ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra elektromechaniky a výkonové elektroniky

BAKALÁŘSKÁ PRÁCE

Moderní řešení digitronových hodin

Martin Vítek **2017**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin VÍTEK**
Osobní číslo:	**E14B0210P**
Studijní program:	**B2644 Aplikovaná elektrotechnika**
Studijní obor:	**Aplikovaná elektrotechnika**
Název tématu:	**Moderní řešení digitronových hodin**
Zadávající katedra:	**Katedra elektromechaniky a výkonové elektroniky**

Zásady pro vypracování:

Navrhněte konstrukci hodin s digitronovým zobrazovačem založené na moderních součástkách.

1. Využijte dostupné HW prostředky.
2. Doplňte moderními prostředky typu synchronizace času přes internet.
3. Realizujte navržené řešení prakticky.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah kvalifikační práce: **30 - 40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí bakalářské práce: **Ing. Petr Weissar, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Datum zadání bakalářské práce: **14. října 2016**

Termín odevzdání bakalářské práce: **8. června 2017**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Prof. Ing. Václav Kůs, CSc.
vedoucí katedry

V Plzni dne 14. října 2016

Abstrakt

Tato bakalářská práce se zabývá seznámením s principem funkce digitronového zobrazovače a konkrétním návrhem s praktickým řešením moderně řešených digitronových hodin, které si synchronizují čas z internetového NTP serveru. První část popisuje digitronový zobrazovač a základy počítačových sítí. Druhá část konkrétně popisuje a vysvětluje jednotlivé části navrženého zapojení digitronových hodin a návrh plošného spoje. Třetí část se zabývá popisem softwaru, který jsem naprogramoval pro řídící mikrokontrolér, který zajišťuje ovládání jednotlivých periférií, zobrazování času a synchronizaci času přes internet.

Klíčová slova

Digitronové hodiny, digitron, doutnavka, zvyšující měnič, snižující měnič, synchronizace času, ARM, multiplex, Ethernet.

Abstract

The bachelor theses presents the principles of the nixie display and specific design and a practical solution of modern nixie clock with synchronization of time from internet NTP server. First part introduces principle of nixie tubes and basics of computer networks. Second part deals with explanation of proposed design and design of the printed circuit board. Third part introduces the software for microcontroller, which is controlling all peripherals, displaying time and synchronizing time over internet.

Keywords

Nixie clock, nixie tube, neon tube, boost converter, buck converter, time synchronization, ARM, multiplexing, Ethernet.

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské/diplomové práce, je legální.

.....

podpis

V Plzni dne 6. 6. 2017 Martin Vítek

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Petrovi Weissarovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce.

Obsah

Obsah **8**

Seznam symbolů a zkratek **10**

Úvod **12**

1. Digitron **13**
1.1 Pracovní charakteristika 15
1.2 Princip doutnavého výboje 16

2 Počítačové sítě **17**
2.1 OSI model 17
2.2 Fyzická vrstva 19
2.3 Spojová vrstva 20
2.4 Síťová vrstva 20
2.5 Transportní vrstva 20
2.6 Aplikační vrstva 21

3 Konstrukce digitronových hodin **22**
3.1 Zdroj 23
3.2 Zvyšující měnič 25
3.3 Digitronový displej 28
3.3.1 Způsoby řízení 28
3.3.2 Digitronový multiplex 30
3.3.3 Doutnavkové oddělovače 32
3.4 Měření elektrických a neelektrických veličin 32
3.4.1 Měření napětí 33
3.4.2 Měření proudu na vstupu měniče 34
3.4.3 Měření proudu na výstupu měniče 34
3.4.4 Měření intenzity osvětlení 35
3.4.5 Měření teploty 36
3.5 Ethernet 37
3.5.1 ENC28J60 37
3.5.2 EEPROM s MAC adresou 39
3.6 Mikrokontrolér 40
3.7 Plošný spoj 42
3.8 Oživení 44

4 Software **46**
4.1 Hlavní smyčka 47
4.2 Nastavení generátorů hodin 48
4.3 Vnější přerušení 49
4.4 Tlačítka 49
4.5 UART 50

4.6 SPI ... 51
4.7 A/D převodník ... 52
4.8 Zvyšující měnič ... 53
4.9 Počítání času ... 54
4.10 Digitronový displej ... 55
4.10.1 BCD dekodér 4028 ... 55
4.10.2 Doutnavkové oddělovače ... 55
4.10.3 Řízení multiplexu digitronů ... 57
4.10.4 Řízení displeje ... 58
4.11 EEPROM ... 58
4.12 ENC28J60 ... 58
4.13 Síťový stack ... 59
4.13.1 Příjem dat ... 60
4.13.2 Odesílání dat ... 61
4.13.3 Zpracování paketů ... 62
4.13.4 Ethernetový rámec ... 62
4.13.5 ARP ... 63
4.13.6 IPv4 ... 64
4.13.7 ICMP ... 65
4.13.8 UDP ... 66
4.13.9 SNTP ... 67

5 Další možné funkce ... 69
5.1 DHCP klient ... 69
5.2 Webový server ... 69
5.3 Telnet ... 69
5.4 NTP klient ... 70
5.5 IPv6 ... 70
5.6 Zobrazování obecných čísel ... 70

6 Závěr ... 71

Seznam literatury a informačních zdrojů ... 72

7 Přílohy ... 1
7.1 Schémata ... 1
7.2 Fotografie ... 8
7.3 Přílohy v elektronické podobě ... 10

Seznam symbolů a zkratek

A/D převodník	převodník analogového signálu na digitální
ARP	Adres Resolution Protocol
BCD	Binary Coded Decimal
C	kapacita
CRC	cyklický redundantní součet (Cyclic Redundancy Check)
D/A převodník	převodník digitálního signálu na analogový
DFLL	Digital Frequency Locked Loop
DHCP	Dynamic Host Configuration Protocol
DRE	Data Register Empty
EEPROM	elektronicky vymazatelná paměť pouze pro čtení (Electricaly Erasable Programmable Read-Only Memory)
EIC	řadič vnějších přerušení (External Interrupt Controller)
f	frekvence
FLASH	nevolatilní elektricky programovatelná paměť s libovolným přístupem
FR4	Flame Retardant
FTP	File Transfer Protocol
GCLK	Generic Clock Controller
GPS	globální polohový systém (Global Position System)
HTTP	Hypertext Transfer Protocol
I	elektrický proud
$I_{a(avg)}$	maximální průměrný anodový proud
$I_{a(peak)}$	maximální pulzní anodový proud
ICMP	Internet Control Message Protocol
I_{DS}	maximální proud mezi drain a source MOSFET tranzistoru
I_f	maximální proud v propustném směru
I_{Lpeak}	pulzní proud indukčností
I_{max}	maximální dovolený proud
I_{OUT}	výstupní proud
IP	Internet Protocol
I_{sat}	saturační proud
ISO	Mezinárodní organizace pro normalizaci (International Organization for Standardization)
L	indukčnost
LCD	displej s tekutými krystaly (Liquid Crystal Display)
LED	svítivá dioda (Light Emitting Diode)
MAC	Media Access Control

MOSFET	unipolární tranzistor s izolovaným hradlem (Metal Oxide Semiconductor Field Effect Transistor)
NTP	síťový protokol pro synchronizaci času (Network Time Protocol)
NVM	Non-Volatile Memory
OSI	Open System Interconnection
PHY	Physical Layer
PPP	Poin-to-Point Protocol
PWM	pulzně šířková modulace (Pulse Width Modulation)
Q_G	náboj hradla MOSFET tranzistoru
R_{DC}	odpor pro stejnosměrný proud
R_{DSon}	odpor MOSFET tranzistoru v sepnutém stavu mezi drain a source
RESRDY	Resul Ready
RTC	hodiny reálného času (Real Time Clock)
SERCOM	Serial Communication Interface
SNTP	jednoduchý síťový protokol pro synchronizaci času (Simple Network Time Protocol)
SPDT	Single Pole, Double Throw
SPI	Serial Peripheral Interface
SSH	Secure Shell
SWD	Serial Wire Debug
TC	časovač/čítač (Timer/Counter)
TCP	Transmission Control Protocol
U	napětí
U_a	anodové napětí
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
U_{DS}	maximální napětí mezi drain a source u MOSFET tranzistoru
U_{ign}	zápalné napětí digitronu/doutnavky
U_{IN}	vstupní napětí
U_m	udržovací napětí
U_{OUT}	výstupní napětí
U_r	maximální napětí v závěrném směru
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VFD	vakuový fluorescentní displej (Vacuum Flurorescent Display)
η	účinnost
ΔI_L	zvlnění proudu
ΔU_{OUT}	zvlnění výstupního napětí

Úvod

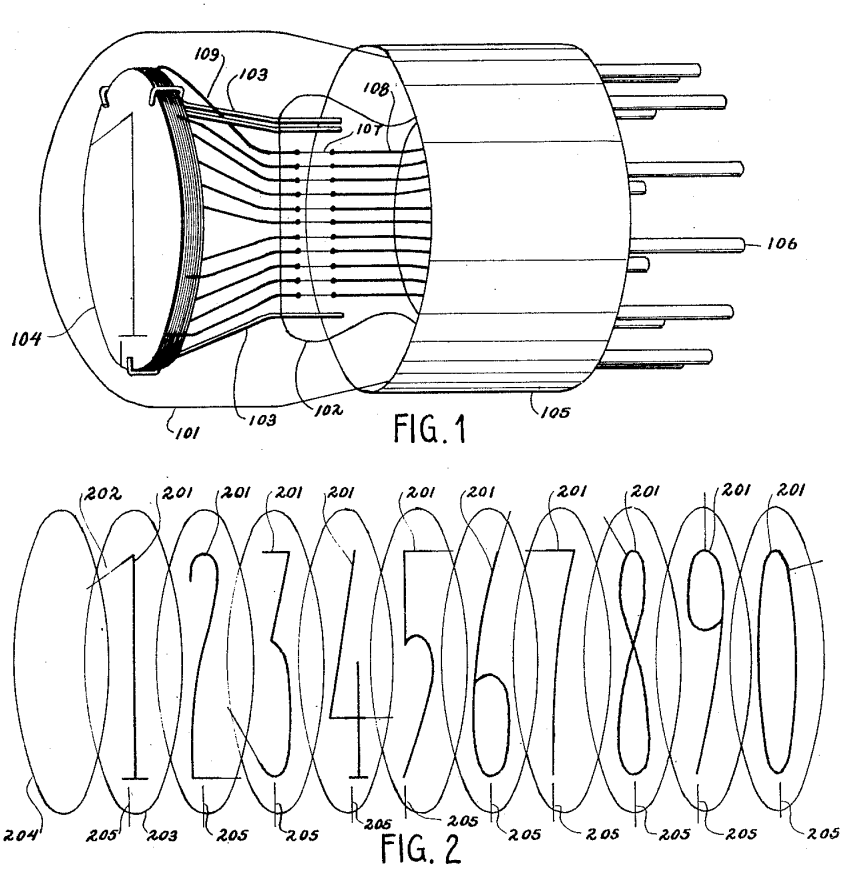
Cílem práce je navrhnout, prakticky realizovat a naprogramovat moderně řešené digitronové hodiny, které pro synchronizaci času využívají internetový NTP server.

Tradiční přístroje, které používaly digitrony jako zobrazovač, řešily jejich ovládání logickými obvody, které byly následovány specializovanými budiči pro digitrony. Ve výsledku se tato řídící logika skládala z velkého množství integrovaných obvodů, které byly navrženy jen na jednu funkci a neumožňovaly tak jednoduchou realizaci změn, které mohl zákazník v budoucnu požadovat, zabíraly velké množství místa a měly velkou spotřebu energie.

Proto jsem při návrhu mých digitronových hodin využil moderní mikrokontrolér ARM Cortex M0+ pro řízení a digitrony jsou ovládány v multiplexu pro snížení počtu ovládacích signálů. Dále hodiny obsahují zvyšující měnič, který vytváří napájecí napětí pro digitrony a ethernetový kontrolér, který zprostředkovává řídícímu mikrokontroléru přístup do internetu.

1. Digitron

Digitron je znaková výbojka plněná plynem s nízkým tlakem. Jednotlivé číslice, nebo jiné znaky, jsou tvořeny tenkým drátkem, který je vytvarovaný do požadovaného tvaru číslice nebo znaku a po připojení zápalného napětí se rozsvítí doutnavým výbojem.



Obrázek 1.1: Nákres vnitřní struktury digitronu z patentu US 2142106. [1]

První digitron byl popsán v patentu *US 2142106* „*Signaling System and Glow Lamps Therefor*“ [1] Hansem P. Boswauem v roce 1934. V tomto patentu digitron popisuje jako skleněnou baňku, ve které se nacházejí za sebou seřazené jednotlivé číslice – katody a mřížka, která je obklopuje, tvoří anodu. Baňka je naplněna neonem, případně i rtutí, argonem, sodíkem a dalšími vhodnými plyny. Při přivedení kladného napětí na anodu a záporného napětí na vybranou katodu se začne vytvářet doutnavý výboj kolem vybrané katody, který je vytvářen elektrony, které vylétají z katody a pozitivními ionty okolního plynu. Katoda se pokryje milimetrovým filmem

světla, který zaručuje čitelnost dané číslice.

První masově prodávané digitrony začala vyrábět firma Burroughs Corporation's Electronic Tube Division v roce 1954 pod označením NIX I. [2] Od tohoto označení je odvozen název nixie nebo nixie tube v anglickém a německém jazyce. V Čechách se vžil název digitron. Od této chvíle se digitrony začaly používat pro displeje měřících a podobných přístrojů, dokud nebyly nahrazeny vakuovými fluorescenčními displeji (VFD), LED displeji a LCD displeji, které mají velkou výhodu v tom, že pro provoz nepotřebují vysoké napětí jako digitrony a LCD displeje (zvláště grafické) umí zobrazovat mnohem více informací.

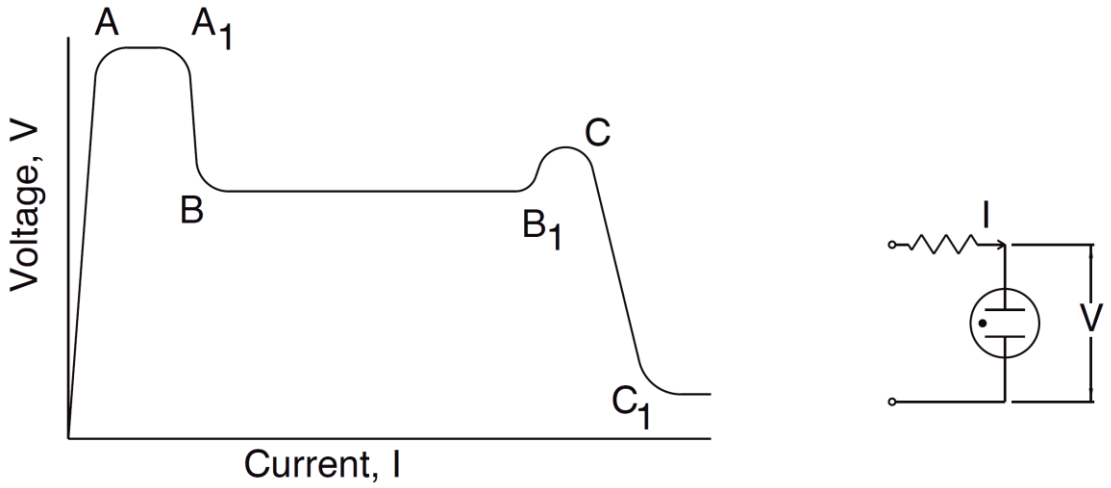
Digitrony se běžně vyráběly v různých velikostech od výšky znaku 9 mm (IN-2, IN-7) až po méně časté s výškou znaku až 50 mm (Z568M). Baňka byla buď čirá, nebo s červeným filtrem, který zvyšuje kontrast. Nejběžnější byly digitrony s číslicemi 0 až 9, které ještě případně obsahovaly desetinou tečku. Dále se vyráběly znakové digitrony, které se používaly například pro reprezentaci jednotek měřené veličiny.

Obrázek 1.3: Detail zhasnutého digitronu.

Obrázek 1.2: Detail rozsvíceného digitronu.

1.1 Pracovní charakteristika

Jelikož se digitron velmi podobá doutnavce, tak budu pracovní charakteristiku popisovat na charakteristice doutnavky.



Obrázek 1.4: Charakteristika doutnavky. [3]

Při zvyšování napětí od nuly neteče doutnavkou nebo digitronem téměř žádný proud, dokud se nepřekročí zápalné napětí U_{ign} (bod A). Zápalné napětí je určeno materiálem elektrod, plynem v baňce a tlakem plynu. V další části, od bodu B, se zapálí doutnavý výboj a napětí na digitronu/doutnavce se sníží na hodnotu udržovacího napětí U_m. Proud je v tomto okamžiku určen hlavně impedancí externího obvodu. Kdyby proud nebyl omezen, tak by došlo k obloukovému výboji mezi elektrodami, které by se tím mohly nenávratně poškodit. K omezení proudu se většinou používá předřadný rezistor. Při dalším zvyšování napájecího napětí nebo procházejícího proudu dochází k malému zvyšování udržovacího napětí až do bodu B_1. V této části charakteristiky jsou oba zobrazovače provozovány při normálním provozu. Maximální proud I_{max} je určen příslušným katalogovým listem a může být překročen při pulzním provozu, pokud střední hodnota proudu nepřekročí I_{max}. Při dalším zvyšování napětí nebo proudu nastane dříve zmiňovaný obloukový výboj, kdy napětí poklesne na hodnotu C_1 [3].

1.2 Princip doutnavého výboje

Při překročení zápalného napětí, se ze záporné elektrody začínají uvolňovat elektrony, které jsou urychlovány elektrickým polem mezi zápornou a kladnou elektrodou. Letící elektron se srazí s atomem plynu a mohou nastat dvě situace.

Zaprvé může letící elektron při srážce poskytnout energii valenčnímu elektronu plynu, který se tak posune na vyšší energetickou hladinu. Na této hladině nějaký čas setrvá a poté se vrátí na původní energetickou hladinu a přitom vyzáří foton. Vlnová délka je určena energií rozdílu energetických hladin, mezi kterými se valenční elektron pohybuje. Tento rozdíl je určen typem použitého plynu. Vlnová délka se dá vypočítat podle vztahu:

$$\lambda = \frac{hc}{E}\ [m] \tag{1.1}$$

kde λ je vlnová délka, h je Planckova konstanta, c je rychlost světla a E je energie.

Zadruhé může dojít k tomu, že letící elektron vyrazí elektron z atomu plynu a tím se z atomu plynu stane kladný iont, který je přitahován k záporné elektrodě. Při srážce iontu se zápornou elektrodou dojde k odstranění části materiálu, který pokrývá elektrody. Tento efekt je umocňován při vyšším proudu elektrodami. Při ztrátě tohoto materiálu dochází ke zvyšování zápalného napětí. U digitronů má tento jev negativní vliv na ostatní záporné elektrody, které tvoří jednotlivé číslice, protože se na nich tento materiál usazuje a časem může dojít k tomu, že se ho na částech číslic, které dlouho nesvítily, usadí tolik, že tyto části přestanou svítit. Jako protiopatření lze aplikovat metodu, kdy se v určitých intervalech postupně rozsvěcí všechny číslice, čímž je usazování částečně zabráněno. Na digitronech, které byly dlouhou dobu v provozu lze pozorovat tento usazený materiál na stěnách skleněné baňky jako jemný bílý povlak (*OBRÁZEK 1.2* a *OBRÁZEK 1.3*).

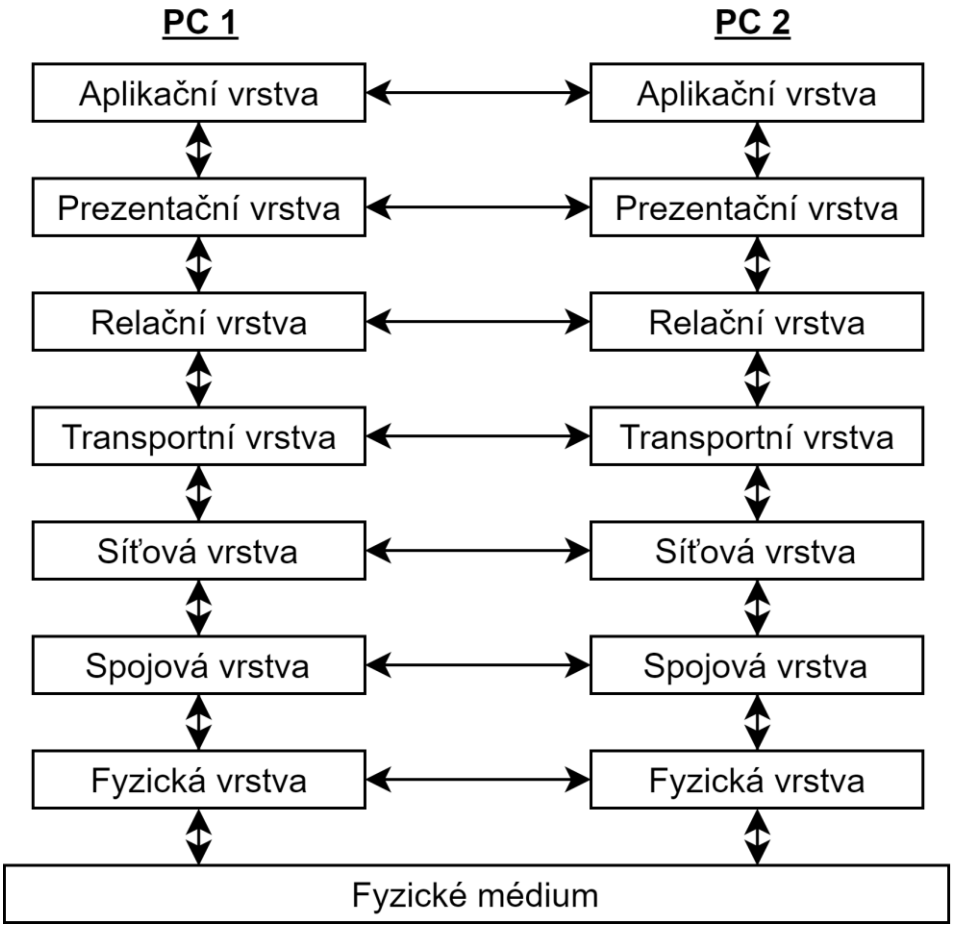
2 Počítačové sítě

Počítačové sítě jsou podstatnou součástí našeho každodenního života, hlavně s ohledem na největší celosvětovou síť, které říkáme internet. Tato kapitola popisuje základy počítačových sítí a komunikace po nich, které jsem použil pro realizaci této práce.

2.1 OSI model

Na počátku vzniku počítačů vznikala potřeba komunikace mezi jednotlivými počítači. Ze začátku používal každý výrobce své vlastní řešení, které se ještě mohlo časem dále vyvíjet. Jak se počítače vyvíjely a přibývalo jejich množství, tak se začaly objevovat problémy s tím, že tato jednotlivá proprietární řešení nebyla vzájemně kompatibilní. Jako odpověď na tento problém vydala v roce 1984 Mezinárodní standardizační organizace normu ISO 7498 *Information technology – Open System Interconnection – Basic Reference Model: The Basic Model* [4], ve které je popsán základní model, který se dodnes používá jako základ pro komunikaci v počítačových sítích.

OSI model se skládá ze sedmi vrstev (*OBRÁZEK 2.1*), které mezi sebou vzájemně komunikují. Komunikace mezi jednotlivými vrstvami probíhá tak, že požadovaná vrstva, například vrstva transportní, předá svá data do vrstvy, která se nachází pod ní, tedy do vrstvy síťové a dále se o ně nezajímá. U příjemce naopak síťová vrstva předá data vrstvě transportní, pro kterou to vypadá, že tato data procházela jen přes síťovou vrstvu a nezajímá se o spodnější vrstvy. V tomto okamžiku se může bez problému změnit typ fyzické nebo spojové vrstvy, protože vrstva transportní a vyšší neřeší tyto spodní vrstvy.

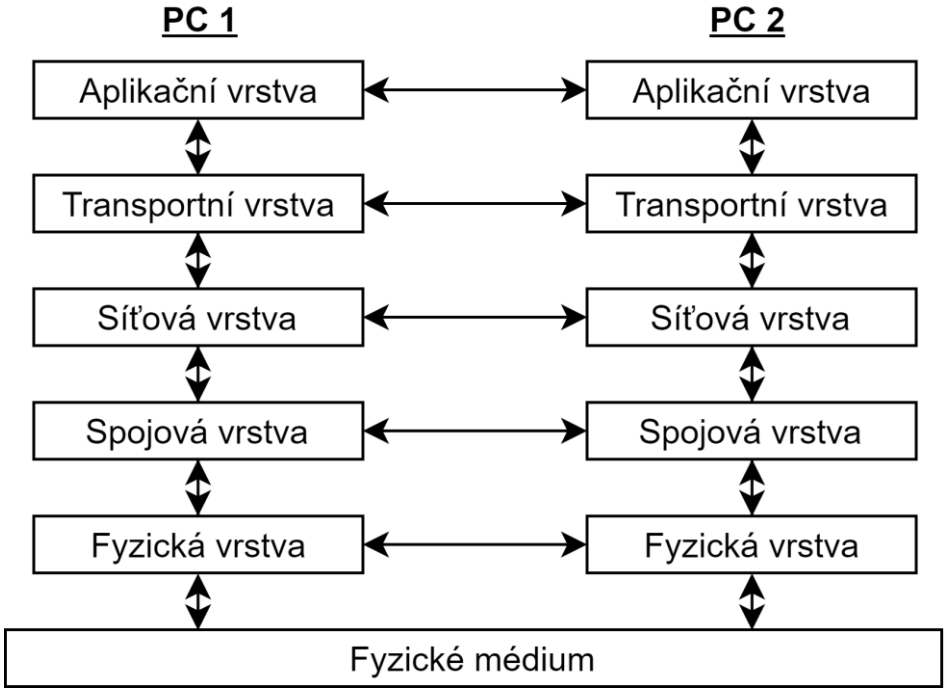


Obrázek 2.1: OSI model.

Tento princip si můžeme přiblížit třeba na příkladu webového prohlížeče, který bude korektně fungovat v případě, kdy je počítač připojen do počítačové sítě přes 1000Base-T Ethernet čtyřmi kroucenými páry UTP kabelu, nebo jestli je připojen bezdrátově přes WiFi.

Při dodržení tohoto modelu tedy vzniká modulární systém, který je ale vzájemně kompatibilní a dává tak možnost používat vhodné prostředky, které vyhovují požadavkům dané aplikace. Tyto prostředky jsou hlavně komunikační protokoly jednotlivých vrstev, přestože tento model neřeší komunikační protokoly, ale jen architekturu vzájemné komunikace počítačů.

Pro běžnou komunikaci, která probíhá po TCP/IP se nepoužívají všechny vrstvy OSI modelu, protože některé vrstvy by přinášely zbytečnou zátěž. Zjednodušený TCP/IP model znázorňuje *OBRÁZEK* 2.2. Jednotlivé vrstvy tohoto modelu popisuji dále v následujících kapitolách, protože jsou důležité pro tuto práci.



Obrázek 2.2: TCP/IP model.

2.2 Fyzická vrstva

Fyzická vrstva zajištuje fyzické propojení jednotlivých počítačů nebo síťových prvků, tedy zajišťuje konverzi datových bitů na fyzické signály (elektrické, optické). Popisuje fyzické, elektrické a logické parametry přenosového kanálu. Samotné fyzické médium není součástí této vrstvy. Řeší se navazování a ukončování fyzického spojení, kolize, jednosměrnost nebo obousměrnost provozu a případné paralelní posílání dat.

Mezi fyzické vrstvy například patří:

- 10Base-T 10Mb/s Ethernet po telefonní kroucené dvojlince

- 100Base-TX 100Mb/s Ethernet po dvou kroucených párech Cat5
- 100Base-FX 100Mb/s Ethernet po dvou vícevidových optických vláknech
- WiFi
- RS-232
- CAN bus

2.3 Spojová vrstva

Spojová vrstva zajišťuje navazování a ukončování spojení, adresaci jednotlivých bodů sítě na základě MAC adres, detekci chyb (kontrola kontrolního součtu přenášeného rámce), opakování přenosu, pokud dojde k chybě. Spojuje systémy, které se nacházejí ve stejné síti. Příklady některých spojových vrstev:

- Ethernet
- Token Ring
- PPP (Point-to-Point Protocol)

2.4 Síťová vrstva

Síťová vrstva spojuje a adresuje systémy, které se nenacházejí ve stejném fyzickém segmentu sítě. Příklady síťových vrstev:

- IP (Internet Protocol)
- IPsec (IP security)
- ARP (Address Resolution Protocol)

2.5 Transportní vrstva

Transportní vrstva zajišťuje spolehlivost a kvalitu přenosu tak, jak to vyžaduje aplikační vrstva. Může poskytovat buď spojově orientovanou službu, která navazuje a udržuje spojení a hlídá, jestli byl přenos úspěšný (TCP) nebo poskytuje nespojovou

službu, která nehlídá přenos a nezajišťuje tak úspěch přenosu (UDP).

Příklady transportních vrstev:

- UDP (User Datagram Protocol)
- TCP (Transmission Control Protocol)
- ICMP (Internet Controll Message Protocol)

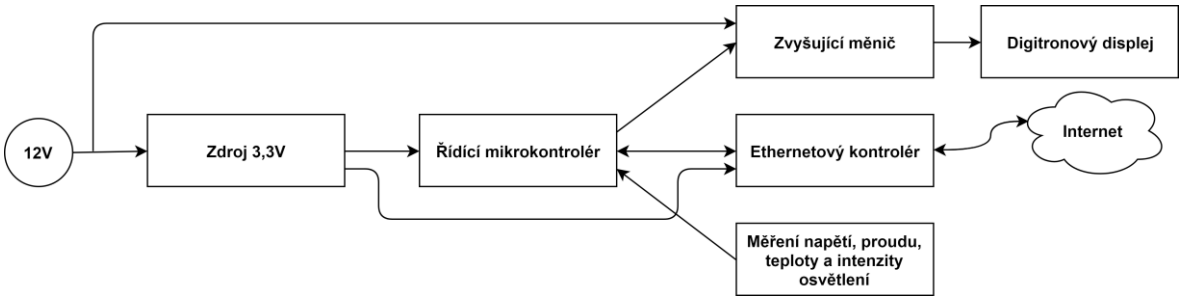
2.6 Aplikační vrstva

Aplikační vrstva poskytuje uživatelské rozhraní k síti pro aplikace, které jsou spuštěny na počítači. Jednotlivé aplikace mají přiřazeno číslo portu, které je identifikuje na transportní vrstvě. Příklady aplikací na aplikační vrstvě:

- HTTP (Hyper Text Ttransport Protocol)
- FTP (File Transfer Protocol)
- NTP (Network Time Protocol)

3 Konstrukce digitronových hodin

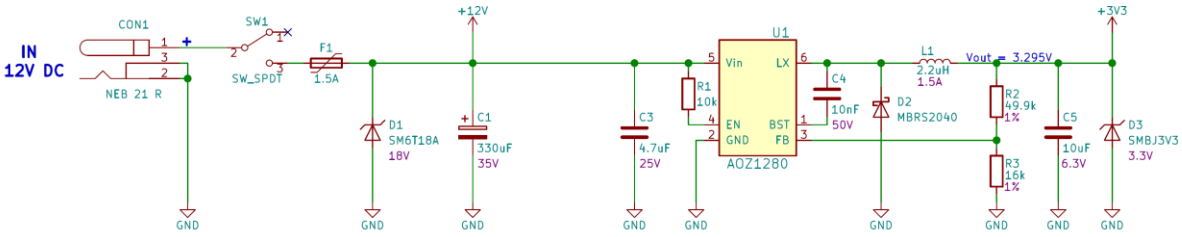
Před samotným návrhem zapojení digitronových hodin jsem si dle zadání této práce určil požadavky, které by konstrukce měla splňovat. Pro zobrazování času tedy budou použity čtyři nebo šest digitronů, které budou řízeny mikrokontrolérem s jádrem AMR Cortex-M. Pro napájení by mělo být využito bezpečné a běžně dostupné napětí 12V s čímž souvisí zvyšující měnič, který bude vytvářet 180V pro napájení digitronů. Tento zvyšující měnič by mohl mít mikrokontrolérem nastavitelné výstupní napětí, aby se mohl přizpůsobit různým typům digitronů. Pro připojení do počítačové sítě a internetu bude využita technologie Ethernet s kabelovým připojením UTP kabelem. Plošný spoj by měl být navržen tak, aby vznikla kompaktní konstrukce, která nebude zbytečně plýtvat místem a bude jednoduchá na obsluhu uživatelem. Ovládací prvky nemají velkou prioritu, protože hodiny si budou synchronizovat čas přes internet. Jednotlivé součástky by měly být co nejekonomičtější, aby byla výsledná cena zařízení co nejmenší, ale aby zároveň neutrpěla kvalita nebo funkce. Blokové schéma znázorňuje *OBRÁZEK 3.1* a jednotlivé bloky jsou podrobně popsány v následujících kapitolách.



Obrázek 3.1: Blokové schéma digitronových hodin.

3.1 Zdroj

Celkem jsou v zařízení potřeba tři napětí. Jako hlavní napájecí napětí jsem zvolil 12V. Toto napětí je využito pro zvyšující měnič, který vytváří napájecí napětí pro digitrony a pro snižující měnič, který vytváří napájecí napětí 3,3V pro mikrokontrolér a Ethernetový kontrolér.



Obrázek 3.2: Schéma napájecího zdroje.

Napájecí napětí je do zařízení přivedeno přes běžný napájecí konektor *CON1* typu NEB 21 R [5], který má vnější průměr 5,5mm a vnitřní průměr 2,1mm. Kladný pól je ve středu, plášť tvoří pól záporný. Konektor je dimenzován na stejnosměrné napětí 24V a proud 2A.

Dále následuje přepínač *SW1 typu* SPDT, ze kterého jsou ale využity jen dva kontakty a slouží tak jako hlavní vypínač celého zařízení. Tento přepínač je dimenzován na stejnosměrné napětí 28V a proud 6A [6]. Za ním je vratná polymerová pojistka *F1* typu SL150-33 [7] s jmenovitým proudem 1,5A která chrání zařízení před zkratem a společně s jednosměrným transilem *D1* tvoří ochranu proti přepětí a proti přepólování napájecího napětí.

Pro vytvoření 3,3V jsem uvažoval nad integrovaným lineárním stabilizátorem, například typem LF33, ale vzhledem k relativně velkému proudu (300mA), který vyžaduje ethernetový kontrolér a ostatní spotřebiče, jsem tuto variantu zavrhl kvůli velkému množství tepla, které by se na něm muselo mařit.

$$P_{loss} = (U_{IN} - U_{OUT}) \cdot I_{OUT} = (12 - 3{,}3) \cdot 0{,}3 = 2{,}61\text{W} \tag{3.1}$$

Vybral jsem tedy integrovaný pulzní snižovací měnič AOZ1280 [8] v pouzdře SOT23-6L, který má rozsah napájecího napětí 3V až 26V, výstupní proud až 1,2A, pracovní kmitočet 1,5MHz a účinnost při U_{IN} = 12V a U_{OUT} = 3,3V by se měla podle katalogového listu pohybovat kolem 85%. Tento měnič obsahuje ochrany proti nadproudu, nízkému vstupnímu napětí a proti přehřátí. Zapojení je převzato z doporučeného zapojení, které uvádí výrobce v katalogovém listu. Pin *EN* (enable) nevyužívám a tak je připojen přes pull-up rezistor *R1* na napájení a měnič je stále zapnutý. Výstupní napětí je určeno odporovým děličem tvořeným rezistory *R2* a *R3* a vypočítá se dle následujícího vzorce:

$$U_{OUT} = 0{,}8 \cdot \left(1 + \frac{R_2}{R_3}\right) = 0{,}8 \cdot \left(1 + \frac{49\,900}{16\,000}\right) = 3{,}295\text{V} \tag{3.2}$$

Cívku jsem použil s výrobcem doporučenou hodnotou indukčnosti 2,2µH. Cívku jsem vybíral tak, aby měla vyšší saturační proud, než je špičkový proud, který jsem vypočítal podle následujících vzorců:

$$\Delta I_L = \frac{U_{OUT}}{f \cdot L} \cdot \left(1 - \frac{U_{OUT}}{U_{IN}}\right) = \frac{3{,}3}{1{,}5 \cdot 10^6 \cdot 2{,}2 \cdot 10^{-6}} \cdot \left(1 - \frac{3{,}3}{12}\right) = 0{,}725\text{A} \tag{3.3}$$

$$I_{Lpeak} = I_{OUT} + \frac{\Delta I_L}{2} = 0{,}5 + \frac{0{,}725}{2} = 0{,}863A \tag{3.4}$$

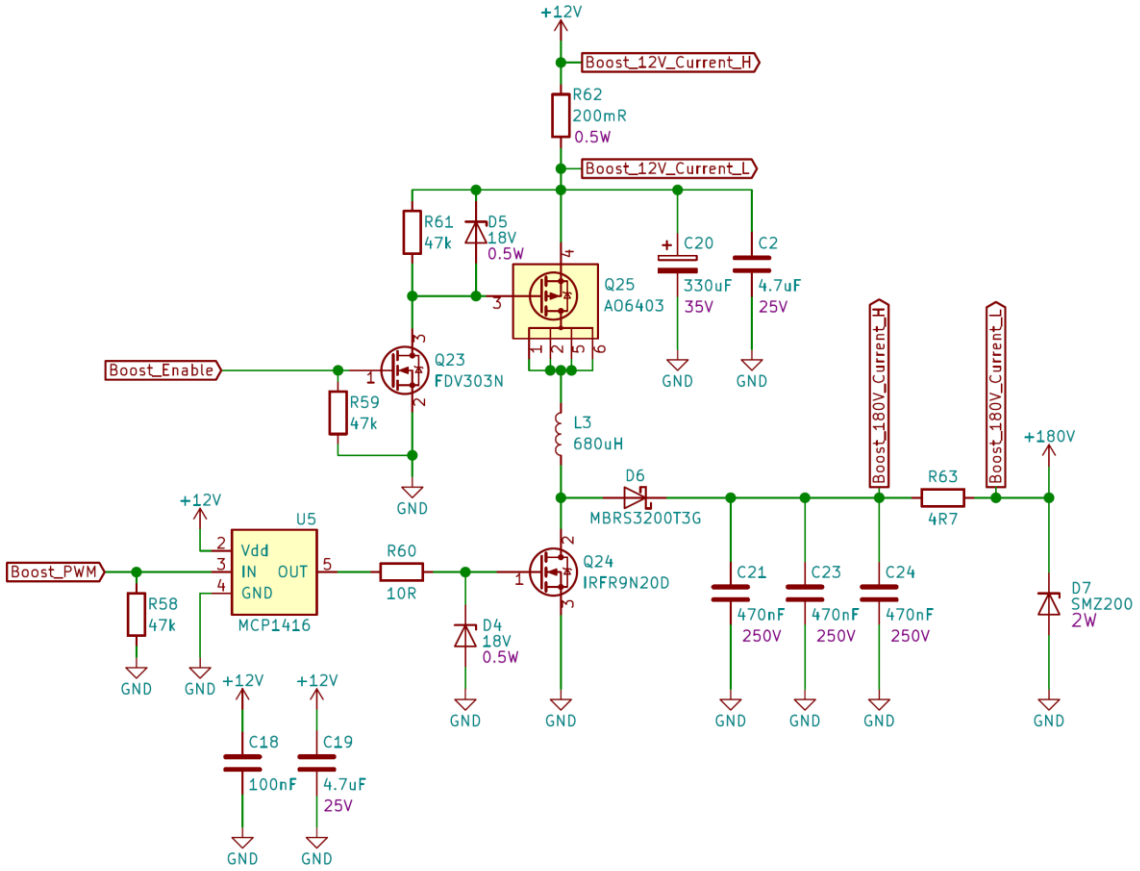
kde ΔI_L je zvlnění proudu induktorem, f je spínací frekvence, L je indukčnost induktoru, I_{Lpeak} je špičkový proud induktorem a I_{OUT} je maximální předpokládaný výstupní proud. Podle vypočítaného špičkového proudu jsem tedy vybral cívku se saturačním proudem 1,55A a odporem 76mΩ v SMD pouzdře 1210 [9].

Jako volnoběžnou diodu jsem zvolil Schottkyho diodu MBRS2040 která má závěrné napětí 40V a proud v propustném směru 2A. Na výstupu měniče je dále

připojen 3,3V transil, který chrání zbytek zapojení před přepětím, které by mohlo nastat při poruše měniče.

3.2 Zvyšující měnič

Digitrony potřebují pro zapálení doutnavého výboje napájecí napětí minimálně 170V (může se lišit podle typu digitronu). Klasické řešení využívalo usměrněné síťové napětí nebo měl hlavní transformátor zařízení zvláštní vinutí, které bylo určené pro napájení digitronů. Jelikož jeden z mých základních požadavků je napájení bezpečným napětím 12V, tak jsem navrhnul jednoduchý zvyšující měnič.



Obrázek 3.3: Schéma zvyšujícího měniče.

Měnič je napájen z hlavního 12V napájení, které pokračuje přes rezistor *R62*, na kterém se měří proud vtékající do měniče. Dále následuje MOSFET tranzistor *Q25* s kanálem typu P, kterým se zapíná a vypíná napájecí napětí pro měnič. Oproti snižujícímu měniči, kdy pro odpojení jeho výstupu stačí pouze zavřít spínací

tranzistor, u zvyšujícího měniče zavření spínacího tranzistoru nestačí, protože na výstupu je stále napájecí napětí měniče. Pokud by na výstupu nastal zkrat, tak mikrokontrolér nemá žádnou možnost, jak na danou situaci reagovat a zareagovala by až vstupní pojistka *F1*, která by ale odpojila celé zařízení. Tranzistor *Q25* je spínán z mikrokontroléru přes pomocný MOSFET tranzistor *Q23*.

Parametry měniče jsem vypočítal podle aplikační zprávy *Basic Calculation of a Boost Converter's Power Stage* [10]. Pro výpočty je potřeba znát:

- Vstupního napětí U_{IN} = 10V
- Výstupní napětí U_{OUT} = 180V
- Zvlnění výstupního napětí ΔU_{OUT} = 0,1V
- Maximální výstupní proud I_{OUT} = 20mA
- Předpokládaná účinnost η = 0,7
- Spínací frekvence f = 187,5kHz

Hodnotu indukčnosti cívky *L3* jsem vypočítal podle přepokládaného zvlnění proudu:

$$\Delta I_L = 0{,}3 \cdot I_{OUT} \cdot \frac{U_{OUT}}{U_{IN}} = 0{,}3 \cdot 0{,}02 \cdot \frac{180}{12} = 0{,}09\text{A} \tag{3.5}$$

$$L = \frac{U_{IN} \cdot (U_{OUT} - U_{IN})}{\Delta I_L \cdot f \cdot U_{OUT}} = \frac{12 \cdot (180 - 12)}{0{,}09 \cdot 187500 \cdot 180} = 663{,}7\mu\text{H} \tag{3.6}$$

Dále jsem vypočítal maximální proud spínacím tranzistorem, který se vypočítá ze střídy řídícího PWM signálu a zvlnění proudu:

$$D = 1 - \frac{U_{IN} \cdot \eta}{U_{OUT}} = 1 - \frac{12 \cdot 0{,}7}{180} = 0{,}953 \tag{3.7}$$

$$I_{SW(max)} = \frac{\Delta I_L}{2} + \frac{I_{OUT}}{1 - D} = \frac{0{,}09}{2} + \frac{0{,}02}{1 - 0{,}953} = 0{,}471\text{A} \tag{3.8}$$

Z vypočítaných hodnot jsem zvolil cívku L3 s parametry L = 680µH; I_{SAT} = 0,95A; R_{DC} = 1,1Ω, diodu *D6* s parametry U_r = 200V; I_F = 3A, spínací MOSFET tranzistor *Q24* s parametry U_{DS} = 200V; I_{DS} = 9,4A; $R_{DS(on)}$ = 0,38Ω; Q_G = 18nC.

Hodnotu kapacity výstupního kondenzátoru jsem určil podle následujícího vztahu:

$$C = \frac{I_{OUT} \cdot D}{f \cdot \Delta U_{OUT}} = \frac{0{,}02 \cdot 0{,}953}{187500 \cdot 0{,}1} = 1{,}017\mu\text{F} \tag{3.9}$$

Použil jsem tedy tři keramické kondenzátory (*C21*, *C23*, *C24*) s parametry C = 470nF; U = 250V, které jsou zapojeny paralelně s výslednou kapacitou 1,41µF. Jako ochrana proti přepětí na výstupu slouží zenerova dioda *D7* se zenerovým napětím 200V. Výstupní proud z měniče je měřen na rezistoru *R63*.

Spínací tranzistor *Q24* je buzen z integrovaného budiče *U5*, který je schopný špičkově dodávat až 1,5A. Buzení hradla MOSFET tranzistoru vyžaduje poměrně vysoké proudy, které nabíjí a vybíjí kapacitu hradla Q_G, aby bylo zaručeno rychlé otevření a zavření tranzistoru a minimalizovaly se spínací ztráty. Kondenzátory C18 a *C19* slouží jako blokovací kondenzátory pro tento budič. Rezistor *R58* zabraňuje tomu, aby byl tranzistor *Q24* otevřený v době, kdy se inicializuje mikrokontrolér a všechny vstupně výstupní piny jsou nastaveny jako vstupy a jejich napěťová úroveň není definována. Rezistor *R60* omezuje strmost proudu to hradla spínacího tranzistoru. Zenerova dioda *D4* chrání hradlo před přepětím.

Měnič je z mikrokontroléru řízen dvěma signály. *Boost_Enable* řídí tranzistor

Q25, který zapíná a vypíná napájecí napětí pro měnič a *Boost_PWM*, který řídí spínací tranzistor *Q24*. Zpětná vazba o výstupním napětí je zpracovávána přes odporový dělič A/D převodníkem mikrokontroléru.

3.3 Digitronový displej

Displej se skládá z šesti digitronů a dvou doutnavek, které tvoří oddělovače. Je tedy možné zobrazit čas ve formátu HH:MM:SS, datum ve formátu DD:MM:RR nebo obecné šesticiferné číslo.

Použil jsem digitrony Philips ZM1000 [11], které mají výšku znaku 14mm v čiré baňce s pohledem z boku. Zvláštností těchto digitronů je to, že mají kontakty se standardní roztečí 2,54mm a jsou určeny do patice. Patice má výhodu v možnosti snadné výměny vadného digitronu a případně by bylo možné vyrobit redukci pro jiný typ digitronů, který má jiné rozložení pinů, která by se zasunula do této patice. Tyto digitrony obsahují standardní číslice 0 až 9 a desetinnou tečku vlevo od číslic. Desetinnou tečku jsem ale nevyužil. Parametry těchto digitronů jsou U_{ign} = 170V; $I_{a(avg)}$ = 2,5mA; $I_{a(peak)}$ = 12mA. Další zajímavost těchto digitronů je primer, což je připojení na anodu, které je vyvedeno na jeden z pinů. Mezi primer a nulový potenciál se připojuje rezistor o hodnotě 10MΩ, který zajišťuje rychlejší ionizaci plynu uvnitř digitronu a zajišťuje tedy rychlejší rozsvěcování číslic, což je výhoda, pokud jsou digitrony použity v aplikaci, kde se rychle přepínají, například v multiplexu.

3.3.1 Způsoby řízení

Digitrony mohou být řízeny přímo, kdy je každá katoda každého digitronu připojena přes tranzistor k mikrokontroléru. V mém případě, kdy mám šest digitronů, každý s deseti číslicemi, bych potřeboval 60 pinů mikrokontroléru, které by byly vyhrazeny pro jejich řízení. Tento způsob je velmi jednoduchý, ale velkou nevýhodou je velké množství ovládacích signálů, které vyžadují mikrokontrolér

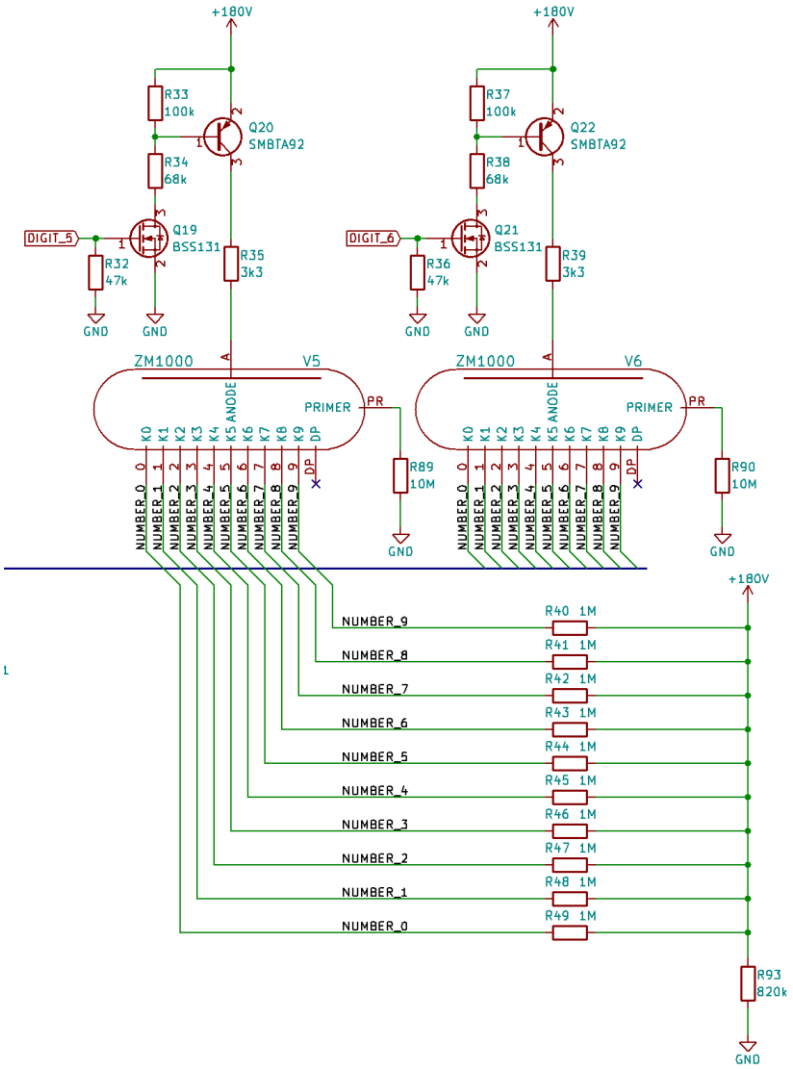
s velkým množstvím pinů a velké množství cest na plošném spoji.

Proto se původně používaly speciální integrované obvody MH74141, které kombinují převodník BCD kódu na kód 1 z deseti s výstupními tranzistory určenými pro spínání katod digitronů. Počet ovládacích signálů se tak u každého digitronu zmenšil z deseti na čtyři a tyto integrované obvody byly jednoduše připojitelné na zbytek číslicových obvodů. Nezvolil jsem tento způsob řízení z důvodu horší dostupnosti těchto obvodů, které se už nevyrábějí a zabíraly by zbytečně moc místa na plošném spoji.

Další možností je použití posuvných registrů, které mají vysokonapěťové výstupy, například TPIC6B595. Jedná se o 8 bitový posuvný registr se sériovým vstupem, který má na výstupech 50V zenerovy diody. V mém případě bych tedy potřeboval osm těchto posuvných registrů a řídící signály by byly jenom tři – data a dva hodinové signály. Nevýhodou tohoto řešení je cena těchto posuvných registrů a potřeba nemalého místa na plošném spoji.

Zvolil jsem tedy řízení v režimu multiplexu, kdy jsou všechny odpovídající katody stejných číslic spojeny, a navíc se ovládá napájení pro anody jednotlivých digitronů. Vždy tedy svítí číslice jen na jednom digitronu a postupně se požadovaná číslice zobrazuje na jednotlivých digitronech a to tak rychle, že pro vnímání lidského oka svítí všechny číslice na všech digitronech. Pro řízení je tedy v mém případě potřeba 10 katodových vodičů a 6 anodových. Nevýhodou tohoto řešení je, že se zmenší jas digitronů, protože svítí jen 1/6 zobrazovacího cyklu a je tedy zapotřebí zvýšit proud digitronem. Řízení anod digitronů popíši jen na digitronu *V6*, na ostatních je zapojení identické.

3.3.2 Digitronový multiplex



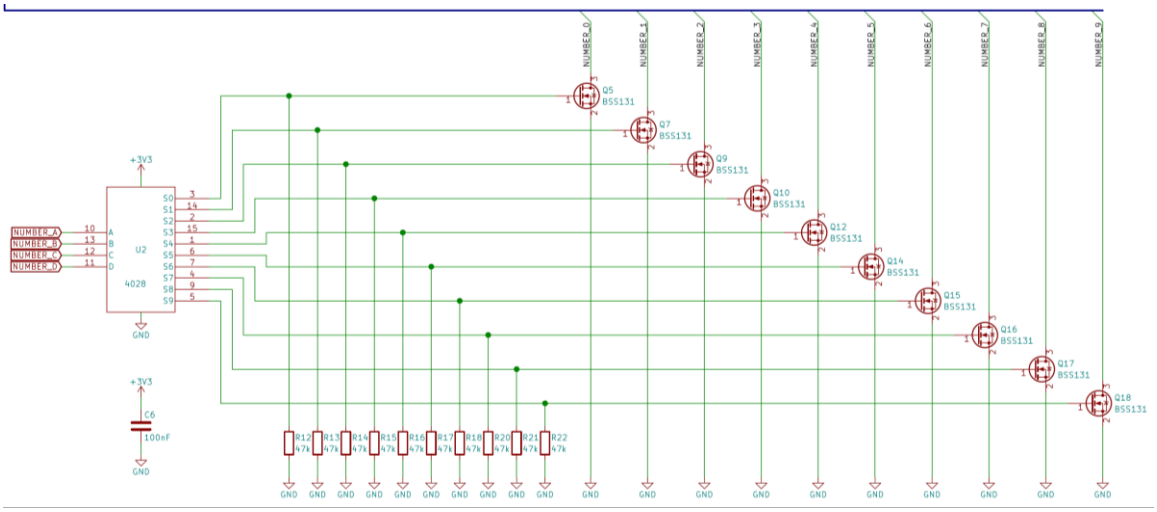
Obrázek 3.4: Schéma spínání anod digitronů v multiplexu.

Anodové napětí je spínáno PNP tranzistorem *Q22*, který je spínán signálem *DIGIT_6* z mikrokontroléru přes MOSFET tranzistor *Q21*. Rezistor *R39* omezuje proud digitronem a jeho hodnotu jsem nejdříve přibližně vypočítal a při oživování jsem ji experimentálně vybral tak, aby digitronem protékal takový proud, který v použitém multiplexovém režimu zajistí dostatečný jas číslic a zároveň se nesmí překročit $I_{a(peak)}$. Zvolil jsem tedy hodnotu 3,3kΩ, se kterou digitronem protéká 7mA. Mezi primer a nulový potenciál je připojen rezistor *R90*, který zajišťuje rychlejší ionizaci. Aby nedocházelo k částečnému svitu neaktivních číslic, tak by na vypnutých katodách mělo být takové napětí, aby mezi těmito katodami a anodou bylo napětí menší, než je U_{ign} a zároveň nesmí být tak vysoké, aby docházelo k tomu, že se tyto

katody začnou pro aktivní katodu chovat jako anody. Tuto podmínku měl splňovat napěťový dělič tvořený rezistory *R93* a *R40* až *R49*. Výsledné napětí na vypnuté katodě by tedy mělo být:

$$U_{k(off)} = U_a \frac{R40}{R93 + R40} = 180 \frac{82 \cdot 10^4}{10^6 + 82 \cdot 10^4} = 81{,}1\text{V} \qquad (3.10)$$

Zde jsem ale udělal chybu a tento napěťový dělič jsem navrhnul špatně. Mnou navržené zapojení se tedy chová tak, že na vypnutých katodách je plné napětí U_a, což způsobovalo, že i když měl digitron vypnutou anodu, tak anodové napětí poskytovaly uvnitř digitronu vypnuté katody, což se projevovalo částečným svitem číslice, která se zobrazovala na vybraném digitronu, i na ostatních digitronech a byla tím snížena čitelnost displeje. Jako částečné řešení jsem na zařízení odstranil rezistory *R40* až *R49*, čímž většina nechtěného svitu číslic zmizela. Pro úplné odstranění problému by bylo nutné udělat poměrně rozsáhlé změny na plošném spoji tak, aby byl napěťový dělič funkční a na neaktivních katodách by bylo přibližně 80V.

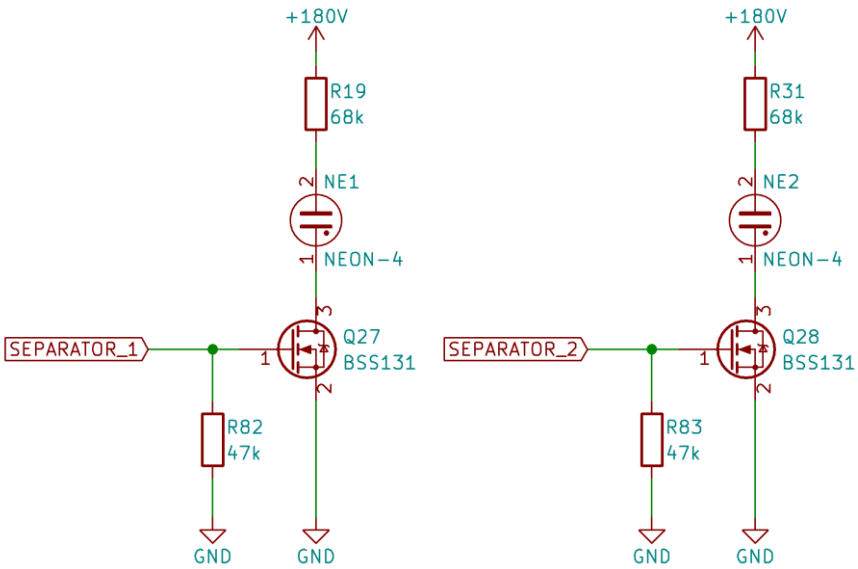


Obrázek 3.5: Schéma spínání katod digitronů v multiplexu.

Abych redukoval počet vodičů nutných pro spínání katod, tak jsem použil 4 bitový dekodér BCD kódu na desítkový výstup 1 z deseti *U2*, který ideálně odpovídá mým požadavkům. Počet vodičů se díky tomuto obvodu snížil z 10 na 4. Zobrazená číslice se ovládá signály *NUMBER_A*, *NUMBER_B*, *NUMBER_C* a *NUMBER_D*. Kondenzátor *C6* slouží jako blokovací pro tento dekodér. Katody jsou spínány

MOSFET tranzistory *Q5*, *Q7*, *Q9*, *Q10*, *Q12*, *Q14*, *Q15*, *Q16*, *Q17* a *Q18*.

3.3.3 Doutnavkové oddělovače



Obrázek 3.6: Schéma zapojení doutnavek.

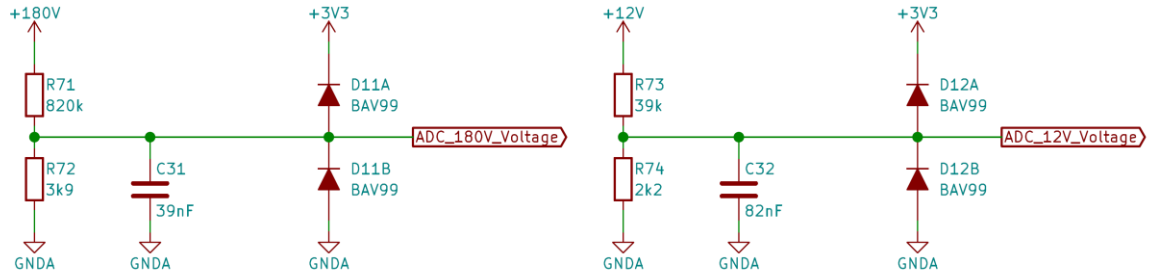
Doutnavky *NE1* a *NE2* jsou typ NEON-4 s pohledem z boku, s průměrem 5mm a výškou 12mm. Doutnavý výboj má světle oranžovou barvu. Elektrické parametry jsou U_{ign} = 135V; I = 0,9mA. Rezistory *R19* a *R31* slouží pro omezení proudu doutnavkami. MOSFET tranzistory *Q27* a *Q28* spínají doutnavky a jsou ovládány z mikrokontroléru signály *SEPARATOR_1* a *SEPARATOR_2*.

3.4 Měření elektrických a neelektrických veličin

Celkově jsem navrhnul měření napětí a proudu na vstupu i výstupu zvyšujícího měniče, měření jeho teploty, teploty okolí a měření intenzity osvětlení.

Pro správnou funkci zvyšujícího měniče je zapotřebí měřit jeho výstupní napětí a měření intenzity osvětlení slouží pro regulaci jasu digitronů. Ostatní měřené veličiny jsou pouze informační a případně poskytují informace pro softwarové ochrany.

3.4.1 Měření napětí



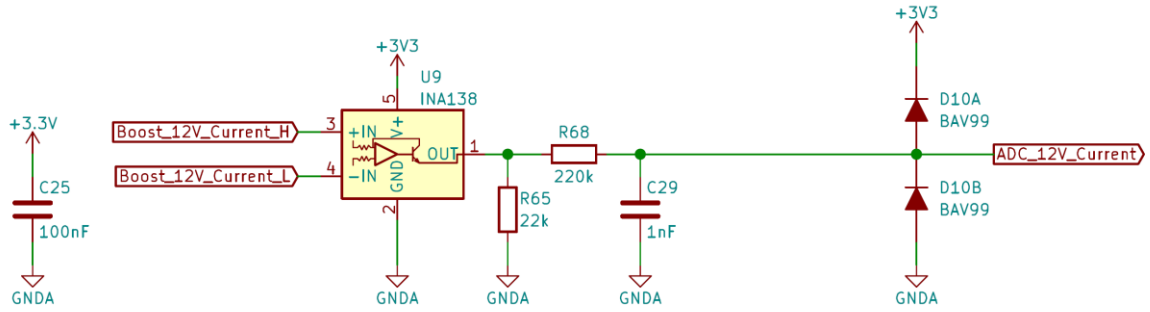
Obrázek 3.7: Schéma měření napětí.

Měření napětí na výstupu zvyšujícího měniče se skládá z napěťového děliče, který je tvořen rezistory *R71* a *R72*, filtru typu dolní propust který je tvořen tímto děličem a kondenzátorem *C31* a ochrannými diodami *D11A* a *D11B*. Dělič je navržen tak, že maximální měřené napětí je 211,3V s krokem po 52mV a mezní frekvence dolní propusti je přibližně 1kHz.

Měření napětí na vstupu zvyšujícího měniče se skládá z napěťového děliče, který je tvořen rezistory *R73* a *R74*, filtru typu dolní propust který je tvořen tímto děličem a kondenzátorem *C32* a ochrannými diodami *D12A* a *D12B*. Dělič je navržen tak, že maximální měřené napětí je 18,7V s krokem po 5mV a mezní frekvence dolní propusti je přibližně 1kHz.

Mikrokontrolér čte napětí svým A/D převodníkem pomocí signálů *ADC_180V_Voltage* a *ADC_12V_Voltage*.

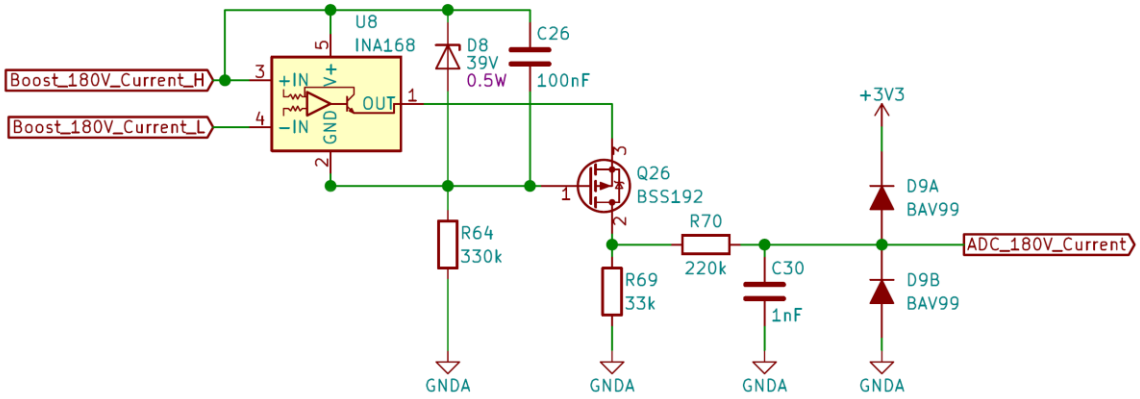
3.4.2 Měření proudu na vstupu měniče



Obrázek 3.8: Schéma měření proudu na vstupu měniče.

Proud je měřen jako úbytek napětí na rezistoru *R62 (OBRÁZEK 3.3)*. Tento úbytek je zesílen specializovaným obvodem *U9* (INA138). Jedná se v podstatě o rozdílový zesilovač s proudovým výstupem. Výstupní napětí je vytvořeno na rezistoru *R65*, který také určuje zesílení. Dále následuje filtr typu dolní propust, který je tvořen rezistorem *R68* a kondenzátorem *C29* a ochranné diody *D10A* a *D10B*. Kondenzátor *C25* je blokovací. Zapojení je navrženo tak, že maximální měřený proud je 1,1A s krokem 277,3µA a mezní frekvence dolní propusti je přibližně 700Hz. Mikrokontrolér čte proud svým A/D převodníkem pomocí signálu *ADC_12V_Current*.

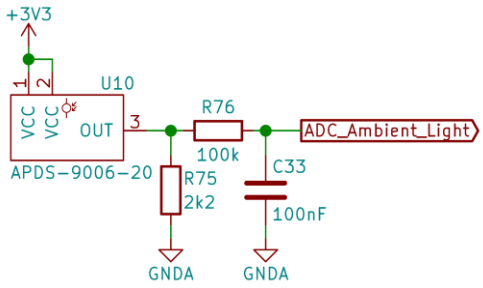
3.4.3 Měření proudu na výstupu měniče



Obrázek 3.9: Schéma měření proudu na výstupu měniče.

Proud je měřen jako úbytek napětí na rezistoru *R63* (*OBRÁZEK 3.3*). Tento úbytek je zesílen specializovaným obvodem *U8* (INA168). Jelikož má INA168 maximální vstupní měřené napětí 60V a v mé aplikaci potřebuji, aby toto napětí bylo vyšší než 180V, tak jsem použil zapojení, které je popsáno v aplikační zprávě SLLA190 [12]. Maximální vstupní měřené napětí je určeno tranzistorem *Q26*, v mém případě -240V. Zenerova dioda *D8* vytváří napájecí napětí pro *U8*, které plave vzhledem ke vstupnímu napětí. Rezistor *R64* určuje proud zenerovo diodou *D8*. Kondenzátor *C26* je blokovací. Výstupní napětí je vytvořeno na rezistoru *R69*, který také určuje zesílení. Dále následuje filtr typu dolní propust, který je tvořen rezistorem *R70* a kondenzátorem *C30* a ochranné diody *D9A* a *D9B*. Zapojení je navrženo tak, že maximální měřený proud je 32,2mA s krokem 7,9µA a mezní frekvence dolní propusti je přibližně 700Hz. Mikrokontrolér čte proud svým A/D převodníkem pomocí signálu *ADC_180V_Current*.

3.4.4 Měření intenzity osvětlení

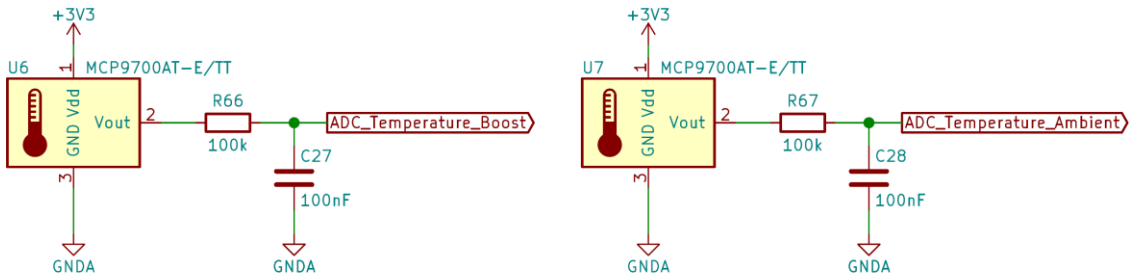


Obrázek 3.10: Schéma měření intenzity osvětlení.

Měření intenzity osvětlení jsem založil na senzoru *U10* (APDS-9006-20), který obsahuje světelný senzor, který má charakteristiku citlivosti na světlo velmi podobnou lidskému oku a tak se hodí do aplikací, kde se například automaticky reguluje jas displeje. Výstup je lineární s intenzitou osvětlení a je kompenzovaný podle teploty a napájecího napětí. Na proudový výstup *U10* je připojen rezistor *R75*, na kterém se vytváří výstupní napětí. Dále následuje filtr typu dolní propust, který je tvořen rezistorem *R76* a kondenzátorem *C33*. Zapojení je navrženo tak, že maximální měřená intenzita světla je 1000lx s krokem 0,2lx a mezní frekvence dolní propusti je přibližně 16Hz. Mikrokontrolér čte intenzitu osvětlení svým A/D

převodníkem pomocí signálu *ADC_Ambient_Light*.

3.4.5 Měření teploty



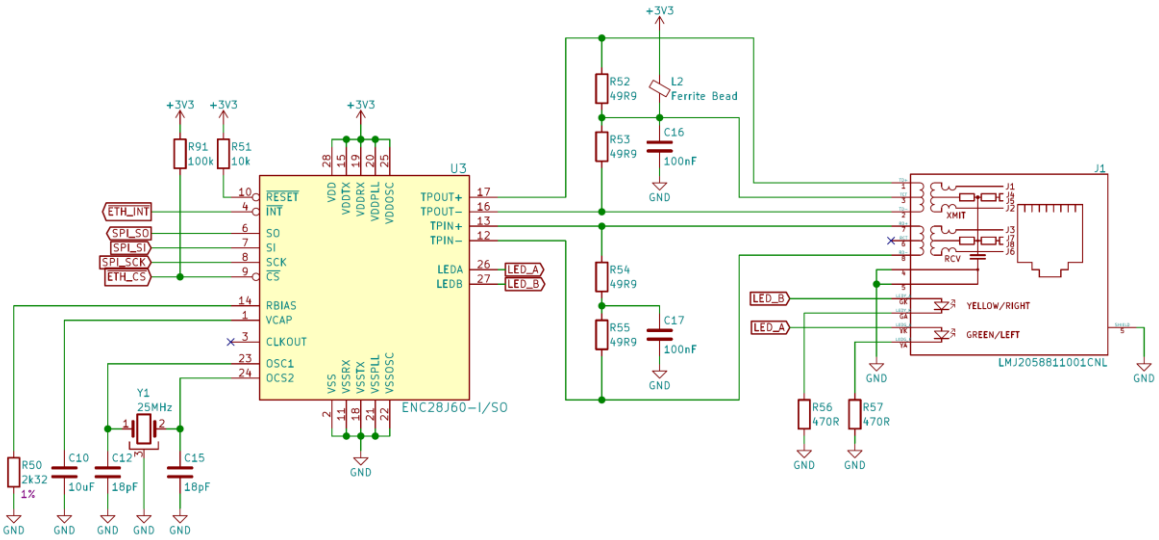
Obrázek 3.11: Schéma měření teploty.

Měření teploty je stejné v případě okolní teploty i teploty zvyšujícího měniče. Budu tedy popisovat jen měření teploty zvyšujícího měniče. Měření teploty je založené na senzoru *U6* (MCP9700), který obsahuje termistor s aktivní kompenzací a výstup je tedy lineární s teplotou. Pokud se teplota zvýší o 1°C, tak výstupní napětí vzroste o 10mV. Za výstupem *U6* následuje filtr typu dolní propust, který je tvořen rezistorem *R66* a kondenzátorem *C27*. Zapojení je navrženo tak, že rozsah měřené teploty je -50°C až 50°C s krokem 0,02°C a mezní frekvence dolní propusti je přibližně 16Hz. Mikrokontrolér čte teploty svým A/D převodníkem pomocí signálů *ADC_Temperature_Boost* a *ADC_Temperature_Ambient*.

V návrhu jsem zapomněl blokovací kondenzátory, což způsobilo poměrně velké rušení na výstupu, tyto blokovací kondenzátory jsem do konstrukce dodatečně doplnil.

3.5 Ethernet

3.5.1 ENC28J60



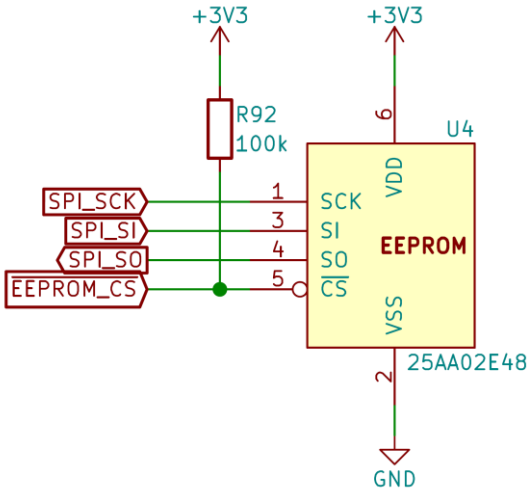
Obrázek 3.12: Schéma zapojení Ethernetu.

Pro připojení do internetu jsem vybral 10Base-T ethernet kontrolér ENC28J60 [13] od firmy Microchip. Tento kontrolér obsahuje rozhraní fyzické vrstvy (PHY) a rozhraní spojové vrstvy (MAC). Fyzická vrstva je kompatibilní s běžně používanými sítěmi 10/100/1000Base-T a podporuje automatickou detekci a korekci polarity, Full-Duplex i Half-Duplex mód. Dále kontrolér obsahuje generátor CRC, 8kB buffer (který je uživatelsky rozdělen na přijímací a vysílací část), DMA pro interní přesuny dat, přijímací filtry a dva programovatelné výstupy pro LED. Pro komunikaci s řídícím mikrokontrolérem se používá sériové rozhraní SPI s frekvencí hodinového signálu až 20MHz.

Další možností ethernetového kontroléru je ENC424J600, který je novější verzí mnou použitého, ale má 100Base-T PHY, 24kB buffer a hardwarovou podporu šifrování nebo například W5500 od firmy Wiznet, který obsahuje 100Base-T PHY, 32kB buffer a TCP/IP stack, což velmi zjednodušuje software mikrokontroléru. ENC28J60 jsem zvolil z důvodu ceny, jednoduchosti, dostupných materiálů a předchozích zkušeností.

Zapojení začíná konektorem *J1*. Jedná se o MagJack® RJ-45 konektor, který má integrované transformátory i další potřebné pasivní prvky a dvě indikační LED diody. Rezistory *R56* a *R57* slouží pro omezení proudu indikačními LED diodami konektoru. Rezistory *R52*, *R53*, *R54*, *R55* a kondenzátory *C16* a *C17* slouží k impedančnímu přizpůsobení. Feritový korálek *L2* slouží pro napájení vysílacího transformátoru a zamezení šíření rušení zpět do zdroje. Kontrolér je napájený z 3,3V a každý napájecí pin má svůj blokovací kondenzátor. Tyto kondenzátory nejsou znázorněny na *OBRÁZKU 3.12*. Pin reset nevyužívám, takže je trvale připojen přes rezistor *R51* na napájení. Rezistor *R50* nastavuje amplitudu signálu na pinech TPOUT+/- a jeho hodnota je určena výrobcem tak, aby byly splněny požadavky podle IEEE 802.3. Kondenzátor *C10* je filtrační kondenzátor pro interní 2,5V napěťový regulátor. Dále je zapotřebí zdroj hodinového signálu 25MHz, který je tvořen krystalem *Y1* a kondenzátory *C12* a *C15*. Pin CLKOUT může poskytovat tento hodinový signál vydělený určitými děličkami pro další obvody, ale v mém zapojení není využit. Pro komunikaci s mikrokontrolérem se využívá sériová sběrnice SPI, která využívá signály *SPI_SO*, *SPI_SI*, *SPI_SCK* a *ETH_CS*. *ETH_CS* je signál pro výběr tohoto čipu na sběrnici SPI a je přes rezistor *R91* připojen na napájení, čímž je zajištěno, že tento čip bude ve výchozím stavu nevybrán. Signál *ETH_INT* je využit pro signalizaci přerušení do mikrokontroléru.

3.5.2 EEPROM s MAC adresou

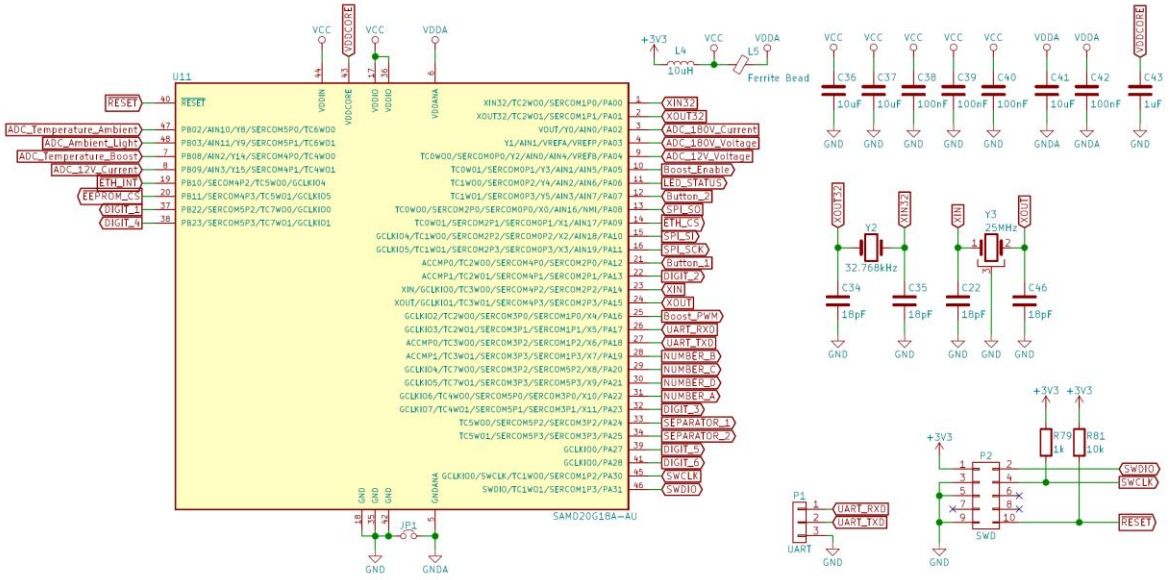


Obrázek 3.13: Schéma zapojení EEPROM.

Každé zařízení, které komunikuje v sítí Ethernet, musí mít svojí unikátní MAC adresu, která ho identifikuje. Pokud se zařízení používá jen na známé lokální síti, tak mu lze přidělit lokální MAC adresu, ale pokud by s touto adresou bylo zapojeno do jiné sítě, kde by se vyskytovalo zařízení se stejnou MAC adresou, tak by nastal konflikt, kdy by obě zařízení vystupovala na síti jako jedno stejné zařízení. Proto má každé komerčně vyrobené zařízení MAC adresu unikátní. Výrobce si kupuje bloky MAC adres od organizace IEEE, které obsahují buď 4 096 adres (individuální blok) nebo 16 777 214 adres (blok pro organizace). Tyto bloky stojí poměrně hodně peněz a to je naprosto vylučuje pro mojí aplikaci.

Další možností získání unikátní MAC adresy jsou paměti EEPROM, které mají ve své paměti od výrobce naprogramovanou unikátní MAC adresu. Vybral jsem EEPROM 25AA02E48 (*U4*) od firmy Microchip [14], která má 2Kb paměti, sériové SPI rozhraní a obsahuje naprogramovanou unikátní MAC adresu. K mikrokontroléru je připojena signály *SPI_SCK*, *SPI_SI* a *SPI_SO*, které sdílí s ethernetovým kontrolérem *U3* a signálem *EEPROM_CS*, který slouží pro výběr tohoto čipu na sběrnici SPI a je ve výchozím stavu nevybrán, což zajišťuje rezistor *R92*.

3.6 Mikrokontrolér



Obrázek 3.14: Schéma zapojení mikrokontroléru.

Pro výběr řídícího mikrokontroléru jsem měl několik požadavků. Musí být založen na jádře ARM Cortex-M a dále musí mít dostatek vstupně výstupních pinů pro řízení multiplexovaného digitronového displeje, řízení zvyšujícího měniče a připojení SPI sběrnice. Musí obsahovat alespoň jeden časovač/čítač s PWM výstupem pro zvyšující měnič a alespoň jeden další časovač/čítač pro časování programu. Dále je nutný A/D převodník a podpora pro SPI sběrnici. Datová paměť musí mít alespoň 5kB, aby se do ní vešly dva ethernetové rámce (2x 1 518B) a aby zbyl prostor pro běh programu.

Tyto požadavky splňuje velké množství mikrokontrolérů, ze kterých jsem vybral SAMD20G18 (*U11*) od firmy Microchip [15]. Tento mikrokontrolér je založen na ARM jádře Cortex-M0+ s 256kB programové paměti a 32kB paměti datové, maximální hodinový kmitočet je 48MHz a disponuje 38 vstupně výstupními piny. Dále obsahuje tyto periferní obvody:

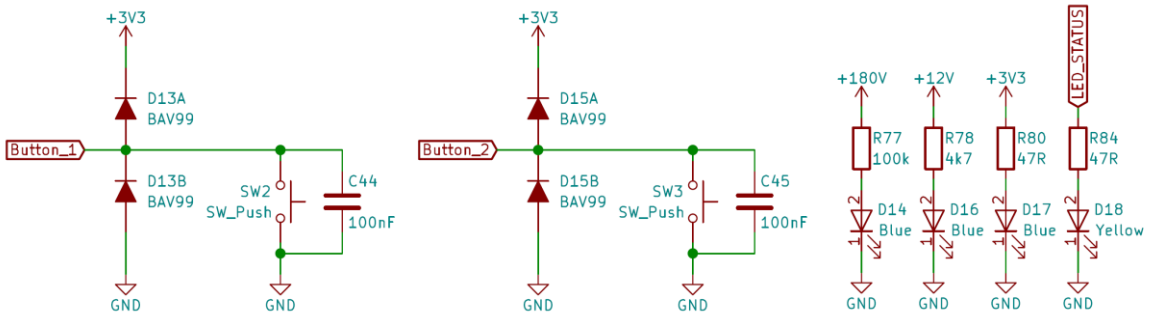
- 6x 16 bitový časovač/čítač
- 6x SERCOM – komunikační modul obsahující USART, SPI, I^2C

- 12 bitový A/D převodník se 14 kanály
- 2x analogový komparátor
- 10 bitový D/A převodník s dvěma kanály
- 16x vstup pro externí přerušení

Napájení pro mikrokontrolér je přivedeno přes tlumivku *L4* pro napájení digitální části mikrokontroléru a dále pokračuje přes feritový korálek *L5*, který filtruje napájení pro část analogovou. Kondenzátory *C36* až *C42* slouží jako blokovací a kondenzátor *C43* je filtrační pro vnitřní napěťový regulátor mikrokontroléru. Zapojení napájení je převzato z doporučeného zapojení od výrobce, které je uvedeno v katalogovém listu.

Pro získání přesného kmitočtu pro mikrokontrolér a počítání času jsem využil externí oscilátor, tvořený krystalem *Y2* a kondenzátory *C34* a *C35*, který je navržen pro 32,768kHz krystal. Z tohoto kmitočtu se pomocí fázového závěsu násobí hlavní 48MHz kmitočet. Dále je v zapojení další externí oscilátor, tvořený krystalem *Y3* a kondenzátory *C22* a *C46*, který podporuje krystaly až do 32MHz. Tento oscilátor jsem ale nevyužil a není osazený.

Z mikrokontroléru je vyveden UART na kolíkovou lištu *P1* a slouží pro odesílání ladících informací. Mikrokontrolér se programuje a ladí přes standardní rozhraní SWD (Serial Wire Debug). Toto rozhraní je vyvedeno na kolíkovou lištu *P2* a je doplněno doporučenými rezistory *R79* a *R81*.



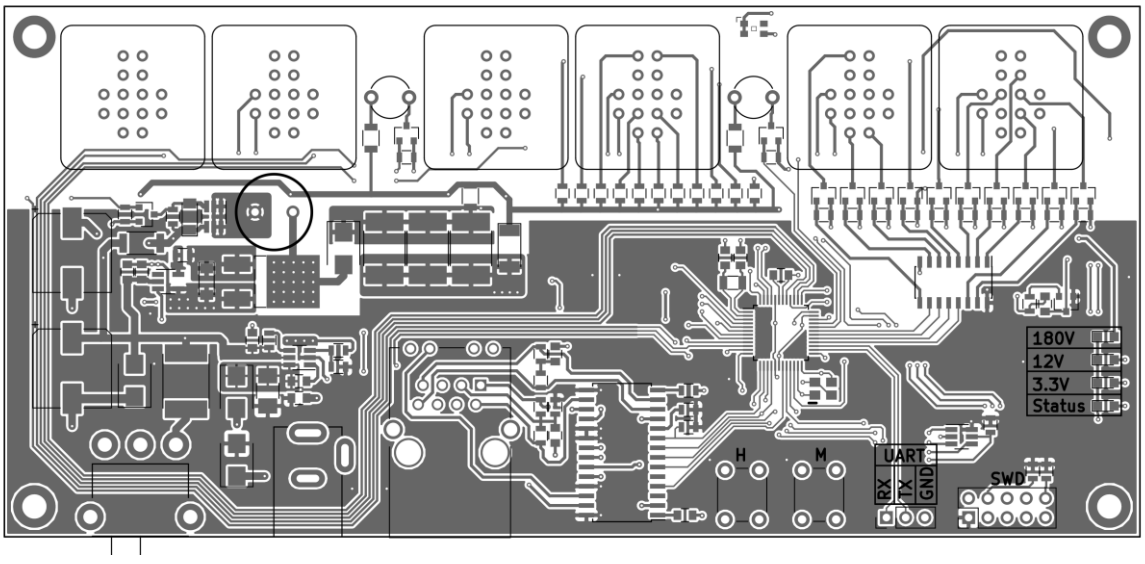
Obrázek 3.15: Schéma zapojení tlačítek a LED diod.

K mikrokontroléru jsou dále připojena dvě tlačítka *SW2* a *SW3*, která slouží pro nastavování času, pokud není dostupné připojení k internetu. Paralelně s tlačítky jsou zapojeny kondenzátory *C44* a *C45*, které potlačují zákmity, které vznikají při spínání. Výstupní signály *Button_1* a *Button_2* jsou zapojeny do vstupů vnějších přerušení mikrokontroléru, které jsou chráněné diodami *D13A*, *D13B* a *D15A*, *D15B*. LED dioda *D18* indikuje stav zařízení a je k mikrokontroléru připojena signálem *LED_STATUS*. LED diody *D14*, *D16* a *D17* indikují přítomnost jednotlivých napájecích napětí, která se v zařízení vyskytují.

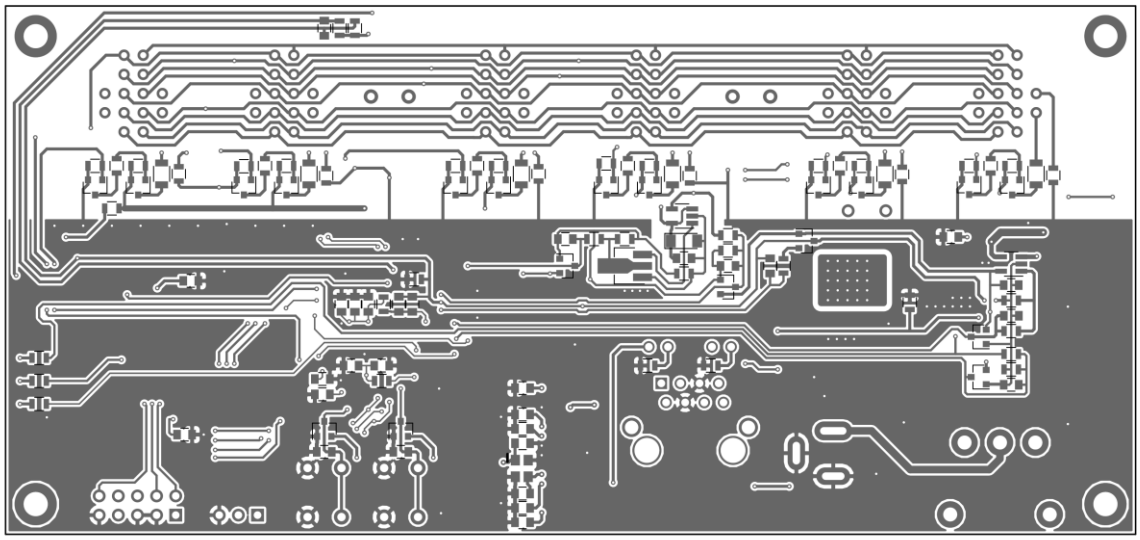
3.7 Plošný spoj

Celé zařízení je navrženo na jedné oboustranné desce plošných spojů o rozměrech 150x70mm. Plošný spoj jsem navrhnul v programu KiCad a nechal vyrobit ve firmě PragoBoard na materiál FR4 o tloušťce 1,5mm s povrchovou úpravou chemické zlato.

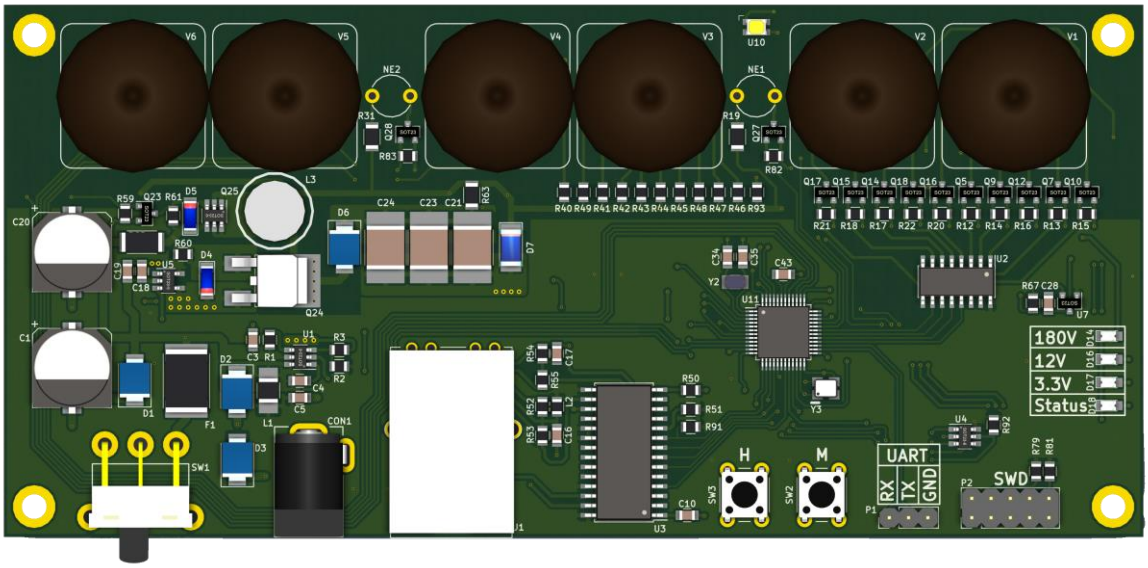
Plošný spoj jsem navrhnul s většinou součástek pro povrchovou montáž a součástky jsou umístěny z obou stran plošného spoje.



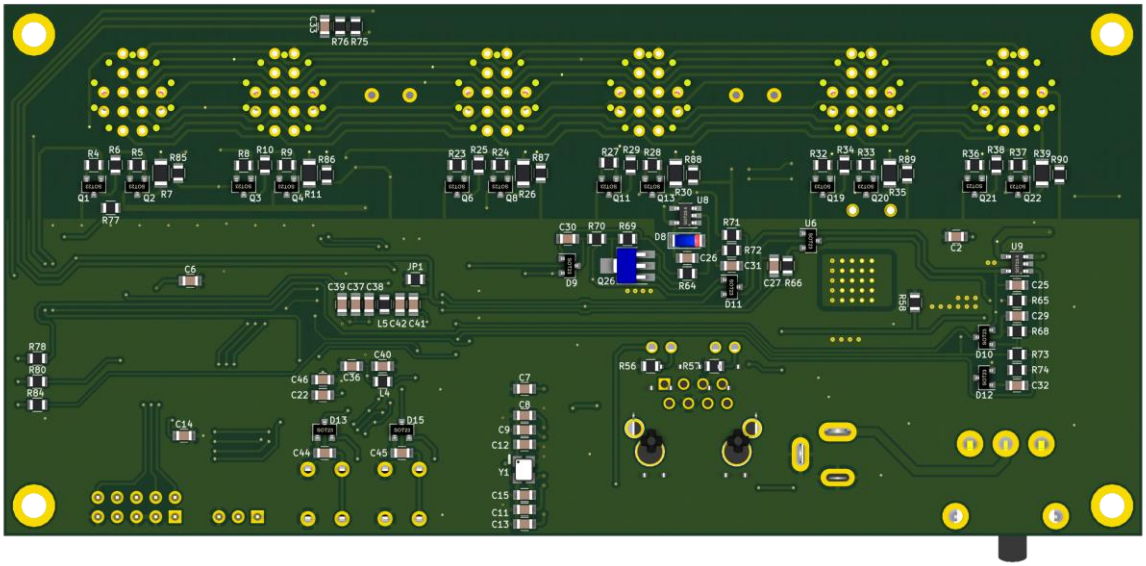
Obrázek 3.16: Plošný spoj strana TOP.



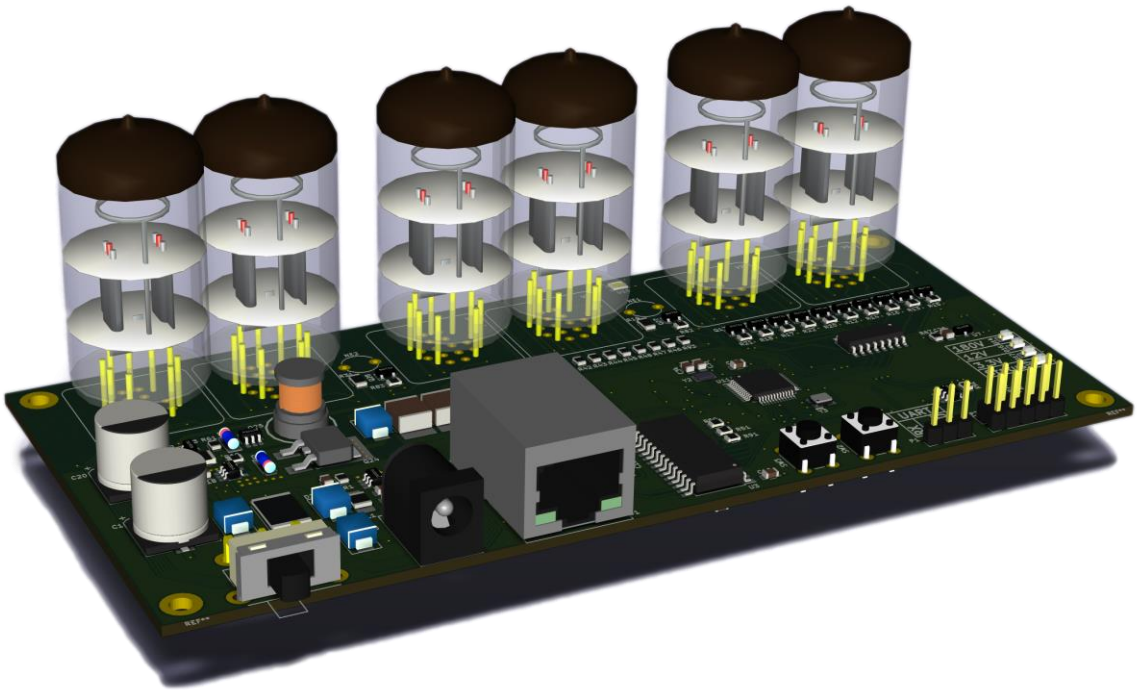
Obrázek 3.17: Plošný spoj strana BOT.



Obrázek 3.18: 3D model plošného spoje, strana TOP.



Obrázek 3.19: 3D model plošného spoje, strana BOT.



Obrázek 3.20: 3D model plošného spoje.

3.8 Oživení

Oživení zařízení probíhalo ve dvou krocích. Nejdříve jsem osadil zdroj (*KAPITOLA 3.1*) a ověřil jsem jeho funkčnost. Zdroj fungoval správně a výstupní napětí snižujícího měniče bylo 3,3V. Na základě této kontroly jsem osadil zbytek součástek a měl jsem jistotu, že všechny další obvody budou mít správné napájecí napětí. Další

ověřování jednotlivých bloků probíhalo postupně při programování. Narazil jsem na problém s pouzdrem čidla intenzity osvětlení *U10* (*OBRÁZEK 3.10*), kterému jsem špatně nakreslil pouzdro. Naštěstí stačilo tento senzor otočit o 180° a nyní funguje korektně. Dále jsem zjistil, že jsem nakreslil obráceně jednu LED diodu v konektoru *J1*.

4 Software

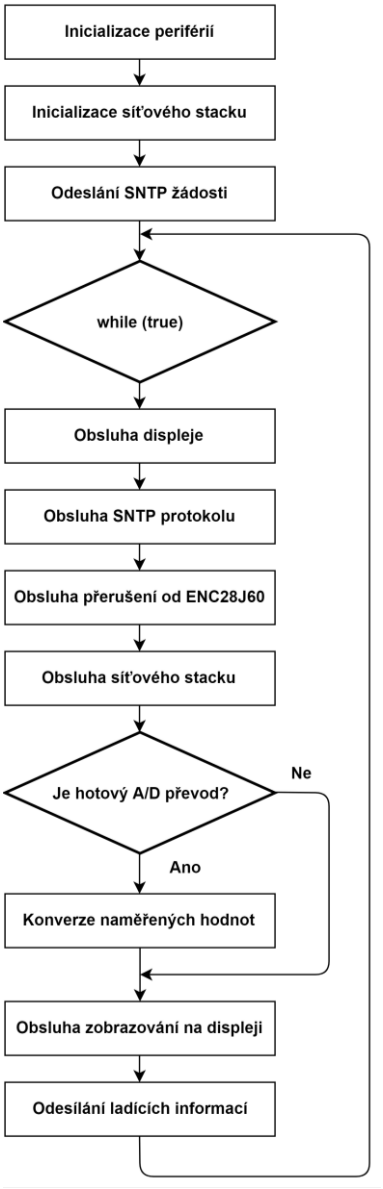
Software pro mikrokontrolér jsem naprogramoval v jazyce C++ verze gnu++14. Pro programování jsem využil Atmel Studio 7, které staví na Microsoft Visual Studiu, díky kterému obsahuje dobré nástroje pro manipulaci s kódem (zvýrazňování syntaxe, doplňování kódu, refraktoring a další) a dále obsahuje nástroje pro ladění kódu na mikrokontroléru s velmi pěkným zobrazením jednotlivých řídících registrů, které mi velmi pomohlo při zprovozňování jednotlivých periférií.

Pro programování a ladění kódu do mikrokontroléru jsem používal oficiální programátor/debugger Atmel ICE se kterým jsem nezaznamenal problémy.

Software jsem se snažil napsat tak, že na začátku se inicializují všechny periférie a poté se začne vykonávat hlavní smyčka programu. Ta je naprogramována tak, aby co nejméně blokovala výpočetní čas mikrokontroléru a je v ní tedy jen nezbytné množství čekacích smyček a časování společně s obsluhou periférií probíhá v přerušeních. Každá periférie, případně funkční blok má svojí třídu, ve které je metoda pro inicializaci a další metody pro obsluhu. Snažil jsem se, aby byl program co nejvíce modulární a aby se jednotlivé moduly (třídy) daly využít i v dalších programech, nebo aby mohly být jednoduše vyměnitelné. Tyto moduly jsou popsané v následujících kapitolách.

Když uživatel zapne hodiny, tak se rozsvítí digitronový displej s vynulovaným časem. Čas lze nastavovat tlačítky *SW2* a *SW3* (*KAPITOLA 4.4*). Po připojení hodin do internetu se automaticky sesynchronizuje čas z NTP serveru a aktualizace času se provádí každých 15 minut. Displej se postupně přepíná mezi zobrazením času, data a teploty (*KAPITOLA 4.10.4*).

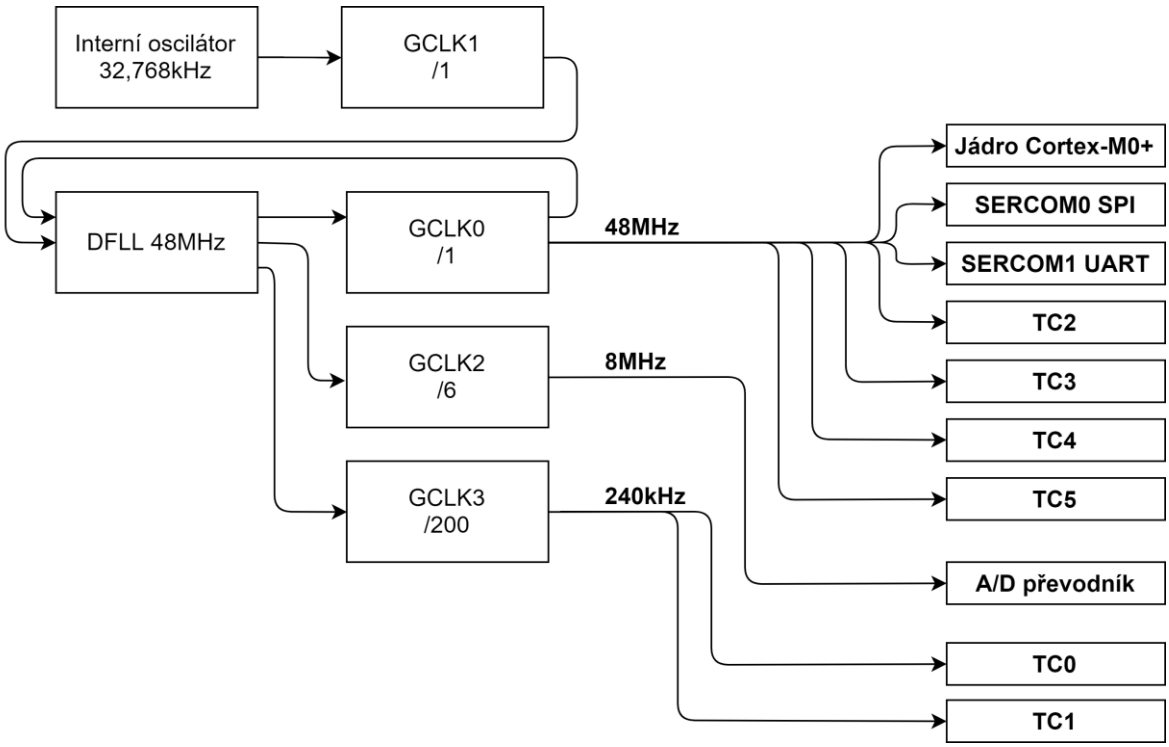
4.1 Hlavní smyčka



Obrázek 4.1: Vývojový diagram hlavní smyčky.

Program začíná inicializací jednotlivých periférií mikrokontroléru, dále se inicializuje ethernetový kontrolér ENC28J60 a síťový stack. Poté se odešle žádost o synchronizaci času na NTP server *clock1.zcu.cz* a začne se vykonávat hlavní smyčka. V této smyčce se volají obsluhy jednotlivých modulů, které buď kontrolují nějaký stav, nebo uplynulý čas a podle toho se buď obsluha vykoná, nebo se program vrátí do hlavní smyčky.

4.2 Nastavení generátorů hodin



Obrázek 4.2: Schéma rozvodu hodinových signálů.

Jako první se po startu programu nastavují zdroje hodinového signálu pro jádro mikrokontroléru a jednotlivé periferie. Inicializaci hodin zajišťuje metoda *void init_clock()* ze třídy *System*.

Původně jsem chtěl pro hlavní systémový kmitočet využít externí 32,768kHz oscilátor, ale při programování jsem se setkal s velkou nestabilitou tohoto oscilátoru, která je nejpravděpodobněji způsobena špatným návrhem plošného spoje, kde není krystal *Y2* dostatečně odstíněn od ostatních signálů. Používám tedy interní 32,768kHz oscilátor, který sice nemá ideální přesnost a jeho kmitočet je závislý na teplotě a napájecím napětí, ale to v mé aplikaci tolik nevadí, protože se čas synchronizuje v pravidelných intervalech z internetu a tím se eliminují tyto nepřesnosti.

Hodinový signál z interního 32,768kHz oscilátoru jde přes GCLK1 (Generic Clock Controller) do fázového závěsu DFLL (Digital Frequency Locked Loop), který pracuje

v uzavřené smyčce a vstupní signál násobí na 48MHz. Výstup Z DFLL je napojen na GCLK0, GCLK2 a GCLK3. GCLK0 slouží jako zdroj hodinového signálu 48MHz pro jádro mikrokontroléru, komunikační moduly SERCOM0 a SERCOM1, a čítače/časovače TC2, TC3, TC4 a TC5. Výstup GCLK0 dále slouží jako porovnávací kmitočet pro DFLL. GCLK2 dělí vstupní kmitočet 6 a výstupní hodinový signál 8MHz je napojen na A/D převodník. GCLK3 dělí vstupní kmitočet 200 a výstupní hodinový signál 240kHz je napojen na čítače/časovače TC0 a TC1.

Rozvod hodinových signálů má několik omezení. Vždy dva po sobě jdoucí čítače/časovače (TC0 a TC1, TC2 a TC3, …) sdílí stejný hodinový signál. Pro běh mikrokontroléru na 48MHz je zapotřebí pro napájení 3,3V nastavit 1 čekací cyklus pro integrovanou FLASH paměť, ale tato informace není v katalogovém zmíněna v sekcích, které se zabývají nastavováním hodin, ale jen v sekci elektrických charakteristik (str. 628: 33.11 NVM Characteristics [15]).

4.3 Vnější přerušení

Vnější přerušení jsou ovládána přes EIC (External Interrupt Controller) a jsou využita tři a to pro signály tlačítek *Button_1* a *Button_2*, která mají detekci hrany nastavenou na vzestupnou i sestupnou hranu a pro signál *ETH_INT* od ethernetového kontroléru *U3*, který má detekci hrany nastavenou na sestupnou hranu. Tyto piny mají povolený vnitřní pull-up rezistor a vstupní filtr, který vzorkuje tři vzorky a majoritní obvod poté vyhodnocuje, jestli se má zavolat dané přerušení.

Inicializaci provádí metoda *void init_eic()* ze třídy *System* a obsluhu jednotlivých přerušení provádějí metody daných bloků, kterých se dané přerušení týká.

4.4 Tlačítka

Pro tlačítka jsem naprogramoval třídu *Button*, která obsahuje metodu *void interrupt_handler()*, která obsluhuje přerušení od daného tlačítka. Pokud je vyvoláno přerušení, tak se uloží stav tlačítka (stisknuto/nestisknuto) a nastaví se

příznak, že toto tlačítko má nový stav. Dále třída obsahuje metody *bool is_set()*, *bool is_serviced()* a *void serviced()*, které slouží jako interface pro získání stavu tlačítka.

Co tlačítka dělají, je naprogramováno ve třídě *NixieClock* v metodě *void buttons_handler()*. Jelikož není potřeba na hodinách nic nastavovat, protože se řídí automaticky přes internet, tak tlačítka slouží k nouzovému nastavení času v případě, že není k dispozici připojení k NTP serveru. Tlačítko *SW3* nastavuje hodiny – při každém stisku se přičte jedna hodina. Tlačítko *SW2* nastavuje minuty – při každém stisku se přičte jedna minuta a vynulují se sekundy.

4.5 UART

Pro výpis ladících informací na počítači jsem využil komunikační modul SERCOM1 (Serial Communication Interface) v režimu UART. Potřebná inicializace a metody pro odesílání dat jsou naprogramovány ve třídě *USART*.

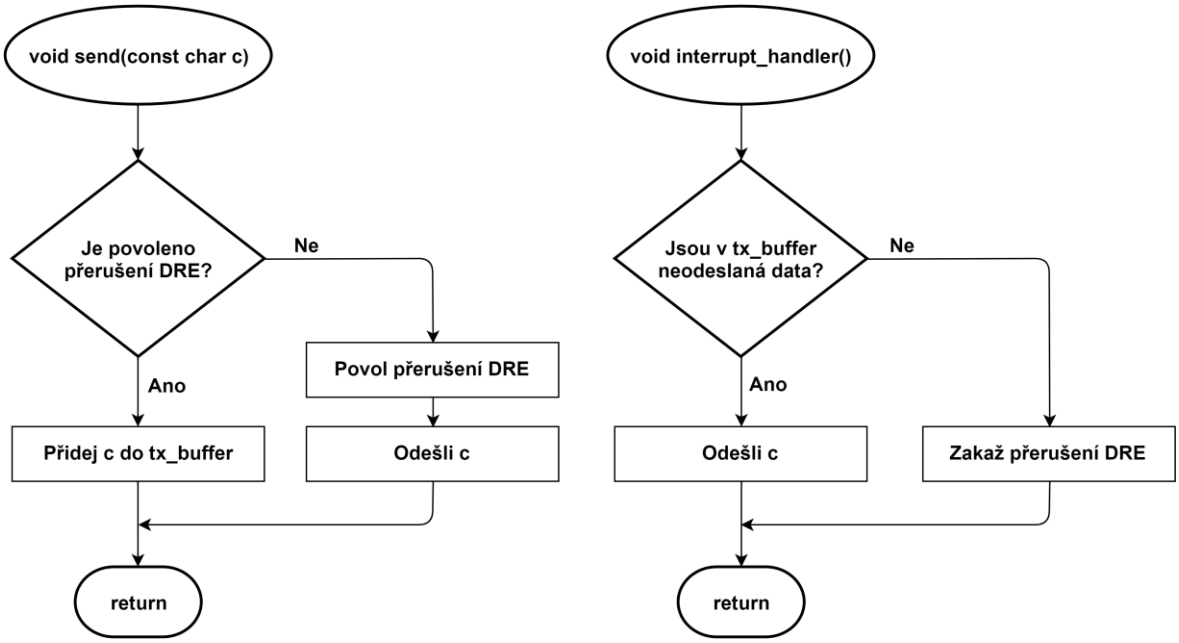
Důležitou informací, která je nutná pro výpočet hodnoty BAUD registru a která není zmíněná v katalogovém listu, je počet vzorků na jeden bit přijímaných dat. Experimentálně jsem zjistil, že vzorků je 16.

$$BAUD = 65\,536 \cdot \left(1 - 16\frac{f_{BAUD}}{f_{clk}}\right) \tag{4.1}$$

Dále je potřeba povolit hodinový signál CLK_SERCOM1_APB, který má být podle katalogového listu zapnutý po resetu a hodinový signál GCLK_SERCOM1_CORE, který je ve výchozím stavu vypnutý.

Odesílání dat probíhá v přerušení DRE (Data Register Empty), které je vyvoláno, pokud je volný odesílací registr DATA (*OBRÁZEK 4.3*). Metody pro odesílání dat tedy plní kruhový buffer *tx_buffer* o velikosti 500B, ze kterého se v přerušení odesílají jednotlivé bajty a nedochází tak ke zpomalování programu, ke kterému by

docházelo, pokud by odesílání probíhalo blokujícím způsobem.



Obrázek 4.3: Vývojový diagram odesílání dat přes UART.

Parametry přenosu jsou následující:

- Modulační rychlost 1 000 000 baud
- 8 datových bitů
- 1 stop bit
- Bez parity

4.6 SPI

Pro komunikaci s ethernetovým kontrolérem *U3* a EEPROM *U4* je použita SPI sběrnice, kterou obsluhuje SERCOM0 v módu SPI master. Inicializaci a obsluhu SPI sběrnice zajišťuje třída *SPI*. Podle elektrických charakteristik, které uvádí katalogový list mikrokontroléru, je maximální frekvence hodinového signálu SPI_SCK 11,9MHz. ethernetový kontrolér podporuje frekvenci až 20MHz a frekvence by měla být vyšší než 8MHz, jinak může dojít k nesprávnému čtení a zapisování registrů [16]. Frekvenci jsem tedy nastavil na 11,8MHz.

Rychlost SPI sběrnice je vzhledem k mikrokontroléru dostatečně vysoká, takže jsem neřešil odesílání a příjem dat přes přerušení, které by navíc zbytečně zkomplikovalo program.

Výběrové signály, které vybírají, se kterým obvodem se bude komunikovat, obsluhuje třída *SPI_select*, která má jednu metodu *void select(bool state)*. Pro každý obvod, který je připojen na SPI sběrnici, je instancována zvláštní instance této třídy. Není tedy kontrolováno, že nebude zároveň vybráno více obvodů.

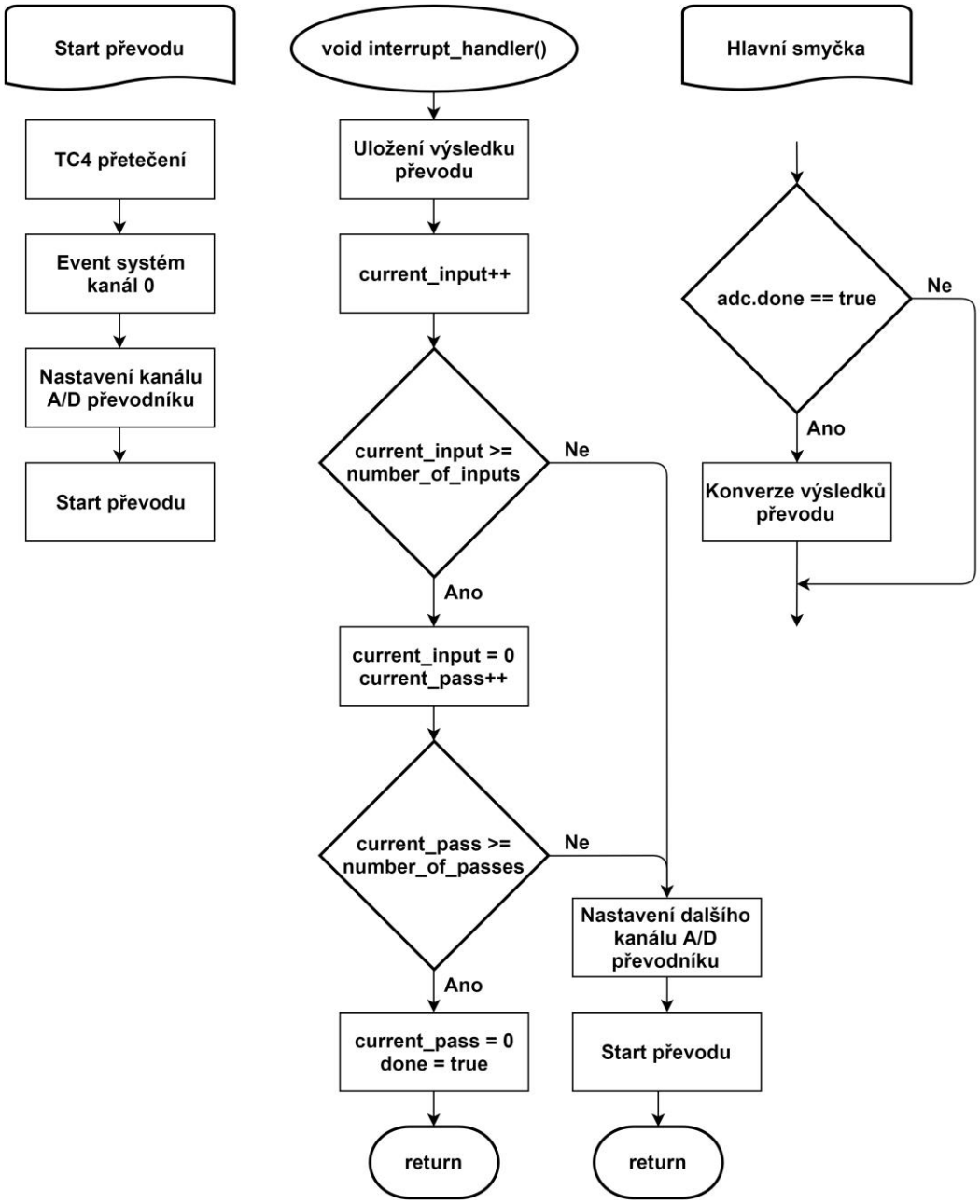
4.7 A/D převodník

Mikrokontrolér obsahuje 12 bitový A/D převodník s rychlostí převodu až 350 000 vzorků/s. Referenční napětí může být buď přivedeno externě, nebo zvoleno z interních. Využívám interní bandgap referenci 1V. Inicializaci a obsluhu A/D převodníku zajišťuje třída *adc_class*.

A/D převodník má nastavenou děličku hodinového signálu na 4 a vstupních 8MHz se tedy dělí na 2MHz, se kterými je vzorkovací frekvence 333 000 vzorků/s. Převod se startuje automaticky bez zásahu mikrokontroléru pomocí kanálu 0 event systému, jehož zdrojem je přetečení čítače TC4. Převod se spouští s frekvencí 2,5kHz.

Po dokončení převodu je vyvoláno přerušení RESRDY, které obsluhuje metoda *void interupt_handler()*. Obsluha se skládá z postupného převodu jednotlivých vstupů a tento převod se opakuje tolikrát, kolik se má převést vzorků pro průměrování a tato hodnota je řízena proměnnou *number_of_passes*. Po převedení všech potřebných vzorků se již nestartuje další převod, ale nastaví se příznak *done*, který je kontrolován v hlavní smyčce. Pokud se v hlavní smyčce vyhodnotí, že je převod hotov, tak se jednotlivé vzorky zprůměrují a převedou se na odpovídající jednotky (V, A, °C, lux). Tyto výpočty jsou prováděny s celými čísly a výsledkem jsou tedy čísla s pevnou řádovou čárkou, což zrychluje výpočty a další zpracování

naměřených hodnot.



Obrázek 4.4: Vývojový diagram A/D převodu.

4.8 Zvyšující měnič

Obsluhu zvyšujícího měniče řeší třída *BoostConverter*. Způsob řízení měniče je velice jednoduchý, ale této aplikaci postačuje. Pro generování pulzně šířkové modulace (PWM) používám čítač/časovač TC2, který generuje 8 bitovou PWM o frekvenci 46,875kHz.

Požadované napětí se nastavuje metodou *void set_voltage(const uint8_t voltage)* a od tohoto nastaveného napětí se určí minimální a maximální hodnota napětí podle konstanty *HYSTERESIS*. Regulátor *void regulator_handler()* se volá po dokončení převodu a konverzi hodnot A/D převodníku a porovnává naměřené napětí s mezemi, které jsou určeny hysterezí. Pokud je naměřené napětí menší než *minimum_voltage*, tak se zvýší střída řídící PWM, pokud je naměřené napětí větší než *maximum_voltage*, tak se střída zmenší.

Metoda *void protections_handler()* poskytuje ochrany měniče a digitronů. Obsahuje ochranu proti přepětí na výstupu měniče – změřená hodnota napětí se porovnává s konstantou *MAXIMUM_VOLTAGE_180V*, pokud je naměřené napětí vyšší, tak se sníží střída PWM a případně se celý měnič vypne tranzistorem *Q25*. Ochrana proti nadproudu na výstupu měniče porovnává změřenou hodnotu proudu s konstantou *MAXIMUM_CURRENT_180V* a pokud je změřený proud vyšší, tak vypne měnič tranzistorem *Q25*. Podobně funguje tepelná ochrana měniče, která porovnává změřenou teplotu s konstantou *MAXIMUM_TEMPERATURE* a pokud je naměřená hodnota vyšší, tak se měnič vypne tranzistorem *Q25*.

4.9 Počítání času

Časové funkce jsem využil ze standardní C knihovny <ctime>. Původně jsem chtěl využít periferní obvod RTC (Real Time Clock), který mikrokontrolér obsahuje, ale kvůli synchronizaci zápisu a čtení jeho registrů by docházelo k dlouhé synchronizaci, která by zpomalovala program. Také bych ho nemohl využít k počítání milisekund, které používám k časování částí programu.

Pro zprovoznění těchto funkcí je potřeba přidat parametr --specs=nosys.specs pro linker a do souboru *samd20_sram.ld* jsem přidal „`end = .;`“ za řádek „`_end = . ;`“.

Čas obsluhuje třída *Time*. Čítač/časovač TC0 je nastaven tak, aby generoval přerušení každou 1ms. Jsou zde nadefinovány funkce *clock_t clock()*, která vrací

počet milisekund *clk_count* od startu programu a *time_t time(time_t *)*, která vrací počet sekund *seconds* od 1. ledna 1970. Proměnné *clk_count* a *seconds* se inkrementují v přerušení od TC0.

Pro získání času a data slouží funkce *void local_time_r(time(NULL), &time_date)*, která do struktury *time_date* uloží potřebné informace. Zde jsem narazil na problém, že tato funkce vrací čas podle lokálního časového pásma, které se na počítači nastavuje podle příslušné proměnné prostředí, kterou nastavuje operační systém. Což znamená, že na mikrokontroléru je časové pásmo nastavené na univerzální čas. Pro nastavení správného časového pásma se musí nejdříve nastavit proměnná prostředí *TZ* funkcí *setenv("TZ","CET-1CEST-2,M3.5.0/+2,M10.5.0/+1",1)* a následně se aktualizuje časové pásmo funkcí *tzset()*. Pokud se toto nastavení provede správně, tak se nastaví i přechody na letní a zpět na zimní čas.

4.10 Digitronový displej

4.10.1 BCD dekodér 4028

Nastavování výstupů pro BCD dekodér 4028, který řídí katody digitronů, obsluhuje třída *BCD4028*. Tato třída umožňuje, že řídící piny mohou být na kterýchkoliv pinech mikrokontroléru, pokud jsou na stejné bráně.

Metoda *set(const uint8_t number)* vystaví řídící signály tak, aby se na digitronu zobrazilo požadované číslo. Pokud je *number* rovno 10 (nebo konstantě *SET_BLANK*), tak se BCD dekodér nastaví do speciálního stavu, kdy jsou všechny výstupy v logické nule, a na digitronu se nezobrazuje žádná číslice.

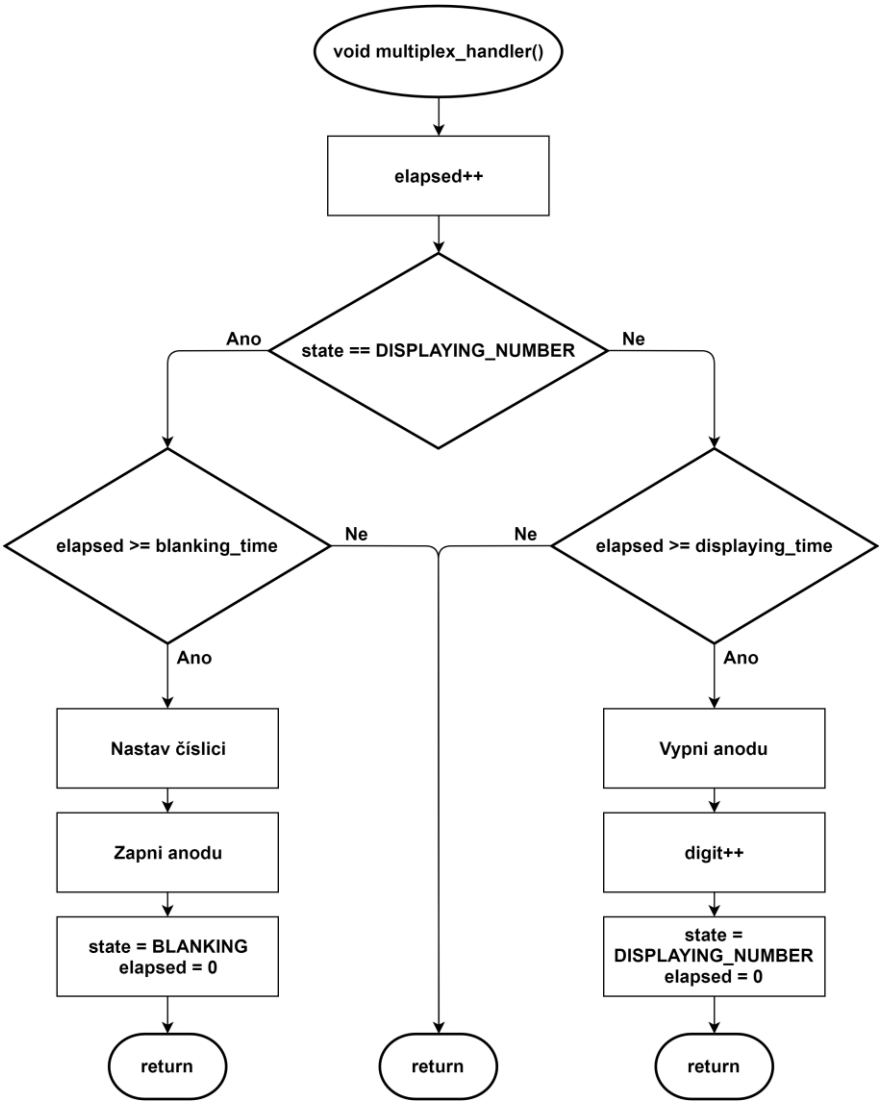
4.10.2 Doutnavkové oddělovače

Doutnavky obsluhuje třída *Separator*. Tato třída inicializuje čítač/časovač TC5, který generuje 16 bitovou PWM s frekvencí 800Hz pro doutnavky. Je tedy možné řídit

jas každé doutnavky zvlášť. Dále třída obsahuje metodu *void set_brightness(const uint8_t value)*, která nastavuje jas v rozmezí 0% až 100% podle logaritmické tabulky, což znamená, že změna jasu odpovídá citlivosti lidského oka. Metoda *void enable_pulsing()* zapíná pulzování doutnavek, kdy 1s se jas zvyšuje a další 1s se jas snižuje. Metoda *void disable_pulsing()* vypíná tento režim. Pulzování obsluhuje metoda *pulsing_handler()*, která se volá z hlavní smyčky.

Při zobrazování času doutnavky pulzují, při zobrazování data svítí na 20% a při zobrazování teploty nesvítí vůbec.

4.10.3 Řízení multiplexu digitronů



Obrázek 4.5: Vývojový diagram řízení multiplexu.

Multiplex je ovládán třídou *NixieDisplay*. Pro časování multiplexu využívám čítač/časovač TC1, jehož frekvence přetečení je 20kHz. Přetečení generuje přerušení, ve kterém se volá metoda *void multiplex_handler()*, která obsluhuje multiplex. Každý digitron má v multiplexu dvě fáze. Nejprve se nastaví číslice a zapne se příslušná anoda. V tomto intervalu digitron svítí. Poté následuje interval, kdy se příslušná anoda vypne a doutnavý výboj zaniká. Normálně může být tento interval poměrně krátký (200μs), ale aby bylo možné řídit jas digitronů, tak jsou tyto dva intervaly vůči sobe proměnné. Celková doba, kdy se obsluhuje jeden digitron, je stále stejná, ale mění se poměr intervalů svícení a zhasnutí podle změřené intenzity osvětlení.

Pro nastavení požadované hodnoty na displej slouží dvě metody *void set(uint8_t hod, uint8_t min, uint8_t sec)*, která nastaví čas, případně datum a *void set(uint8_t temperature)*, která nastaví teplotu.

4.10.4 Řízení displeje

To co se má zobrazit na displeji řeší dvě metody ze třídy *NixieClock*. Metoda *void screen_handler()* přepíná zobrazování času, data a teploty. Čas se zobrazuje 30s, datum 5s a teplota 10s. Pokud je zmáčknuto některé z tlačítek, tak se displej přepne na zobrazování času. Metoda *void display_handler()* aktualizuje každou sekundu obsah displeje aktuálním časem, datem nebo teplotou, podle vybraného režimu.

4.11 EEPROM

Pro EEPROM *U4* (25AA02E48) jsem naprogramoval třídu *EEPROM_25AA02E48*, která jí obsluhuje. Komunikace s touto EEPROM je velice jednoduchá a zajišťují ji metody *void read(...)* a *void write(...)*. Pokud se za sebou zapisuje několik bloků, tak před dalším zápisem je nutné zkontrolovat bit WIP (Write in Progress) ve status registru a případně počkat, dokud se tento bit nevynuluje. Před prvním zápisem je také nutné nastavit bit WEL (Write Enable Latch). MAC adresa je uložená v posledních šesti bajtech paměti a pro její získání slouží metoda *void read_EUI(uint8_t* buffer)*.

4.12 ENC28J60

Komunikace s ethernetovým kontrolérem *U3* je naprogramována ve dvou třídách. Třída *ENC28J60_SPI* implementuje SPI příkazy pro čtení a zápis registrů, nastavování jednotlivých bitů a podobně. Třída *ENC28J60* implementuje metody pro inicializaci, blokující odesílání a přijímání dat, obsluhu přerušení a metody pro ladící výstup. Toto rozdělení jsem zvolil z důvodu lepší přehlednosti a výměnou třídy *ENC28J60_SPI* lze jednoduše změnit přístup, s jakým se přistupuje na SPI sběrnici.

Inicializaci vykonává metoda *void init()*. Při inicializaci se nastavuje rozdělení bufferu na přijímací (6599B) a vysílací část (1580B), nastavují je přijímací filtry na příjem rámců se správným CRC, unicastových rámců a rámců, které odpovídají ARP zprávě. Dále se inicializuje MAC a PHY na Half-Duplex režim a závěrem se povolí vybrané zdroje přerušení.

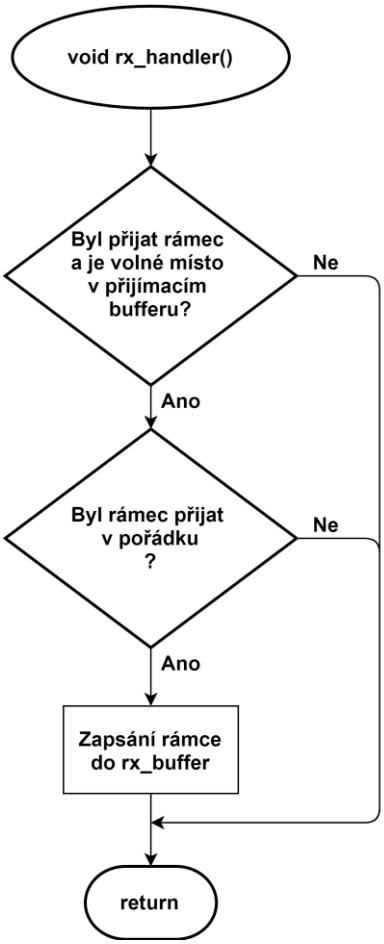
ENC28J60 má velké množství vad, které jsou popsány v erratech [16] a musí se brát v úvahu při inicializaci i při další komunikaci. Implementoval jsem všechny potřebná řešení pro mou revizi čipu B7. Tato revize je zatím poslední a měly by ji mít všechny prodávané čipy. Při programování a zprovozňování komunikace s tímto čipem jsem se inspiroval u projektu *Tuxgraphics NTP clock with DHCP client, version 2.X* [17], který mi pomohl vyřešit některé problémy, které neměly úplně zjevné řešení.

Pro příjem a odesílání ethernetového rámce využívám externí přerušení od *U3*, takže mikrokontrolér nemusí čekat, až se rámec odešle a může mezitím obsluhovat další požadavky. Používám přerušení při příjmu rámce, změně stavu připojení a úspěšném odeslání rámce. Obsluha přerušení *void interrupt_handler()* nastavuje příznak *pending_interrupt*, který se kontroluje v hlavní smyčce metodou *void get_interrupt_source()*, která zkontroluje jednotlivé bity zdrojů přerušení a podle nich nastaví jednotlivé příznaky zdrojů přerušení. Tyto příznaky slouží pro další podprogramy.

4.13 Síťový stack

Jednoduchá a minimální implementace síťového stacku je naprogramována ve třídě *NixieStack*. Podporuje protokoly ARP, ICMP, UDP a SNTP. Na začátku programu je potřeba nastavit lokální IP adresu, masku sítě a IP adresu brány.

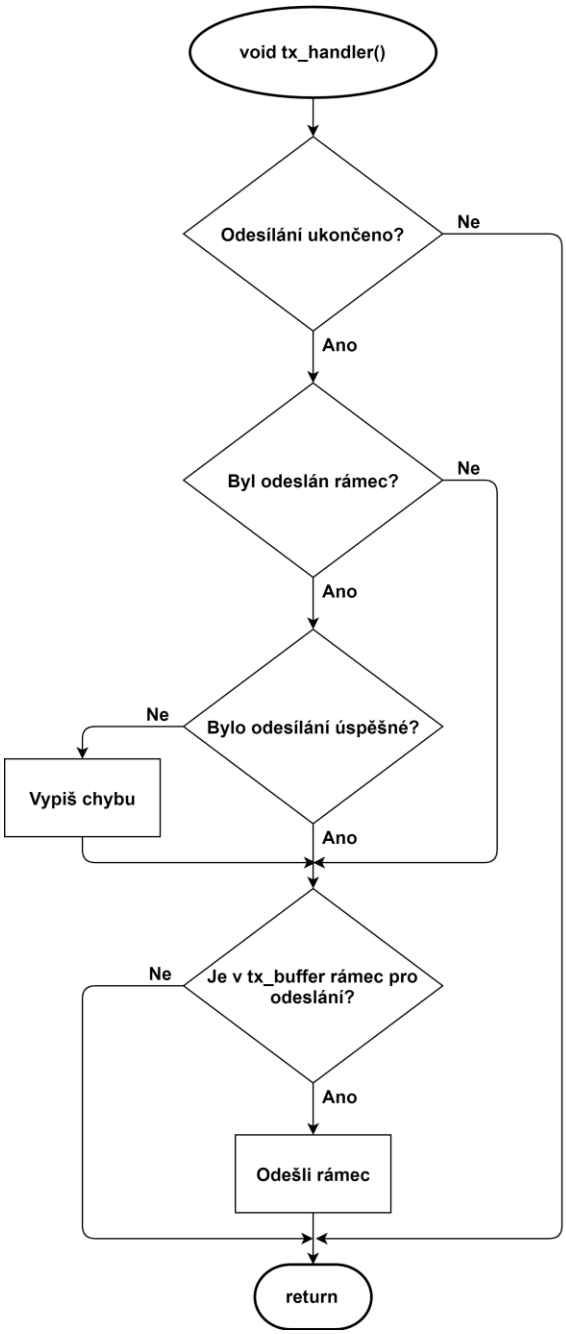
4.13.1 Příjem dat



Obrázek 4.6: Vývojový diagram příjmu ethernetových rámců.

Příjem dat obsluhuje metoda *void rx_handler()*, která kontroluje, jestli byt přijat nový rámec. Pokud ano a pokud je místo v přijímacím bufferu, tak se zkontroluje status vektor přijatého rámce a pokud byl rámec přijat správně, tak se uloží to přijímací bufferu *rx_buffer*.

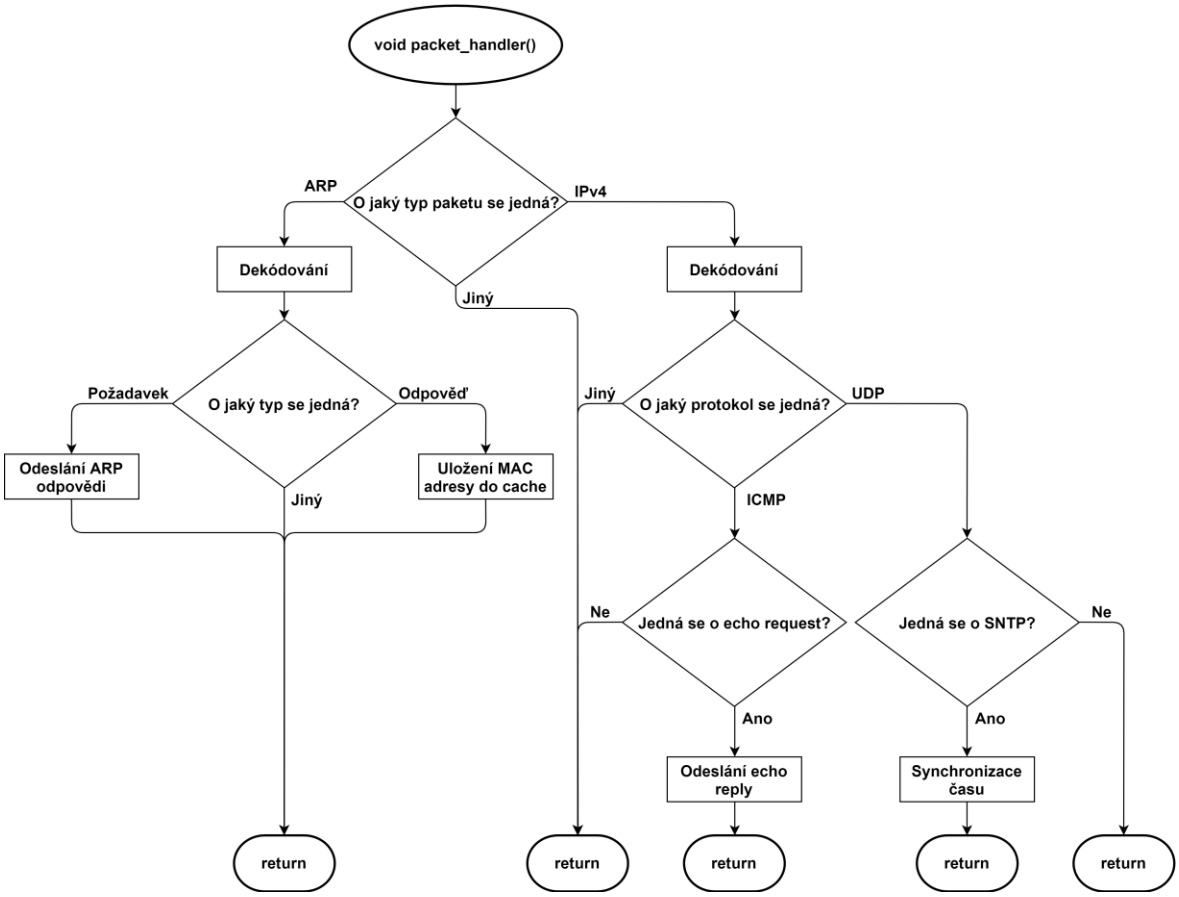
4.13.2 Odesílání dat



Obrázek 4.7: Vývojový diagram odesílání ethernetových rámců.

Nejprve se kontroluje, jestli se odesílá rámec, pokud ne, tak pokud byl právě rámec odeslán, tak se zkontroluje jeho status vektor a v případě, že rámec nebyl odeslán správně, tak se po UARTu odešle oznámení o chybě. Dále se zkontroluje, jestli je v odesílacím bufferu *tx_buffer* nový rámec, který čeká na odeslání. Pokud ano, tak se odešle.

4.13.3 Zpracování paketů



Obrázek 4.8: Vývojový diagram zpracování paketů.

Přijatý ethernetový rámec se dekóduje a podle typu se přijatá data předávají do dalších tříd, které obsluhují daný typ protokolu. Implementovány jsou protokoly síťové vrstvy ARP a IPv4, transportní vrstvy ICMP a UDP a z aplikační vrstvy je implementován SNTP protokol.

4.13.4 Ethernetový rámec

Tabulka 4.1: Ethernetový rámec.

MAC adresa cíle	MAC adresa zdroje	Délka/typ	Data	CRC
6B	6B	4B	46 – 1500B	4B

Ethernetový rámec obsahuje:

- Cílovou MAC adresu – pokud se komunikuje po lokální síti, tak se použije MAC adresa cíle, pokud se rámec posílá – do jiné sítě, tak se použije MAC adresa brány (routeru)
- Zdrojovou MAC adresu
- Délku/typ – pokud je menší než 1500, tak se jedná o délku dat, pokud je větší, tak udává typ přenášeného protokolu (IPv4 – 0x0800, ARP – 0x0806)
- Data
- CRC

Délku přijatých dat určují podle počtu přijatých bajtů, od kterých odečtu délku hlavičky.

4.13.5 ARP

Tento protokol se používá k zjištění MAC adresy k požadované IP adrese a je obsluhován třídou *ARP*. ARP paket má následující strukturu:

- Typ hardware 2B – v případě Ethernetu 0x0001
- Typ protokolu 2B – v případě IPv4 0x0800
- Délka hardware 1B – počet bajtů MAC adresy (6)
- Délka protokolu 1B – počet bajtů IP adresy (4)
- Operační kód 2B – typ zprávy
    - 0x0001 ARP požadavek
    - 0x0002 ARP odpověď
- MAC adresa odesílatele
- IP adresa odesílatele
- MAC adresa příjemce
- IP adresa příjemce

Pokud je potřeba zjistit MAC adresu požadované IP adresy (MAC adresa je požadována ethernetovým rámcem), tak se paket vyplní následovně. MAC adresa odesílatele je lokální MAC adresa. IP adresa odesílatele se vyplní buď lokální IP adresou, nebo pokud není přidělena lokální IP adresa, tak se použije adresa 0.0.0.0. MAC adresa příjemce je neznámá a tak se použije adresa 00:00:00:00:00:00. IP adresa příjemce je IP adresa, ke které se hledá MAC adresa. Ethernetový rámec se odešle na broadcastovou MAC adresu FF:FF:FF:FF:FF:FF. Sestavení tohoto paketu řeší metoda *void assembly_request(const IP& ip, Ethernet_payload& payload)*.

Pokud na síti existuje zařízení s požadovanou IP adresou, tak sestaví ARP odpověď, která je vyplněna následovně. MAC odesílatele je hledaná MAC adresa. IP adresa odesílatele je jeho IP adresa (k té se hledala MAC adresa). MAC a IP adresy příjemce jsou adresy odesílatele požadavku. Ethernetový rámec s tímto paketem se odešle na unicastovou adresu, odkud přišel požadavek. Sestavení tohoto paketu řeší metoda *void assembly_reply(const IP& requested_ip, const MAC& requested_mac, const IP& ip, const MAC& mac, ethernet_payload& payload)*.

Aby se při odeslání každého IP paketu na stejnou IP adresu nemusela tímto způsobem hledat MAC adresa, tak jsem implementoval jednoduchou cache, ve které je uloženo posledních 20 IP adres s odpovídajícími MAC adresami.

4.13.6 IPv4

Pro komunikaci mezi jednotlivými zařízeními na síti a mezi sítěmi se využívá IP (Internet Protokol). Tento protokol adresuje jednotlivá zařízení podle IP adres. IPv4 má adresy dlouhé 4B a IPv6 je má dlouhé 16B. Implementoval jsem pouze IPv4, který je dnes ještě běžně používaný, i když by už měl být dávno nahrazen IPv6. IP pakety obsluhuje třída *IP_protocol*. IP paket má následující strukturu:

- IHL 4b – délka hlavičky
- Verze 4b – verze IP (4 nebo 6)

- TOS 1B – typ služby
- TL 2B – celková délka paketu
- ID 2B – identifikace fragmentovaných zpráv
- Offset 2B – offset fragmentované zprávy
- TTL 1B – Time to Live
    - Počet směrovačů, přes které může paket projít, než se zahodí (při průchodu směrovačem se TTL o 1 sníží)
- Protokol 1B – typ přenášeného paketu
    - 0x01 ICMP
    - 0x06 TCP
    - 0x11 UDP
- Checksum 2B – kontrolní součet hlavičky
- Zdrojová IP adresa 4B
- Cílová IP adresa 4B
- Data

Metoda *bool decode(const ethernet_payload& rx_payload, ethernet_payload& tx_payload)* dekóduje příchozí pakety a předává přijatá data dalším třídám podle typu protokolu.

Metoda *void assembly(IP_packet* packet, const IP& dst_ip, uint16_t& length, uint8_t protocol)* slouží k sestavení IP paketu. Při sestavování IP paketu je zapotřebí spočítat kontrolní součet hlavičky. Funkci pro tento výpočet jsem převzal z [18], jedná se o funkci, která je optimalizovaná na rychlost výpočtu.

4.13.7 ICMP

Protokol ICMP (Internet Control Message Protocol) slouží k hlášení chyb na sítí, k diagnostice a pro směrování paketů. Implementoval jsem pouze malou část tohoto protokolu a to příkazy *echo request* a *echo reply*, které používá příkaz ping, kterým se běžně zjišťuje, jestli je dané zařízení připojeno do sítě. ICMP obsluhuje třída

ICMP_protocol. ICMP echo paket má následující strukturu:

- Type 1B – typ ICMP paketu
  - 0x00 echo reply
  - 0x08 echo request
- Code 1B – subtyp ICMP paketu
- Checksum 2B – kontrolní součet celého paketu
- ID 2B – identifikátor, který spojuje požadavek a odpověď
- Sequence 2B – číslo pokusu
- Data – nespecifikována

Pokud je přijat ICMP paket, tak je zpracován metodou *bool decode(const uint8_t* rx_data, const uint16_t length, uint8_t* tx_data)*, která kontroluje, jestli se jedná o *echo request*, pokud ano, tak odešle odpověď *echo reply*.

Testoval jsem, jak dlouho trvá přijetí, zpracování a odeslání zprávy mým zařízením pomocí nástrojem ping na lokální síti a průměrná doba odpovědi mého zařízení byla 5ms.

4.13.8 UDP

Pro komunikaci jednotlivých služeb, které jsou spuštěné na zařízení, slouží UDP (User Datagram Protocol). Tento protokol slouží k přenášení nekritických dat, protože nezaručuje, že příjemce zprávu obdrží a ani že zprávy dojdou ve stejném pořadí, ve kterém byly odeslány. Výhodou tohoto protokolu je úspora přenášených dat a přenos zprávy je rychlejší. Problém s nezaručením doručení řeší protokol TCP (Transmission Control Protocol). Protože hlavním cílem mého zařízení je synchronizace času z NTP serveru, tak jsem implementoval pouze UDP, který využívá NTP protokol. Aby více aplikací mohlo používat UDP, tak UDP obsahuje porty a každá aplikace poslouchá a odesílá na určitém portu a tím je zaručeno, že zpráva dojde vždy k té správné aplikaci. Obsluhu UDP protokolu zajišťuje třída *UDP_protocol*

a struktura UDP paketu vypadá následovně:

- Zdrojový port 2B – port, ze kterého byl paket odeslán
- Cílový port 2B – port, na který má paket přijít
- Délka 2B – délka celého paketu
- Checksum 2B – nepovinný kontrolní součet celého paketu
- Data – data, která jsou předána aplikaci

Metoda *void decode(const uint8_t* udp_data, const uint16_t length)* předává přijatá data aplikaci, která je určena cílovým portem. V mém případě to je tedy port 123 pro NTP.

4.13.9 SNTP

V počítačových sítích se pro synchronizaci času běžně používá NTP (Network Time Protocol), který umožňuje synchronizaci až na tisíciny vteřiny. Organizace provozují vlastní NTP server, nebo se využívá některého z veřejně dostupných. NTP servery jsou řízeny podle GPS nebo podle některých přesných hodin (celsiové, rubidiové). Klient postupně pošle několik požadavků na NTP server se svým lokálním časem. Server mu odpoví svým lokálním čase a časem přijetí zprávy. Z těchto informací si klient synchronizuje čas i s korekcí doby, kterou cestoval požadavek a odpověď po síti. Tento proces se několikrát opakuje, dokud nemá klient sesynchronizovaný čas na požadovanou přesnost.

Jelikož v mé aplikaci mi stačí přesnost na 1 sekundu, tak jsem neimplementoval celý NTP protokol, ale jen jednodušší variantu SNTP verze 4 (Simple Network Time Protocol), která je popsána v RFC 4330 [19]. Pro získání času se odešle jen jedna žádost a aktuální čas se přečte přímo z odpovědi a neprovádí se žádná další korekce.

NTP pracuje s časem ve formátu 4B celého počtu sekund od 1. 1. 1900 a 4B zlomku sekundy. Po jednoduchém přepočtu je tedy možné použít celý počet sekund

a aktualizovat s ním proměnou *seconds*, která obsahuje systémový čas. Čas se synchronizuje po startu zařízení a poté v 15 minutových intervalech.

5 Další možné funkce

5.1 DHCP klient

Nyní je IP adresa zařízení napevno uložená v kódu pro mikrokontrolér, což velmi omezuje použití zařízení, pokud by se připojovalo do různých sítí. Do zařízení by bylo možné naprogramovat DHCP (Dynamic Host Configuration Protocol) klienta, který by po připojení získal všechny potřebné informace o síti včetně IP adresy pro zařízení, které by tak mohlo bez problémů fungovat ve všech sítích, které obsahují DHCP server.

5.2 Webový server

Zařízení je připojeno do internetu a bylo by ho možné ovládat přes webový prohlížeč počítače nebo mobilního telefonu. Bylo by zapotřebí naprogramovat TCP protokol, po kterém se přenáší HTTP zprávy a musel by se implementovat jednoduchý webserver, který by mohl například poskytovat stránku, na které by se zobrazovaly provozní parametry zařízení, a některé parametry by bylo možné nastavovat. Mikrokontrolér má dostatek programové i datové paměti, takže by to neměl být problém.

5.3 Telnet

Aby se ladící výstup nemusel číst před UART a UART<->USB převodník, tak by mohl být implementovaný protokol telnet, na který by se dalo připojit z počítače, kam by se odesílaly zprávy ze zařízení a případně by se textovými příkazy daly měnit parametry zařízení. Lepší by byla implementace SSH protokolu, který slouží ke stejnému účelu jako telnet, ale komunikace je šifrována. Ale právě šifrování je bez hardwarové akcelerace velmi náročné na výpočetní výkon a je otázka, jestli by to mikrokontrolér zvládl upočítat tak, aby komunikace byla plynulá.

5.4 NTP klient

Pro přesnější synchronizaci by mohl být implementovaný celý NTP protokol. Stačilo by pouze změnit požadavek na NTP server a naprogramovat algoritmus pro počítání korekce přijatého času.

5.5 IPv6

Může se stát, že bude zařízení připojeno do sítě, kde funguje jen IPv6 protokol a na takové síti by zařízení nefungovalo. Řešením by tedy byla implementace IPv6 protokolu, který v budoucnu plně nahradí IPv4.

5.6 Zobrazování obecných čísel

Na digitronovém displeji by mohla být možnost zobrazovat obecná čísla, která by byla posílána z počítače nebo přes počítačovou sít. Zařízení by poté sloužilo i jako displej uživatelských hodnot.

6 Závěr

Cílem této práce bylo navrhnout, zhotovit a naprogramovat moderně řešené digitronové hodiny, které používají internetový NTP server pro synchronizaci času. Navrhnul jsem zapojení, které řídí mikrokontrolér ARM Cortex-M0+ a dále obsahuje zdroj pro elektroniku, zvyšující měnič pro digitrony, digitronový displej, který zobrazuje čas, datum a okolní teplotu, měření elektrických a neelektrických veličin a ethernetový kontrolér. Výpočetní výkon mikrokontroléru se ukázal být dostatečný pro obsluhování všech potřebných periférií a funkcí programu. Práce splnila všechny požadavky zadání.

I přes některé problémy, na které jsem narazil při programování periférií mikrokontroléru a ethernetového kontroléru, se mi úspěšně podařilo hodiny zprovoznit i se spolehlivou synchronizací času. Někdy jsem sice musel použít jiný přístup, než jaký jsem plánoval, abych obešel některá omezení, se kterými jsem nepočítal. V zapojení jsou například celkem zbytečné dolní propusti u měření napětí a proudu na výstupu zvyšujícího měniče, které omezují možnosti regulace tohoto měniče. Desku plošného spoje jsem nenavrhl ideálně a některé rychlé signály ruší signály pro A/D převodník a externí krystalový oscilátor 32,768kHz. Nešťastná chyba je špatně navržený napěťový dělič, který měl zamezit svícení neaktivních číslic v digitronech. Bez něj se občas na některých digitronech rozsvěcí číslice, které svítit nemají.

Nejsou to ale problémy, které by zabraňovaly správné funkci zařízení. Zařízení jsem testoval několik desítek hodin a nezaznamenal jsem problémy.

Seznam literatury a informačních zdrojů

[1] **BOSWAU, Hans P.** *Signaling system and glow lamps therefor. Patent US 2142106* United States of America, 3. 1. 1939.

[2] **VEENEMAN, Dan.** Origins of the Nixie Tube. *DecodeSystems.* [Online] 26. 9. 2016. [Citace: 20. 5. 2017.] Dostupné z: http://www.decodesystems.com/nixie-history.html.

[3] **VINDING, Sannah.** Neon Application Information. *VCClite.* [Online] VCC, 10. 2. 2017. [Citace: 20. 5. 2017.] Dostupné z: https://vcclite.com/neon-application-information/.

[4] **ISO/IEC 7492-1:1994.** *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model. Second edition* Svýcarsko, 1994.

[5] **Lumberg.** Katalogový list NEB 21 R. *Lumberg.* [Online] 1. 3. 2017. [Citace: 20. 5. 2017.] Dostupné z: http://downloads.lumberg.com/datenblaetter/neb21r.pdf.

[6] **CANAL ELECTRONIC CO., LTD.** Kalalogový list SL19-123. *TME.* [Online] 25. 1. 2006. [Citace: 20. 5. 2017.] Dostupné z: http://www.tme.eu/cz/Document/8790ddf534d60e194ea4328cb05e27d2/sl19123.pdf.

[7] **ECE.** Katalogový list SL150-33. *ECE.* [Online] 20. 5. 2015. [Citace: 20. 5. 2017.] Dostupné z: http://www.ece.com.tw/upload/products_1_20150520111324.pdf.

[8] **ALPHA & OMEGA SEMICONDUCTOR.** Katalogový list AOZ1280. *AOSMD.* [Online] 1. 8. 2011. [Citace: 20. 5. 2017.] Dostupné z: http://www.aosmd.com/res/data_sheets/AOZ1280CI.pdf.

[9] **MURATA.** Katalogový list LQH32PN2R2NN0L. *Murata.* [Online] 20. 5. 2017. [Citace: 20. 5. 2017.] Dostupné z: http://psearch.en.murata.com/inductor/product/LQH32PN2R2NN0%23.pdf.

[10] **HAUKE, Brigitte.** Aplikační zpráva - Basic Calculation of Boost Converter's Power Stage. *Texas Instruments.* [Online] 1. 1. 2014. [Citace: 20. 5. 2017.] Dostupné z: http://www.ti.com/lit/an/slva372c/slva372c.pdf.

[11] **Philips.** Katalogový list ZM1000. *Tube Tester.* [Online] 1. 3. 1970. [Citace: 20. 5. 2017.] Dostupné z: http://www.tube-tester.com/sites/nixie/dat_arch/ZM1000_ZM1000R.pdf.

[12] **STEELE, Jerry.** Aplikační zpráva - Extending Voltage Range of Current Shunt Monitor. *Texas Instruments.* [Online] 1. 2. 2006. [Citace: 20. 5. 2017.] Dostupné z: http://www.ti.com/lit/an/slla190/slla190.pdf.

[13] **Microchip.** Katalogový list ENC28J60. *Microchip.* [Online] 1. 1. 2012. [Citace: 20. 5. 2017.] Dostupné z: http://ww1.microchip.com/downloads/en/DeviceDoc/39662e.pdf.

[14] **Microchip**. Katalogový list 25AA02E48. *In: Microchip.* [Online] 1. 7. 2016. [Citace: 20. 5. 2017.] http://ww1.microchip.com/downloads/en/DeviceDoc/20002123F.pdf.

[15] **Atmel.** Katalogový list SAMD20G18. *Microchip.* [Online] 1. 9. 2016. [Citace: 20. 5. 2017.] Dostupné z: http://ww1.microchip.com/downloads/en/DeviceDoc/atmel-42129-sam-d20_datasheet.pdf.

[16] **Microchip.** Errata ENC28J60. *Microchip.* [Online] 1. 1. 2010. [Citace: 20. 5. 2017.] Dostupné z: http://ww1.microchip.com/downloads/en/DeviceDoc/80349c.pdf.

[17] **SOCHER, Guido.** Tuxgraphics NTP clock with DHCP client, version 2.X. *Tuxgraphics.* [Online] 16. 2. 2013. [Citace: 20. 5. 2017.] Dostupné z: http://tuxgraphics.org/electronics/201302/avr-ntp-clock-2x.shtml.

[18] **SHAW, Graham.** Calculate an Internet Protocol checksum in C. *microHOWTO.* [Online] b.r. [Citace: 20. 5. 2017.] Dostupné z: http://www.microhowto.info/howto/calculate_an_internet_protocol_checksum_in_c.html.

[19] **MILLS, David.** Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI. *IETF Tools.* [Online] 1. 1. 2006. [Citace: 20. 5. 2017.] Dostupné z: https://tools.ietf.org/html/rfc433.

[20] **WATOROWSKI, Tomasz.** IN-12 Nixie Clock. *Mighty Devices.* [Online] 29. 12. 2014. [Citace: 20. 5. 2017.] Dostupné z: http://mightydevices.com/?p=379.

[21] **KOZIEROK, Charles M.** *The TCP/IP Guide.* [Online] [Citace: 20. 5. 2017.] Dostupné z: http://www.tcpipguide.com/.

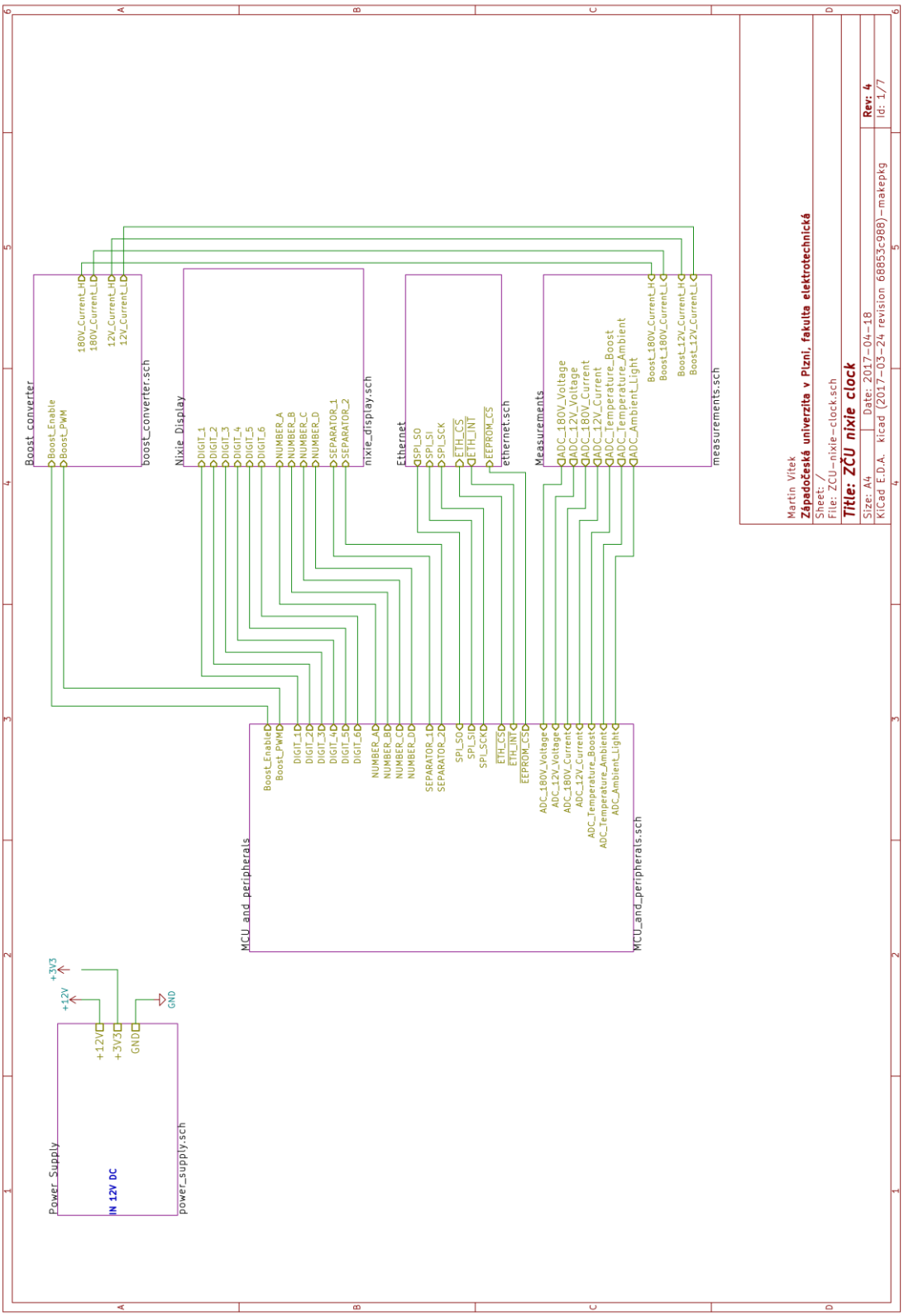
[22] **PLUMMER, David C.** RFC 826. *IETF Tools.* [Online] 1. 11. 1982. [Citace: 20. 5. 2017.] Sostupné z: https://tools.ietf.org/html/rfc826.

[23] **POSTEL, J.** RFC 768. *IETF Tools.* [Online] 28. 8. 1980. [Citace: 20. 5. 2017.] Dostupné z: https://tools.ietf.org/html/rfc768.

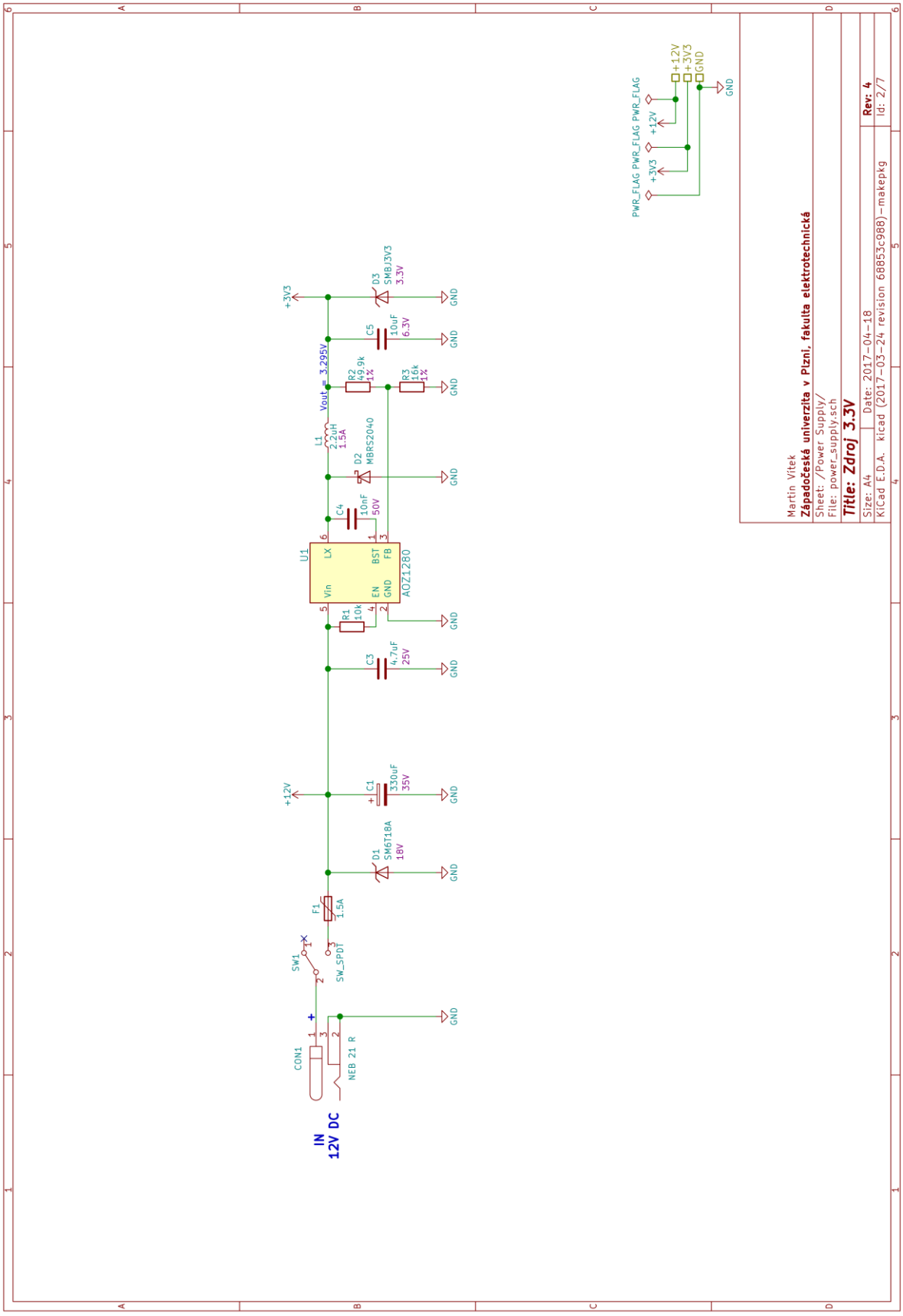
[24] **Information Sciences Institute University of Southern California.** RFC 791. *IETF Tools.* [Online] 1. 10. 1981. [Citace: 20. 5. 2017.] Dostupné z: https://tools.ietf.org/html/rfc791.

7 Přílohy

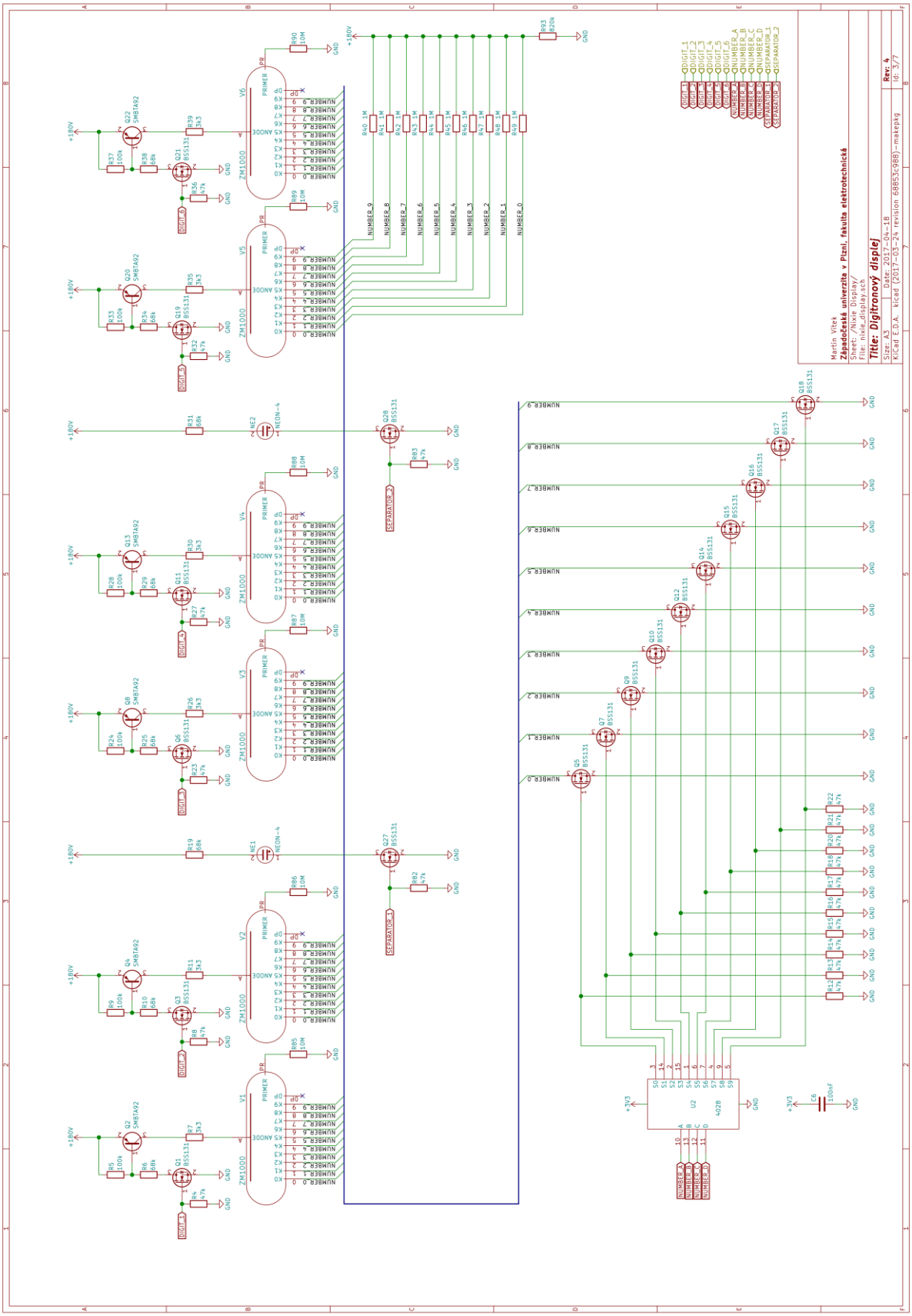
7.1 Schémata



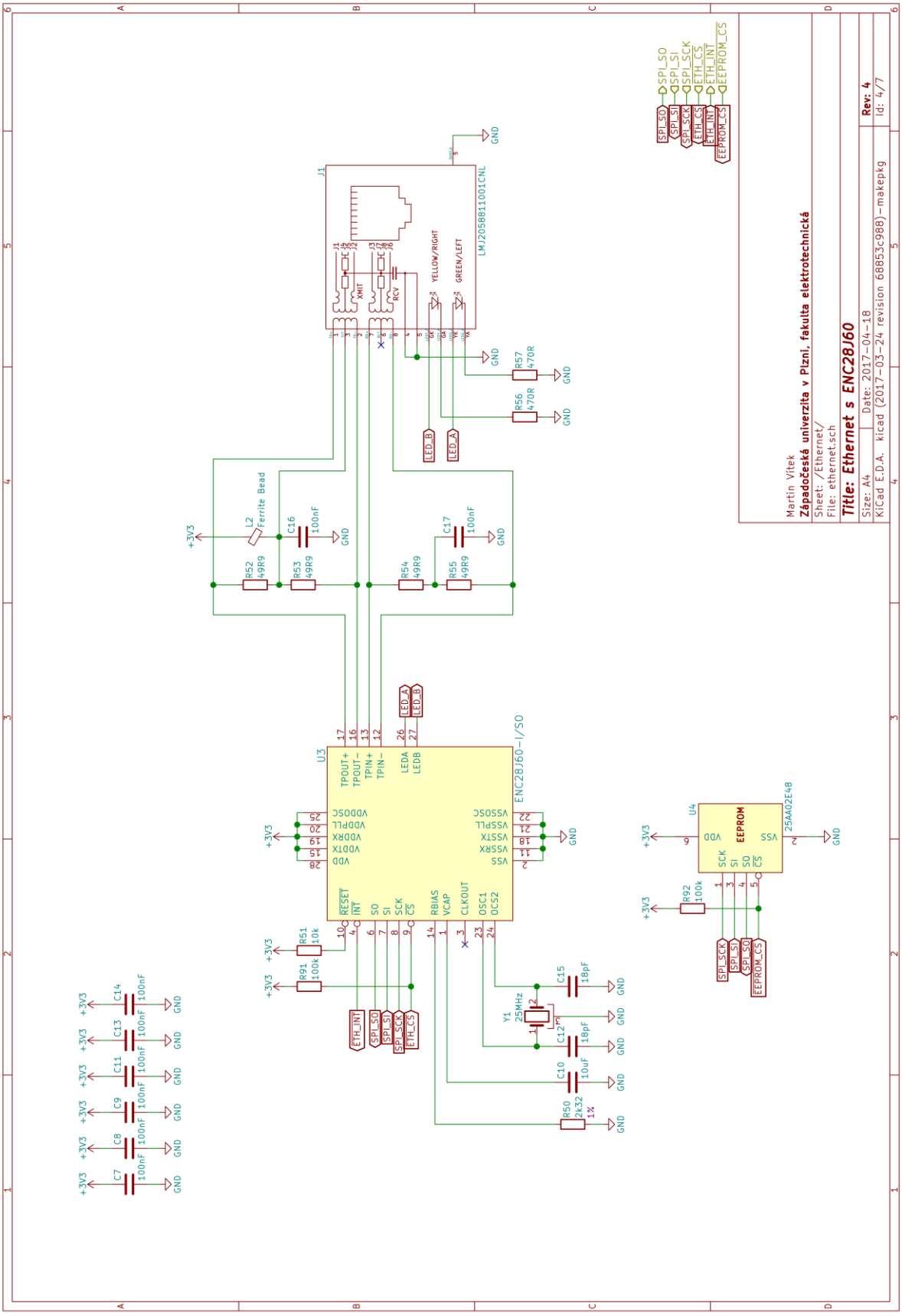
Obrázek 7.1: Blokové schéma.



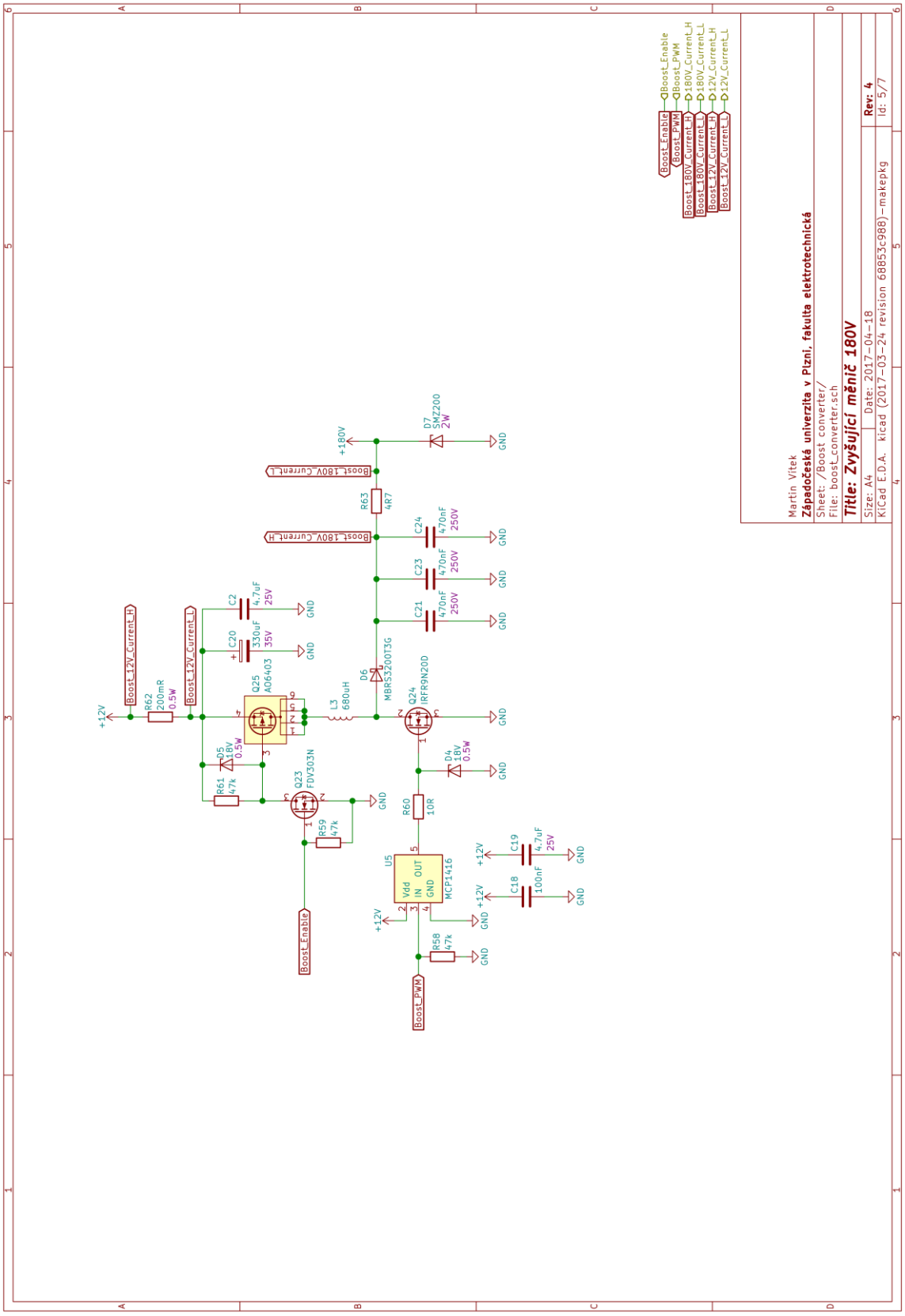
Obrázek 7.2: Schéma zdroje.



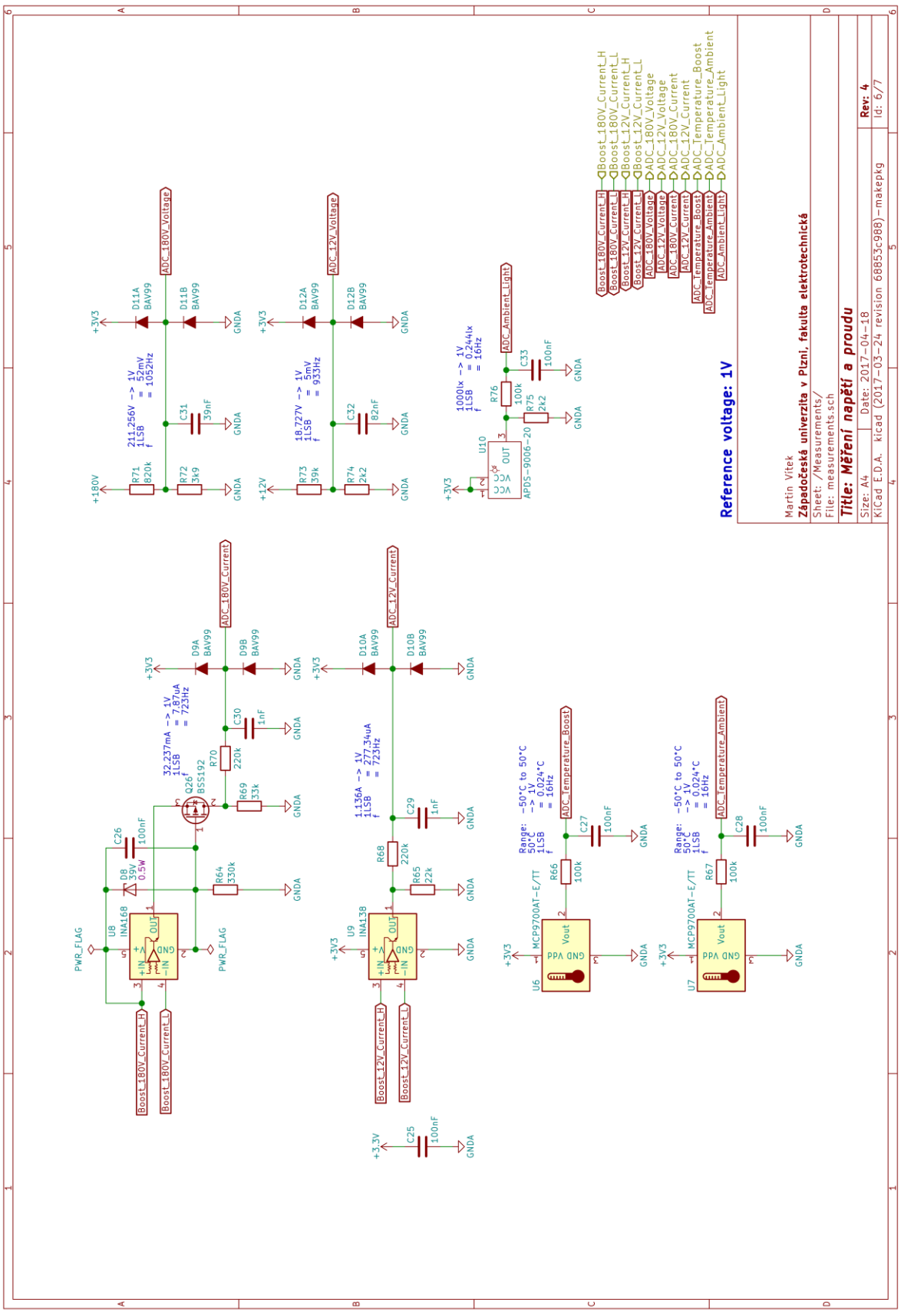
Obrázek 7.3: Schéma digitronového displeje.



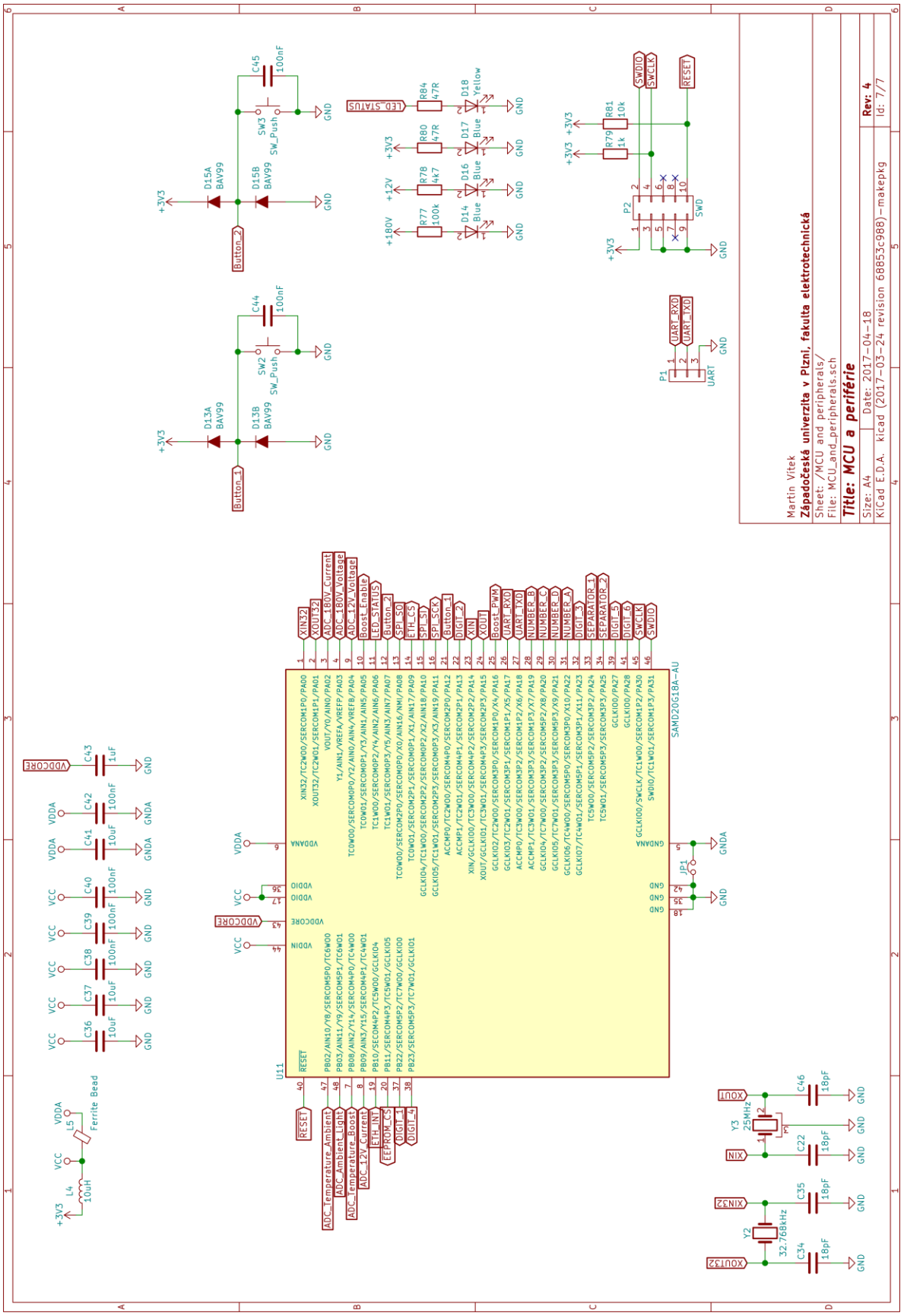
Obrázek 7.4: Schéma zapojení Ethernetu.



Obrázek 7.5: Schéma zvyšujícího měniče.

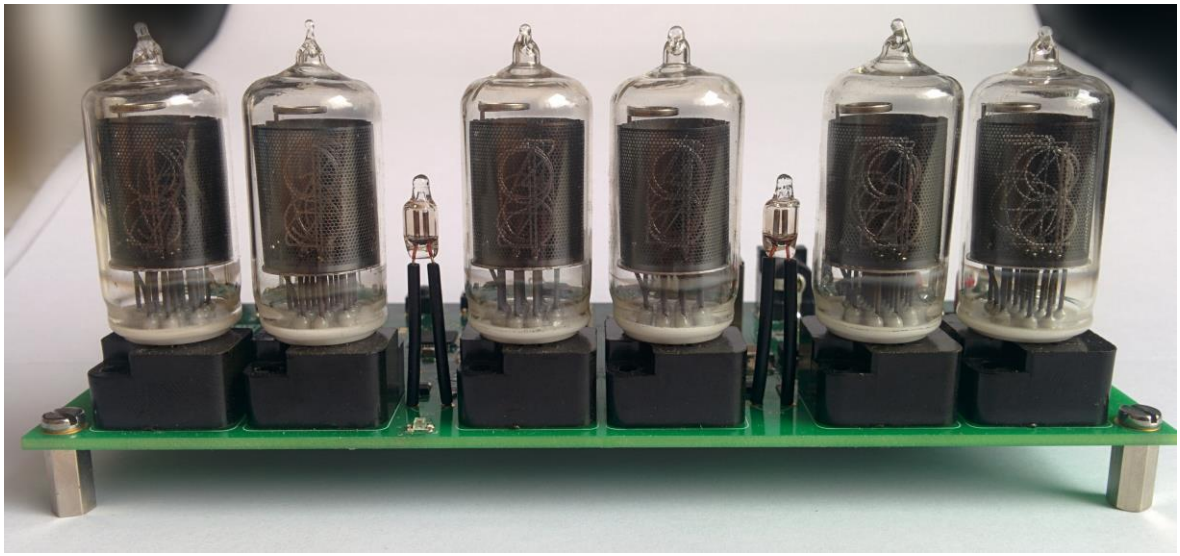


Obrázek 7.6: Schéma měření elektrický a neelektrických veličin.

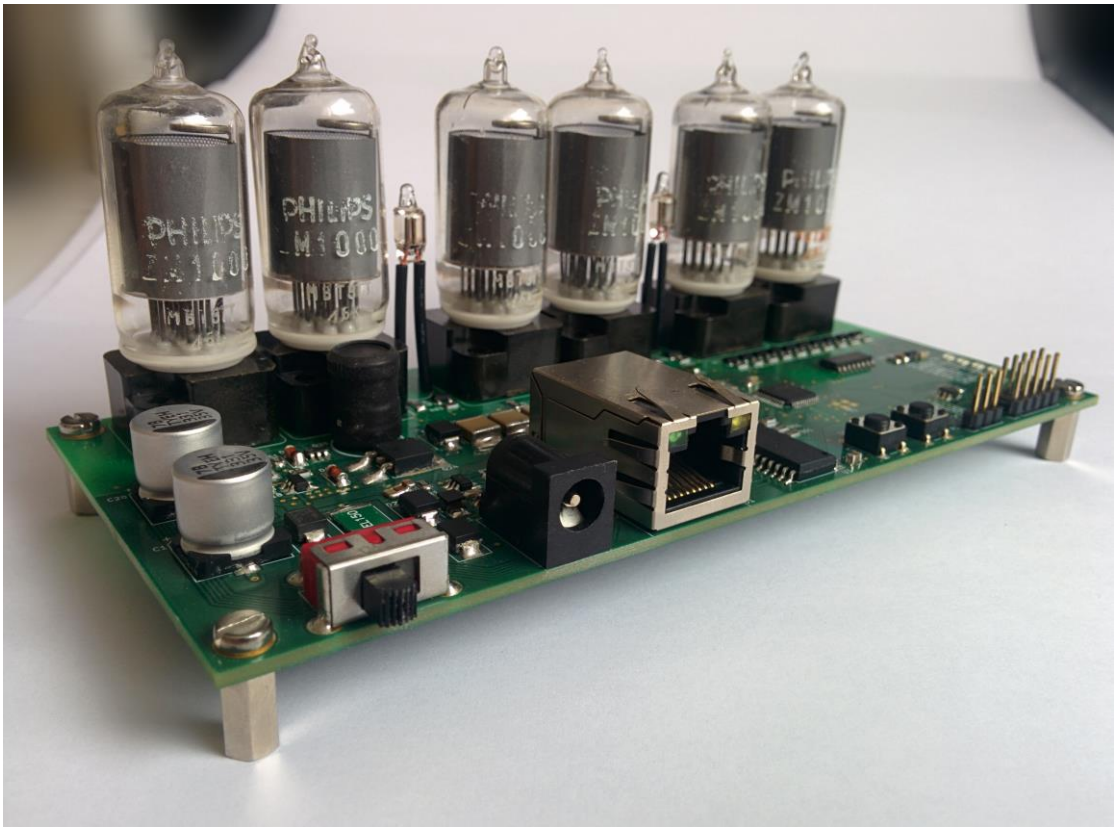


Obrázek 7.7: Schéma zapojení mikrokontroléru.

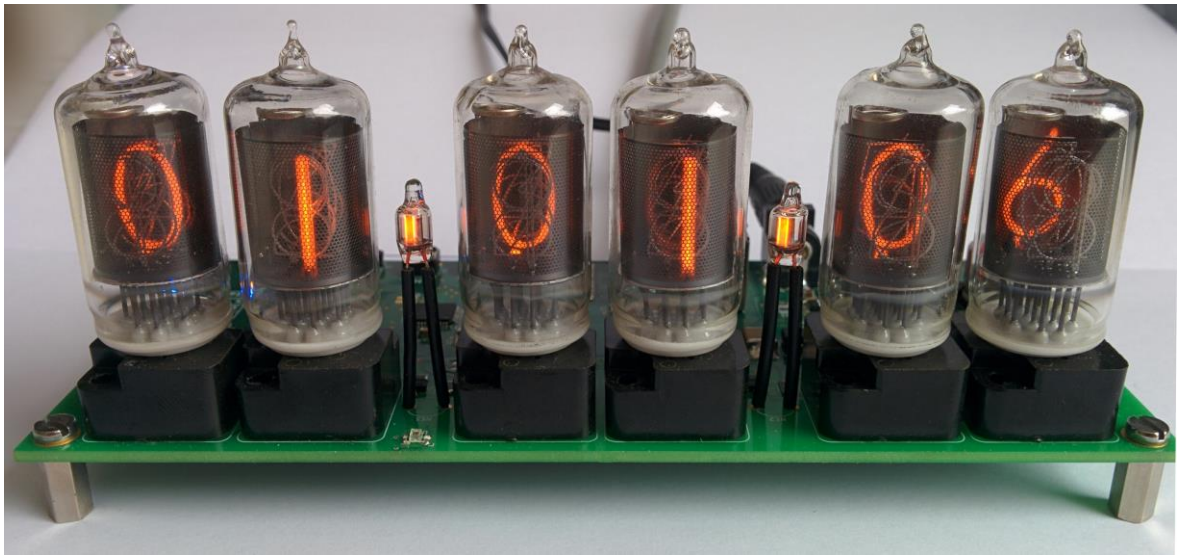
7.2 Fotografie



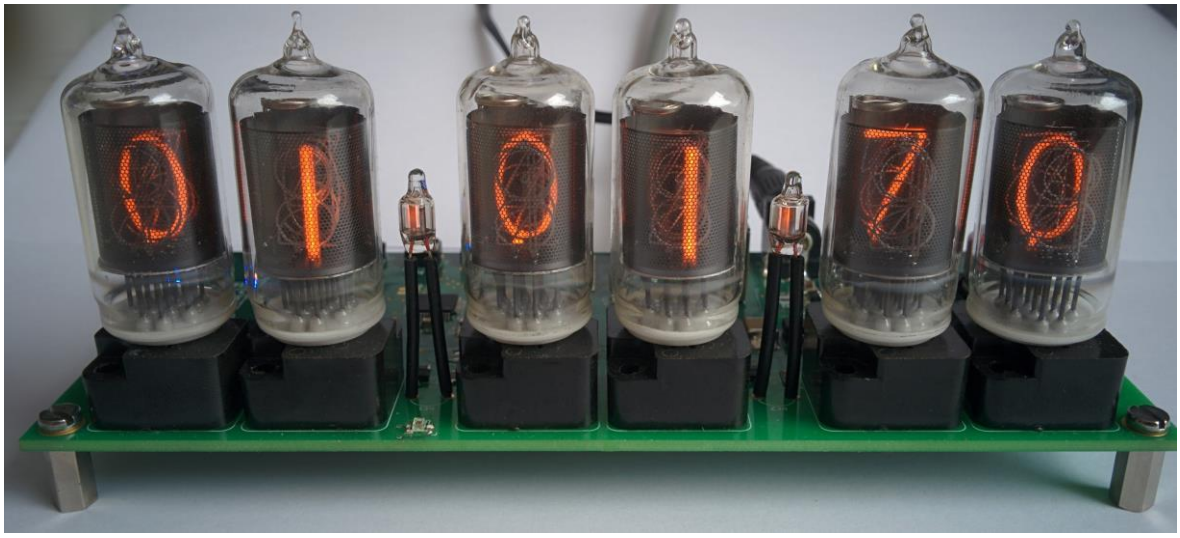
Obrázek 7.8: Přední strana digitronových hodin.



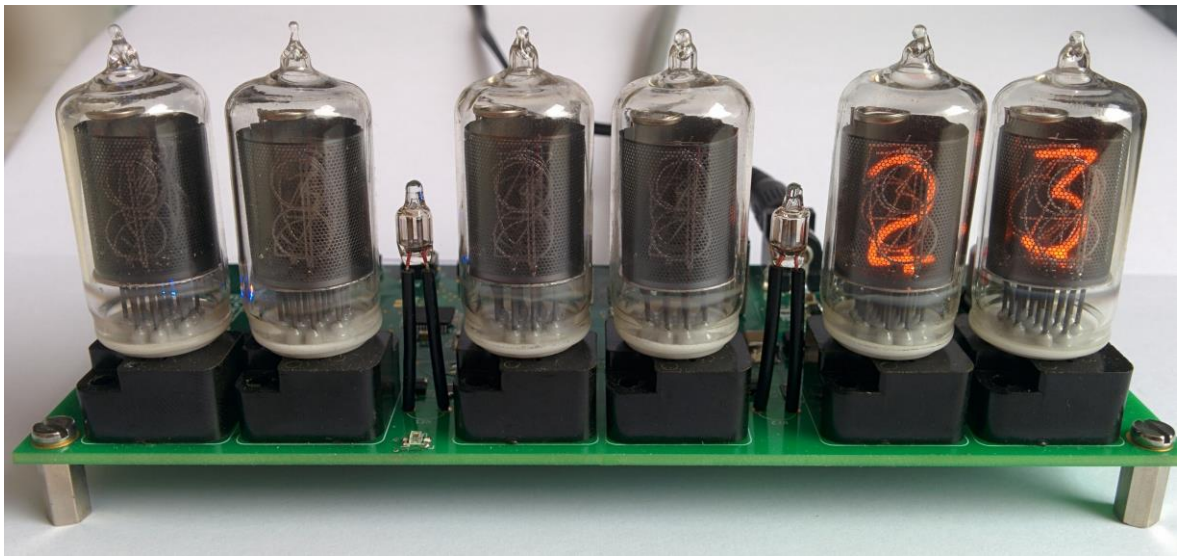
Obrázek 7.9: Zadní strana digitronových hodin.



Obrázek 7.10: Zobrazení času (1:1:06).



Obrázek 7.11: Zobrazení data (1. 1. 1970).



Obrázek 7.12: Zobrazení teploty (23°C).

7.3 Přílohy v elektronické podobě

Součástí elektronické verze na CD-ROM jsou následující položky:

1. Vlastní bakalářská práce ve formátu PDF.
2. Schémata ve formátu PDF.
3. Schémata a plošný spoj ve formátu programu KiCad.
4. Podklady pro výrobu plošného spoje ve formátu Gerber a Excellon.
5. Program pro mikrokontrolér včetně souborů projektu Atmel Studia 7.
6. Fotografie zařízení obsažené v tištěné příloze.