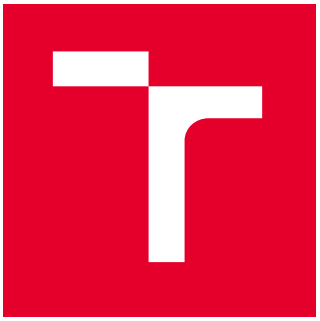


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ETHERNET NA RASPBERRY PI PICO S POUŽITÍM PIO

RASPBERRY PI PICO WITH ETHERNET

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Jan Zima**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. František Burian, Ph.D.**

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Jan Zima ***ID:*** 240474

Ročník: 3 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Ethernet na Raspberry pi pico s použitím PIO

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je navrhnout propojení jednodeskového mikropočítače Raspberry pi pico s čipem PHY realizující fyzickou vrstvu spojení s Ethernetovou sítí. Úkolem studenta není implementace kompletního TCP/IP stacku pro komunikaci po ethernetu, ale pouze datového generátoru, který dokáže odesílat naměřená data po ethernetové síti druhému počítači. Důraz je tedy kladen na správnou obsluhu signálů rozhraní MII, konfiguraci a detekci připojeného portu pomocí SNI/MDIO//I2C a v případě použití víceportového MAC i konfiguraci switche a směrování odesílaných packetů na konkrétní upstream port. Při realizaci není vyžadována implementace ARP protokolu, dostačuje komunikace na lokálním segmentu sítě.

1. Seznamte se s jednodeskovým mikropočítačem Raspberry pi pico
2. Seznamte se s principy ethernetové komunikace, protokoly UDP, IP, MAC, MII
3. Navrhněte schema propojení RP2040 s MAC čipem umožňující komunikaci pomocí (R)MII
4. Vytvořte program v PIO assembleru, který realizuje obousměrné rozhraní MII
5. Vytvořte testovací aplikaci, která umožňuje datagramovou komunikaci protokolem UDP/IP
6. Vytvořte program pro PC, který dokáže data přijímat.
7. Realizované komunikační spojení s počítačem demonstrujte.

DOPORUČENÁ LITERATURA:

RP2040 Assembly Language Programming: ARM Cortex-M0+ on the Raspberry Pi Pico. Apress Berkeley, CA, 2021. ISBN 978-1-4842-7753-9.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 22.5.2024

Vedoucí práce: Ing. František Burian, Ph.D.

Ing. Miroslav Jirgl, Ph.D.
předseda rady studijního programu

Abstrakt

Práce se zaměřuje na realizaci ethernetové komunikace použitím jednodeskového mikropočítače Raspberry Pi Pico a čipu fyzické vrstvy. Jedná se o obousměrnou komunikaci tohoto mikropočítače s druhým počítačem na lokální síti (LAN) použitím rozhraní MII. Mikropočítač využívá ethernetovou síť pro odesílaní dat, získaných ze snímačů vzdálenosti. Pro počítač, který přijímá data po ethernetové síti, je vytvořen jednoduchý program, kterým jsou přijatá data zobrazena. Práce se mimo jiné zabývá samotným návrhem desky plošných spojů, kde jsou jednotlivé komponenty propojeny.

Klíčová slova

Raspberry Pi Pico, KSZ8081MLX, PIO, Ethernet, MII

Abstract

This bachelor thesis focuses on the implementation of ethernet communication using microcomputer Raspberry Pi Pico and a physical layer chip. The communication is intended to be bidirectional between the microcomputer and other computer on a local area network (LAN) using the well-known MII interface. Microcomputer uses this network to communicate data which are acquired by distance sensors. A software which is outputting those acquired data to the console is implemented on the second computer. This thesis also focuses on a PCB design itself where individual electronic parts are connected making a functional circuit.

Keywords

Raspberry Pi Pico, KSZ8081MLX, PIO, Ethernet, MII

ZIMA, Jan. Ethernet na Raspberry Pi Pico s použitím PIO. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2024. 63 s., 2 s. příloh. Semestrální práce. Vedoucí práce: Ing. František Burian, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	Jan Zima
VUT ID studenta:	240474
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	Ethernet na Raspberry Pi Pico s použitím PIO

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

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

.

V Brně dne: 12. května 2024 ----------

podpis autora

Poděkování

Hlavní poděkování patří vedoucímu bakalářské práce Ing. Františku Burianovi, Ph.D. za jeho odbornou pomoc, ochotu vysvětlovat sebemenší detaily, ochotu předat jeho znalosti při konzultacích a za vedení celé bakalářské práce.

V Brně dne: 12. května 2024 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

Úvod

V dnešní době se člověk pravidelně setkává s množstvím zařízení, které poskytují možnost komunikace s dalším zařízením. Jedním typem takovéhoto propojení může být právě použití ethernetového kabelu UTP.

Příklad využití takovéhoto propojení v prosté domácnosti může být propojení tiskárny napřímo s PC (Personal Computer) nebo propojení těchto dvou zařízení za použití LAN (Local Area Network) portů routeru (směrovače), tím se stávají součástí tzv. lokální sítě. Na lokální síti může být mezi sebou propojeno několik zařízení použitím různých topologií.

V průmyslu lze komunikaci přes ethernet využít například propojením bezpečnostních kamer budovy nebo pro propojení komunikačních modulů výrobního procesu jako jsou PLC moduly.

Pro realizaci této komunikace je potřeba využít některé rozhraní. V mé práci využívám rozhraní MII (Media Independent Interface), jedná se o jedno ze základních a realizačně jednoduchých rozhraní. Rozhraní nabízí rychlost přenosu až 100 megabit za sekundu (Mbps).

Jednočipové počítače neboli mikrokontroléry integrují na jednom čipu kromě procesoru také paměti, řadič přerušení a složitější periférie. Mezi ně mohou být zařazeny grafické karty, A/D nebo D/A převodníky, čítače a časovače. Většinou nabízí několik různých periférií, kterými lze realizovat více druhů aplikací. Z pohledu použití se může jednat třeba o řízení zavlažovacích systémů zahrady. Výhoda jejich použití spočívá v nízké pořizovací ceně, menší paměťové a výpočetní nároky a jsou typické jejich malými rozměry.

V této práci se zaměřím na vytvoření modulu, který bude realizovat ethernetovou komunikaci použitím mikropočítače Raspberry Pi Pico. Tento mikropočítač je zvolen na základě netypické periférie PIO, kterou jeho kontrolér nabízí. Jedná se o periférii, která je přímo určená pro implementaci vysokorychlostních rozhraní (příkladem může být DVI). V mém případě periférii PIO využiji k implementaci ethernetového rozhraní MII. Dostupné ethernetové moduly jsou většinou pomalé, a proto se jeví výhodné použít právě zmíněný mikropočítač RPP, který je možné provozovat až na 133 MHz.

1. MIKROPOČÍTAČ RASPBERRY PI PICO

Jednodeskový mikropočítač Raspberry Pi Pico (dále jen RPP) je založený na bázi mikrokontroléru RP2040 od společností Raspberry Pi, pocházející z Velké Británie. Některé z vlastností mikropočítače:

- 2 x 133 MHz Arm Cortex M0+ procesor
- 264 kB SRAM čipové paměti
- 2 MB externí paměti FLASH (RP2040 podporuje práci až s 16 MB externí pamětí FLASH)
- DMA řadič
- Režim spánku / klidu pro snížení spotřeby
- 26 x multifunkčních GIPO pinů
- 3 x 12 bitový ADC
- 16 x PWM kanálů
- 8 x programovatelných I/O (PIO) stavových automatů

1.1 Pravděpodobně použité periférie

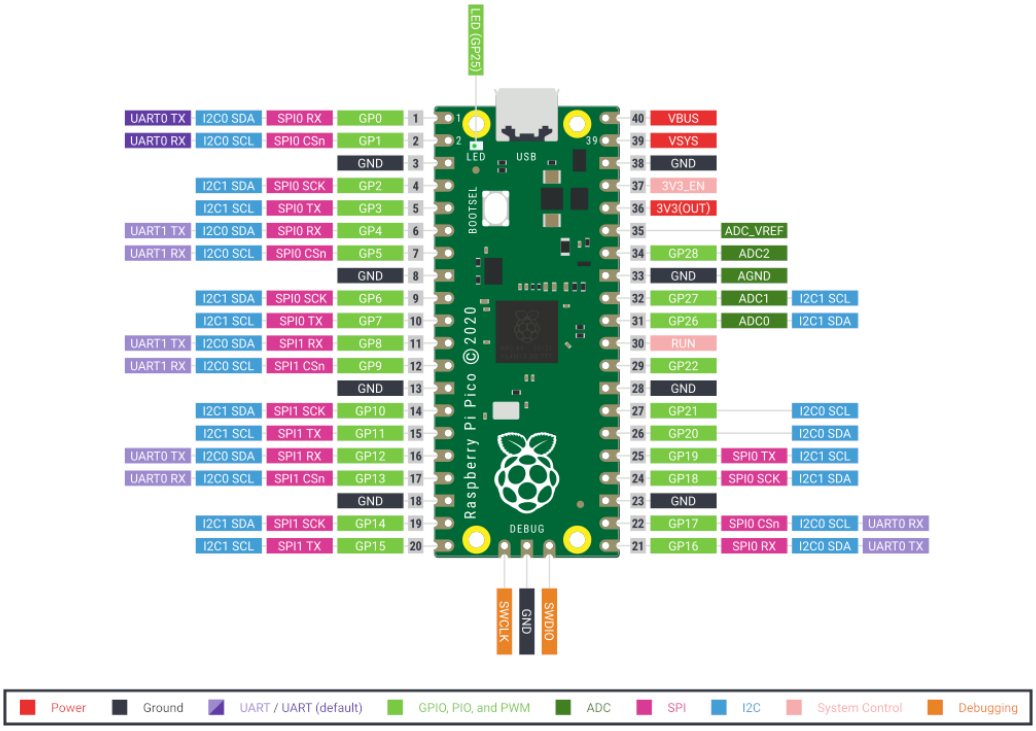
Periférie si lze představit jako přídavný hardware k procesoru. Tímto doplněním samotného procesoru, který vykonává pouze instrukce, o hardware jako čítače, hodiny reálného času nebo komunikační rozhraní USB apod. vzniká mikrokontrolér. Periférie tedy poskytuje možnost komunikace procesoru s jiným zařízením. [9]

Semestrální část práce se primárně zabývá rešerší. Z tohoto důvodu jsou uvedeny periférie, které budou nejspíše využity při pozdější implementaci softwaru. Tyto předpoklady jsou založeny na zkušenostech z jiných projektů, u kterých jsem pracoval s tímto mikrokontrolérem.

1.1.1 GPIO

Je periférie zpřístupňující digitální piny mikrokontroléru (komunikace s okolím). Slouží k připojení některé z vybraných periférií na fyzický pin kontroléru. Periférie poté nastavuje/sleduje digitální hodnoty (0/1) tohoto pinu. Tento pin může být propojen pouze s jedním signálem. Nelze na jeden pin propojit dva signály z dvou různých periférií v jeden okamžik. [1]

GPIO piny, na které lze jednotlivé periférie připojit (inicializovat je) jsou specifikované výrobcem mikrokontroléru na takzvaném „pinout“ diagramu.



Obrázek 1.1 Pinout mikrokontroléru Raspberry Pi Pico [1]

Další možností je přímé ovládání GPIO pinu. Uživatel může GPIO pin inicializovat na vstup / výstup a poté číst / nastavit jeho digitální hodnotu nebo připojit pull-down/up rezistory. [1]

1.1.2 ADC

Analogově digitální převodník se využívá pro převod extérního analogového signálu (například ze snímače s analogovým výstupem) na digitální. Ten je poté vhodný pro další zpracování procesorem. [1]

RPP poskytuje 12 bitový A/D převodník s vzorkováním až 500 ksps (kilo samples per second) a rozhraní pro řadič DMA (bude popsán později). Při nepřetržitém vzorkování umožňuje snímat až 4 vstupy (1 z těchto vstupů je integrovaný teplotní snímač na RP2040 tedy 3 extérní analogové signály). [1]

Obsahuje navíc vyrovnávací paměť typu FIFO (first in first out), díky které se jednotlivé vzorky uchovávají, a to v pořadí, v jakém byly pořízeny. Bez použití této paměti je na výstupu převodníku vždy pouze jedna aktuální digitální hodnota, která byla získána převodem aktuální analogové hodnoty. [1]

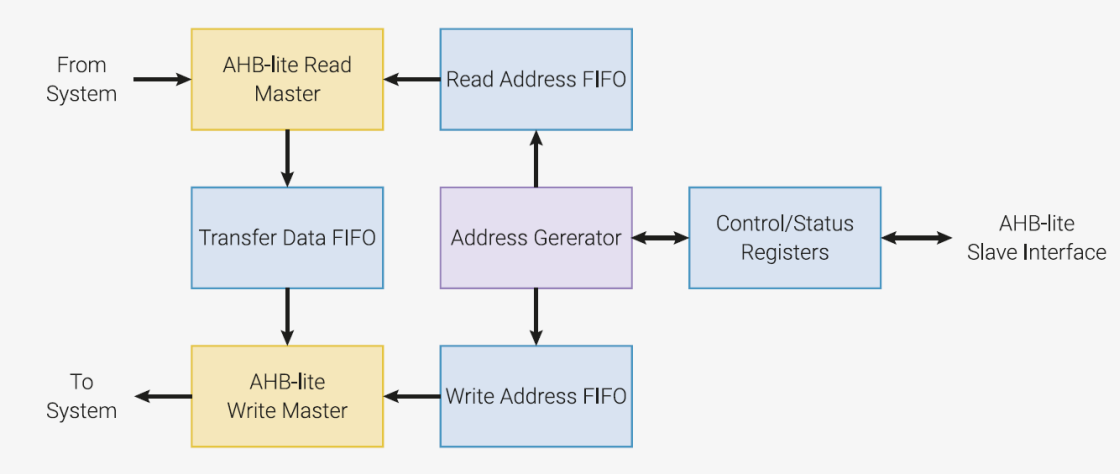
Převodník může generovat přerušení na základě zaplnění této paměti (hraniční hodnotu zaplnění paměti můžeme nastavit pomocí prahu (anglicky threshold). [1]

1.1.3 DMA

Řadič DMA se využívá k přenášení velkého množství dat mezi vyrovnávací pamětí periférií a RAM (random access memory) mikrokontroléru (obousměrně) nebo přesunem dat z jednoho místa v RAM paměti do druhého místa v RAM paměti.

Výhodou tohoto řadiče je větší propustnost dat oproti procesoru mikrokontroléru. DMA řadič umožňuje zároveň zápis a čtení až 32 bitových dat co každý hodinový cyklus. Je možné využít až 12 kanálů DMA přesunů zároveň, mezi kterými se tato propustnost dat patřičně rozdělí. [1]

Obrázek 1.2 znázorňuje architekturu DMA řadiče. V konfiguračním registru (Control register) programátor nastavuje adresy, mezi kterými se data budou přenášet, velikost těchto dat v bitech (u RP2040 je to 8 / 16 / 32 bitů) a další dílčí specifikace přenosu. Stavovým registrem (Status register) je možné sledovat stav práce DMA řadiče, například pokud je určitý DMA kanál zaneprázdněn (provádí převod dat) nebo jestli v něm nastala nějaká chyba. Read master poté provede jedno čtení z adresy, která je uchována v Read Address FIFO, každý hodinový cyklus. Obdobně write master provádí zápis. [1]

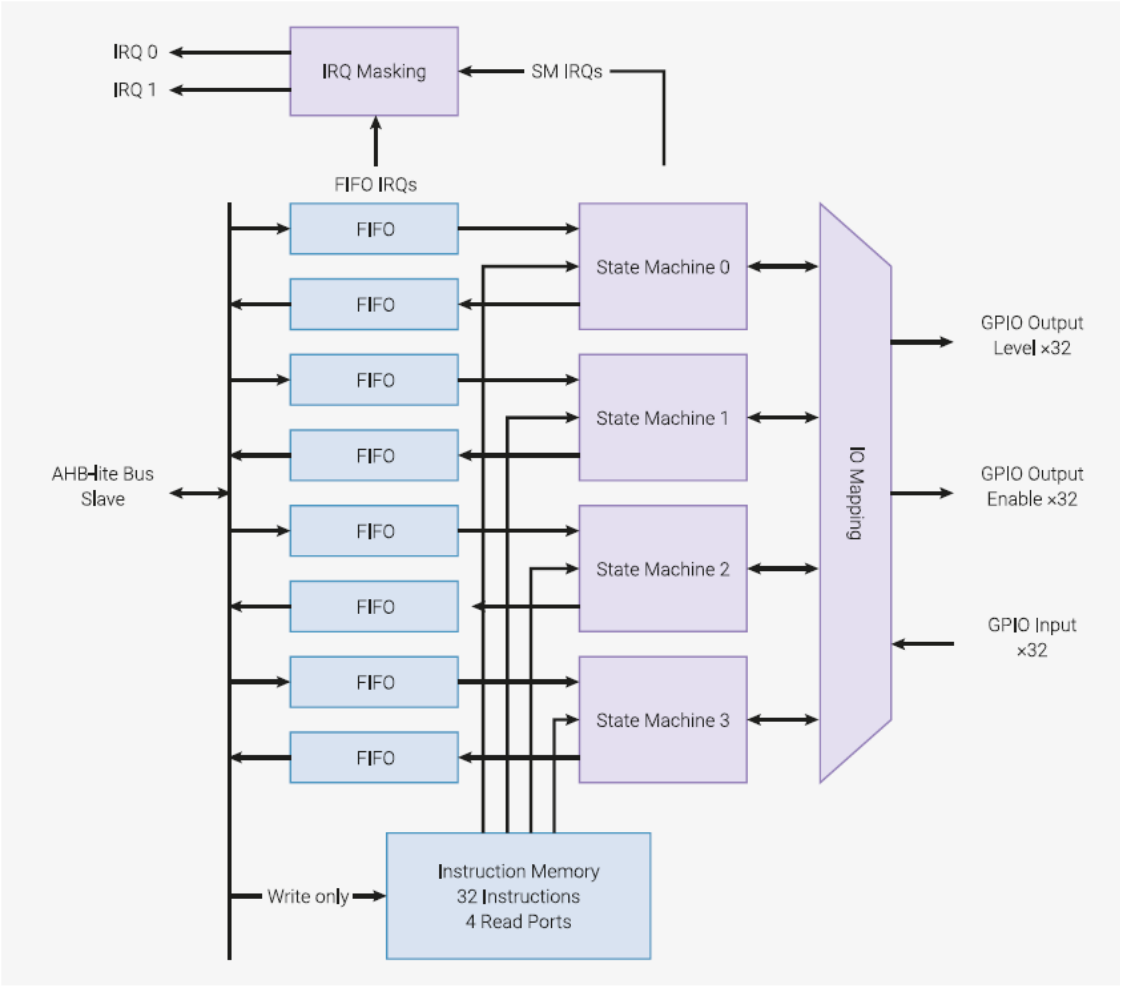


Obrázek 1.2 Architektura řadiče DMA [1]

Procesor tedy musí pouze nakonfigurovat kanál DMA (případně spustit jeho činnost, je-li to nutné) a dále se již o něj nestarat. Může tedy během činnosti DMA řadiče vykonávat jiný důležitý kód nebo vstoupit do režimu spánku pro úsporu energie.

1.2 PIO

Kontrolér obsahuje dva hardwarově totožné PIO (programmable input/output) bloky. Je to univerzální hardware, který podporuje IO rozhraní jako I2C, DVI, UART apod. Je možné si taktéž vytvořit jiné rozhraní dle libosti, třeba i takové, které neodpovídá žádnému standardu. Nebo pokud nejsou 2 UART instance kontroléru dostačující, lze si naimplementovat 3. takové rozhraní právě za pomocí PIO. [1]



Obrázek 1.3 PIO diagram [1]

PIO lze programovat obdobným způsobem jako procesor. PIO blok obsahuje 4 stavové automaty, které sekvenčně vykonávají sled instrukcí (program) z instrukční paměti. Tato paměť je společná pro všechny 4 stavové automaty. Ke každému automatu je připojen 4x32 bitový FIFO TX a FIFO RX. Lze jej také sloučit pouze do jednoho směru na 8x32 bitový FIFO. [1]

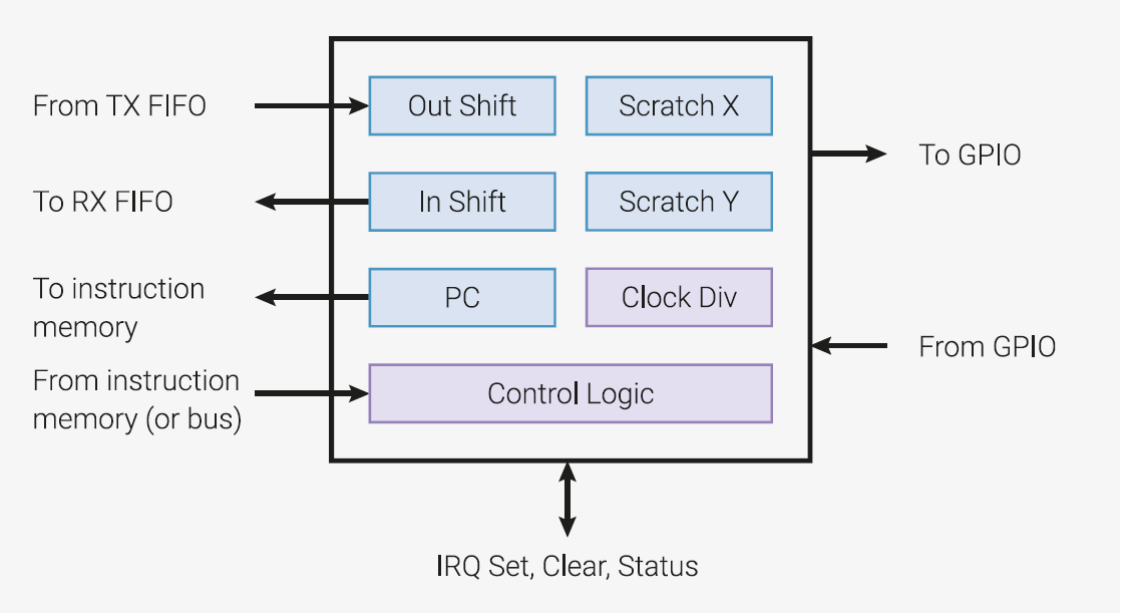
Tyto stavové automaty jsou specializované na I/O operace. Jejich činnost je jasně determinovaná a jejich časovaní je přesné. [1]

1.2.1 Programátorský model

Software nahraje program do instrukční paměti stavových automatů, nakonfiguruje stavový automat, jeho výstup na libovolný GPIO pin a poté spustí stavový automat. PIO programy lze vytvořit pomocí asembler kódu (tzv. „pioasm“). [1]

Po spuštění je interakce se stavovým automatem přes DMA, přerušení a konfigurační registry [1].

Každou z PIO instrukcí lze vykonat v jednom hodinovém taktu, pokud se nejedná o instrukci, která dodatečně pozastavuje své vykonání (WAIT instrukce čekající na splnění podmínky). Po dokončení instrukce lze taktéž specifikovat zpoždění až o 31 hodinových cyklů. [1]



Obrázek 1.4 Diagram stavového automatu PIO [1]

Na obrázku 1.4 je zobrazena vnitřní struktura jednoho stavového automatu PIO bloku. Obsahuje 4 malé registry.

Output Shift Register (OSR) slouží k načtení až 32 bitové hodnoty z TX FIFO pomocí **PULL** instrukce. Poté lze obsah OSR přesunout na požadovanou destinaci (není nutně pouze GPIO pin) pomocí **OUT** instrukce. Směr přesunu z OSR je možné nastavit v konfiguračním registru. [1]

Input Shift Register (ISR) zapisuje až 32 bitovou hodnotu do RX FIFO pomocí **PUSH** instrukce. Hodnoty, které zapisuje do tohoto RX FIFO jsou načteny **IN** instrukcí z některého zdroje (není nutně pouze GPIO pin). Směr, z kterého se bude ISR zaplňovat, lze nastavit v konfiguračním registru. [1]

Oba tyto registry mohou pracovat autonomně při nakonfigurování tzv. **AUTOPUSH** a **AUTOPULL** v konfiguračním registru. ISR / OSR provádí poté **PUSH / PULL** autonomně při dosažení přednastavené hodnoty „threshold.“ [1]

Scratch registry X / Y slouží jako destinace / zdroj pro instrukce **IN / OUT / SET / MOV** nebo je lze využít jako proměnné cyklu. [1]

Program Counter (PC) ukazuje na aktuální instrukci, která se vykonává. Podle této instrukce se řídí kontrolní logika (Control Logic) stavového automatu.

Clock Divider slouží pro zpomalení referenčního taktu systémových hodin. Stavové automaty PIO instancí jsou taktovány pomocí systémových hodin, ty lze softwarově

nastavit až na 133 MHz. Takt o 133 MHz však může být pro aplikaci až příliš vysoký. Pomalejšího vykonávání programu stanovým automatem lze dosáhnout přidáním zpoždění až o 31 cyklů nebo nastavit systémové hodiny na frekvenci, na které bude aplikace fungovat. Zpomalovat však celý mikrokontrolér kvůli samotné aplikaci nebo zbytečné přidávání zpoždění není úplně vhodné řešení. Proto je výhodné pouze změnit hodiny stavového automatu pomocí dělení systémových hodin. Z 100 MHz systémových hodin lze vytvořit 10 MHz hodiny stavového automatu pomocí nastavení dělícího poměru v konfiguračním registru na hodnotu 10. [1]

Toto řešení může být efektivní i z pohledu šetření instrukční paměti PIO bloku. Příkladem může být aplikace, která využívá 4 UART rozhraní o různé rychlosti. Bez možnosti dělení hodin stavových automatů by bylo pro realizaci nutné do instrukční paměti nahrát 4 totožné programy pouze s jinými hodnotami zpožďovacích cyklů. Pravděpodobně by však nebylo možné všechny 4 programy do paměti nahrát, neboť instrukční paměť je pouze jedna o velikosti 32 instrukcí a je sdílená pro všechny 4 stavové automaty. Vytvoří se tedy pouze jeden program, který bude totožný pro všechny 4 stavové automaty. Poté se každý stavový automat nastaví na jinou frekvenci pomocí dělení systémových hodin. Takové řešení je bezproblémové, všechny stavové automaty mohou souběžně vykonávat jeden program (jedna instrukce může být ve stejný moment prováděna všemi 4 automaty).

2. PRINCIP ETHERNETOVÉ KOMUNIKACE

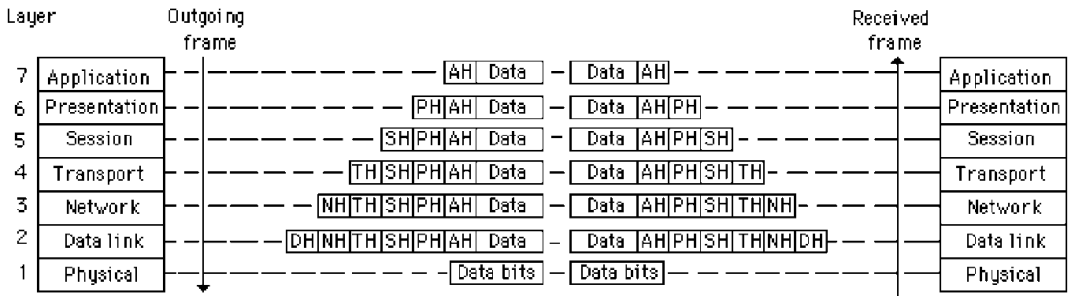
2.1 Model OSI

Organizace ISO (International Organization for Standardization) sestrojila model, který má napomoci implementaci komunikace systémů. OSI (Open Systems Interconnection) model definuje implementaci komunikace za použití 7 vrstev. Každá vrstva má své určité funkce (označováno jako rozhraní), které vykonává. Horní vrstvy obsahují výstupy jednotlivých nižších vrstev. Tedy každá vrstva poskytuje svůj výstup (službu, kterou vykonává) horní sousedící vrstvě. Výhodou je, že změna funkcionality jedné vrstvy se neprojeví na zbytku modelu. [2]

Model pouze demonstruje postup implementace komunikačního rozhraní systémů. Není potřeba striktně dodržet postup podle tohoto modelu. Příkladem může být protokol TCP/IP, který ve své implementaci využívá pouze 4 vrstvy.

Obrázek 2.1 zobrazuje tok rámců přes různé vrstvy tohoto modelu. Jednotlivé vrstvy a jejich funkce jsou následně popsány. Data jsou při průchodu každou vrstvou doplněny o takzvané hlavičky (header). Tyto hlavičky typicky slouží pro přídavné kontroly či informace. Jedná se třeba o bity pro kontrolu správnosti packetů, adresy odesílatele a příjemce nebo typ použitého komunikačního protokolu. Ve schématu se jedná o hlavičky AH (Application Header), PH (Presentation Header), SH (Session Header), TH (Transport Header), NH (Network Header) a DH (Data Link Header). [3]

U vrstev se většinou některé vlastnosti těchto hlaviček „opakují.“ Jako příklad lze uvést detekování chyb v packetech (error check). Pokud je na vyšší vrstvě (vrstva 5) zjištěna chyba packetu, není potřeba opakovat celý proces odeslání tohoto packetu od vrstvy 1, ale lze dohledat vrstvu, na které byl tento packet přijat v pořádku, a opakovat odeslání packetu až od této vrstvy. [2]



Obrázek 2.1 OSI model pro komunikační systémy [3]

1. **Fyzická vrstva (Physical Layer)**
   Základní vrstva, Specifikuje pravidla pro fyzické médium (měděný kabel, optické vlákno, elektromagnetické vlny, kterým se zařízení propojují a přenášejí přes něj

tok bitů. Specifikuje tvar signálů i rychlost, kterou bity přenáší. Toto spojení typicky koresponduje s některými standardami jako RS-232 nebo RJ-45. [2][3][8]

Propojení systémů většinou bývá point-to-point (mezi dvěma zařízeními) nebo při sdíleném médiu jako je rádiové vysílání se jedná o tzv. „broadcast." Takový přenos informace je dostupný všem zařízením, které toto komunikační médium využívají. Avšak tento přenos příjme pouze to zařízení, pro které je určený, ostatní budou tento přenos ignorovat. Příkladem těchto sdílených komunikačních médií jsou bezdrátové sítě nebo lokální sítě LAN. Taktéž určuje směr komunikace (simplex / half-duplex / full-duplex). [2][3][8]

2. **Linková vrstva (Data Link Layer)**
   Tato vrstva tvoří rámce z toku jednotlivých bitů přenášených médiem fyzické vrstvy. Jsou to bajtově synchronizované rámce dat (frames). Přenášená zpráva je většinou rozdělena do několika rámců po stovkách až tisícovkách bajtů [8]. Dále definuje způsob, kterým zařízení dostávají povolení pro získání takto vytvořených rámců přidáním fyzických adres odesílatele a příjemce. Obě konečné zařízení, které jsou fyzicky propojeny, potřebují být koordinované co se týče komunikace. Tedy kdy jedno zařízení má „poslouchat" druhé zařízení a kdy mu naopak má odpovídat. Také poskytuje bity pro kontrolu poškození přenášených rámců a mimo jiné je schopen tento rámec opravit. [2][3][8]

3. **Síťová vrstva (Network Layer)**
   Zahrnuje uspořádání logického propojení mezi příjemcem a odesílatelem. Z různých dostupných datových uzlů tvoří cestu, po které budou dvě zařízení komunikovat. Na této vrstvě se z rámců stávají packety, které v sobě nesou adresy odesílatele a příjemce. Jedná se o adresy internetového protokolu (IP). Packety putují po různých vzájemně propojených sítích a různými cestami. Tím se tedy zpráva, například rozdělená na 5 částí, může u příjemce objevit v jiném pořadí, to způsobuje fakt, že se jedná o vrstvu s paketovým přepínáním (packet swtiching) a každý packet nemusí nutně putovat po stejné cestě. Vrstva se tedy stará o jejich opětovné složení do sekvence, v které byla zpráva odeslána. [2][3][8]

4. **Transportní vrstva (Transport Layer)**
   Garantuje správný přenos packetů po tom, co byla síťovou vrstvou nadefinována trasa přes jednotlivé uzly sítě. Stará se tedy o jednotlivé uzly, kterými packety putují. Dále sleduje, jestli packet byl doručen v pořádku (error control) a taktéž sleduje sekvenci, v které packety mají být přijaty. [2][3][8]

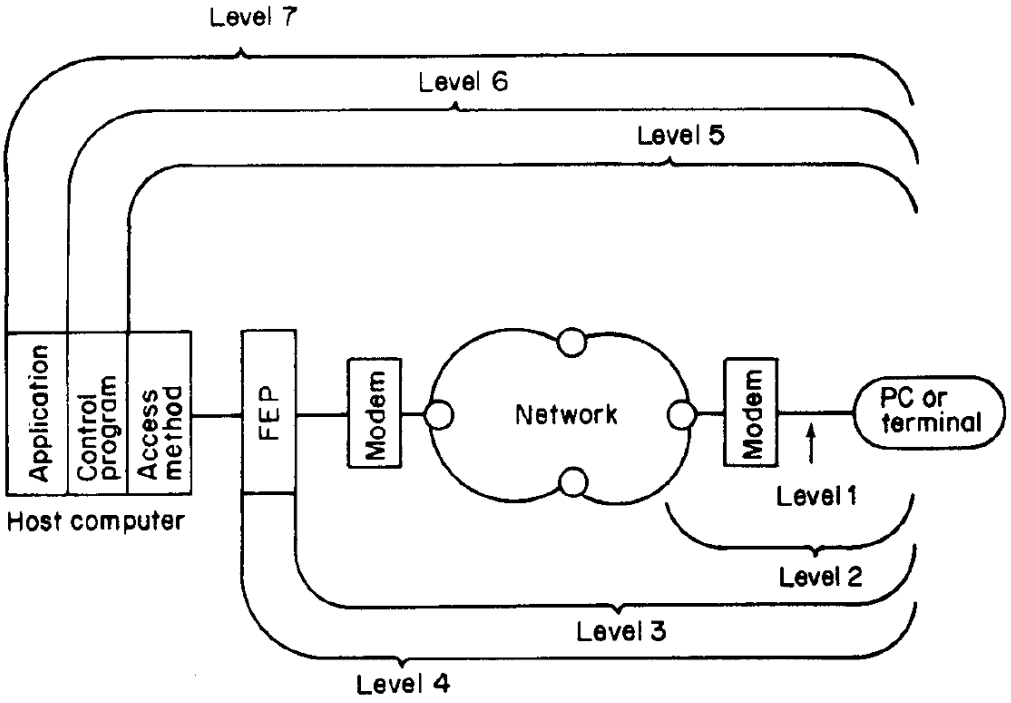
   Nejvíce užívanými transportními protokoly jsou TCP (Transmission Control Protocol) a UDP (User Datagram Protocol). Oba se zároveň typicky používají s kombinací s IP protokolem. [2][3][8]

5. **Relační vrstva (Session Layer)**
   Jedná se o program na straně koncového uživatele, který se stará o nastavení, zahájení a udržení komunikace. Řídí hardware PC propojený se sítí. To znamená, že je závislý na operačním systému tohoto PC. K packetům jsou připojeny znaky synchronizace, které napomáhají k poskládání správné sekvence těchto packetů. [2][3][8]

6. **Prezenční vrstva (Presentation Layer)**
   Tato vrstva se stará o transformování přijatých dat, jejich formátování a syntaxe, která vyhoví konečnému zařízení. Jedná se třeba o kódování zprávy, některých příkazů grafického rozhraní, formáty datových složek, komprese dat a podobné. [2][3][8]

7. **Aplikační vrstva (Application Layer)**
   Tato vrstva funguje jako přístup pro jednotlivé funkce, které jsou vykonávány celým ISO modelem. Jedná se například o přenos složek a přístup do databáze. Mezitím, co první 4 vrstvy jsou relativně pevně specifikované (co se týče jejich funkcionalit). Poslední 3 vrstvy se mohou lišit podle použité sítě. [2][3][8]

   Obrázek 2.2 schematicky znázorňuje místo, kde se jednotlivé vrstvy nachází a jak terminál interaguje s aplikací na hostujícím zařízení (Host Computer).



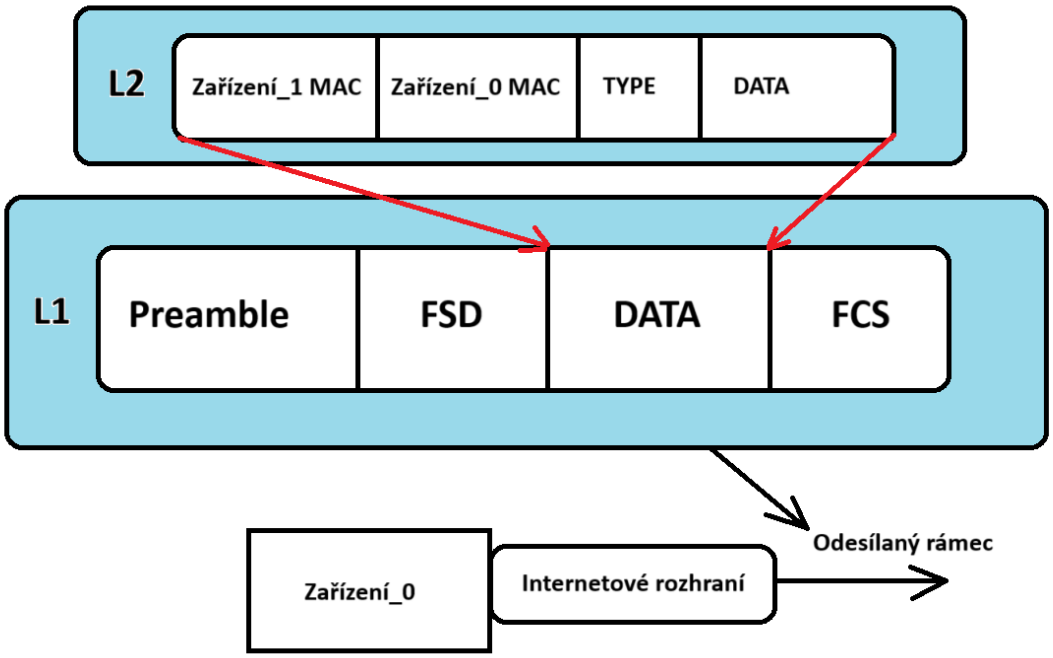
Obrázek 2.2 Schéma jednotlivých vrstev OSI [3]

2.2 Ethernet

Ethernetový protokol je standard od institutu IEEE (Institute of Electrical and Electronics Engineers) jedná se o standard 802.3. Ten popisuje první dvě vrstvy ethernetové komunikace. Jedná se o fyzickou vrstvu a linkovou vrstvu, ta je označována jako LLC (Logical Link Control) a adresový protokol MAC (Medium Access Control). Ethernetový protokol se tedy stará o obsluhu prvních 2 z celkových 7 vrstev OSI modelu. [4][7]

Obrázek 2.3 znázorňuje obsah ethernetového rámce. Lze si povšimnout, že rámec obsahuje další náležitosti, než jen samotná data (tok bitů přenášené zprávy). Slouží k ošetření přenosu datového toku. Linková vrstva (L2) je označována jako ethernetový packet. Ten obsahuje hlavičku (packet header). V této hlavičce se nachází MAC adresa příjemce, MAC adresa odesílatele a type (EtherType), ten určuje protokol, podle kterého jsou data v packetu uložena (někdy jsou tyto data označována jako „payload“). Rozlišuje protokoly jako: IPv4, ARP, IPv6, IEEE 802.1Q, LLDP a mnoho dalších. Na základě tohoto typu se rozliší obsah dat. Může se jednat samotnou přenášenou zprávu nebo o packet vyšší vrstvy L3. [4][7]

Preamble je střídavá sekvence 1 a 0, které napodobují hodinový signál tak, aby se druhé zařízení, které má rámec přijmout, synchronizovalo s bitovým tokem. Před přijmutím FSD (Frame Sequence Delimiter) sekvence by měl hardware cílového zařízení být již synchronizován na bitový tok rámce. FSD poté signalizuje začátek dat packetu, lze se setkat s tím, že FSD není v rámci zmíněno. Důvodem je, že v některé literatuře se FSD sekvence zahrnuje pod Preamble. Po přijetí veškerých dat následuje sekvence FCS (Frame Check Sequence), ta slouží k dohledání případných chyb v přijatém rámci. Může se jednat třeba o rozšířený matematický model CRC (Cyclic Redundancy Check), ten v sobě nese hodnotu, kterou lze získat pomocí sečtení celého rámce. Pokud by tedy některý z bitů v celém rámci byl poškozen, je možné, že se součet celého rámce změní a nebude tak sedět s hodnotou udávanou v FCS. [4][7]



Obrázek 2.3 Ethernetový protokol [4]

2.3 MAC

Jedná se o druhou vrstvu referenčního ISO modelu (Data Link Layer). Je to fyzická adresa zařízení. Slouží k identifikaci zařízení v síti. Někdy je označována jako hardware adresa nebo „burned in address." Z tohoto vyplývá, že takováto MAC adresa je unikátní. Switch by nebyl schopen správně nasměrovat přeposílaný rámec v síti, kde by byly dvě totožné adresy různých zařízení (pokud se jedná o rámec, který má na tuto adresu být zaslán). ARP (Address Resolution Protocol) z této fyzické adresy poté vytváří logickou adresu IP. [4]

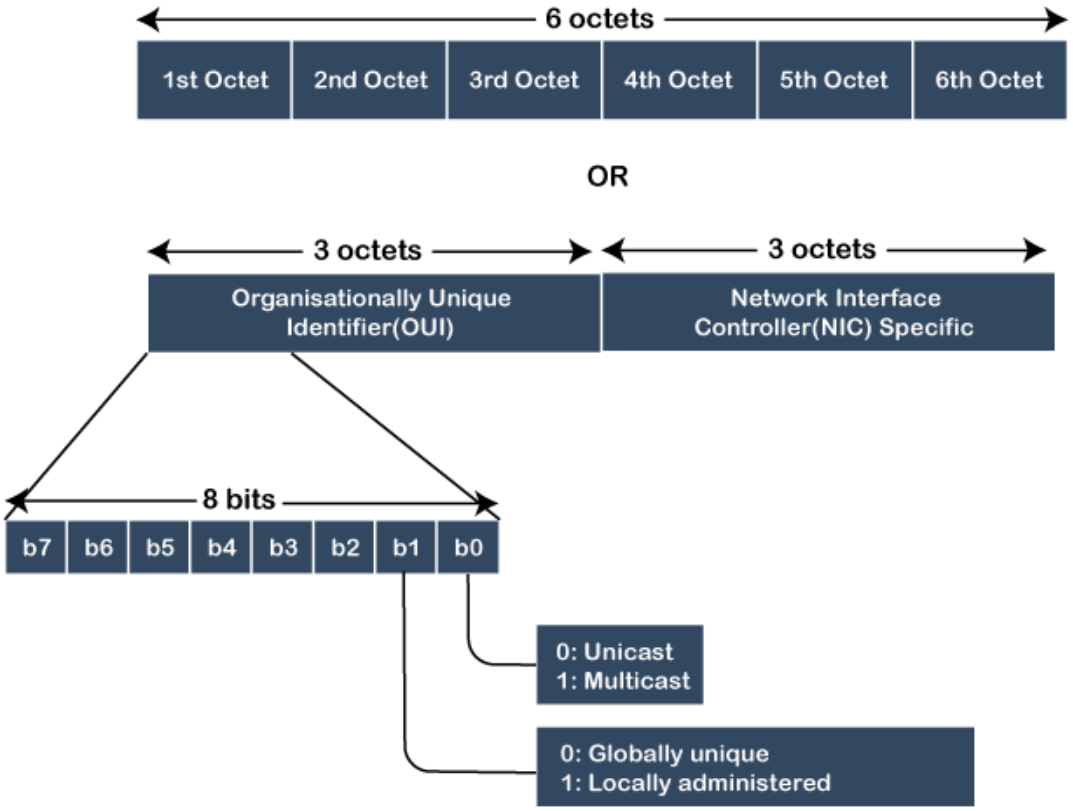
MAC Adresy mají formát 12 hexadecimálních znaků (48 bitů) typicky oddělených po párech. Tyto adresy jsou dány výrobcem zařízení. Příklady formátů zápisu takových adres jsou: **5B:4C:01:F7:00:E9 / 00-B7-82-CC-60-EF / 000.45E.9B3.06E / 025486-3B6548**. Prvních 24 bitů je použito pro OUI (Organization Unique Identifier), ty jsou přiděleny organizací IEEE. Zbylých 24 bitů NIC (Network Interface Controller) si specifikuje výrobce zařízení. Typy adres se liší na Unicast, Multicast, Broadcast. [7]

1. **Unicast**

   Taková adresa reprezentuje specifický NIC na síti. Rámec s takovouto adresou příjemce je zaslán pro rozhraní, na kterém se tato adresa nachází. Rámec je zaslán pouze pro jedno zařízení, které odpovídá této adrese. Unicast adresy jsou specifické tím, že poslední bit v prvním bajtu je nastaven na 0. [7]

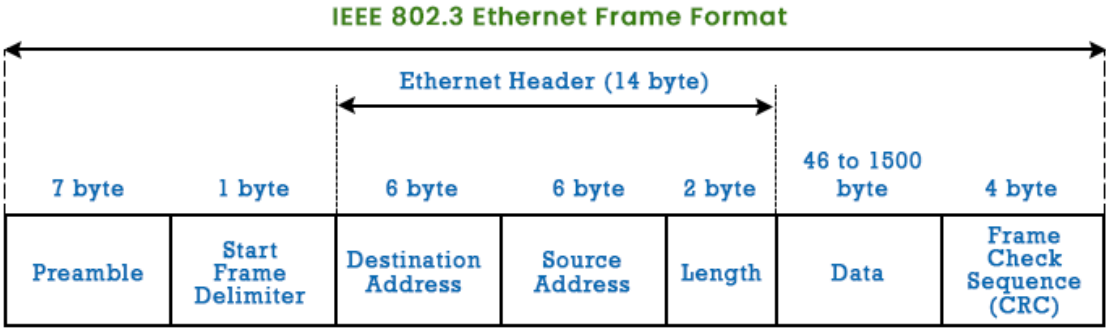
2. **Multicast**
   Odesílající zařízení posílá rámec několika zařízením síti. Multicast adresy jsou specifické tím, že poslední bit v prvním bajtu je nastaven na 1. [7]

3. **Broadcast**
   Takováto adresa cíli na všechny zařízení, které se na síti nacházejí. Broadcast adresa má ve všech 48 bitech 1, je unikatní a její tvar je **FF:FF:FF:FF:FF:FF**. Pokud tedy jedno zařízení chce poslat zprávu všem uživatelům na síti, volí broadcast adresu. [7]



Obrázek 2.4 Části adresy MAC [7]

Na obrázku 2.4 jsou znázorněné jednotlivé části MAC adresy. Oktet (octect) je jiný název pro bajt, tedy skupinu 8 bitů. Obrázek 2.5 poté zobrazuje strukturu ethernetového rámce i s velikostmi jednotlivých částí. Data packetu mohou být délky od 46 do 1500 bajtů. V případě, že odesílaná zpráva má méně jak 46 bajtů, je zpráva automaticky doplněna o nulové bity tak, aby obsahovala alespoň 46 bajtů.

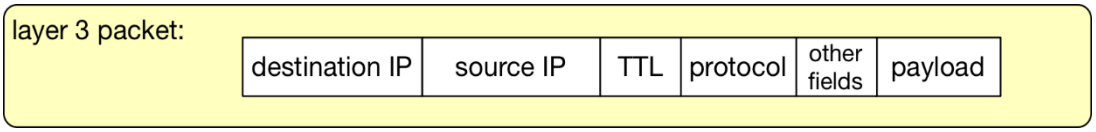


Obrázek 2.5 Ethernetový rámec [7]

2.4 IP

Internetový protokol (IP) se nachází na 3 vrstvě (síťová vrstva). Obdobně jako MAC, IP v sobě obsahuje IP adresy odesílatele a příjemce. Tento protokol nemusí být zpravidla použit v lokální síti, jeho hlavní úloha je při komunikací mezi více IP sítěmi. Komunikace v jedné ethernetové síti lze realizovat bez použití internetového protokolu. [4][7]

IP packet se skládá z hlavičky a dat. Hlavička obsahuje IP adresu (logické adresy) odesílatele a příjemce. TTL (Time To Live), který udává počet skoků v síti po jednotlivých routerech (přeposílaní packetů), po kterém se packet „zahazuje“ (končí jeho životnost). Protocol udává použitý protokol z vyšší vrstvy. Jedná se o známé protokoly jako jsou TCP (Transport Control Protocol) a UDP (User Datagram Protocol). Často lze tedy vidět kombinace jako TCP/IP nebo UDP/IP. Dále obsahuje části, specifikující funkci internetového protokolu. [4]



Obrázek 2.6 Packet internetového protokolu [4]

Cesta, kterou bude packet putovat, je generována algoritmem. V routerech se poté nachází tabulka propojování (routing table). IP adresa se porovná v této tabulce a při shodě se router rozhodne packet poslat daným směrem. [7]

IP adresa je unikátní pro zařízení připojené k internetové síti. Obsahují numerické hodnoty jako **192.168.1.2.** Existují dva typy adres, veřejné a privátní.

1. **Veřejná adresa**

   Taktéž se ji říká extérní adresa, jelikož se objevuje na sféře WAN (Wide Area Network). WAN si lze představit jako sít sítí, propojuje jednotlivé lokální LAN sítě mezi sebou. Adresa slouží pro komunikaci mimo lokální síť, umožňuje přístup

k internetu. Tato adresa poskytuje dálkový přístup k našemu zařízení. Je generována ISP (Internet Service Provider), poskytovatelem internetového připojení. Za službu pro vytvoření této adresy se platí, příkladem jsou služky poskytující internetové připojení jako O2, UPC, Vodafone a jiné. [7]

2. **Privátní adresa**

Interní adresa se vyskytuje na síti LAN. Slouží pro komunikaci v lokální síti. Nevyskytují se na internetu čili k nim není přístup zvenčí. Může se jednat o adresu tiskárny nebo mobilního telefonu. Zbytek zařízení na stejné lokální síti „vidí“ toto zařízení s privátní adresou, kdežto zařízení mimo lokální adresu jej nevidí, mohou k němu ovšem přistoupit, pokud znají lokální adresu routeru této lokální sítě. Tato adresa nezávisí na poskytovateli, služba je zdarma. [7]

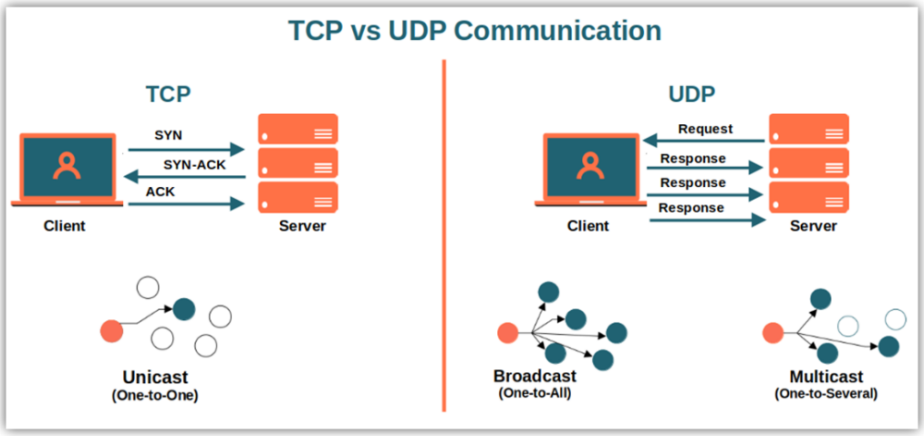
2.5 UDP/TCP

User datagram protocol (UDP) slouží k odesílání datagramů za použití internetového protokolu IP. Jedná se o protokol 4. vrstvy (transportní vrstva). Obsahuje minimum komunikačních mechanizmů jako je ověření navázání spojení, synchronizace konvových zařízení, čekání na potvrzení přijetí packetu a další. [7]

Díky absence komunikačních mechanizmů je schopen velice rychle odesílat packety. Stačí mu pouze požadavek koncového zařízení a na základě toho začne odesílání packetů. Jelikož neřeší stav propojení s koncovým zařízením, je označován jako nespolehlivý. Nepotřebuje se virtuálně propojit s koncovým zařízením. Pořadí odesílaných datagramů není nijak označené, tedy na koncovém zařízení nemusí být datagramy přijaty ve správném pořadí. Datagramy se taktéž mohou při přenosu ztratit. [7]

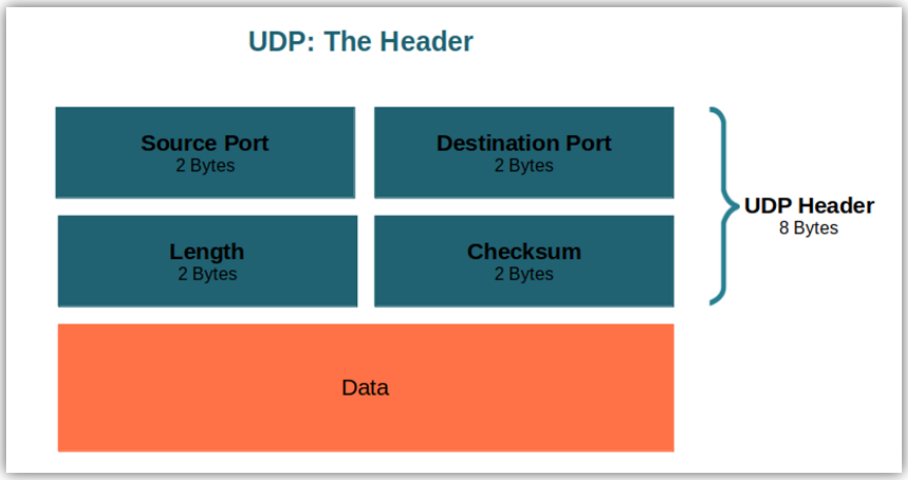
Tento protokol se dá využít třeba pro streamování videí. Potvrzovat tisíce packetů videa během přenosu by bylo velice náročné a výrazně by snížilo rychlost přenosu. Ztráta pár datagramů během streamování videa nemusí být ani pozorovatelná a lze tedy ztrátu packetů ignorovat. [7]

Rozdíl protokolu UDP a TCP je znázorněn na obrázku 2. TCP protokol se musí před zahájením přenosu synchronizovat s koncovým zařízením. Při odeslání packetu je požadováno potvrzení obou stran a při případné ztrátě nebo chybě packetu je tento packet znovu odeslán. Komunikace probíhá mezi dvěma uživateli (point-to-point). V případě UDP protokolu stačí, aby se detekoval požadavek (request) na zahájení přenosu datagramů a UDP okamžitě začíná posílat datagramy na síť, z které požadavek přišel (není přímo propojený s koncovým uživatelem). Nemá takzvanou „handshake“ komunikaci. Umožňuje multicast nebo broadcast komunikaci. [7]



Obrázek 2.7 Rozdíl mezi TCP a UDP protokoly [6]

UDP protokol obdobně obsahuje hlavičku a data. Hlavička obsahuje 2 bajtové adresy portů odesílatele a příjemce, celkovou velikost UDP packetu a hodnotu kontrolního součtu. Zahrnutí kontrolního součtu packetu je pouze volitelné [7].



Obrázek 2.8 UDP packet [5]

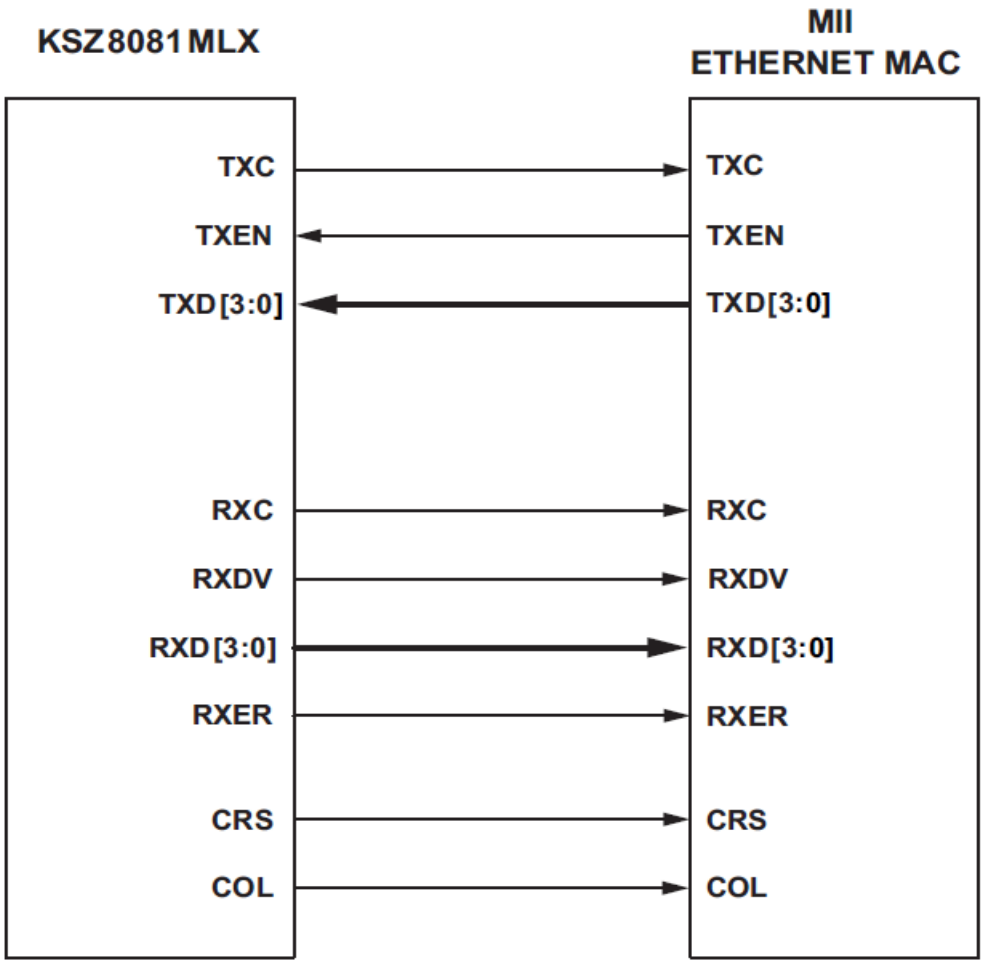
2.6 Rozhraní MII

Standard definovaný v IEEE 802.3. MII (Media Independent Interface) slouží jako rozhraní mezi MAC-PHY. PHY je označení pro transceiver čip (vysílá i přijímá), čip někdy může poskytovat i integrované MAC (Media Access Control) rozhraní. MII obsahuje dvě sběrnice, datové a řídící. Independent znamená, že rozhraní je nezávislé na použitém procesoru a přenosovém médiu (měděný drát, optické vlákno

či elektromagnetické vlny). Umožňuje realizovat přenos 10 Mbps (Megabitů za sekundu) a 100 Mbps. [10]

Rozhraní je typicky realizováno 18 signály. Z toho 2 signály (MDIO/MDC) je možné sdílet mezi více PHY čipy. MDIO a MDC signály tvoří rozhraní MII Management (MIIM), který slouží ke sledování stavu a konfiguraci čipu PHY pomocí MAC procesoru. Jedná se o synchronní sériovou komunikaci podobnou I2C. Po připojení PHY čipu se čip nastaví pomocí tzv. „autonegotiation“ (automatické vyjednávání) nebo je nastaven MAC procesorem přes toto rozhraní. [11][12]

Obrázek 2.9 zobrazuje rozhraní MII mezi PHY čipem KSZ8081MLX a MAC procesorem (třeba RPP). Lze si povšimnout, že čip realizuje MII pomocí 15 signálů (nevyužívá signál TXER). U označení [3:0] se jedná o 4 bity (nibble). Podrobněji je tento čip rozepsán v kapitole 3.1.



Obrázek 2.9 MII rozhraní PHY čipu KSZ8081MLX [11]

3. PROPOJENÍ RP2040 S MAC ČIPEM

3.1 Výběr čipu KSZ8081MLX

Jako čip fyzické vrstvy PHY byl zvolen KSZ8081MLXCA (kde typ CA určuje teplotní rozsah pro komerční užití 0-70 °C). Čip nabízí 10BASE-T (10 Mbps) nebo 100BASE-TX (100 Mbps) ethernetový PHY transceiver pro přenos a příjem dat po CAT-5 (UTP) kabelu. Čip poskytuje MII rozhraní bez integrovaného MAC. Digitální vstupy / výstupy lze provozovat na 1.8 / 2.5 / 3.3 V. MIIM rozhraní umožňuje MAC procesoru kompletní přístup do stavových a řídících registrů čipu KSZ8081MLX. Lze také využít pin přerušení, který MAC procesor informuje o změně stavu PHY čipu (eliminuje neefektivní dotazování se „polling“). [11]

U čipu lze taktéž řídit jeho spotřeba pomocí až 4 různých úsporných módů. Přehled signálů MII je znázorněn na obrázku 3.1.

MII Signal Name	Direction with Respect to PHY, KSZ8081 Signal	Direction with Respect to MAC	Description
TXC	Output	Input	Transmit Clock (2.5 MHz for 10 Mbps; 25 MHz for 100 Mbps)
TXEN	Input	Output	Transmit Enable
TXD[3:0]	Input	Output	Transmit Data[3:0]
RXC	Output	Input	Receive Clock (2.5 MHz for 10 Mbps; 25 MHz for 100 Mbps)
RXDV	Output	Input	Receive Data Valid
RXD[3:0]	Output	Input	Receive Data[3:0]
RXER	Output	Input or not required	Receive Error
CRS	Output	Input	Carrier Sense
COL	Output	Input	Collision Detection

Obrázek 3.1 Signály MII čipu KSZ8081MLX [11]

3.2 Schéma a deska plošných spojů

Před návrhem desky plošných spojů bylo potřeba navrhnout schéma propojení RPP a čipu KSZ8081MLX. Schéma je k dohledání v příloze A. Pro návrh schématu i desky plošných spojů byl využit software EasyEDA.

Výrobce čipu PHY specifikuje některé potřebné součástky pro funkčnost čipu. Jedná se třeba o filtrační kondenzátory napájecích cest, strap-in konfiguraci apod. Tyto doporučení tedy jsou převzaty nebo případně převzaty s dodatečnou úpravou. Použitý RJ-45 konektor má v sobě integrovaný magnetický modul (magnetika). Magnetika nebo též oddělovací transformátory se používají pro splnění elektrických požadavků. Jedná se o elektrickou izolaci, vybalancování signálů, impedanční přizpůsobení a EMC (elektromagnetická kompatibilita) [13]. Pro napájení je použit step-down měnič

MP1584EN, který byl z 5 V výstupu přizpůsoben na 3.3 V. Na signály fyzické vrstvy mezi RJ-45 konektorem a čipem KSZ8081MLX je možné přidat přepěťovou ochranu použitím diodového pole SLVU2.8-4.

U vysokofrekvenčních signálů je potřeba brát v potaz délky tras signálů. Je důležité, aby signál od vysílače do přijímače připutoval s předstihem a jeho úroveň byla pevně daná v moment snímání úrovně signálů přijímačem. Roli zde hrají hodnoty jako propagation delay (zpoždění signálu mezi vysílačem a přijímačem) a t_{setup} (čas, po který musí být hodnota ustálená a neměnná před příchodem hrany hodin). Pro výpočet zpoždění signálu na DPS (Deska Plošných Spojů) je použit vzorec (3.1), dohledatelný v [14].

$$t_{pd} \approx 85 \cdot \sqrt{E_{reff}} \text{ [ps/in]}, \tag{3.1}$$

Kde E_{reff} je efektivní dielektrická konstanta použitého substrátu DPS a je dána jako $(0.64 \cdot E_r + 0.36)$ [14]. Hodnota E_r u použitého DPS je přibližně 4.7. Hodnota, o kterou se signál zpožďuje je poté:

$$t_{pd} \approx 85 \cdot \sqrt{0.64 \cdot 4.7 + 0.36} \approx 156 \text{ [ps/in]}$$

Výsledek je v pikosekundách na palec, převedením na cm získáme hodnotu 61.42 ps/cm. Tedy signál se přenese o 1 cm za 61.42 pikosekund. Nyní je potřeba zjistit periodu hodin čipu PHY a jeho požadavek na t_{setup}. V mé aplikaci bude čip sloužit pro přenos 100 Mbps, tedy 100BASE-TX. Pro tento přenos je použit hodinový kmitočet o frekvenci 25 MHz. Z toho vyplývá perioda 40 ns. Hodnotu t_{setup} výrobce udává na 10 ns [11]. Při vysílaní dat je tedy zapotřebí, aby nejpozději po uplynutí 30 ns z MAC procesoru doputoval signál TX[0:3] k čipu PHY a měl jasně definovanou hodnotu (1 / 0), která se dalších 10 ns nezmění. Pro praktické využití se tento interval sníží na polovinu, tedy 15 ns. Nyní tento požadavek lze ověřit na základě vypočítaného zpoždění signálu t_{pd}. Maximální délka trasy je určena jako trasa, kterou signál „uběhne“ za 15 ns.

$$l_{max} = \frac{t_{max}}{t_{pd}} = \frac{15\,000}{61.42} = 244{,}22 \text{ cm}$$

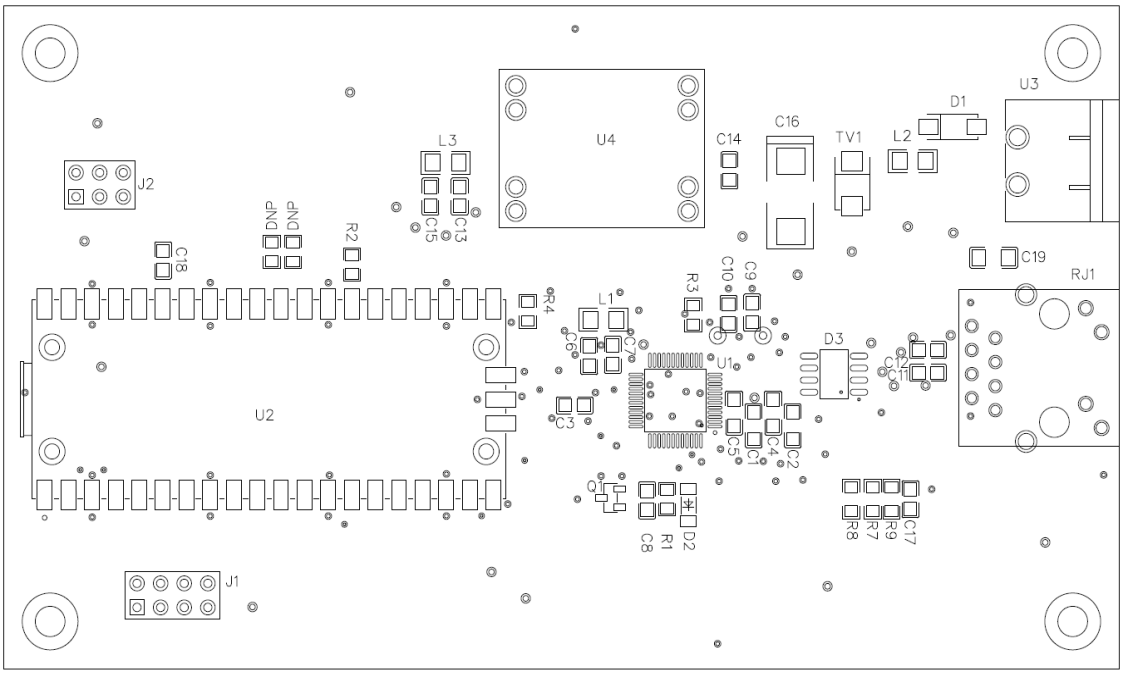
V mém případě jsou délky cest TX signálů v rozmezí 3.04 - 4.16 cm a délka signálu TXC je 2.50 cm. V nejhorším případě bude zpoždění signálu:

$$t_{dp}(TXC, TX) = t_{dp} \cdot l = 61.42 \cdot (4.16 + 2.50) = 409.06 \text{ ps}$$

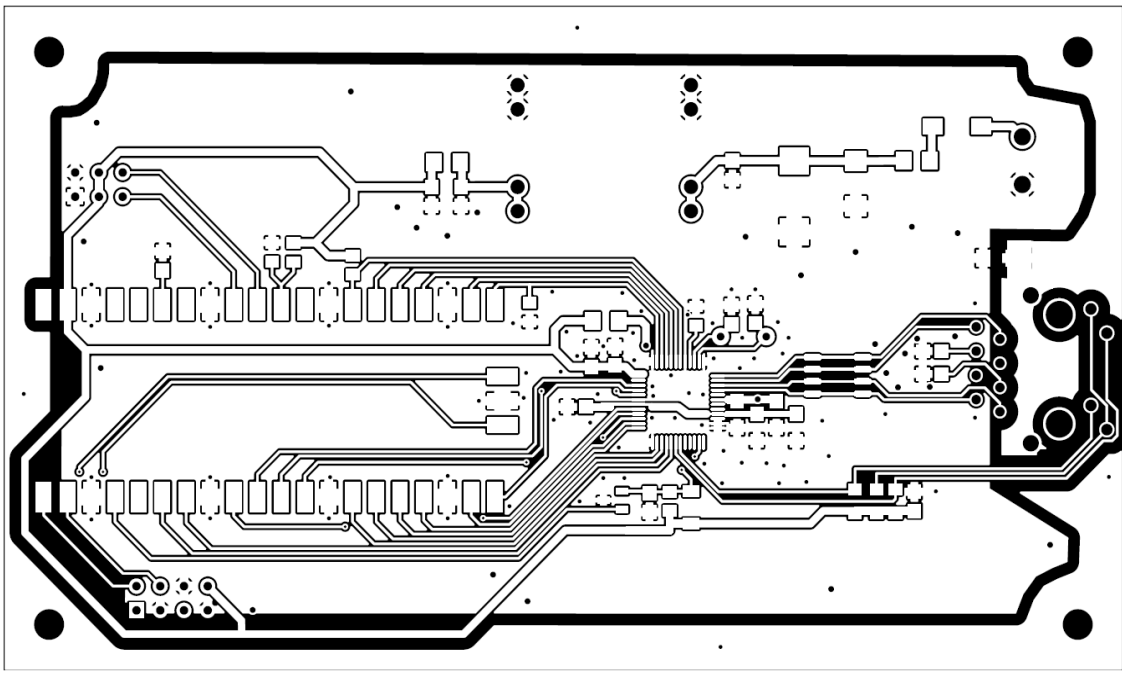
Vypočtené hodnoty jsou přibližné, nicméně lze konstatovat, že vypočtené zpoždění signálu je několikanásobně menší jak maximální doporučené zpoždění a bude splňovat časovací podmínky pro přenos 100TBASE-TX.

Následující obrázky znázorňují návrh DPS. Okraj desky je oddělený. Jedná se o oddělení „chassis“ země a signálové země. Footprint konektoru RJ-45 jsem vytvořil podle datasheetu pro použitý konektor LMJTAB881243ML, jelikož v softwaru EasyEDA jsem tento footprint nenašel. U některých kondenzátorů a rezistorů jsem rozšířil footprint tak, aby pod nimi bylo možné lépe vést cesty. Okolo důležitých částí

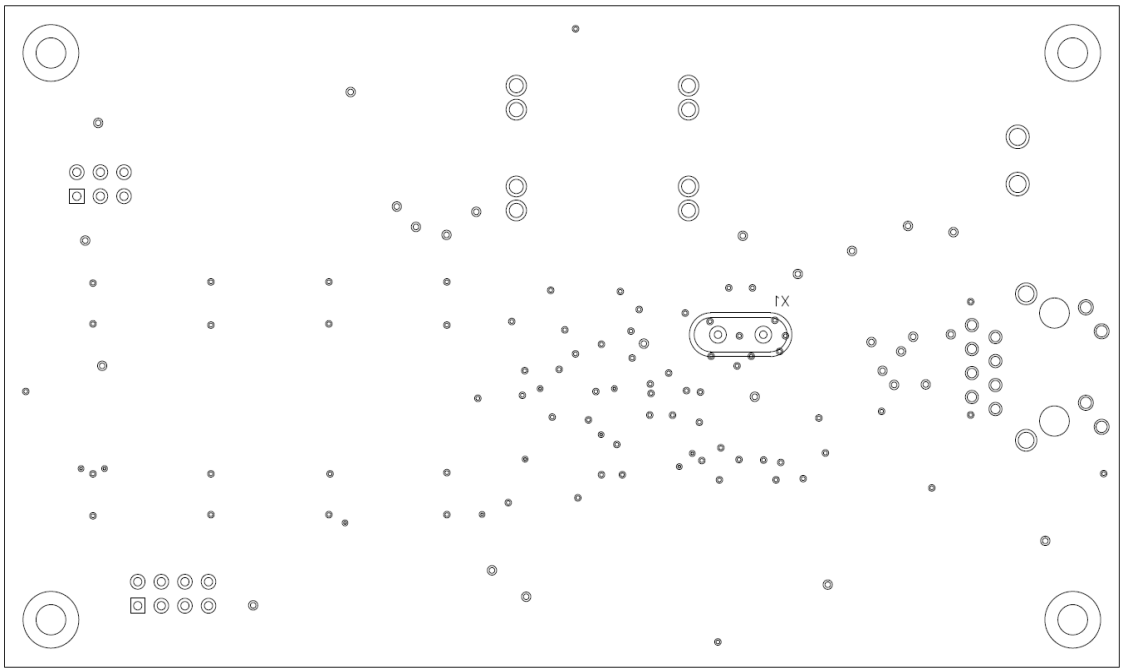
jako je oscilátor či napájecí cesta Vcc (širší cesta) jsou umístěny via pro eliminaci velkých ploch, na kterých vzniká rušení nižších frekvencích. Deska obsahuje dva pinheader konektory, jedná se o konektor pro snímače vzdálenosti a konektor pro debugování (ladění) mikrokontroléru RP2040, jejich pinout je znázorněn na obrázku 3.7.



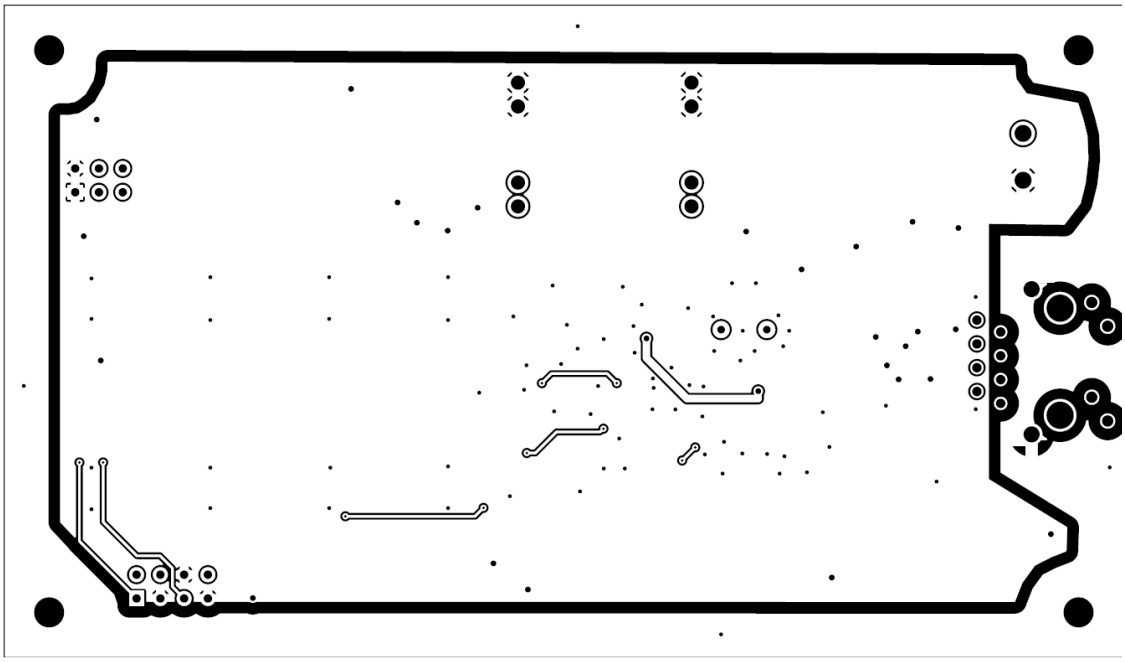
Obrázek 3.2 Horní montážní návrh PCB



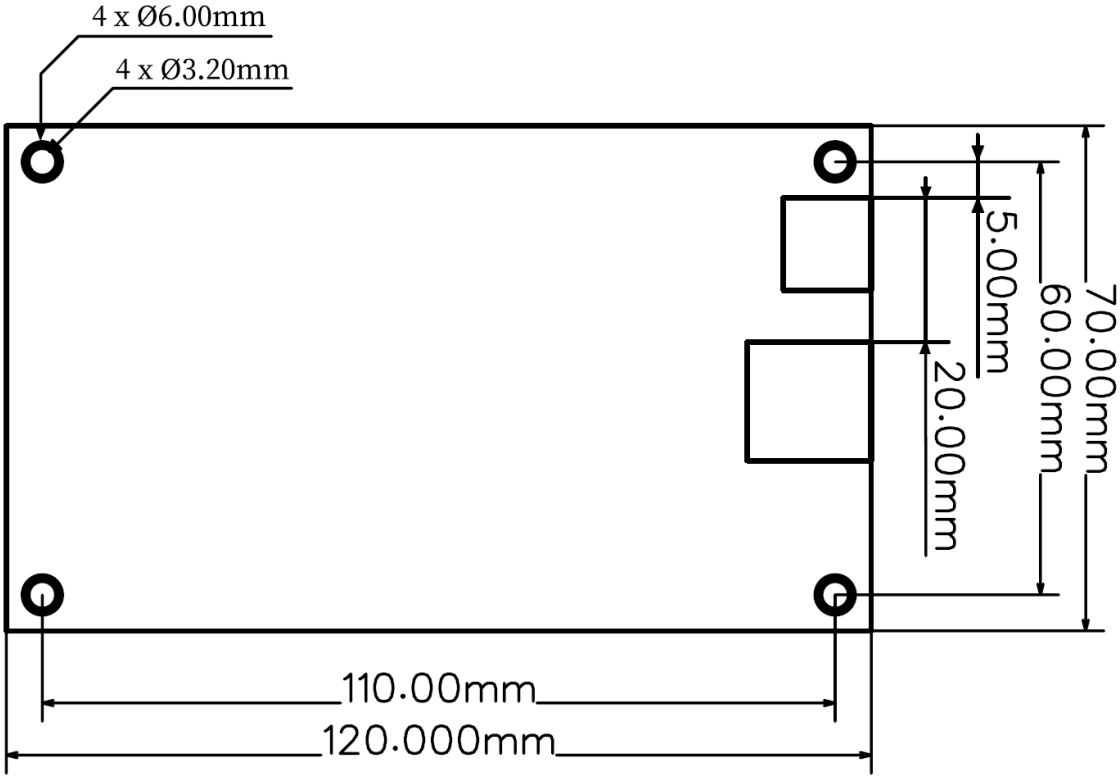
Obrázek 3.3 Horní vrstva DPS



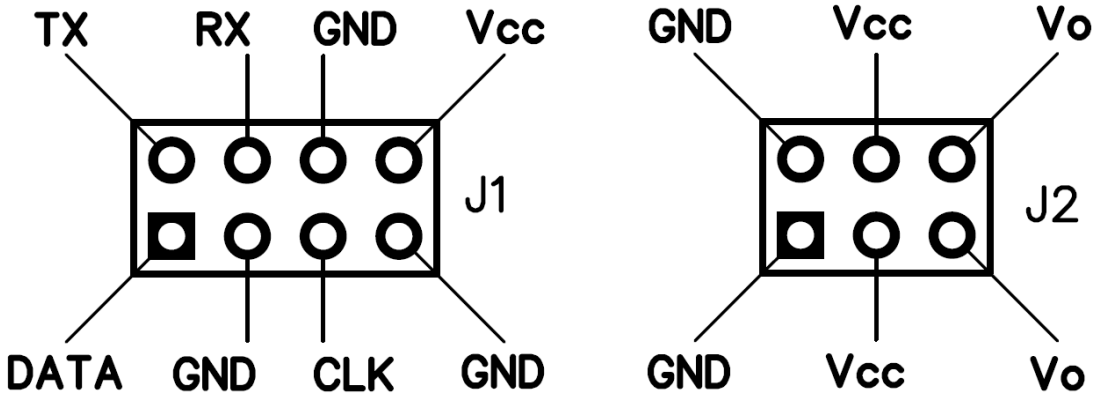
Obrázek 3.4 Dolní montážní návrh PCB



Obrázek 3.5 Dolní vrstva PCB

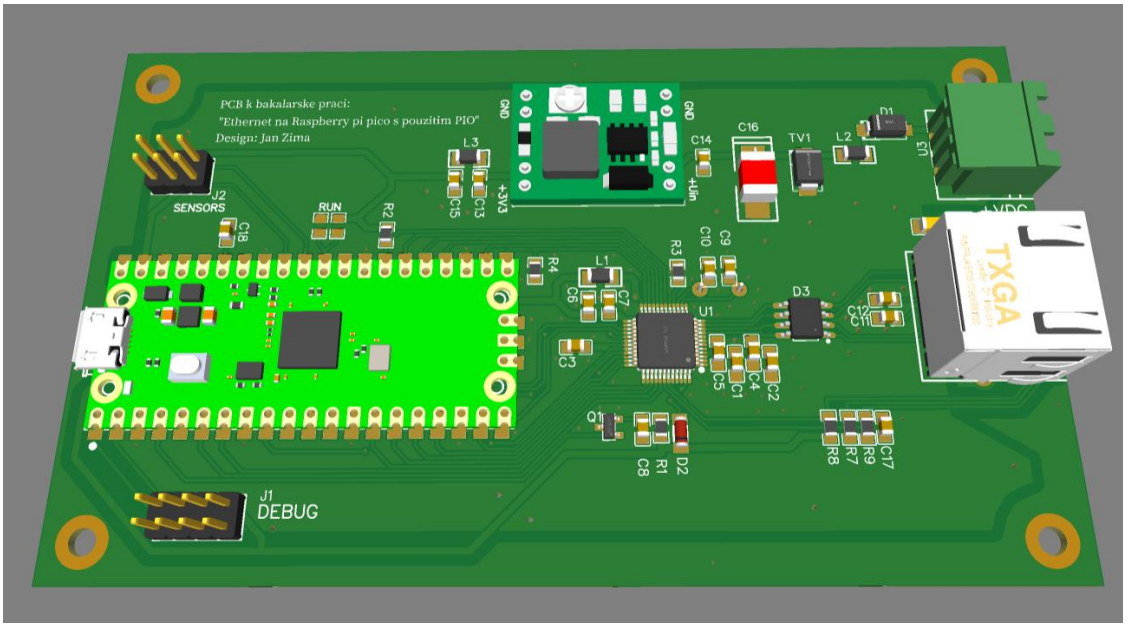


Obrázek 3.6 Rozměry DPS



Obrázek 3.7 Pinout pinheaderů J1 a J2

V softwaru EasyEDA lze zobrazit taktéž 3D model navržené DPS. Model je zobrazen na obrázku 3.8. Většině footprintů bylo potřeba přiřadit 3D modely z knihovny. Pro některé footprinty však knihovna neposkytuje 3D modely a byly tedy byly využity modely vytvořené komunitou. Jedná se o model mikropočítače RPP a step-down měnič. 3D Model mikropočítače RPP byl převzat od uživatele „Lukevanluijn“ a model step-down měniče od uživatele „imextrade.“

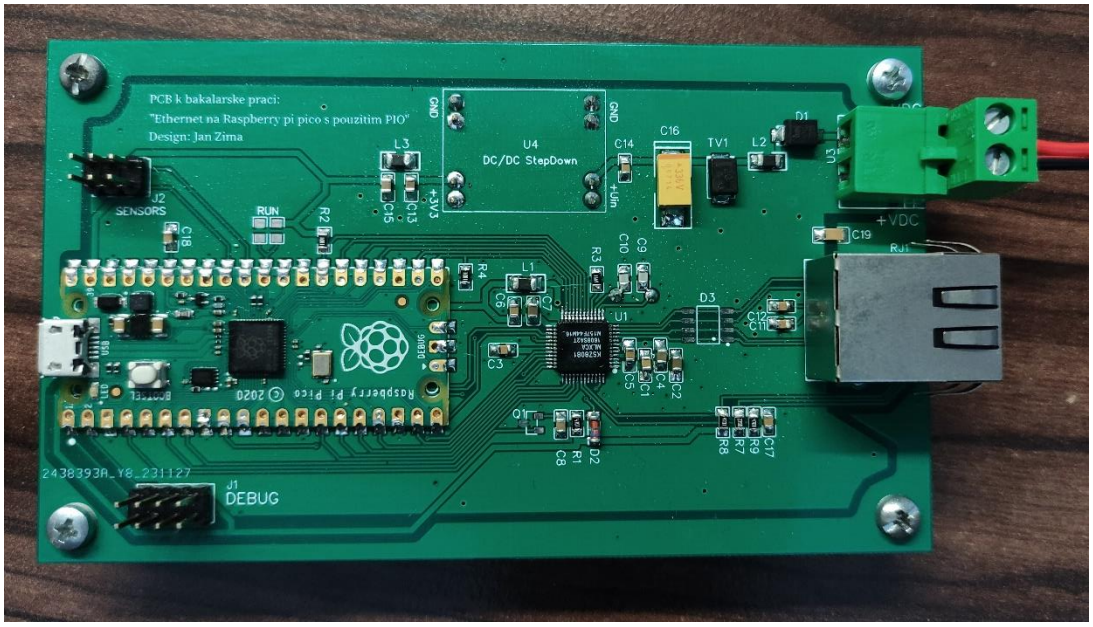


Obrázek 3.8 3D model DPS

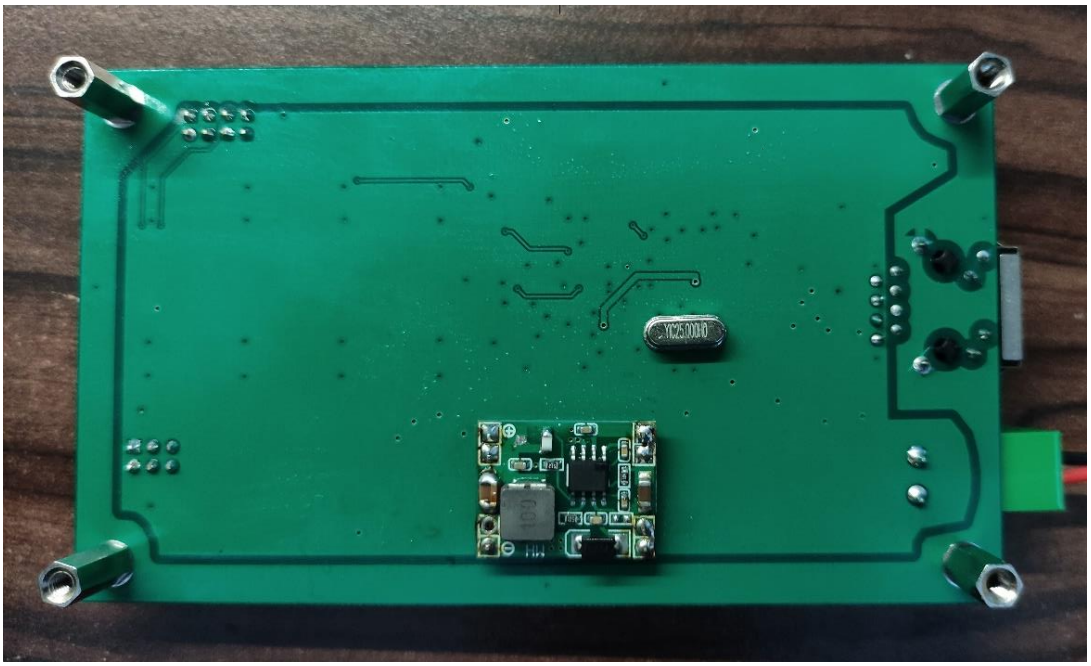
3.3 Funkční prototyp

Při finálním návrhu DPS je potřeba vygenerovat pomocí softwaru tzv. gerber soubory. Ty se poté předávají výrobci DPS a slouží jako výrobní podklady. Pro výrobu mé DPS byl zvolen čínský výrobce JLCPCB.

Jako poslední krok je zapájení samotných elektronických součástek na DPS. Použité součástky v mé bakalářské práci lze dohledat ve schématu, a to v příloze A. Obrázek 3.9 a 3.10 obsahují poté mnou zhotovené DPS. Lze si povšimnout, že step-down měnič je zapájený na dolní straně DPS, jelikož jeho footprintu v softwaru EasyEDA má prohozené piny. Naštěstí se tato maličkost dala vyřešit právě osazením ze spodu. Při finálních úpravách se taktéž změnil napájecí konektor, u kterého jsou prohozeny terminály GND a V_{CC} oproti původnímu použitému konektoru. Tento fakt jsem zjistil až při samotném oživování DPS a na DPS jsou tedy značení -VDC a +VCD prohozeny. Na desce zbývá ještě osadit P-Mosfet tranzistor Q1, který se musel dodatečně objednat. Jedná se o signál resetu čipu PHY, tudíž není nijak kritický pro ladění softwaru. Taktéž na desce není osazen obvod pro přepěťovou ochranu D3, jedná se o přídavnou ochranu, kterou může uživatel využít.



Obrázek 3.9 Vrchní vrstva realizované DPS



Obrázek 3.10 Spodní vrstva realizované DPS

4. VÝVOJ FIRMWARU A PIO PROGRAMU

Pří vývoji firmwaru jsem využil k inspiraci již existující repozitář na GitHubu [15]. K realizaci ethernetové komunikace však využívá rozhraní RMII (Reduced Media Independent Interface) a jiný čip fyzické vrstvy. Nicméně slouží jako dobrý návod, jak začít s implementací mého firmwaru.

Pro vývoj svého firmwaru jsem založil taktéž repozitář na GitHubu, kde je veškerý vytvořený kód firmwaru dostupný k nahlédnutí [16].

4.1 Rozhraní MIIM

Jako první krok jsem se zabýval tím, jak správně komunikovat s čipem pro jeho následovné nakonfigurování. K tomuto je potřeba využít rozhraní MIIM. Toto rozhraní využívá dva signály. Jsou jimi MDC a MDIO. U tohoto rozhraní je potřebné dodržet rychlost MDC. Výrobce čipu KSZ8081MLX v datasheetu udává nejvyšší rychlost, při které toho rozhraní může komunikovat, na 10 MHz [11].

Nejprve byly vytvořeny funkce, které realizují nastavení nebo čtení bitu na signálu MDIO. Definice jednotlivých pinů jako je KSZ_MDC, KSZ_MDIO a podobné je možné nalézt v souboru „defines.h“ [16].

```
/*********************************************************************
static void mii_mdio_out(bool bit){
      gpio_put(KSZ_MDC, 0);
      gpio_put(KSZ_MDIO, bit);
      gpio_put(KSZ_MDC, 1);
}
static bool mii_mdio_in(){
    gpio_put(KSZ_MDC, 0);
    gpio_put(KSZ_MDC, 1);

    bool bit = gpio_get(KSZ_MDIO);
    return bit;
}
/************************soubor: miim.c [4-16]**********************
```

Z tohoto zápisu však není patrné, jakou frekvencí rozhraní komunikuje. Pro ověření splnění maximální frekvence 10 MHz se budu tedy muset podrobněji podívat na zápis tohoto kódu pomocí asemblerových instrukcí.

Nejprve provedu výpočet, kolik cyklů procesoru je minimálně potřeba pro splnění tohoto požadavku. Aktuálně je mikrokontrolér řízen systémovým taktem SYS_CLK = 125 MHz. Tedy vykonání jednoho cyklu trvá 125 MHz. Se zvyšujícím se počtem instrukcí bude klesat frekvence, kterou je daný celek instrukcí vykonán.

$$f_{miim} = \frac{f_{sys}}{N} \text{ [MHz]}, \tag{4.1}$$

Kde f_{sys} je systémová frekvence kontroléru a N je počet cyklů, potřebný k vykonání sledu instrukcí, které procesor vykoná. Pro zjištění potřebného počtu cyklů je vztah poté:

$$N = \frac{f_{sys}}{f_{miim}} = \frac{125\,000\,000}{10\,000\,000} = 12.5 \ [-]$$

Je tedy potřeba, aby výše zmíněné funkce *mii_mdio_out* / *mii_mdio_in* obsahovali alespoň 13 asemblerových instrukcí pro dodržení maximální frekvence 10 MHz. Po dosazení hodnoty 13 za parametr *N* do vztahu 4.1 je výsledná frekvence rozhraní MIIM rovna přibližně 9.62 MHz.

Nyní je možné si zobrazit zmíněnou funkci rozebranou do asemblerových instrukcí.

```
  707: }
0x1000032e              <mii_mdio_out+42>           f3e7      b.n    0x10000318 <mii_mdio_out+20>
  706:     sio_hw->gpio_clr = mask;
0x10000330              <mii_mdio_in+0>             d023      movs   r3, #208        ; 0xd0
0x10000332              <mii_mdio_in+2>             1b06      lsls   r3, r3, #24
0x10000334              <mii_mdio_in+4>             8022      movs   r2, #128        ; 0x80
0x10000336              <mii_mdio_in+6>             5203      lsls   r2, r2, #13
0x10000338              <mii_mdio_in+8>             9a61      str    r2, [r3, #24]
  697:     sio_hw->gpio_set = mask;
0x1000033a              <mii_mdio_in+10>            5a61      str    r2, [r3, #20]
  675:     return !!((1ul << gpio) & sio_hw->gpio_in);
0x1000033c              <mii_mdio_in+12>            5b68      ldr    r3, [r3, #4]
0x1000033e              <mii_mdio_in+14>            5b0d      lsrs   r3, r3, #21
0x10000340              <mii_mdio_in+16>            0120      movs   r0, #1
0x10000342              <mii_mdio_in+18>            1840      ands   r0, r3
  37:     return bit;
0x10000344              <mii_mdio_in+20>            7047      bx     lr
0x10000346              0000      movs r0, r0
  13: static uint ethernet_frame_crc(const uint8_t *data, int length){
0x10000348              <ethernet_frame_crc+0>      30b5      push   {r4, r5, lr}
```

Obrázek 4.1 Asemblerový kód funkce MIIM rozhraní

Příklad je zobrazen pro funkci *mii_mdio_in* jelikož obsahuje méně instrukcí než funkce *mii_mdio_out*. Na obrázku 4.1 lze vidět, že se funkce nachází v paměti FLASH na adrese *0x10000330* až *0x10000344* (červené ohraničení).

Na žlutě zvýrazněném řádku lze zleva vidět prvně místo v paměti *0x10000344*, označení pro toto místo *<mii_mdio_in+20>*, hexadecimální hodnota použité instrukce *7047*, název instrukce *bx* a registr této instrukce *lr* (který v sobě nese návratovou adresu z podprogramu a předává ji do registru PC).

Dohromady tuto funkci tedy tvoří 11 instrukcí. Na první pohled se může zdát, že maximální rychlost není dodržená. Je však potřeba zjistit, kolik cyklů potřebují jednotlivé instrukce k jejich vykonání. To zjistím z datasheetu procesoru Cortex-M0+ [17].

Instrukce *str* (store), *ldr* (load) a *bx* (branch) v mém případě potřebují dva cyklu pro své vykonání. Zbytek instrukcí se vykoná v jednom cyklu. 4 z 11 instrukcí tedy potřebují k svému vykonání 2 cykly. Finální počet je tedy 15 cyklů.

Po dosazení do zmíněného vztahu 4.1 je výsledná frekvence funkce *mii_mdio_in* rozhraní MIIM přibližně 8.33 MHz. Pro funkci *mii_mdio_out* je tato frekvence ještě nižší, jelikož se skládá z více instrukcí.

Tímto jsem si ověřil správné dodržení maximální rychlosti přenosu dat na rozhraní MIIM. Je však nutno podotknout, že pro analýzu kódu na úrovni asembleru jsem musel využít *build* programu pomocí *debug mode* namísto *release mode*. U finální verze, vytvořené pomocí *relase mode,* je možné, že bude kód více optimalizován (debug mode obsahuje více informací a neoptimalizuje kód, je tak umožněno programátorovi detailnější sledování a krokování programu při jeho vývoji) a zmenší se tak jeho počet instrukcí, to může vést k větší frekvenci tohoto rozhraní. Je tedy nutné správnost ověřit na obou módech. V mém případě je toto rozhraní funkční při obou módech. Je dokonce možné, že i při překročení frekvence, danou výrobcem, bude toto rozhraní plně funkční.

Pro rozhraní MIIM je definovaný následující rámec:

	Preamble	Start of Frame	Read/ Write OP Code	PHY Address Bits[4:0]	REG Address Bits[4:0]	TA	Data Bits[15:0]	Idle
Read	32 1's	01	10	000AA	RRRRR	Z0	DDDDDDDD_DDDDDDDD	Z
Write	32 1's	01	01	000AA	RRRRR	10	DDDDDDDD_DDDDDDDD	Z

Obrázek 4.2 Rámec MIIM rozhraní [11]

Na obrázku 4.2 jsou znázorněny rámce *write* i *read* MIIM rozhraní. Rozdíl mezi nimi lze vidět v OpCode a dále v TA (Turn Around). Při vpisování dat do registru vyšle MAC procesor hodnotu TA „10“ v opačném případě, tedy chce-li MAC vyčíst hodnotu registru, musí linku MDIO uvolnit. Pro nastavení jednotlivých bitů na lince MDIO používám právě výše zmíněnou funkci *mii_mdio_out*. Při čtení registru se po odeslání adresy registru uvolní linka MDIO pomocí nastavení MDIO pinu do vstupu (tedy uvolnění linky). Poté je pro čtení hodnoty registru využita funkce *mii_mdio_in*.

Dále si v datasheetu naleznu kontrolní registr čipu PHY a jeho jednotlivé bity pro správné nastavění 10 / 100 Mbps. Pro zvolený čip KSZ8081MLX je tento registr jako první a jeho hodnota je 0x0. Jedná se o 16 bitový registr. Hodnota bitu 0.13 v tomto registru určuje rychlost přenosu dat. Je však potřeba deaktivovat bit 0.12 „Auto Negotiation Enable,“ který případné nastavení bitu 13 přepisuje [11].

Pro nastavení rychlosti je tedy třeba nastavit bit 0.12 do nuly a bit 0.13 podle požadované rychlosti. Pro 100 Mbps je tato hodnota 1 a pro hodnotu 10 Mbps je 0 [11].

	Nibble 1				Nibble 2				Nibble [3:4]	
	Reset	Loopback	Speed	Auto-Neg	Power-D	Isolate	Reset AN	Duplex	COL	Reserveed
Bit	0.15	0.14	0.13	0.12	0.11	0.10	0.9	0.8	0.7	0.[6:0]
Hodnota	0	0	0 / 1	0	0	0	0	0	0	000_0000
100 Mbps	0x2				0x0				0x00	
10 Mbps	0x0				0x0				0x00	

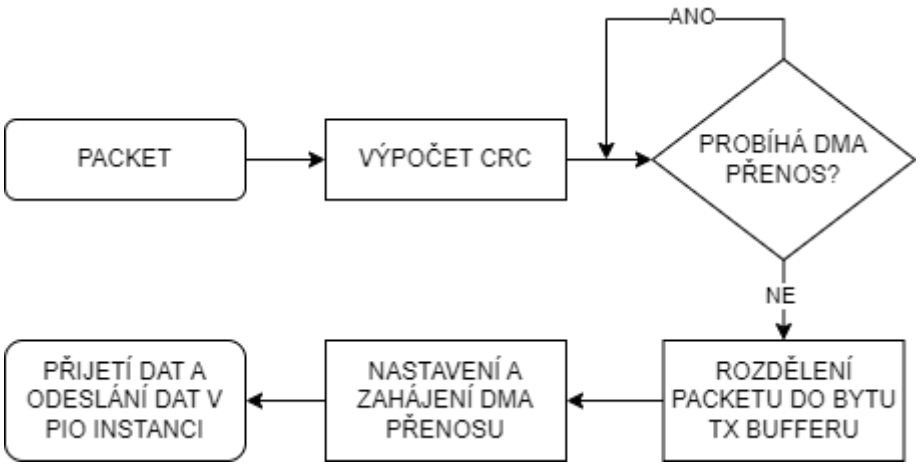
Obrázek 4.3 Hodnota kontrolního registru KSZ8081MLX pro nastavení rychlosti ethernetu

Výsledné data v hexadecimální podobě, které budu vysílat pro jednotlivé rychlosti jsou 0x0000 (10 Mbps) a 0x2000 (100 Mbps). K zapsání hodnot do registru jsem využil funkci *mii_mdio_write*, která je dohledatelná v souboru *miim.c* na mém GitHub repozitáři [16]. Po zapsání hodnoty 0x0 do kontrolního registru čipu PHY bliká u ethernetového konektoru RJ45 pouze jedna oranžová LED dioda (tedy při toku dat), druhá LED dioda nesvítí. Při zapsaní hodnoty 0x2000 taktéž problikává oranžová LED dioda, a navíc svítí zelenou barvou druhá LED dioda (tedy zelená barva označuje rychlost ethernetového připojení 100 Mbps).

Tímto jsem ověřil funkcionalitu MIIM rozhraní. Taktéž byl ověřen návrh desky plošného spoje, neboť se jednalo o první test čipu PHY a ethernetového konektoru.

4.2 Ethernet funkce

Funkce pro odeslání packetu po ethernetu je znázorněna na obrázku 4.4. Kód ethernet funkce je k nahlédnutí v souboru *ethernet.c*. Do funkce je předán odesílaný packet a jeho velikost. Jako první se provede kontrola velikosti packetu tak, aby odpovídal IEEE standardu. Pokud je tedy předávaná hodnota velikosti menší jak 60, je automaticky nastavena na hodnotu 60 (realizuje doplnění nulami). Poté je proveden výpočet CRC, jehož hodnota bude odeslána společně s daty packetu. Následně se čeká na případné dokončení předešlého přenosu na daném DMA kanálu. Pokud neprobíhá nebo byl přenos dokončen, je do TX bufferu naformátován odesílaný packet. Pro prvotní oživení bylo zvoleno formátování bufferu, kdy jednotlivý byte v sobě nese 4 bity dat a 1 bit pro hodnotu TXEN. Toto řešení není úplně optimální, ale dovoluje mi si výrazně usnadnit PIO kód. Následně je nastaven kanál DMA, který odstartuje přenos dat do TX FIFO příslušného stavového automatu v dané PIO instanci (viz Obrázek 1.4). Po přijetí těchto dat začne stavový automat vykonávat PIO program, kterým jsou data interpretována do čipu PHY.



Obrázek 4.4 Průběh odesílání ethernetového packetu

Při plnění bufferu jsem narazil na chybu, která se stala při návrhu DPS. Při inicializaci pinů v PIO konfiguraci jsou piny inicializovány po sobě od předávaného pinu (který slouží jako base) a jejich pořadí nelze změnit.

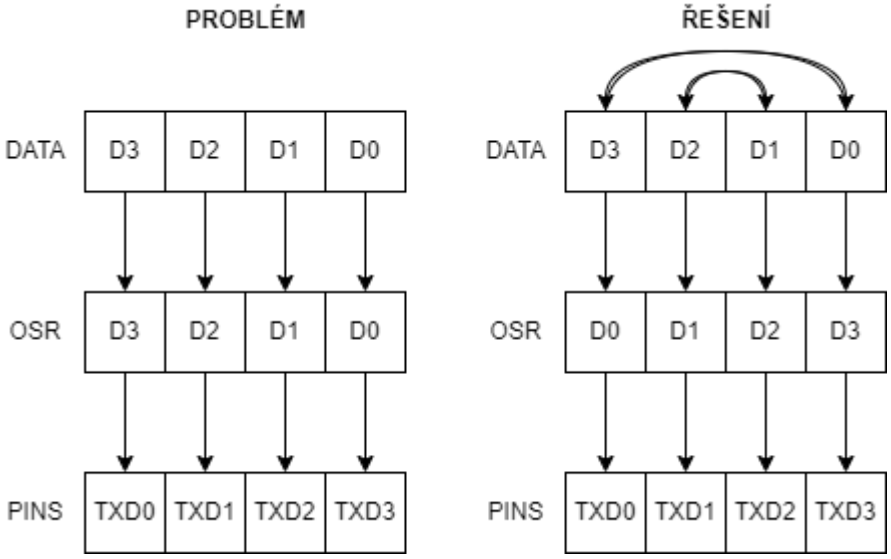
Příkladem může být nastavení pinů pro instrukci OUT (která nastaví počet pinů podle hodnot v OSR). Pro moji desku jsou piny pro odesílání navržený v následujícím pořadí:

```
/*********************************************************************
#define KSZ_TXEN _u(9)
#define KSZ_TXD3 _u(10)
#define KSZ_TXD2 _u(11)
#define KSZ_TXD1 _u(12)
#define KSZ_TXD0 _u(13)
#define KSZ_TXC _u(14)
/**********************soubor: defines.h [13-18]**********************
```

Při nastavení pinů, ovládaných instrukcí OUT je předána báze a počet pinů, který má být nastaven. Tedy při použití báze 10 a počtu pinů 4 jsou nastaveny piny v pořadí: TXD3, TXD2, TXD1 a TXD0. Pří použití instrukce „*out pins 4*“ Jsou 4 bity z OSR přivedeny na piny v pořadí, v jakém byly nastaveny. Tedy pro můj případ bude bit 0 přiveden na TXD3, bit 1 na TXD2… Při odeslání hodnoty A (1010) bych tedy na čipu PHY obdržel hodnotu 5 (0101).

Další ulehčení by bylo, kdyby piny TXEN a TXC byly navrženy vedle sebe (tedy třeba pin 9 a 10). Jelikož PIOASM umožňuje nastavení pinů pomocí funkcionality side-set, která je vykonána paralelně s prováděnou instrukcí. Tím je ušetřen jeden cyklus a zároveň jedno místo v paměti instrukcí, neboť tato funkcionalita je součástí prováděné instrukce [1]. V mém případě by však side-set dvou pinů nastavil bity TXEN a TXD3 namísto TXEN a TXC.

Nejedná se však o chyby, které by zamezily funkčnosti. Pouze mi ztížila realizaci firmwaru. Problém řeším tak, že při formátování TX bufferu je jednotlivý nibble dat vložen v přeházeném pořadí. Řešení je znázorněno na obrázku 4.5.



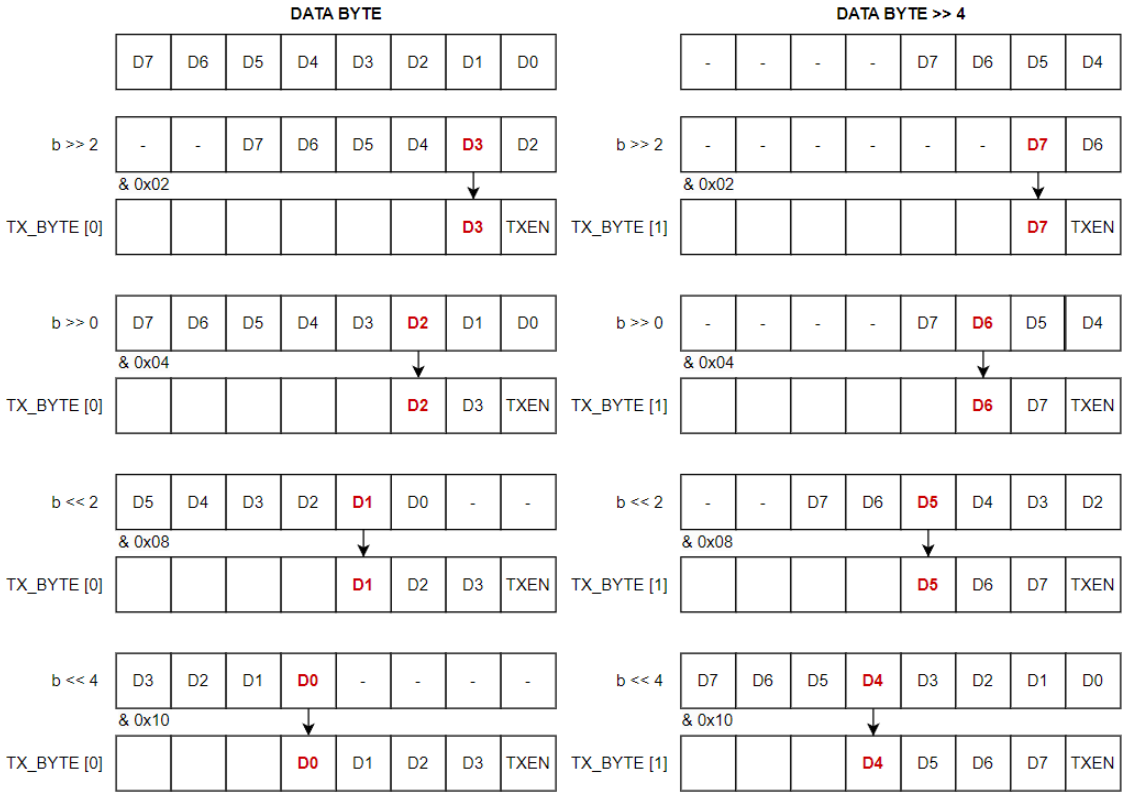
Obrázek 4.5 Prohození pořadí pinů pro PIO

Část kódu pro plnění bufferu TX poté vypadá následovně:

```
/*********************************************************************
// PREAMBLE
    // D0   D1   D2   D3   TXE
    // [1]  [0]  [1]  [0]  [1]
    for (int i = 0; i < 15; ++i)
        tx_frame_bits[index++] = 0x15;


    // SFD
    // D0   D1   D2   D3   TXE
    // [1]  [0]  [1]  [1]  [1]
    for (int i = 0; i < 1; ++i)
        tx_frame_bits[index++] = 0x17;

    // DATA
    for (int i = 0; i < length; ++i){
        uint8_t b = tx_buffer[i];
        tx_frame_bits[index++] = 0x01 | ((b >> 2) & 0x02)
                                      | (b & 0x04)
                                      | ((b << 2) & 0x08)
                                      | ((b << 4) & 0x10);
        b >>= 4;
        tx_frame_bits[index++] = 0x01 | ((b >> 2) & 0x02)
                                      | (b & 0x04)
                                      | ((b << 2) & 0x08)
                                      | ((b << 4) & 0x10);
    }
/*********************soubor: ethernet.c [40-76]*********************
```

Při plnění dat je první bit nastaven na 1 a odpovídá signálu TXEN, který bude po celý čas odesílaní nastaven v úrovni 1. Poté je proveden bitový posun celého bajtu o 2 doprava společně s bitovým AND, který mi „propustí“ hodnotu na druhém bitu (0x02). Pro lepší

představu je tento postup znázorněn na obrázku 4.6. Jako poslední je obdobným způsobem vložena hodnota CRC.



Obrázek 4.6 Plnění dat do TX bufferu

Po doplnění dat nastavím kanálu DMA source a destination adresy (tedy TX buffer a FIFO buffer stavového automatu) a počet přenášených bytů.

4.3 PIO program

První PIO program se skládá z pouhých 3 instrukcí, každá z instrukcí obsahuje dodatečně side-set, kterým tvořím takt hodin TXC.

```
/*****************************************************************************
.program mii_tx
.side_set 1
.wrap_target
    nop          side 0
    out pins 5   side 0 [1]  ; 16 / 160 ns setup time (with 125 MHz clk)
    nop          side 1 [1]
.wrap
/*********************soubor: mii_tx.pio [68-74]**********************
```

Instrukci nop využívám k vytvoření zmíněného taktu hodin. Při testování jsem zjistil, že hodiny nemusí být symetrické. Tím mi je umožněno provozovat mikrokontrolér na 125 MHz místo 100 MHz. Jeden cyklus je vykonán v 8 nanosekundách.

První *nop* instrukce zabere jeden cyklus a nastaví TXC do nuly pomocí side-setu. Následně je vykonána instrukce *out*, která 5 bitů z OSR přivede na pin TXEN a TXD0-3. Zároveň drží signál TXC v nule s přidáním zpoždění o jeden cyklus pomocí „[1],“ dohromady trvá 2 cykly. Jako poslední je instrukce *nop*, která pouze nastaví signál TXC do jedničky a přidává zpoždění o 1 cyklus a taktéž zabere 2 cykly.

V PIO kódu je potřeba dodržet časování dat na pinech TXD[3:0] vůči TXC. Výrobce čipu PHY tyto doby pro TX režim stanovuje na 10 ns (100 Mbps) a 120 ns (10 Mbps) [11]. Instrukci *out* provádí stavový automat v 2 cyklech s TXC v nule před jeho nastavením, tím je výsledná doba nastavení dat před vzestupnou hranou TXC 16 ns (100 Mbps) nebo 160 ns (10 Mbps). Držení hodnoty po vzestupné hraně TXC je nulové. Na modulu RPP je signál dat delší jak signál TXC a dochází tedy k fázovému posuvu. To má za následek to, že vzorkování dat proběhne o trochu dřív, jak náběh hrany TXC.

Asemblerový kód PIO je tedy tvořen 5 cykly. Co 5 cyklů je odeslán jeden nibble dat. Při systémových hodinách 125 MHz je perioda vykonávání celého programu 25 MHz. Po vynásobení 4 bity je tedy dosaženo požadované rychlosti 100 Mb za sekundu.

Další částí je nastavení stavového automatu, který bude daný PIO kód vykonávat.

```
/*********************************************************************
static inline void mii_10mhz_tx_init(PIO pio, uint sm, uint offset, uint
pin) {
    pio_gpio_init(pio, pin);        // TXEN
    pio_gpio_init(pio, pin + 1);    // TX3
    pio_gpio_init(pio, pin + 2);    // TX2
    pio_gpio_init(pio, pin + 3);    // TX1
    pio_gpio_init(pio, pin + 4);    // TX0
    pio_gpio_init(pio, pin + 5);    // TXC

    pio_sm_set_consecutive_pindirs(pio, sm, pin, 6, true);

    pio_sm_config c = mii_10mhz_tx_program_get_default_config(offset);
    sm_config_set_out_pins(&c, pin, 5);
    sm_config_set_sideset_pins(&c, pin + 5);

    sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_TX);
    sm_config_set_out_shift(&c, true, true, 5);

    sm_config_set_clkdiv(&c, 10);

    pio_sm_init(pio, sm, offset, &c);
    pio_sm_set_enabled(pio, sm, true);
}
/*******************soubor: mii_tx.pio [102-123]*********************
```

V tomto kódu probíhá základní inicializace. Za zmínku spíše stojí dodatečná konfigurace spojení RX i TX FIFO stavového automatu na jeden velký TX FIFO, neboť tento stavový automat bude pracovat pouze v módu TX. Dále je využito funkce „autopull“ ve funkci *sm_config_set_out_shift*, tato funkce taktéž nastavuje směr plnění OSR (tedy zleva nebo zprava) a hraniční hodnotu pro autopull (v mém případě 5).

Díky autopullu je hardwarově zajištěno automatické načítání hodno dat z TX FIFO do OSR. Není tedy potřeba použití instrukce *pull* a ušetří se místo v instrukční paměti PIO instance. Pro můj první PIO kód tento detail není tak důležitý, neboť kód vyplňuji pseudoinstrukcí *nop* pro vytvoření hodinového taktu, a tedy první instrukce *nop* by mohla být vyměněna za instrukci *pull*. Při tvorbě optimalizovaného PIO programu je však možné, že se tato funkcionalita využije.

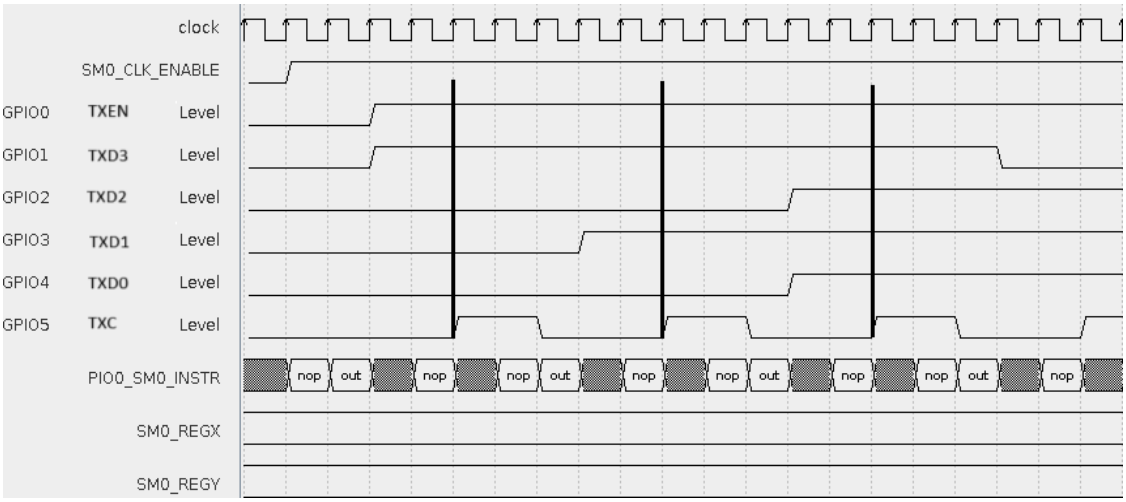
Poslední a důležitá funkce je nastavení dělícího poměru systémových hodin. Mezi nastavením pro 10 Mbps a 100 Mbps je rozdíl právě funkce *sm_config_set_clkdiv,* která nastaví dělící poměr. Výchozí PIO program tvoří rozhraní 100 Mbps. Pro vytvoření 10 Mbps stačí tedy pouze referenční hodiny systému podělit 10 (Jeden cykl stavového automatu bude trvat 10 cyklů systémových hodin). Tím dostávám periodu vykonávání PIO programu 2.5 MHz.

Jelikož nelze PIO programy debugovat za běhu, využil jsem pro ověření a zároveň vytvoření časového diagramu volně dostupný emulátor PIO instance [18]. Při buildnutí emulátoru na operačním systému Windows nastalo k chybě, jelikož soubory makefile jsou psány pro platformu Linux a v těchto souborech měl Windows problém se zaměněním dvojteček za středníky (bylo by nutné v každém makefile souboru tyto syntaxe zaměnit). Emulátor jsem tedy musel spustit na virtuálním PC s operačním systémem Debian 12.

Na obrázku 4.7 je časový diagram PIO programu. Jednotlivé bity jsou ve stejném pořadí, jako na mém modulu, pouze nemají příslušné offsety. V emulátoru je inicializováno FIFO na ručně zadané hodnoty:

fifo --pio=0 --sm=0 --enqueue --tx --value 0x00000003
fifo --pio=0 --sm=0 --enqueue --tx --value 0x0000000b
fifo --pio=0 --sm=0 --enqueue --tx --value 0x0000001f
fifo --pio=0 --sm=0 --enqueue --tx --value 0x0000001d

Lze vidět, že je dodrženo mezery dvou cyklů před náběžnou hranu TXC a střídy 3/2. Při druhém náběhu TXC (svislá černá čára) jsou nastaveny bity TXEN, TXD3 a TXD1. V této posloupnosti se jedná o binární hodnotu „01011“ odpovídající hexadecimální hodnotě 0x0B, která je shodná s druhou hodnotou v FIFO. Tímto jsem si ověřil návrh mého PIO programu.



Obrázek 4.7 Časový diagram PIO programu

4.4 Testování ethernetu

Pro otestování ethernetu bude program neustále odesílat neměnný packet, který se pokusím zachytit na koncovém zařízení pomocí programu Wireshark.

Testovací packet jsem si vytvořil pomocí programu Netcat. Na své domácí lokální síti jsem si v konzoli Windows PowerShell poslal testovací Netcat packet z jednoho počítače na druhý. Jeden počítač sloužil k naslouchání UDP packetů na zvoleném portu pomocí skriptu:

ncat -ul -p 1234.

Druhý počítač odeslal testovací packet skriptem:

echo "Testovaci zprava z NetCat" | ncat -u -p 192.168.1.50 1234.

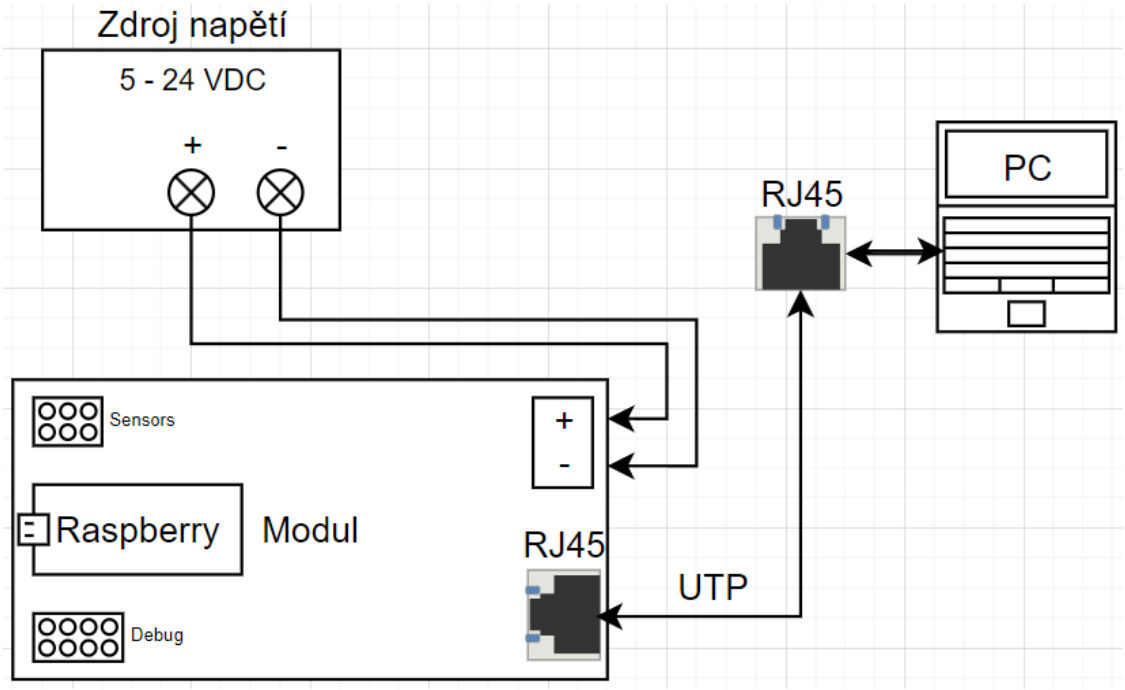
Tento packet jsem zachytil na konzoli naslouchajícího počítače a zároveň jsem jej zachytil v programu Wireshark, z kterého jsem si poté jednotlivé hodnoty bytů opsal a definoval v mém kódu (viz níže).

```
/**********************************************************************
// Test packet
static uint8_t netcat_packet[] = {
    //  Ethernet II
    0xC8, 0x7F, 0x54, 0xA5, 0x83, 0x5D,     // Destination MAC
    0xC8, 0x7F, 0x54, 0x69, 0x8A, 0xDA,     // Source MAC
    0x08, 0x00,                             // Type (IPv4)
    //  IP
    0x45, 0x00,                   // Header Length, Diff. Services Field
    0x00, 0x37,                             // Total Length
    0xA2, 0xD8,                             // Identification
    0x00, 0x00,                             // Fragment Offset
    0x80,                                   // Time To Live
    0x11,                                   // Protocol (UDP)
    0x13, 0xD9,                             // Header checksum
    0xC0, 0xA8, 0x01, 0x82,                 // Source Address
```

```
    0xC0, 0xA8, 0x01, 0x32,                 // Destination Address
    //  UDP
    0xD5, 0x2A,                             // Source Port
    0x04, 0xD2,                             // Destination Port
    0x00, 0x23,                                 // Length
    0xD4, 0xF3,                                 // Checksum
    0x54, 0x65, 0x73, 0x74, 0x6F, 0x76, 0x61,
    0x63, 0x69, 0x20, 0x7A, 0x70, 0x72,         // Data
    0x61, 0x76, 0x61, 0x20, 0x7A, 0x20, 0x4E,
    0x65, 0x74, 0x43, 0x61, 0x74, 0x0D, 0x0A    // Data
    };
/*********************soubor: defines.h [73-96]***********************
```

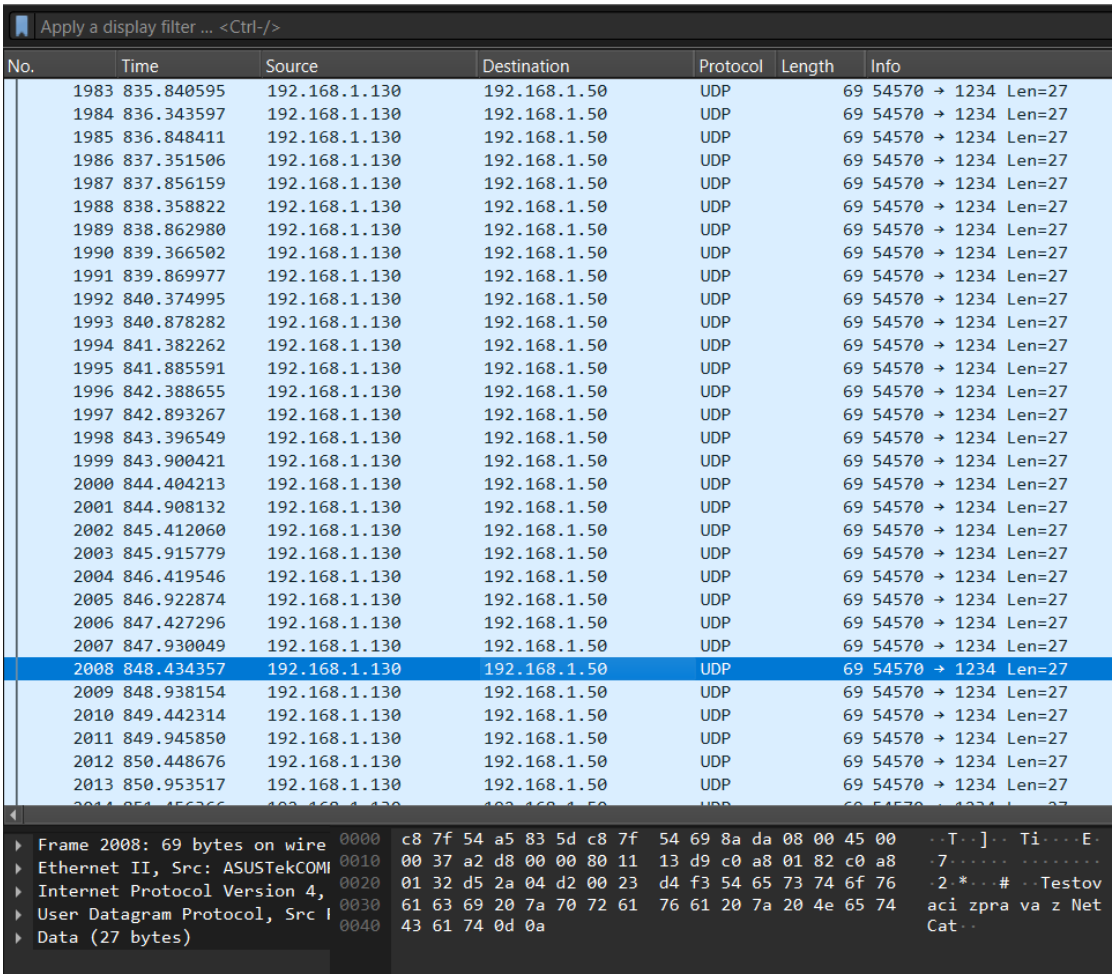
Výhoda takto vytvořeného packetu z Netcat je, že díky jeho neměnnosti vím, že hodnoty checksumů v různých vrstvách jsou správné a nijak se mi při testování nezmění. Packet následně předávám zmíněné ethernet funkci, která jej vhodně naformátuje do TX bufferu a zahájí DMA přenos.

Následně je potřeba modul zapojit podle obárzku 4.8 a na přijímacím zařízení spustit program Wireshark, který bude sledovat činnost ethernetového portu zařízení. Do modulu je potřeba taktéž nahrát firmware. První možností je nahrání uf2 souboru, dostupného z mého GitHub repozitáře v sekci „releases“ [16]. Druhá možnost je nahrát samotný zdrojový kód pomocí SWD rozhraní. K tomuto je využit debugger, kterým lze navíc běh firmwaru krokovat a sledovat. V mém případě se jedná o druhé RPP, které má v sobě nahraný firmware picoprobe, realizující most USB na SWD. Detailnější návod, jak využít druhé RPP jako debugger, je v dokumentaci „Getting started with Raspberry Pi Pico“ dodatek A [19].



Obrázek 4.8 Propojeni modulu k testování ethernetu

Po realizaci zapojení dle obrázku 4.8 a nahrání firmwaru do modulu jsem úspěšně odchytil odesílaný packet v programu Wireshark. Zachycení lze vidět na obrázku 4.9, kde v hlavním okně jsou vidět přijaté packety. V dolním pravém rohu jsou poté jednotlivé byty jednoho zvoleného packetu a napravo od nich je vidět jejich převedení na znaky. Z prvních znaků nelze vyčíst nic rozumného, neboť se jedná právě o zmíněné hlavičky, adresy apod. Na konci jsou však již samotná data a lze vidět správný příjem zprávy „Testovaci zprava z NetCat“.



Obrázek 4.9 Zachycení packetu v programu Wireshark

Zde jsem ověřil další a zásadní funkčnost mého modulu. Testování jsem provedl jak pro 10 Mbps, tak pro 100 Mbps.

5. OPTIMALIZACE ETHERNETU

Po úspěšném otestování základní ethernet funkce je dalším krokem optimalizovat PIO program i funkci pro formátování ethernetových packetů. Rovněž bude zprovozněno ADC pro vzorkování hodnot snímačů vzdálenosti. Tyto hodnoty se poté přepíší do nového packetu, který bude taktéž odeslán do koncového zařízení.

5.1 Optimalizovaný PIO program

Cílem optimalizace PIO programu je odlehčit ethernetové funkci, která do TX bufferu plní preamble, SFD a IFG sekvenci. Důvodem je to, že tyto sekvence jsou vždy stejné a neměnné. Bude tedy výhodné je tvořit přímo uvnitř samotného PIO programu.

```
/*****************************************************************
; Overlap pins with OUT / SET instruction for Preamble + SFD part
;
;            TXC     TX0     TX1     TX2     TX3     TXEN
;             ^       ^       ^       ^       ^       ^
;             |       |       |       |       |       |
;            side    out     out     out     out     set
;                     +       +       +       +
;                    set     set     set     set
.program mii_opt_tx
.side_set 1 opt
.wrap_target
    pull block          side 0      ; pull counter val
    out null, 24
    mov y, osr                      ; move counter to scratch Y
    out null, 8                     ; discrad the counter after storing
;                                   ; it into scratch Y
    set x, 14                       ; set preamble counter

    preamble:
    set pins 0b10101    side 0 [2]
    jmp x-- preamble    side 1 [1]

;   SFD
    set pins 0b10111    side 0 [2]
    set x, 23           side 1 [1]  ; Prepare X counter for IFG

    data:                           ; data + crc
    pull block          side 0
    out pins 4          side 0 [1]  ; out 4 bits from OSR
    nop                 side 1 [1]

    out pins 4          side 0 [2]  ; out 4 bits from OSR
    jmp y-- data        side 1 [1]

    ifg:
    set pins 0b00000    side 0 [2]
    jmp x-- ifg         side 1 [1]
    ;irq wait 0          side 0     ; Used for monitoring in C code
.wrap
/*********************soubor: mii_tx.pio [1-40]**********************
```

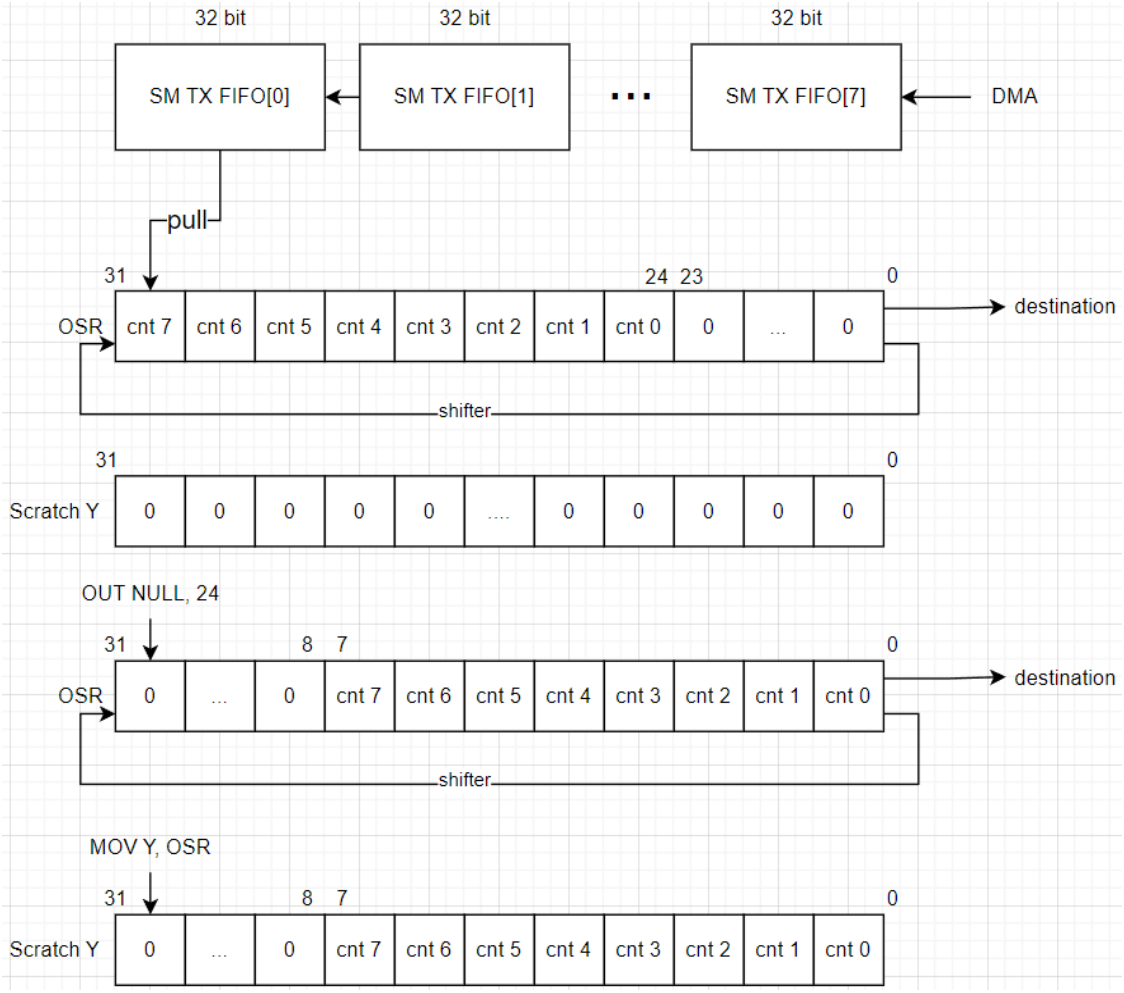
V tomto programu je využito instrukce *set*, kterou lze nastavit piny na zvolenou hodnotu. Výhoda této instrukce je, že může ovládat piny, které jsou zároveň ovládány instrukcí *out*. Přehled instrukcí, které řídí jednotlivé piny, je na samém začátku kódu. Pro debugování je možné odkomentovat poslední instrukci, která nastaví bit přerušení PIO instance a čeká na jeho vynulování. Tím je signalizováno správné vykonání programu. Zvenčí je možné sledovat stav bitu přerušení dané instance (sekce 5.3).

Program první hodnotu z OSR vloží do scratch registeru Y, tato hodnota bude reprezentovat počet bytů, který packet obsahuje. Po pullnutí hodnoty z FIFO do OSR je nejprve prvních 24 bitů zahozeno, neboť instrukce *mov* nerespektuje nastavený směr shiftování bitů jako u instrukce *out*. Následně si posledních 8 bitů (které odpovídají počtu odesílaných bytů) nakopíruje do zmíněného registru a následně obsah OSR vymaže další instrukcí *out null,* kde null specifikuje zahození daného počtu bitů. Chování je znázorněno na obrázku 5.1.

Následně je poslána preamble a SFD. Po SFD je výhodné nastavit scratch registr X na 23. Jedná se o počet IFG nibblů, které od sebe oddělují jednotlivé packety. Tímto efektivně zároveň vytvořím náběžnou hranu TXC. Po SFD následuje nastavování bitů dat a ke konci CRC. Během tohoto je každým cyklem dekrementován scratch registr Y, držící počet bytů k odeslání. Nakonec je odeslána pauza IFG a program se wrapem vrací na začátek, kde čeká na příjem dat FIFO, signalizující nový packet.

Program bohužel nemůže využít funkci autopull, jelikož by po instrukci *out null, 24* byl překročen treshold a do OSR by byla automaticky vložena nová data před vykonáním instrukce *mov*, tím bych ztratil počet přenášených bytů. Dále je toto řešení limitováno na packety, kde počet bytů dat a CRC je menší nebo rovna hodnotě 255. V mém případě předem vím, že toto bude dodrženo. Řešení však bude spočívat v proházení bitů a bytů ve funkci, která formátuje packet do TX bufferu (tím by bylo zároveň umožněno použití funkce autopull) a tím umožní vkládat místo 8 bitové hodnoty 32 bitovou hodnotu.

Konfigurace optimalizovaného PIO programu se nijak zásadně neliší od původního PIO programu, změna je pouza nastavením *set* instrukce, vypnutí autopullu a zmenšení pinů, které jsou řízeny instrukcí *out* o jedna.



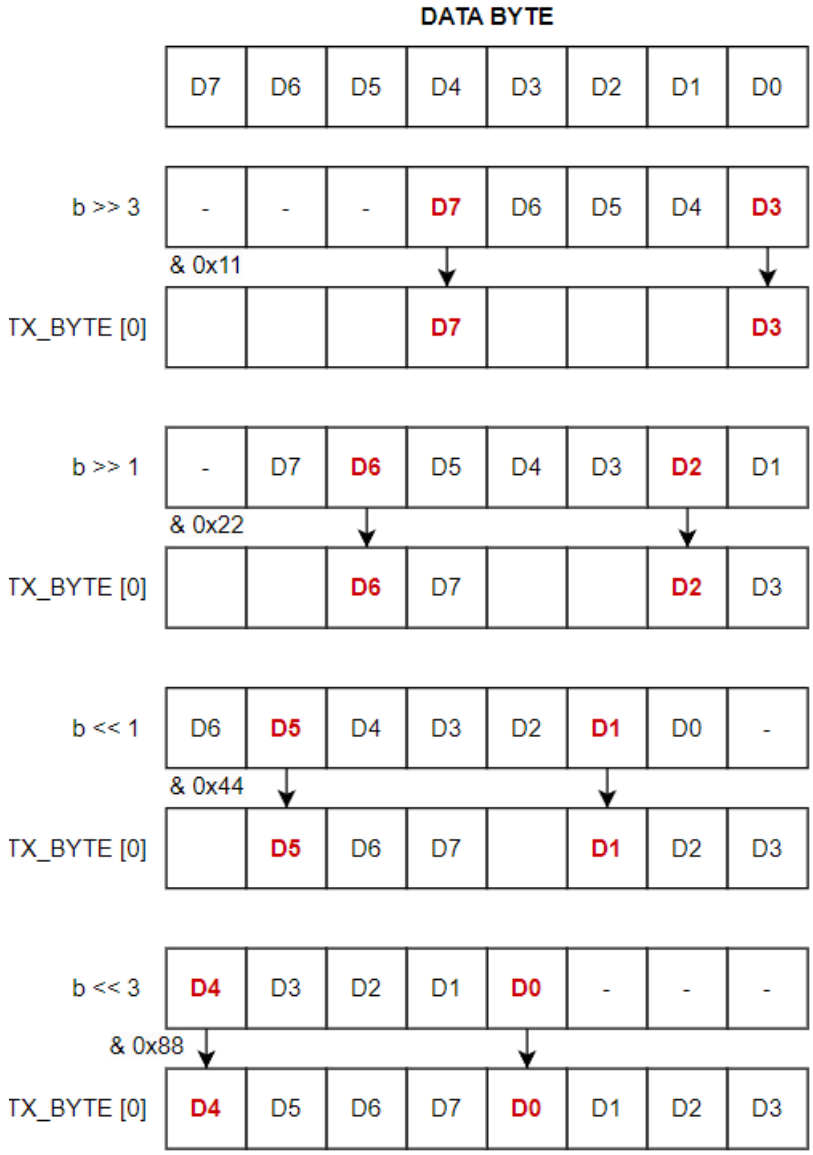
Obrázek 5.1 Načtení počtu přenášených bytů

5.2 Optimalizovaná ethernet funkce

Ethernetová funkce rezervuje první byte v TX bufferu pro hodnotu přenášených bytů packetu. Jedná se o hodnotu, kterou si PIO program načítá do scratch registru Y.

Funkce nyní plní pouze data bez statické hodnoty TXEN. Je však opět potřeba provést prohození bitů nibblů. Operace je však díky tomuto přehlednější, něž v případě první ethernet funkce. Prohození je znázorněno na obrázku 5.2, výhodou také je, že nyní není potřeba jeden byte rozdělit do dvou bytů jako v případě s hodnotou TXEN.

```
/*********************************************************************
void mii_ethernet_output_opt(uint8_t* tx_buffer, int length){
    // pad
    if (length < 60)
        length = 60;

    uint crc = ethernet_frame_crc(tx_buffer, length);

    uint8_t index = 1;  // Reserve first value for index after we feed
;                          in the data
                        // If the packet exceeds length of 255 there is
;                          a need
                        // to change the uint8_t type and topology

    // DATA
    for (int i = 0; i < length; i++){
        uint8_t b = tx_buffer[i];
        tx_frame[index++] =             ((b >> 3) & 0x11)
                                      | ((b >> 1) & 0x22)
                                      | ((b << 1) & 0x44)
                                      | ((b << 3) & 0x88);
    }

    // CRC
    for (int i = 0; i < 4; i++){
        uint8_t b = ((uint8_t*)&crc)[i];
        tx_frame[index++] =             ((b >> 3) & 0x11)
                                      | ((b >> 1) & 0x22)
                                      | ((b << 1) & 0x44)
                                      | ((b << 3) & 0x88);
    }

    // Decrement by two since first byte is sent before
    // Decrementing loop counter and one byte is the
    // counter value itself
    tx_frame[0] = index-2;

    dma_channel_configure(
        tx_dma, &tx_dma_config,
        &pio0->txf[sm_tx],
        tx_frame,
        index,
        true
    );
}
/********************soubor: ethernet.c [105-147]**********************
```



Obrázek 5.2 Plnění dat do TX bufferu optimalizované ethernet funkce

5.3 ADC a snímače

Kód ADC bude prováděn druhým procesorem mikrokontroléru RPP. Tím zavedu druhé vlákno, jehož činnost mi nebude zatěžovat hlavní vlákno. Toto lze provést zavoláním funkce *multicore_launch_core1(adc_main)*. Tato funkce inicializuje stack druhého jádra a předá ze vstupního parametru počáteční adresu kódu, který bude na tomto jádře vykonáván.

Můj návrh bude využívat vzorkování „one-shot.“ Neboť hodnoty snímačů budu vzorkovat přibližně 10x za sekundu a zcela jistě se tím nepřiblížím k vzorkování až 500 ksps (viz sekce 1.1.2). Nemá tedy smysl využívat vyrovnávací paměť FIFO. Vytížení ADC čipu v mém případě bude tedy nízké. Pro navzorkované hodnoty bude

vytvořena funkce, která je přepíše do odesílaného packetu. Bude potřeba zajistit, aby při provádění formátování packetu do TX bufferu nedošlo k navzorkování nových hodnot, které by následně přepsaly hodnoty staré. V krajním případě by tak mohlo nastat, že by hodnota prvního snímače obsahovala vzorek z meření *X* a před vložením hodnoty druhého snímače z meření X by proběhlo meření X+1, které by zapříčinilo vložení hodnoty z vzorkování X+1. Výsledný packet by tedy obsahoval hodnoty z dvou měření namísto jednoho. Tomuto zabráním vytvořením proměnné typů volatile bool „sampling," která bude udávat tempo mezi dvěma jádry (vlákny).

Níže jsou uvedeny části kódu ADC vlákna a funkce vykonávající vzorkování snímačů. Deklarace struktur snímačů a další náležitosti lze nalézt v mém GitHub repozitáři (soubor sensors.h) [16]. Snímače jsem k desce připojil shodně podle rozložení pinů pinheaderu J2 v obrázku 3.7.

```
/*********************************************************************
uint8_t sensor_sample(struct sensor* s){
    uint left_index = 0;
    adc_select_input(s->adc_input);
    s->sample_val = (float)adc_read()/ADC_12B_LEVEL*ADC_PIN_LEVEL;

    printf("sample val = %.2f ", s->sample_val);

    // Saturate min & max
    if(s->sample_val > s->max_val) s->sample_val = s->max_val;
    else if(s->sample_val < s->min_val) s->sample_val = s->min_val;

    uint right_index = 1;
    for(; right_index < s->size; ++right_index){
        if(!(s->sample_val < s->lut_volt[right_index])){
            left_index = right_index-1;
        break;
        }
    }

    //                example for values between 50 - 60 cm
    //                60   - (60-50)     * (val-1.05) / (1.25-1.05)
    //                i[2] - (i[2]-i[1]) * (val-i[2]) / (i[1]-i[2])
    return
(uint8_t)((float)s->lut_cm[right_index]-((float)s->lut_cm[right_index-
(float)s->lut_cm[left_index])*(s->sample_val-
s->lut_volt[right_index])/(s->lut_volt[left_index]-
s->lut_volt[right_index]));
}
/*******************soubor: sensors.c [8-32]************************
```

Ve funkci je využito maker ADC_12B_LEVEL a ADC_PIN_LEVEL. První makro udává rozlišení ADC převodníku což pro 12 bitovou hodnotu je rozmezí 0 až 4095. Druhé makro specifikuje maximální úroveň, která se na pinu může vyskytnout. Softwarově tento limit neměním a je tedy zanechána základní hodnota 3.3 V, která je vhodná pro oba zvolené snímače.

Po navzorkování hodnoty vstupního pinu je hodnota převedena z bitové hodnoty ADC_{sample} na hodnotu napětí U_s vzorcem 5.1.

$$U_s = \frac{ADC_{sample}}{4095} \cdot 3.3 \text{ [V]}, \tag{5.1}$$

Pro přepočet napětí na vzdálenost jsem si definoval tabulky napětí a vzdáleností z datasheetů použitých snímačů [20][21]. Příklad výpočtu je zakomentován na konci funkce před klíčovým slovem *retrun,* kde proměnná *val* označuje hodnotu napětí získanou vzorcem 5.1.

```
/*********************************************************************
////////////////
// ADC Thread //
////////////////

void adc_main(void){
  adc_init();
  adc_gpio_init(RPP_SENSOR1);
  adc_gpio_init(RPP_SENSOR2);
  struct sensor sensor0 = sensors_init(0);
  struct sensor sensor1 = sensors_init(1);

  uint32_t packet_count = 0;
  while(true){
    // Important especially when it comes to RELEASE mode
    dma_channel_wait_for_finish_blocking(tx_dma);

    sampling = true;
    sleep_ms(100);

    //while(pio_interrupt_get(pio0, 0)) printf("PIO IRQ!\n\n");

    write_sensor_data(sensor_sample(&sensor1), sensor_sample(&sensor2),
                                                    ++packet_count);
    sampling = false;
  }
}
/*******************soubor: main.c [8-32]****************************
```

Vlákno ADC inicializuje jednotlivé vstupní piny a samotnou instanci ADC. Dále je volána funkce pro inicializaci struktur snímačů (přiřazení tabulky hodnot z datasheetu, vstupního kanálu…). Následuje hlavní smyčka, kde si lze všimnout udávání tempa dvou vláken pomocí bool proměnné *sampling.* Lze odkomentovat a využít sledování činnosti PIO programu, jak je zmíněno v sekci 5.1. Nakonec je zde funkce, která vkládá hodnoty měřené vzdálenosti na pevně určené místo v datech packetu. Pro detailnější náhled využitých funkcí je možné nahlédnout do mého GitHub repozitáře (soubory sensors.c a ethernet.c) [16].

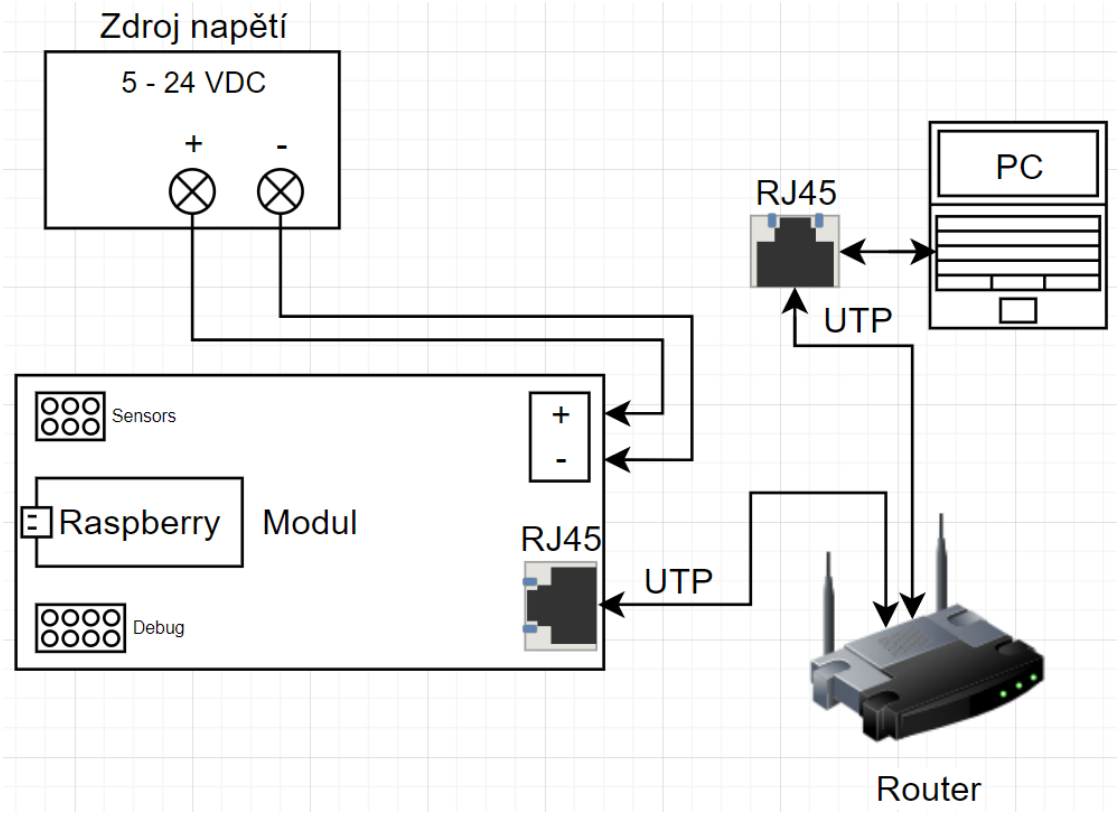
6. APLIKACE

Po optimalizaci a otestování firmwaru je posledním krokem vytvoření reálné aplikace. Aplikace bude vytvořena v programu Netcat. Bude se jednat o interpretaci dat ze snímačů a počet odeslaných packetů.

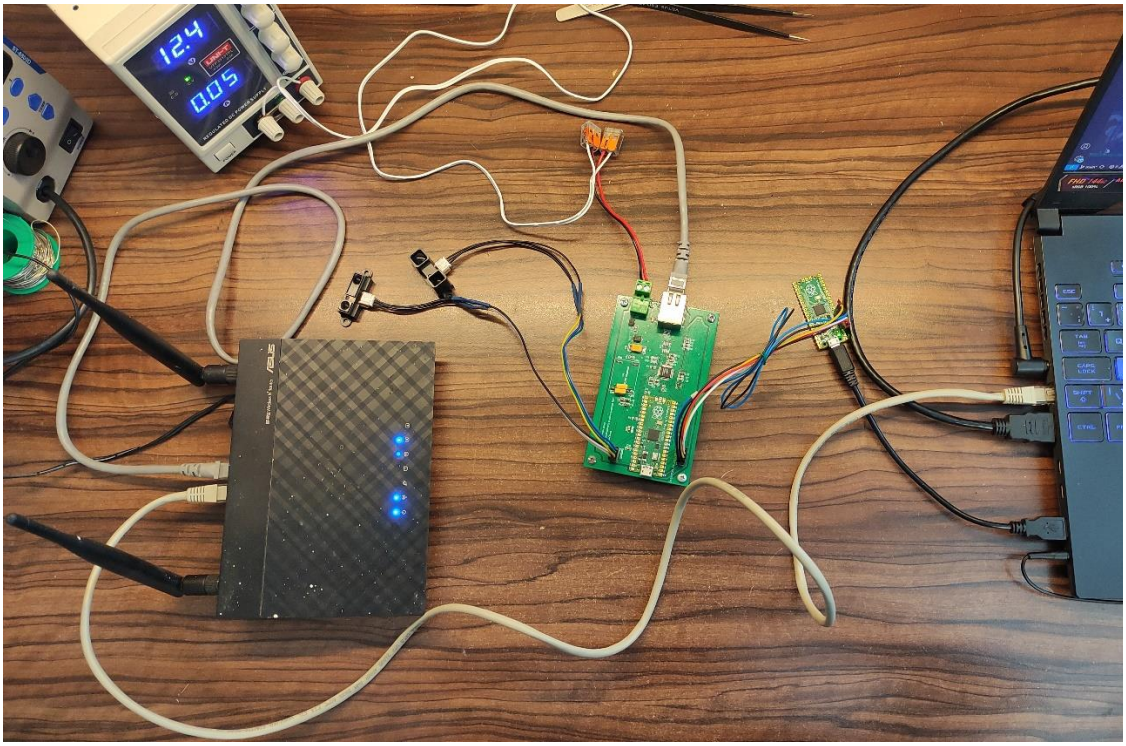
Data ze snímačů jsou proměnná a odesílaný packet tedy není již statický. Tato bakalářská práce se zaobírá implementací ethernetu, ten zahrnuje první dvě vrstvy z celého OSI modelu. Třetí vrstva (IP) v sobě nese kontrolní hodnotu samotné IP hlavičky, která se nijak nemění. Problém však vznikne u čtvrté vrstvy (transportní vrstva), která ve své hlavičce nese kontrolní hodnotu odesílaného packetu. Jelikož data ze snímačů se v čase mění, bude kontrolní hodnota packetu vždy jiná. To znamená, že by bylo nutné implementovat další funkci, která tuto hodnotu vypočte a uloží do odesílaného packetu. Jedná se však o funkci vyšší vrstvy, nespadající do vrstev ethernetu (a tedy samotné práce). Lze však říci koncovému zařízení, že tato kontrolní suma se nebude provádět (bude ji ignorovat). Toto chování se konfiguruje vložením hexadecimální hodnoty 0x0000 na pozici kontrolní hodnoty transportní vrstvy v packetu (v sekci 4.4 by se jednalo o změnu hodnoty z 0xD4, 0xF3 na 0x00, 0x00 v části UDP – checksum u testovacího packetu).

Jelikož práce zahrnuje pouze určitou část vrstev ISO modelu, je tento modul určen pro činnost s použitím dalších síťových prvků jako je switch / bridge / router (které poskytují další funkce ISO, třeba pro odchycení porušeného packetu). Pro funkčnost modulu však nejsou nutností. Takovéto zapojení je poté znázorněno na obrázku 6.1, který se od zapojení 4.8 liší přidáním routeru mezi modulem a koncovým zařízením (PC).

Finální aplikaci jsem zapojil pomocí obrázku 6.1. Dodatečně jsou k pinheaderům připojený zmíněné snímače vzdálenosti a druhé RPP, sloužící jako debugger mého firmwaru. Jelikož nemám k dispozici switch nebo bridge, používám v zapojení router. Rozdílem u směrování packetů je ten, že router pracuje na 3. vrstvě (síťová) a směruje je podle adresy IP. Switch a bridge pracují na 2. vrstvě (linková) a směrují packety na základě adresy MAC. Fyzické zapojení aplikace je poté znázorněno na obrázku 6.2.



Obrázek 6.1 Propojení modulu při aplikaci



Obrázek 6.2 Fyzické propojení modulu při aplikaci

Při testování ethernetu programem Wireshark nebylo nutné dodržovat některé detaily, neboť tento program sleduje veškeré dění na zvoleném portu (zaznamenává packety s odlišnou adresu IP / MAC, než je adresa koncového zařízení, poškozené packety, packety se špatným CRC…). Program Netcat však tyto náležitosti dodržuje a je tedy nutné nastavit správný port, na kterém bude aplikace poslouchat a nastavit statickou IP adresu na mém koncovém zařízení (taktéž již vyřešený problém s proměnou kontrolní hodnotou čtvrté vrstvy). Odesílaný packet je modifikací testovacího packetu, kde byla IP adresa příjemce stanovena routerem mé domácí sítě LAN (protokolem DHCP). Po odpojení ethernetového kabelu z mé domácí sítě tuto adresu ztrácím a je tedy nutností manuálně nastavit statickou adresu IP.

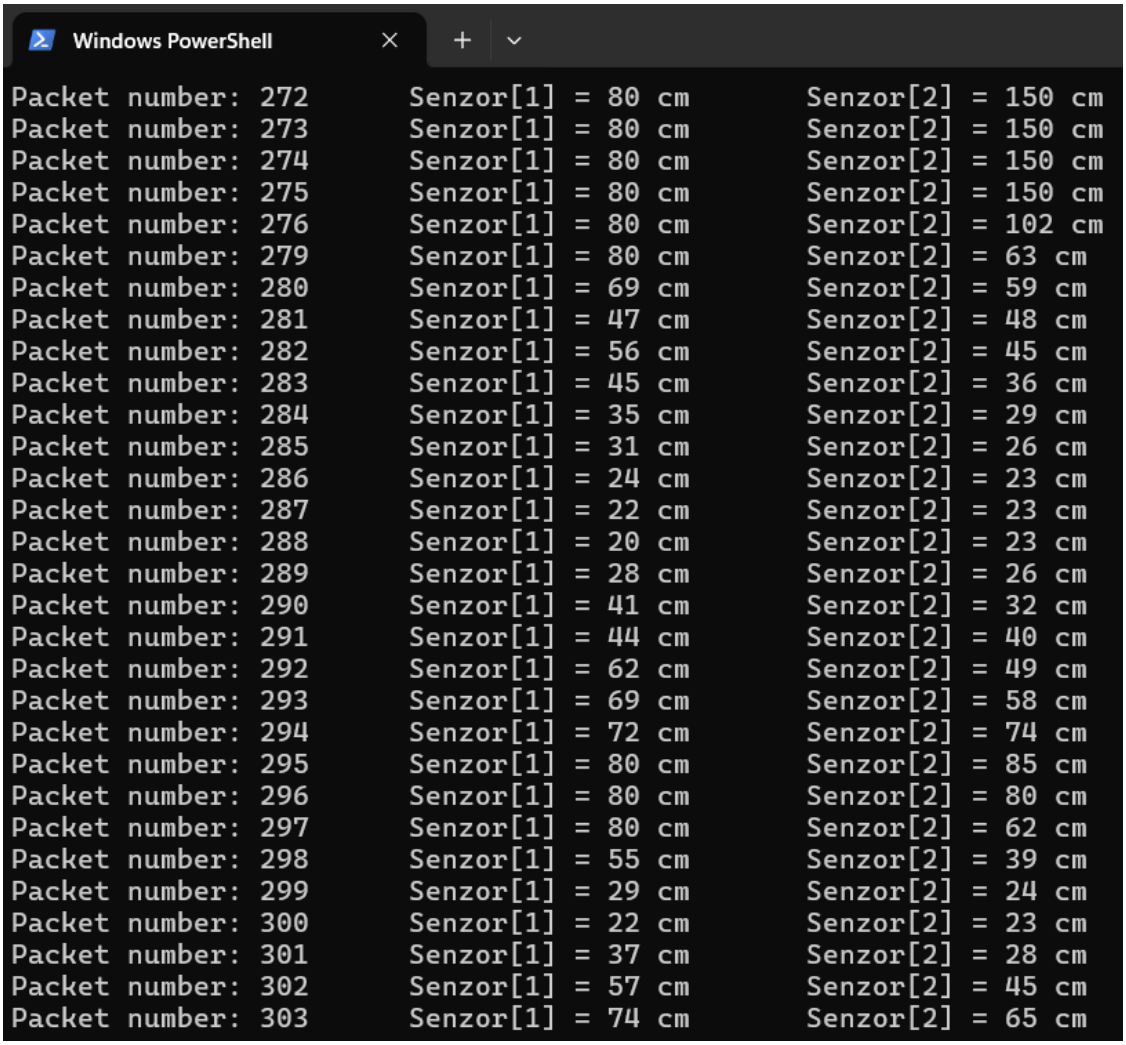
V nastavení operačního systému jsem u ethernetu zvolil manuální přiřazení adresy IP. Vybraná je verze IPv4, adresu IP volím podle testovacího packetu na hodnotu *192.168.1.50* a Subnet masku na hodnotu *255.255.0.0*.

Po nastavení statické adresy IP je vytvořena aplikace Netcat v konzoli Windows Powershell. Pro naslouchání UDP packetů je zadán následující skript:

ncat -l -u -p 1234

Skript nese příznaky ***-l*** pro naslouchání, ***-u*** specifikuje packety UDP a ***-p*** specifikuje port, na kterém je naslouchán**o**.

Posledním krokem je u zapojení z obrázku 6.2 nahrát do modulu firmware ve verzi release (pokud nebyl již nahrán předem). Nad snímači vzdálenosti jsem poté pohyboval bílým papírem tak, aby snímače zaznamenávali různé vzdálenosti. Zachycené packety aplikace jsou znázorněny na obrázku 6.3.



Obrázek 6.3 Aplikace ethernetu v programu Netcat

7. ZÁVĚR

V této bakalářské práci jsem se zaměřil na vývoj ethernetového modulu. Pro modul jsem navrhnul desku plošných spojů (DPS) na bázi mikropočítače Raspberry Pi Pico, kterým ovládám čip fyzické vrstvy KSZ8081MLX. Po oživení DPS jsem pro modul vytvořil firmware, který realizuje ethernetovou komunikaci rozhraním MII. Nakonec je vytvořena aplikace, která modul využívá k interpretaci dat ze snímačů vzdálenosti a jejich odeslání do koncového zařízení.

S mikropočítačem Raspberry Pi Pico mám zkušenosti z jiných projektů, v první fázi jsem si tedy pouze trochu více připomněl periférii PIO a rozmyslel, které ostatní periferie kontroléru RP2040 použiji. Následně jsem se seznámil s principy ethernetové komunikace. Jednalo se nejprve o seznámení se s celým modelem ISO/OSI, použité protokoly, stavba samotného ethernetového packetu. Nakonec jsem se seznámil s čipem fyzické vrstvy KSZ8081MLX, který realizuje rozhraní MII. Tento postup zahrnují sekce 1 a 2.

Velkou část práce tvoří návrh desky plošných spojů, její návrh mi zabral přibližně 3 měsíce. Navrhování DPS pro komunikační modul obnáší dost detailů, které je třeba dodržet pro funkčnost celého modulu. Při návrhu DPS jsem získal velké množství zkušeností a různých dodatečných informací. Detailněji je návrh popsán v sekci 3.

Pro modul jsem poté vybral jednotlivé elektronické součástky v souladu se schématem a zapájel je na prototyp DPS. Funkčnost byla následně ověřena jednoduchým programem.

Následně jsem se zabýval vytvořením firmwaru, zahrnující obsluhu ethernetu, vytvoření ethernet packetu, kód PIO asembleru, obsluhu snímačů a dalších detailů. Po vytvoření funkčního firmwaru jsem se určitý čas zabýval jeho optimalizací. Tímto se zabývá sekce 4 a 5.

Sekce 6 poté ukazuje aplikaci bakalářské práce v programu Netcat. Program přijímá packety, obsahující data ze snímačů a zobrazuje tyto data na konzoli.

Bakalářská práce byla pro mě velice přínosná ať už z hlediska návrhu, tak z hlediska teorie sítí a síťové komunikace. Za velký úspěch považuji zdařilé oživení prvotního návrhu DPS. Při další iteraci bych se zaměřil na zmenšení rozměrů DPS, přidání filtračních kondenzátorů zejména u step-down měniče (nebo zvolením kvalitnějšího step-down měniče) a změně pořadí signálu TX. V aktuálním stavu neřeším příjem dat. PIO program pro RX je vytvořen, chybí však implementace funkce, obsluhující přijaté packety.

LITERATURA

[1] RP2040 Datasheet [online]. 2023-06-14. 2023 [cit. 2023-12-23]. Dostupné z: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

[2] *John P. CLARK, Martin. Data Networks, IP and the Internet. Online. 2003. The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ*, England, 2003. ISBN ISBN: 0-470-84856-1. [cit. 2023-12-23].

[3] *HELD, Gilbert. Data Communications Networking Devies: Operation, Utilization and LAN and WAN Internetworking. Online. 4.* Baffins Lane, Chichester, West Sussex PO191UD, England, 1999. ISBN 0-470-84182-6. [cit. 2023-12-23].

[4] *Online. Ethernet Protocol — Computer Networking 0.5 documentation.* Dostupné z: https://cot-cn.cougarnet.uh.edu/docs/compnet/012-ethernet.html#. [cit. 2023-12-23].

[5] *Online. Cheapsslsecurity.* Dostupné z: https://cheapsslsecurity.com/blog/how-udp-works-a-look-at-the-user-datagram-protocol-in-computer-networks/. [cit. 2023-12-23].

[6] *devfish. Online. The backbone of internet communication: TCP/IP & UDP. 2023.* Dostupné z: https://velog.io/@devfish/Network-UDP-TCPIP-HTTP. [cit. 2023-12-23].

[7] *Online. Javatpoint Computer Network.* Dostupné z: https://www.javatpoint.com/ethernet-frame-format. [cit. 2023-12-23].

[8] *Online. ISO/OSI Model and it's Layers - Physical to Application | Studytonight.* Dostupné z: https://www.studytonight.com/computer-networks/complete-osi-model. [cit. 2023-12-23].

[9] *Online. Microcontroller Peripherals.* Dostupné z: https://www.labcenter.com/blog/sim-microcontroller-peripherals/. [cit. 2023-12-23].

[10] *JONES, Mike. Interfacing Fast Ethernet to Processors.* Online. S. 8. Dostupné z: https://www.microchip.com/content/dam/mchp/documents/OTH/ApplicationNotes/ApplicationNotes/FastEtherProcess-WP.pdf. [cit. 2023-12-28].

[11] *KSZ8081MLX 10BASE-T/100BASE-TX Physical Layer Transciever [online]. 2016, 56 [cit. 2023-12-28]*. Dostupné z: https://ww1.microchip.com/downloads/en/DeviceDoc/10BASE-T-100BASE-TX-Physical-Layer-Transceiver-DS00002264B.pdf

[12] *"HOODAAJAY99" HOODA, Ajay. MAC-PHY-and-MII-Interface. Online. Roč. 2019.* Dostupné z: https://github.com/hoodaajay99/linux-kernel-device-drivers/blob/master/docs/36-Network-Drivers/04-MAC-PHY-and-MII-Interface/MAC-PHY-and-MII-Interface.md. [cit. 2023-12-28].

[13] *MICROSEMI. Application Note Magnetics Guide. Online. 2018, s. 12*. Dostupné z: https://ww1.microchip.com/downloads/en/Appnotes/VPPD-01740.pdf. [cit. 2023-12-29].

[14] *FEHEEMUDDIN, Mohamed. What is Signal Propagation Delay in PCBs? Online. 2023.* Dostupné z: https://www.protoexpress.com/blog/signal-propagation-delay-pcb/. [cit. 2023-12-30].
[15] *MISTRY, Sandeep. Pico-rmii-ethernet. Online. 2021, s. 1.* Dostupné z: https://github.com/sandeepmistry/pico-rmii-ethernet. [cit. 2024-03-15].
[16] *ZIMA, Jan. MII-Ethernet-on-Raspberry-Pi-Pico. Online. 2024, s. 1*. Dostupné z: https://github.com/Lomqe/MII-Ethernet-on-Raspberry-Pi-Pico. [cit. 2024-03-15].
[17] *Cortex-M0+ Technical Reference Manual r0p1. Online. 2013.* Dostupné z: https://developer.arm.com/documentation/ddi0484/c/Programmers-Model/Instruction-set-summary. [cit. 2024-03-26].
[18] *RP2040 PIO Emulator. Online.* Dostupné z: https://rp2040pio-docs.readthedocs.io/en/latest/introduction.html. [cit. 2024-04-23].
[19] *Getting started with Raspberry Pi Pico. Online.* 2020. Dostupné z: https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf#swd-debug. [cit. 2024-04-25].
[20] *GP2D12 Data Sheet. Online*. Dostupné z: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/1/GP2D12.pdf . [cit. 2024-04-26].
[21] *Gp2y0a02yk_e. Online.* Dostupné z: https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a02yk_e.pdf. [cit. 2024-04-26].

SEZNAM SYMBOLŮ A ZKRATEK

Zkratky:

PIO	Programmable Input Output
PHY	Physical Layer
RPP	Raspberry Pi Pico
GPIO	General Programmable Input Output
UTP	Unshielded Twisted Pairs
PC	Personal Computer / Program Counter
SNI	Server Name Indication
PLC	Programmable Logic Controller
I2C	Inter-Integrated Circuit
UART	Universal Asynchronous Receiver-Transmitter
DVI	Digital Visual Interface
USB	Universal Serial Bus
LAN	Local Area Network
WAN	Wide Area Network
MII	Media Independent Interface
MIIM	Media Independent Interface Management
RMII	Reduced Media Independent Interface
PCB	Printed Circuit Board
DPS	Deska Plošných Spojů
GPIO	General Purpose Input Output
ISR	Input Shift Register
OSR	Output Shift Register
ADC	Analog to Digital Converter
DMA	Direct Memory Access
FIFO	First in First out
Ksps	Kilosamples per second
Mbps	Megabits per second
RAM	Random Access Memory
ISO	Internal Organization for Standardization
OSI	Open Systems Interconnection
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
HTTP	Hypertext Transfer Protocol
NFS	Network File System

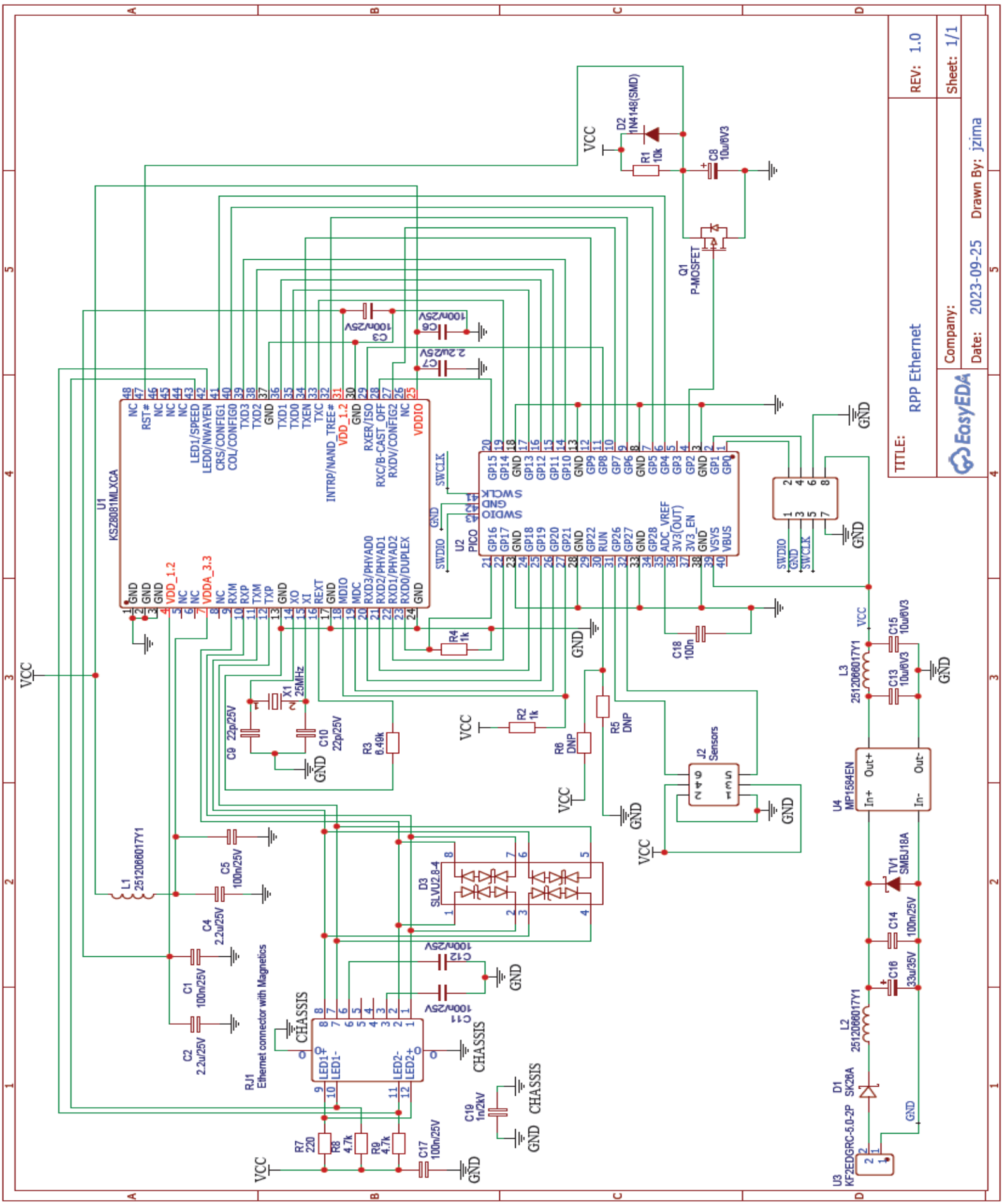
IEEE	Institute of Electrical and Electronics Engineers
MAC	Media Access Control
LLC	Logical Link Control
FSD	Frame Sequence Delimiter
FCS	Frame Checksum Sequence
CRC	Cyclic Redundancy Check
OUI	Organization Unique Identifier
NIC	Network Interface Controller
TTL	Time to Live
ISP	Internet Service Provider
LLDP	Link Layer Discovery Protocol
ARP	Address Resolution Protocol
DHCP	Dynamic Host Configuration Protocol
EMC	Electromagnetic Compatibility

Symboly:

t_{dp}	zpoždění signálu	(ps/in)
t_{max}	maximální zpoždění signálu	(ns)
l_{max}	maximální délka trasy DPS	(cm)
l	délka trasy	(cm)
E_r	dielektrická konstanta substrátu DPS	(-)
E_{reff}	efektivní dielektrická konstanta substrátu DPS	(-)

SEZNAM PŘÍLOH

Příloha A - Schéma propojení Raspberry Pi Pico a KSZ8081LMX



Příloha B - Struktura zdrojového kódu