == 0)
 42      *p++ = '0';
 43    p[i2a(p, num)]='\0';
 44    return output_buff;
 45  }
 46  char* my_uitoa(char *output_buff, uint16_t num){
 47    char *p = output_buff;
 48    if(0){
 49      *p++ = '-';
 50      num *= -1;
 51    } else if(num == 0)
 52      *p++ = '0';
 53    p[i2a(p, num)]='\0';
 54    return output_buff;
 55  }
 56  //end of int to decimal string converter
 57  void int2dec4digit(char *output_buff, uint16_t num) {
 58    output_buff = "----";
 59    div_t qr;
 60    uint8_t pos = 4;
 61    uint16_t n = num;
 62    if(num < 10000)
 63    {
 64      while(pos){
 65        pos--;
 66        qr = div(n, 10);
 67        n = qr.quot;
 68        output_buff[pos] = qr.rem + '0';
 69      }
 70    }
 71  }
 72
 73  void int2dec4digitpoint(char *output_buff, uint16_t num, uint8_t pointlocation) { //converts
integer including decimal point
 74    output_buff = "-----";
 75    output_buff[pointlocation] = DECIMAL_POINT_ASCII;
 76    div_t qr;
 77    uint8_t pos = 5;
 78    uint16_t n = num;
 79    if(num < 10000)
 80    {
 81      while(pos){
 82        pos--;
 83        if(pos == pointlocation) pos--; //do not overwrite decimal point
 84        qr = div(n, 10);
 85        n = qr.quot;
 86        output_buff[pos] = qr.rem + '0';
 87      }
 88    }
 89  }
 90
 91  void insertDecimalPointByPointer(char *inputString, uint8_t pointLocation)  //in-place
insert of decimal point into string
 92  {
 93    char readchar;
 94    uint8_t pos = 0;
 95    while (inputString[pos++]) {}; //seek forward to null-terminator, pos ends up one later
 96    while(pos) //we can't read from zero pos and write to negative pos
 97    {
 98      pos--;
 99      if (pos == pointLocation) {
100        inputString[pos] = DECIMAL_POINT_ASCII;
101        break;
102      }
103      inputString[pos+1] = inputString[pos];  //shift string right by one char
104    }
105  }
106
107  char* insertDecimalPointByArgument(char *inputString, uint8_t pointLocation)  //inserts
decimal point into string at specified location
108  {
109    char *outputString = "\0\0\0\0\0\0";
110    uint8_t inPos = 0;
111    uint8_t outPos = 0;
112    char readchar;
113    do {
114      if(outPos == pointLocation){
115        outputString[outPos] = '.';
```

```
      outPos++;
    }
    readchar = inputString[inPos];
    outputString[outPos] = readchar;
    inPos++; outPos++;
  } while (readchar); //null-termination
  return outputString;
}

char* insertDecimalPointRangeBased(char *inputString, uint8_t range)
{
  if(range == 1) return insertDecimalPointByArgument(inputString, 3);
  else return insertDecimalPointByArgument(inputString, 1);
}

void printchar(char myChar) {
  if (myChar != '\0') serial_send(myChar);
}

void printstring(char *s){
  uint8_t offset = 0;
  char thisChar = 0;
  do {
    thisChar = s[offset];
    printchar(thisChar);
    offset++;
  } while (thisChar != '\0');
}

uint16_t sqrtlong(uint32_t num) //sq. root of 32-bit unsigned int into 16 bits, 82.4 µs
{
  uint16_t approx = 0;
  uint16_t newapprox = 0;
  for(uint8_t shift = 15; shift < 16; shift--)
  {
    newapprox = approx | (1 << (shift));
    if((((uint32_t)newapprox) * ((uint32_t)newapprox)) <= num) approx = newapprox;
  }
  return approx;
}

```

./lib/dsp_kit_menuitems.h

```
#ifndef __HDO_MENUITEMS_H
#define __HDO_MENUITEMS_H
#include <avr/eeprom.h>
#include "dsp_kit_comms.h"

// DECLARATIONS
void load_settings_eeprom();
void update_settings_eeprom();
void apply_setting(uint8_t setting);
void lcd_drawValue(uint8_t index, uint8_t selected);
void lcd_drawMenuPage(uint8_t pageIndex, uint8_t selectedIndex, uint8_t clear);
void apply_freqnum(uint8_t freq_num);
void write_eeprom();
void read_eeprom();

#define MENUITEM_IS_SELECTED 1
#define MENUITEM_ISNT_SELECTED 0

#define MENUPAGE_OVERDRAW 0
#define MENUPAGE_CLEAR_BEFORE_DRAW 1

typedef struct dsp_setting_item {
  uint8_t *const values; //possible values index
  const uint8_t valuesize; //size of above array
  uint8_t *const currentvalueindex; //pointer to variable
  const uint8_t rect_x; // starting column of menu item rectangle
  const uint8_t valu_x; // starting column of value string
  const uint8_t unit_x; // starting column of unit string
  const uint8_t rect_w; // total width of menu item rectangle
  const uint8_t legd_x; // starting column of legend string
  const char     *LCD_unit_string;  //    LCD unit string
  const char  *serial_unit_string;  // serial unit string
  const char     *LCD_legd_string;  //    LCD legend string
  const char  *serial_legd_string;  // serial legend string
  const char    **LCD_value_strings; //points to array of    LCD value strings
  const char **serial_value_strings; //points to array of serial value strings
```

```
} dsp_setting;

#define NUMBER_OF_SETTINGS 2
uint8_t currentrangeindex = 2; //initialize range as 10
const dsp_setting menuitems[NUMBER_OF_SETTINGS];

#include "dsp_kit_menuitems.c"

#endif
```

./lib/dsp_kit_menuitems.c

```
#include "dsp_kit_menuitems.h"
#include "graphics.h"
#include "dsp_kit_buttons.h"
#include "leds.h"
#include <avr/eeprom.h>

//uint8_t range = 1;
uint8_t freqnum = 0;
uint8_t eval_flag = 0;
uint8_t fast_evals = 0;
int16_t samplesbe = 0;
uint32_t cumsamplesS = 0;

uint8_t selected_menuitem = 0xff; // -1 = none

uint8_t rangevalues[] = {1, 3, 10};
//RANGE VALUE STRINGS
  const char *range_lcd_strings[] = {"\xF4""1", "\xF4""3", "10"};
  const char *range_serial_strings[] = {"1", "3", "10"};

uint8_t currentfreqindex = 1;  //initialize freq as 217
//FREQUENCY VALUE STRINGS
  const char *freq_lcd_strings[] = {"\xF6""183,\x11", "\xF6""216,\x13", "\xF6""283,\x11",
"\xF6""760", "1060"};
  const char *freq_serial_strings[] = {"183.33", "216.67", "283.33", "760", "1060"};
  uint8_t freqvalues[] = {0};

#define RANGE menuitems[0]
#define HDO_F menuitems[1]
#define RANGE_VALUE RANGE.values[*RANGE.currentvalueindex]
#define RANGE_INDEX *RANGE.currentvalueindex

const dsp_setting menuitems[NUMBER_OF_SETTINGS] = {
  { //RANGE
    rangevalues, 3, //pointer and size of possible value array
    &currentrangeindex, //pointer to variable storing current value index
    //menu dimensions: rx, vx, ux, rw, lx
                       0,  2, 17, 26, 12,
    "V", //LCD unit string
    "V", //serial unit string
    "\x0C", //LCD legend string: "↕"
    "Range", //serial legend string
    range_lcd_strings,
    range_serial_strings,
  },
  { //HDO_f
    freqvalues, 5, //pointer and size of possible value array (not needed so NULL pointer)
    &currentfreqindex, //pointer to variable storing current value index
    //menu dimensions: rx, vx, ux, rw, lx
                      35, 37, 70, 49, 49,
    "Hz", //LCD unit string
    "Hz", //serial unit string
    "HDO \x12", //LCD legend string: "HDO ƒ"
    "HDO f", //serial legend string
    freq_lcd_strings,
    freq_serial_strings,
  }
};
const dsp_setting *hdo_f_ = &menuitems[1];

uint8_t EEMEM settings_storage[NUMBER_OF_SETTINGS];

void write_eeprom() {
  // compile setting value indices into compact array of bytes...
  uint8_t settings_bytes[NUMBER_OF_SETTINGS];
  for (uint8_t i = 0; i < NUMBER_OF_SETTINGS; i++) {
    settings_bytes[i] = *menuitems[i].currentvalueindex;
  }
```

```
  68    //              ...then write RAM → EEPROM if different
  69    eeprom_update_block(settings_bytes, settings_storage, NUMBER_OF_SETTINGS);
  70  }
  71
  72  void read_eeprom() { //broken? bad MCU?
  73    // initialize compact array of setting indices...
  74    uint8_t settings_bytes[NUMBER_OF_SETTINGS];
  75    //             ..read block RAM ← EEPROM...
  76    eeprom_read_block(settings_bytes, settings_storage, NUMBER_OF_SETTINGS);
  77    // ... do a sanity check on each and apply settings
  78    for (uint8_t i = 0; i < NUMBER_OF_SETTINGS; i++) {
  79      if(settings_bytes[i] < menuitems[i].valuesize) *menuitems[i].currentvalueindex =
settings_bytes[i];
  80      apply_setting(i);
  81    }
  82  }
  83
  84  //MENU PAGES
  85    #define NUMBER_OF_PAGES 1
  86    uint8_t settings_menu_currentpage = 0;
  87    uint8_t menuPages[] = {0, NUMBER_OF_SETTINGS}; // first item by page; array of
NUMBER_OF_PAGES+1 length: last value must be NUMBER_OF_SETTINGS
  88
  89  uint8_t range_gain[] = {30, 10, 3}; // gain after first biquad required to correct for range
  90
  91  void updateRangeLED() {
  92    PORTC = portC_rangeLED[*RANGE.currentvalueindex];
  93     DDRC =  ddrC_rangeLED[*RANGE.currentvalueindex];
  94  }
  95
  96  void settingsMenuDeselect() {
  97    selected_menuitem = 0xFF; //-1 = none
  98    settings_menu_currentpage = 0;
  99    lcd_drawMenuPage(settings_menu_currentpage, selected_menuitem, 0);
 100    //update_settings_eeprom();
 101  }
 102  void settingsMenuSelectNext() {
 103    selected_menuitem++;
 104    if(selected_menuitem >= NUMBER_OF_SETTINGS) { settingsMenuDeselect(); return;} //end
 105    if(selected_menuitem >= menuPages[settings_menu_currentpage+1]){ //new page reached!
 106      settings_menu_currentpage++;
 107      lcd_drawMenuPage(settings_menu_currentpage, selected_menuitem, 1);
 108    }
 109    else {
 110      if(selected_menuitem) lcd_drawMenuItem_newway(selected_menuitem-1,
MENUITEM_ISNT_SELECTED); //show previous as deselected
 111      lcd_drawMenuItem_newway(selected_menuitem, MENUITEM_IS_SELECTED);
 112    }
 113    //update_settings_eeprom();
 114  }
 115  uint8_t settings_just_changed = 1;
 116  void apply_setting(uint8_t settingIndex){ //update corresponding global variables in to
match setvalue array
 117    uint8_t range_num;
 118    range_num = *RANGE.currentvalueindex;
 119    switch (settingIndex)
 120    {
 121    case 0: //range
 122      showLED(range_num + 1);
 123      fast_evals = 0; //prevents overwriting digits
 124      break;
 125    case 1: //HDO f
 126      settings_just_changed = 1;
 127      apply_freqnum(*menuitems[1].currentvalueindex);
 128      break;
 129    default: break;
 130    }
 131    lcd_drawunit(range_num);
 132    settings_just_changed = 1;
 133    samplesbe = 0;
 134    eval_flag = 0;
 135    fast_evals = 0;
 136    cumsampleS = 0;
 137    printstring(menuitems[settingIndex].serial_legd_string);
 138    printstring(": ");
 139    printstring(menuitems[settingIndex].serial_value_strings[*menuitems[settingIndex].currentv
alueindex]);
 140    printstring(" ");
 141    printstring(menuitems[settingIndex].serial_unit_string);
 142    printstring("\r\n");
```

```
143  }
144
145  void settings_change_value(int8_t delta){ // inc (Δ=+1) or dec (Δ=-1) value, apply change
146    if(selected_menuitem != 0xFF) { //something is selected
147      uint8_t newvalue = *menuitems[selected_menuitem].currentvalueindex + delta;
148      if (newvalue >= 0xf0) newvalue += menuitems[selected_menuitem].valuesize;
149      if (newvalue >= menuitems[selected_menuitem].valuesize) newvalue -=
menuitems[selected_menuitem].valuesize;
150      *menuitems[selected_menuitem].currentvalueindex = newvalue;
151
152      lcd_drawMenuItem(selected_menuitem, MENUITEM_IS_SELECTED);
153      apply_setting(selected_menuitem);
154    }
155  }
156
157  void takeButtonActions(uint8_t buttonActions) {
158    for(uint8_t i = 0; i < 6; i++)
159    {
160      if(buttonActions & 0x01)
161      {
162        switch (i)
163        {
164        case 0:     //PD5 = SL pressed
165          settingsMenuSelectNext();
166          break;
167        case 4:     //PD6 = DN held
168        if(canonicalButtonState != (1 << 6)) break; //ensure only DN held
169        case 1:     //PD6 = DN pressed
170          settings_change_value(-1); //decrement value
171          break;
172        case 5:     //PD7 = UP held
173        if(canonicalButtonState != (1 << 7)) break; //ensure only UP held
174        case 2:     //PD7 = UP pressed
175          settings_change_value(+1); //increment value
176          break;
177        case 3:     //PD5 = SL held
178          buttonHeldFor[0] = 0x7ffe; //disables repeated actions for select button
179          settingsMenuDeselect();
180          break;
181        default:  //impossible
182          break;
183        }
184        buttonActionFlags &= ~(1 << i);  //clear event flag
185      }
186      buttonActions = buttonActions >> 1;
187    }
188  }
```

./lib/dsp_kit_pwm.h

```
 1  #ifndef __DSP_KIT_PWM_H
 2  #define __DSP_KIT_PWM_H
 3  #include <avr/io.h>
 4
 5  //DECLARATIONS
 6    void init_pwm();
 7    void set_pwm_value(uint8_t pwm_value);
 8
 9  #include "dsp_kit_pwm.c"
10  #endif
```

./lib/dsp_kit_pwm.c

```
 1  #include "dsp_kit_pwm.h"
 2  #define MAX_DEFLECTION 180
 3
 4  void init_pwm(){
 5    //using 8-bit TC2 in fastPWM mode for PWM on OC2B pin (PD3)
 6    TCCR2A = (3 << COM2B0) | (3 << WGM20); //OC2B pin inverted FastPWM to un-invert amp in HW
 7    TCCR2B = (0 << WGM22) | (2 << CS20); //top of FastPWM is 0xFF; CLK/8 ~ f = 7812.5 Hz
 8  };
 9
10  void set_pwm_value(uint8_t pwm_value){
11    OCR2B = pwm_value;
12    //note: the OCR2B register is double-buffered to update on TOV2 so we can't undercut
TCNT2, not that it matters at 7.8 kHz
13  };
14
```

./lib/dsp.h

```
#ifndef __DSP_H
#define __DSP_H
void startSampleTimer();
void processSample();
void checkForEval();
void evalFastUpdate();
void evalSlowUpdate();
#include "dsp_kit_menuitems.h"
#include "dsp.c"

#endif
```

./lib/dsp.c

```
#include "dsp.h"
#include "dsp_kit_buttons.h"
#include "dsp_kit_menuitems.h"
#include "dsp_kit_pwm.h"
#include "dsp_kit_comms.h"
#include <avr/io.h>
#include <avr/interrupt.h>

#define MIN_SIGNED_INT 0x8000

#define ADC_IN_OFFSET 512 //removes DC offset of ADC prescaler (as Chebyshev-2 HPF won't)
#define SAMPLES_BEFORE_EVAL 500 //analog output at 10 S/s

#define MFP_HPF 512 //multiplication factor for coeffs in HPF
#define MFP_HPF_SHIFT 9 //512 = 2^9

uint8_t dummy = 0; //used to read ADC value to clear interrupt
volatile int x_max = MIN_SIGNED_INT;
volatile int y_max = MIN_SIGNED_INT;
uint32_t rms;
uint16_t scaled_result;
uint8_t z_range_comp[3] = {30, 10, 3};
uint32_t prevcumsamplesS = 0; // sample squares pushed to S => MS => RMS calculation

#include "dsp_coeffs.h"

void startSampleTimer() { //use TC0 to take samples at accurate time
  TCCR0A = 2; //CTC mode of TC0
  TCCR0B = 3; //TC0 prescaler 64x
  OCR0A =  49; //reset TC0 every 50 counts (200 µs / 5 kHz @ f_XTAL = 16 MHz & prescaler 64)
  ADMUX = (0 << ADLAR) | PC0; //ADC ref = AREF pin @ 2.5V (431); ADC right-adjust result;
CH2 (ADC1) for direct ADC test option
  ADCSRA = (1 << ADEN)  //ADC enable
         | (1 << ADATE) //ADC auto-trigger enable
         | (7 << ADPS0) //ADPS (ADC prescaler) = 128
         | (1 << ADIE)  //ADC interrupt enable
         | (1 << ADSC);  //single conversion start
  ADCSRB = 3; //ADC Trigger Source = OCR0A (p308 in datasheet)
  //ADCSRB = 5; // DEBUG: ADC Trigger Source = OCR1B (p308 in datasheet)
  TIMSK0 = 2; //interrupt on OCR0A
  sei();
}

void apply_freqnum(uint8_t freq_num) { //changes global variables in response to freqnum
  cli(); //disables measurements while coefficients are being changed; not that it matters
  samplesbe = 0;  //counts samples before evaluation
  cumsamplesS = 0;
  khw = bpf_coeffs[0 + 12*freq_num];
  khw1= bpf_coeffs[1 + 12*freq_num];
  khw2= bpf_coeffs[2 + 12*freq_num];
  kwz = bpf_coeffs[3 + 12*freq_num];
  kww1= bpf_coeffs[4 + 12*freq_num];
  kww2= bpf_coeffs[5 + 12*freq_num];
  kyp = bpf_coeffs[6 + 12*freq_num];
  kyp1= bpf_coeffs[7 + 12*freq_num];
  kyp2= bpf_coeffs[8 + 12*freq_num];
  kph = bpf_coeffs[9 + 12*freq_num];
  kpp1= bpf_coeffs[10+ 12*freq_num];
  kpp2= bpf_coeffs[11+ 12*freq_num];
  MFP_2_SHIFT = bpf_shift[0 + 2*freq_num];
  MFP_2A_SHIFT = bpf_shift_A[0 + 2*freq_num];
  MFP_3_SHIFT = bpf_shift[1 + 2*freq_num];
  MFP_MAG_COMP = bpf_shift_A[1 + 2*freq_num];
  char* resultstring = insertDecimalPointRangeBased("----", RANGE_INDEX);
  fast_evals = 0; //prevents overwriting digits
```

```
  lcd_drawunit(RANGE_INDEX);
  lcd_drawresultstring(resultstring);
}

void processSample() { //runs at sample freq
  //TIFR0 = 1 << OCF0A; //clears interrupt vector
  PORTD = 0xE4; //D2=HIGH signals DSP start for oscilloscope timing
  //MAIN DSP CALCULATION
  x = ((ADCH * 256) + ADCL - ADC_IN_OFFSET) >> 0;
  v = ((int32_t)(kvx*x+kvv1*v1+kvv2*v2)) >> MFP_HPF_SHIFT; //biquad 1 ◄ input + feedback
  dummy = ADCH;
  z = ((int32_t)(kzv*v+kzv1*v1+kzv2*v2)) >> (MFP_HPF_SHIFT-1); //biquad 1 ► output
  z *= z_range_comp[RANGE_INDEX];

  w = ((int32_t)(kwz*z+kww1*w1+kww2*w2)) >> MFP_2_SHIFT;   //biquad 2 ◄ input + feedback
  dummy = ADCL;
  h = ((int32_t)(khw*w+khw1*w1+khw2*w2)) >> (MFP_2_SHIFT-MFP_2A_SHIFT); //biquad 2 ► output
  p = ((int32_t)(kph*h+kpp1*p1+kpp2*p2)) >> MFP_3_SHIFT;   //biquad 3 ◄ input + feedback
  dummy = ADCH;
  y = ((int32_t)(kyp*p+kyp1*p1+kyp2*p2)) >> (MFP_3_SHIFT);   //biquad 3 ► output
  //HISTORY SHIFT
    v2 = v1; v1 = v;
    w2 = w1; w1 = w;
    p2 = p1; p1 = p;

  //CUMULATE SQUARES
  cumsamplesS += (y*y); //cumulates sum of squares for RMS calculation
  samplesbe++;
  if (samplesbe == SAMPLES_BEFORE_EVAL){
    samplesbe = 0;
    eval_flag += 1;
    prevcumsamplesS = cumsamplesS;
    cumsamplesS = 0;
  }

  ADCSRA |= (1<<ADIF); //clear interrupt flag
  checkButtons();
  PORTD = 0xE0; //D2=LOW signals DSP end
}

void checkForEval() {  //runs outside interrupts, checks whether it's time to evaluate
  if(eval_flag) evalFastUpdate();
  if (fast_evals > 4) evalSlowUpdate();
}

uint16_t rms_lcd;

void evalFastUpdate() {
  rms = sqrtlong(prevcumsamplesS / (eval_flag)); //actually not rMs, avoiding division 4 mean
  eval_flag --;
  rms *= gain_comp[*menuitems[1].currentvalueindex]; rms = rms >> (5-MFP_MAG_COMP);
//compensates gain of bandpass filters
  uint8_t pwm_value = (((uint32_t)rms)*9)/500;
  if(rms > 12000) pwm_value = 210;
  set_pwm_value(pwm_value);

  uint16_t rms_display = rms;
  if(RANGE_INDEX == 1) rms_display = rms * 3 / 10; //unscales range=3 to actual mV
  rms_lcd += rms_display;
  char* displaystring;
  displaystring = int2dec4digitpointArg(rms_display, range_dp[RANGE_INDEX]);
  printstring(displaystring);
  printstring("\r\n");
  fast_evals++;
}

void evalSlowUpdate() {
  rms_lcd /= fast_evals;
  char* displaystring;
  displaystring = int2dec4digitpointArg(rms_lcd, range_dp[RANGE_INDEX]);
  if(!settings_just_changed) lcd_drawresultstring(displaystring);
  rms_lcd = 0;
  fast_evals = 0;
  settings_just_changed = 0;
}

EMPTY_INTERRUPT(TIMER0_COMPA_vect); //only starts ADC; CPU does not serve interrupt

ISR(ADC_vect) {processSample();}
```

./lib/graphics.h

```
#ifndef __HDO_GRAPHICS_H
#define __HDO_GRAPHICS_H
//declarations
  void lcd_splashscreen();
  void lcd_drawresultstring(char *resultstring);
  void lcd_drawunit(uint8_t range);
  void lcd_drawMenuItem_newway(uint8_t menuitem_index, uint8_t selected);
  void lcd_drawMenuItem(uint8_t index, char* value, uint8_t selected);
  void lcd_drawLegend_newway(uint8_t index);
#include "dsp_kit_menuitems.h"

#include "graphics.c"
#endif
```

./lib/graphics.c

```
// draws project-relevant graphics on Nokia 5110 display using library
#include "graphics.h"
#include "PCD8544_HWSPI.h"
#include "dsp_kit_menuitems.h"
#define FONT_22PT_WIDTH 15
#define FONT_22PT_GAP 2

void lcd_splashscreen() {
  lcd_draw7ptString("TrpisovskyV_", 3, 0, 14, 84, 0, 0); serial_dispatch();
  lcd_draw7ptString("SelektivniVoltmetr_", 4, 0, 0, 84, 0, 0); serial_dispatch();
  lcd_draw7ptString("BB_2024", 5, 0, 22, 84, 0, 0); serial_dispatch();
  lcd_drawBitmap(ke_fei_logo_pointer, 252);
}

void lcd_drawresultstring(char *resultstring){ //draw 4 digit result in mV or mV/10
  uint8_t col = 0;
  for(uint8_t i = 0; i < 5; i++) {
    switch (resultstring[i])
    {
      case 0:  //null-terminated early??
        i = 5; //forces break from for-loop
        break;
      case '.': //decimal point is here
        col -= (FONT_22PT_WIDTH + FONT_22PT_GAP - FONT_7PT_CHARACTER_WIDTH); //comma width
incl. padding is 5px as opposed to 17px; drawing the comma is done by lcd_drawrange()
        break;
      case '-': //used instead of digits as N/A symbol but could be for negative numbers
        lcd_bar(0x00, 0, col, FONT_22PT_WIDTH);
        lcd_bar((0b11 << 3), 1, col, FONT_22PT_WIDTH); //'—' dash: 2 pixels high fullwidth
rectangle in the very middle
        lcd_bar(0x00, 2, col, FONT_22PT_WIDTH);
        break;
      default: //digit
        lcd_draw22ptNum(0,col,(resultstring[i]-'0'));
        break;
    }
    col += FONT_22PT_WIDTH + FONT_22PT_GAP; //15 px width + 2 px gap
  }
}

void lcd_drawunit(uint8_t rangeindex){ //draw unit for range
  lcd_bar(0x00, 0, 0, 3*84); //blank upper half of display
  if (!rangeindex){
    lcd_draw7ptString("mV", 1, 73, 0, 11, 0, 0);
  }
  else {
    lcd_draw7ptString("V", 1, 73, 6, 11, 0, 0);
  }
  lcd_draw7ptString("\x0b", 2, 16+(range_dp[rangeindex]-1)*(FONT_22PT_WIDTH+FONT_22PT_GAP),
0, 5, 0, 0); //decimal comma
  char* displaystring;
  if(rangeindex) displaystring = "-.---"; else displaystring = "---.-";
  lcd_drawresultstring(displaystring);
}

void lcd_drawMenuItem(uint8_t menuitem_index, uint8_t selected){
  dsp_setting menuitem = menuitems[menuitem_index];
  const char* valuLCDstring = menuitem.LCD_value_strings[*menuitem.currentvalueindex];
  const char* unitLCDstring = menuitem.LCD_unit_string;

  uint8_t fillchar = 0x00;
  if(selected) fillchar = 0xfc;
```

```
 60    lcd_draw7ptString(valuLCDstring, 3, menuitem.rect_x, menuitem.valu_x-menuitem.rect_x,
menuitem.unit_x-menuitem.rect_x, fillchar, 4);
 61    lcd_draw7ptString(unitLCDstring, 3, menuitem.unit_x, 0, menuitem.rect_w+menuitem.rect_x-
menuitem.unit_x, fillchar, 4);
 62    if(selected) fillchar = 0x7f;
 63    lcd_draw7ptString(valuLCDstring, 4, menuitem.rect_x, menuitem.valu_x-menuitem.rect_x,
menuitem.unit_x-menuitem.rect_x, fillchar, -4);
 64    lcd_draw7ptString(unitLCDstring, 4, menuitem.unit_x, 0, menuitem.rect_w+menuitem.rect_x-
menuitem.unit_x, fillchar, -4);
 65  }
 66
 67  void lcd_drawLegend(uint8_t menuitem_index) {
 68    dsp_setting menuitem = menuitems[menuitem_index];
 69    lcd_draw7ptString(menuitem.LCD_legd_string, 5, menuitem.legd_x, 0, 0, 0, 0);
 70  }
 71
 72  void lcd_drawMenuPage(uint8_t pageIndex, uint8_t selectedIndex, uint8_t clear){
 73    if(clear) lcd_bar(0x0, 3, 0, 252); //blank lower half of display
 74    for (uint8_t index = menuPages[pageIndex]; index < menuPages[pageIndex+1]; index++){
 75      uint8_t selected = (selectedIndex == index); //boolean
 76      lcd_drawMenuItem_newway(index, selected);
 77      lcd_drawLegend_newway(index);
 78    }
 79  }
 80
```

./lib/leds.h

```
  1  #ifndef LEDS_H
  2  #define LEDS_H
  3
  4  #include <avr/io.h>
  5
  6  /* RANGE LED SCHEMATIC
  7
  8  "1"  ┌─|<]─┐
  9       "     │
 10  "3"  ├─[>|─┼─|<]─┐ "10"
 11       "     "     │
 12       │ 2kΩ │     │
 13       │     │     │
 14     PC5   PC4   PC3   */
 15                            //none,     "1"    ,     "3"    ,    "10"
 16  uint8_t portC_rangeLED[4] = {0b0, 0b00010000, 0b00100000, 0b00001000};
 17  uint8_t  ddrC_rangeLED[4] = {0b0, 0b00110000, 0b00110000, 0b00011000};
 18
 19  void showLED(uint8_t num){
 20    DDRC  =  ddrC_rangeLED[num];
 21    PORTC = portC_rangeLED[num];
 22  }
 23
 24  #endif
```

./src/main.c

```
#define _XTAL_FREQ 16000000UL
#ifndef __AVR_ATmega328__
#define __AVR_ATmega328__
#endif

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "../lib/dsp_kit_comms.h"
#include "../lib/serial.h"
#include "../lib/PCD8544_HWSPI.h"
#include "../lib/graphics.h"
#include "../lib/dsp_kit_menuitems.h"
#include "../lib/dsp.h"
#include "../lib/dsp_kit_pwm.h"

void setup() {
  _delay_loop_2(65535);
  // PIN ASSIGNMENTS
    PORTB= 0b00001110;  //chip select for LCD goes high ASAP
    //     7 ┌──────XTAL        7         N/A         7 ┌──────B-DN
    //     6 │┌─────XTAL        6 ┌──────!RST        6 │┌─────B-UP
    //     5 ││┌────SCK         5 │┌─────LED3         5 ││┌────B-SL
    //     4 │││┌───CIPO        4 ││┌────LED2         4 │││    NC
    //     3 ││││┌──COPI        3 │││┌───LED1         3 │││┌───PWM2
    //     2 │││││┌─!CS         2 ││││    NC          2 ││││┌──(DEBUG) LED
    //     1 ││││││┌PWM1        1 │││└──ADC1         1 │└└└└──TXD
    //     0 ▼▼▲▼▲▲▲ NC          0 ▼▲▲▲ ▼▼ADC0        0 ▼▼▼ ▲▲▲ NC
    DDRB = 0b00101110;    DDRC = 0b00111000;    DDRD = 0b00001110;
                                                PORTD= 0b11101101; // button pullup, PWM 1
  //read_eeprom(); removed since this caused the program to hang; bad MCU?
  // INTERFACE INITIALIZATION
  serial_init();
  printstring("\r\nTrpisovskyV_SelektivniVoltmetr_BB_2024\r\n");
  init_pwm();
  lcd_startdisp();
  set_pwm_value(180);
  lcd_splashscreen();
  for (uint8_t i = 0; i < 255; i++) _delay_loop_2(65535);
  lcd_wipe();
  lcd_drawMenuPage(0, -1, 1);
  apply_freqnum(*menuitems[1].currentvalueindex);
  apply_setting(0);
  buttonActionFlags = 0;
  startSampleTimer();
}

void loop() {
  checkForEval();
  takeButtonActions(buttonActionFlags);
}

int main(void) {
  setup();
  for(;;) loop(); //imitates Arduino's setup-loop architecture
}
```

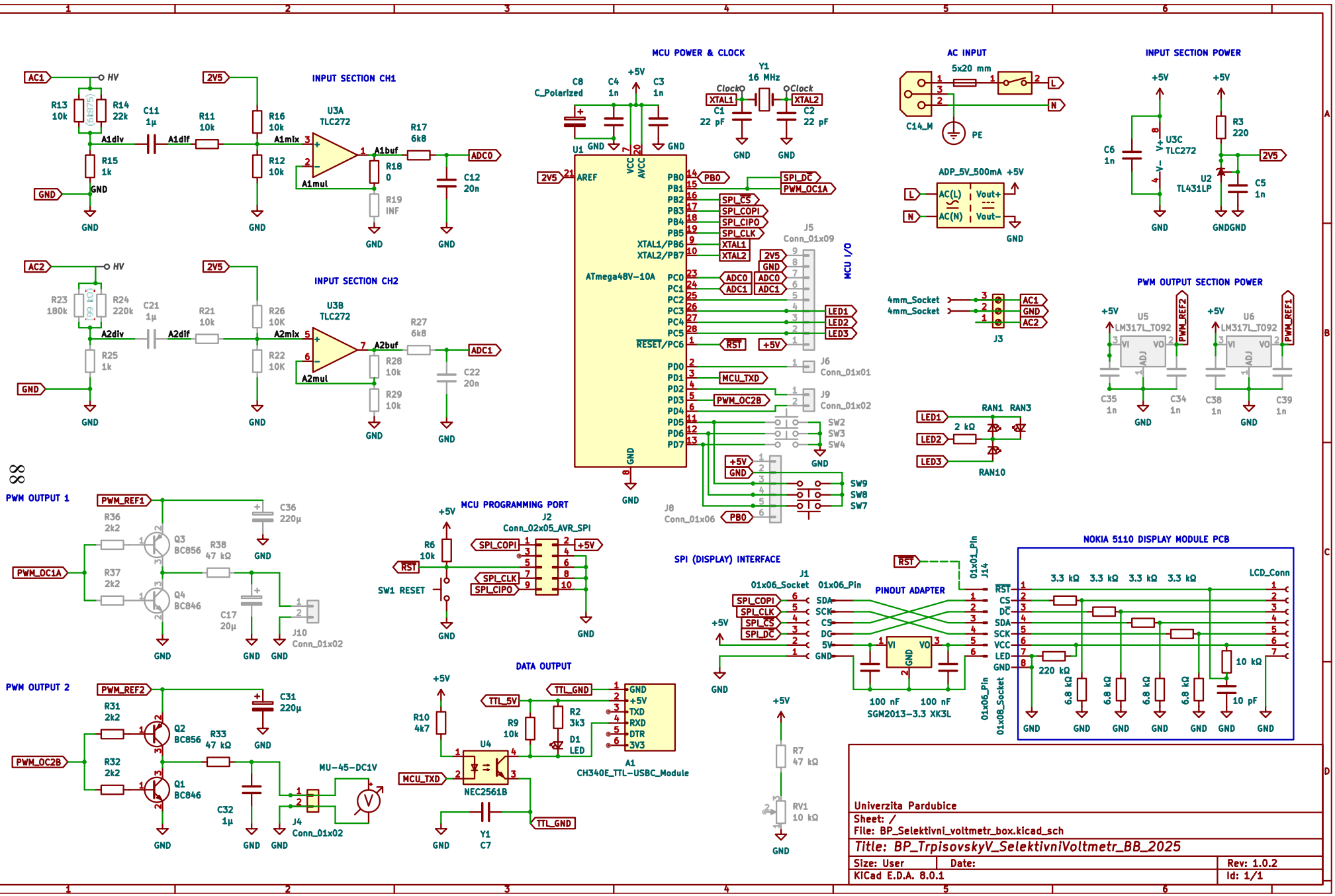
Příloha B – Rozvržení dat v části RAM řadiče displeje

Adresa	Bity/pixely MSB 7	6	5	4	3	2	1	LSB 0	Hex
168	0	0	0	0	0	0	0	0	00
169	0	0	0	0	0	0	0	0	00
170	0	0	0	0	0	0	0	1	01
171	0	0	0	0	0	0	1	0	02
172	0	0	0	0	0	1	0	0	04
173	0	0	0	0	1	0	0	1	09
174	0	0	0	1	0	0	1	0	12
175	0	0	0	1	0	1	0	0	14
176	0	0	1	0	1	0	0	1	29
177	0	0	1	0	1	0	1	0	2A
178	0	1	0	0	1	0	1	0	4A
179	0	1	0	1	0	1	0	0	54
180	0	1	0	1	0	1	0	1	55
181	0	1	0	1	0	1	0	1	55
182	1	1	1	1	1	1	1	1	FF
183	0	1	0	1	0	1	0	1	55
184	0	1	0	1	0	1	0	1	55
185	0	1	0	1	0	1	0	0	54
186	0	1	0	0	1	0	1	0	4A
187	0	0	1	0	1	0	1	0	2A
188	0	0	1	0	1	0	0	0	28
189	0	0	0	1	0	1	0	0	14
190	0	0	0	1	0	0	1	1	13
191	0	0	0	0	1	0	0	0	08
192	0	0	0	0	0	0	0	0	00
193	0	0	0	0	0	0	0	0	00
194	0	0	0	0	0	0	1	1	03
195	0	0	0	0	0	0	1	0	02
196	0	0	0	0	0	0	1	0	02
197	0	0	0	0	0	0	1	0	02
198	0	0	0	0	0	0	0	0	00
199	0	0	0	0	0	0	1	1	03
200	0	0	0	0	0	0	0	0	00
201	0	0	0	0	0	0	0	0	00
202	0	0	0	0	0	0	0	0	00
203	0	0	0	0	0	0	0	1	01
204	0	0	0	0	0	0	1	0	02
205	0	0	0	0	0	0	1	0	02
206	0	0	0	0	0	0	0	1	01
207	0	0	0	0	0	0	0	0	00
208	0	0	0	0	0	0	1	1	03
209	0	0	0	0	0	0	0	0	00
210	0	0	0	0	0	0	0	0	00
211	0	0	0	0	0	0	0	0	00
212	0	0	0	0	0	0	0	1	01
213	0	0	0	0	0	0	1	0	02
214	0	0	0	0	0	0	1	0	02
215	0	0	0	0	0	0	0	1	01
216	0	0	0	0	0	0	0	0	00
217	0	0	0	0	0	0	1	1	03
218	0	0	0	0	0	0	1	0	02
219	0	0	0	0	0	0	1	0	02
220	0	0	0	0	0	0	1	0	02
221	0	0	0	0	0	0	0	0	00
222	0	0	0	0	0	0	0	0	00
223	0	0	0	0	0	0	0	0	00
224	0	0	0	0	0	0	0	0	00
225	0	0	0	0	0	0	0	0	00
226	0	0	0	0	0	0	0	0	00
227	0	0	0	0	0	0	0	0	00
228	0	0	0	0	0	0	0	0	00
229	0	0	0	0	0	0	0	0	00
230	0	0	0	0	0	0	0	0	00
231	0	0	0	0	0	0	0	0	00
232	0	0	0	0	0	0	0	0	00
233	0	0	0	0	0	0	0	0	00
234	0	0	0	0	0	0	0	0	00
235	0	0	0	0	0	0	0	0	00
236	0	0	0	0	0	0	0	0	00
237	0	0	0	0	0	0	0	0	00
238	0	0	0	0	0	0	0	0	00
239	0	0	0	0	0	0	0	0	00
240	0	0	0	0	0	0	0	0	00
241	0	0	0	1	1	1	1	0	1E
242	0	0	1	1	1	0	0	0	38
243	0	1	1	1	0	1	1	1	77
244	0	1	1	1	0	1	1	1	77
245	0	1	1	1	1	0	0	0	78
246	0	1	1	1	1	1	1	1	7F
247	0	1	1	1	1	1	1	1	7F
248	0	1	0	1	0	1	1	1	57
249	0	0	1	1	1	1	1	1	3F
250	0	0	0	1	1	1	1	0	1E
251	0	0	0	0	0	0	0	0	00

Adresa	Bity/pixely MSB 7	6	5	4	3	2	1	LSB 0	Hex
84	0	0	0	0	1	0	0	0	08
85	0	1	1	1	1	1	1	0	7E
86	1	0	0	0	1	0	0	1	89
87	0	0	1	1	1	1	0	0	3C
88	1	1	0	0	1	0	1	1	CB
89	0	0	0	1	1	0	0	0	18
90	0	1	1	0	1	1	1	0	6E
91	1	0	0	0	1	0	0	1	89
92	0	0	1	1	1	1	0	0	3C
93	0	1	0	0	1	0	1	0	4A
94	1	0	0	0	1	0	0	1	89
95	1	0	0	0	1	0	1	1	8B
96	0	0	0	0	1	0	1	0	0A
97	0	0	0	0	1	1	0	0	0C
98	1	1	1	1	1	1	1	1	FF
99	0	0	0	0	1	0	0	0	08
100	0	0	0	0	1	0	0	0	08
101	1	0	0	0	0	0	0	1	81
102	0	0	0	0	0	0	0	0	00
103	0	0	0	0	0	0	0	0	00
104	0	0	0	0	0	0	0	0	00
105	0	0	0	0	0	0	0	0	00
106	1	1	1	0	1	1	1	1	EF
107	1	0	1	0	0	0	1	0	A2
108	1	0	1	0	0	1	0	1	A5
109	0	0	1	0	1	0	0	0	28
110	1	1	1	0	0	1	0	1	E5
111	1	0	1	0	1	0	1	1	AB
112	1	0	1	0	1	1	1	0	AE
113	0	0	1	0	0	0	0	1	21
114	0	0	0	0	1	1	1	1	0F
115	1	1	1	0	1	0	0	1	E9
116	0	0	0	0	0	0	0	0	00
117	0	0	0	0	1	0	0	0	08
118	0	0	0	0	0	0	0	0	00
119	1	1	1	0	0	0	0	0	E0
120	0	0	0	0	0	1	1	0	06
121	0	0	0	0	1	1	0	1	0D
122	1	1	1	0	1	0	1	0	EA
123	0	0	0	0	0	0	0	0	00
124	1	1	1	0	1	1	1	1	EF
125	1	0	1	0	0	0	0	0	A0
126	1	0	1	0	0	1	1	0	A6
127	0	1	0	0	1	1	0	1	4D
128	1	1	0	0	1	0	1	0	CA
129	0	0	1	0	0	0	0	0	20
130	0	0	1	0	1	1	1	1	2F
131	0	1	0	0	0	1	1	0	46
132	0	0	0	0	1	0	0	1	09
133	1	1	1	0	0	0	0	1	E1
134	1	0	1	0	1	1	1	1	AF
135	1	0	1	0	1	0	0	1	A9
136	0	0	1	0	0	0	0	0	20
137	0	0	0	0	1	1	1	1	0F
138	0	0	0	0	0	0	0	1	01
139	0	0	0	0	0	1	1	0	06
140	0	0	0	0	1	0	0	1	09
141	0	0	0	0	0	1	1	0	06
142	0	0	0	0	0	0	0	1	01
143	0	0	0	0	1	1	1	1	0F
144	0	0	0	0	1	0	0	1	09
145	0	0	0	0	0	1	1	0	06
146	0	0	0	0	1	1	0	1	0D
147	0	0	0	0	1	0	1	0	0A
148	0	0	0	0	0	0	0	0	00
149	0	0	0	0	0	1	1	0	06
150	0	0	0	0	1	0	0	1	09
151	0	0	0	0	1	0	0	1	09
152	0	0	0	0	1	1	1	1	0F
153	0	0	0	0	0	0	0	1	01
154	0	0	0	0	1	1	1	0	0E
155	0	0	0	0	0	0	0	0	00
156	0	0	0	0	1	1	1	1	0F
157	0	0	0	0	0	0	0	1	01
158	0	0	0	0	1	1	1	0	0E
159	1	0	0	0	0	0	0	0	80
160	1	0	0	0	1	1	1	0	8E
161	1	0	0	0	0	0	0	0	80
162	1	0	0	0	1	1	1	1	8F
163	1	0	0	0	0	1	1	0	86
164	1	0	0	0	1	0	0	1	89
165	0	0	0	1	0	0	1	1	13
166	0	0	0	0	1	1	0	0	0C
167	0	0	0	0	0	1	1	1	07

Adresa	Bity/pixely MSB 7	6	5	4	3	2	1	LSB 0	Hex
0	0	0	0	0	0	0	0	0	00
1	0	0	0	0	0	0	0	0	00
2	1	0	0	0	1	0	0	0	88
3	0	1	0	0	1	1	0	0	4C
4	0	0	0	1	1	1	1	0	1E
5	1	0	1	1	1	1	0	0	BC
6	0	0	1	1	1	0	0	0	38
7	0	1	1	1	0	0	0	0	70
8	1	1	1	0	0	1	0	0	E4
9	1	1	0	1	0	1	0	0	D4
10	1	0	0	1	0	0	1	0	92
11	0	0	1	0	1	0	1	0	2A
12	1	0	1	0	1	0	1	0	AA
13	1	0	1	0	1	0	1	0	AA
14	1	1	1	1	1	1	1	1	FF
15	1	0	1	0	1	0	1	0	AA
16	1	0	1	0	1	0	1	0	AA
17	0	0	1	0	1	0	1	0	2A
18	0	1	0	1	0	0	1	0	52
19	0	1	0	1	0	1	0	0	54
20	0	0	0	1	0	1	0	0	14
21	0	0	1	0	1	0	0	0	28
22	1	0	0	0	1	0	0	0	88
23	0	0	0	1	0	0	0	0	10
24	0	0	0	0	0	0	0	0	00
25	1	0	0	0	0	0	0	0	80
26	0	0	0	0	0	0	0	0	00
27	0	0	0	0	0	0	0	0	00
28	0	0	0	0	0	0	0	0	00
29	0	0	0	0	0	0	0	0	00
30	1	0	0	0	0	0	0	0	80
31	0	0	0	0	0	0	0	0	00
32	0	0	0	0	0	0	0	0	00
33	0	0	0	0	0	0	0	0	00
34	0	0	0	0	0	0	0	0	00
35	0	0	0	0	0	0	0	0	00
36	0	0	0	0	0	0	0	0	00
37	0	0	0	0	0	0	0	0	00
38	0	0	0	0	0	0	0	0	00
39	0	0	0	0	0	0	0	0	00
40	1	0	0	0	0	0	0	0	80
41	0	0	0	0	0	0	0	0	00
42	0	0	0	0	0	0	0	0	00
43	0	0	0	0	0	0	0	0	00
44	0	0	0	0	0	0	0	0	00
45	0	0	0	0	0	0	0	0	00
46	1	0	0	0	0	0	0	0	80
47	0	0	0	0	0	0	0	0	00
48	0	0	0	0	0	0	0	0	00
49	0	0	0	0	0	0	0	0	00
50	1	0	0	0	0	0	0	0	80
51	0	0	0	0	0	0	0	0	00
52	0	0	0	0	0	0	0	0	00
53	0	0	0	0	0	0	0	0	00
54	0	0	0	0	0	0	0	0	00
55	0	0	0	0	0	0	0	0	00
56	0	0	0	0	0	0	0	0	00
57	0	0	0	0	0	0	0	0	00
58	0	0	0	0	0	0	0	0	00
59	1	0	0	0	0	0	0	0	80
60	0	0	0	0	0	0	0	0	00
61	0	0	0	0	0	0	0	0	00
62	0	0	0	0	0	0	0	0	00
63	0	0	0	0	0	0	0	0	00
64	0	0	0	0	0	0	0	0	00
65	0	0	0	0	0	0	0	0	00
66	0	0	0	0	0	0	0	0	00
67	0	0	0	0	0	0	0	0	00
68	1	0	0	0	0	0	0	0	80
69	0	0	0	0	0	0	0	0	00
70	0	0	0	0	0	0	0	0	00
71	0	0	0	0	0	0	0	0	00
72	0	0	0	0	0	0	0	0	00
73	0	0	0	0	0	0	0	0	00
74	0	0	0	0	0	0	0	0	00
75	0	0	0	0	0	0	0	0	00
76	1	0	0	0	0	0	0	0	80
77	0	0	0	0	0	0	0	0	00
78	1	0	0	0	0	0	0	0	80
79	0	0	0	0	0	0	0	0	00
80	0	0	0	0	0	0	0	0	00
81	0	0	0	0	0	0	0	0	00
82	0	0	0	0	0	0	0	0	00
83	0	0	0	0	0	0	0	0	00

Příloha C – Schéma zařízení – následující strana


INPUT SECTION CH1
INPUT SECTION CH2
MCU POWER & CLOCK
AC INPUT
INPUT SECTION POWER
PWM OUTPUT SECTION POWER
PWM OUTPUT 1
PWM OUTPUT 2
MCU PROGRAMMING PORT
DATA OUTPUT
SPI (DISPLAY) INTERFACE
NOKIA 5110 DISPLAY MODULE PCB
MCU I/O
PINOUT ADAPTER
ATmega48V-10A
CH340E_TTL-USBC_Module
SGM2013-3.3 XK3L
Univerzita Pardubice
Sheet: /
File: BP_Selektivni_voltmetr_box.kicad_sch
Title: BP_TrpisovskyV_SelektivniVoltmetr_BB_2025
Size: User
Date:
Rev: 1.0.2
KiCad E.D.A. 8.0.1
Id: 1/1

Pozn.: Součástky s příznakem Do Not Populate (Neosazovat) jsou označeny šedě a je pro ně na desce ATMEGA DSP KIT vyhrazené místo, nejsou však součástí selektivního voltmetru a zůstaly neosazeny. Referenční kód je používán pro identifikaci součástek při osazování hlavní desky, součástky umístěné mimo ni jej nemají.

Příloha D – Fotografie provedení