ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA
STROJNÍ

U12105
Ústav mechaniky, biomechaniky a mechatroniky

Adaptace existující řídící sběrnice CAN robota PUMA200 na sběrnici Ethercat

Adaptation of the existing CAN control bus of the PUMA200 robot to Ethercat

BAKALÁŘSKÁ PRÁCE

Vypracoval: Michal Kolařík

Vedoucí práce: Ing. Martin Nečas, MSc., Ph.D.

Rok: 2022

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kolařík** Jméno: **Michal** Osobní číslo: **487976**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Adaptace existující řídicí sběrnice CAN robota PUMA200 na sběrnici Ethercat

Název bakalářské práce anglicky:

Adaptation of the existing CAN control bus of the PUMA200 robot to Ethercat.

Pokyny pro vypracování:

1. Proveďte rešerši na téma komunikačních protokolů s ohledem na realtimové řízení
2. Adaptujte existující řízení robota Puma200 z komunikační sběrnice CAN na Ethercat s taktem řízení polohy 1ms
3. Vytvořte jednoduchý ovládací HMI panel pro robota
4. Kriticky zhodnoťte dosažené výsledky

Seznam doporučené literatury:

[1] Gerardus, B., EtherCAT A Complete Guide - 2019 Edition, 5STARCooks
[2] Firemní literatura: Texas Instruments pro procesory TMS320F28379D, drv8711, sprui77a
[3] Firemní literatura: Vývojový kit Teensy 4.0, www.pjrc.com

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Nečas, MSc., Ph.D. odbor mechaniky a mechatroniky FS

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **22.04.2022** Termín odevzdání bakalářské práce: **15.08.2022**

Platnost zadání bakalářské práce: _____

Ing. Martin Nečas, MSc., Ph.D. podpis vedoucí(ho) práce	doc. Ing. Miroslav Španiel, CSc. podpis vedoucí(ho) ústavu/katedry	doc. Ing. Miroslav Španiel, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

4.5. 2022
Datum převzetí zadání

Podpis studenta

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně pod vedením Ing. Martina Nečase, MSc., Ph.D., s použitím literatury uvedené na konci bakalářské práce v seznamu použité literatury.

V Praze dne: ....

Michal Kolařík

Poděkování

Rád bych tímto poděkoval vedoucímu své bakalářské práce Ing. Martinu Nečasovi, MSc., Ph.D., za jeho cenné rady, lidský přístup a ochotu mi věnovat čas i do pozdních večerních hodin. Dále bych chtěl poděkovat mým rodičům a přítelkyni za velkou podporu během studia.

Michal Kolařík

Anotační list

Název práce:	Adaptace existující řídící sběrnice CAN robota PUMA200 na sběrnici Ethercat
Autor:	Michal Kolařík
Ústav:	U12105 – Ústav mechaniky, biomechaniky a mechatroniky
Druh Práce:	Bakalářská práce
Vedoucí práce:	Ing. Martin Nečas, MSc., Ph.D.
Klíčová slova:	Aplikační sběrnice, Real-time řízení, Přenos dat, Souřadnicový systém, Modul, Master, Slave
Title:	Adaptation of the existing CAN control bus of the PUMA200 robot to Ethercat
Author:	Michal Kolařík
Keywords:	Fieldbus, Real-time control, Data transmission, Coordinate system, Module, Master, Slave
Počet stran:	85
Počet obrázků:	98
Počet tabulek:	0

Anotace: Tato bakalářská práce se zabývá adaptací sběrnice Ethercat do existujícího řídícího systému robota PUMA200 se sběrnicí CAN. Teoretická část práce se zabývá komunikačními protokoly používanými v real-time řízení. Praktická část se zabývá spojením hardwarových prvků tak, aby bylo možné robota ovládat pomocí dotykového displeje. Data z displeje mají být zpracovány a poslány do robota real-time za pomocí desky Teensy, softwaru TwinCAT a simulačního modelu vytvořeného v programu MATLAB/Simulink.

Abstract: This bachelor thesis deals with adaptation of Ethercat bus to existing control system of the PUMA200 robot with CAN bus. The theoretical part delas with the communication protocols used in real-time control. The practical part deals with connecting hardware components so that the robot can be controlled with the touch screen. Data from the touch screen are supposed to be processed and sent to the robot real-time with the help of the Teensy board, TwinCAT software and simulation model made in MATLAB/Simulink program.

OBSAH

1 Úvod 7

2 Motivace a cíle 8

3 Rešerše 9

3.1 Model ISO/OSI 9

3.2 Rozdělení komunikačních protokolů 10

3.3 Konvenční Fieldbusy 11

3.3.1 CAN 11

3.3.2 PROFIBUS DP 13

3.3.3 CC-link 15

3.3.4 ControlNet 16

3.4 Průmyslový Ethernet 19

3.4.1 Úvod do Ethernetu 19

3.4.2 Potřeba průmyslového Ethernetu 20

3.4.3 Ethernet/IP 20

3.4.4 Ethernet Powerlink 21

3.4.5 PROFINET IO 24

3.4.6 SERCOS III 26

3.4.7 EtherCAT 28

4 Implementace sběrnice EtherCAT 34

4.1 Použitý hardware 34

4.1.1 Teensy 4.0 34

4.1.2 Modul EasyCAT PRO 35

4.1.3 Dotykový displej 36

4.1.4 TI Delfino Launchpad 37

4.1.5 Výkonový modul 38

4.2 Systémový návrh 39

4.2.1 Konfigurace modulu EasyCAT PRO 40

4.2.2 Struktura dotykového displeje 41

4.2.3 Řízení pomocí Teensy 4.0 45

4.2.4 Řízení v prostředí MATLAB/Simulink 55

4.2.5 TwinCAT 3 65

5 Závěr 77
6 Použitá literatura 78
7 Seznam obrázků 82
8 Přílohy práce 85

1 Úvod

S neustále se zrychlujícím technickým vývojem a zvyšujícími se nároky na cenu se čím dál více automatizačních elementů (robotů, automatů, PLC) stává nedílnou součástí výrobních procesů. Zařízení automatizace se stávají sofistikovanější a kladou se na ně každým rokem větší nároky, apeluje se u nich především na schopnost řešit složitější úlohy za účelem zvýšení kvality a produktivity práce nebo na rychlost a bezpečnost během úkonů.

Aby bylo možné komponenty automatizace řídit, je důležité k nim přenášet data o tom, co mají vykonávat. *Data* jsou klíčovým prvkem v moderní průmyslové automatizaci a jedna z výzev obnáší je zpracovat a distribuovat real-time. Za tímto účelem se během uplynulých desítek let vyvinulo spousta komunikačních protokolů. Pojem komunikační protokol obecně představuje formát přenášených dat a soubor pravidel, na kterých se musí zařízení v komunikační síti předem domluvit, aby byla schopna si mezi sebou správně vyměňovat data. *Real-time* v tomto kontextu znamená, že daný systém musí splnit úkol nebo skupinu úkolů nejpozději v námi určený čas. Tento čas se různí od aplikace k aplikaci. Bavíme-li se například o řízení pohybu robota, požadujeme přenos dat a jeho reakci minimálně v řádu milisekund.

Pří návrhu řídícího systému je tedy volba a implementace správného komunikačního protokolu kruciální a je tedy žádoucí si osvojit znalosti v tomto odvětví automatizace.

2 Motivace a cíle

Motivací této bakalářské práce je rozšířit znalosti užívaných komunikačních protokolů (aplikačních sběrnic) s ohledem na řízení v reálném čase. Cílem této práce je aplikovat tyto znalosti a vylepšit stávající řídící systém robota PUMA200, navrženého Ing. Michaelem Valáškem při tvorbě jeho diplomové práce. Důvod k přechodu ze sběrnice CAN na sběrnici EtherCAT tkví především v problémech při posílání dat real-time, se kterými se robot potýkal. Při implementaci CAN sběrnice byla data příliš zpožděna či dokonce docházelo k chybám při přenosu, proto byl implementován na straně vysílače (PC) buffer, zajišťující akumulaci dat a jejich následné poslání do řídící smyčky, čímž byla zajištěna plynulost pohybu robota. Tento problém by měla eliminovat implementace sběrnice EtherCAT. Dalším cílem je návrh HMI (*Human Machine Interface*) v podobě řízení robota pomocí dotykového displeje pro výukové účely na Fakultě strojní ČVUT v Praze.

Samotný text práce je rozdělen do tří částí:

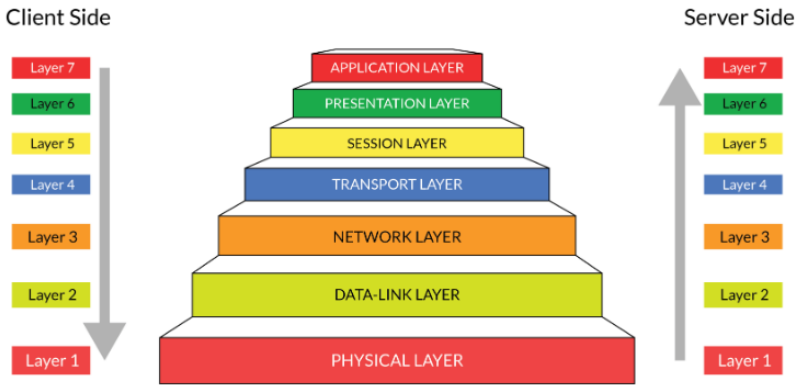
- Rešerše sběrnic v automatizaci, kde bude podrobněji popsána sběrnice EtherCAT, která bude použita v případě této práce
- Realizace implementace sběrnice EtherCAT. Tato část bude mít dvě podčásti:
    - Použitý hardware
    - Systémový návrh a popis jednotlivých bloků systémového návrhu
- Kritické zhodnocení výsledků

3 Rešerše

3.1 Model ISO/OSI

Při popisu jednotlivých komunikačních protokolů bude často odkazováno na vrstvy referenčního modelu ISO/OSI, a proto považuji za vhodné se o tomto modelu krátce zmínit.

Open Systems Interconnect (OSI) *Reference Model*, vyvinutý v roce 1983 mezinárodní organizací ISO[1] je hierarchický model používaný k popisu struktury a funkce komunikačních protokolů. Tento model definuje sedm vrstev, z nichž každá plní přesně vymezené funkce a zároveň se nestará o funkčnost a implementaci vrstev nižších. Komunikace spočívá v tzv. zapouzdřování na straně odesílatele, kdy vyšší vrstva vloží PDU[2] do protokolu nižší vrstvy. Na straně příjemce jsou naopak data jednotlivými vrstvami rozbalovány směrem k vyšším vrstvám. Struktura modelu ISO/OSI je vidět na obr. 1.



Obrázek 1 - Referenční model ISO/OSI [1]

Níže jsou popsány funkce jednotlivých vrstev modelu ISO/OSI (od nejnižší vrstvy k nejvyšší):

- Fyzická vrstva – jejím úkolem je zajistit přenos jednotlivých bitů mezi příjemcem a odesílatelem prostřednictvím fyzické přenosové cesty. Fyzická vrstva řeší otázky technického charakteru (úroveň napětí, přenosové médium, tvar konektorů)
- Linková vrstva – má za úkol pomocí služeb fyzické vrstvy zajistit bezchybný přenos celých bloků dat (řádově stovek bytů), označovány jako rámce. Linková vrstva má za úkol také rozpoznat začátek a konec rámce i jeho jednotlivé části. Jelikož se fyzická vrstva nezajímá o interpretaci jednotlivých bitů, tak jednotlivé chyby v přenosu zjistí až linková vrstva
- Síťová vrstva – zajišťuje směrování (*routing*) přenášených rámců, označovaných jako pakety, pokud spojení vede přes jeden či více mezilehlých uzlů. Síťová vrstva tedy zajišťuje volbu vhodné trasy (*route*)

[1] ISO – International Standard Organization
[2] PDU – Protocol Data Unit, procesní jednotka

přes mezilehlé uzly a také postupné předávání jednotlivých paketů po této trase od původního odesilatele až ke konečnému příjemci

- Transportní vrstva – zabývá se komunikací koncových účastníků (tzv. end-to-end komunikací), tedy komunikací mezi původním odesilatelem a příjemcem. Při odesílání dat zajišťuje transportní vrstva rozdělení zprávy do jednotlivých paketů a při příjmu zase zprávu skládá z jednotlivých paketů dohromady
- Relační vrstva – jejím úkolem je navázat, udržovat a spravovat spojení mezi koncovými účastníky, poskytuje služby jako je autorizace
- Prezentační vrstva – má na starosti potřebné konverze přenášených dat, jelikož jednotlivé uzlové počítače mohou používat odlišnou vnitřní reprezentaci těchto dat. Například konverze kódování znaků ASCII na kódování EBCDIC apod. Úkolem této vrstvy může být například i šifrování dat
- Aplikační vrstva – poskytuje služby přímo jednotlivým aplikacím. Aplikační vrstva představuje okno, prostřednictvím kterého mohou uživatelé vidět výsledky služeb zajišťovaných všemi předcházejícími vrstvami. Příkladem funkce zajišťovaných touto vrstvou je elektronická pošta, prohlížení webových stránek, souborové přenosy apod. [2]

3.2 Rozdělení komunikačních protokolů

Přísné požadavky průmyslových sítí historicky vedly k vytvoření široké škály aplikačně specifických protokolů. V literatuře se namísto komunikačního protokolu v průmyslové automatizaci můžeme setkat s názvem průmyslová či aplikační sběrnice, nebo také Fieldbus. Drátové a bezdrátové sítě mají každá své vlastní sady sběrnic. Navzdory velkému množství sběrnic, které jsou k dispozici, většina drátových sítí dnes využívá dva typy [3]:

- Konvenční Fieldbusy
- Fieldbusy založené na Ethernetu (průmyslový Ethernet)

V další části budou popsány vybrané protokoly používané především při deterministickém vysokorychlostním (real-time) řízení. Bezdrátové sítě a jejich protokoly budou v této práci vynechány.

3.3 Konvenční Fieldbusy

3.3.1 CAN

Controller Area Network (CAN) je průmyslová sběrnice vyvinutá Německou firmou BOSCH v roce 1986 z důvodu zvýšené poptávky na řízení elektroniky v automobilovém průmyslu. CAN je sériová multi-master[3] komunikační sběrnice standardizovaná organizací ISO. Rychlost CAN sběrnice může dosáhnout až 1 Mbps. CAN protokol z referenčního modelu ISO/OSI používá fyzickou a linkovou vrstvu. Pro fyzickou vrstvu platí standard ISO 11898-2 a pro linkovou vrstvu standard ISO 11898-1. Aby mohl být využit plný potenciál této sběrnice, je CAN bus součástí protokolů vyšších vrstev, kde je možné navíc používat funkce jako správa sítě, diagnóza sítě, přístup k datům apod. Příkladem protokolů vyšších vrstev založených na CAN bus jsou DeviceNet, CanKingdom, CANopen (CANopen FD) a další. [4] [5]

Sběrnici CAN tvoří dva vodiče označované *CAN_H* a *CAN_L*, nejčastěji ve formě kroucené dvojlinky. CAN definuje dvě hodnoty bitů na sběrnici dané rozdílem napětí těchto dvou vodičů:

- *Dominant*
- *Recessive*

Dominant reprezentuje logickou „1“ a *recessive* logickou „0“. CAN pro komunikaci používá čtyř typů zpráv:

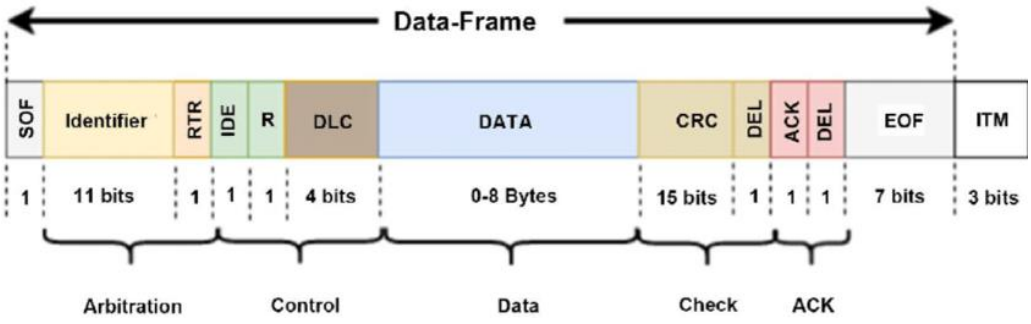
- datová zpráva
- žádost o data
- informace o chybě
- informace o přetížení

Datová zpráva a žádost o data mohou být ve dvou formátech. První je tzv. standardní formát (délka identifikátoru 11 bitů) a druhý tzv. rozšířený formát (délka identifikátoru 29 bitů). Nejdůležitější typ zprávy je datová zpráva (obr. 2), ta se skládá z následujících částí:

- *Start Of Frame* (SOF) – začátek rámce, musí být *dominant* (1 bit)
- *Arbitration Field* – řízení přístupu ke sběrnici a identifikátor zprávy velikost (11 bitů)
- *Remote Request* (RTR)– udává, zda jde o datovou zprávu nebo žádost o přístup ke sběrnici (1 bit)
- *Control Field* – řídící pole, musí být *dominant* (2 bity)
- *Data length* – délka datové zprávy (4 bity)
- *Data Field* – datová oblast (0-8 bytů)
- *Cyclic Redundant Check* (CRC) – kontrolní součet (15 bitů)
- *CRC Delimiter* (DEL) – oddělovač, musí být *recessive* (1 bit)
- *Acknowledgement* (ACK) – potvrzení (2 bity), z toho 1 bit (ACK) a 1 bit oddělovač (DEL)

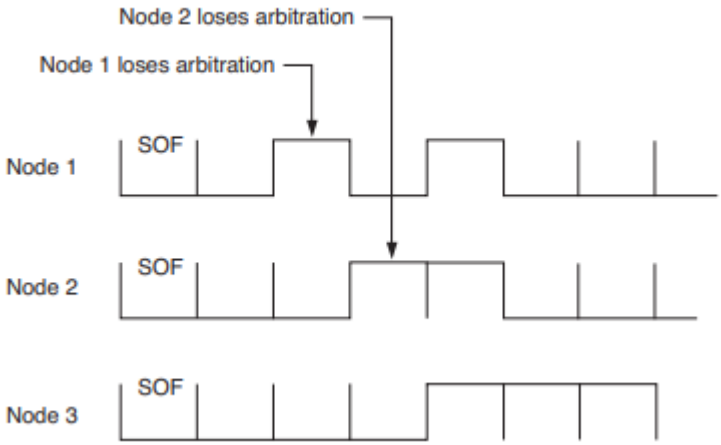
[3] Multi-master – možnost mít na sběrnici více řídících prvků

- *End Of Frame* (EOF) – konec zprávy, musí být *recessive* (7 bitů),
- *Interframe Space* (ITM) – mezera mezi zprávami, musí být *recessive* (3 bity)



Obrázek 2 - Datový rámec sběrnice CAN [6]

Jelikož se jedná o sběrnici typu multi-master, může každé připojené zařízení zahájit vysílání. Zprávu vysílá ten, kdo zahájí vysílání první. Pokud zahájí vysílání více zařízení současně, přístup ke sběrnici získá zařízení s větší prioritou (nižším identifikátorem). Každé zařízení porovnává vysílaný bit s hodnotou na sběrnici a zjistí-li, že se hodnota liší (jedinou možností je *dominant* bit na sběrnici a *recessive* bit vysílaný zařízením), okamžitě přeruší vysílání. Přerušení nazývá *arbitration loss*[4] (obr. 3). Zařízení, které nezískalo přístup ke sběrnici musí vyčkat, než bude sběrnice opět volná (*Bus Free*). [4] [5]



Obrázek 3 - Princip arbitrace sběrnice CAN [5]

3.3.1.1 CAN FD

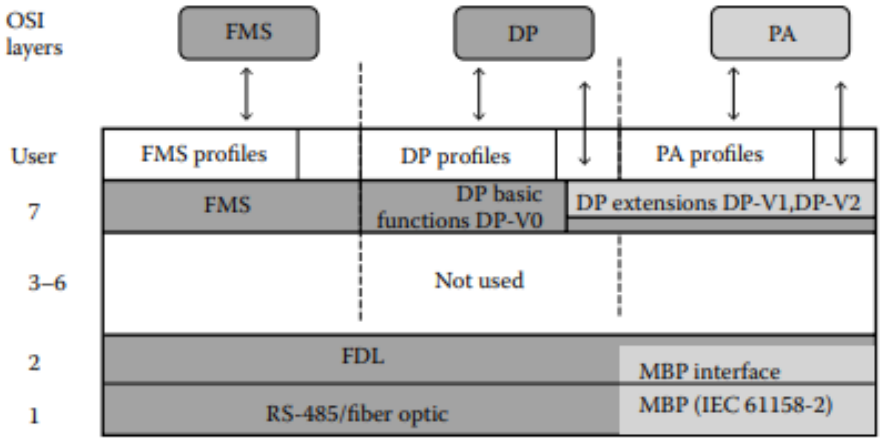
BOSCH v roce 2012 vytvořil novou verzi sběrnice CAN ve formě CAN FD. Hlavní rozdíl CAN FD oproti klasické CAN sběrnici je schopnost přenosu dat délky až 64 bytů s rychlostí až 8 Mbps. [7]

[4] Arbitration loss – ztráta arbitrace

3.3.2 PROFIBUS DP

PROcess FIeldBUS – Decentralised Periphery (PROFIBUS DP) je jedno z řešení (mimo PROFIBUS FMS a PROFIBUS PA) prověřeného standardu PROFIBUS pro otevřené sítě typu Fieldbus, vyvinutý v roce 1989 Německou vládou ve spolupráci s několika firmami. PROFIBUS DP se uplatňuje při potřebě vysokorychlostního řízení. Využívá se ve většině zařízení v průmyslové automatizaci, tj. ve výrobních linkách, frekvenčních měničích, a především při ovládání I/O[5] modulů vzdálených od centrální řídící stanice (proto také název *Decentralised Periphery* – decentralizovaná od ostatních periferií).

Z referenčního modelu ISO/OSI používá PROFIBUS DP tři vrstvy – fyzickou, linkovou a aplikační. Fyzická vrstva staví (ve většině případů) na standardu RS-485, kde je jako přenosové médium použita stíněná kroucená dvojlinka. V jiném případě se pro přenos aplikuje technologie optického vlákna. U obou případů lze připojit až 126 zařízení (za pomocí opakovače) k jedné sběrnici a oba tyto standardy jsou schopny přenosu dat s rychlostí až 12 Mbps. Samotná pravidla přenosu, ať už je to přístup ke sběrnici, detekce chyb apod., je úkolem linkové vrstvy, kdy PROFIBUS DP využívá standardu IEEE 802.4. Aplikační vrstva funguje jako rozhraní mezi aplikačními programy a tzv. DP profily, které jsou definovány nad aplikační vrstvou (obr. 4). Profily definují běžné funkce, které má každé zařízení v síti PROFIBUS DP. Tyto funkce se liší dle použité verze sběrnice PROFIBUS DP (DPV0 a následníků DPV1 a DPV2). [8]



Obrázek 4 - ISO/OSI model sběrnice PROFIBUS [8]

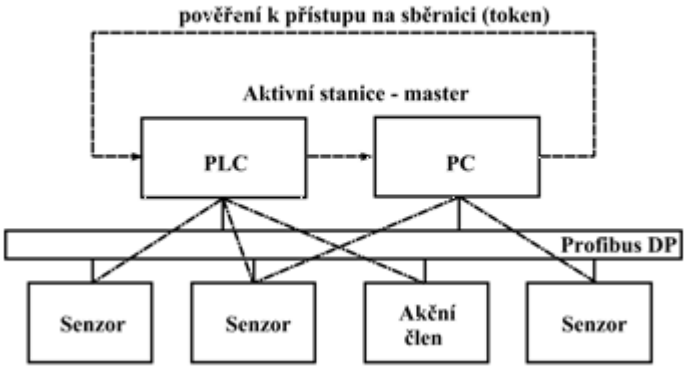
[5] I/O – Input/Output

3.3.2.1 Typy zařízení v síti PROFIBUS DP

- *PROFIBUS DP Master Class 1* (DPM1) – master třídy 1, obvykle používán pro výměnu cyklických dat s připojenými slave[6] zařízeními. Obvykle jsou to buď PC nebo PLC[7]
- *PROFIBS DP Master Class 2* (DPM2) – master třídy 2, je používán pro zasílání acyklických dat pro realizaci diagnostiky sítě a pro počáteční konfiguraci zařízení ve verzích DPV1 a DPV2. DPM2 nemusí být připojeny do sítě permanentně ale lze je použít pouze pro počáteční konfiguraci.
- *DP Slave* – zařízení, které přijímá příkazy od mastera. Ve verzi DPV2 je možná přímá komunikace mezi slave zařízeními.

3.3.2.2 Způsob komunikace

Master komunikuje se slave zařízeními pomocí *polling*[8] metody, tedy přenášení dat se uskutečňuje cyklicky s danou frekvencí. Pokud master v určitém cyklu pošle datagram s žádostí o data, slave zařízení na něj odpoví. K jedné sběrnici může být připojeno i více řídících zařízení (masterů). Pokud síť obsahuje víc než jednoho mastera (obr. 5), řeší PROFIBUS DP přístup ke sběrnici pomocí metody předávání tzv. tokenu, kdy přístup ke sběrnici dostane ten master, který má v držení tento token. Token si mezi sebou master zařízení předávají. Čas, po který může daný master token držet je předem definován dle požadavků. [8]



Obrázek 5 - Struktura sítě PROFIBUS DP s metodou přístupu pomocí tokenu [9]

Pro přenos dat jsou v linkové vrstvě implementovány dvě komunikační služby – *Send and Request Data with acknowledge* (SRD) a *Send Data with No acknowledge* (SDN). U služby SRD lze přenášet data ke vzdálené stanici a současně data od této stanice přijímat. Používá se pro cyklickou výměnu dat mezi masterem a slavem. SDN je služba používaná pro přenos zpráv typu *broadcast*[9] nebo *multicast*[10]. [10]

[6] Slave – „otrok", neboli zařízení, které je řízeno masterem
[7] *Programmable Logic Controller* (PLC) – programovatelný logický automat
[8] Polling – neustálé dotazování
[9] Broadcast – zaslání zprávy všem zařízením v síti
[10] Multicast – zaslání zprávy určité skupině zařízení v síti

3.3.3 CC-link

Control & Communication link (CC-link) je velmi rychlá komunikační sběrnice, vyvinutá firmou Mitsubishi Electric Corporation, používaná především v Asii. Později se tato sběrnice stala jednou z rodiny protokolů CC-link, které v současné chvíli spravuje *CC-link Partner Association* (CLPA). Kromě originálního CC-link (standard ISO 15745-5) probíraném v této části se můžeme setkat se standardy CC-link Safety, CC-link/LT a protokoly CC-link založené na Ethernetu – CC-link IE (*Control/Field/Field Basic/Field Motion*). Mezi výhody CC-link patří osvědčený výkon, poměrně levná realizace sítě a systémová kompatibilita tisíce produktů mnoha výrobců, kteří jsou členy CLPA. CC-link sběrnice se uplatňuje v mnoha oblastech automatizace, jelikož, jak už bylo řečeno, umožňuje kompatibilitu širokého spektra výrobků. Nalezneme ho například v průmyslových PC, PLC, ventilech, digitálních a analogových I/O zařízení, servopohonech, regulátorech teploty a mnoho dalších. [11]

Pro fyzickou vrstvu aplikuje CC-link standard RS-485, jako přenosové médium používá třížílový stíněný kabel. Rychlost přenášených dat může dosáhnout až 10 Mbps. Na sběrnici může být připojeno až 64 zařízení.

Sběrnice CC-link definuje dva typy datové komunikace:

- Cyklický přenos
- Dočasný přenos.

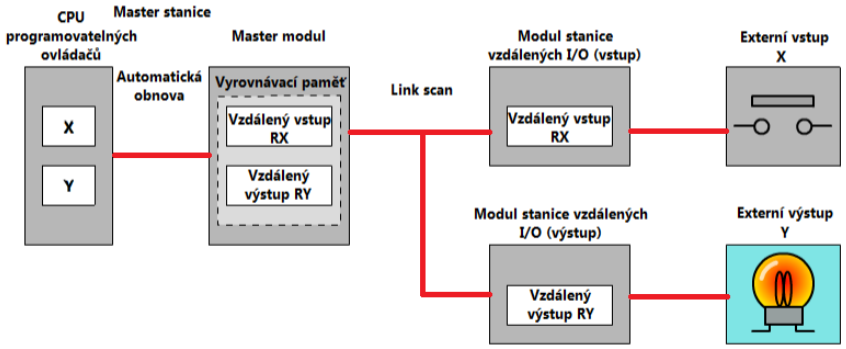
Cyklický přenos cyklicky a automaticky odesílá a přijímá data ve stejném intervalu. Data posílaná v cyklickém přenosu se dělí na bitová data, která zahrnují informaci ON/OFF (ZAPNUTO/VYPNUTO) a *word* data[11], zahrnující analogové informace. Dočasný přenos odesílá a přijímá data mezi cyklickými přenosy, pouze pokud existuje požadavek na komunikaci mezi jednotlivými PLC v síti. Příslušný typ přenosu se aplikuje v závislosti na tom, se kterou stanicí probíhá komunikace. CC-link definuje 4 druhy stanic [12] :

- Master stanice – stanice, která řídí celou síť, v síti musí být alespoň jedna master stanice, příkladem může být např. PLC nebo PC.
- Lokální stanice – provádí dočasné přenosy s master stanicí nebo jinou lokální stanicí a cyklický přenos bitových dat a *word* dat s ostatními stanicemi. Příkladem může být například PLC, PC nebo HMI.
- Vzdálená zařízení – stanice, která používá pouze cyklické přenosy s obou druhů (bitové i *word*). Příkladem jsou analogové I/O zařízení, servopohony, indikátory apod.
- Vzdálená I/O stanice – stanice, která používá pouze cyklické bitové přenosy. Příkladem je digitální I/O zařízení nebo elektromagnetický ventil.

Bitové informace jsou přenášeny za použití proměnných vzdálených vstupů (RX) a proměnných vzdálených výstupů (RY). Schéma je naznačeno na obr. 6. Proměnné vzdálených I/O a proměnné CPU programovatelného kontroléru jsou automaticky aktualizovány poté, co master provede Link scan[12].

[11] Word data – data o velikosti slova, typicky 16 bitů

[12] Link scan – úkon, při němž master stanice skenuje stav slave stanic prostřednictvím sítě (linku). Od odeslání dat z master stanice do jejich přijetí každou slave stanicí je provedena série operací. Obecně platí, že čím menší je celkový počet připojených zařízení, tím kratší je doba link scanu, což zlepšuje reakci vzdálených I/O zařízení.



Obrázek 6 - Schéma zasílání dat k externím vstupům a výstupům sběrnice CC-link [13]

3.3.4 ControlNet

ControlNet je otevřený síťový komunikační protokol, který splňuje vysoké požadavky na real-timové řízení v automatizaci. Typ zařízení, ve kterých figuruje komunikace pomocí ControlNetu zahrnuje například PLC, I/O rozvaděče, pohony, počítače, roboty apod. ControlNet je součástí rodiny protokolů CIP, které spravuje společnost ODVA (OpenDeviceNet Vendors Association. Inc.). ControlNet je obsažen ve standardech EN 50170 a IEC 61558.

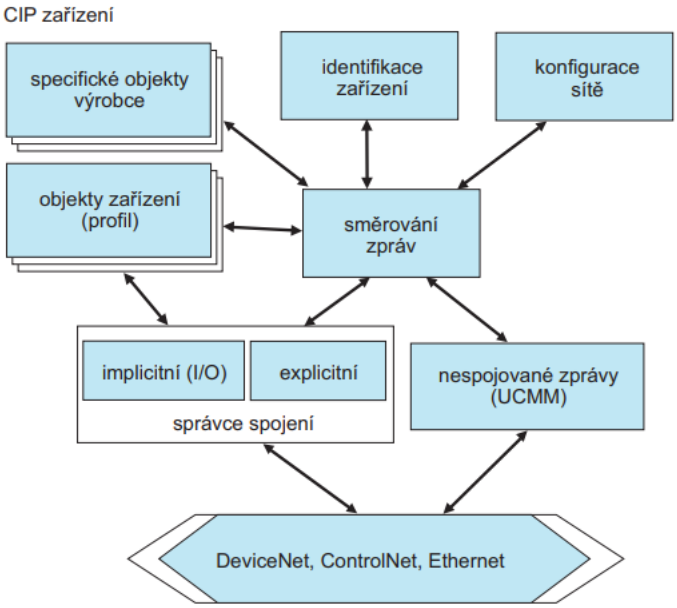
3.3.4.1 Protokol CIP

Jak již bylo zmíněno, ControlNet je součástí rodiny protokolů CIP. CIP je otevřený síťový protokol vytvořený ODVA, organizací, která spravuje komunikační protokoly založené na CIP. Konkrétními protokoly jsou, vyjma protokolu ControlNet, protokoly DeviceNet, CompoNet a Ethernet/IP. Původně byl CIP vytvořen z toho důvodu, že sběrnice používané v procesním řízení byly dříve optimalizovány pouze pro specifické aplikace. Pokud měly být uplatněny pro odlišnou aplikaci, nedokázaly splnit výkonnostní požadavky. Výrobci se museli rozhodnout, kterou sběrnici zvolit pro danou pasáž ve výrobě, nemohli se tedy spoléhat pouze na jeden typ sběrnice a nebyli schopni vytvořit ucelenou výrobu. Největším problémem byla ale neschopnost interoperability více protokolů. Tyto problémy vyřešil právě protokol CIP, který je schopný bez větších potíží spojit výše uvedené protokoly do ucelené sítě. [14]

CIP tvoří u všech protokolů vrchní vrstvy (relační, prezentační a aplikační) referenčního modelu ISO/OSI. CIP je striktně objektově orientovaný protokol – každé zařízení je podle protokolu reprezentováno skupinou objektů. Každý objekt obsahuje atributy (data), služby (příkazy) a specifikaci funkcí (reakce na události). V rámci CIP je definováno, jaká data musí obsahovat každý objekt. Existují tři skupiny objektů – povinné, aplikační a objekty definované výrobcem. Povinnými objekty jsou [15]:

- *Identification object* – objekt definující zařízení
- *Message router object* – objekt specifikující předávání zpráv
- *Connection object* – objekt pro správu spojení
- *Network link object* – jeden nebo několik objektů s parametry konfigurace komunikační sítě

Aplikační objekty obsahují data specifická pro komunikující zařízení a jsou vázány na typ a funkci těchto zařízení. Skupina aplikačních objektů tvoří profil zařízení. Výrobci si dle svých požadavků mohou vytvořit i vlastní objekty. Vzájemné vazby objektů v rámci zařízení naznačuje obr. 7.



Obrázek 7 - Objektový model zařízení s rozhraním CIP [15]

3.3.4.2 Fyzická vrstva

Jako fyzického média je na sběrnici ControlNet použito buď koaxiálního kabelu nebo optického vlákna. Maximální délka koaxiálního kabelu je 1000 m. S použitím opakovačů je možno dosáhnout vzdálenosti vyšších, opakovačů může být na sběrnici maximálně 5. Při použití optického vlákna lze dosáhnout vzdálenosti i několika km. Topologie sítě může být lineární, stromová, hvězdicová nebo jejich kombinace. Rychlost sběrnice dosahuje až 5 Mbps. Maximální počet uzlů na sběrnici je 99. [14]

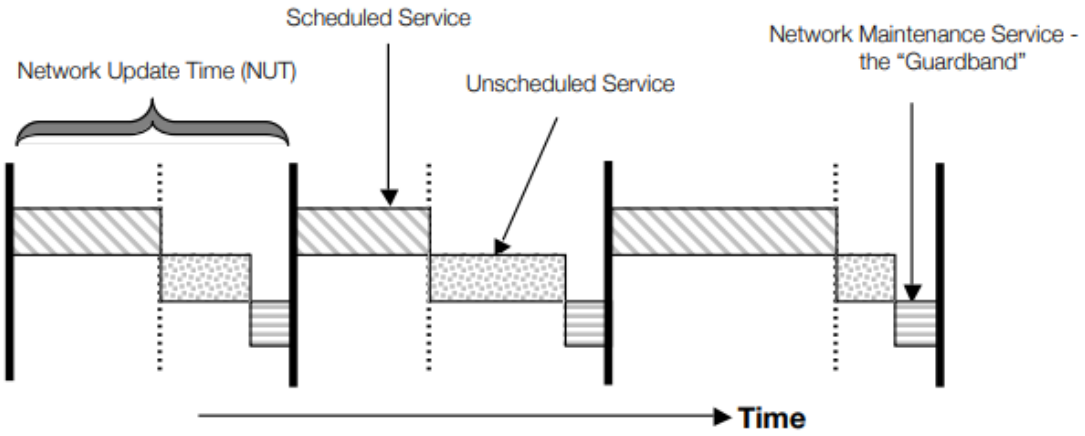
3.3.4.3 Linková vrstva

ControlNet může být sběrnice typu master-slave, multi-master nebo peer-to-peer[13]. Přístup ke sběrnici je realizován pomocí algoritmu zvaném *Concurrent Time Domain Multiple Access* (CTDMA). CTDMA reguluje datový provoz na sběrnici. Obr. 8 ilustruje mechanismus přístupu ke sběrnici. Data jsou zasílána v pravidelném intervalu zvaném *Network Update Time* (NUT). Z obr. 8 je patrné, že NUT je fixní. NUT je možné nastavit na čas v rozmezí 2-100 ms. Časový interval NUT je rozdělen do tří sekcí:

- *Scheduled*
- *Unscheduled*
- *Guardband*

[13] Peer-to-peer nebo také slave-slave komunikace je komunikace mezi podřízenými zařízeními v síti

Scheduled sekce je určena pro posílání časově kritických real-time dat. Sekce *Unscheduled* a *Guardband* jsou určeny pro méně kritická data. Posílání zpráv funguje na principu předávání tokenu. Token má při aktualizaci NUT v držení uzel s nejnižší adresou (MACID). Ve *Scheduled* sekci má možnost každý uzel poslat právě jednu zprávu a pokud zprávu pošle nebo žádnou zprávu poslat nehodlá, MACID se inkrementuje o jedničku a token se předá dalšímu uzlu. V sekcích *Unscheduled* a *Guardband* je uzlům umožněno poslat více než jednu zprávu, ale doručení není zaručeno. [14]



Obrázek 8 - Přístup na sběrnici ControlNet pomocí algoritmu CTDMA [14]

3.3.4.4 Síťová a transportní vrstva

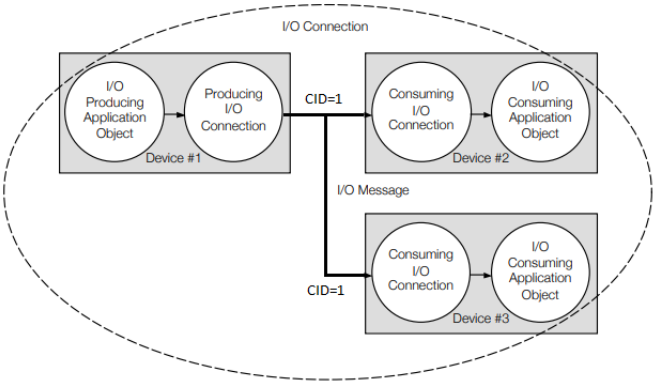
ControlNet používá dva druhy přenosu [14]:

- Explicitní přenos – je určen k přenosu typu žádost-odpověď při komunikaci dvou uzlů sítě. Explicitní posílání zpráv probíhá vždy sekci *Unscheduled* v NUT.
- Implicitní přenos je určen pro časově kritické posílání I/O zpráv. Obvykle bývá typu *multicast*. Implicitní přenos se uskutečňuje vždy v sekci *Scheduled* v NUT.

Komunikace u ControlNetu je dvojího typu [14]:

- Spojovaná komunikace – typ komunikace, která musí mít předem vytvořené spojení. Tato komunikace je využívána buď při posílání real-time dat nebo při frekventovaných explicitních přenosech. O navázání spojení mezi uzly stará tzv. *Unconnected Message Manager* (UCMM). Při navázání spojení se mezi komunikujícími uzly vytvoří tzv. *Connection ID* (CID). Poté komunikace probíhá na principu producent-konzument. Tento princip zaručuje efektivní využití sběrnice, jelikož se každá zpráva posílá pouze jednou (i v případě, že je stejná zpráva určena více uzlům). Jednotlivé uzly sítě se sami podle CID rozhodnout, zda jim je daná zpráva určena. Pokud má spojovaná komunikace probíhat oběma směry, je zapotřebí dvou CID. Příklad spojové komunikace pro přenos implicitních zpráv je naznačen na obr. 9.

- Nespojovaná komunikace je typ komunikace, která nemusí mít předem vytvořené spojení. Používá se při procesu navazování spojení nebo při zasílání nefrekventovaných, málo kritických dat.



Obrázek 9 - Multicast I/O implicitní spojení [14]

3.4 Průmyslový Ethernet

3.4.1 Úvod do Ethernetu

Ethernet byl vyvinut firmou Xerox PARC v roce 1976. Je to nejpopulárnější souhrn technologií používaných v sítích *Local Area Network* (LAN). Specifikace fyzické a linkové vrsvy Ethernetu určuje standard IEEE 802.3. Ethernet se stal populární především díky skvělému kompromisu mezi rychlostí, cenou a snadnou instalací.

3.4.1.1 Fyzická vrstva

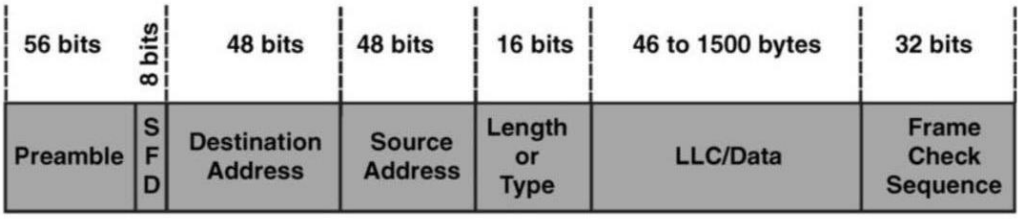
Jelikož byl Ethernet představen na trh před mnoha lety, prošel několika generacemi díky postupnému zvyšování rychlosti komunikace. Jednotlivé generace jsou *Standard Ethernet* (10 Mbps), *Fast Ethernet* (100 Mbps), *Gigabit Etheret* (1 Gbps) a *Ten Gigabit Ethernet* (10 Gbps). Druh Ethernetu se popisuje ve formátu rychlostBASE-typ přenosového média. Například označení 10GBase-T znamená, že je použit Ethernet s přenosovou rychlostí 10 Gbps a jako přenosového média je použito kroucené dvojlinky (T-Twisted pair). Dalším médiem hojně používaných při kabeláži Ethernetu je *Fiber* neboli optické vlákno. Ve většině případů je Ethernet zakončen koncovkou RJ-45. Komunikace je tzv. *full-duplex* (plně duplexní), tedy je možná komunikace v kabelu oběma směry. Délka kabelu mezi dvěma zařízeními by neměla přesáhnout 100 m u kroucené dvojlinky a 2 km u optického vlákna. [16]

3.4.1.2 Linková vrstva – ethernetový rámec

Pro komunikaci v linkové vrstvě Ethernetu je důležitý tzv. ethernetový rámec, který obsahuje veškeré potřebné informace pro úspěšný přenos dat. Ethernetový rámec je zobrazen na obr. 10 a skládá se z následujících polí [16]:

- *Preamble* – sekvence střídajících jedniček a nul (10101010). Toto pole synchronizuje hodiny mezi odesílatelem a příjemcem.

- *Start Frame Delimiter* (SFD) – pole obsahující data 10101011, která umožňují příjemci rozpoznat začátek rámce. Poslední dva bity (11) upozorňují příjemce, že následující pole je cílová adresa
- *Destination Adress* – cílová adresa (6 bitů)
- *Source Adress* – zdrojová adresa (6 bitů)
- *Length or Type* – pole obsahující délku datového pole nebo pole značící typ protokolu vyšší vrstvy. Pokud se jedná o typ protokolu, je číslo decimálně vždy větší než 1500
- *Data* – pole se samotnými daty (46-1500 bytů)
- *Frame Check Sequence* – kontrolní součet



Obrázek 10 - Ethernetový rámec [16]

3.4.2 Potřeba průmyslového Ethernetu

Ethernet, vyvinutý před více než čtyřiceti lety, byl zprvu využíván především pro komunikaci mezi zařízeními v kancelářském prostředí. Jelikož byl postupem času požadavek na real-time komunikaci pomocí Ethernetu, vyvinul se postupem času průmyslový Ethernet. Průmyslový Ethernet se od klasického kancelářského Ethernetu liší v robustnosti kabelů a v hardwaru, aby byl schopen odolávat vibracím a drsnému prostředí s vysokou teplotou, vlhkostí a prachem. Dále se liší například v prioritizaci dat a především v determinismu. [17]

3.4.3 Ethernet/IP

Ethernet/IP je jeden z nejpoužívanějších protokolů v automatizaci. Ethernet/IP je součástí rodiny protokolů CIP, spravovaných společností ODVA. Mezi základní charakteristiky tohoto protokolu patří rychlost přenosu až 100 Mbps, jitter[14] 1 ms, možnost mnoha různých topologií (prstencové, stromové, liniové).

3.4.3.1 ISO/OSI model protokolu Ethernet/IP

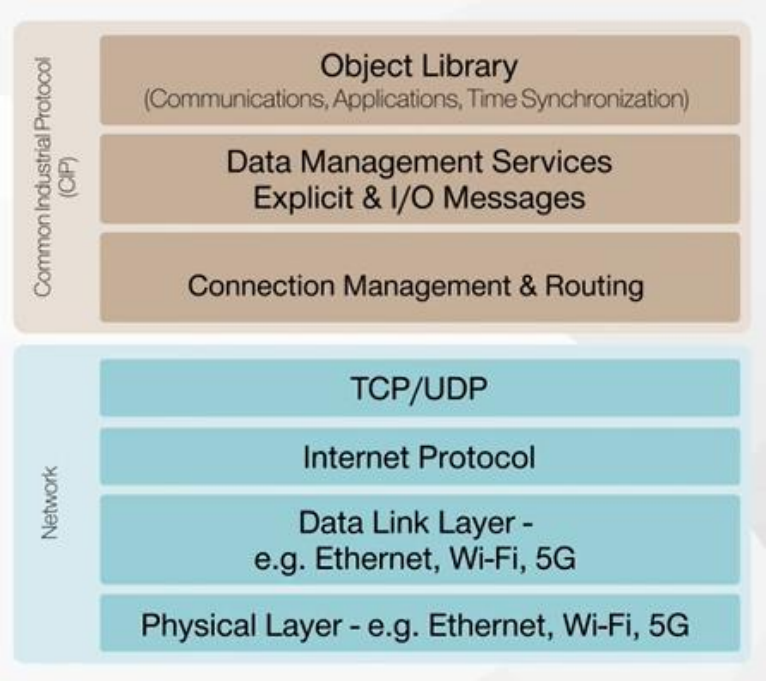
Ethernet/IP používá protokol CIP pro vrchní část síťového modelu (aplikační, prezentační a relační vrstvu) a nasazuje ho na standardní síťové technologie jako je Ethernet, Wi-fi nebo 5G. Podobně jako u protokolu ControlNet používá dvou typů přenosu zpráv:

- Explicitní přenos práv pro výměnu časově nekritických dat
- implicitní přenos zpráv pro přenos časově kritických real-time dat.

Transportní vrstvu tvoří protokoly *User Datagram Protocol* (UDP) a *Transmission Control Protocol* (TCP). UDP je protokol s nízkým zpožděním a používá se pro I/O komunikaci za

[14] Jitter – kolísání velikosti zpoždění paketů při průchodů sítí

použití implicitních přenosů zpráv. UDP je typ nespojové komunikace, tedy před zahájením komunikace nemusí navázat spojení s cílovým zařízením – tím pádem je rychlejší, ale méně spolehlivější než TCP. TCP umožňuje posílání velkého množství dat tak, aby se zachovala jejich integrita. Je to typ spojové komunikace, kdy se naváže spojení mezi oběma komunikujícími uzly. Zachování integrity a zaručení přenosu všech dat dělá TCP spolehlivějším, ale pomalejším protokolem oproti UDP. Síťová vrstva je realizována pomocí protokolu *Internet Protocol* (IP) [18]. Na obr. 11 je zobrazeno schéma modelu ISO/OSI protokolu Ethernet/IP.



Obrázek 11 - ISO/OSI model protokolu Ethernet/IP [18]

3.4.4 Ethernet Powerlink

Ethernet Powerlink (EPL) je jeden z nejrychlejších real-time komunikačních protokolů spravovaný skupinou *Ethernet Powerlink Standardization Group* (EPSG) a představený na trh rakouskou firmou Bernecker & Rainer Industrie Elektronik GMBH v roce 2001. EPL je obsažen ve standardech 61158-300, IEC-61158-600 a IEC 61558-500.

Mezi základní vlastnosti EPL patří deterministická komunikace pomocí spojení mechanismů *polling* a *time slot*[15], komunikační cyklus pod 200 µs, použití rozbočovačů namísto přepínačů ke zmenšení jitteru až pod 1 µs, možnost realizace mnoha topologií a jejich kombinace, podpora metody *hot plugging*, kdy se zařízení může kdykoliv připojit a odpojit ze sítě, aniž by se musela restartovat celá síť, možnost připojení až 240 zařízení, komunikace pomocí modelu producent-konzument i pomocí modelu klient-server. EPL se využívají v sítích obsahující zařízení typu PLC, I/O, HMI, pohybové kontroléry, bezpečnostní senzory apod. [19] [20]

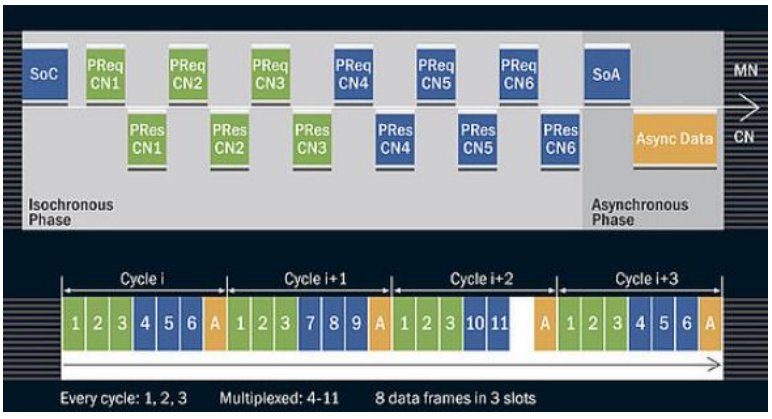
[15] Time-slot – přiřazení doby pro přenos

3.4.4.1 Vrstvy protokolu EPL a způsob komunikace

První i druhá vrstva EPL je plně kompatibilní se standardem Ethernet. Co se týče linkové vrstvy, je zde možnost dvou módů – základní Ethernet TCP/IP mód a Powerlink mód. EPL může pracovat jak v real-time módu, tak v módu časově nekritických dat. V základním Ethernet TCP/IP módu se může zařízení nakonfigurovat na real-time mód. Každé zařízení EPL se v základním módu může připojit do jakékoli sítě Ethernet bez ohledu na to, zda síť pracuje v režimu reálného času či nikoli. Powerlink mód umožňuje stanici pracovat real-time. Real-time komunikace probíhá za použití komunikačního cyklu, jehož dobu lze nastavit. Komunikační cyklus je rozdělen na následující části [19]:

- *Star Of Cycle* (SOC) – začátek cyklu
- Izochronní přenos
- Asynchronní přenos

EPL je průmyslová sběrnice typu master-slave. Master (PC nebo PLC) se nazývá *Managing Node* (MN) a slave zařízení se nazývá *Controlled node* (CN). Každý cyklus začíná spouštěcí posloupností SoC zasílanou masterem sítě. Na základě SoC si všechny řízené uzly synchronizují své hodiny. Následuje fáze izochronního přenosu časově kritických dat. Každý přenos má přiřazený svůj *time slice*[16] , aby se dodržel determinismus. Z obr. 12 je zřejmé rozdělení přenosového cyklu EPL v módu Powerlink. Arbitr sítě posílá *Poll Request*[17] (Preq) každému uzlu a vyzvaný řídící uzel vyšle v přesně definovaném čase bezprostředně po výzvě MN do sítě *Poll Response*[18] (Pres), kterou mohou přijímat všechny uzly. Po části izochronní následuje asynchronní část přenosu, kdy MN umožní jednotlivým uzlům poslat časově nekritická data. Asynchronní fázi spouští opět MN startovací posloupností nazývanou *Start Of Asynchronous* (SoA). V asynchronním přenosu se typicky přenášejí konfigurační data, diagnostická data apod. Aby bylo co nejlépe využito přenosové pásmo, lze v módu Powerlink realizovat tzv. *Multiplexing* (prokládaný režim) naznačený na obr. 12, kdy jsou údaje nevyžadující přenos dat každém cyklu přenášeny v rámci sdílených časových oken, která jsou sdílena několika uzly. [19]



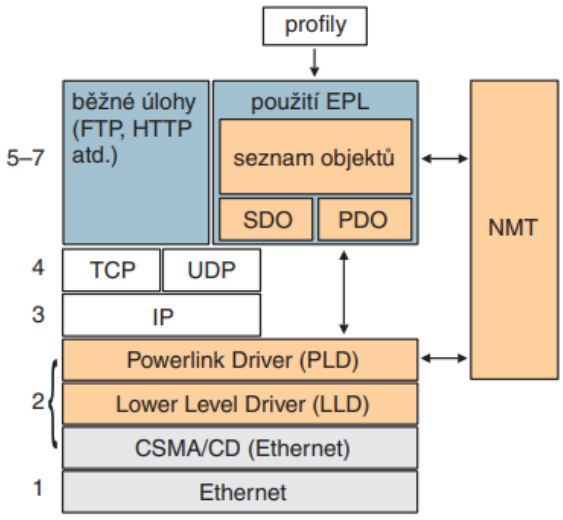
Obrázek 12 - Přenosový cyklus Ethernet Powerlink v horní části a multiplexing v dolní části [20]

[16] Time-slice – doba pro přenos
[17] Poll Request – žádost o přenos
[18] Poll Response – odpoveď s daty

Podrobný komunikační model je zobrazen na obr. 13. Součástí linkové vrstvy jsou tzv. *Powerlink Driver* (PLD) a *Low Layer Driver* (LLD), které zapouzdřují funkce fyzické a linkové vrstvy a umožňují tak rychle přenášet časově kritická data. Z obr. 13 je dále patrné, že síťová a transportní vrstva jsou využívány jen pro přenos časově nekritických dat. Pro přenos real-time dat je vytvořen softwarový obchvat těchto vrstev. Vrstvy 5-7 jsou uskutečněny pomocí standardu CANopen, podle kterého bylo vytvořeno již mnoho profilů automatizačních komponent různých výrobců, kteří se tak mohou svými výrobky zapojit do jednotného systému CANopen. Aby mohla zařízení spolu komunikovat v sériové síti, je zapotřebí aby obsahovala:

- *Network Management* (NMT) – systém řízení sítě
- Seznam objektů a modely zařízení
- *Process Data Object* (PDO) – objekty provozních dat
- *Service Data Object* (SDO) – objekty servisních dat
- Profily zařízení, což jsou standardizované definice dat, parametrů a funkcí určitých typů zařízení, jako jsou pohony, moduly I/O, snímače polohy atd.

Časově kritická výměna dat se uskutečňuje pomocí objektů PDO přenášených v izochronní části přenosového cyklu EPL s využitím komunikace modelu producent-konzument, který umožňuje posílat data z jednoho uzlu do více uzlů současně. Parametry, funkce a časově nekritická data jsou naopak přenášeny prostřednictvím objektů typu SDO při použití pomalejšího komunikačního modelu klient-server v asynchronní části cyklu. [19] [20]



Obrázek 13 - Komunikační model Ethernet Powerlink [19]

3.4.5 PROFINET IO

PROFINET (PROcess FIeld NETwork) je otevřený průmyslový standard založený na Ethernetu založený společností PROFIBUS & PROFINET International. PROFINET je standardizován v IEC 61158 a IEC 61784. Podobně jako PROFIBUS je i PROFINET vyvinut a používán podle povahy úloh ve více variantách tak, aby pokryl celou oblast automatizace.

PROFINET lze rozdělit na dvě varianty [21]:

- *PROFINET CBA* (Component Based Automation), představující koncept komunikace pro distribuované řízení, jehož komponenty jsou autonomní řídící systémy s vlastní inteligencí, tvořící celek – například výrobní linky.
- *PROFINET IO*, který je určen pro komunikaci s decentralizovanými periferiemi, podobně jako PROFIBUS DP a který bude dále rozebrán v této části.

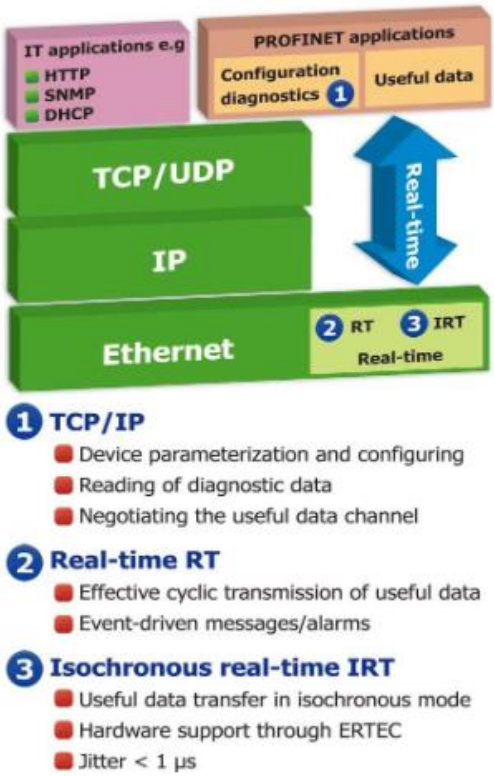
Typy zařízení v PROFINET IO [21]:

- *IO-Controller*, který řídí komunikaci v celé síti. Zabezpečuje výměnu signálů mezi ostatními účastníky v síti. Většinu je to zařízení obsahující řídící program.
- *IO-Device* je podřízený účastník sítě komunikujícím s IO-Controller. Typicky jde o vzdálené I/O zařízení, inteligentní měřící členy, akční členy apod.
- *IO-Supervisor*, což je zařízení s rozhraním HMI nebo diagnostické zařízení v síti

3.4.5.1 Komunikace

Standard PROFINETu IO definuje tři komunikační třídy [22]:

- PROFINET *Non-Real Time* (NRT), který se používá při acyklické, časově nenáročné výměně dat. Pro přenos používá protokol TCP/IP nebo UDP/IP při času cyklu kolem 100 ms.
- PROFINET *Real-Time* RT pro přenos časově kritických dat s cyklem okolo 10 ms buď za použití čistě ethernetové vrstvy a obejití protokolů TCP/IP a UDP/IP jako je zobrazeno na obr. 14 nebo pro přenos především konfiguračních dat za použití protokolů TCP/IP a UDP/IP.
- PROFINET *Isochronous Real Time*, který byl vyvinut především pro náročné aplikace vyžadující vysokorychlostní real-time řízení pohybu, s časovým cyklem okolo 250 µs a jitterem 1 µs.

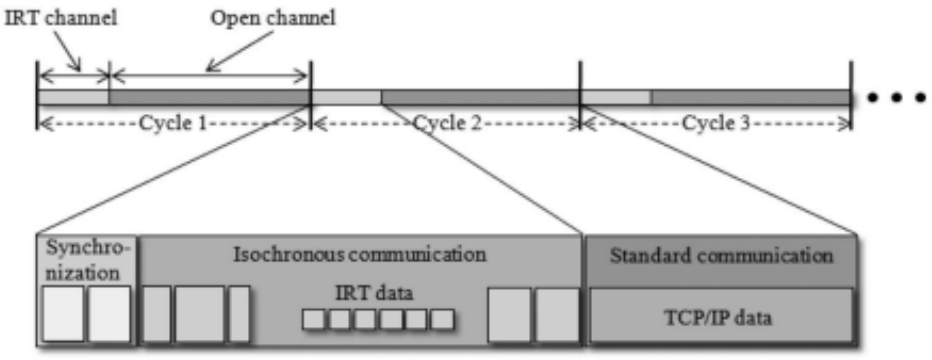


f

Obrázek 14 - Profinet IO komunikace [22]

3.4.5.2 Princip funkce IRT

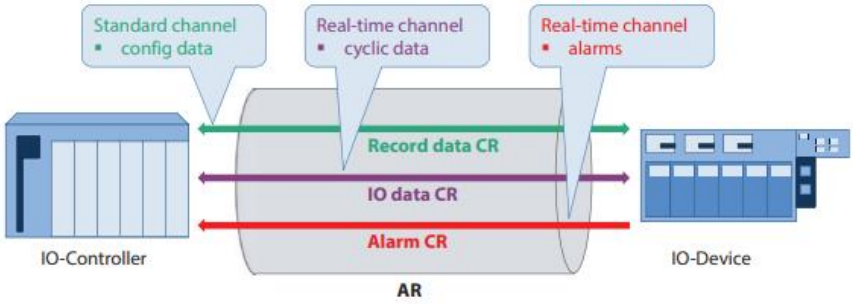
Princip IRT spočívá v *time-slice* mechanismu rozdělující cyklus na deterministickou a otevřenou část. Přenášení dat v IRT se skládá jak z IRT, tak z RT a NRT, kde pro každou část je vyhrazena určitá šířka pásma, jak je ukázáno na obr. 15. Časově kritická data jsou přenášena v deterministickém kanálu, který je časově synchronizován s použitím speciálního hardware jednotlivých účastníků sítě. Použitím speciálního hardwaru lze dosáhnout cyklu z 250 µs na 31,25 µs. [23]



Obrázek 15 - Komunikační cyklus PROFINET IRT [23]

3.4.5.3 Komunikace mezi zařízeními

Aby mohl *IO-controller* komunikovat s podřízenými účastníky sítě, je nutné založit komunikační cestu. Tuto komunikační cestu zakládá *IO-controller* během spuštění systému na základě konfiguračních dat, které *jsou IO-controlleru* zadány. Na počátku komunikace se vytvoří tzv. aplikační relace (AR). Uvnitř aplikační relace se současně vytvoří komunikační relace (CR), tj. komunikační kanály pro výměnu cyklických dat, acyklických dat a alarmových dat (obr. 16). [24]



Obrázek 16 - Aplikační a komunikační relace Profinetu IO [24]

3.4.6 SERCOS III

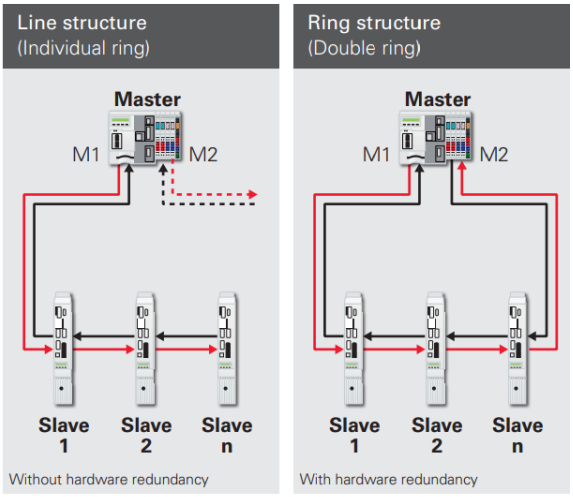
SErial Real time COmmunication System (SERCOS III) je otevřená a IEC kompatibilní univerzální sběrnice pro real-time komunikaci založenou na standardu Ethernet představená v roce 2005, spravovaná společností SERCOS International e.V. Sercos III, třetí generace sběrnice SERCOS je používaná především v oblasti vysokorychlostního pohybového řízení.

3.4.6.1 Vlastnosti

SERCOS III pracuje bez rozbočovačů a přepínačů a tím rapidně snižuje zpoždění. Stejně jako PROFINET je schopen přenosu rychlosti až 31,25 µs a jitterem menším než 1 µs. SERCOS III používá topologii prstencovou nebo liniovou. Zařízení mohou být připojena do sítě za běhu (*hot plugged*). Je možná komunikace jak master-slave, tak i komunikace master-master a slave-slave. [23]

3.4.6.2 Topologie

Každé zařízení SERCOS III má dva ethernetové porty – vstupní port spojený s výstupním portem předchozího zařízení a výstupní port propojený se vstupním portem zařízení následujícího. Zařízení jsou spojena pomocí CAT5e ethernetového kabelu. Jak bylo zmíněno výše, SERCOS III může být nakonfigurován na prstencovou nebo liniovou topologii. S liniovou topologií má master zařízení zapojeno kabel pouze v jednom portu do portu následujícího slave zařízení. Data procházejí postupně každým uzlem za sebou. Poslední slave zařízení pošle data zpět k master zařízení. Přidáním jednoho kabelu, který propojí poslední slave zařízení s master zařízením vznikne prstencová topologie. V tomto případě master posílá data oběma směry z obou portů současně. Výhoda prstencové topologie je v podpoře tzv. redundance, kdy při poruše kteréhokoli spojení začne SERCOS III fungovat dál jako dvě liniové topologie. Obě topologie jsou vidět na obr. 17. [23]



Obrázek 17 - SERCOS III topologie, liniová vlevo a prstencová vpravo [25]

3.4.6.3 Konfigurace komunikačního cyklu

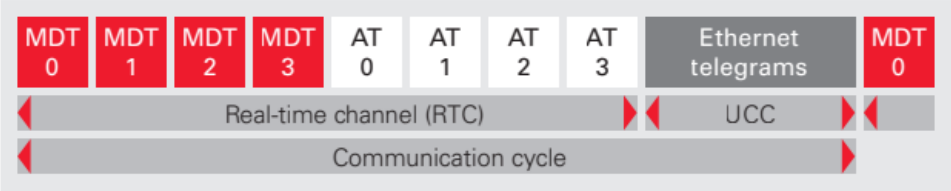
Komunikace v SERCOS III je založená na *time-slot* mechanismu v cyklickém zasílání telegramů. Komunikační cykly mohou být nastaveny od času 31,25 µs až po 65 ms. Konfigurace komunikačního cyklu SERCOS III je zobrazena na obr. 18. Cyklus je rozdělen do dvou časových kanálů:

- *Real Time Channel* (RTC) – real-time kanál pro posílání real-time dat.
- *Unified Communication Channel* (UCC) – unifikovaný komunikační kanál

Časově nenáročná data jsou pomocí protokolů TCP/IP nebo UDP/IP posílána v kanálu UCC. V RTC je použito dvou typů telegramů:

- *Master Data Telegram* (MDT)
- *Acknowledge Telegram* (AT)

MDT obsahuje informace poskytnuté masterem, čtené slave zařízeními. AT je poslán slave zařízením s odpovědí. Komunikační cyklus se může skládat z více než jednoho MDT a AT. [23]



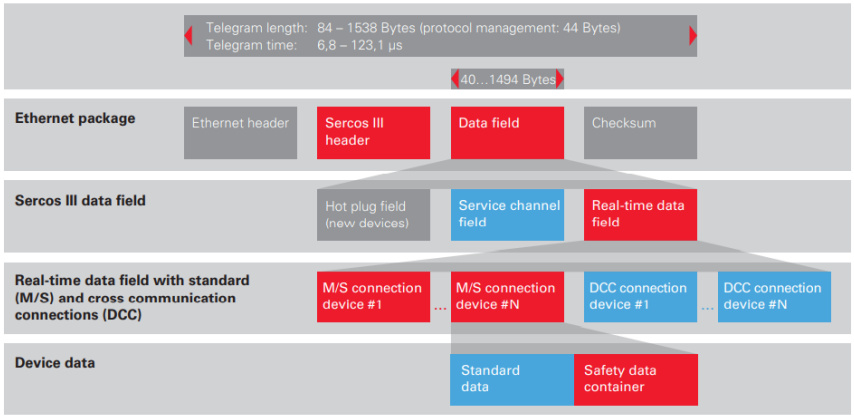
Obrázek 18 - Konfigurace komunikačního cyklu SERCOS III [25]

Hlavička telegramu MDT nebo ADT je vložena do standardního ethernetového rámce a obsahuje pozici daného telegramu v komunikačním cyklu. Dále v datagramu MDT a AT je datové pole, které je rozděleno na další tři dílčí pole. Tato pole jsou *hot plug*, servisní pole a pole s real-time daty. *Hot plug* pole vyměňuje data se slave zařízením, které bylo připojeno do sítě za běhu. Servisní pole obsahuje celkový počet komunikačních kanálů

pro výměnu acyklických dat mezi master a slave zařízením. Pole s real-time daty obsahuje acyklická data, cyklická data nebo data pro synchronizaci hodin. Pole s real-time daty obsahuje jednotlivé komunikační spojení mezi master a slave zařízením (M/S) nebo komunikaci mezi dvěma slave zařízeními (DCC), respektive master zařízeními (C2C). Jednotlivá spojení obsahují standartní, popřípadě bezpečnostní data. Detail telegramu SERCOS III lze vidět na obr. 19. [25]

3.4.6.4 Peer-to-peer komunikace

Jak již bylo zmíněno, SERCOS III umožňuje komunikaci nejen na principu master-slave, ale i komunikaci mezi jednotlivými zařízeními stejné úrovně, a to díky full-duplexní charakteristice Ethernetu. Jak u prstencové, tak u liniové topologie projde každý telegram dvakrát každým uzlem. To umožňuje danému uzlu poslat data jinému uzlu stejné úrovně. Dva typy peer-to-peer komunikace jsou podporovány v SERCOS III, jsou to controller-to-controller (C2C) pro výměnu dat mezi dvěma master zařízeními a slave-to-slave (DCC) pro výměnu dat mezi slave zařízeními. [23]



Obrázek 19 - Struktura telegramu SERCOS III [25]

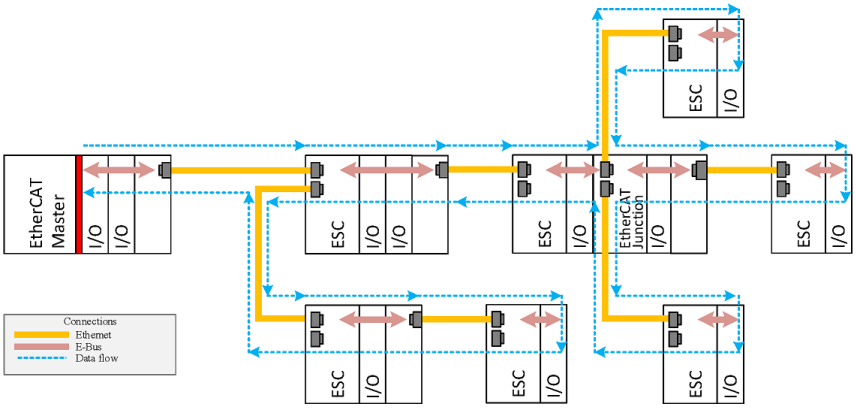
3.4.7 EtherCAT

Ethernet for Control Automation Technology (EtherCAT) je Fieldbus založený na Ethernetu, vyvinutý německou společností Beckhoff Automation v roce 2003. EtherCAT byl primárně vyvinut pro použití v průmyslové automatizaci pro vysokorychlostní, real-time řízení. EtherCAT je ze všech zmíněných protokolů založených na Ethernetu nejrychlejší aplikační sběrnicí. Otevřená technologie EtherCATu je obsažena ve standardech IEC61158, 617184, 61800 a ISO 15745. [26]

3.4.7.1 Popis sběrnice EtherCAT

Vyjma absence rozbočovačů a přepínačů je hlavní důvod vysoké rychlosti EtherCATu princip předávání dat mezi master a slave zařízením. Data se předávají principem „za letu“ („*on the fly*“). Ke sběrnici EtherCAT lze možno připojit až 65 535 zařízení. EtherCAT používá IEEE 802.3 ethernetovou fyzickou vrstvu a standardní ethernetové rámce. EtherCAT je sběrnice typu master-slave, kde slave zařízení mohou být zapojena fyzicky v nejrůznějších topologiích (strom, prsten, hvězda, linie). Fyzicky jsou zařízení zapojena v sérii a logicky pracují ve smyčce, jak je naznačeno na obr. 20. EtherCAT používá full-

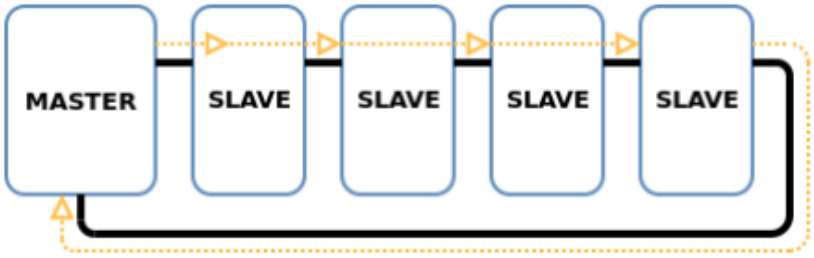
duplexní komunikaci prostřednictvím CAT5 kabelů. Pro připojení master zařízení do sítě je zapotřebí pouze jediného ethernetového portu. [26]



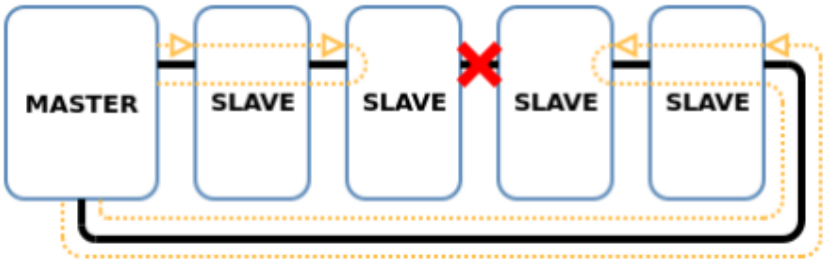
Obrázek 20 - Tok dat v logické smyčce sbernice EtherCAT [27]

3.4.7.2 Odolnost sítě proti fyzickému přerušení

Při propojení druhého ethernetového portu s posledním slave zařízením vznikne fyzická prstencová topologie, která je odolnější proti fyzickému přerušení (podporuje redundanci) oproti liniové topologii. Princip je naznačen na obr. 21, resp. na obr. 22. Master při detekci chyby kvůli fyzickém porušení vytvoří dvě logické komunikační smyčky, a tak může komunikace probíhat i nadále. [26]



Obrázek 21 - Průchod paketu EtherCAT při prstencovém zapojením bez poruchy [28]

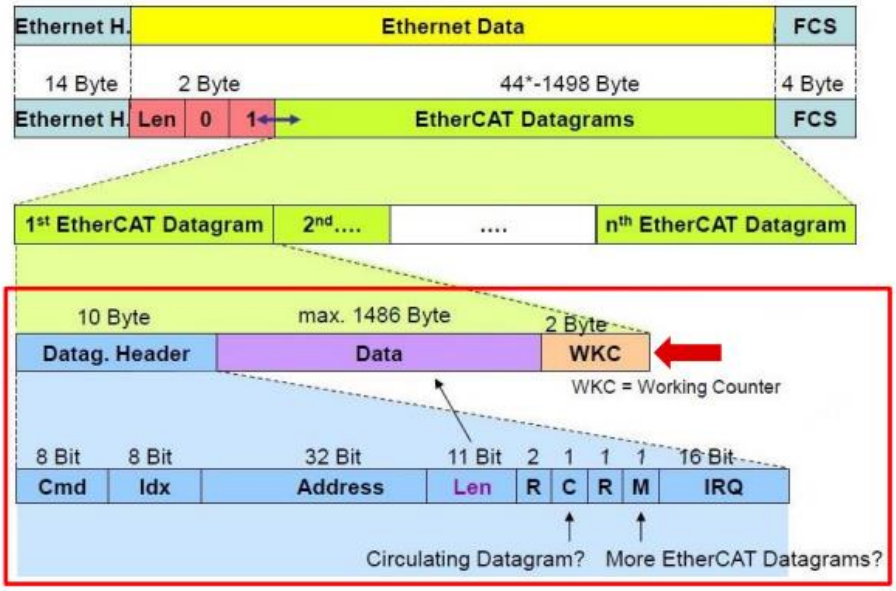


Obrázek 22 - Průchod paketu EtherCAT při prstencovém zapojením s poruchou [28]

3.4.7.3 EtherCATový rámec

EtherCAT používá standardní ethernetový rámec s odlišnou identifikační hlavičkou pro EtherCAT. Každý rámec obsahuje jeden nebo více datagramů, které obsahují svou vlastní hlavičku, data a čítač (*working counter*). Pole hlavičky datagramu obsahuje různé informace, jako jsou příkazy, adresy, délka dat atd. Čítač je zvýšen s každou interakcí

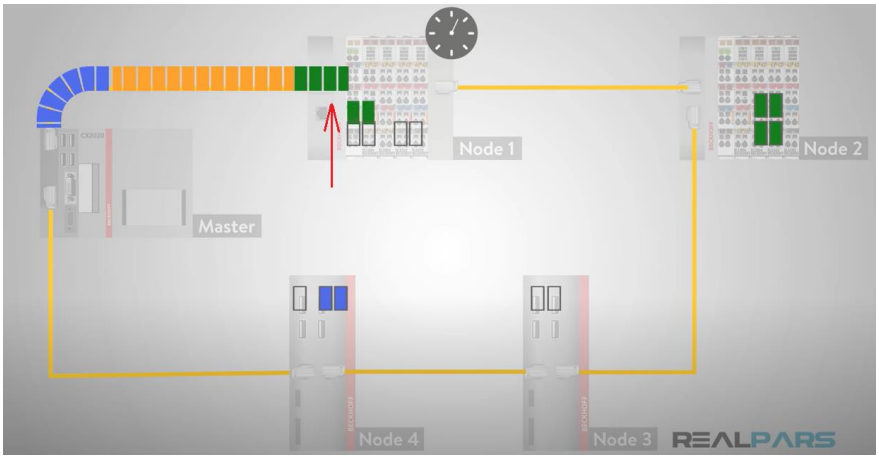
obsaženou v daném datagramu. Například, pokud příkaz v datagramu je číst, resp. zapisovat, tak při úspěšném čtení, resp. zápisu, se čítač zvětší o jedničku. EtherCAT master monitoruje tento čítač a dokáže detekovat chyby, pokud nastanou problémy v síti [6]. Struktura rámce je vidět na obr. 23.



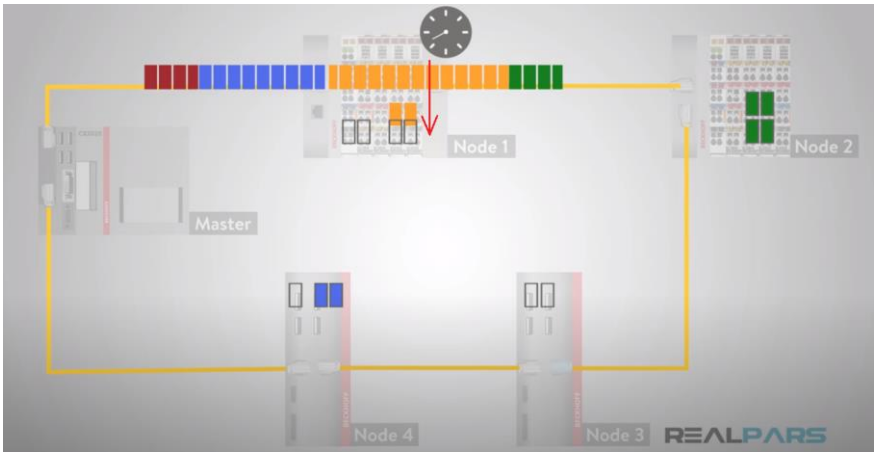
Obrázek 23 - Struktura EtherCATového rámce [26]

3.4.7.4 Komunikace „on the fly“

Jak bylo řečeno na začátku, EtherCAT komunikace probíhá „*on the fly*“. To znamená, že data jsou přijata a vložena do EtherCATového rámce, zatímco rámec proběhne každým uzlem v plné své rychlosti. Proces extrakce a vkládání dat je zajištěn pomocí hardware, a tak není zapotřebí speciálního software. Tento proces není omezen velikostí datagramu, který může být v rozmezí od 1 bitu až po 16 kilobytů [26]. Princip „*on the fly*“ je naznačen na obr. 24 a obr. 25.



Obrázek 24 - Slave zařízení zapisuje do EtherCAT datagramu „za letu“ [29]



Obrázek 25 - Slave zařízení čte ze stejného EtherCAT datagramu „za letu“ [29]

3.4.7.5 FMMU

Field Memory Management Unit (FMMU) zajišťuje mapování skutečných fyzických adres zařízení na logické adresy. To umožňuje zefektivnit komunikaci omezením množství datagramů, jelikož se jedna logická adresa může namapovat na více adres fyzických [26].

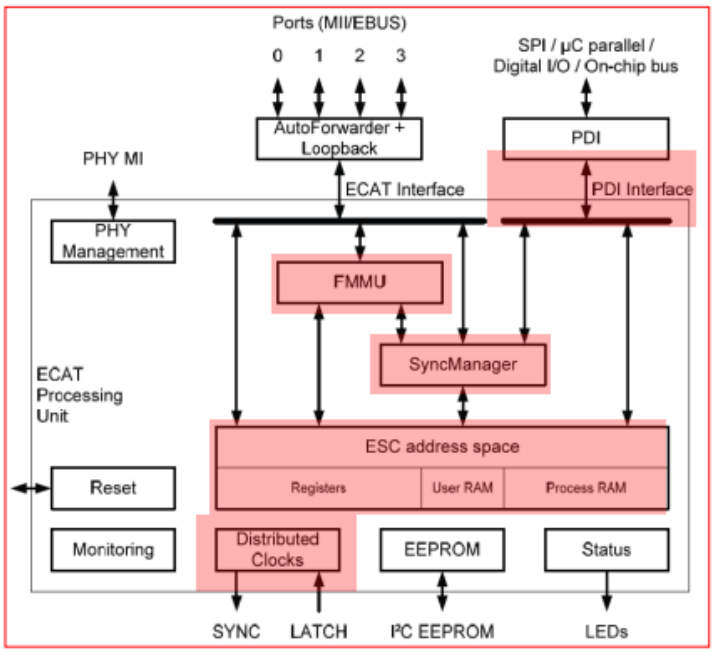
3.4.7.6 Distribuované hodiny

Pro práci s deterministickými daty používá EtherCAT tzv. distribuované hodiny. Distribuované hodiny umožňují zařízením sdílet stejný EtherCATový systémový čas. EtherCAT master kontroluje zpoždění mezi každým slave zařízením pomocí referenčních hodin, které jsou typicky přiděleny prvnímu slave zařízení v síti. Synchronizace hodin začíná zasláním synchronizačního datagramu mastera ke všem slave zařízením. Slave zařízení poskytnou masterovi svůj lokální čas. Master podle lokálních časů jednotlivých slave zařízení vypočítá offset, který si slave zařízení přičte ke svému lokálnímu času. Tímto způsobem je zajištěno, že signál v síti i se stovkami slave zařízeními se neliší o více než 100 nanosekund. [30]

3.4.7.7 EPU

Ethercat Processing Unit (EPU) je logické jádro každého EtherCAT slave zařízení. Obsahuje registry, paměťové elementy a elementy zpracovávající data. Obsah konkrétních komponent EPU je zobrazen na obr. 26. Každé EPU obsahuje:

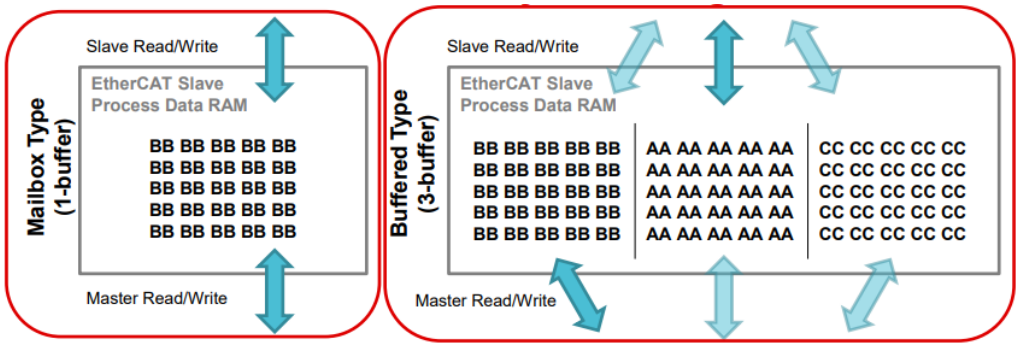
- *Dual-Port RAM* (DPRAM) – paměť typu RAM
- *FMMU*
- *SyncManager* – mechanismus zajišťující konzistentnost výměny dat skrz mailboxy mezi master a slave zařízením
- *Process Data Interface* (PDI) – rozhraní propojující slave aplikaci se slave zařízením (například skrz SPI)
- Distribuované hodiny



Obrázek 26 - Logické jádro EtherCAt slave zařízení [31]

3.4.7.8 SyncManager

SyncManager chrání DPRAM před současným přístupem od master zařízení prostřednictvím EtherCAT sítě a mikrokontroléru prostřednictvím PDI. *SyncManager* nabízí dva typy konfigurace – mailbox a buffer. Mailbox je v podstatě jednoslotový buffer. V mailbox konfiguraci se k DPRAM dostane pouze jedno zařízení, např. EtherCAT master. Druhé zařízení musí počkat, dokud první zařízení nedokončí svoji akci – tedy zapisující zařízení musí nejdříve zapsat data předtím, než je druhé zařízení zapíše nebo musí nejdříve přečíst data předtím, než data do DPRAM zapíše druhé zařízení. Buffer mód využívá víceslotového bufferu. Tyto tři buffery zajišťují doručení dat a přístup k nejnovějším datům kdykoli je o ně žádáno. Pokud například EherCAT master zapisuje, druhé zařízení nemusí čekat, než samo zapíše, ale jednoduše začne zapisovat do následujícího bufferu. Pokud chce jakékoli zařízení číst data, bude odkázáno na buffer s nejnovějšími daty [31]. Mailbox a buffer konfigurace je naznačena na obr. 27.



Obrázek 27 - EtherCAT komunikace v mailbox módu (vlevo) a v buffer módu (vpravo) [31]

3.4.7.9 EtherCAT State Machine

Přístup k individuálním funkcím záleží na stavu EtherCAT slave zařízení, který je kontrolován EtherCAT statovým automatem. V každém stavu může být slave zařízení od EtherCAT mastera posílány specifické příkazy. Těmito stavy jsou [31]:

- *Init* – EtherCAT slave se v tomto stavu nachází po svém zapnutí, žádná komunikace není možná
- *Pre-Operational* – Když se slave dostane z *Init* stavu do *Pre-Op* stavu, mailbox je inicializován, to znamená, že mailbox komunikace je možná v tomto stavu. V tomto stavu nelze posílat procesní data.
- *Safe operational* – zde je povolena jak mailbox komunikace, tak procesní komunikace. V tomto stavu jsou vyhodnocována pouze vstupní data, výstupní data jsou držena v bezpečnostním stavu.
- *Operational* – je možná jak procesní, tak mailbox komunikace

4 Implementace sběrnice EtherCAT

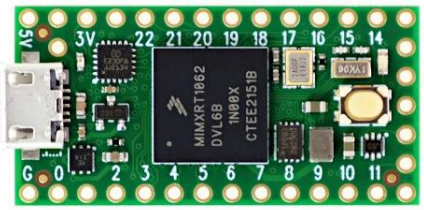
4.1 Použitý hardware

Následující část se bude zabývat použitými hardwarovými prvky v novém řídícím systému

4.1.1 Teensy 4.0

Teensy 4.0 (obr. 28) je vývojová deska s 32bitovým procesorem ARM Cortex-M7 vyvinutá firmou PJRC. Mezi hlavní specifikace této desky patří [32]:

- Taktovací frekvence procesoru 600 MHz
- 1984 kB Flash Paměť
- Připojení USB s rychlostí až 480 Mbit/s
- 1024 kB pamět RAM
- 1 kB paměť EEPROM
- 40 vstupně-výstupních pinů

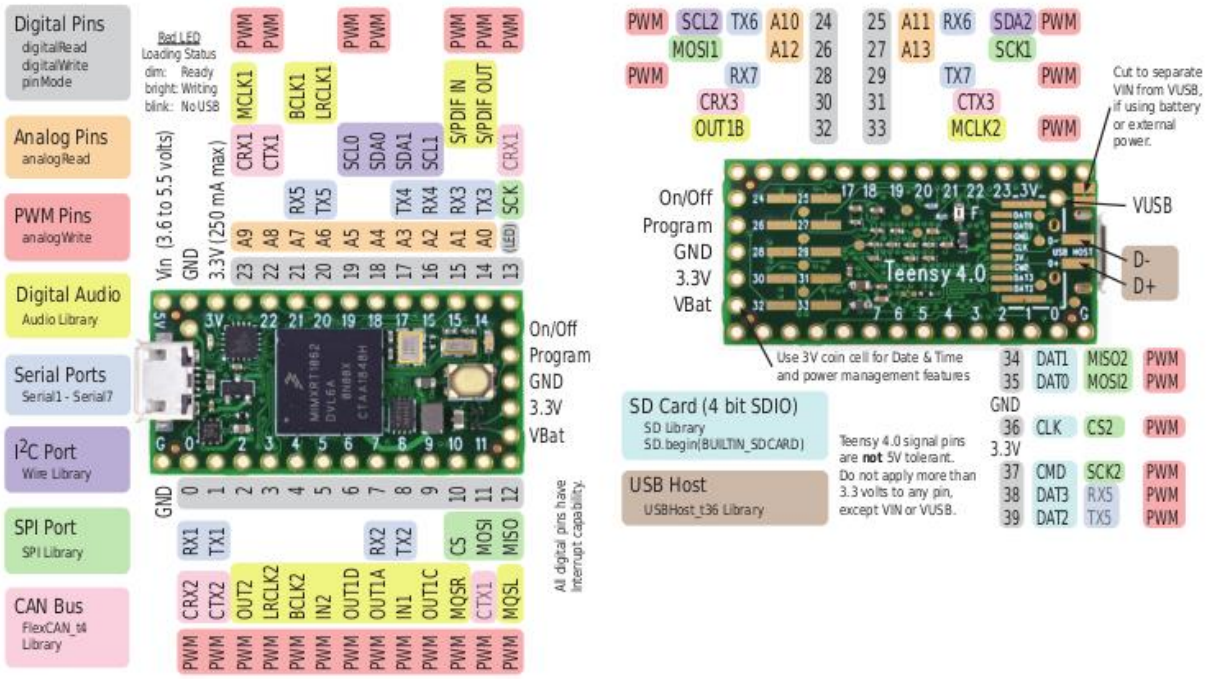


Obrázek 28 - Vývojová deska Teensy 4.0 [32]

Jednotlivé piny desky *Teensy 4.0* nemohou být použity pro všechny různé funkce. Funkce, které podporují jednotlivé piny lze vyčíst ze schématu na obr. 29. *Teensy 4.0* disponuje piny s následujícími funkcemi:

- 40 pinů pro digitální vstupy a výstupy
- 14 pinů pro analogové vstupy
- 31 pinů podporuje PWM[19]
- Připojení sedmi zařízení pro komunikaci přes sériový port (piny značené RX, TX)
- Připojení tří SPI sběrnic (piny značené MISO, MOSI, SCK, CS)
- Připojení tří sběrnic CAN (piny značené CRX, CTX)
- Připojení tří I2C sběrnic (piny značené SCL, SDA)

[19] PWM – *Pulse Width Modulation* – pulzně šířková modulace



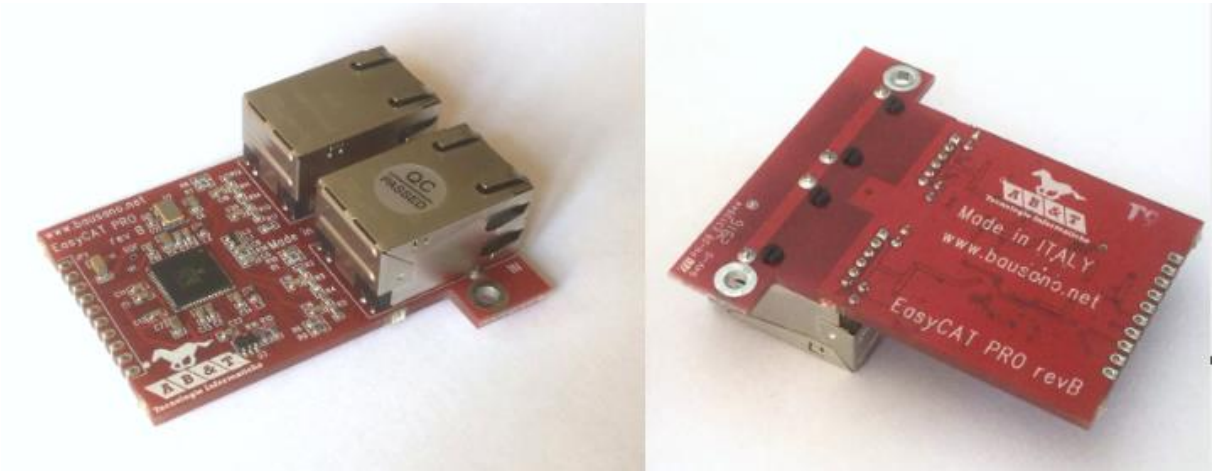
Obrázek 29 - Schéma pinů na přední straně (vlevo) a na zadní straně (vpravo) desky Teensy 4.0 [32]

4.1.1.1 Software

Programování desky se uskutečňuje ve vývojovém prostředí Arduino IDE s addonem Teensyduino. Nejdříve je třeba nainstalovat program Arduino a poté Teensyduino, což je instalační soubor, který přidá podporu pro *Teensy* do prostředí Arduino IDE.

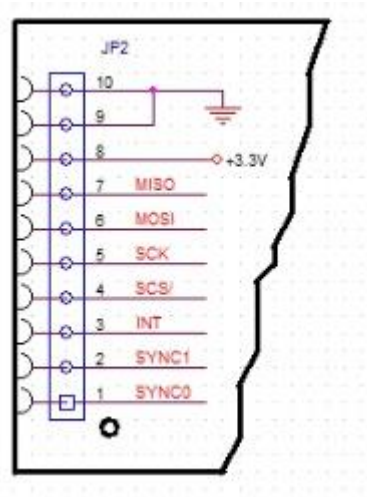
4.1.2 Modul EasyCAT PRO

EasyCAT PRO (obr. 30) je modul vytvořený italskou společností Bausano podporující propojení EtherCAT sběrnice s mikrokontrolérem (v našem případě *Teensy*) přes sběrnici SPI. Pomocí tohoto modulu a *Teensy* se vytvoří EtherCAT slave.



Obrázek 30 - Modul EasyCAT PRO [33]

Modul *EasyCAT PRO* je nakonfigurován na 32 bytů vstupních a 32 bytů výstupních dat pro komunikaci s masterem na EtherCAT sběrnici. Tyto hodnoty lze nakonfigurovat až na 128 bytů. Schéma modulu je znázorněno na obr. 31.



Obrázek 31 - Schéma pinů modulu EasyCAT PRO [33]

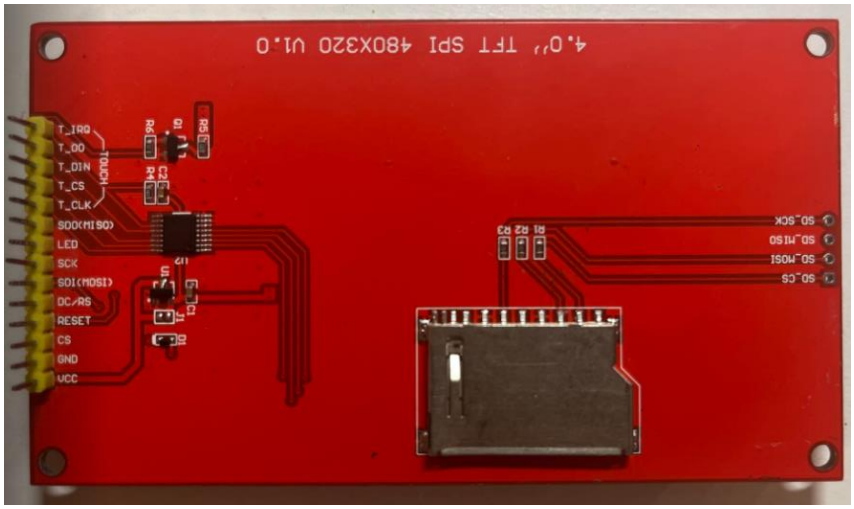
4.1.3 Dotykový displej

K implementaci HMI je zvolen dotykový displej značky *LaskaKit* (obr. 32). Tento displej má následující parametry [34]:

- Velikost 4“
- Rozlišení 480x320 bodů
- 65 tisíc barev
- SPI datové rozhraní
- Odporový dotykový displej s SPI datovým rozraním
- Pracovní napětí 3,3-5V
- Pro zobrazování je v displeji použit driver lLI9486, pro dotykovou část displeje je použit driver XPT2046

Obrázek 32 - Čtyřpalcový dotykový displej LaskaKit [34]

Popis pinů lze nalézt na zadní straně displeje (obr. 33).



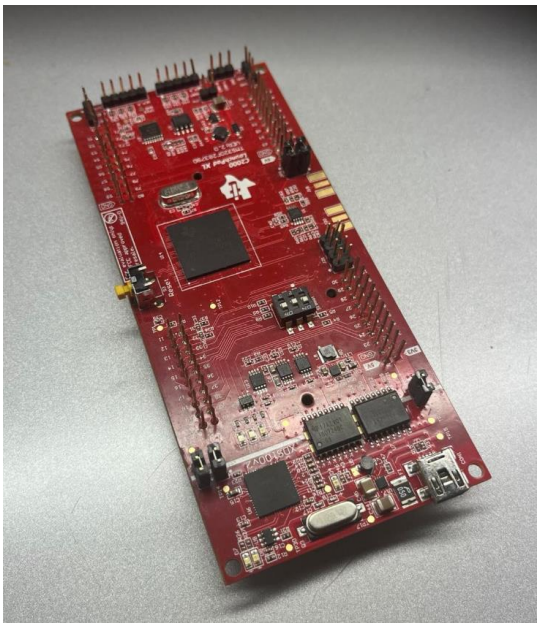
Obrázek 33 - Zadní strana dotykového displeje s popisem pinů

4.1.4 TI Delfino Launchpad

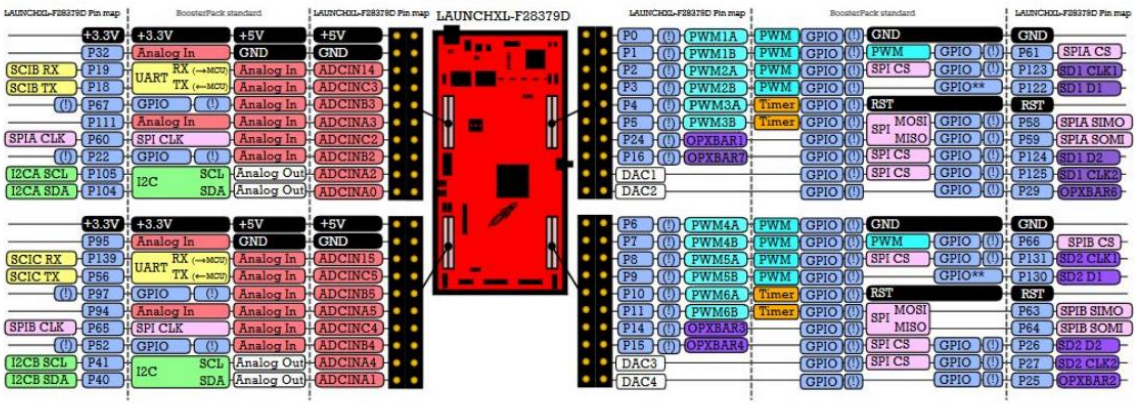
Pro řízení motorů robota byla ponechána již implementovaná platforma *C2000 Delfino Launchpad F28379D* (dále jen *F28379D*) s procesorem *TMS320D28069M* od společnosti Texas Instruments. Nejdůležitější vlastnosti platformy *F28379D* pro stávající projekt jsou:

- Rychlost procesoru 200 MHz
- Flash paměť 1024 kB
- Paměť RAM 204 kB
- Možnost připojení až tří sensorů pohybu (encoderů)
- Podpora připojitelných modulů (BoosterPacks)
- Softwarová podpora v Simulinku
- Piny pro PWM

Důležitým aspektem je možnost připojení inkrementálních kvadraturních optických encoderů. Díky polohovým datům z encoderů jsou pomocí PWM signálů řízeny otáčky motorů robota. Pro PWM signály je v Simulinku k dispozici softwarová podpora, která poskytuje blok *ePWM*. Pro čtení signálů z encoderů je v Simulinku k dispozici blok *eQEP*. Ukázka platformy *F28379D* je na obr. 34 a schéma možnosti zapojení pinů je znázorněno na obr. 35. [35]



Obrázek 34 - LaunchPad F28379D



Obrázek 35 - Schéma možností zapojení pinů F28379D [36]

4.1.5 Výkonový modul

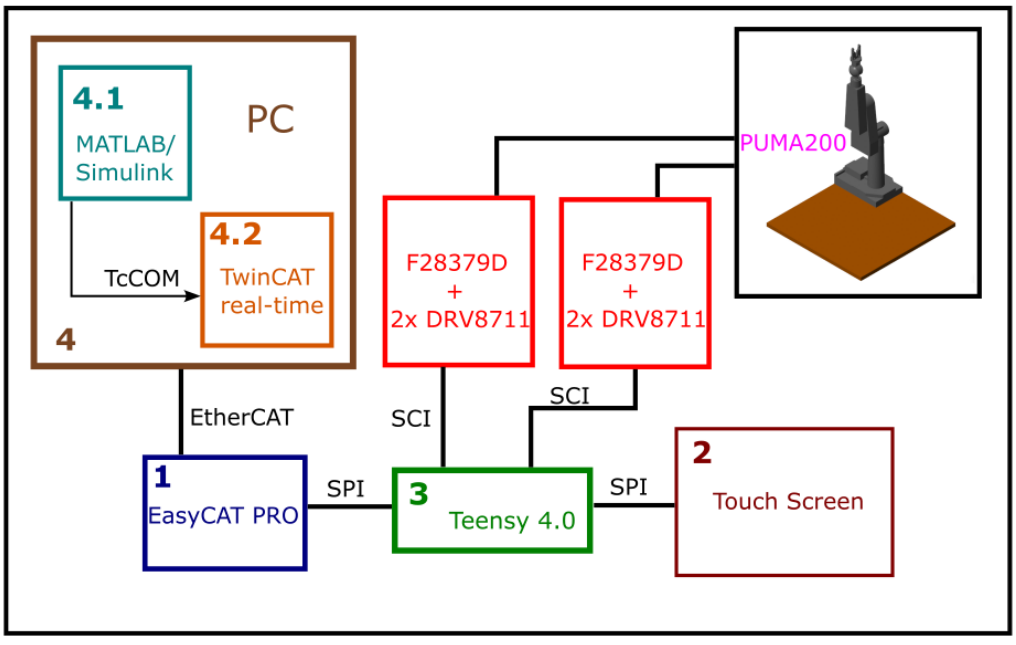
Stejně jako u platformy *F28379D* i výkonový modul pro ovládání proudu do motorů byl ponechán ve stávajícím projektu. Jedná se o modul *Boosterpack DRV8711* (dále jen *DRV8711*) k vidění na obr. 36 . Rozsah napájecího napětí u tohoto modulu je 8-52V a trvalý proud 4,5A. Modul *DRV8711* je plně kompatibilní s platformou *F28379D*. Primárně je modul určen pro krokový motor, lze ho ale překonfigurovat na dva *H-můstky* pro řízení DC motorů. Pro jeho nastavení je potřeba přepsat registry přes sběrnici SPI. Důležitým aspektem je, že tento výkonový modul je schopen řídit pouze dva motory, resp. brzdy motorů, a je tedy zapotřebí dvou těchto modulů. Na platformu *F28379D* je možnost připojení dvou těchto *DRV8711* modulů. Celkově tedy budou použity dvě platformy *F28379D* a čtyři moduly *DRV8711*. [35]

Samotné elektrické zapojení a řízení motorů pomocí platformy *F28379D* a modulů *DRV8711* již dále nebude komentováno, jelikož to není předmětem této bakalářské práce.

Obrázek 36 - Výkonový modul DRV8711 [37]

4.2 Systémový návrh

Na obr. 37 je vyobrazen nový systémový návrh k řízení robota PUMA200. Jednotlivé hlavní bloky systémového návrhu jsou označeny čísly. V následujících částech této práce budou funkce těchto bloků popsány. Cílem bylo řízení robota pomocí dotykového displeje. Displej komunikuje s *Teensy* pomocí sběrnice SPI. Data z displeje jsou posílány do *Teensy*, kde se zpracují a odešlou se skrz SPI do modulu *EasyCAT PRO*. *EasyCAT PRO* dále komunikuje přes sběrnici EtherCAT s masterem v PC – TwinCATem. Kombinací TwinCATu a programu MATLAB/Simulink se data zaslána modulem *EasyCAT PRO* zpracují a výstupem z TwinCATu budou natočení jednotlivých os robota, respektive pomocí převodovek budou výstupy z TwinCATu ve formě počtu pulzů, které se zašlou zpět pomocí sběrnice EtherCAT do modulu *EasyCAT PRO* a přes sběrnici SPI opět do *Teensy*. *Teensy* poté data pošle do platforem *F28379D* skrz sériový port (SCI), kde je již implementován systém řídící motory robota.

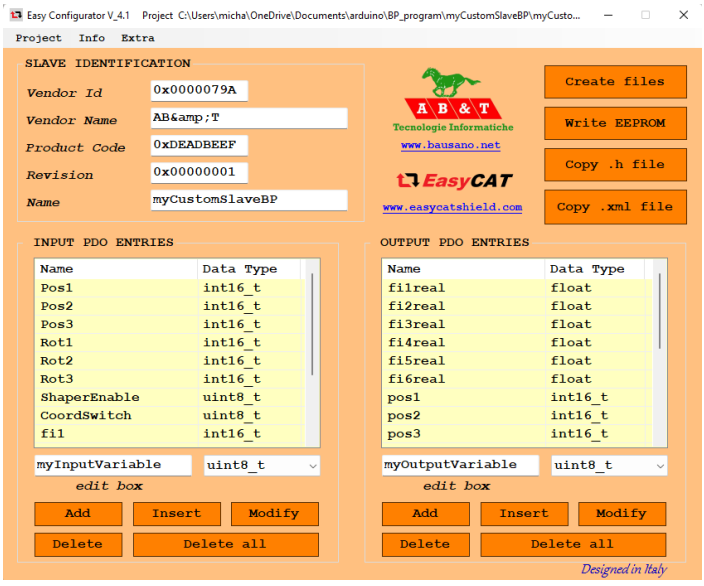


Obrázek 37 - Systémový návrh k řízení robota PUMA200

4.2.1 Konfigurace modulu EasyCAT PRO

Prvním blokem systémového návrhu je blok na obr. 37 označený číslem 1. Aby správně fungovala komunikace mezi slave zařízením a master zařízením, je potřeba nakonfigurovat slave zařízení (*EasyCAT PRO*). Konfigurace modulu *EasyCAT PRO* se uskutečňuje za pomocí konfigurátoru, který je poskytnut k modulu na stránkách firmy Bausano [33]. Konfigurátor je zobrazen na obr. 38. První je potřeba kliknout na ikonu *Project* a poté na ikonu *New*. Tím se nám vytvoří nový projekt. Konfigurátor je rozdělen na část vstupních dat (levá část obr. 38) a část výstupních dat (pravá část obr. 38). Zde se přidají (pomocí ikony *Add*) jednotlivá vstupní a výstupní data a jejich datové typy, která budou posílána mezi masterem a modulem *EasyCAT PRO*. Poté je potřeba projekt uložit. Dále je zapotřebí připojit modul k PC za pomocí ethernetového kabelu s konektorem RJ-45. V konfigurátoru se po připojení modulu k PC klikne na ikonu *Create files*. Tímto se nám ve složce uloženého projektu vygenerují následující soubory:

- *EtherCAT Slave Information* (ESI) soubor s příponou .xml – tento soubor musí mít každé slave zařízení. Soubor XML jednoduše popisuje konfiguraci daného slave zařízení a je potřeba ho poskytnout masterovi, aby věděl, jak používat zdroje dostupné na slave zařízení. Vygenerovaný XML soubor se musí vložit do speciální složky ve TwinCATu, konkrétně do složky *C:\TwinCAT\3.1\Config\Io\EtherCAT*.
- *Slave Information Interface* (SSI) soubor s příponou .bin – základní informace obsažené v XML souboru jsou také popsány v binárním formátu do paměti EEPROM slave zařízení. Při stisknutí ikonky *Write EEPROM* v konfigurátoru se nám otevře průzkumník souborů a bude po nás vyžadováno zvolit daný vygenerovaný BIN soubor. Po zvolení daného BIN souboru se nám nakonfiguruje modul *EasyCAT PRO*.
- *Header file .h* – hlavičkový soubor, který je potřeba vložit do složky, kde máme vytvořený program pro *Teensy* v Arduino IDE. Tento soubor obsahuje informace o proměnných (vstupech a výstupech), které se budou posílat skrz SPI mezi *Teensy* a *EasyCAT PRO*.



Obrázek 38 - Konfigurátor pro EasyCAT PRO

Pro komunikaci modulu EasyCAT PRO s deskou *Teensy* pomocí SPI je na stránkách firmy Bausano k dispozici také knihovna *EasyCAT_V2_0* [33], která musí být přidána do složky *arduino/libraries*.

4.2.2 Struktura dotykového displeje

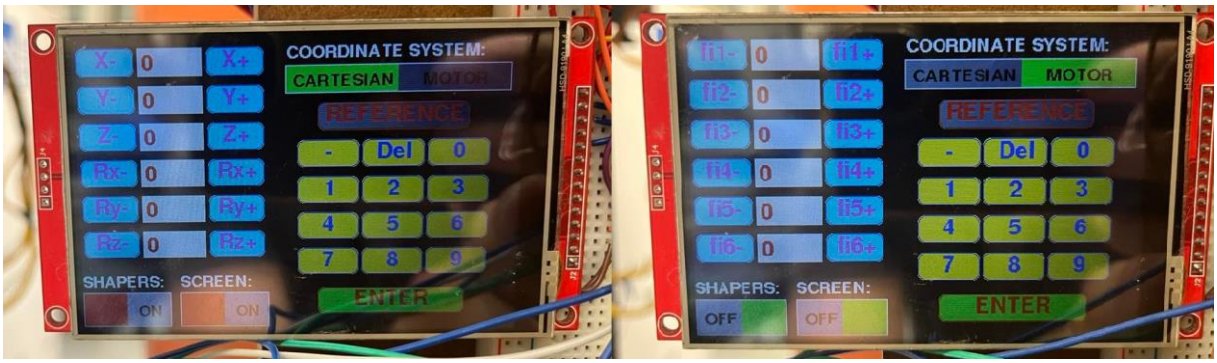
Na obr. 39 lze vidět počáteční obrazovka po připojení displeje k napájení a k *Teensy*. Na levé části displeje můžeme vidět tlačítka ovládající polohu robota v prostoru. Tlačítka *X-*, *X+*, *Y-*, *Y+*, *Z-*, *Z+* určují polohu robota vzhledem k počáteční pozici koncového efektoru a tlačítka *Rx-*, *Rx+*, *Ry-*, *Ry+*, *Rz-*, *Rz+* určují orientaci koncového efektoru. Aktuální hodnoty souřadnic lze vidět v prostředním sloupečku v bílých rámečcích mezi tlačítky. Při stisknutí tlačítek se začne inkrementovat, resp. dekrementovat daná hodnota. V levé dolní části displeje jsou dva vypínače:

- *SHAPERS* – tlačítko, které nese informaci o zapnutí/vypnutí *shaperů*, tedy při stisknutí na hodnotu *ON* se data v Simulinku pustí přes bloky *shaperů* do převodovek a výstupů. Podrobněji budou *shapery* popsány v kapitole MATLAB/Simulink.
- *SCREEN* – tlačítko, které vypíná a zapíná interakci s displejem. Pokud je tlačítko vypnuté, nelze měnit jakékoli hodnoty na displeji. Aby mohl uživatel mačkat tlačítka a měnit hodnoty, je potřeba mít oba vypínače, tedy *SHAPERS* a *SCREEN*, v pozici *ON*.

V pravé horní části můžeme vidět přepínání mezi kartézskými a motorovými souřadnicemi. Při přepnutí na motorové souřadnice (obr. 39) se nám místo kartézských souřadnic v levé části obrazovky objeví tlačítka *fi1-*, *fi1+*, *fi2-*, *fi2+*, *fi3-*, *fi3+*, *fi4-*, *fi4+*, *fi5*, *fi5+*, *fi6-*, *fi6+*, která ovládají přímo natočení (ve stupních) jednotlivých os robota.

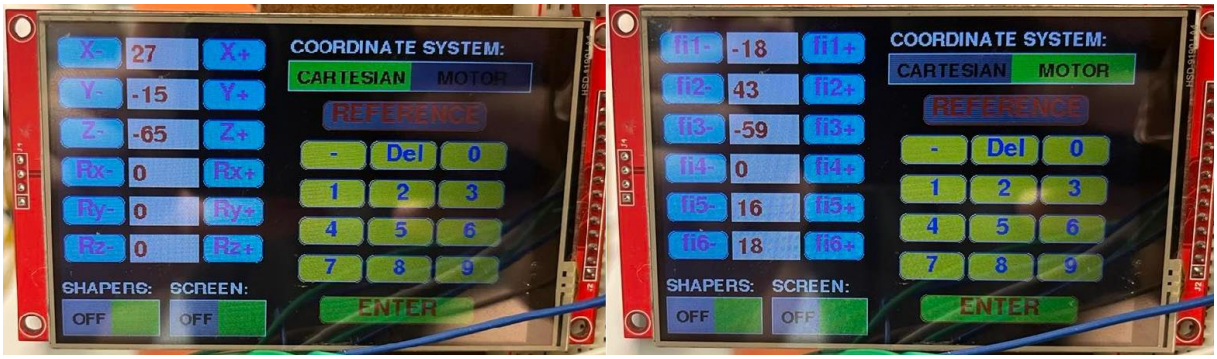
Pod tlačítky pro zvolení souřadnicového systému je tlačítko *REFERENCE*, které po stisknutí vynuluje všechny hodnoty a tím se robot dostane do počáteční pozice.

V pravé části můžeme dále vidět klávesnici s čísly. Pokud by uživatel chtěl hodnoty v bílých rámečcích zadat manuálně, aniž by musel držet tlačítka a čekat, než se robot dostane na danou hodnotu, může jednoduše klikat do bílých rámečků a měnit hodnoty pomocí klávesnice. Ve finále potvrdí zadané souřadnice tlačítkem *ENTER* v pravé dolní části.



Obrázek 39 - Displej s kartézskými souřadnicemi (vlevo) a motorovými souřadnicemi (vpravo)

Další vlastností displeje je taková, že při přejetí robota na určitou pozici a přepnutí na druhý souřadnicový systém má robot zůstat na místě, tedy při přepínaní se hodnoty v bílých rámečcích musí neustále přepočítávat přímou kinematikou z motorových souřadnic do kartézských souřadnic a inverzní kinematikou z kartézských souřadnic do motorových souřadnic. Ukázka je na obr. 40.



Obrázek 40 - Přepočet mezi souřadnicovými systémy na displeji

4.2.2.1 Knihovna pro komunikaci displeje s Teensy

Jedním z problému bylo nalezení vhodné knihovny pro komunikaci displeje s *Teensy* pomocí SPI. Knihovna, která byla prodejcem dodána bohužel fungovala pouze pro desky *Arduino* a nebylo možné ji použít pro *Teensy*. Na internetu bylo sice k nalezení mnoho knihoven pro driver ILI9486, kterým disponuje tento displej, avšak jeden z dalších problémů, který nastal bylo, že dané knihovny byly implementovány pro displej v módu 16 bitů, tedy že každá barva displeje je realizována pomocí 16 bitů. I když v technickém listu driveru ILI9486 byla popsána možnost konfigurace displeje na 16 a 18 bitů, knihovny pro 16 bitů pro tento displej nefungovaly. Nakonec se podařilo nalézt kompatibilní knihovnu na *GitHubu*[20] s názvem *TFT_eSPI* od uživatele *Bodmer* [38]. Na fóru *GitHub* bylo později zjištěno, že displej opravdu pravděpodobně nefunguje v 16bitovém módu. [39]

Po stažení knihovny *TFT_eSPI* je nejdříve nutné ji vložit do v kořenové složce *arduino* do složky *libraries*. Poté v knihovně otevřeme program *Touch_calibrate.ino*, který nalezneme ve složce *TFT_eSPI\examples\Generic*. Tento program po nás bude vyžadovat dotyky na rozích displeje. Poté nám vygeneruje kalibrační data, která budou v programu pro *Teensy* použita. Jelikož je knihovna *TFT_eSPI* univerzální pro spousta driverů, je dále potřeba v hlavičkovém souboru *User_Setup.h* nastavit typ driveru, piny použité pro SPI, frekvenci SPI renderování barev na displeji a frekvenci pro SPI komunikaci při použití dotyku. Nastavení v souboru *User_Setup.h* lze vidět na obr. 41, obr. 42 a obr. 43.

[20] GitHub – webová služba podporující vývoj software za pomocí verzovacího nástroje Git.

```
// Only define one driver, the other ones must be commented out
//#define ILI9341_DRIVER       // Generic driver for common displays
//#define ILI9341_2_DRIVER     // Alternative ILI9341 driver, see https://github.com/Bodmer/TFT_eSPI/issues/1172
//#define ST7735_DRIVER      // Define additional parameters below for this display
//#define ILI9163_DRIVER     // Define additional parameters below for this display
//#define S6D02A1_DRIVER
//#define RPI_ILI9486_DRIVER // 20MHz maximum SPI
//#define HX8357D_DRIVER
//#define ILI9481_DRIVER
 #define ILI9486_DRIVER
//#define ILI9488_DRIVER     // WARNING: Do not connect ILI9488 display SDO to MISO if other devices share the SPI bus (TFT SDO does NOT tristate when CS is high)
//#define ST7789_DRIVER      // Full configuration option, define additional parameters below for this display
//#define ST7789_2_DRIVER    // Minimal configuration option, define additional parameters below for this display
//#define R61581_DRIVER
//#define RM68140_DRIVER
//#define ST7796_DRIVER
//#define SSD1351_DRIVER
//#define SSD1963_480_DRIVER
//#define SSD1963_800_DRIVER
//#define SSD1963_800ALT_DRIVER
//#define ILI9225_DRIVER
//#define GC9A01_DRIVER
```

Obrázek 41 - Nastavení driveru ILI9486

```
#define TFT_MISO 12
#define TFT_MOSI 11
#define TFT_SCLK 13
#define TFT_CS   19  // Chip select control pin
#define TFT_DC    4  // Data Command control pin
#define TFT_RST   3  // Reset pin (could connect to RST pin)
//#define TFT_RST  -1  // Set TFT_RST to -1 if display RESET is connected to ESP32 board RST

// For ESP32 Dev board (only tested with GC9A01 display)
// The hardware SPI can be mapped to any pins

//#define TFT_MOSI 15 // In some display driver board, it might be written as "SDA" and so on.
//#define TFT_SCLK 14
//#define TFT_CS   5  // Chip select control pin
//#define TFT_DC   27  // Data Command control pin
//#define TFT_RST  33  // Reset pin (could connect to Arduino RESET pin)
//#define TFT_BL   22  // LED back-light

#define TOUCH_CS 18     // Chip select pin (T_CS) of touch screen

//#define TFT_WR 22    // Write strobe for modified Raspberry Pi TFT only
```

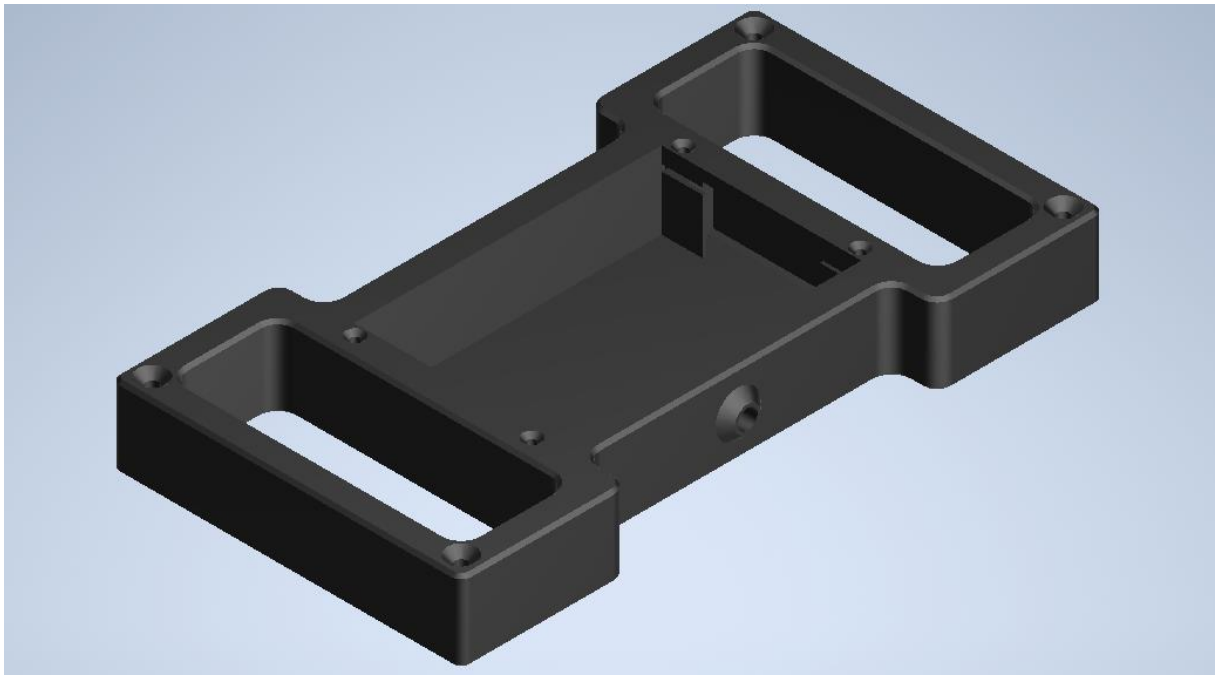
Obrázek 42 - Nastavení pinů pro SPI komunikaci s displejem

```
//#define SPI_FREQUENCY   1000000
// #define SPI_FREQUENCY   5000000
 //#define SPI_FREQUENCY  10000000
  #define SPI_FREQUENCY  20000000
//#define SPI_FREQUENCY  8000000
//#define SPI_FREQUENCY  27000000
// #define SPI_FREQUENCY  40000000
// #define SPI_FREQUENCY  55000000 // STM32 SPI1 only (SPI2 maximum is 27MHz)
// #define SPI_FREQUENCY  80000000

// Optional reduced SPI frequency for reading TFT
#define SPI_READ_FREQUENCY  20000000

// The XPT2046 requires a lower SPI clock rate of 2.5MHz so we define that here:
#define SPI_TOUCH_FREQUENCY  2500000
```

Obrázek 43 - Nastavení frekvence SPI pro renderování obrazu a frekvence SPI pro dotyk

4.2.2.2 Držák displeje

Jeden z designových prvků pro ovládání robota je návrh držáku pro displej. V programu Autodesk Inventor byl navrhnut model držáku a poté vytisknut na 3D tiskárně. Návrh v Inventoru lze vidět na obr. 44 a výsledný výtisk na obr. 45.



Obrázek 44 - Návrh držáku displeje v programu Autodesk Inventor

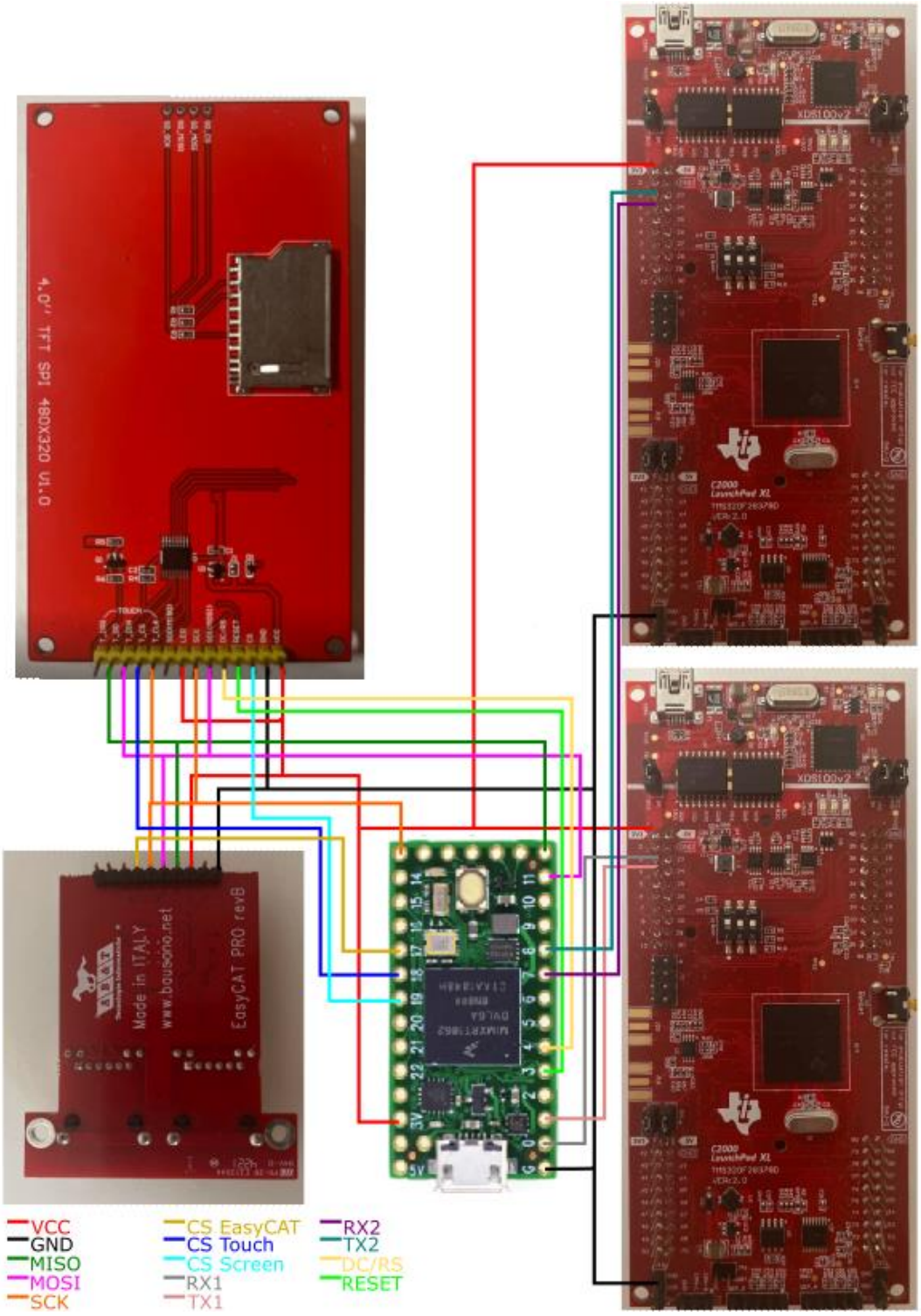
Obrázek 45 - Hotový výtisk držáku displeje

4.2.3 Řízení pomocí Teensy 4.0

Jádro celého systémového návrhu tvoří blok číslo 3 na obr. 37 týkající se desky *Teensy*. V následujících podkapitolách bude popsáno schéma zapojení komponent s *Teensy*, způsob komunikace a program pro *Teensy* v Arduino IDE.

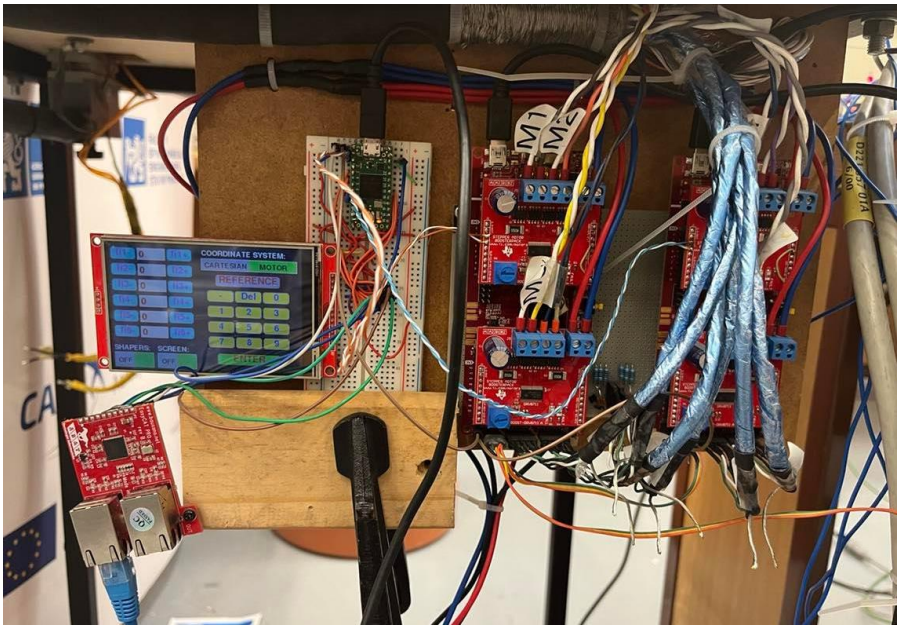
4.2.3.1 Schéma zapojení komponent k Teensy

Na obr. 46 je zobrazeno schéma zapojení jednotlivých komponent k *Teensy*.



Obrázek 46 - Schéma zapojení jednotlivých komponent k Teensy

Na obr. 47 lze vidět reálné zapojení komponent k *Teensy* v laboratoři.



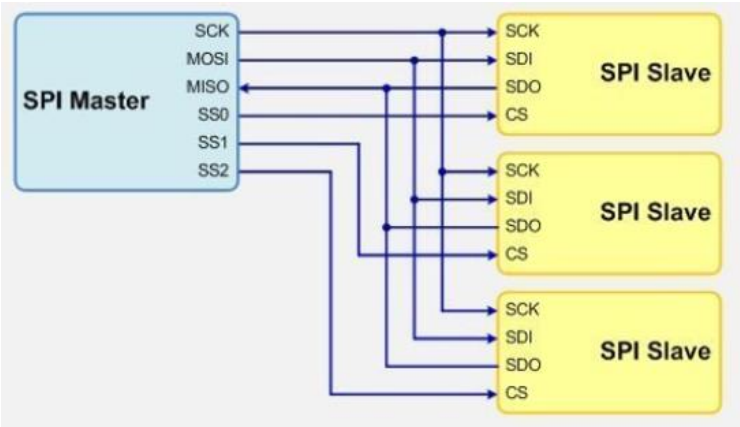
Obrázek 47 - Reálné zapojení všech komponent k Teensy v laboratoři

4.2.3.2 Komunikace pomocí SPI

Serial Peripheral Interface (SPI) je jedna z forem sériových sběrnic sloužící pro propojení dvou či více uzlů. Sběrnice SPI se používá především pro komunikaci s pamětí typu *EEPROM*, senzory, textovými a grafickými LCD panely, A/D a D/A převodníky, hodinami reálného času (RTC) apod. Jak je vidět na obr. 48, standardní SPI sběrnice obsahuje jedno master zařízení připojené k několika slave zařízení pomocí 4 linek [40]:

- *Serial Clock* (SCK) – linka pro synchronizaci hodin mezi masterem a slavem
- *Master Out Slave In* (MISO) – linka pro posílání dat od slave zařízení k masterovi
- *Master In Slave Out* (MOSI) – linka pro posílání dat od mastera k slave zařízení
- *Slave Select* (SS) nebo také *Chip Select* (CS) – linka určující, které zařízení má komunikovat s masterem.

Linky SCK, MISO a MOSI jsou sdílené jednotlivými slave zařízeními, zatímco každý slave má unikátní CS linku.

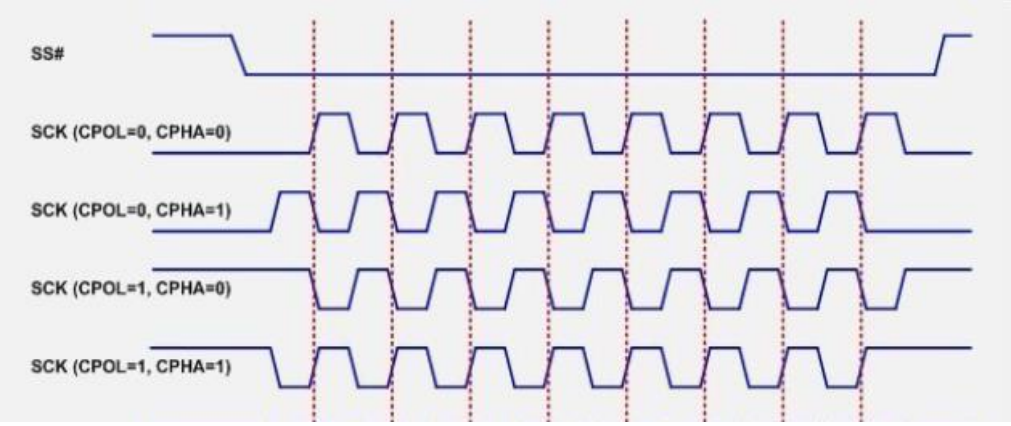


Obrázek 48 - Slave zařízení připojená k masterovi na SPI sběrnici pomocí čtyř linek [40]

Módy SPI sběrnice:

Sběrnici SPI je možno nakonfigurovat na čtyři různé módy (obr. 49) pomocí volby polarity hodin (*Clock Polarity* – CPOL) a vzorkování dat při sestupné či vzestupné hraně (*Clock Phase* – CPHA) hodinového signálu. Jednotlivé Módy jsou následující [40]:

- Mode 0 - CPOL a CPHA jsou „0“. Znamená to, že hodiny jsou ve výchozím nastavení v logické „0“ a data se začínají vzorkovat při každé následující vzestupné hraně hodinového signálu.
- Mode 1 – CPOL je „0“ a CPHA je „1“. Znamená to, že hodiny jsou ve výchozím nastavení v logické „0“ a data se začínají vzorkovat každou sestupnou hranu hodinového signálu.
- Mode 2 – CPOL je „1“ a CPHA je „0“. Znamená to, že hodiny jsou ve výchozím nastavení rovny logické „1“ a vzorkování dat probíhá každou sestupnou hranu hodinového signálu.
- Mode 3 – CPOL je „1“ a CPHA je „1“. Znamená to, že hodiny jsou ve výchozím nastavení v logické „1“ a data se vzorkují každou vzestupnou hranu.

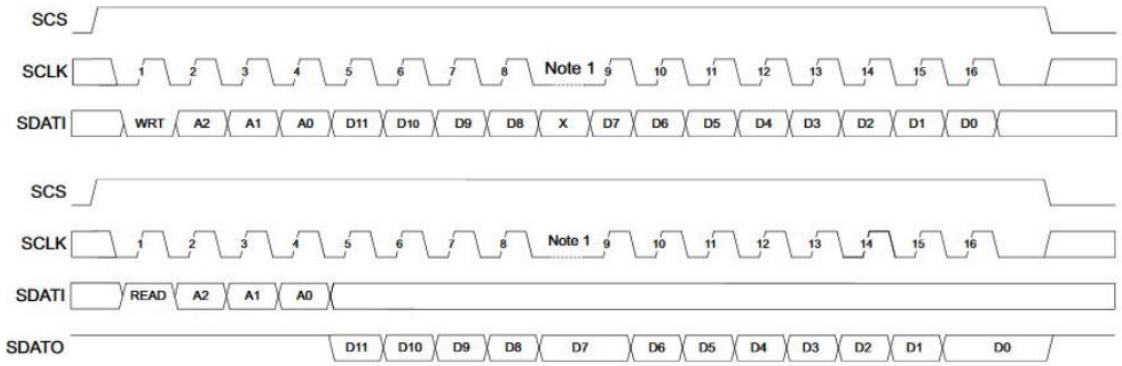


Obrázek 49 - Módy SPI sběrnice

Princip komunikace skrz SPI:

Pokud chce master komunikovat s daným slave zařízením, zašle z příslušného SS pinu na CS linku logickou „0“. Na lince SCK začne pro synchronizaci zasílání zpráv běžet hodinový signál určité frekvence – v případě komponent v této práci je zvolena hodinová frekvence 20 MHz pro renderování obrazu na displeji, 2,5 MHz pro dotyková data na displeji a 8,5 MHz pro komunikaci mezi *EasyCAT PRO* a *Teensy*. Dále, dle zvoleného módu (obvykle Mód 0), začne slave zařízení vzorkovat data na lince MOSI. První bity na MOSI (SDATI) lince jsou instrukční a určují druh komunikace – zápis nebo čtení a adresu (například registru), na kterou se mají data zapsat. Pokud se jedná o zápis dat na slave zařízení, probíhá zaslání zprávy o velikosti 8 nebo 16 bitů od mastera směrem ke slave zařízení dále pouze na lince MOSI. MISO (SDATO) linka je v tomto případě ignorována. Pokud první instrukční bity na MOSI lince indikují čtení dat od slave zařízení, začne slave na lince MISO zasílat datové pulzy k masterovi. Než se přečte daná zpráva od slave zařízení je ignorována linka MOSI. Po dokončení transakce zašle master na danou SS linku logickou „1“ a tím se přeruší

komunikace s daným slave zařízením. Příklad zápisu a čtení pomocí SPI je naznačen na obr. 50. [40]



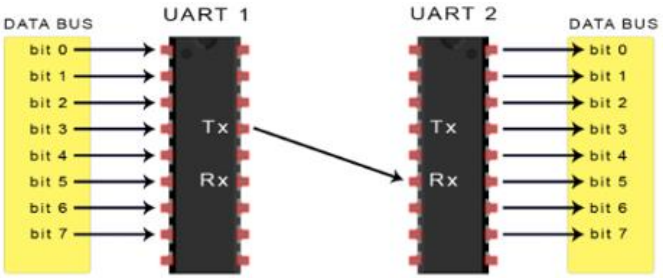
Obrázek 50 - Zápis (nahoře) a čtení (dole) dat na sběrnici SPI [35]

4.2.3.3 SCI/UART

Serial Communication Interface (SCI), také známý pod názvem *Universal asynchronous receiver-transmitter* (UART) je jedna z nejjednodušších plně-duplexních komunikací mezi dvěma zařízeními. Pro komunikaci je potřeba pouze dvou linek [41]:

- Linka pro odesílání dat – TX
- Linka pro přijímání dat – RX

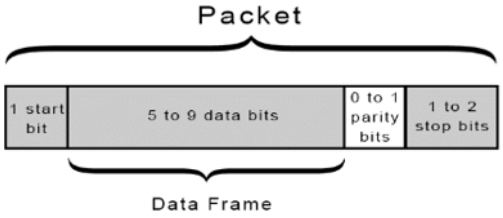
Linka TX konverguje paralelní data od kontrolního zařízení (CPU) na formu sériových dat (bity se posílají po lince TX za sebou), která se posílají na linku RX druhého zařízení, kde se zpátky konvergují sériová data na paralelní data obr. 51.



Obrázek 51 - Bity přicházející paralelně se pošlou sériově přes linku TX na linku RX, kde se zpět konvergují na paralelní data [41]

Datový paket UART

Data zasílána linkou TX jsou organizována do paketů (obr. 52). Každý paket obsahuje 1 *start* bit, 5 až 9 datových bitů, volitelný *parity* bit a jeden až dva *stop* bity.



Obrázek 52 - Datový paket poslaný po lince TX [41]

Start bit

Linka TX je obvykle v logické „1“, pokud nechce posílat data. Pro zaslání dat zašle UART na linku TX *start* bit (logickou „0“) a linka RX začne přijímat data. Data se přijímají na frekvenci zvanou *baud rate*. *Baud rate* musí být nastaven stejně u odesílacího i přijímacího zařízení. V případě komunikace mezi *Teensy* a platformami *F28379D* je baud rate nastaven na hodnotu 6,25 Mbps.

Datový rámec

Datový rámec obsahuje samotná užitečná data, která se posílají z linky RX na linku TX. Data jsou v délce 5 až 9 bitů, pokud se nepoužije *parity* bit a v délce 5-8 bitů při použití *parity* bitu. Linky musí být předem domluveny na délce zprávy, aby se datové bity nezaměňovaly s *parity* bitem či *stop* bitem.

Parity bit

Parity bit se používá pro kontrolu poškození dat. Pokud se *parity* bit hodná hodnotě „1“, je v datovém paketu celkově lichý počet bitů s hodnotou „1“. Pokud je *parity* bit nastaven na „0“, je v datovém paketu sudý počet bitů s hodnotou „1“. Přijímač si podle *parity* bitu spočítá počet „1“ v datové správě a pokud lichost, resp. sudost celkových „1“ sedí tak ví, že jsou data zaslána v pořádku. Před začátkem komunikace se podobně jako u datových bitů musí přijímač a odesílatel domluvit, zda použijí *parity* bit, aby se nezaměňoval se *stop* bitem.

Stop bit

Stop bit signalizuje konec datového paketu. Linka TX přejde z logické „0“ do logické „1“ po dobu jednoho nebo dvou taktů. [41]

4.2.3.4 Program v Arduino IDE

V této podkapitole bude popsána struktura programu pro *Teensy* v Arduino IDE.

Komunikace s displejem

Pro komunikaci Teensy s displejem skrz sběrnici SPI je použita, jak již bylo zmíněno, knihovna *TFT_eSPI*. Tato knihovna nám umožňuje základní funkce jako je barva pozadí obrazovky, kreslení obdélníků, kruhů, rámečků, linek, vypisování textových řetězců a čísel o daném fontu. Dále nám umožňuje indikovat, zda se uživatel dotknul obrazovky a v jakém místě obrazovky (souřadnice *x* a *y*). Další užitečnou funkcí knihovny je tvorba tlačítek. Knihovna nabízí jen základní funkce a pro plné využití této knihovny je potřeba naprogramovat i funkce vlastní, je tedy zřejmé, že není možné používat knihovnu bez určité znalosti programovacího jazyka C/C++

Vrchní část programu *Teensy* v prostředí Arduino IDE obsahuje proměnné a konstanty používané v programu. Na obr. 53 (vlevo) je příklad definovaných konstant, což jsou obvykle souřadnice daných obdélníků, nápisů či tlačítek na displeji. Dále jsou na obr. 53 (vpravo) definované pro nás důležité proměnné vztahující se k hodnotám souřadnic. V poli *Pos* budou uchovávána souřadnice *x*, *y*, *z* koncového efektoru a v poli *Rot* natočení

koncového efektoru *Rx, Ry, Rz*. V poli *MotorCoord* se uchovávají natočení os robota. Knihovna *TFT_eSPI* umí vypisovat pouze čísla typu *integer*, proto jsou i hodnoty daných souřadnic typu *integer*. Hodnoty přepínačů (*SHAPERS*, *SCREEN*, *COORDINATE SYSTEM*) jsou uchovávány v proměnných typu *boolean*. Další pole používaná v programu jsou například řetězce (pole *numberBuffer*). Řetězec znaků se vytváří, pokud chce uživatel zadat souřadnice manuálně. Řetězec se pak musí konvergovat na číslo typu *integer*.

```
// Keypad start position, key sizes and spacing
#define KEY_X 36 // Centre of key
#define KEY_Y 30
#define KEY_W 62 // Width and height
#define KEY_H 30
#define KEY_SPACING_X 80 // X and Y gap
#define KEY_SPACING_Y 20
#define KEY_TEXTSIZE 1   // Font size multiplier

// Using two fonts since numbers are nice when bold
#define LABEL1_FONT &FreeSansOblique12pt7b
#define LABEL2_FONT &FreeSansBold12pt7b
#define LABEL3_FONT &FreeSansBold9pt7b

// Numeric display box size and location
#define DISP_X 1
#define DISP_Y 10
#define DISP_W 70
#define DISP_H 30
#define DISP_TSIZE 3
#define DISP_TCOLOR TFT_CYAN


// Switch position and size
#define FRAME_X 1
#define FRAME_Y 280
#define FRAME_W 100
#define FRAME_H 40

#define FRAME_COORD_X 230
#define FRAME_COORD_Y 30
#define FRAME_COORD_W 240
```

```
int Pos[3] = {0, 0, 0}; //end efector of robot, init is 0,0,0
int Rot[3] = {0, 0, 0}; // orienteation angles, init is 0,0,0
int MotorCoord[6]={0, 0, 0, 0, 0, 0}; //motor coordinates

char numberBuffer[6][NUM_LEN + 1] = {"", "", "", "", "", ""}; //u
uint8_t numberIndex[6] = {0, 0, 0, 0, 0, 0}; //used to index posi
bool SwitchesOn[2] = {false, false}; //state of switches
bool CoordSwitch = false; // Choosed coordinate system
```

Obrázek 53 - Příklad definovaných konstant týkajících se velikosti a pozice tlačítek (vlevo) a důležité proměnné používané v programu (vpravo)

Jelikož je program velmi obsáhlý a komplexní budou jen stručně popsány některé naprogramované funkce a principy programu týkající se displeje.

V části *void setup*, se pomocí funkce *tft.init* inicializuje displej a vykreslí se úvodní obrazovka s výchozími hodnotami všech souřadnic (nastavené na hodnotu „0“). Oblast *loop* v programu funguje jako smyčka stále se opakujícího kódu. V každé smyčce program kontroluje, zda se nestisknula tlačítka. Jelikož pro interakci s displejem musí být tlačítka *SHAPERS* a *SCREEN* ve stavu *ON*, tak se nejdříve kontroluje funkcí *checkforswitches*, zda se tlačítka nestisknula a pokud ano, změní se jejich hodnota na *true*. Jakmile jsou obě tlačítka ve stavu *ON*, začne program indikovat stisknutí i ostatních tlačítek displeje. Ve výchozí pozici je zvolen kartézský souřadnicový systém. Program tedy kontroluje stisknutí tlačítek pro inkrementaci a dekrementaci souřadnic a pokud se tyto hodnoty mění, přepisuje hodnoty na displeji a zároveň je ukládá do proměnných. Kontrola stisknutí tlačítek pro inkrementaci a dekrementaci kartézských souřadnic je realizována pomocí funkce *checkforcartesian*. Pro kontrolu motorových souřadnic je identicky naprogramovaná funkce *checkformotor*. Část funkcí *checkforswitches* a *checkforcartesian* jsou znázorněny na obr. obr. 54 a obr. 55.

```
void checkforswitches()
{
  uint16_t t_x = 0, t_y = 0;
  bool pressed = tft.getTouch(&t_x, &t_y);

  //check if some switch was pressed
  if (pressed){
    for (uint8_t i = 0; i < 2; i++) {
      if (SwitchesOn[i]) {
        if ((t_x > REDBUTTON_X + i * (FRAME_W + 10)) && (t_x < (REDBUTTON_X + i * (FRAME_W + 10) + REDBUTTON_W))) {
          if ((t_y > REDBUTTON_Y) && (t_y <= (REDBUTTON_Y + REDBUTTON_H))) {
            redBtn(i);
          }
        }
      }
      else {
        if ((t_x > GREENBUTTON_X + i * (FRAME_W + 10)) && (t_x < (GREENBUTTON_X + i * (FRAME_W + 10) + GREENBUTTON_W))) {
          if ((t_y > GREENBUTTON_Y) && (t_y <= (GREENBUTTON_Y + GREENBUTTON_H))) {
            greenBtn(i);
          }
        }
      }
    }


    //check for coordinates system, but only if screen and shapers are enabled and coordinates are done setting manually
    if(SwitchesOn[0]==1 && SwitchesOn[1]==1 && !ManualSet)
    {
      if (CoordSwitch) {
        if ((t_x > CARTBUTTON_X) && (t_x < (CARTBUTTON_X + CARTBUTTON_W))) {
         if ((t_y > CARTBUTTON_Y) && (t_y <= (CARTBUTTON_Y + CARTBUTTON_H))) {
           CartBtn();
           DrawCartesianKeys();
           setvaluescartesian();
         }
        }
      }
      else {
        if ((t_x > MOTBUTTON_X) && (t_x < (MOTBUTTON_X + MOTBUTTON_W))) {
          if ((t_y > MOTBUTTON_Y) && (t_y <= (MOTBUTTON_Y + MOTBUTTON_H))) {
            MotBtn();
            DrawMotorKeys();
            setvaluesmotor();
          }
        }
```

Obrázek 54 - Ukázka části kódu funkce checkforswitches

```
 void checkstatecartesian()
 {
    for (uint8_t b = 0; b < 13; b++) {
       if (key[b].justReleased()) {
         key[b].drawButton();     // draw normal
       }
       if (key[b].isPressed()) {
         key[b].drawButton(true);  // draw invert
         tft.setTextDatum(TL_DATUM);        // Use top left corner as text coord datum
         tft.setFreeFont(&FreeSansBoldl2pt7b);  // Choose a nicefont that fits box
         tft.setTextColor(DISP_TCOLOR);
         //TConst=1;
         //EASYCAT.BufferIn.Cust.TConst = TConst;
         if (b == 0) {
           Pos[0]--;
           EASYCAT.BufferIn.Cust.Pos1 = Pos[0];
           tft.fillRect(DISP_X + DISP_W, DISP_Y + 0 * (DISP_H + 10), DISP_W, DISP_H, TFT_BLACK);
           tft.drawNumber(Pos[0], DISP_X + DISP_W + 4, DISP_Y + 6 + 0 * (DISP_H + 10));
         }

         if (b == 1) {
           Pos[0]++;
           EASYCAT.BufferIn.Cust.Pos1 = Pos[0];
           tft.fillRect(DISP_X + DISP_W, DISP_Y + 0 * (DISP_H + 10), DISP_W, DISP_H, TFT_BLACK);
           tft.drawNumber(Pos[0], DISP_X + DISP_W + 4, DISP_Y + 6 + 0 * (DISP_H + 10));
         }

         if (b == 2) {
           Pos[1]--;
           EASYCAT.BufferIn.Cust.Pos2 = Pos[1];
           tft.fillRect(DISP_X + DISP_W, DISP_Y + 1 * (DISP_H + 10), DISP_W, DISP_H, TFT_BLACK);
           tft.drawNumber(Pos[1], DISP_X + DISP_W + 4, DISP_Y + 6 + 1 * (DISP_H + 10));
         }
```

Obrázek 55 - Ukázka části kódu funkce checkforcartesian

Nejobtížnější zřejmě bylo naprogramovat funkci *checkmanualset*, která kontroluje, zdali uživatel nechce zadat hodnoty ručně. Funkce kontroluje, které souřadnice chce uživatel přepsat, ukládá si dané hodnoty do řetězců a poté je musí konvergovat z řetězců na čísla a vypsat na obrazovku. Ukázka kódu je na obr. 56.

```
void checkmanualset()
{
    //if you wanted to change the coordinates manually and you did not hit the enter button so far
    if (ManualSet && !key[25].justPressed()) {
        //clear square that you want to set the coordinate manually in
        if (ClearNumber) {
            tft.fillRect(DISP_X + DISP_W, DISP_Y + whichonechange * (DISP_H + 10), DISP_W, DISP_H, TFT_BLACK);
            ClearNumber = false;
        }
        for (uint8_t b = 13; b < 25; b++) {
            if (key[b].justReleased()) key[b].drawButton();
            if (key[b].justPressed()) {
                key[b].drawButton(true);  // draw invert
                tft.setTextDatum(TL_DATUM);        // Use top left corner as text coord datum
                tft.setFreeFont(&FreeSansBold12pt7b);  // Choose a nicefont that fits box
                tft.setTextColor(DISP_TCOLOR);
                //if delete button is hit
                if (b == 14) {
                    numberBuffer[whichonechange][numberIndex[whichonechange]] = 0;
                    if (numberIndex > 0) {
                        numberIndex[whichonechange]--;
                        numberBuffer[whichonechange][numberIndex[whichonechange]] = 0;
                    }
                    tft.fillRect(DISP_X + DISP_W, DISP_Y + whichonechange * (DISP_H + 10), DISP_W, DISP_H, TFT_BLACK);
                    tft.drawString(numberBuffer[whichonechange], DISP_X + DISP_W + 4, DISP_Y + 6 + whichonechange * (DISP_H + 10));
                }
                //if number button or minus button is hit, create strings of number and write it to the square
                if (b != 14) {
                    switch (whichonechange) {
                    case 0:
                        if (numberIndex[whichonechange] < NUM_LEN) {
                            numberBuffer[whichonechange][numberIndex[whichonechange]] = keyLabel[b][0];
                            numberIndex[whichonechange]++;
                            numberBuffer[whichonechange][numberIndex[whichonechange]] = 0;
                            tft.drawString(numberBuffer[whichonechange], DISP_X + DISP_W + 4, DISP_Y + 6 + whichonechange * (DISP_H + 10));
                        }
                        break;
                    case 1:
                        if (numberIndex[whichonechange] < NUM_LEN) {
                            numberBuffer[whichonechange][numberIndex[whichonechange]] = keyLabel[b][0];
                            numberIndex[whichonechange]++;
                            numberBuffer[whichonechange][numberIndex[whichonechange]] = 0;
                            tft.drawString(numberBuffer[whichonechange], DISP_X + DISP_W + 4, DISP_Y + 6 + whichonechange * (DISP_H + 10));
                        }
                        break;
                    case 2:
```

Obrázek 56 - Ukázka části kódu funkce checkmanualset

Program interagující s displejem obsahuje mnoho dalších proměnných a funkcí, které si lze prohlédnout v příloze. Program obsahuje také komentáře k jednotlivým funkcím a proměnným pro lepší orientaci.aa

Komunikace s modulem EasyCAT PRO

Pro komunikaci mezi *Teensy* a *EasyCAT PRO* je použita knihovna, která je plně přesunuta do hlavičkového souboru *EasyCAT.h*. V programu se nejdřív vytvoří instance typu *EasyCAT. Instance se vytvoří pomocí zápisu EasyCAT EASYCAT (17).* Číslo v závorce určuje číslo pinu CS na desce *Teensy*. Poté se již pracuje s nakonfigurovanými proměnnými v modulu *EasyCAT PRO*. Vstupní data do modulu se zapíšou z *Teensy* do proměnných značených *EASYCAT.BufferIn.Cust.XXX.* Poté se funkcí *EasyCAT.MainTask* pošlou vstupní data přes SPI k modulu. Tato data budou z modulu posílána masterovi (TwinCAT). Zároveň se z modulu ve výstupních proměnných značených *EASYCAT.BufferOut.Cust.XXX* pošlou do *Teensy* výstupní proměnné zaslané TwinCATem.

Ukázka uložení hodnot do vstupních proměnných v *Teensy* a později přes SPI zaslané do modulu *EasyCAT PRO* a TwinCATu je znázorněna na obr. 57. Na obr. 58 je příklad uložení výstupních hodnot z TwinCATu přes modul *EasyCAT PRO* a SPI do kartézských souřadnic v *Teensy*. Toho se využívá například při již zmiňovaném přepínání souřadnicových systémů, kdy se v TwinCATu spočítají dopřednou kinematikou kartézské souřadnice z natočení os a tyto souřadnice se zašlou do *Teensy*, aby se mohly vypsat na displej.

```
EASYCAT.BufferIn.Cust.Pos1 = Pos[0];
EASYCAT.BufferIn.Cust.Pos2 = Pos[1];
EASYCAT.BufferIn.Cust.Pos3 = Pos[2];
EASYCAT.BufferIn.Cust.Rot1 = Rot[0];
EASYCAT.BufferIn.Cust.Rot2 = Rot[1];
EASYCAT.BufferIn.Cust.Rot3 = Rot[2];
```

Obrázek 57 - Uložení kartézských souřadnic do vstupních proměnných modulu EasyCAT PRO

```
Pos[0] = EASYCAT.BufferOut.Cust.pos1;
Pos[1] = EASYCAT.BufferOut.Cust.pos2;
Pos[2] = EASYCAT.BufferOut.Cust.pos3;
Rot[0] = EASYCAT.BufferOut.Cust.rot1;
Rot[1] = EASYCAT.BufferOut.Cust.rot2;
Rot[2] = EASYCAT.BufferOut.Cust.rot3;
```

Obrázek 58 - Uložení výstupních proměnných z TwinCATu přes modul EasyCAT PRO do proměnných v Teensy

V programu si lze všimnout toho, že zde figuruje další proměnná nakonfigurována v modulu *EasyCAT PRO* a tou je proměnná *State*. Tato proměnná je periodicky zasílána z TwinCATu do *Teensy* a obsahuje hodnotu „1“, pokud je bezchybná komunikace mezi *Teensy* a TwinCATem a hodnotu „0“ pokud komunikace z nějakého důvodu neprobíhá. Proměnná *State* tedy funguje jako kontrolní proměnná, která dá na straně platforem *F28379D* informaci o poruše komunikace a příkaz k zastavení robota. Na začátku každé smyčky *loop* je kontrolována hodnota proměnné *State* a při zaznamenání poruchy se hodnota vypínačů nastaví do stavu *OFF*, zároveň se do výstupů poslaných na platformy *F28379D* pošlou poslední validní hodnoty pulzů před přerušením, uložených v proměnných *I1-I6*. Tímto způsobem zůstane robot při poruše na místě. Ukázka kontroly stavu na EtherCAT sběrnici je zobrazena obr. 59. Formát posílaných dat do platforem *F28379D* pomocí SCI je popsán v následujícím odstavci.

```
void loop(void) {

    if (State<1)
    {
    // turn of switches if ethercat collapses
    SwitchesOn[0]=0;
    SwitchesOn[1]=0;

    // sending values of last valid states of motors via SCI
    Serial1.write("S"); //start bit so the receiver (f28379d) will not read from the middle of byte sending

    Serial1.write(S[0]); //Sending State of ethercat bus
    Serial1.write(S[1]);
    Serial1.write(S[2]);
    Serial1.write(S[3]);

    Serial1.write(I1[0]); //sending first float - last valid state of motor 1
    Serial1.write(I1[1]);
    Serial1.write(I1[2]);
    Serial1.write(I1[3]);

    Serial1.write(I2[0]); //sending second float - last valid state of motor 2
    Serial1.write(I2[1]);
    Serial1.write(I2[2]);
    Serial1.write(I2[3]);

    Serial1.write(I3[0]); //sending third float - last valid state of motor 3
    Serial1.write(I3[1]);
    Serial1.write(I3[2]);
    Serial1.write(I3[3]);
```

Obrázek 59 - Ukázka kódu kontrolující stav na EtherCAT sběrnici

Komunikace s platformami F28379D

Pro zasílání hodnot do platforem *F28379D* je použit sériový port (SCI). Původně bylo plánováno zasílání dat pomocí sběrnice SPI. Na straně přijímače se ale data zasílaná skrz SPI často četla uprostřed toku zasílaných dat, a tedy v nesmyslném pořadí. Druhý důvod pro nepoužití SPI sběrnice byl fakt, že nakonec ani nemohla být použita z důvodu obsazených pinů pro SPI komunikaci na straně platforem F28379D výkonovými moduly DRV8711.

Pro zasílání hodnot pomocí SCI je již v Arduinu IDE k dispozici knihovna. Ve funkci void__setup se nejdříve příkazy *Serial1.Begin(6250000)* a *Serial2.begin(6250000)* nastaví *baud rate* pro komunikaci s první a druhou platformou *F28379D*. Poté se příkazem *Serial.write* posílají data do platforem. Data se posílají z TwinCATu do *Teensy* a obsahují počty pulzů, které mají být na encoderech. Počty pulzů se v platformách *F28379D* přijímají datovém typu *single* (*float*). Knihovna v prostředí Arduino IDE umožňuje zasílání pouze jednotlivých bytů, a tak tedy musela být hodnota float rozdělena na 4 byty, které se postupně poslaly přes SCI do platforem. Takto se za sebou poslalo celkem 32 bytů – jedna kontrolní proměnná a 3 *float* hodnoty pro první tři osy robota do první platformy a podobně pro druhé tři osy do druhé platformy. Sériová komunikace začíná startovacím bytem značeným „S" a ukončovacím bytem značeným „E". Tyto byty zajišťují, že se na straně přijímače (*F28379D*) přečtou hodnoty ve správném pořadí. Ukázka konverze *float* hodnoty na jednotlivé byty a poslání přes SCI pro první tři osy je vidět na obr. 60.

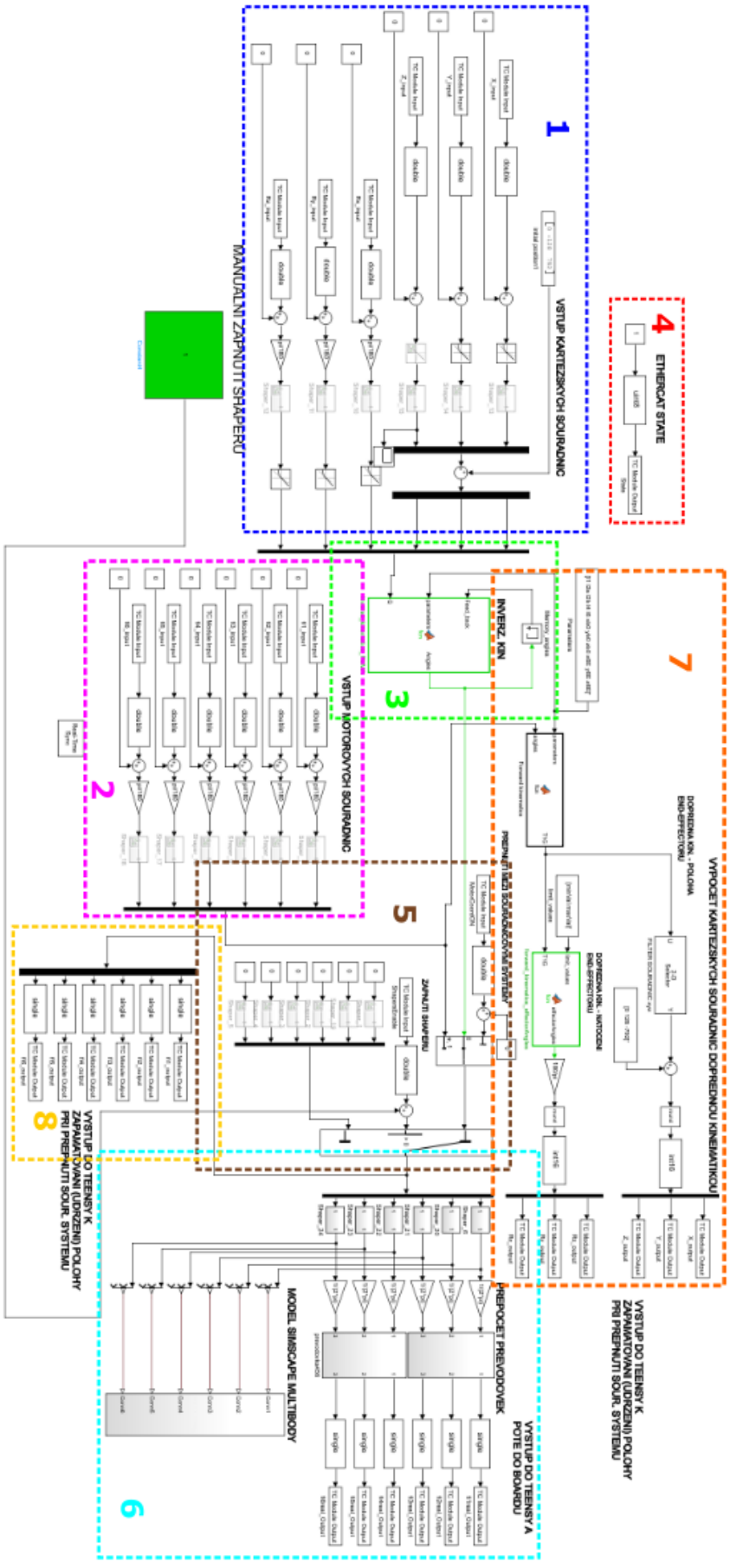
```
b1 = (byte *)&EASYCAT.BufferOut.Cust.fi1real; // this way I split floa
b2 = (byte *)&EASYCAT.BufferOut.Cust.fi2real;
b3 = (byte *)&EASYCAT.BufferOut.Cust.fi3real;

Serial1.write("S"); //start bit so the receiver (f28379d) will not rea

Serial1.write(S[0]); //sending State of ethercat bus
Serial1.write(S[1]);
Serial1.write(S[2]);
Serial1.write(S[3]);

Serial1.write(b1[0]); //sending first float
Serial1.write(b1[1]);
Serial1.write(b1[2]);
Serial1.write(b1[3]);

Serial1.write(b2[0]); //sending second float
Serial1.write(b2[1]);
Serial1.write(b2[2]);
Serial1.write(b2[3]);

Serial1.write(b3[0]); //sending third float
Serial1.write(b3[1]);
Serial1.write(b3[2]);
Serial1.write(b3[3]);

Serial1.write("E"); //end bit
```

Obrázek 60 - Konverze floatových dat na pole bytů a zaslání přes SCI do první platformy F28379D

Tímto byla stručně popsána struktura programu v *Teensy*. Zajištění fungujícího programu zařizující komunikaci *Teensy* s ostatními hardwarovými prvky se sice může zdát jako jednoduchý úkol, avšak za úspěšnou funkčností tohoto programu stojí desítky pracovních hodin a úvah.

4.2.4 Řízení v prostředí MATLAB/Simulink

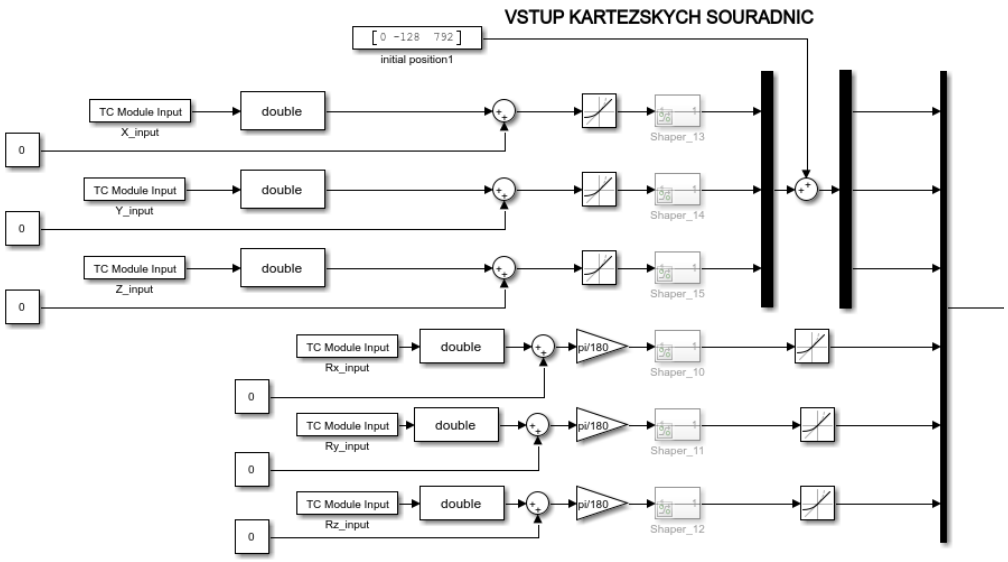
V systémovém návrhu na obr. 37 se nyní nacházíme v bloku číslo 4 týkajícího se PC, konkrétně v bloku značeným 4.1. Zde bude popisován model vytvořený v prostředí MATLAB/Simulink, který zajišťuje inverzní a dopřednou kinematiku a pracuje se vstupními a výstupními daty ve spolupráci s TwinCATem. Celý model lze vidět na obr. 61. Model je pro přehlednost rozdělen do 8 pomyslných bloků, jejichž funkce bude popsána v následujících částech.



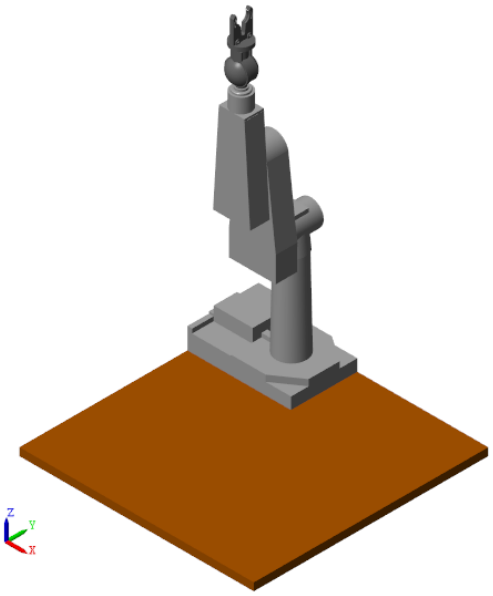
Obrázek 61 - Model v Simulinku

4.2.4.1 Blok 1

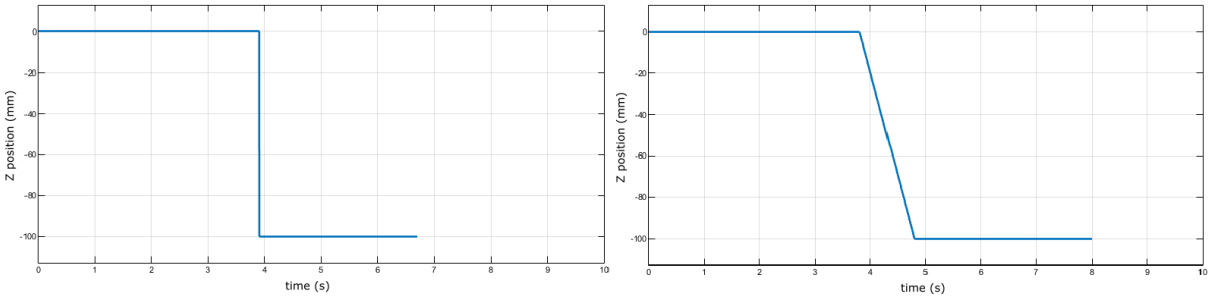
Blok číslo 1 (obr. 62) se týká vstupních proměnných zasílaných z displeje. Jak již bylo řečeno, hodnoty z displeje se skrz *Teensy* a *EasyCAT PRO* dostanou do PC, kde jsou zpracovány v TwinCATu. Aby mohl model v Simulinku přijímat vstupy, resp. výstupy z TwinCATu, je potřeba využít bloky TC Module Input, resp. TC Module Output. Vstupy v tomto bloku nesou informace o pozici *x, y, z* a natočení *Rx, Ry, Rz* koncového efektoru, které jsou poslány do funkčního bloku počítajícího inverzní kinematiku. Pozice *x, y, z* jsou brány vzhledem k počáteční pozici koncového efektoru. Počáteční pozice robota je taková, že robot má obě ramena směrem „vzhůru". Počáteční pozici lze vidět na vytvořeném modelu v *Simscape Multibody* (obr. 63), který byl vytvořen ve spolupráci s kolegou Bc. Simeonem Dragenčevem a na němž byl kontrolován vytvořený model v Simulinku. Na obr. obr. 62 si lze všimnout také bloku *rate limiter*, který se předřadil před vstupem do inverzní kinematiky. Důvodem použití tohoto bloku je omezení skokových změn na vstupu. Inverzní kinematika počítá nejvhodnější konfiguraci pohybu robota tím způsobem, že bere v úvahu hodnoty v předchozím časovém okamžiku. Při skokové změně se může stát, že si robot vybere nesmyslnou dráhu přejezdu na dané místo, jelikož má informaci pouze o počáteční a koncové pozici a nemá možnost konfigurovat vhodnou trasu. Blok *rate limiter* ze skokové změny udělá lineární průběh a inverzní kinematika bude mít k dispozici více hodnot poloh mezi počáteční a koncovou hodnotou a možnost výpočtu vhodné převodní trajektorie robota. Ukázka průběhu vstupní hodnoty bez bloku *rate limiter* a s blokem *rate limiter* je vyobrazena na obr. 64.



Obrázek 62 - Vstup kartézských souřadnic z displeje



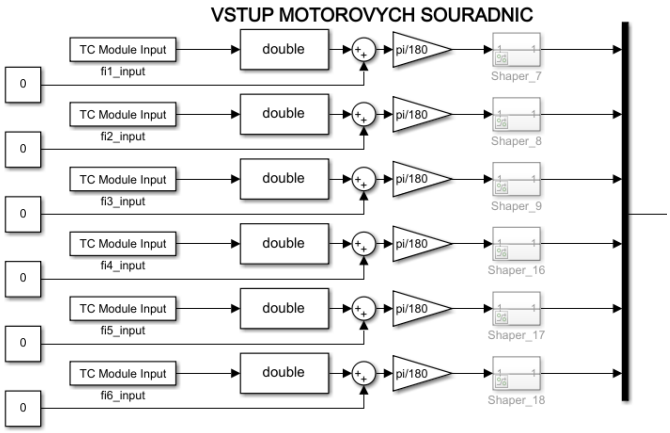
Obrázek 63 - Počáteční poloha robota v Simscape Multibody



Obrázek 64 - vstup do inverzní kinematiky bez použití bloku rate limiter (vlevo) a s použitím bloku rate limiter (vpravo)

4.2.4.2 Blok 2

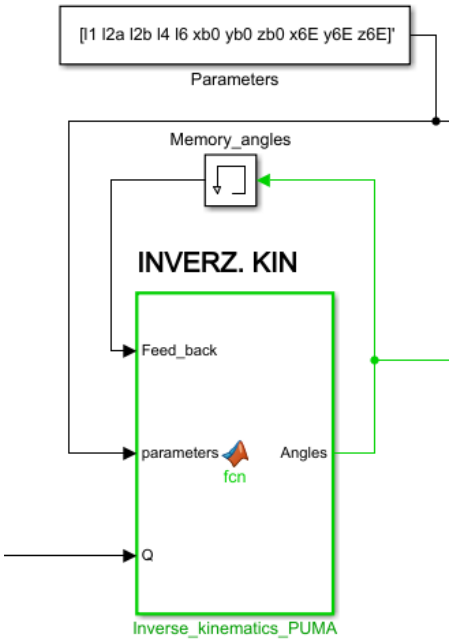
Blok dva (obr. 65) má funkci vstupu motorových souřadnic po přepnutí na daný motorový souřadnicový systém na displeji. Zde již není potřeba mít předřadný blok *rate limiter*, jelikož hodnoty z motorových souřadnic se posílají přímo do převodovek bez nutnosti počítat inverzní kinematiku.



Obrázek 65 - Vstup motorových souřadnic v Simulinku

4.2.4.3 Blok 3

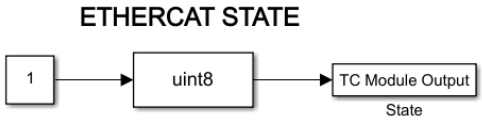
Blok 3 (obr. 66) obsahuje inverzní kinematiku převzatou od Ing. Jindřicha Dvořáka [42]. Tato inverzní kinematika je vylepšená oproti původní verzi především o implementaci nejlepší konfigurace polohy robota s ohledem na přechozí polohy. Ve funkčním bloku s inverzní kinematikou byly pouze upraveny rozměry tak, aby mohla být kinematika použita na robota PUMA200.



Obrázek 66 - Funkční blok počítající inverzní kinematiku

4.2.4.4 Blok 4

Blok 4 (obr. 67) v Simulinku posílá na výstup konstantní hodnotu „1". Tato hodnota se poté ukládá do již zmiňované *proměnné State*, která je posílána do *Teensy*. Pokud se EtherCAT komunikace přeruší, přeruší se i spojení mezi *EasyCAT PRO* a TwinCATem a veškeré hodnoty nakonfigurované v modulu se změní na „0". Důvodem přerušení komunikace může být fyzické přerušení komunikace nebo softwarové přerušení, kdy se robot dostane do singulární polohy a program v Simulinku implementovaný v TwinCATu se přeruší kvůli matematické chybě.

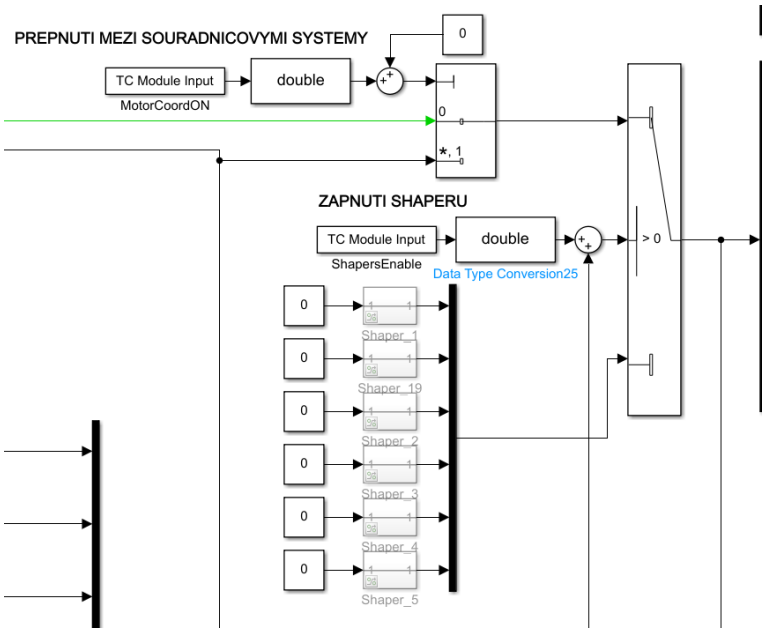


Obrázek 67 - Proměnná State posílaná z TwinCATu do Teensy

4.2.4.5 Blok 5

Blok 5 (obr. 68) zobrazuje funkci přepínání mezi souřadnicovými systémy, kdy se při hodnotě proměnné *MotorCoord* „0" propustí data vypočítaná z inverzní kinematiky a při hodnotě „1" data nesoucí přímo informace o natočení os. Dále lze v bloku vidět vstup

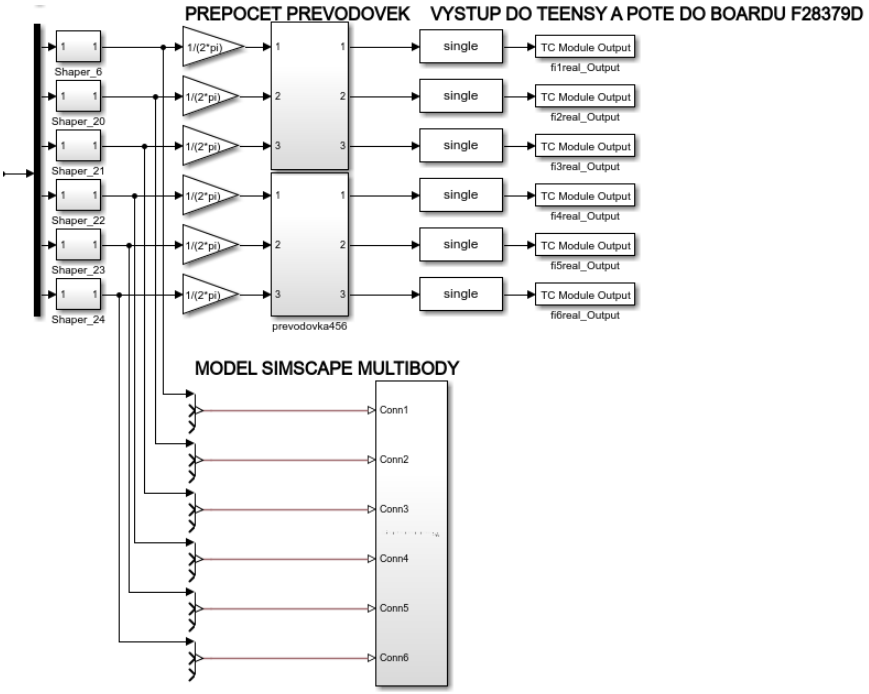
ShapersEnable, který odpovídá hodnotě vypínače *SHAPERS* na displeji a propouští data dále v modelu k blokům *SHAPERY* při hodnotě „1“. Při hodnotě „0“ jsou na výstupu nulové hodnoty natočení a robot zůstává v počáteční pozici.



Obrázek 68 - Blok zobrazující funkci přepínání souřadnicových systémů a zapnutí shaperů

4.2.4.6 Blok 6

Blok 6 (obr. 69) vyobrazuje, jak se data ze vstupů již dostala přes bloky *shaperů*, kdy data začnou vstupovat do převodovek a zkorigovaná data se zašlou na výstup do desky *Teensy*. Bloky *shaperů* a bloky převodovek byly převzaty již z hotové práce Ing. Michaela Valáška [35]. *Shapery* mají za úkol konvergovat skokovou změnu dat na postupnou, plynulou změnu dat, aby se na výstupu nezobrazovaly skokové hodnoty a nedocházelo k okamžitým změnám polohy motorů robota, což by mohlo motory poškodit. Před převodovkou se hodnoty úhlových natočení změní na otáčky, otáčky se v převodovce převedou na počty žádaných pulzů, které mají být na vstupu do řízení. Dále se pomocí již zmíněných bloků TC Module Output data pošlou přes TwinCAT do *Teensy*, které je pošle do platforem *F28379D*. Na straně robota se díky datům z encoderů poskytují polohová a rychlostní data potřebná pro PID regulátory.

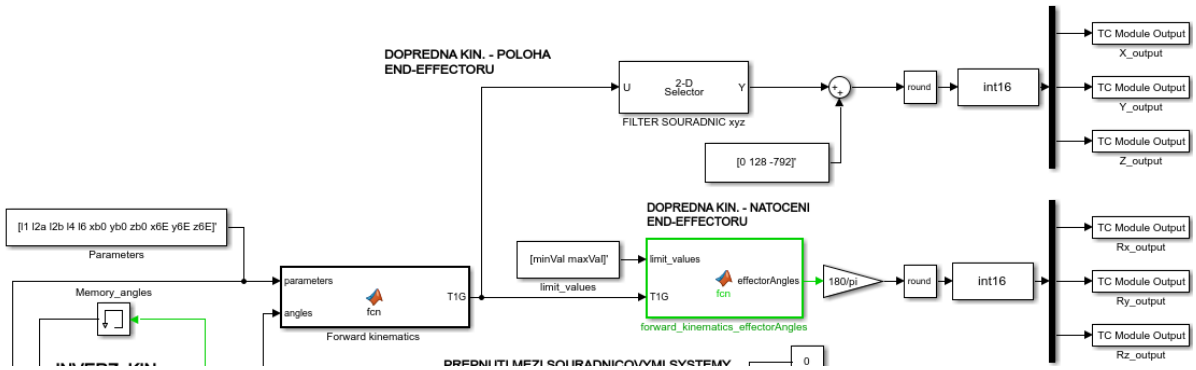


Obrázek 69 - Blok vyobrazující výstupy z modelu v Simulinku

Data ze vstupů se vložila do Simulinkového subsystému k vidění v dolní části obr. 69 s modelem vytvořeným v *Simscape Multibody* a testovala se správnost pohybu robota PUMA200.

4.2.4.7 Blok 7

Blok 7 (obr. 70) se zabývá dopřednou kinematikou, tedy nalezením polohy bodu *x, y, z* a orientacemi (natočením) *Rx, Ry, Rz* koncového efektoru. Nejprve se ve funkčním bloku *Forward kinematics* vypočítá výsledná transformační matice robota značená $\boldsymbol{T}_{1G}$. Z transformační matice se poté spočítají natočení a poloha koncového efektoru. Tato část modelu je nutná pro uchování robota ve stávající poloze při přepnutí souřadnicového systému. Pokud uživatel přepne z motorových souřadnic do kartézských, je potřeba, aby se v Simulinku přepočítaly aktuální motorové souřadnice na kartézské souřadnice, a poslaly se přes EtherCAT sběrnici do *Teensy*, ve kterém se přepíšou příslušné proměnné a ty se vypíšou na displej. Následující část se bude zabývat popisem robota dopřednou kinematikou.

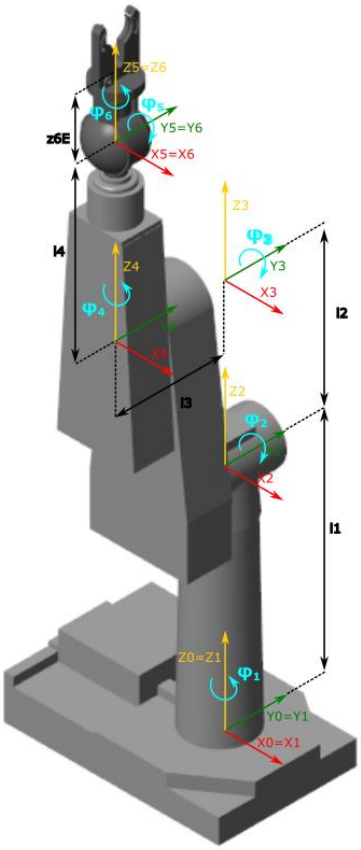


Obrázek 70 - Blok řešící dopřednou kinematiku a zaslání souřadnic do Teensy

Dopředná kinematika robota PUMA200

Zavedení souřadnicových systémů na robotovi je zobrazeno na obr. 71. Od počátku souřadnicového systému $(x_0, y_{0,} z_0)$ se dostaneme do souřadnicového systému koncového efektoru $(x_6, y_{6,} z_6)$ pomocí násobení příslušných transformačních matic. Výsledná transformační matice $(\boldsymbol{T}_{1G}')$., která vznikne násobením dílčích transformačních matic již obsahuje prvky, ze kterých lze získat konečnou pozici a natočení koncového efektoru. Z obr. 71 lze odečíst jednotlivá natočení a posunutí, ze kterých vytvoříme výslednou transformační matici $\boldsymbol{T}_{1G}'$ [43]:

$$\boldsymbol{T}_{1G}' = \boldsymbol{T}_{\varphi z}(\varphi_1)\boldsymbol{T}_z(l_1)\boldsymbol{T}_{\varphi y}(\varphi_2)\boldsymbol{T}_z(l_2)\boldsymbol{T}_{\varphi y}(\varphi_3)\boldsymbol{T}_y(-l_3)\boldsymbol{T}_{\varphi z}(\varphi_4)\boldsymbol{T}_z(l_4)\boldsymbol{T}_{\varphi y}(\varphi_5)\boldsymbol{T}_{\varphi z}(\varphi_6) \quad (1)$$



Obrázek 71 - Zavedení souřadnicových systémů na robotovi

Abychom se dostali do žádaného bodu v souřadnicovém systému koncového efektoru, je ještě potřeba vynásobit matici $\boldsymbol{T}_{1G}'$ translační maticí s offsetem koncového efektoru *z6E*:

$$\boldsymbol{T}_{1G} = \boldsymbol{T}_{1G}'\boldsymbol{T}_z(z6E) = \begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

Matici $\boldsymbol{T}_{1G}$ získáme i v případě, kdybychom se nejdříve násobením translačními maticemi dostali do koncového bodu a poté bod natočili rotačními maticemi kolem příslušných os [44]:

$$\boldsymbol{T}_{1G} = \boldsymbol{T}_x(x)\boldsymbol{T}_y(y)\boldsymbol{T}_z(z)\boldsymbol{T}_{\varphi x}(\varphi_x)\boldsymbol{T}_{\varphi y}(\varphi_y)\boldsymbol{T}_{\varphi z}(\varphi_z) \quad (3)$$

Kde x, y, z je výsledná pozice koncového bodu a $\varphi_x, \varphi_y, \varphi_z$ jsou natočení kolem os.

Za pomocí symbolického toolboxu v programu MATLAB lze určit, že:

$$\boldsymbol{T}_x(x)\boldsymbol{T}_y(y)\boldsymbol{T}_z(z)\boldsymbol{T}_{\varphi x}(\varphi_x)\boldsymbol{T}_{\varphi y}(\varphi_y)\boldsymbol{T}_{\varphi z}(\varphi_z) = \tag{4}$$

$$= \begin{bmatrix} cos(\varphi_y)cos(\varphi_z) & -cos(\varphi_y)sin(\varphi_z) & sin(\varphi_y) & x \\ cos(\varphi_x)sin(\varphi_z) + cos(\varphi_z)sin(\varphi_y)sin(\varphi_x) & cos(\varphi_x)cos(\varphi_z) - sin(\varphi_y)sin(\varphi_x)sin(\varphi_z) & -cos(\varphi_y)sin(\varphi_X) & y \\ sin(\varphi_x)sin(\varphi_z) - cos(\varphi_x)cos(\varphi_z)sin(\varphi_y) & cos(\varphi_z)sin(\varphi_x) + cos(\varphi_x)sin(\varphi_y)sin(\varphi_z) & cos(\varphi_y)cos(\varphi_x) & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Porovnáním (2) a (4) získáme:

$$\begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \tag{5}$$

$$= \begin{bmatrix} cos(\varphi_y)cos(\varphi_z) & -cos(\varphi_y)sin(\varphi_z) & sin(\varphi_y) & x \\ cos(\varphi_x)sin(\varphi_z) + cos(\varphi_z)sin(\varphi_y)sin(\varphi_x) & cos(\varphi_x)cos(\varphi_z) - sin(\varphi_y)sin(\varphi_x)sin(\varphi_z) & -cos(\varphi_y)sin(\varphi_X) & y \\ sin(\varphi_x)sin(\varphi_z) - cos(\varphi_x)cos(\varphi_z)sin(\varphi_y) & cos(\varphi_z)sin(\varphi_x) + cos(\varphi_x)sin(\varphi_y)sin(\varphi_z) & cos(\varphi_y)cos(\varphi_x) & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A tedy hledané souřadnice koncového bodu jsou:

$$x = s_{14} \tag{6}$$

$$y = s_{24}$$

$$z = s_{34}$$

Nyní je ještě zapotřebí nalézt úhly natočení $\varphi_x, \varphi_y, \varphi_z$. Z matice (5) je zřejmé, že v intervalu $(0; \pi)$:

$$\varphi_{y1} = asin\,(s_{31}) \tag{7}$$

$$\varphi_{y2} = \pi - asin\,(s_{31})$$

A v intervalu $(-\pi; 0)$:

$$\varphi_{y1} = asin\,(s_{31}) \tag{8}$$

$$\varphi_{y2} = \pi + asin\,(s_{31})$$

V obou případech MATLAB volí řešení takové, aby platilo:

$$abs(\varphi_y) < \frac{\pi}{2} \tag{9}$$

K nalezení úhlu φ_x si v matici (5) všimneme, že:

$$\frac{s_{23}}{s_{33}} = -tan\,(\varphi_x) \tag{10}$$

Z toho vyplývá, že:

$$\varphi_x = -atan2(s_{23}, s_{33}) \tag{11}$$

Pokud by platilo, že $cos(\varphi_y) > 0$, platilo by $\varphi_x = -atan2(s_{23}, s_{33})$. Pokud by platilo, že $cos(\varphi_y) < 0$, pak $\varphi_x = -atan2(-s_{23}, -s_{33})$. Jelikož platí, že MATLAB volí řešení z rovnice -9, pak $cos(\varphi_y) > 0$ a platí, že $\varphi_x = atan2(s_{23}, s_{33})$. Případ, kdy $cos(\varphi_y) = 0$ bude probrán později.

Obdobně postupujeme u úhlu φ_z , kdy si v matici (5) všimneme, že:

$$\frac{s_{12}}{s_{11}} = -tan\,(\varphi_z) \tag{12}$$

Jelikož platí $\cos(\varphi_y) > 0$, pak:

$$\varphi_z = -atan2(s_{12}, s_{11}) \tag{13}$$

U případu, kdy $cos(\varphi_y) = 0$, platí že $\varphi_y = \pm\frac{\pi}{2}$. Každý případ bude řešen zvlášť.

V případě $\varphi_y = +\frac{\pi}{2}$ platí rovnice:

$$s_{21} = cos(\varphi_x)sin(\varphi_z) + cos(\varphi_z)sin(\varphi_x) = sin(\varphi_x + \varphi_z) \tag{14}$$

$$s_{31} = sin(\varphi_x)sin(\varphi_z) - cos(\varphi_x)cos(\varphi_z) = cos(\varphi_x + \varphi_z) \tag{15}$$

S použitím rovnic (14) a (15) získáme:

$$\varphi_x + \varphi_z = atan2(s_{21}, s_{31}) \tag{16}$$

$$\varphi_z = atan2(s_{21}, s_{31}) - \varphi_x \tag{17}$$

V případě $\varphi_y = -\frac{\pi}{2}$ platí rovnice:

$$s_{21} = cos(\varphi_x)sin(\varphi_z) - cos(\varphi_z)sin(\varphi_x) = sin(\varphi_z - \varphi_x) \tag{18}$$

$$s_{31} = sin(\varphi_x)sin(\varphi_z) + cos(\varphi_x)cos(\varphi_z) = cos(\varphi_x - \varphi_z) \tag{19}$$

S použitím rovnic (18) a (19) a získáme:

$$\varphi_{z-}\varphi_x = atan2(s_{21}, s_{31}) \tag{20}$$

$$\varphi_z = atan2(s_{21}, s_{31}) + \varphi_x \tag{21}$$

Vidíme tedy, že rovnice (17) a (21) mají nekonečně mnoho řešení. Program v případě $cos(\varphi_y) = 0$ volí $\varphi_x = 0$. [45]

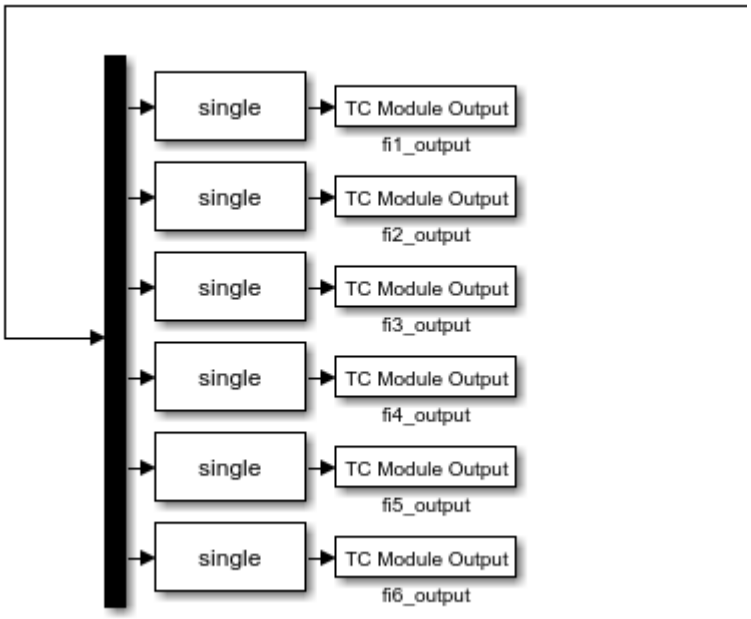
Tímto jsme získali všechny úhly natočení koncového efektoru. Ukázka kódu v MATLABu je vidět na obr. 72.

```
function effectorAngles = fcn(limit_values,T1G)
minVal=limit_values(1);
maxVal=limit_values(2);

if (T1G(1,3) >= minVal) && (T1G(1,3) <= maxVal)
        psi=0;
        phi=atan2(T1G(2,1),T1G(3,1))-psi;
        theta=pi/2;
elseif  (T1G(1,3) >= -maxVal) && (T1G(1,3) <= -minVal)
        psi=0;
        phi=atan2(T1G(2,1),T1G(3,1))+psi;
        theta=-pi/2;
else
           theta=asin(T1G(1,3));
           psi=-atan2(T1G(2,3),T1G(3,3));
           phi=-atan2(T1G(1,2),T1G(1,1));
end
       effectorAngles=[psi,theta,phi];
```

Obrázek 72 - Kód určující úhly natočení koncového efektoru

4.2.4.8 Blok 8

Blok 8 má podobnou funkci jako Blok 7. Při přepnutí z kartézských souřadnic na motorové souřadnice se díky výstupům na obr. 73. pošlou data o natočení os spočítané z inverzní kinematiky do *Teensy*, které je přepíše na displej. Hodnoty natočení se nemohou brát z výstupů v bloku 6, jelikož hodnoty těchto výstupů jsou kvůli *shaperům* zpožděny.



Obrázek 73 - Nezpožděné hodnoty natočení os zaslané do Teensy

4.2.5 TwinCAT 3

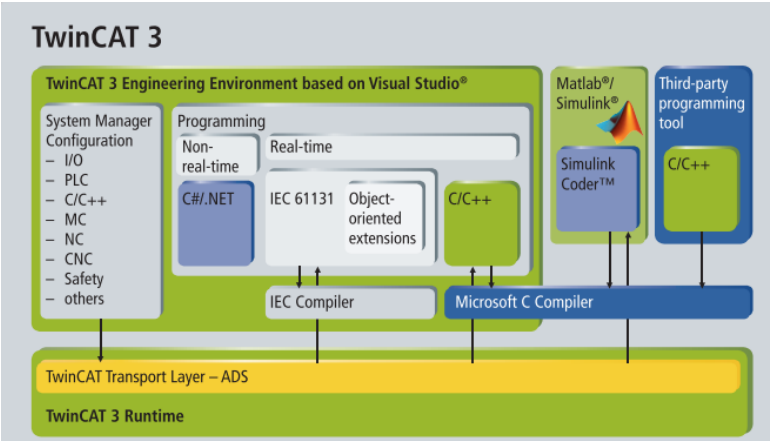
Poslední diskutovanou částí v systémovém návrhu na obr. 37 bude popis softwaru TwinCAT, který hraje v komunikaci na EtherCAT sběrnici roli mastera.

The Windows Control and Automation Technology (TwinCAT) je řídící software vyvinut firmou *Beckhoff Automation*. Tato společnost tvrdí, že téměř všechny druhy řídících aplikací mohou být se softwarem TwinCAT vytvořeny. K realizaci aplikací může uživatel využívat různé programovací jazyky (C, C++, MATLAB/Simulink). Filozofie TwinCAT 3 je přesunuta do koncepce řídícího software formou nezávislých softwarových modulů, což zajišťuje, že ovládání komplexních moderních strojů je zjednodušeno a inženýrské úsílí je sníženo. TwinCAT může být integrován do existujících softwarových vývojových prostředí, jako je například *Visual Studio 2019*. Další vlastností TwinCAT 3 je, že poskytuje real-time prostředí, kde mohou být TwinCAT moduly načteny, spuštěny a ovládány [46]. Jak bylo zmíněno výše, každý modul může být vytvořen pomocí různých programovacích jazyků. Struktura softwaru TwinCAT 3 jako celku je rozdělena na dvě části:

- Vývojové prostředí (*eXtended Automation Engineering* – XAE)
- Runtime (*eXtended Automation Runtime* – XAR)

Tyto dvě části jsou propojené transportní vrstvou ADS (*Automation Device Specification*)

Inženýrské prostředí TwinCAT a modulární struktura jsou vidět na obr. 74, respektive na obr. 75.



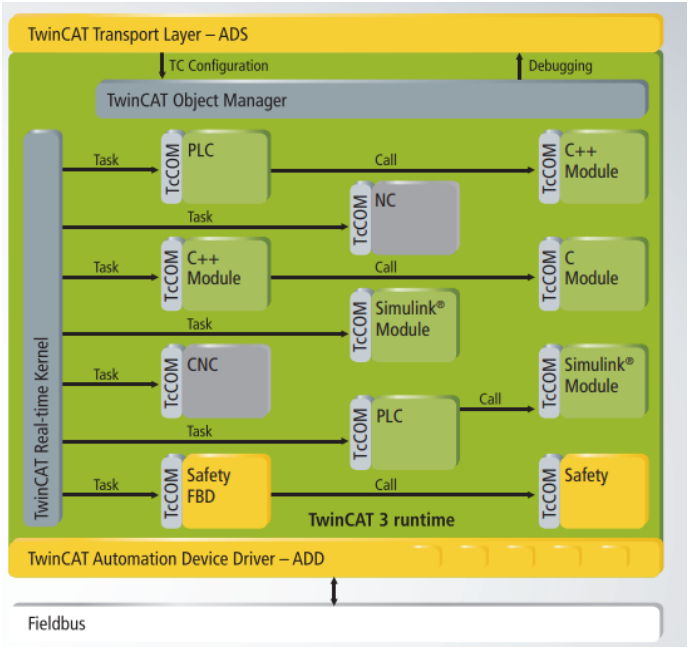
Obrázek 74 - Inženýrské prostředí TwinCAT 3 [47]

Další výhodou TwinCAT 3 je podpora vícejádrových procesorů, kdy se mohou různé moduly a jejich úkoly přiřadit různým procesorům v PC. Tato vlastnost poskytuje moderním PC využít naplno své výkonové limity [46].

Systémové požadavky pro TwinCAT 3 jsou následující [48]:

- TwinCAT 3 XAE:
  - Operační systém:
    - Windows 7 Service Pack 1
    - Windows Server 2008 R2 SP1
    - Windows Server 2012 (R2)
    - Windows 10
  - Hardware:
    - Rychlost procesoru 1.8 Ghz nebo rychlejší
    - 2 GB RAM (Doporučeno 4 GB RAM)
    - Volné místo v paměti počítače – alespoň 10 GB, pokud ještě nebylo nainstalováno vývojové prostředí *Visual Studio*
    - Rychlost disku v PC – doporučeno používat SSD
    - Grafická karta, která podporuje rozlišení alespoň 720p (1208x720). Je doporučeno rozlišení *FullHD* (1920x1080) nebo vyšší
- TwinCAT 3 XAR:
  - Operační systém:
    - Windows Embedded Standard 2009: podporován až do TwinCAT 3.1 Build 4022
    - Windows 7
    - Windows Embedded Standard 7
    - Windows 10
    - Windows 10 LTSB

- Hardware:
  - 500 MB volného místa v paměti
  - Velikost RAM paměti v závislosti na aktivované konfiguraci



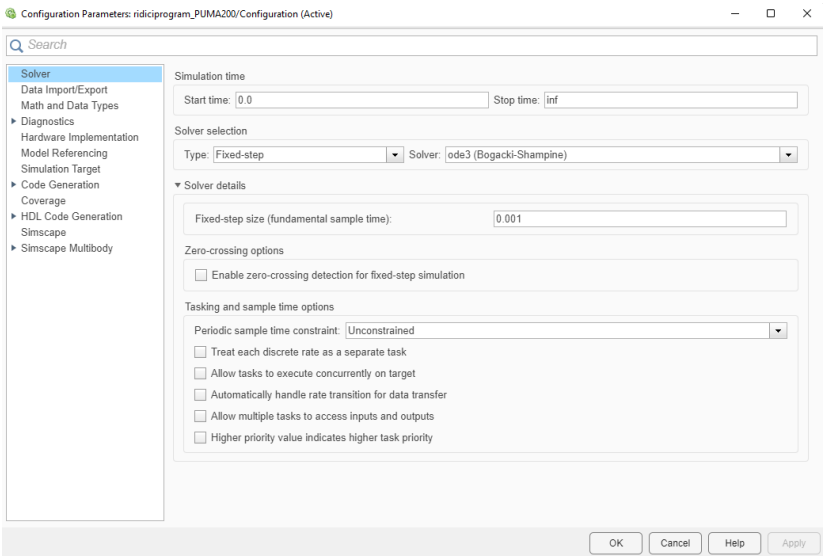
Obrázek 75 - Modulární struktura běhového prostředí TwinCAT 3 [47]

Pro spolupráci softwarů TwinCAT a MATLAB/Simulink je potřeba ze stránek firmy *Beckhoff* stáhnout a nainstalovat podporu pro MATLAB (*TE1401*) a pro Simulink (*TE1400*). Při použití rozšíření *TE1400* a *TE1401* je možné spouštět analýzy a simulace vytvořené v MATLABu a Simulinku v TwinCATu za běhu v reálném čase.

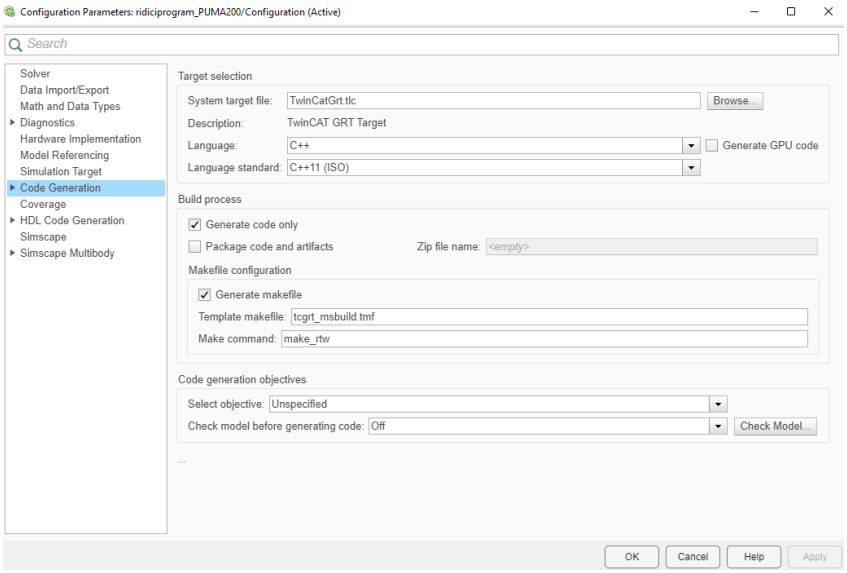
Požadavky pro instalaci *TE1400* a *TE1401* od verze 2.x.xxxx.x jsou následující [49]:

- MATLAB 2019a nebo vyšší
  - Simulink® and Simulink Coder™ Toolbox
  - MATLAB® and sMATLAB Coder™
- Visual Studio 2017 nebo vyšší (Proffesional, Ultimate nebo ekvivalentní edici)
  - Během instalace musí být zakliknuta možnost *Vývoj desktopových aplikací pomocí C+*
- Twincat 3.1.424.7 nebo vyšší
  - Instalace Twincat 3.1 musí být provedena až po instalaci *Visual Studio*

Po vytvoření modelu v MATLAB/Simulink je potřeba vytvořit tzv. *TcCOM* objekt, což je objekt, pomocí kterého je model v Simulinku integrován do TwinCATu, kde se může model spustit a analyzovat v reálném čase. Pro vytvoření *TcCOM* modulu se v nastavení modelu v Simulinku nastaví parametry dle obr. 76 a obr. 77.



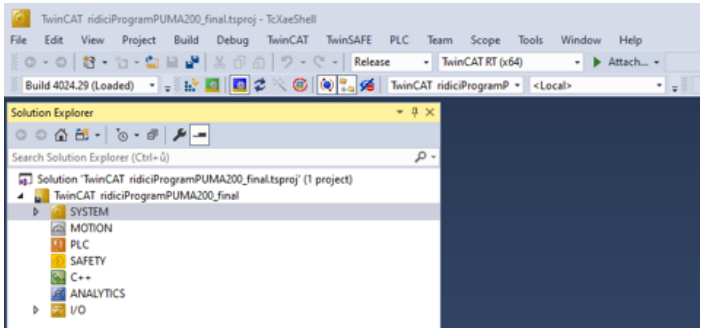
Obrázek 76 - Nastavené parametry záložce Solver



Obrázek 77 - Nastavení parametru v záložce Code Generation

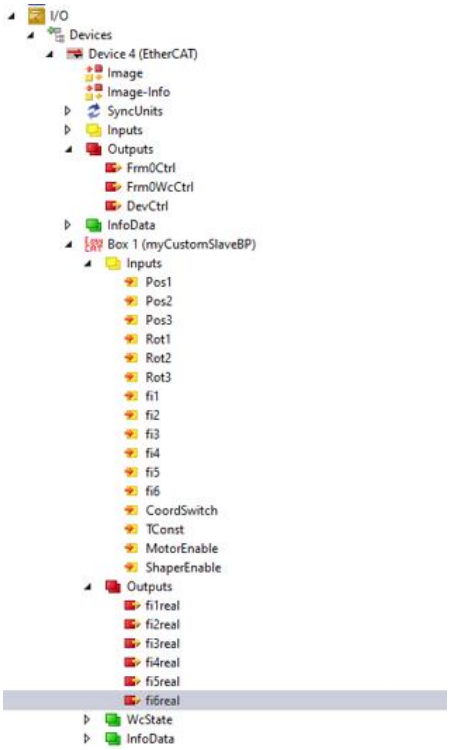
Ve finále musíme kliknout v Simulinku na Sestavení (*Build*). Tím se nám vytvoří *TcCOM* objekt.

V softwaru TwinCAT vytvoříme nový projekt. Nově vytvořený projekt lze vidět na obr. 78.



Obrázek 78 - Nový projekt v TwinCATu

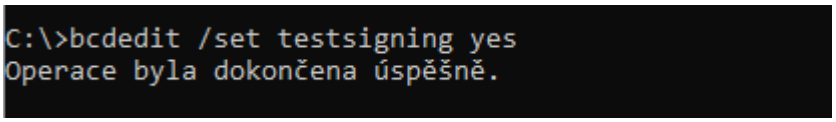
Nyní v záložce I/O klikneme pravým tlačítkem myši na *Devices* a klikneme na možnost *Scan. Tím* načte slave modul *EasyCAT PRO* připojený ethernetovým kabelem k počítači a také vstupní a výstupní proměnné, které se v modulu nakonfigurovaly (obr. 79).



Obrázek 79 - Načtený modul EasyCAT PRO v prostředí TwinCAT s nakonfigurovanými proměnnými

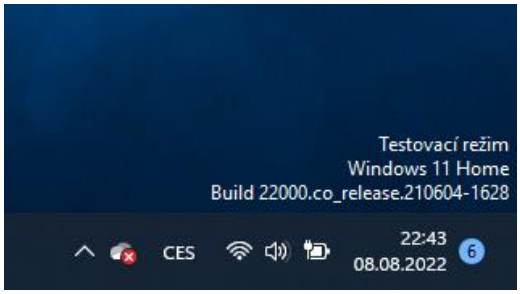
Nyní je ještě zapotřebí vytvořený *TcCOM* objekt tzv. podepsat za použití vlastního certifikátu, který je třeba vytvořit.

Jako první je potřeba přejít do tzv. testovacího režimu Windows. To se provede otevřením příkazové řádky (CMD) ve Windows jako správce, kde napíšeme příkaz *bcdedit /set testsigning yes* a potvrdíme (obr. 80).



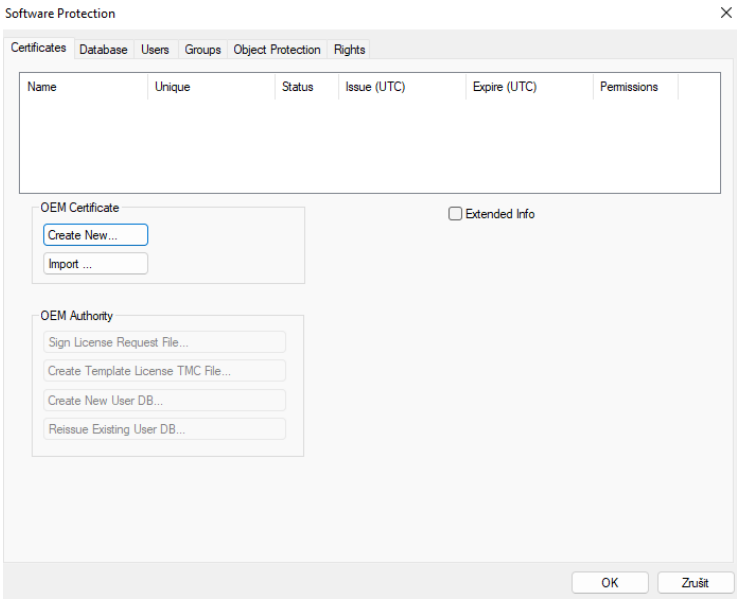
Obrázek 80 - Nastavení testovacího módu Windows

Poté je potřeba PC restartovat. Po spuštění systému bychom měli vidět v pravém dolním rohu úvodní obrazovky, že jsme v testovacím režim (obr. 81).



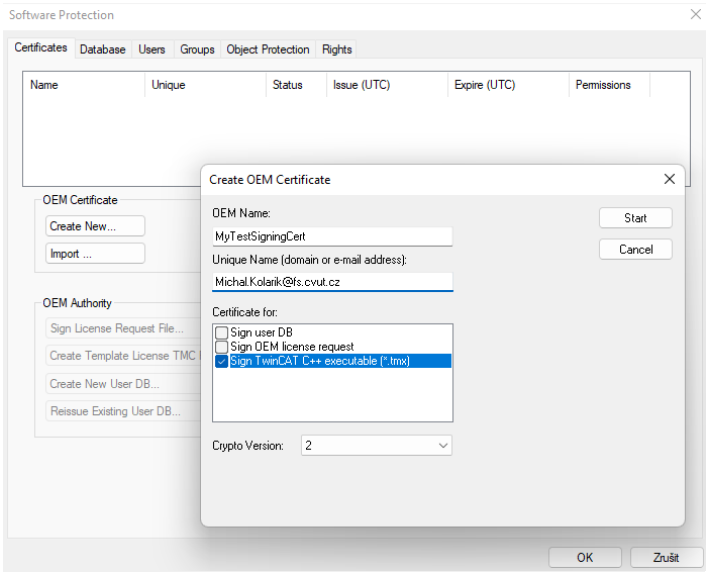
Obrázek 81 - Testovací režim Windows

Nyní je potřeba v TwinCATU v záložce *TwinCAT* na horní liště zvolit ikonu *Software Protection*. Otevře se nám následující okno (obr. 82):



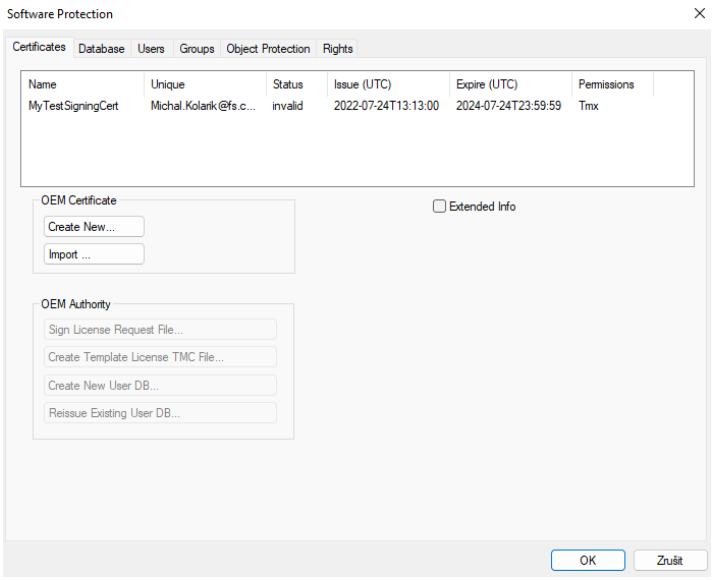
Obrázek 82 - Okno Software Protection

Poté klikneme na ikonu *Create New* a zaklikneme možnost *Sign TwinCAT C++ executable*. Zároveň do pole *OEM Name* zvolíme jméno certifikátu (například *MyTestSigningCert*) a do pole *Unique Name* vložíme e-mailovou adresu (obr. 83).



Obrázek 83 - Vyplněné inforamce v okně Software Protection

Klikneme na tlačítko *Start*. Otevře se nám složka, kde uložíme vytvořený certifikát. Zároveň po nás bude vyžadováno vytvoření hesla. Po zadání hesla a potvrzení se nám objeví v okně *Software Protection* námi vytvořený certifikát (obr. 84).

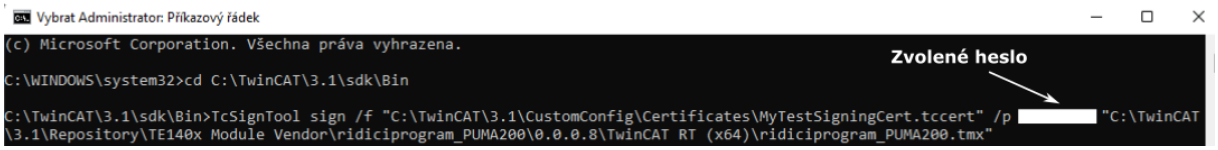


Obrázek 84 - Vytvořený certifikát

Nyní je třeba otevřít příkazový řádek (CMD) ve Windows jako správce a přejít do složky *C:\TwinCAT\3.1\sdk\Bin*. Poté příkazem *TcSignTool sign* podepíšeme náš vytvořený objekt. Příkazu *TcSignTool sign* je nejdříve zapotřebí předat následující parametry:

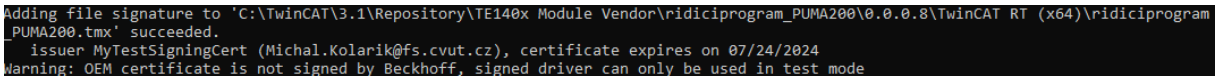
- */f* + umístění našeho certifikátu
- */p* + naše heslo zvolené při vytváření certifikátu + cestu k vytvořenému TMX souboru, který se vytvořil po sestavení v Simulinku

Celý příkaz lze vidět na obr. 85.



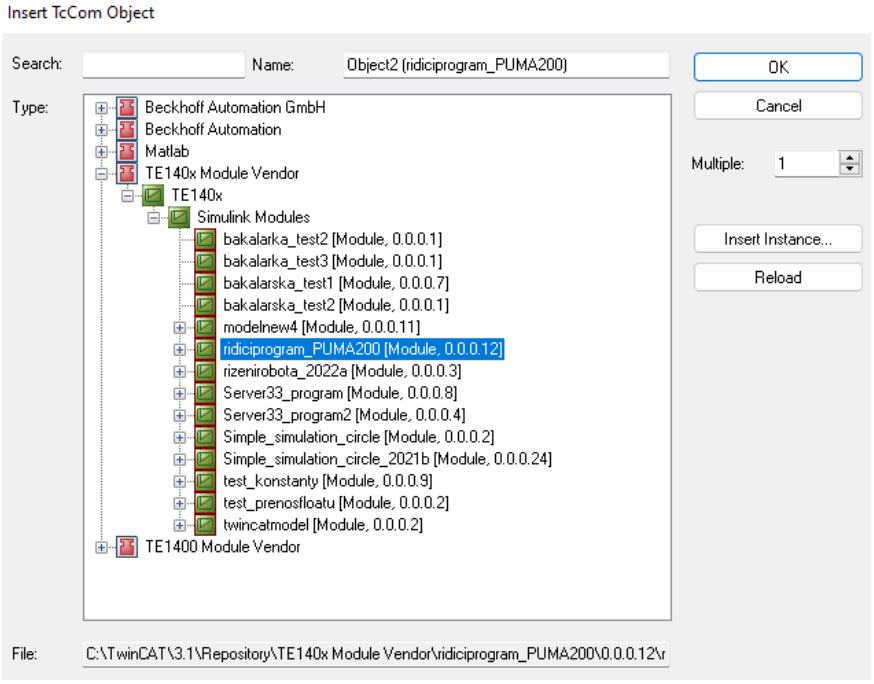
Obrázek 85 - Příkaz v CMD pro podepsání vytvořeného TwinCAT objektu

Po potvrzení klávesou *Enter* se nám objeví úspěšné podepsání pomocí certifikátu (obr. 86).



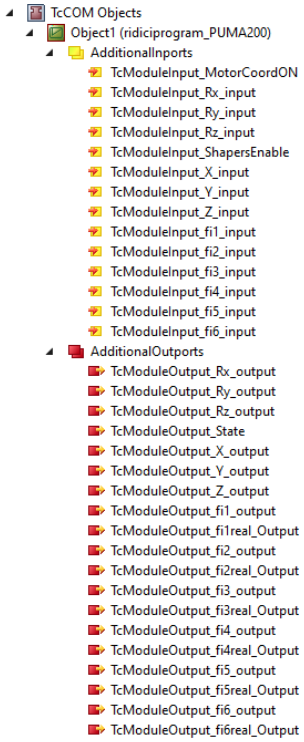
Obrázek 86 - Úspěšné podepsání certifikátem

V TwinCATu přejdeme do záložky *SYSTEM* a klikneme pravým tlačítkem na *TcCOM Objects*. Poté zvolíme ikonu *Add New Item*, kde vybereme náš *TcCOM* objekt. Potvrdíme tlačítkem *OK* (obr. 87).



Obrázek 87 - Přidání TcCOM modulu do TwinCATu

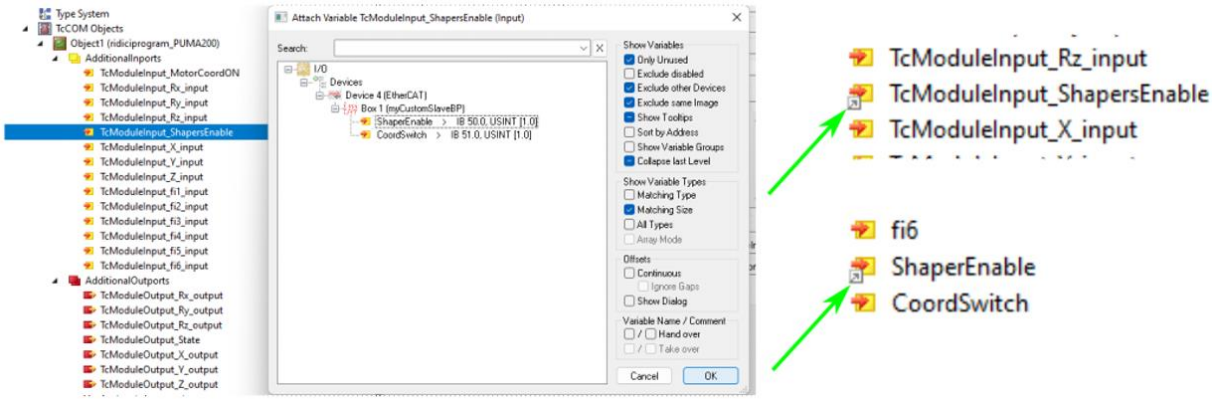
V záložce *TcCOM Objects* se nám objeví náš objekt se vstupy a výstupy, které se vytvořily pomocí bloků *TC Module Input* a *TC Modul Output* v Simulinku (obr. 88).



Obrázek 88 - Vstupy a výstupy vytvořené pomocí TwinCAT bloků v Simulinku

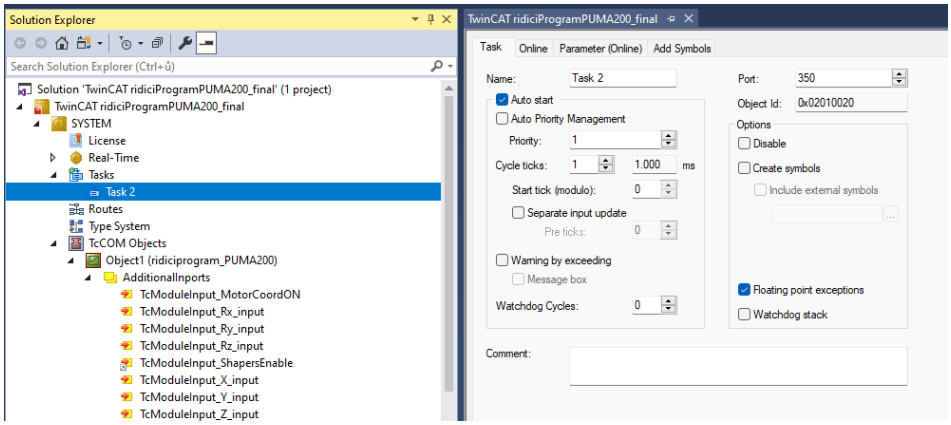
Nyní je potřeba propojit vstupy a výstupy v Simulinku s danými proměnnými nakonfigurovanými v modulu *EasyCAT PRO*. Dvojím kliknutím na jednotlivé vstupy a výstupy v Simulinku se nám otevře okno, ve kterém daný vstup, resp. výstup slinkujeme s příslušnou proměnnou v *EasyCAT PRO*. Po potvrzení klávesou *OK* by se měla objevit malá

šipka u proměnné v Simulinku i u proměnné v modulu *EasyCAT PRO*. Příklad propojení výstupu *TcModulOutpu__ShapersEnable*. V Simulinku s proměnnou *ShaperEnable* v modulu *EasyCAT PRO* je naznačen na obr. 89.



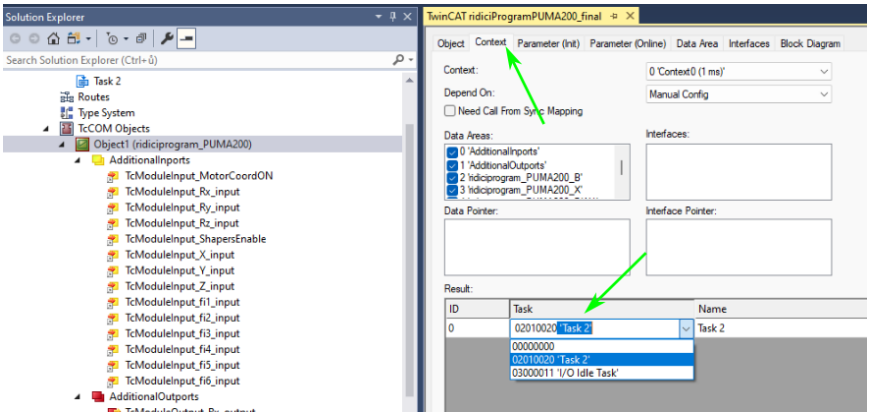
Obrázek 89 - Slinkování proměnných v Simulinku a EasyCAT PRO

V Záložce *SYSTEM* klikneme pravým tlačítkem na *Task* a zvolíme *New Item*. Tím vytvoříme novou úlohu, ve které nastavíme, s jakou periodou bude TwinCAT zpracovávat data real-time. Zvolíme takt 1 ms, jak je naznačeno na obr. 90.



Obrázek 90 - Zvolení taktu 1 ms pro zpracování dat real-time

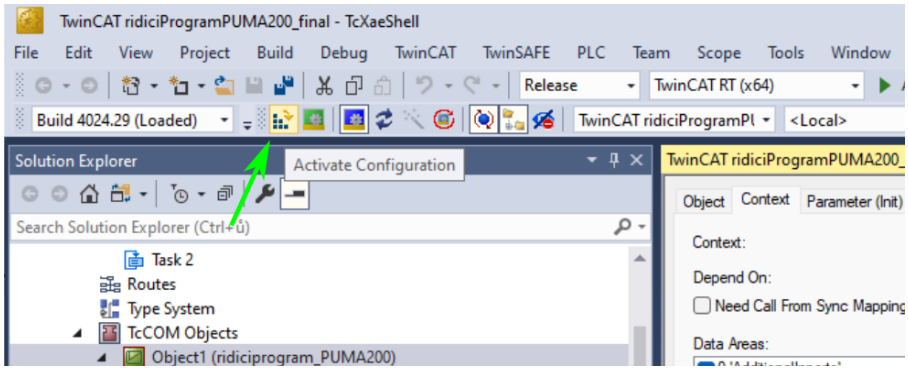
U našeho *TcCOM* modulu přejdeme na záložku *Context* a slinkujeme ho s danou úlohou (obr. 91).



Obrázek 91 - Slinkování TcCOM objektu s vytvořenou úlohou

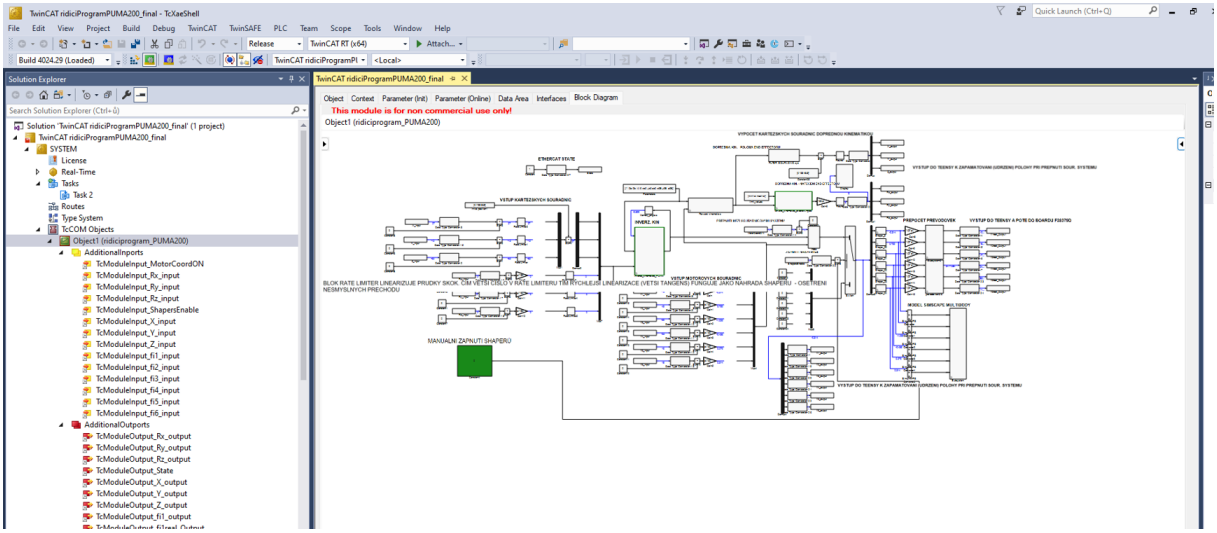
Než spustíme konfiguraci, měly by být již všechny hardwarové prvky propojené a program v *Teensy* by měl být spuštěný.

Aktivujeme konfiguraci pomocí ikony *Activate Configuration* (obr. *92*).



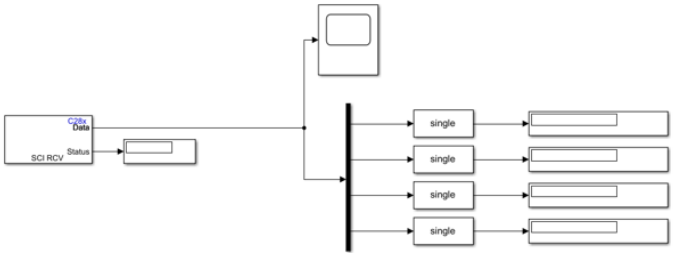
Obrázek 92 - Spuštění konfigurace

V TwinCATu lze nyní v záložce Block Diagram u našeho TcCOM modulu vidět integrovaný model, kde můžeme sledovat real-time hodnoty (obr. 93)



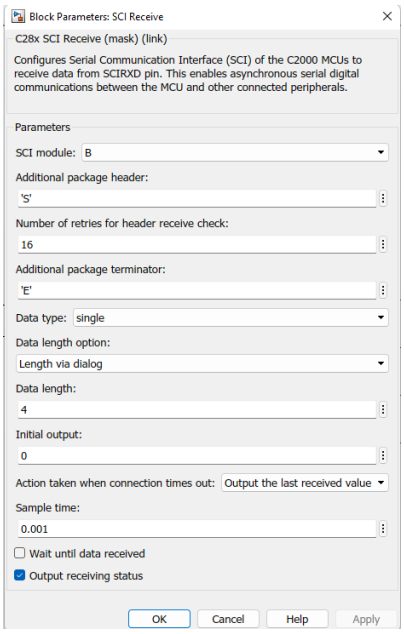
Obrázek 93 - Model Simulinku integrovaný v prostředí TwinCAT

Než se výstupy z *Teensy* připojily na platformy *F28379D* na robotovi, byl vytvořen Simulinkový model, pomocí kterého se sledovala přijatá data skrz SCI do jiné platformy *F28379D*. Model je vidět na obr. 94. Pro přijímání dat pomocí SCI je v Simulinku k dispozici blok *SCI Receive*.



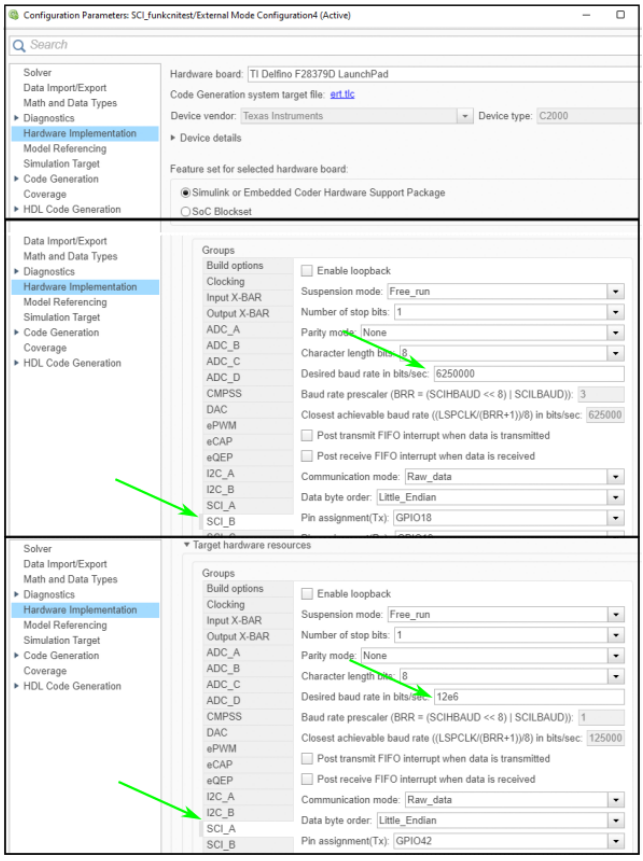
Obrázek 94 – Model v Simulinku nahraný do platformy F28379D pro sledování přijatých dat z Teensy

Nastavení bloku *SCI receive* lze vidět na obr. 95.



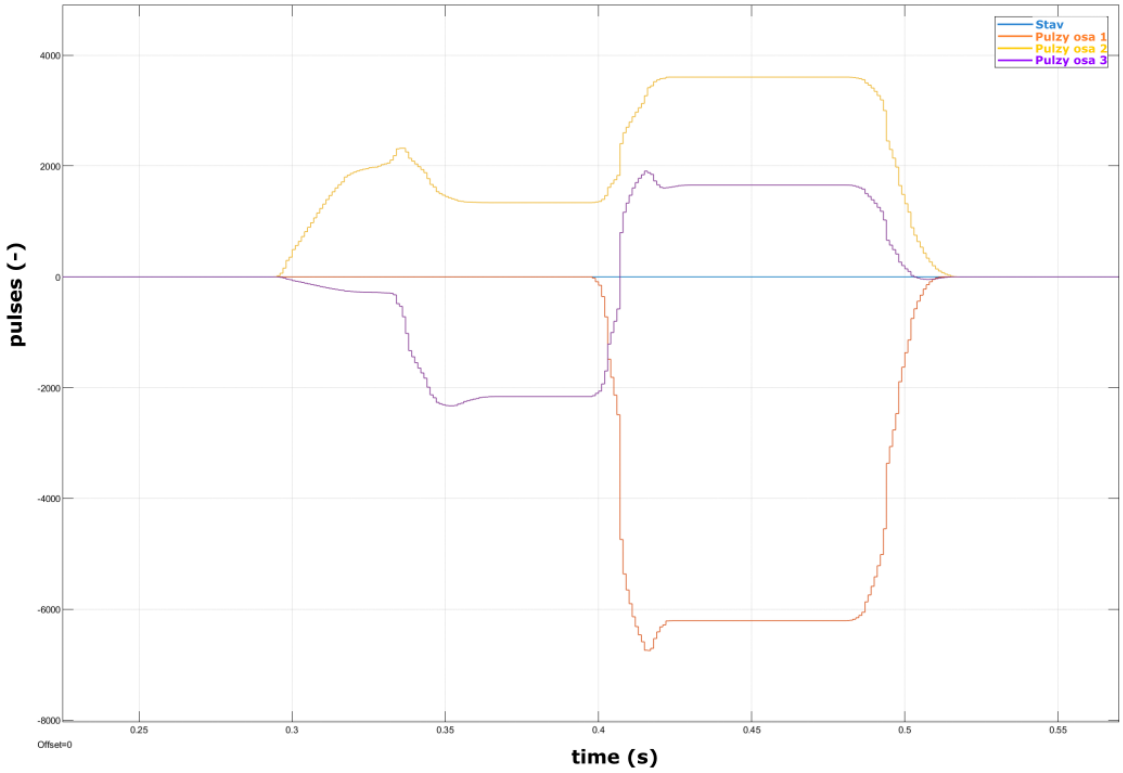
Obrázek 95 - Nastavení bloku SCI Receive

V nastavení modelu lze vidět na obr. 96. Je potřeba v záložce *Hardware Implementation* zvolit platformu TI Delfino F28379D a *baud rate* v záložce *SCI__B*, který odpovídá rychlosti přenosu dat z *Teensy* do platformy. V záložce *SCI__A* se zvolí *baud rate*, který odpovídá rychlosti přenosu dat z platforem do PC v externím módu.



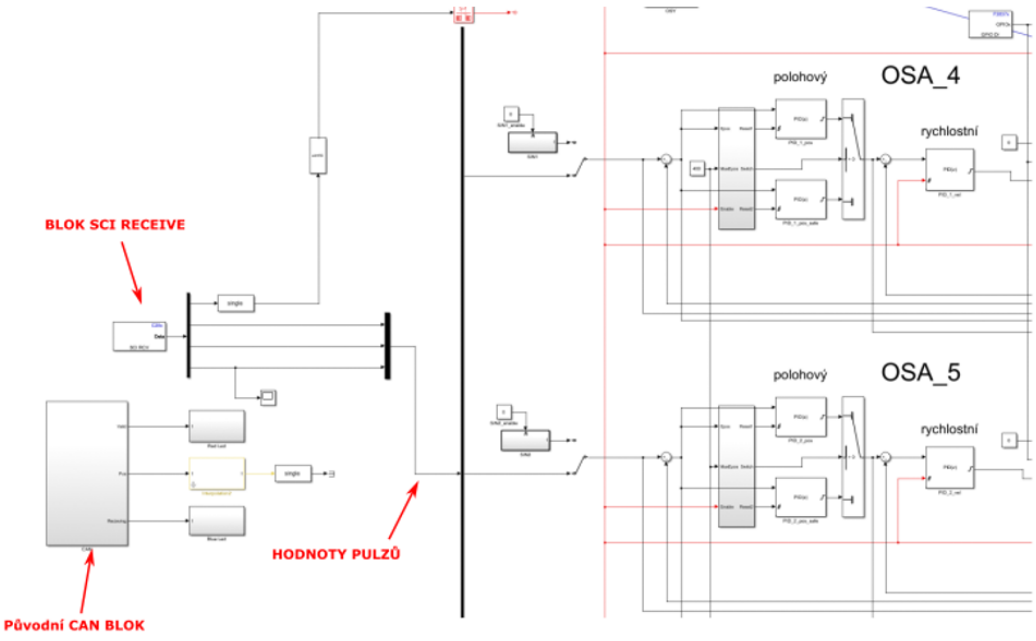
Obrázek 96 - Nastavení desky a baud rate sériové komunikace

Na obr. 97 lze vidět stav EtherCAT sběrnice a průběhy natočení na prvních třech osách při přepínání hodnot na displeji.



Obrázek 97 - Průběh pulzů na prvních třech osách a stav EtherCAT sběrnice

V původním řídícím programu nahradíme blokem *SCI Receive* původní blok *CAN*, pomocí kterého se přijímala data přes CAN. Bylo také nutné změnit GPIO u encoderů na původní hodnoty v nastavení modelu, jelikož se tyto hodnoty změnily s aktualizací na MATLAB 2022a. Nový model lze vidět na obr. 98.



Obrázek 98 - Úprava původního programu pro přijímání dat pomocí SCI

5 Závěr

V práci byla nejprve provedena rešerše průmyslových sběrnic používaných v automatizaci, a to s ohledem na real-time řízení. Snaha byla u každé sběrnice popsat princip komunikace mezi master a slave zařízeními a funkci jednotlivých vrstev modelu ISO/OSI. Podrobněji byl popsán princip EtherCAT sběrnice, která je obsahem této práce.

V další části této práce byly popisovány jednotlivé hardwarové prvky, pomocí kterých bude realizována nová komunikace s robotem PUMA200. Volba hardwaru závisela především na dostupnosti knihoven pro SPI komunikaci s *Teensy*.

V práci jsou dále popisovány jednotlivé bloky systémového návrhu. V prvním bloku systémového návrhu nejdříve je popsána konfigurace EtherCAT slave zařízení ve formě modulu *EasyCAT PRO*. Jedním z úkolů této práce bylo vytvoření HMI pomocí displeje, který je diskutovaný v následujícím bloku systémového návrhu. Displej komunikoval s *Teensy* správně a pulzy přijaté sériovou komunikací na straně platforem *F28379D* odpovídaly hodnotám na displeji. Ve třetí části bloku je stručně popsán program pro *Teensy* napsaný v prostředí Arduino IDE, který zajišťuje komunikaci mezi modulem *EasyCAT PRO*, displejem a platformami *F28379D*. Poslední blok se zabývá softwarem na straně PC. Nejdříve je popsán model vytvořený v programu MATLAB/Simulink, který pracuje se vstupními hodnotami z displeje a zařizuje inverzní kinematiku. Poté je popsán software TwinCAT 3, který figuruje jako master na EtherCAT sběrnici. Díky možnosti integrace Simulinkového modelu do prostředí TwinCAT je možné vypočítat inverzní kinematiku a zpracovávat data real-time s daným taktem 1 ms.

Výsledkem byla realizace Implementace EtherCAT sběrnice. Výsledek této práce ovšem nesplnil očekávání. Předávání dat do platforem *F28379D* přes sériovou komunikaci SCI bylo bohužel žalostné. Osy robota se sice podařilo pomocí nového řídícího systému rozpohybovat a dostat do žádané polohy, avšak pohyb byl trhaný a chyběla mu plynulost. Z časových důvodů již nebylo možné tento problém vyřešit. Důvodů může být několik. Nejpravděpodobnějším problémem je zřejmě asynchronnost sériové komunikace. Z práce je zřejmé, že se data posílají přes mnoho sběrnic. Nejdříve je potřeba hodnoty z displeje poslat přes sběrnici SPI do *Teensy*. *Teensy* poté pošle data přes SPI do modulu *EasyCAT PRO*, který si vyměňuje data skrz sběrnici EtherCAT s TwinCATem. Hodnoty se pak musejí znovu zaslat Přes SPI do *Teensy* a teprve poté se přes sériovou komunikaci SCI dostanou do platforem *F28379D*. Optimální by bylo implementovat EtherCAT sběrnici přímo na platformy *F28379D*, což bohužel nebylo možné kvůli neexistující softwarové podpoře EtherCAT v programu MATLAB/Simulink.

Problémy s předáváním dat pomocí sériového portu je v plánu do budoucna eliminovat. Další zdokonalování řídícího systému však touto prací skončit nemusí. Řídící systém postrádá například kontrolu přejíždění do singulárních poloh. Další možností je zařídit natáčení os robota pomocí natáčení displeje – například pomocí gyroskopu. Existuje mnoho dalších vylepšení, která jsou omezena pouze lidskou představivostí.

6 Použitá literatura

[1] *OSI model layers functions and protocols – BytesofGigabytes* [online]. [cit. 2022-07-22]. Dostupné z: https://bytesofgigabytes.com/networking/osi-model/

[2] *Jiří Peterka: Referenční model ISO/OSI - sedm vrstev* [online]. [cit. 2022-07-22]. Dostupné z: https://www.earchiv.cz/a92/a213c110.php3

[3] POWERS, Nick. *Industrial Protocols Comparison: IIoT Industrial Networks | Arrow.com* [online]. [cit. 2022-07-22]. Dostupné z: https://www.arrow.com/en/research-and-events/articles/industrial-connectivity-protocols

[4] *Sběrnice CAN* [online]. [cit. 2022-07-22]. Dostupné z: http://www.elektrorevue.cz/clanky/03021/index.html?fbclid=IwAR3vTBlY1yK5xnjngu5biWRXwpw4k9XDzEm8foAZRTPO7UKbNVUU-A265-4

[5] COOK, J A a J S FREUDENBERG. Controller Area Network (CAN) EECS 461, Fall 2008 * [online]. [cit. 2022-07-22]. Dostupné z: https://www.eecs.umich.edu/courses/eecs461/doc/CAN_notes.pdf

[6] ALSHAMMARI, Abdulaziz, Mohamed A. ZOHDY, Debatosh DEBNATH a George CORSER. Classification Approach for Intrusion Detection in Vehicle Systems. *Wireless Engineering and Technology* [online]. 2018, 9(4), 79–94. ISSN 2152-2294. Dostupné z: doi:10.4236/WET.2018.94007

[7] *CAN Protocols | Bosch Semiconductors for Automotive* [online]. [cit. 2022-07-22]. Dostupné z: https://www.bosch-semiconductors.com/ip-modules/can-protocols/

[8] SEN, Sunit Kumar. *Fieldbus and Networking in Process Automation* [online]. CRC Press, 2017. ISBN 9781315215792. Dostupné z: doi:10.1201/b16891

[9] *Průmyslová sběrnice Profibus | Vývoj.HW.cz* [online]. [cit. 2022-07-23]. Dostupné z: https://vyvoj.hw.cz//navrh-obvodu/rozhrani/prumyslova-sbernice-profibus.html

[10] *Technický popis - Přehled* [online]. [cit. 2022-07-23]. Dostupné z: http://www1.fs.cvut.cz/cz/U12110/site/profibus/PROFIBUS-uvod-cz.htm

[11] KŘÍBSKÝ, Petr. *Využití moderních průmyslových sběrnic v náročných prostředích*. Plzeň, 2016. Disertační práce. Západočeská univerzita v Plzni, Fakulta elektrotechnická. Vedoucí práce Doc. Ing. Jiří Skála, Ph.D.

[12] CC-Link Family-compatible Products [online]. [cit. 2022-07-23]. Dostupné z: https://www.cc-link.org/en/material/documents/cc1103_18_I.pdf

[13] *CC-link vzdělávací kurz* [online]. [cit. 2022-07-23]. Dostupné z: https://www.mitsubishielectric.com/fa/assist/e-learning/pdf/cze/1-CC-Link_fod_cze.pdf?fbclid=IwAR1xZ1XhHe8LxpFrk-mOP8dFycRbA5IwXAbCBcIGkNig33WIF7Rwkjh9bD4

[14] SCHIFFER, Viktor. Common industrial protocol (CIP™) and the family of CIP networks. In: *Industrial Communication Technology Handbook, Second Edition* [online]. 2017 [cit. 2022-07-23]. ISBN 9781482207330. Dostupné z: doi:10.1201/b17365

[15] *Časopis Automa: Průmyslový Ethernet IX: EtherNet/IP, EtherCAT* [online]. [cit. 2022-07-23]. Dostupné z: https://automa.cz/cz/casopis-clanky/prumyslovy-ethernet-

ix-ethernet/ip-ethercat-2008__10__37910__6510/

[16] REYNDERS, Deon., Steve. MACKAY a E. WRIGHT. *Practical industrial data communications : best practice techniques*. Elsevier, 2005. ISBN 9780750663953.

[17] *What Is the Difference Between Ethernet and Industrial Ethernet? | Analog Devices* [online]. [cit. 2022-07-23]. Dostupné z: https://www.analog.com/en/technical-articles/what-is-the-difference-between-ethernet-and-industrial-ethernet.html

[18] *EtherNet/IP™ | ODVA Technologies | Industrial Automation* [online]. [cit. 2022-07-23]. Dostupné z: https://www.odva.org/technology-standards/key-technologies/ethernet-ip/

[19] *Časopis Automa: Průmyslový Ethernet VIII: Ethernet Powerlink, Profinet* [online]. [cit. 2022-07-23]. Dostupné z: https://automa.cz/cz/casopis-clanky/prumyslovy-ethernet-viii-ethernet-powerlink-profinet-2008__05__37288__6341/

[20] *POWERLINK basics* [online]. [cit. 2022-07-23]. Dostupné z: https://www.ethernet-powerlink.org/fileadmin/user__upload/Dokumente/Dokumente/PL__FLY__Basics__en__WEB.pdf

[21] *Časopis Automa: Komunikačný systém Profinet IO* [online]. [cit. 2022-07-23]. Dostupné z: https://automa.cz/cz/casopis-clanky/komunikacny-system-profinet-io-2006__07__31231__560/?fbclid=IwAR3tfn8UPPSBaCdmQaq--Ce0zqHw7Faw3o4gSqyao9k4wu__rSIeJGYaTYNA

[22] *ICP DAS* [online]. [cit. 2022-07-23]. Dostupné z: https://www.icpdas.com/en/product/guide+Industrial__Communication+Fieldbus__Communication+PROFINET

[23] SEN, Sunit Kumar. *Fieldbus and Networking in Process Automation - 2nd edition* [online]. CRC Press, 2021. ISBN 9781003149941. Dostupné z: doi:10.1201/9781003149941

[24] PROFINET System Description [online]. [cit. 2022-07-23]. Dostupné z: https://us.profinet.com/wp-content/uploads/2012/11/PROFINET__SystemDescription__ENG__2014__web.pdf?fbclid=IwAR34MwXWGwMAeNTptLH8CivFzVonCZZH2LmLfAWNf2N5TKLFmbE8MJGu__uo

[25] *plug & play - Sercos, the automation bus* [online]. [cit. 2022-07-23]. Dostupné z: https://www.sercos.org/downloads/brochures/?tx__vdsercosdownloads__downloads%5Bfile%5D=Sercos__Sercos__III__Broschuere__2016__EN__US__V02__web.pdf&tx__vdsercosdownloads__downloads%5Baction%5D=download&tx__vdsercosdownloads__downloads%5Bcontroller%5D=File&cHash=584b637215724278782d4ba6b3f5cfc2

[26] *EtherCAT Protocol: Introduction to EtherCAT | TI.com Video* [online]. [cit. 2022-07-23]. Dostupné z: https://training.ti.com/ethercat-protocol-introduction-ethercat?context=1137766-1147233-1147212

[27] SMOŁKA, Ireneusz a Jacek STÓJ. Utilization of SDN Technology for Flexible EtherCAT Networks Applications. *Sensors 2022, Vol. 22, Page 1944* [online]. 2022, 22(5), 1944 [cit. 2022-07-23]. ISSN 1424-8220. Dostupné z: doi:10.3390/S22051944

[28] HADÁMEK, Pavel. *SLAVE MODUL VYUŽÍVAJÍCÍ KOMUNIKAČNÍ PROTOKOL ETHERCAT.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta Elektrotechniky a komunikačních technologií. Vedoucí práce Ing. Soběslav Valach.

[29] *What is EtherCAT (Ethernet for Control Automation)? | RealPars* [online]. [cit. 2022-07-23]. Dostupné z: https://realpars.com/ethercat/

[30] *EtherCAT Protocol: Feature Highlight - Distributed Clocks | TI.com Video* [online]. [cit. 2022-07-23]. Dostupné z: https://training.ti.com/ethercat-protocol-feature-highlight-distributed-clocks?context=1137766-1147233-1147213

[31] *EtherCAT Protocol: EtherCAT Slave Overview | TI.com Video* [online]. [cit. 2022-07-23]. Dostupné z: https://training.ti.com/ethercat-protocol-ethercat-slave-overview?context=1137766-1147233-1147214

[32] *Teensy® 4.0* [online]. [cit. 2022-08-09]. Dostupné z: https://www.pjrc.com/store/teensy40.html

[33] *EasyCAT PRO* [online]. [cit. 2022-08-09]. Dostupné z: https://www.bausano.net/en/hardware/easycat-pro.html

[34] *4.0" 480x320 TFT displej, ST7796, SPI, dotykový | LaskaKit* [online]. [cit. 2022-08-09]. Dostupné z: https://www.laskakit.cz/4-0--480x320-tft-displej--st7796--spi--dotykovy/

[35] VALÁŠEK, Michael. *Návrh a zprovoznění nového řídícího systému průmyslového robota PUMA200*. Praha, 2018. Diplomová práce. České vysoké učení technické v Praze, Fakulta strojní. Vedoucí práce Ing. Martin Nečas, MSc., Ph.D.

[36] *LAUNCHXL-F28379D Overview* [online]. [cit. 2022-08-09]. Dostupné z: https://www.ti.com/lit/ug/sprui77c/sprui77c.pdf?ts=1660005592433

[37] *BOOST-DRV8711 Evaluation board | TI.com* [online]. [cit. 2022-08-09]. Dostupné z: https://www.ti.com/tool/BOOST-DRV8711

[38] *GitHub - Bodmer/TFT_eSPI: Arduino and PlatformIO IDE compatible TFT library optimised for the Raspberry Pi Pico (RP2040), STM32, ESP8266 and ESP32 that supports different driver chips* [online]. [cit. 2022-08-09]. Dostupné z: https://github.com/Bodmer/TFT_eSPI

[39] *DMA for ILI9488 · Issue #642 · Bodmer/TFT_eSPI · GitHub* [online]. [cit. 2022-08-09]. Dostupné z: https://github.com/Bodmer/TFT_eSPI/issues/642?fbclid=IwAR0yvFDoqcNFqniPxnf3jkEOq-YJOQRDom3svVCItOI3mIb8g2zGvdBs7t4

[40] *SPI Tutorial – Serial Peripheral Interface Bus Protocol Basics* [online]. [cit. 2022-08-06]. Dostupné z: https://www.corelis.com/education/tutorials/spi-tutorial/

[41] *Basics of UART Communication* [online]. [cit. 2022-08-06]. Dostupné z: https://www.circuitbasics.com/basics-uart-communication/

[42] DVOŘÁK, Jindřich. *Řízení a kinematická kalibrace robota Stäubli RX-60*. rojní. Ved. Praha, 2021. Diplomová práce. České vysoké učení technické v Praze, Fakulta strojní. Vedoucí práce Ing. Martin Nečas, MSc., Ph.D.,

[43] KUKULA, Pavel a Michael VALASEK. Kinematical solution by structural approximation. *Proceedings of the 5th International Workshop on Computational Kinematics* [online]. 2009, 323–330. Dostupné z: doi:10.1007/978-3-642-01947-0_40

[44] HANSEN, John M. Kinematics and Dynamics of Machinery: Stejskal, Vladimir & Valášek, Michael: (Marcel Dekker, New York, 1996, 512 pages + diskette. ISBN 0-8247-9731-0). *European Journal of Mechanics - A/Solids* [online]. 1998, 17(4), 701–702. ISSN 0997-7538. Dostupné z: doi:10.1016/S0997-7538(99)80030-8

[45] *Computing Euler angles from a rotation matrix* [online]. [cit. 2022-08-09]. Dostupné z: https://mino-git.github.io/rtcw-wet-blender-model-tools/publications/EulerToMatrix.pdf

[46] ČEKALOVÁ, Adéla. *Řízení motoru s kvadraturním enkodérem prostřednictvím modulu Beckhoff EL7342*. Praha, 2020. Bakalářská práce. České vysoké učení technické v Praze, Fakulta strojní. Vedoucí práce Ing. Martin Nečas, MSc., Ph.D.

[47] *TwinCAT 3* [online]. [cit. 2022-08-09]. Dostupné z: https://download.beckhoff.com/download/document/catalog/TwinCAT__3__Booklet.pdf

[48] *TwinCAT 3 system requirements* [online]. [cit. 2022-08-09]. Dostupné z: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3__c/6778772363.html&id=8154196111104659877

[49] *TE1400 installation* [online]. [cit. 2022-09-08]. Dostupné z: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3__c/6778772363.html&id=8154196111104659877

7 Seznam obrázků

Obrázek 1 - Referenční model ISO/OSI [1] 9
Obrázek 2 - Datový rámec sběrnice CAN [6] 12
Obrázek 3 - Princip arbitrace sběrnice CAN [5] 12
Obrázek 4 - ISO/OSI model sběrnice PROFIBUS [8] 13
Obrázek 5 - Struktura sítě PROFIBUS DP s metodou přístupu pomocí tokenu [9] 14
Obrázek 6 - Schéma zasílání dat k externím vstupům a výstupům sběrnice CC-link [13] 16
Obrázek 7 - Objektový model zařízení s rozhraním CIP [15] 17
Obrázek 8 - Přístup na sběrnici ControlNet pomocí algoritmu CTDMA [14] 18
Obrázek 9 - Multicast I/O implicitní spojení [14] 19
Obrázek 10 - Ethernetový rámec [16] 20
Obrázek 11 - ISO/OSI model protokolu Ethernet/IP [18] 21
Obrázek 12 - Přenosový cyklus Ethernet Powerlink v horní části a multiplexing v dolní části [20] 22
Obrázek 13 - Komunikační model Ethernet Powerlink [19] 23
Obrázek 14 - Profinet IO komunikace [22] 25
Obrázek 15 - Komunikační cyklus PROFINET IRT [23] 25
Obrázek 16 - Aplikační a komunikační relace Profinetu IO [24] 26
Obrázek 17 - SERCOS III topologie, liniová vlevo a prstencová vpravo [25] 27
Obrázek 18 - Konfigurace komunikačního cyklu SERCOS III [25] 27
Obrázek 19 - Struktura telegramu SERCOS III [25] 28
Obrázek 20 - Tok dat v logické smyčce sbernice EtherCAT [27] 29
Obrázek 21 - Průchod paketu EtherCAT při prstencovém zapojením bez poruchy [28] 29
Obrázek 22 - Průchod paketu EtherCAT při prstencovém zapojením s poruchou [28] 29
Obrázek 23 - Struktura EtherCATového rámce [26] 30
Obrázek 24 - Slave zařízení zapisuje do EtherCAT datagramu „za letu“ [29] 30
Obrázek 25 - Slave zařízení čte ze stejného EtherCAT datagramu „za letu“ [29] 31
Obrázek 26 - Logické jádro EtherCAt slave zařízení [31] 32
Obrázek 27 - EtherCAT komunikace v mailbox módu (vlevo) a v buffer módu (vpravo) [31] 32
Obrázek 28 - Vývojová deska Teensy 4.0 [32] 34
Obrázek 29 - Schéma pinů na přední straně (vlevo) a na zadní straně (vpravo) desky Teensy 4.0 [32] 35
Obrázek 30 - Modul EasyCAT PRO [33] 35
Obrázek 31 - Schéma pinů modulu EasyCAT PRO [33] 36
Obrázek 32 - Čtyřpalcový dotykový displej LaskaKit [34] 36
Obrázek 33 - Zadní strana dotykového displeje s popisem pinů 37
Obrázek 34 - LaunchPad F28379D 38
Obrázek 35 - Schéma možností zapojení pinů F28379D [36] 38
Obrázek 36 - Výkonový modul DRV8711 [37] 39
Obrázek 37 - Systémový návrh k řízení robota PUMA200 39
Obrázek 38 - Konfigurátor pro EasyCAT PRO 40
Obrázek 39 - Displej s kartézskými souřadnicemi (vlevo) a motorovými souřadnicemi (vpravo) 41
Obrázek 40 - Přepočet mezi souřadnicovými systémy na displeji 42
Obrázek 41 - Nastavení driveru ILI9486 43
Obrázek 42 - Nastavení pinů pro SPI komunikaci s displejem 43
Obrázek 43 - Nastavení frekvence SPI pro renderování obrazu a frekvence SPI pro dotyk 43

Obrázek 44 - Návrh držáku displeje v programu Autodesk Inventor 44
Obrázek 45 - Hotový výtisk držáku displeje 44
Obrázek 46 - Schéma zapojení jednotlivých komponent k Teensy 45
Obrázek 47 - Reálné zapojení všech komponent k Teensy v laboratoři 46
Obrázek 48 - Slave zařízení připojená k masterovi na SPI sběrnici pomocí čtyř linek [40] 46
Obrázek 49 - Módy SPI sběrnice 47
Obrázek 50 - Zápis (nahoře) a čtení (dole) dat na sběrnici SPI [35] 48
Obrázek 51 - Bity přicházející paralelně se pošlou sériově přes linku TX na linku RX, kde se zpět konvergují na paralelní data [41] 48
Obrázek 52 - Datový paket poslaný po lince TX [41] 48
Obrázek 53 - Příklad definovaných konstant týkajících se velikosti a pozice tlačítek (vlevo) a důležité proměnné používané v programu (vpravo) 50
Obrázek 54 - Ukázka části kódu funkce checkforswitches 51
Obrázek 55 - Ukázka části kódu funkce checkforcartesian 51
Obrázek 56 - Ukázka části kódu funkce checkmanualset 52
Obrázek 57 - Uložení kartézských souřadnic do vstupních proměnných modulu EasyCAT PRO 53
Obrázek 58 - Uložení výstupních proměnných z TwinCATu přes modul EasyCAT PRO do proměnných v Teensy 53
Obrázek 59 - Ukázka kódu kontrolující stav na EtherCAT sběrnici 53
Obrázek 60 - Konverze floatových dat na pole bytů a zaslání přes SCI do první platformy F28379D 54
Obrázek 61 - Model v Simulinku 56
Obrázek 62 - Vstup kartézských souřadnic z displeje 57
Obrázek 63 - Počáteční poloha robota v Simscape Multibody 58
Obrázek 64 - vstup do inverzní kinematiky bez použití bloku rate limiter (vlevo) a s použitím bloku rate limiter (vpravo) 58
Obrázek 65 - Vstup motorových souřadnic v Simulinku 58
Obrázek 66 - Funkční blok počítající inverzní kinematiku 59
Obrázek 67 - Proměnná State posílaná z TwinCATu do Teensy 59
Obrázek 68 - Blok zobrazující funkci přepínání souřadnicových systémů a zapnutí shaperů 60
Obrázek 69 - Blok vyobrazující výstupy z modelu v Simulinku 61
Obrázek 70 - Blok řešící dopřednou kinematiku a zaslání souřadnic do Teensy 61
Obrázek 71 - Zavedení souřadnicových systémů na robotovi 62
Obrázek 72 - Kód určující úhly natočení koncového efektoru 64
Obrázek 73 - Nezpožděné hodnoty natočení os zaslané do Teensy 65
Obrázek 74 - Inženýrské prostředí TwinCAT 3 [46] 66
Obrázek 75 - Modulární struktura běhového prostředí TwinCAT 3 [46] 67
Obrázek 76 - Nastavené parametry záložce Solver 68
Obrázek 77 - Nastavení parametru v záložce Code Generation 68
Obrázek 78 - Nový projekt v TwinCATu 68
Obrázek 79 - Načtený modul EasyCAT PRO v prostředí TwinCAT s nakonfigurovanými proměnnými 69
Obrázek 80 - Nastavení testovacího módu Windows 69
Obrázek 81 - Testovací režim Windows 69
Obrázek 82 - Okno Software Protection 70
Obrázek 83 - Vyplněné inforamce v okně Software Protection 70

Obrázek 84 - Vytvořený certifikát....71
Obrázek 85 - Příkaz v CMD pro podepsání vytvořeného TwinCAT objektu....71
Obrázek 86 - Úspěšné podepsání certifikátem....71
Obrázek 87 - Přidání TcCOM modulu do TwinCATu....72
Obrázek 88 - Vstupy a výstupy vytvořené pomocí TwinCAT bloků v Simulinku....72
Obrázek 89 – Slinková+ní proměnných v Simulinku a EasyCAT PRO....73
Obrázek 90 - Zvolení taktu 1 ms pro zpracování dat real-time....73
Obrázek 91 - Slinkování TcCOM objektu s vytvořenou úlohou....73
Obrázek 92 - Spuštění konfigurace....74
Obrázek 93 - Model Simulinku integrovaný v prostředí TwinCAT....74
Obrázek 94 – Model v Simulinku nahraný do platformy F28379D pro sledování přijatých dat z Teensy....74
Obrázek 95 - Nastavení bloku SCI Receive....75
Obrázek 96 - Nastavení desky a baud rate sériové komunikace....75
Obrázek 97 - Průběh pulzů na prvních třech osách a stav EtherCAT sběrnice....76
Obrázek 98 - Úprava původního programu pro přijímání dat pomocí SCI....76

8 Přílohy práce

Na přiloženém DVD nalezneme následující soubory:

- PDF soubor s texem bakalářské práce
- Adresář *Simulink*, který obsahuje Simulinkový model z části 4.2.4
- Adresář *AktualizovanyProgram*, který obsahuje program Ing. Michaela Valáška adaptovaný na sériovou komunikaci
- Adresář *Program_BP*, který obsahuje program pro *Teensy* napsaný v Arduinu IDE a potřebné knihovny
- Adresář *EasyCAT*, který obsahuje soubory vytvořené konfigurátorem pro modul *EasyCAT PRO*
- Adresář *TwinCAT*, který obsahuje vytvořený projekt v TwinCATu