
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS


# VYTVOŘENÍ KNIHOVNY DO SIMULINKU PRO MĚŘICÍ ZAŘÍZENÍ

CREATING A LIBRARY IN SIMULINK FOR MEASURING DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS


AUTOR PRÁCE — Kryštof Petr
AUTHOR

VEDOUCÍ PRÁCE — Ing. Martin Appel
SUPERVISOR



BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

# Zadání bakalářské práce

| | |
|---|---|
| Ústav: | Ústav mechaniky těles, mechatroniky a biomechaniky |
| Student: | **Kryštof Petr** |
| Studijní program: | Aplikované vědy v inženýrství |
| Studijní obor: | Mechatronika |
| Vedoucí práce: | **Ing. Martin Appel** |
| Akademický rok: | 2021/22 |

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

## Vytvoření knihovny do Simulinku pro měřicí zařízení

**Stručná charakteristika problematiky úkolu:**

Cílem této práce bude vytvoření knihovny do Simulinku. Tato knihovna bude obsahovat řadu bločků, které umožní čtení a zápis do připojené elektronické jednotky přes USB. Jedná se o měřicí a řídicí jednotku, která má řadu vstupních a výstupních periferií. Tato jednotka je již funkční a má definovanou komunikaci s počítačem. Komunikace využívá výhod čipu FTDI a to snadnou identifikaci zařízení a navázaní komunikace. Součástí této práce bude i testování a vytvoření ukázkových úloh.

**Cíle bakalářské práce:**

1. Seznámit se se současným stavem programu
2. Vytvoření knihovny do Simulinku pro čtení a zápis na veškeré periferie zařízení
3. Změření rychlosti komunikace pro různě komplikované modely
4. Vytvoření několika ukázkových úloh včetně podrobného návodu pro studenty

**Seznam doporučené literatury:**

CORKE, Peter I. Robotics, vision and control: fundamental algorithms in MATLAB. Berlin: Springer, 2011. Springer tracts in advanced robotics, v. 73. ISBN 9783642201448.

GREPL, Robert. Kinematika a dynamika mechatronických systémů. Brno: Akademické nakladatelství CERM, 2007. ISBN 978-80-214-3530-8.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2021/22

V Brně, dne

L. S.

| | |
|---|---|
| prof. Ing. Jindřich Petruška, CSc.<br>ředitel ústavu | doc. Ing. Jaroslav Katolický, Ph.D.<br>děkan fakulty |

# ABSTRAKT

Tato práce se zabývá tvorbou knihovny do Simulinku pro komunikaci s cílovým zařízením. V teoretické části jsou popsány různé možnosti, kterými by šla komunikace vyřešit. V praktické části jsou pak výsledky jejich vzájemného porovnání a dále varianty real-time synchronizace simulace. Výstupem práce je knihovna do Simulinku a také soubor ukázkových úloh pro její snadnější pochopení.

**Klíčová slova**

Simulink, MATLAB, komunikace, knihovna, FTDI čip, real-time synchronizace

# ABSTRACT

This thesis deals with the creation of a library in Simulink for communication with the target device. The theoretical part describes the various options by which communication could be solved. In the practical part are the results of their mutual comparison and also variants of real-time synchronization of the simulation. The output of this work is a library for Simulink and also a set of sample tasks for easier understanding.

**Key words**

Simulink, MATLAB, communication, library, FTDI chip, real-time synchronization

## BIBLIOGRAFICKÁ CITACE

PETR, Kryštof. *Vytvoření knihovny do Simulinku pro měřicí zařízení.* Brno, 2022. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/139611. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Martin Appel.

## PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou práci na téma **Vytvoření knihovny do Simulinku pro měřicí zařízení** vypracoval samostatně s použitím odborné literatury a pramenů, uvedených v seznamu, který tvoří přílohu této práce.

Brno …… ……..…

Kryštof Petr

## PODĚKOVÁNÍ

Chci poděkovat Ing. Martinu Appelovi za jeho trpělivost a za cenné rady, které mi poskytl při vypracování této závěrečné práce. Děkuji rovněž Ing. Martinu Formánkovi za pomoc při pochopení jeho práce.

# OBSAH

# 1 Úvod

Elektronická komunikace je nezbytnou součástí všech mechatronických systémů, ať už se jedná o čtení dat ze senzorů, nebo řízení aktuátorů. Důležitým parametrem komunikace je její rychlost, protože umožňuje jednak přepravu většího objemu dat, ale také lepší možnost regulace.
Komunikaci také využíváme při odesílání dat do počítače pro jejich následné vyhodnocení. Díky tomu lze využít dobré uživatelské rozhraní a větší výpočetní výkon s použitím programů jako je například MATLAB/Simulink.

Tato práce se zabývá komunikací se zařízením Mechatronics Starter Pack, které je určené pro použití při výuce. Uplatnění může nalézt zejména v předmětech RDO (Modelování a simulace) nebo RIR (Inteligentní řídicí systémy). Zde studentům poskytne možnost praktického vyzkoušení navrženého PID regulátoru včetně vizuálního povědomí o efektech změn jeho jednotlivých parametrů. Další typickou úlohou je filtrace signálu ze senzorů nebo jednoduché úlohy, jako řízení LED pomocí tlačítka, pro osvojení principů programování v Simulinku.

Zařízení MSP komunikuje přes USB. Výhodou toho je zejména univerzalita, protože dnes nalezneme USB port na každém počítači. Toto připojení je jednoduché a nevyžaduje žádný speciální adaptér typu RS-232 karta. Komunikaci přes USB umožňuje čip FT2232HL od FTDI umístěný přímo na desce.

Jako uživatelské rozhraní pro komunikaci s MSP zařízením slouží knihovna v Simulinku. Simulink je uživatelsky příjemný grafický programovací jazyk. Je vhodným nástrojem pro studenty, protože se používá nejen v akademické sféře, ale i ve výzkumu a v průmyslu. Snadno a přehledně se v něm dají vytvářet regulační smyčky nebo kaskádní regulace. Dokáže také simulovat nelineární systémy a nabízí velké množství toolboxů.

Cílem této práce bylo navázat na diplomovou práci Ing. Martina Formánka, nalézt optimální způsob komunikace s cílovým zařízením a následně jej využít při tvorbě knihovny.

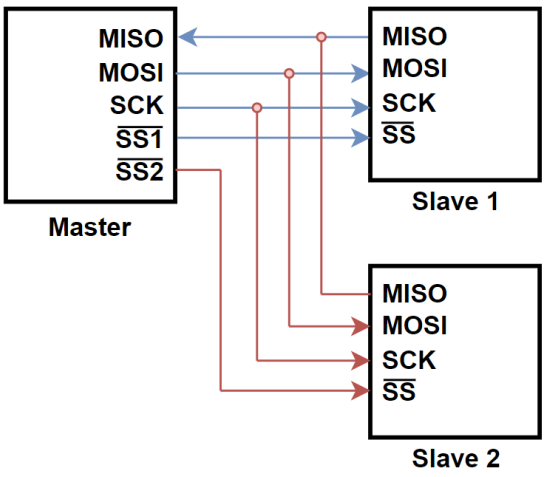
# 2 Rešerše

## 2.1 Komunikační protokoly

Protokol je standard, kterým se řídí elektronická komunikace. Jedním ze základních dělení je na synchronní, které používají hodinový signál pro synchronizaci přenosu, a asynchronní bez hodinového signálu.
Komunikační protokoly se také dělí podle přenosového režimu. Spojení, ve kterém mohou data putovat pouze v jednom směru se označuje jako simplexní, pokud je možné posílání v obou směrech, ale ne současně, jedná se o poloviční duplex (half-duplex), a při komunikaci oběma směry zároveň mluvíme o tzv. plném duplexu (full-duplex). Dalším typem dělení je podle typu přenosu na paralelní a sériové. Při paralelním přenosu je posíláno více bitů najednou po několika vodičích, zatímco při sériové komunikaci jsou data posílána po bitech v řadě za sebou. Mezi nejběžnější sériové protokoly patří SPI, $I^2C$ a UART. [1]

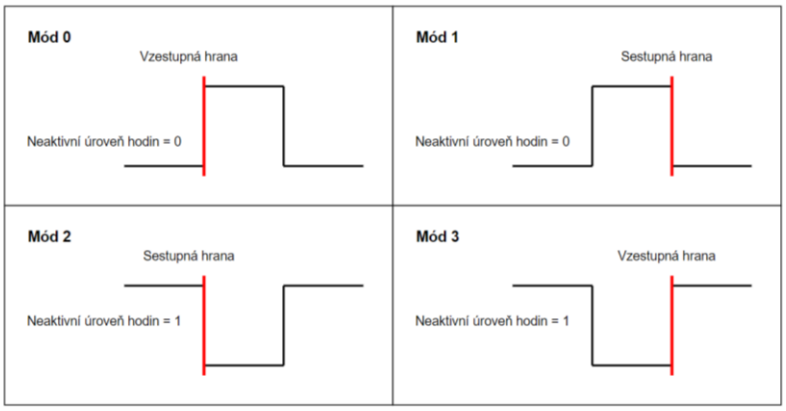
### 2.1.1 SPI

Serial Peripheral Interface je synchronní protokol. Hodinový signál je na výstupu SCK. Pro zápis Master využívá pin MOSI (Master Out, Slave In), a pro čtení pin MISO (Master In, Slave Out), viz obrázek 2.1. Díky oddělení těchto signálů na dvou různých vodičích je SPI plně duplexní.
SS (Slave Select), někdy také CS (Chip Select), slouží k výběru Slave zařízení. Tento pin používá invertovanou logiku, tzn. že Slave je zvolen, pokud je SS ve stavu logické 0. Teoreticky je možné, aby jeden Master měl neomezené množství Slave zařízení, ale protože každý Slave má svoji vlastní SS linku, narůstá počet vodičů.
Master zahájí komunikaci výběrem Slave zařízení nastavením odpovídajícího SS pinu do stavu logické 0. Pak může začít přenos dat v jednom nebo obou směrech. Čtení dat může probíhat ve čtyřech módech, v závislosti na polaritě a fázi hodinového signálu, jak je vidět na obrázku 2.2. Po skončení přenosu je SS pin nastaven zpět do logické 1.
SPI se využívá zejména tam, kde je podstatná rychlost komunikace. Příkladem mohou být SD karty, displeje nebo komunikace mezi mikrokontrolery. [2]




**Obrázek 2.1** Schéma zapojení SPI

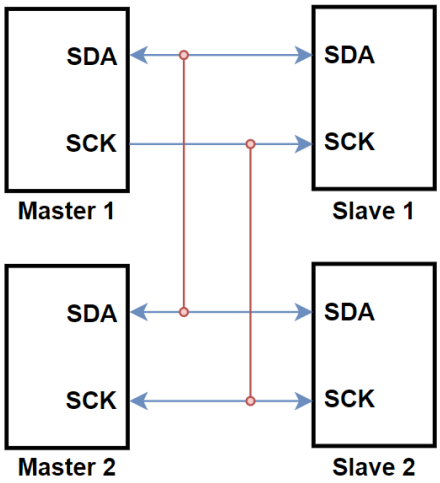
| Mód | Polarita | Fáze |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |




**Obrázek 2.2** SPI módy

### 2.1.2 I²C

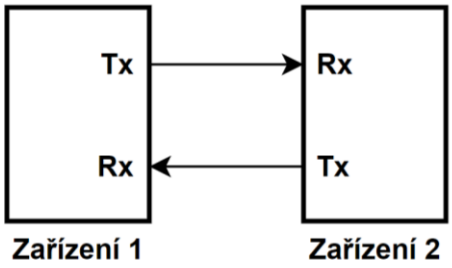
Název pochází z anglického Inter-Integrated Circuit. Jde o sériový protokol, jehož hlavními výhodami jsou zaprvé možnost integrace více Master zařízení do stejné komunikační sítě a zadruhé spojení velkého množství zařízení za využití pouze dvou datových linek, jak je znázorněno na obrázku 2.3. Těmito linkami jsou SCK, což je hodinový signál, a SDA neboli Serial Data. SDA slouží nejen pro posílání dat, ale také pro volbu cílového zařízení. Každému zařízení je přidělena 7bitová adresa. Pokud by bylo zapotřebí spojit více než 128 zařízení, je možné použít 10bitový adresový systém. Protože je SDA linka společná, je tento protokol pouze poloviční duplexní.
Každé Master zařízení generuje vlastní hodinový signál a tyto signály je nutné před zahájením komunikace navzájem synchronizovat. Vlastní komunikace probíhá tak, že jeden z Masterů pošle na společnou SDA linku adresu zařízení, se kterým chce komunikovat, a také bit, kterým sdělí, jestli bude přijímat, nebo vysílat. Příjemce potvrdí navázání komunikace posláním ACK bitu. Příjemce také posílá ACK bit po každých 8 bitech zprávy jako signalizaci, že data v pořádku dorazila. Pro ukončení komunikace pošle Master stop signál.
I²C se typicky využívá pro komunikaci se senzory, A/D nebo D/A převodníky a dalšími periferiemi. Kvůli adresování a ACK bitům je složitější a pomalejší, než SPI. [2] [3]




**Obrázek 2.3** Schéma zapojení I²C

### 2.1.3 UART

UART, celým názvem Universal Asynchronous Receiver/Transmitter, je, jak už z názvu vyplývá, asynchronní. Protože nemá hodinový signál, obě zařízení musí mít nastavenou stejnou přenosovou rychlost (angl. baud rate). Pokud by tomu tak nebylo, může dojít k přetečení bufferu a ztrátě dat.
Používá se ke komunikaci mezi dvěma zařízeními. Každé z nich má dvě datové linky, TX pin pro zápis dat a RX pin pro příjem, přičemž TX pin prvního z nich je připojen na RX pin druhého, a naopak, znázorněno na obrázku 2.4. UART je proto plně duplexní.
Komunikace je zahájena posláním start-bitu, pak je odeslána vlastní zpráva a následuje stop-bit. Data jsou posílána po datových blocích, z nichž každý se skládá z 1 start-bitu, 8bitové zprávy, včetně paritního bitu pro kontrolu správnosti, a 1 nebo 2 stop-bitů.
UART se využívá například pro komunikaci mezi počítačem a externím zařízením. [3]




**Obrázek 2.4** Schéma zapojení UART

| **Protokol** | SPI | $I^2C$ | UART |
|---|---|---|---|
| **Rychlost** | až 20 Mbps | až 3,4 Mbps | max. 230-460 kbps |
| **Duplex (přenosový režim)** | Full-duplex | Half-duplex | Full-duplex |
| **Celkový počet zařízení** | teoreticky neomezený, ale narůstá počet vodičů | až 128 (nebo 1024) | 2 |
| **Počet Master/Slave zařízení** | Master: 1<br>Slave: 1 i více | Master: 1 i více<br>Slave: 1 i více | Master: 1<br>Slave: 1 |
| **Typ komunikace** | synchronní | synchronní | asynchronní |

**Tabulka 2.1** Porovnání komunikačních protokolů [2] [3]

### 2.1.4 Porovnání a shrnutí

V tabulce 2.1 je vzájemné porovnání protokolů. Účelem této práce je komunikace s jedním cílovým zařízením, s důrazem na rychlost a stabilitu komunikace. SPI je pro tuto aplikaci dobrou volbou nejen díky vysoké rychlosti, ale synchronizace hodinovým signálem je vhodná kvůli stabilitě. Zároveň nabízí do budoucna snadnou možnost přidání dalšího zařízení.

## 2.2 Periferie

Níže jsou základní informace o periferiích použitých na desce MSP, jejich označení, parametry apod. Další informace je možné nalézt v datasheetech, které jsou umístěné v seznamu literatury.

Do Simulinku přijdou hodnoty rozdělené do bytů s datovým typem *uint8*. Ty je potřeba upravit, aby na výstupu bloku byla konečná hodnota v požadovaných jednotkách s datovým typem *double*. K tomu slouží přepočtové vzorce, které vycházejí nejen z vlastností samotných senzorů, ale také z obvodového zapojení na DPS. Přepočetní vzorce jsou v přílohách na konci této práce.

### 2.2.1 DC motor s enkodérem

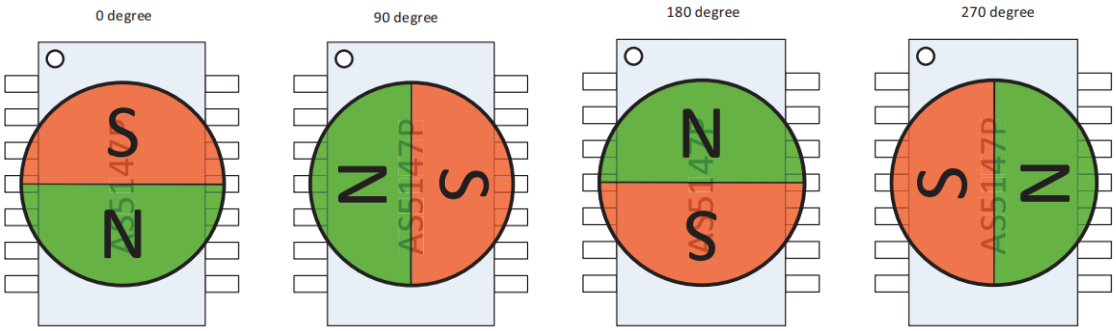
Tento DC motor od společnosti Pololu Robotics and Electronics s produktovým číslem 4883 má na hřídeli integrovaný kvadraturní enkodér. Rozlišení enkodéru je 48 pulzů na otáčku (CPR). Motor má převodovku s převodovým poměrem 20,4. Napájecí napětí motoru je 12 V. Při zatížení na maximální účinnost má motor rychlost 230 otáček za minutu, moment přibližně 34,3 mNm a výkon 0,84 W. [4]

### 2.2.2 Čidlo proudu

Senzor TMCS1100A4 k měření proudu využívá Hallova jevu. Vstupní proud protéká vnitřním vodičem s odporem 1,8 mΩ, což generuje magnetické pole. Intenzitu tohoto pole měří vnitřní Hallův senzor. Jedná se o lineární čidlo s vysokou přesností ±0,4 %. Umožňuje měřit stejnosměrný i střídavý proud v rozsahu od -5,75 A do 12 A. [5]

### 2.2.3 Magnetický enkodér pro setrvačník

Magnetický enkodér AS5147P-HTSM s vysokým rozlišením také využívá k měření Hallův jev. Měří úhlové natočení magnetu nad tímto senzorem (obrázek 2.5), který se v případě MSP zařízení otáčí zároveň se setrvačníkem. Rozlišení je možné nastavit na 1024, 512 nebo 256 pulzů na otáčku. Robustní design potlačuje vliv vnějšího homogenního magnetického pole. [6]




**Obrázek 2.5** Detekce úhlu natočení magnetickým senzorem [6]

### 2.2.4 Teplotní čidlo

Senzor s označením MCP9700T-E/LT využívá lineární obvod s termistorem. Dokáže měřit teplotu v rozmezí od -40 °C do +125 °C s přesností ±4 °C. Poskytuje levné řešení pro aplikace, kde je potřeba měřit relativní změnu teploty. Při měření relativní změny teploty v intervalu 0 °C až +70 °C je přesnost čidla ±1 °C. [7]

### 2.2.5 Akcelerometr, gyroskop

Senzor BMX160 obsahuje akcelerometr, gyroskop a magnetometr, které měří data ve všech třech osách X, Y, Z. Rozlišení akcelerometru je nastavitelné na hodnoty ±2 g, ±4 g, ±8 g, ±16 g. Rozlišení gyroskopu lze přepínat mezi hodnotami 125, 250, 500, 1000 a 2000 °/s. [8]

## 2.3 FTDI čip

Cílové zařízení využívá čip FT2232HL od společnosti Future Technology Devices International Limited (FTDI). Tento čip obsahuje 2 bloky MPSSE (Multi-Protocol Synchronous Serial Engine), které umožňují snadné propojení synchronních sériových zařízení s USB portem. Je tedy možné s cílovým zařízením komunikovat protokoly jako SPI, $I^2C$ nebo JTAG přes USB port. Čip FT2232HL poskytuje také možnost využít protokol UART. [9]

MPSSE blok není součástí každého FTDI čipu, ale je to specifikum vybraných čipů z FT-série, například FT2232D, FT2232H nebo FT4232H. Jiné čipy, kupříkladu FT232RL, umožňují komunikaci přes USB pomocí UART protokolu, ale nikoliv SPI nebo $I^2C$. [10]

Pro používání čipu je zapotřebí nainstalovat ovladač D2XX, který umožňuje přímý přístup k USB zařízení prostřednictvím DLL knihovny. Tuto knihovnu je možné volat přímo z MATLABu. Obsahuje například funkce pro otevření komunikačního kanálu, čtení a odesílání dat nebo zavření kanálu po ukončení komunikace. [11]

## 2.4 Implementace dalších jazyků s nástroji Simulink

Toto téma je důležité proto, že se tím otevírají například možnosti využívání knihoven pro jiné programovací jazyky. Jiné jazyky také mohou být lépe uzpůsobené pro některé aplikace. Python, Java a další jazyky rovněž nabízejí možnost volání funkcí napsaných v MATLABu, ale tím se v této kapitole nezabývám. [12]

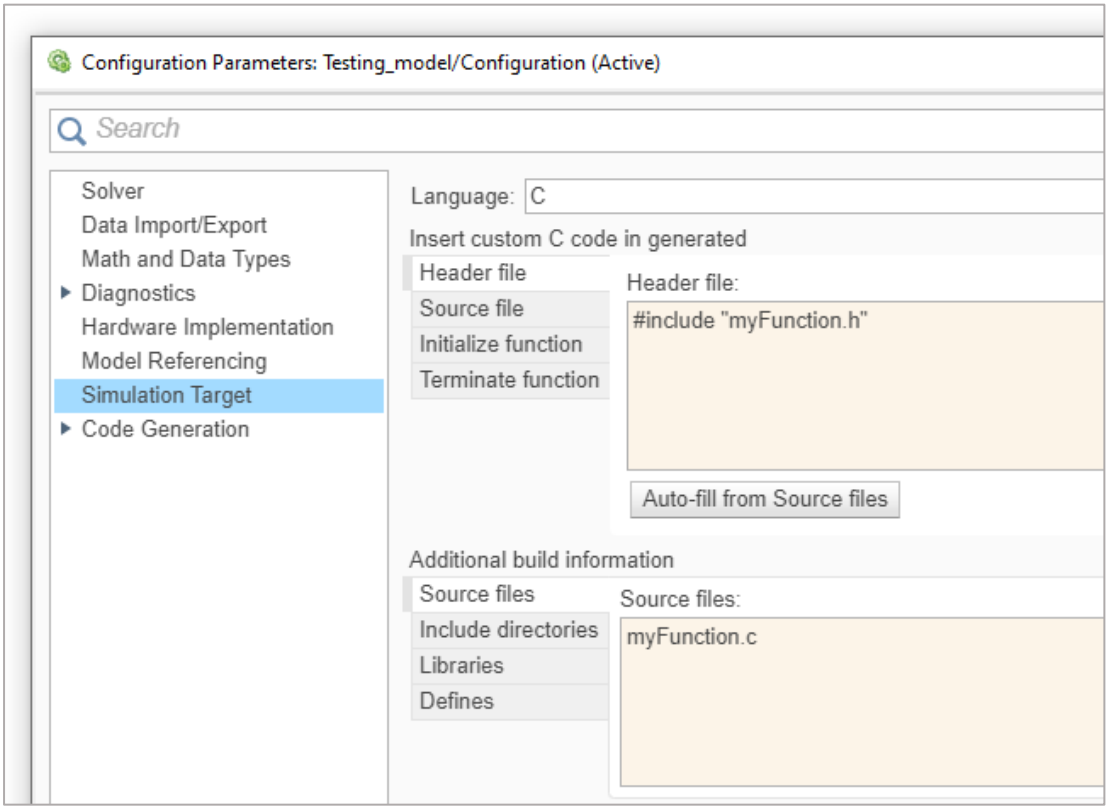
V Simulinku existuje několik možností, kterými lze použít kód napsaný v jazyce C. Nezbytným předpokladem pro všechny způsoby je MEX, celým názvem MATLAB executable. Je to rozhraní mezi C a MATLABem a umožňuje využívat v MATLABu nebo Simulinku nejen C, ale také C++ a Fortran.
Instalaci C/C++ kompilátoru je možné provést pomocí doplňku (Add-On) v záložce *Home*, sekci *Environment*, tlačítko *Add-Ons* > *Get Add-Ons*. Pak stačí vyhledat MinGW a nainstalovat. Nastavení výchozího kompilátoru, který MEX používá, je možné manuálně změnit výběrem z možností po zadání příkazu

```
mex -setup
```

Nebo lze MEX nastavit pro příslušný jazyk jedním z příkazů

```
mex -setup c
mex -setup cpp
mex -setup Fortran
```

[13] [14]




**Obrázek 2.6** Ukázka specifikace hlavičkových a zdrojových souborů

Aby bylo možné použít C skript v simulaci, nejprve je nutné specifikovat zdrojové a hlavičkové soubory. Uvnitř Simulinku v záložce *Modeling* je tlačítko *Model Settings*. Tím se otevře okno *Configuration Parameters*. Po přepnutí do záložky *Simulation Target* můžeme zadat potřebné soubory, jako na obrázku 2.6. Taktéž je zde možnost přidat knihovny. Pokud je zdrojový soubor v jazyce C++, dá se to nastavit rovněž v této záložce.

Při používání funkcí psaných v C vždy platí, že se datové typy vstupů a výstupů musejí shodovat s těmi, které jsou deklarované ve zdrojových souborech.

### 2.4.1 Coder.ceval

Jedním z nejjednodušších způsobů implementace C nebo C++ kódu je použití příkazu *coder,ceval* uvnitř bloku *MATLAB function*.

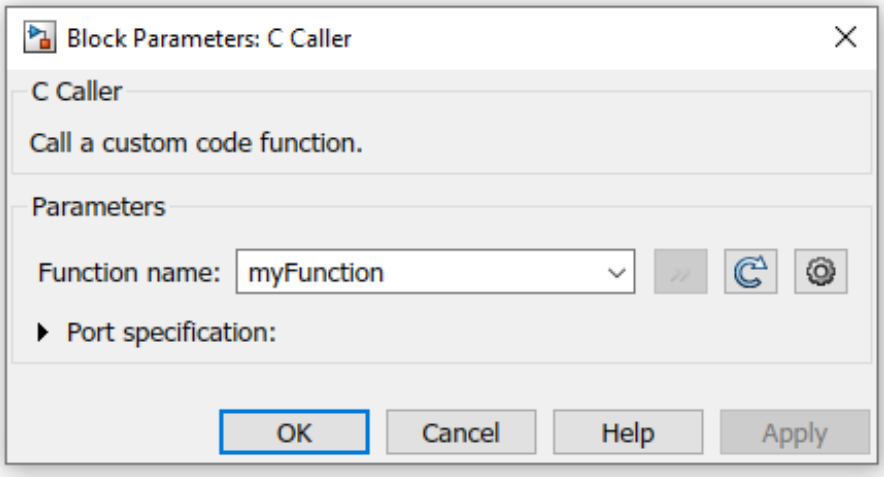
Výstupní proměnnou je nutné předem inicializovat, tedy přiřadit ji nějakou počáteční hodnotu, aby Simulink znal její velikost a datový typ.

```
function y = fcn(a,b)
y = 0.0;
y = coder.ceval('myFunction', a, b);
```

[15]

### 2.4.2 C Caller

Po rozkliknutí bločku stačí vyplnit název funkce a potvrdit. Tím se automaticky vyplní jména a datové typy vstupních i výstupních portů podle proměnných, které jsou deklarovány v definici funkce.
Jsou podporovány nejen C funkce, ale i C++. [16]




**Obrázek 2.7** Nastavení bloku C Caller

### 2.4.3 C Function

*C Function* rovněž umožňuje spustit jak C, tak i C++. Je novější než *C Caller* a poskytuje mnohem více možností. Velkou výhodou je možnost nastavení počtu, názvů, datových typů a velikosti vstupních i výstupních signálů.
Umožňuje volat několik externích C funkcí z jednoho bloku a také používat funkce ze standardní C knihovny *math.h* přímo z bloku *C Function*.

Uvnitř bloku je několik sekcí, které lze upravovat:

Output Code
Kód psaný v C, který se spustí v každém kroku simulace.

Start Code
Inicializační kód, který se spustí pouze jednou na začátku simulace.

Terminate Code
Kód, který se spustí, když simulace skončí.

Symbols
Nastavení vstupních a výstupních portů a dalších proměnných, například perzistentních, které si zachovávají hodnotu i do dalšího kroku. [17]

### 2.4.4 S-Function

Pro použití S-funkce (*system-function*), nebo také Level-1 S-funkce, stačí umístit blok a vyplnit název MEX souboru. Všechny porty se samy nastaví podle zdrojového souboru. Zdrojový soubor S-funkce musí být v cestě (*path*) MATLABu.

Nejprve je potřeba vytvořit MEX soubor. K tomu existuje více způsobů.
Z již hotového C nebo C++ skriptu lze vygenerovat soubor MEX pomocí Legacy Tool Code. Zdrojové a hlavičkové soubory musí být při generování umístěné v cestě MATLABu.

```
def = legacy_code('initialize')
def.SFunctionName = 'MEX_file_name'
def.OutputFcnSpec = 'double y1 = myFunction(double u1, double u2)'
def.HeaderFiles = {'myFunction.h'}
def.SourceFiles = {'myFunction.c'}

legacy_code('sfcn_cmex_generate',def)   % generate S-function written in C

legacy_code('compile',def)   % compile to get a MEX file
```

Další možností je využít bloček *S-Function Builder* v Simulinku, v němž je sekce pro napsání C kódu. Také lze přidávat a upravovat porty. Pak stačí vygenerovat MEX tlačítkem *Build*. [18]

### 2.4.5 Level-2 MATLAB S-Function

Další typ S-funkcí, ovšem tyto jsou psané v MATLABu a jejich zdrojový soubor je typu **.m*. Nastavení vstupních a výstupních portů probíhá pomocí kódu ve zdrojovém souboru, stejně tak i nastavení paměti (tzv. *Dwork*) a dalších parametrů bloku.

Šablona pro základní Level-2 S-funkci je dostupná z Command Window příkazem

```
edit([matlabroot,'/toolbox/simulink/blocks/msfuntmpl_basic.m'])
```

nebo existuje i pokročilejší šablona

```
edit([matlabroot,'/toolbox/simulink/blocks/msfuntmpl.m'])
```

[19]

### 2.4.6 Coder.extrinsic

Z MATLABu je možné volat funkce psané v jazyce Python. Lze to tedy provést i v Simulinku, například přes blok *MATLAB Function*. Problém je, že Simulink nedokáže takový model klasicky zkompilovat a vygenerovat kód. Jednoduchým řešením je použít příkaz *coder.extrinsic*, který řekne Simulinku, že z dané funkce nemá generovat kód, ale spustit ji jako klasickou funkci v MATLABu. [20]

Kód v *MATLAB Function*:

```
function y = fcn(u)
y = 0;
coder.extrinsic('py.fileName.myFunction')
y = py.fileName.myFunction(u);
```

[21]

# 3 Realizace komunikace

Pokoušel jsem se o vytvoření komunikace pomocí S-funkce, ovšem bez úspěchu. Vždy se podařilo vyřešit pouze dílčí problémy.

Nejprve byly potíže s datovými typy *PVOID* a *BOOL* ze standardní C knihovny *windows.h*, neboť hlavičkové soubory knihovny MPSSE nenačetly *windows.h*. Řešením bylo připsat příkaz

```
#include <windows.h>
```

na začátek hlavičkového souboru. Dalším problémem bylo dvojité definování typu *bool*. Jednou z knihovny *stdbool.h*, kterou používá Simulink, a podruhé jej definuje hlavička knihovny MPSSE jako

```
typedef unsigned char   bool;
```

Po přejmenování *bool* ve zdrojových souborech knihovny MPSSE a jejím novém vygenerováním se odstranil i tento problém. Zatím poslední chybová hláška

```
undefined reference to '__imp___acrt_iob_func'
```

se nepodařila odstranit. Zkoušel jsem načíst knihovnu ve Visual Studio Code. To se obešlo bez problémů. Z toho lze usoudit, že chyba je na straně Simulinku.

Z porovnání rychlosti (viz dále) se ukázalo, že původní metoda komunikace s využitím listeneru je dobrou volbou, proto jsem na ni navázal a udělal několik úprav (viz kapitola 3.2).

## 3.1 Statistické testování komunikace

Abych mohl vybrat optimální způsob spuštění kódu ke komunikaci a real-time synchronizace (jinak synchronizace s reálným časem), vytvořil jsem množství různých simulací, ze kterých jsem ukládal naměřená data. K měření jsem použil funkci *tic-toc*.

### 3.1.1 Způsoby implementace kódu

Celková rychlost komunikace je dána nejen rychlostí komunikačního kanálu, ale také rychlostí simulace. Proto je dobré zvolit co nejrychlejší způsob volání knihovny komunikačních funkcí.

Pro porovnání rychlosti jsem vytvořil jednoduchou funkci, a také několik modelů, z nichž každý tuto funkci spustí jinou metodou.

Porovnával jsem tyto metody

- coder.ceval – C skript volaný funkcí *coder.ceval* z bloku *MATLAB Function*
- C Caller – blok *C Caller*
- C Function – blok *C Function*
- S-Function – blok *S-Function*
- L-2 S-Function – blok *Level-2 MATLAB S-Function*
- MATLAB Function – funkce napsaná přímo v bloku *MATLAB Function*
- externí funkce – volání vlastní funkce v externím skriptu z *MATLAB Function*
- listener – callback listeneru přidaném k bločku v Simulinku
- Python – Python skript volaný pomocí *coder.extrinsic* z *MATLAB Function*

Protože Simulink musí nejprve načíst model, první simulace vždy trvala o poznání delší dobu. U každého způsobu byla tato doba různá. Abych eliminoval tuto chybu měření, spustil jsem jednou každý model, aby Simulink načetl všechny modely, a až poté provedl měření.

Testy jsem provedl se dvěma různými skripty a s různým nastavením modelů.
Test 1: složitější skript, který trvá delší dobu a je spuštěn méněkrát. Test byl zaměřen na menší množství časově náročnějších úkolů. Celkový počet kroků simulace = 10 000.
Test 2: jednodušší skript s velkým počtem opakování. Test byl zaměřen na velké množství spuštění. Celkový počet kroků simulace = 1 000 000.

Výsledky jsou v tabulce 3.1. Jsou to délky celé simulace, resp. průměrné hodnoty z 10 měření.

| metoda | test 1 [s] | test 2 [s] |
|---|---|---|
| coder.ceval | 1,416 | 2,710 |
| C Caller | 2,154 | 3,718 |
| C Function | 2,146 | 3,703 |
| S-Function | 1,382 | 2,667 |
| L-2 S-Function | 1,316 | 62,807 |
| MATLAB Function | 0,345 | 2,512 |
| externí funkce | 0,356 | 2,529 |
| listener | 0,368 | 2,515 |
| Python | 20,020 | 19,857 |

**Tabulka 3.1** Rychlost simulace při použití různých metod spouštění testovacího skriptu

Z naměřených dat vychází jako nejlepší varianta MATLAB skript, a to ať už jako samostatný skript, blok *MATLAB Function*, nebo callback listeneru. Dobré výsledky má při vysoké frekvenci spouštění také S-funkce (Level-1).

### 3.1.2 Real-Time

Real-time neboli synchronizace s reálným časem znamená zpomalení simulace na rychlost klasických hodinek. To umožňuje uživateli kupříkladu měnit parametry modelu v průběhu simulace.
Synchronizace v operačním systému je problematickou záležitostí, jedná se o tzv. *soft real-time*, takže záruka skutečného času je pouze přibližná a nikdy se nepodaří dokonale. Existují ovšem lepší a horší metody, každá má nějaké výhody a nevýhody. Zde je přehled těch, které jsem vyzkoušel.

**Simulink Desktop Real-Time**
Jednoduchý způsob real-time synchronizace je umístěním bločku *Real-Time Synchronization* z toolboxu *Simulink Desktop Real-Time* kamkoliv uvnitř modelu. Umožňuje real-time synchronizaci pro nastavitelný krok.
Kromě samotného toolboxu je zapotřebí ještě nainstalovat *Real-Time kernel*. To poskytuje rozhraní s operačním systémem a dává Simulinku nejvyšší prioritu pro využívání procesoru. K instalaci *Real-Time kernel* slouží příkaz

```
sldrtkernel -install
```

[22]

**Simulation Pacing**
Rovněž jednoduše nastavitelný způsob. Jde o funkci zabudovanou v Simulinku, která umožňuje zpomalení simulace. Zapnutí této funkce a míra zpomalení lze nastavit pod tlačítkem pro zahájení simulace (*Run*). Také existuje možnost programového nastavení příkazem

```
set_param(model, 'EnablePacing', 'on')
set_param(model, 'PacingRate', value)
```

Poprvé byla tato funkce uvedena ve verzi R2018a. [23] [24]

**Period mode**
Tato funkce je převzata z původní práce od Ing. Martina Formánka. Využívá objekt měřící čas a listener. Princip fungování je prostý – porovnává délku aktuálního kroku simulace s požadovaným krokem a pokud je aktuální krok kratší, počká, než se rozdíl vyrovná.
Problémem této synchronizace je, že pokud nějaký krok zabere více času, než je nastaveno v parametrech modelu, zpoždění zůstává až do konce simulace. Běžně je však toto zpoždění pouze nepatrné. [25]

**Adjusting mode**
Také tato funkce byla použita v původním projektu. Funguje obdobně jako *Period mode*, ovšem porovnává nejen délku aktuálního kroku, ale i délku celé simulace. Díky tomu je možné vyrovnávat zpoždění vzniklé náročnějšími kroky simulace, typicky při inicializaci. [25]

**Real-Time Pacer**
Jedná se o rozšíření pro Simulink. Bylo vytvořeno ve starší verzi Simulinku, proto jsem ho musel před použitím upravit.
Využívá Level-2 S-funkci (viz kapitola 2.4.4) a příkaz *pause*. Na konci kroku vypočte o kolik času je napřed oproti požadovanému času a pak simulaci na tuto dobu pozastaví.
K fungování vyžaduje zdrojový soubor pro S-funkci. [26]

**Porovnání**

Za účelem testování jsem vytvořil jednoduchý model a postupně jej spustil pro každý způsob synchronizace. Délka kroku byla vždy nastavena na 1 ms. V každém kroku jsem zapsal čas a vytvořil tak vektor časových hodnot. Výsledky jsem vykreslil do grafů (obrázek 3.1).
Bezprostředně po inicializaci jsou kroky řádově delší, proto jsem vykreslil grafy a vypočetl hodnoty pro ustálený stav. Tím získají výsledky větší vypovídající hodnotu pro běžné použití při simulaci.

Ideální synchronizace by v grafu vypadala jako vodorovná přímka v čase 1 ms. To by znamenalo, že každý krok zabere vždy přesně zvolený čas 1 ms, nikdy méně ani více.

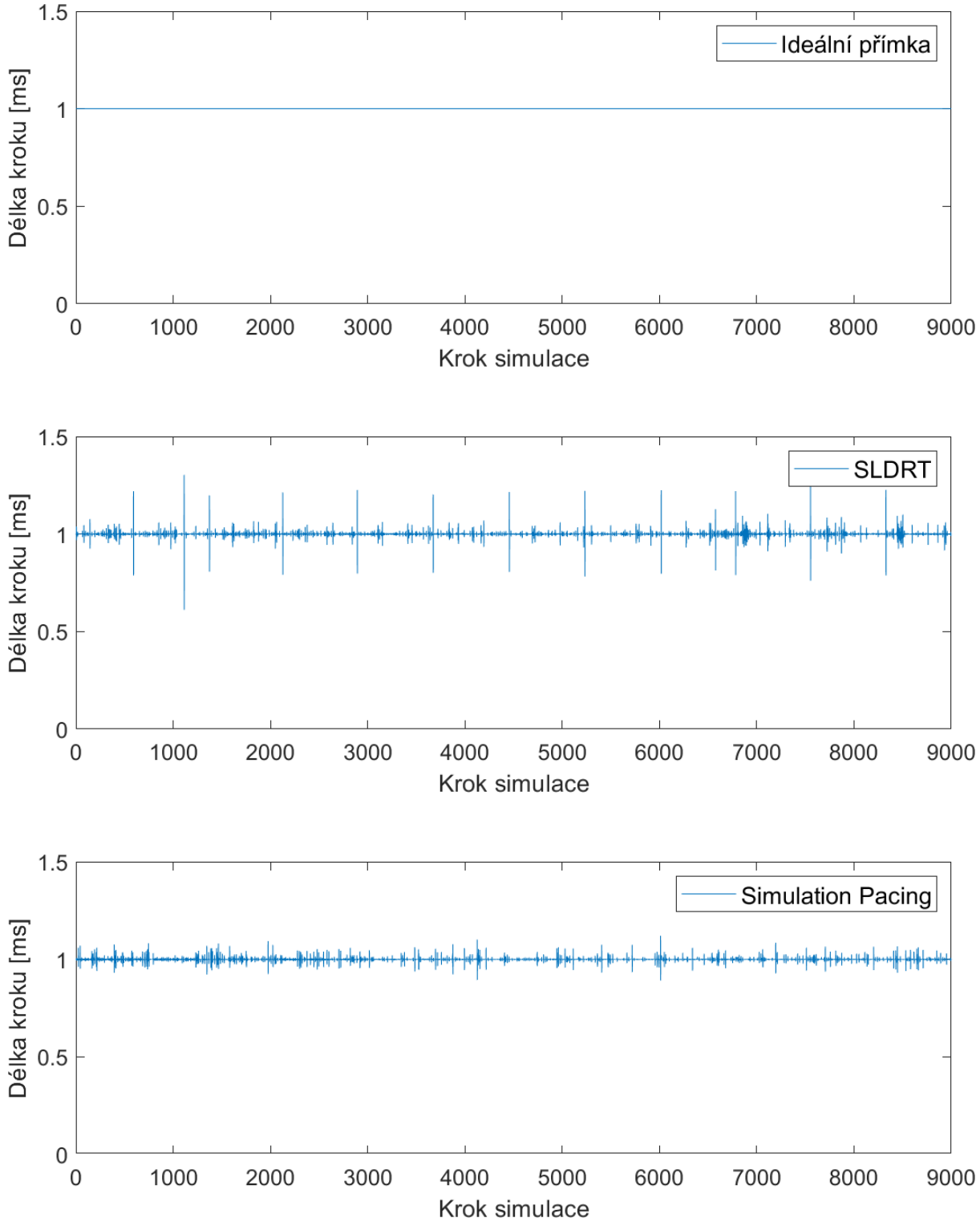
Nejlepší hodnota (tj. nejbližší ideální přímce) je v tabulce uvedena tučně.

*Adjusting mode* zkracuje délku některých kroků, proto jsem vypočetl i hodnoty se zanedbáním těchto dorovnávacích kroků. Tento soubor jsem označil jako *Adjusting (úprava)*.

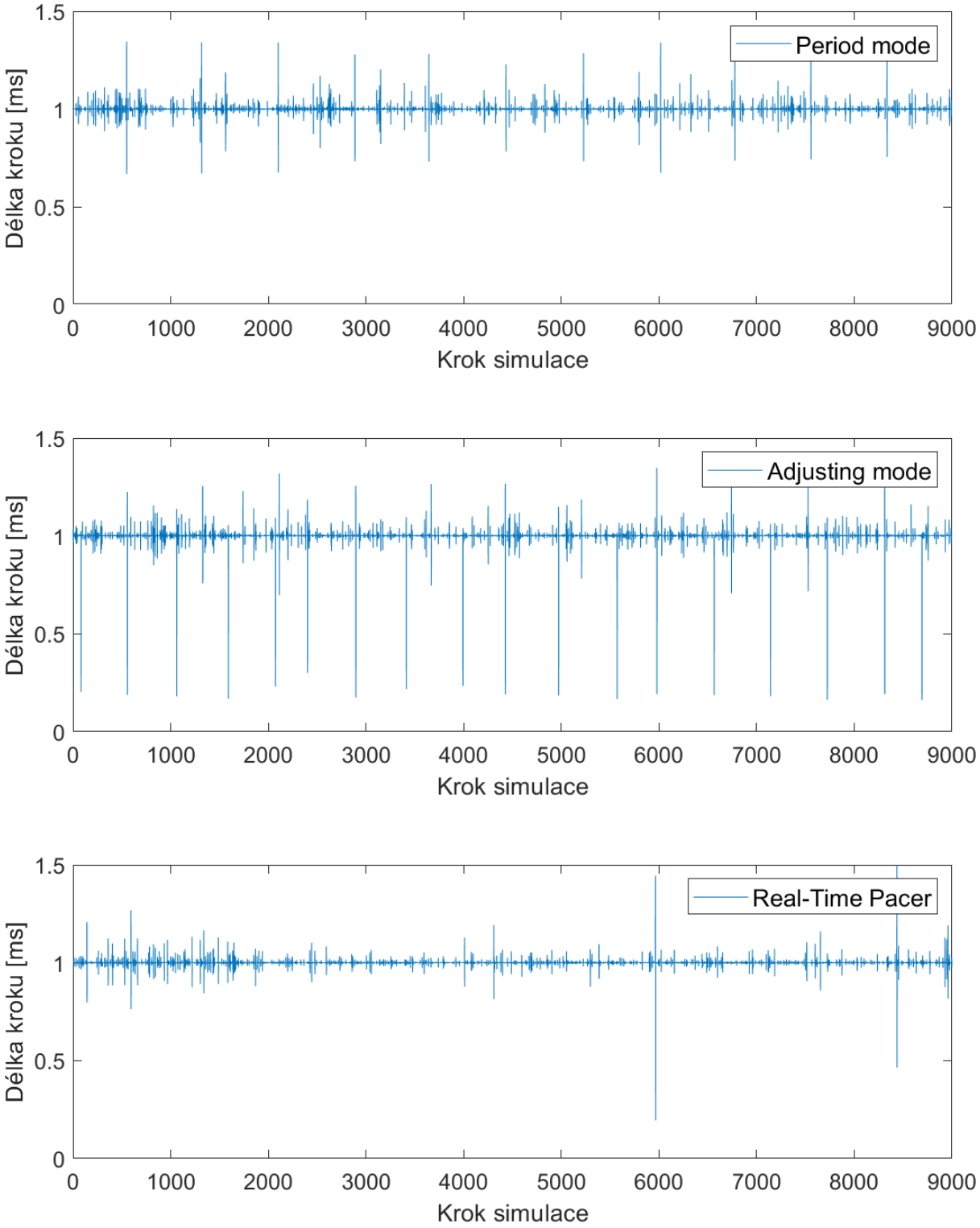
| varianta RT | Max [ms] | Min [ms] | Rozsah [ms] | Modus [ms] | Medián [ms] |
|---|---|---|---|---|---|
| Ideální přímka | 1 | 1 | 0 | 1 | 1 |
| SLDRT | 1,303 | 0,612 | 0,691 | **0,9999** | **0,9999** |
| Simulation Pacing | **1,121** | **0,891** | **0,229** | 0,999 | 0,9990 |
| Period mode | 1,345 | 0,670 | 0,676 | 1,0005 | 1,0006 |
| Adjusting mode | 1,348 | 0,162 | 1,186 | 1,0008 | 1,0011 |
| Real-Time Pacer | 1,533 | 0,194 | 1,338 | 1,0002 | **0,9999** |
| Adjusting (úprava) | 1,348 | 0,698 | 0,651 | 1,0008 | 1,0011 |

| varianta RT | Průměr [ms] | Směrodatná odchylka [ms] | Rozptyl ($\cdot 10^3$) |
|---|---|---|---|
| Ideální přímka | 1 | 0 | 0 |
| SLDRT | 0,99996 | 0,015 | 0,230 |
| Simulation Pacing | 0,99919 | **0,009** | **0,079** |
| Period mode | 1,00084 | 0,021 | 0,456 |
| Adjusting mode | 0,99997 | 0,041 | 1,671 |
| Real-Time Pacer | **1,00000** | 0,019 | 0,358 |
| Adjusting (úprava) | 1,00158 | 0,019 | 0,372 |

**Tabulka 3.2** Výsledky statistického porovnání variant real-time



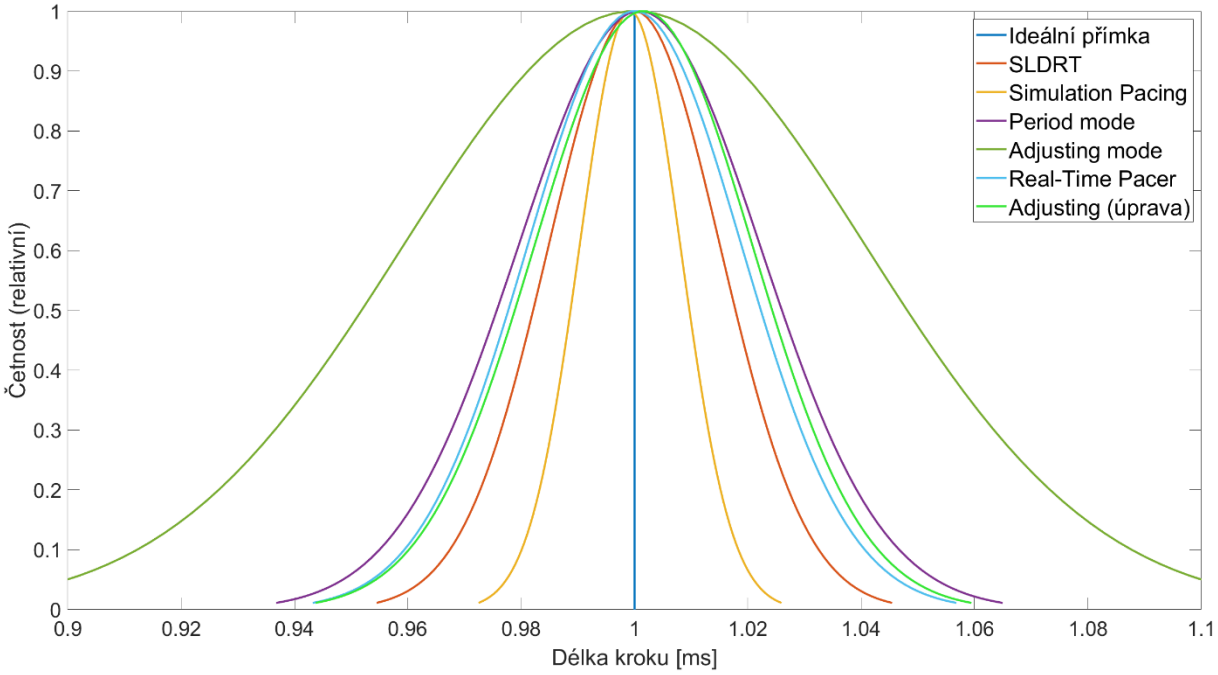

**Obrázek 3.1a)** Stabilita různých variant real-time




**Obrázek 3.1b)** Stabilita různých variant real-time

Pro vizuální srovnání všech variant jsem využil funkci *histfit*, která rozdělí osu hodnot na stejně velké intervaly a každému přiřadí četnosti podle výskytu hodnoty v datovém souboru, to jest vytvoří histogram. Následně vytvoří Gaussovu křivku, která představuje rozložení pravděpodobností výskytu jednotlivých hodnot.
Všechny křivky získané tímto způsobem jsem vykreslil do jednoho grafu (obrázek 3.2) a jejich velikosti normalizoval, aby je bylo možné vzájemně porovnat v měřítku.

Čím užší je Gaussova křivka, tím více se daná metoda blíží ideální přímce.
Z pohledu na *Adjusting mode* a upravenou variantu je patrné, jak velký efekt mají dorovnávací kroky při statistickém zpracování.




**Obrázek 3.2** Porovnání variant real-time

## Závěr

*Simulation Pacing* vykazuje velmi dobré výsledky, a navíc nevyžaduje žádné doplňky jako toolbox nebo MEX kompilátor. Proto jsem jej zvolil jako výchozí metodu real-time synchronizace.
Jako záložní variantu pro starší verze Simulinku jsem doplnil *Adjusting mode*, který také nevyžaduje žádné doplňky. Po zanedbání vlivu dorovnávacích kroků jsou výsledky *Adjusting mode* a *Period mode* velmi podobné. *Adjusting mode* dokáže navíc vyrovnávat i celkový čas simulace, na rozdíl od *Period mode*, což může být přínosem zejména u delších simulací.

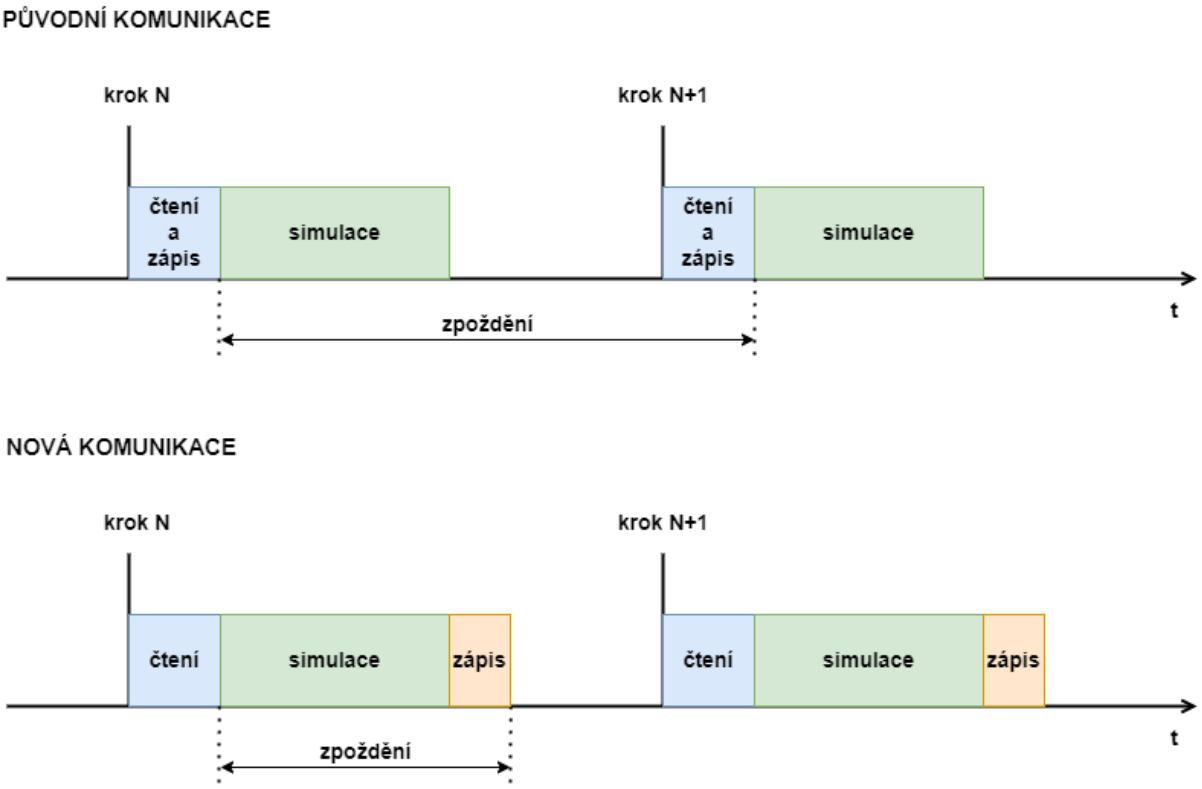
## 3.2 Komunikace s využitím objektů

Základem objektově orientovaného programování je třída, která může mít vlastnosti a metody. Objekty jsou tvořeny jako instance konkrétní třídy a mají všechny její vlastnosti i metody. V reakci na nějakou událost (event), např. změnu hodnoty jedné z jeho vlastností, objekt vysílá ostatním objektům oznámení. Listener sleduje tato oznámení a může na ně zareagovat svým callbackem, což je jeho odpověď na jednu konkrétní událost.

### Původní komunikce

Každý model musí obsahovat bloček *MSP Config*, který pomocí callbacků *InitFcn*, *StartFcn* a *StopFcn* řídí komunikaci se zařízením.
Na začátku simulace se vytvoří objekt, jehož konstruktor otevře a nastaví komunikační kanál pomocí knihovny MPSSE. Dále je vytvořen listener, který v každém kroku simulace posílá data ze Simulinku do zařízení a čte příchozí data. Příchozí zprávu pak zapíše do *Data Store* bloku, odkud jsou dostupná všem dalším bločkům knihovny MSP.

V původní komunikaci jsou data odesílána i přijímána najednou v každém kroku simulace. Pracuje se tedy s daty, která byla doručena v minulém kroku. Jinak řečeno reakce na data z mikrokontroleru bude vždy zpožděná o délku jednoho celého kroku simulace. Protože komunikace pracuje s fixní délkou kroku, je na konci každého kroku pauza na synchronizaci.




**Obrázek 3.3** Srovnání původní a nové komunikace

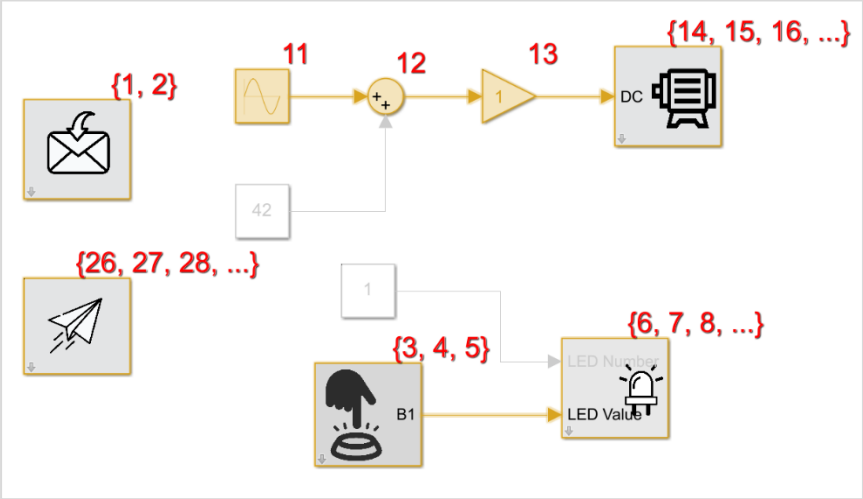
**Nová komunikace**

Pro regulaci je potřeba, aby data byla co nejaktuálnější, každé zpoždění zhoršuje kvalitu regulace. Je tudíž žádoucí, aby čtení dat probíhalo těsně před simulací a zápis těsně po simulaci, jak je možné vidět na obrázku 3.3. Tím se docílí, že nezávisle na délce kroku budou data odeslána do zařízení v nejkratším možném čase. Změna délky kroku pak nemá vliv na rychlost akčního zásahu.

Komunikace v nové knihovně je proto rozdělena mezi blok *MSP Read*, který přečte data ze zařízení, a blok *MSP Write*, který pošle data do zařízení.

Pro zajištění správnosti pořadí spuštění všech bloků je využita *priorita*, která je na začátku simulace přidělena inicializační funkcí všem blokům tak, že *MSP Read* je v pořadí jako úplně první a *MSP Write* jako úplně poslední, jak je možné vidět na obrázku 3.4.
Jedinou výjimkou jsou konstanty, které tvoří samostatnou skupinu bloků a spouští se až po ostatních blocích. To ovšem nevadí, neboť jejich hodnota zůstává v každém kroku stejná.

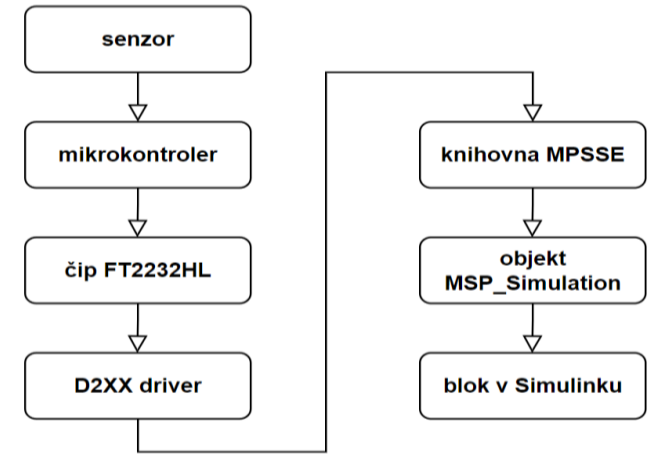
Pořadí spouštění bloků je možné zobrazit v záložce *Debug*, sekci *Diagnostics*, tlačítko *Information Overlays* > *Blocks* > *Execution Order*.




**Obrázek 3.4** Ukázka pořadí spouštění bloků

Při inicializaci modelu je vytvořen objekt *MSP_Simulation*, ten otevře komunikační kanál a nastaví parametry modelu vč. pořadí bloků, a poté jsou vytvořeny listenery. Objekt *MSP_Simulation* je automaticky vytvořen ve Workspace, a díky tomu k němu mohou snadno přistupovat všechny bloky v Simulinku.

Všechny vrstvy komunikace směrem z mikrokontroleru do Simulinku jsou znázorněny ve schématu na obrázku 3.5.




**Obrázek 3.5** Schéma vrstev komunikace

## 3.3 Měření rychlosti komunikace

Rychlost komunikace je objem dat přenesený za jednotku času, k výpočtu je proto potřeba znát objem dat poslaných v jednom kroku simulace a délku kroku.

Celkový objem dat odeslaných v jednom kroku se skládá ze zprávy odeslané před simulací (44 bytů) a zprávy odeslané po simulaci (10 bytů), dohromady 54 bytů. Po vynásobení osmi dostáváme, že je v každém kroku odesláno 432 bitů.

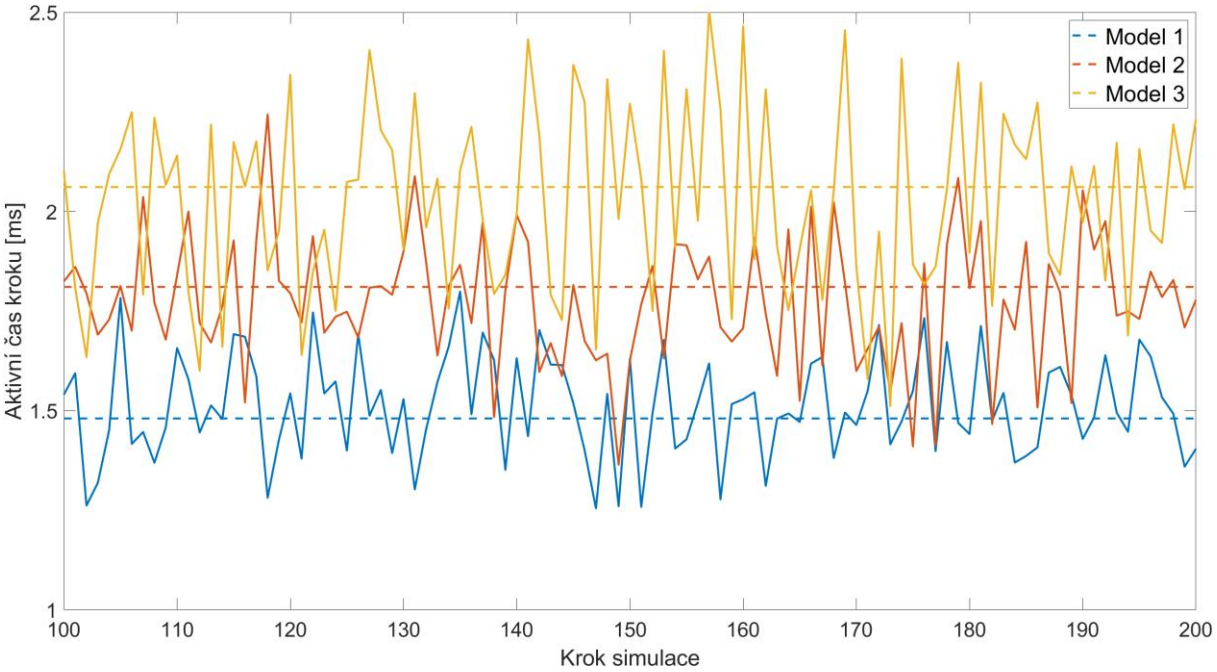
Při měření délky kroku nás zajímá pouze aktivní čas, tedy bez real-time synchronizace. V každém kroku jsem zapsal čas na začátku před posláním prvního bytu a na konci po skončení simulace a odeslání posledního bytu. Odečtením těchto hodnot jsem získal aktivní čas daného kroku.
Tímto způsobem jsem získal vektor hodnot a z něj následně vypočetl průměrnou hodnotu, kterou jsem zanesl do tabulky 3.3.

Rychlost komunikace jsem dále získal vydělením objemu dat průměrným časem.
Měření jsem provedl pro tři různě složité modely.

| | Model 1 | Model 2 | Model 3 |
|---|---|---|---|
| **Počet bloků v modelu** | 12 | 88 | 346 |
| **Průměrná délka kroku** | 1,481 ms | 1,811 ms | 2,061 ms |
| **Rychlost komunikace** | 291,7 kbps | 238,5 kbps | 209,6 kbps |

**Tabulka 3.3** Rychlost komunikace pro různé modely

Rychlost komunikace se liší v závislosti na složitosti modelu. Pro testované modely se pohybovala mezi 200-300 kbps.
Jedná se o průměrné hodnoty, rychlost v průběhu simulace totiž kolísá (viz obrázek 3.6).




**Obrázek 3.6** Délky kroků a průměrné hodnoty

## 3.4 Knihovna do Simulinku

### Vytváření knihovny

K vytvoření vlastní knihovny stačí otevřít Simulink a na startovní stránce zvolit *Blank Library*. Knihovna je soubor typu **.slx*, ve kterém lze vytvářet a upravovat bloky stejně jako v klasickém modelu Simulinku. Pro přehlednost je možné bloky rozdělit do subsystémů, čímž se v prohlížeči vytvoří hierarchie. [27]

Aby se knihovna zobrazovala v prohlížeči Simulinku, je potřeba nastavit její viditelnost příkazem

```
set_param(gcs,'EnableLBRepository','on');
```

a také vytvořit funkci *slblocks.m* ve stejné složce, jako je knihovna uložena

```
function blkStruct = slblocks
% This function specifies that the library 'mylib'
% should appear in the Library Browser with the
% name 'My Library'

    Browser.Library = 'mylib';

    Browser.Name = 'My Library';

    blkStruct.Browser = Browser;
```

[28]

Blokům lze vytvořit masku, které je možné přidat parametry nastavitelné po rozkliknutí. Masce lze také nastavit ikonu. Masky mají callbacky, jež jsou volány například po změně parametrů.

### Knihovna MSP

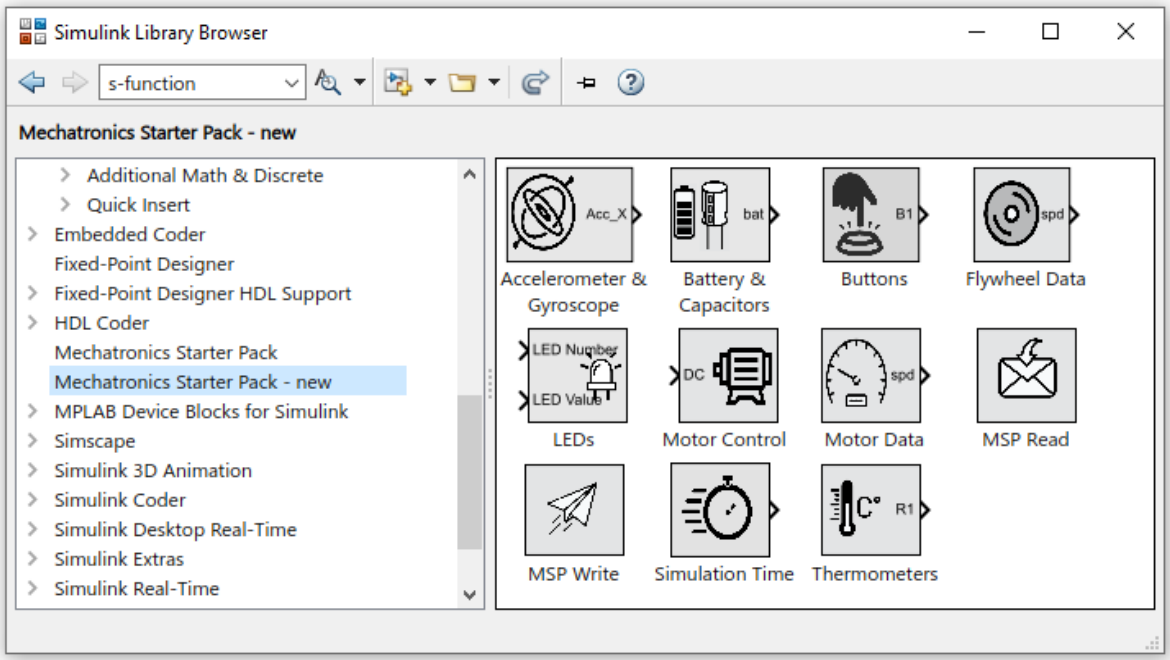
Nová knihovna je vytvořena podle vzoru té původní, kterou vytvořil Ing. Martin Formánek, neboť jeho knihovna je dobře vymyšlena a zpracována. Také jsem použil některé funkce napsané pro původní knihovnu.

Nová knihovna byla vytvořena ve verzi MATLABu R2021b v operačním systému Windows 10. Pro používání MSP knihovny je nejprve nutné nainstalovat D2XX driver od FTDI, a také *MSP toolbox*, který je v elektronických přílohách této práce.

Každý model musí obsahovat bločky *MSP Read* a *MSP Write*. Nic dalšího není potřeba nastavovat, vše je ošetřeno uvnitř bločku *MSP Read*.
Nejkratší doporučená délka kroku je 2 ms. Pro kratší krok bude komunikace fungovat, ale není zaručena real-time synchronizace. Automatická délka kroku je nastavena na 10 ms.

Data jsou z mikrokontroleru posílána jako řetězec *uint8* bytů. Některé hodnoty jsou rozloženy do více bytů a je potřeba je nejprve spojit zpět do jednoho čísla. K tomu slouží funkce *typecast*. Při jejím použití z *MATLAB Function* je ovšem rozměr výstupního signálu menší než velikost vstupu a je třeba nastavit velikost výstupu uvnitř bloku *MATLAB Function* přes tlačítko *Ports & Data Manager*.

Předávání dat z mikrokontroleru mezi jednotlivými bloky knihovny je vyřešené pomocí *Data Store* a výběru konkrétních elementů.




**Obrázek 3.7** Vytvořená knihovna

**Popis bločků knihovny**

MSP Read
Jeden blok *MSP Read* musí být obsažen v každém modelu.
Inicializační funkce vytvoří objekt, čímž zahájí komunikaci a zkontroluje parametry modelu. V každém kroku se pak stará o čtení dat z MSP zařízení a zápisu do *Data Store*. Tato data jsou pak k dispozici všem blokům knihovny.

MSP Write
Jeden blok *MSP Write* musí být obsažen v každém modelu.
Uvnitř bloku je funkce, která poskládá informace o zapnutí LED do jednoho bytu. Dále také funkce pro odeslání dat do MSP zařízení. Ve starších verzích Simulinku se rovněž stará o real-time synchronizaci.

LEDs
První port je výběr čísla LED. Vstupem může být jedna hodnota nebo vektor hodnot. Dostupná čísla LED jsou 1-6.
Druhým portem je LED status, který může být 0 nebo 1, podle toho, jestli se má zvolená LED zhasnout nebo rozsvítit.
Oba vstupy je možné po rozkliknutí bloku přepnout zdroj signálu na *Internal* a nastavit jejich hodnoty vnitřně z nabídky parametrů v masce.

Motor Control
Vstup *DC* (Duty Cycle) je střída signálu v rozmezí od -100 do +100. Hodnota je v procentech maximálního rozsahu a znaménko značí směr.
Port *Frq* je nastavení spínací frekvence H-můstku. Minimální povolená hodnota je 200 Hz, maximální 20 kHz.
Po přepnutí na *Internal* je také možné nastavit obě hodnoty uvnitř bloku.

Simulation Time
Výstup bloku je pseudoreálný čas pro aktuální krok simulace. Skutečně reálný čas by musel být měřen například krystalem na desce PLC (viz kapitola 3.1.2 Real-time).
Liší se od bloku *Clock*, který je součástí Simulinku, protože *Clock* ukazuje ideální čas, který by měl aktuálně být podle toho, ve kterém kroku simulace je. Jinak řečeno ukazuje spíše číslo kroku než skutečný čas. Naproti tomu *Simulation Time* měří čas simulace s využitím funkce *tic-toc* a naměřený čas dává na výstup bloku.

Buttons
Po zvolení čísla tlačítka 1-6 je na výstupu při stisknutém tlačítku 1, při uvolnění tlačítka 0.

Motor Data
Bloček nabízí souhrn dat týkajících se motoru. Po rozkliknutí je možné na výstup zvolit data z enkodéru motoru pro pozici [rad] nebo rychlost [rad/s], nebo proud motorem [mA].

Flywheel Data
Data z magnetického enkodéru pro setrvačník. Opět je na výběr pozice [rad] a rychlost [rad/s].

Thermometers
Nabízí výběr teploty dvou rezistorů a okolní teploty ve °C.

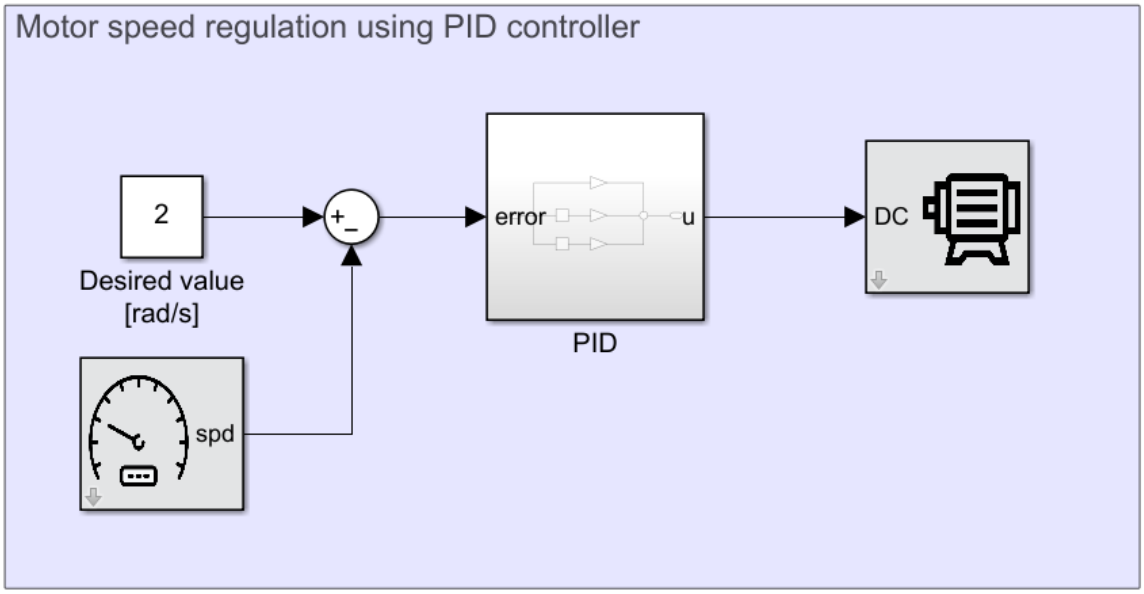
Battery & Capacitors
Napětí na jednom ze dvou kondenzétorů, nebo napětí baterie [mV].

Accelerometer & Gyroscope
Bloček nabízí možnost volby senzoru a také osy X, Y, nebo Z.

# 4 Demonstrační úlohy

### Úloha 1 – regulace rychlosti motoru

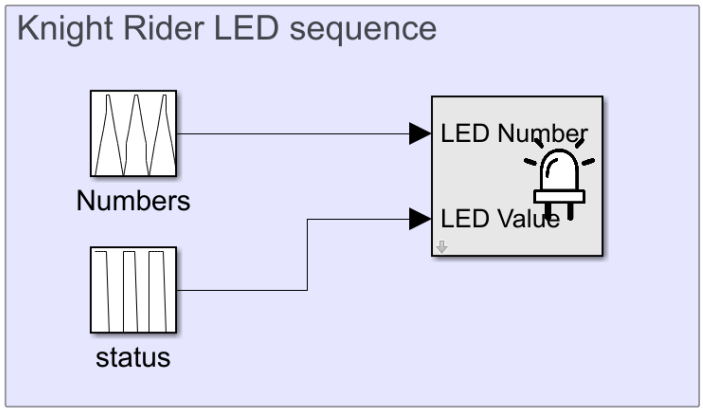
PID regulátor je sestaven jako součet proporcionální, integrační a derivační složky. Uvnitř bloku lze jednoduše měnit hodnoty a sledovat, jak se chová motor.
Na vstupu PID regulátoru je regulační odchylka a na výstupu akční zásah v podobě střídy motoru.




**Obrázek 4.1** Ukázka regulace rychlosti motoru

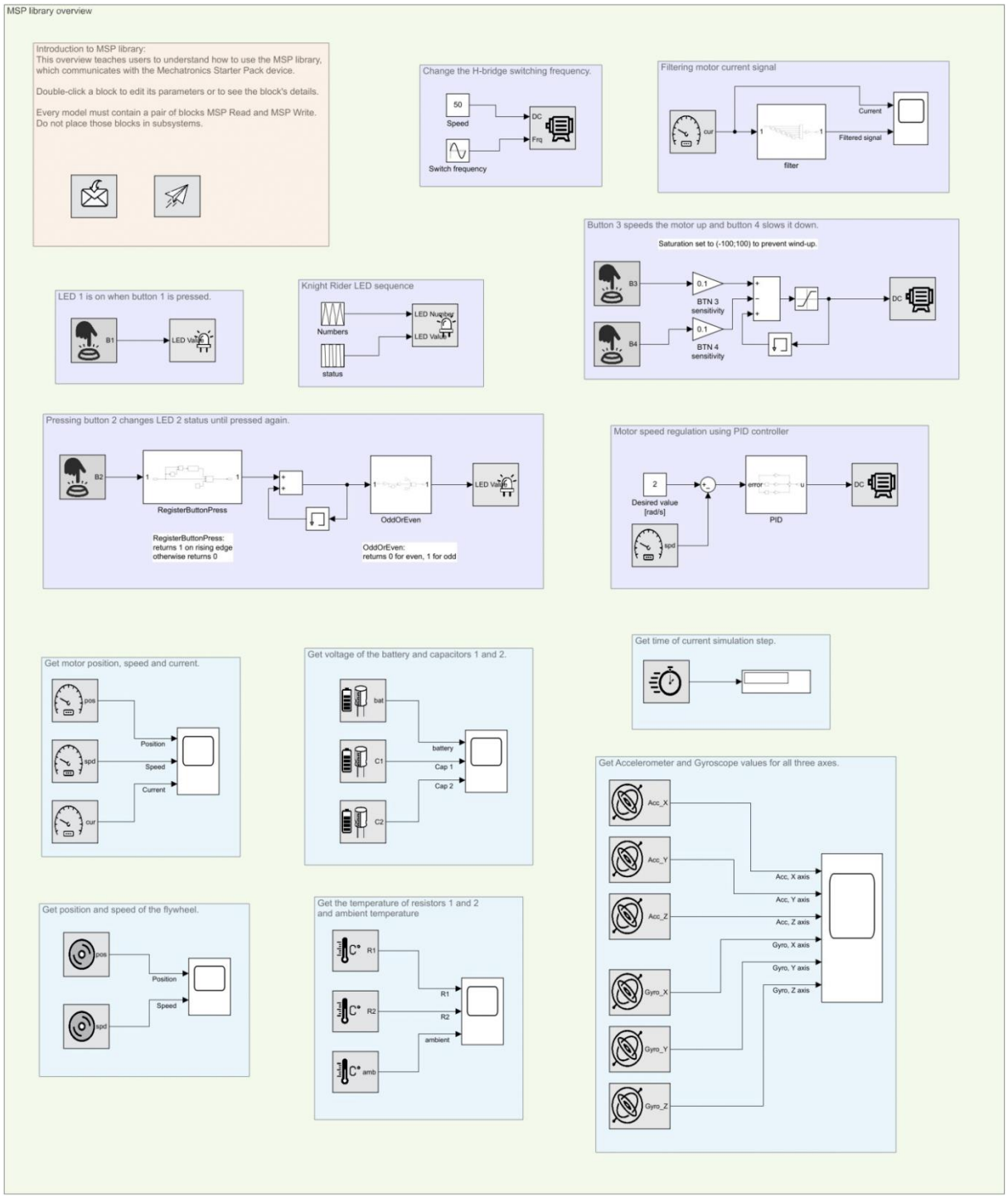
### Úloha 2 – ovládání LED

LEDky se postupně rozsvěcejí a zhasínají v opačném pořadí, čímž vytvoří efekt Knight Rider. Úloha vysvětluje použití bloku *LEDs*, zejména typ signálu na vstupu jednotlivých portů.




**Obrázek 4.2** Ukázka ovládání bloku *LEDs*

Přehled celého souboru vytvořených úloh je vidět na obrázku 4.3.




**Obrázek 4.3** Vytvořený soubor ukázkových úloh

# 5 Závěr

Cílem této práce bylo nalezení optimálního způsobu komunikace s cílovým zařízením.
V řešeršní části jsem nejprve porovnal komunikační protokoly, jejich výhody a nevýhody. Nejvhodnějším z nich je pro tuto aplikaci SPI, zejména kvůli rychlosti.

Následuje kapitola o periferiích na nové desce, kde jsou uvedeny základní informace a parametry. Přepočtové vzorce použité v nové knihovně jsem umístil do příloh.
Dále jsem popsal čip FT2232H, který obsahuje 2 bloky MPSSE, díky čemuž je umožněno snadné připojení zařízení přes USB. Pro navázání spojení a přenos dat je nutné použít speciální DLL knihovnu od FTDI.
Kapitola 2.4 se věnuje různým způsobům, kterými je možné volat externí skript, jenž by využíval tuto DLL knihovnu, z prostředí Simulink. Uvádím také předpoklady nutné pro jejich použití. Konkrétně popisuji například C funkci, S-funkce (Level-1 i Level-2) nebo Python.

V praktické části jsem porovnával rychlost těchto metod při spouštění testovacího modelu. Vytvořil jsem dva modely, jeden složitější s menším počtem opakování, a druhý jednodušší, který se zaměřoval na vysokou frekvenci spouštění.
Jednou z testovaných variant byl i původní způsob komunikace. Ten se v testech ukázal jako velmi dobrý, proto jsem jej použil při tvorbě nové komunikace.
Také jsem testoval různé možnosti real-time synchronizace. Některé vyžadují instalaci toolboxu nebo MEX kompilátoru, pro jiné stačí Simulink. Jako výchozí variantu jsem vybral *Simulation Pacing*, která měla dobré výsledky a je integrovanou součástí Simulinku. Pro starší verze Simulinku, které *Simulation Pacing* nenabízejí jsem doplnil ještě záložní variantu *Adjusting mode* použitou i v původní práci. Vzájemné porovnání je dobře vidět na obrázku 3.2.

V kapitole 3.2 jsem shrnul původní a novou komunikaci a rozdíly mezi nimi. Zatímco v původní verzi probíhalo čtení i zápis najednou, v nové probíhá čtení dat těsně před simulací a zápis těsně po ní. Tím dochází ke zkrácení zpoždění reakce, což umožňuje hladší regulaci.

Rychlost komunikace jsem měřil nepřímo jako podíl objemu dat přenesených v jednom kroku (432 bitů) a průměrné aktivní délky kroku. Rychlost závisí na složitosti modelu. Maximální naměřená rychlost byla necelých 300 kbps.

Poznatky uvedené v kapitole 3.1 jsem dále zužitkoval při tvorbě nové knihovny. Ze zdrojových souborů jsem vyrobil toolbox, po jehož instalaci je knihovna plně připravena k použití.
Pro použití knihovny stačí do modelu přidat bločky *MSP Read* a *MSP Write*, o ostatní nastavení parametrů modelu se za uživatele postará inicializační funkce knihovny.

Na konec jsem přidal ještě demonstrační úlohy, které vysvětlují práci s knihovnou. Soubor úloh je součástí elektronické přílohy.

# SEZNAM POUŽITÝCH ZDROJŮ

[1] GOFTON, Peter W. *Sériová komunikace*. Praha: Grada Publishing, 1995, 240 s. ISBN 80-7169-131-3.

[2] *UART vs I2C vs SPI – Communication Protocols and Uses.* [online]. [cit. 2022-05-05]. Dostupné z: https://www.seeedstudio.com/blog/2019/09/25/uart-vs-i2c-vs-spi-communication-protocols-and-uses/

[3] *UART vs SPI vs I2C | Difference between UART, SPI and I2C.* [online]. [cit. 2022-05-05]. Dostupné z: https://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html

[4] Pololu: *25D Metal Gearmotors.* [online]. [cit. 2022-05-11]. Dostupné z: https://www.pololu.com/file/0J1829/pololu-25d-metal-gearmotors.pdf

[5] *TMCS1100 1% High-Precision, Basic Isolation Hall-Effect Current Sensor With ±600-V Working Voltage.* [online]. [cit. 2022-05-11]. Dostupné z: https://www.ti.com/lit/ds/symlink/tmcs1100.pdf?ts=1652418537693&ref_url=https%253A%252F%252Fwww.mouser.de%252F

[6] *AS5147P, 14-Bit On-Axis Magnetic Rotary Position Sensor with 12-Bit Binary Incremental Pulse Count and for 28krpm High Speed Capability.* [online]. [cit. 2022-05-11]. Dostupné z: https://ams.com/documents/20143/36005/AS5147P_DS000328_2-00.pdf/847d41be-7afa-94ad-98c2-8617a5df5b6f

[7] *Low-Power Linear Active Thermistor ICs.* [online]. [cit. 2022-05-11]. Dostupné z: https://ww1.microchip.com/downloads/en/DeviceDoc/20001942G.pdf

[8] *BMX160, Small, low power 9-axis sensor.* [online]. [cit. 2022-05-11]. Dostupné z: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmx160-ds0001.pdf

[9] *FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC Datasheet Version 2.6.* [online]. [cit. 2022-05-07]. Dostupné z: https://ftdichip.com/wp-content/uploads/2020/07/DS_FT2232H.pdf

[10] *FTDI MPSSE Basics Version 1.1.* [online]. [cit. 2022-05-07]. Dostupné z: https://www.ftdichip.com/Documents/AppNotes/AN_135_MPSSE_Basics.pdf

[11] *User Guide For libMPSSE - SPI Version 1.1.* [online]. [cit. 2022-05-07]. Dostupné z: https://ftdichip.com/wp-content/uploads/2020/08/AN_178_User-Guide-for-LibMPSSE-SPI.pdf

[12] *Using MATLAB with Other Programming Languages.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/products/matlab/matlab-and-other-programming-languages.html

[13] *MinGW-w64 Compiler.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/matlab/matlab_external/install-mingw-support-package.html

[14] *Change Default Compiler.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/matlab/matlab_external/changing-default-compiler.html

[15] *Coder.ceval: Call external C/C++ function.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/simulink/slref/coder.ceval.html

[16] *C Caller: Integrate C code in Simulink.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/simulink/slref/ccaller.html

[17] *C Function: Integrate and call external C/C++ code from a Simulink model.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/simulink/slref/cfunction.html

[18] *Integrate C Functions Using Legacy Code Tool.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/simulink/sfg/integrating-existing-c-functions-into-simulink-models-with-the-legacy-code-tool.html

[19] *Write Level-2 MATLAB S-Functions.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html

[20] *Coder.extrinsic: Declare a function as extrinsic and execute it in MATLAB.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/simulink/slref/coder.extrinsic.html

[21] *How to Call Python from MATLAB | MATLAB and Python Together, Part 1*. YouTube [online]. 6. 5. 2021 [cit. 2022-05-07]. Dostupné z: https://youtu.be/TQOEcUdw9Wk

[22] *Install Real-Time Kernel.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/sldrt/ug/real-time-windows-target-kernel.html

[23] *Simulation Pacing.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/simulink/ug/simulation-pacing.html

[24] *Simulink Release Notes.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/simulink/release-notes.html?rntext=&startrelease=R2018a&endrelease=R2018a&groupby=release&sortby=descending&searchHighlight=

[25] FORMÁNEK, Martin. *Educating model for mechatronics: model development and fast USB communication.* Brno, 2020. [cit. 2022-05-07]. Master's thesis. Brno University of Technology, Faculty of Mechanical Engineering. Supervisor Ing. Martin Appel, 80 pages.

[26] Gautam Vallabha (2022). Real-Time Pacer for Simulink (https://www.mathworks.com/matlabcentral/fileexchange/29107-real-time-pacer-for-simulink), MATLAB Central File Exchange. Retrieved May 7, 2022.

[27] *Create Custom Library.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/simulink/ug/creating-block-libraries.html

[28] *Add Libraries to the Library Browser.* [online]. MathWorks [cit. 2022-05-07]. Dostupné z: https://www.mathworks.com/help/simulink/ug/adding-libraries-to-the-library-browser.html

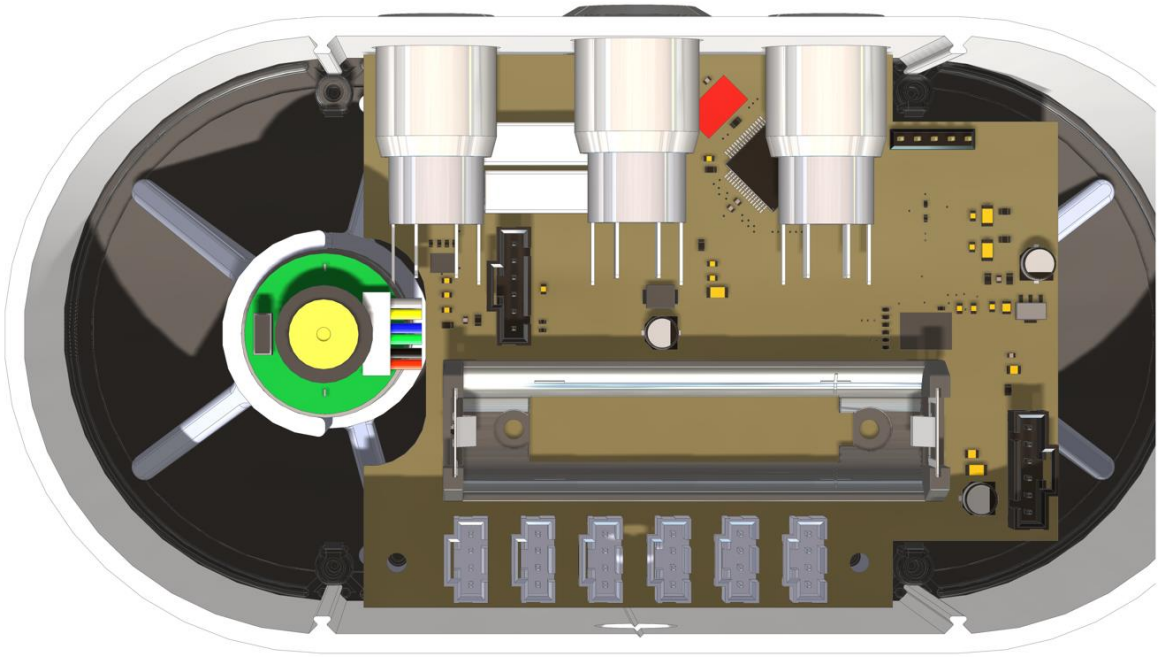
# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| **A/D** | Analog-to-Digital |
| **ACK** | Acknowledge |
| **ADC** | Analog-to-Digital Converter (analogově digitální převodník) |
| **CPR** | Counts Per Revolution |
| **CS** | Chip Select |
| **D/A** | Digital-to-Analog |
| **DC** | Direct Current (stejnosměrný proud) |
| **DC** | Duty Cycle (střída signálu) |
| **DLL** | Dynamic-Link Library |
| **FTDI** | Future Technology Devices International |
| **$I^2C$** | Inter-Integrated Circuit |
| **IMU** | Inertial Measurement Unit (inerciální měřicí jednotka) |
| **JTAG** | Joint Test Action Group |
| **LED** | Light-Emitting Diode |
| **LSB** | Least Significant Byte (nejméně významný byte) |
| **MEX** | MATLAB Executable |
| **MISO** | Master In, Slave Out |
| **MOSI** | Master Out, Slave In |
| **MPSSE** | Multi-Protocol Synchronous Serial Engine |
| **MSB** | Most Significant Byte (nejvýznamější byte) |
| **MSP** | Mechatronics Starter Pack |
| **PLC** | Programmable Logic Controller (programovatelný logický automat) |
| **RT** | Real-Time |
| **SCK** | Serial Clock |
| **SDA** | Serial Data (sériová data) |
| **SLDRT** | Simulink Desktop Real-Time |
| **SPI** | Serial Peripheral Interface |
| **SS** | Slave Select |
| **UART** | Universal Asynchronous Receiver-Transmitter |
| **USB** | Universal Serial Bus |

# PŘÍLOHY

## A. VIZUÁLNÍ NÁHLED NOVÉHO MSP ZAŘÍZENÍ

**Obrázek A.1** Pohled zepředu



**Obrázek A.2** Pohled zespodu na řídicí desku

**Obrázek A.3** Pohled zezadu

## B. NOVÁ ZPRÁVA PRO MSP

Zpráva má dvě části – první je odeslána na začátku simulace a druhá na konci.
Vždy jedno zařízení odesílá data a druhé kontrolní zprávu.

**Byty společné pro všechny zprávy**
CDM = 114 pro čtení a CMD = 119 pro zápis (command, řídící příkaz)
CRC – Cyclic Redundancy Check (kontrola správnosti zprávy)
CR = 13 (ASCII Carriage Return)

Pokud je číslo rozdělené do více bytů, nejvýznamnější byte (MSB) je v tabulce označený číslem 1 (např. *mSPD_1*).

### Čtení dat z MSP

**READ: MSP-Simulink**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| CMD | BTNs | mENC_1 | mENC_2 | mENC_3 | mENC_4 | mCUR_1 | mCUR_2 | mSPD_1 | mSPD_2 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| fENC_1 | fENC_2 | fENC_3 | fENC_4 | fSPD_1 | fSPD_2 | R1T_1 | R1T_2 | R2T_1 | R2T_2 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| AMBT_1 | AMBT_2 | BAT_1 | BAT_2 | C1_1 | C1_2 | C2_1 | C2_2 | ACCX_1 | ACCX_2 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| ACCY_1 | ACCY_2 | ACCZ_1 | ACCZ_2 | GYRX_1 | GYRX_2 | GYRY_1 | GYRY_2 | GYRZ_1 | GYRZ_2 |
| 41 | 42 | 43 | 44 | | | | | | |
| ERROR | CRC_1 | CRC_2 | CR | | | | | | |

**Obrázek B.1** Zpráva z MSP do Simulinku při čtení před simulací

**BTNs**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| rezerva | rezerva | BTN6 | BTN5 | BTN4 | BTN3 | BTN2 | BTN1 |

**Obrázek B.2** Detail bytu tlačítek (byte číslo 2)

**READ: Simulink-MSP**

| 1 | 2 | 3 | 4 | 5...44 |
|---|---|---|---|---|
| CMD | CRC_1 | CRC_2 | CR | 0 |

**Obrázek B.3** Zpráva ze Simulinku do MSP při čtení před simulací

**Zápis dat do MSP**

**WRITE: MSP-Simulink**

| 1 | 2...9 | 10 |
|---|---|---|
| CMD | 0 | CR |

**Obrázek B.4** Zpráva z MSP do Simulinku při zápisu po simulaci

**WRITE: Simulink-MSP**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| CMD | LED | mDC_1 | mDC_2 | mFRQ_1 | mFRQ_2 | LOGIC | CRC_1 | CRC_2 | CR |

**Obrázek B.5** Zpráva ze Simulinku do MSP při zápisu po simulaci

**LED**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| rezerva | rezerva | LED6 | LED5 | LED4 | LED3 | LED2 | LED1 |

**Obrázek B.6** Detail bytu LED (byte číslo 2)

**LOGIC**

| | |
|---|---|
| 7 | rezerva |
| 6 | rezerva |
| 5 | RST_FLYWHEEL_ENC |
| 4 | RST_MOT_ENC |
| 3 | MOT_RESISTOR_SWITCH |
| 2 | CAP_CHRG |
| 1 | CAP_DISCHRG |
| 0 | PWR_EN |

**Obrázek B.7** Detail řídícího bytu (byte číslo 7)

Zkratky ve schématech zpráv:

BTNs – hodnoty tlačítek
mENC – data enkodéru motoru
mCUR – proud motorem
mSPD – rychlost motoru
fENC – enkodér setrvačníku
fSPD – rychlost setrvačníku
R1T, R2T – teplota rezistoru 1, resp. 2
AMBT – okolní teplota
BAT – napětí baterie
C1, C2 – napětí na kondenzátoru 1, resp. 2
ACC – hodnota akcelerometru v příslušné ose
GYR – hodnota gyroskopu v příslušné ose
ERROR – error nebo status

LED – hodnoty LED
mDC – požadovaná rychlost motoru
mFRQ – frekvence spínání H-můstku
LOGIC – řídící byte

## C. PŘEPOČTOVÉ VZORCE PRO DATA Z PERIFERIÍ

Přepočtové vzorce vycházejí nejen z vlastností samotných senzorů, ale také z obvodového zapojení na DPS, a jsou výsledkem práce týmu MechLabu. Vzorce samotných periferií a některé konstanty lze najít v datasheetech, které jsou v seznamu literatury.
Proměnné ve vzorcích mají jiné značení než v datasheetech.

Všechny vzorce jsou tvaru

$$\boldsymbol{y} = f(\boldsymbol{u}) \tag{1}$$

kde
$\boldsymbol{u}$ je hodnota obdržená ze zařízení (datový typ uint8)
$\boldsymbol{y}$ je hodnota na výstupu bloku knihovny (datový typ double)

### DC motor s enkodérem

Rovnice pro pozici a rychlost motoru

$$\boldsymbol{y} = \frac{\boldsymbol{u} \cdot 2\pi}{CPR \cdot K} \tag{2}$$

Hodnoty:
$CPR = 48$ $(rozlišení\ enkodéru\ motoru, "Counts\ Per\ Revolution")$
$K = 20{,}4$ $(převodový\ poměr\ motoru)$

Údaje v datasheetu:
Typ: 4883 (produktové číslo společnosti Pololu Robotics and Electronics)
$CPR$ – strana 1
$K$ – strana 3

**Čidlo proudu**

Rovnice pro proud motorem

$$\boldsymbol{y} = \frac{\frac{\boldsymbol{u}}{R_{ADC}} \cdot V_{REF} + G_{CUR} \cdot V_{CUR0}}{G_{CUR} + 1} - V_{CUR0} \over \alpha_{CUR} \tag{3}$$

Hodnoty:
$V_{REF} = 3000\ mV \quad (referenční\ napětí)$
$R_{ADC} = 4096 \quad (rozlišení\ ADC)$
$G_{CUR} = 1{,}2 \quad (zesílení\ operačního\ zesilovače)$
$V_{CUR0} = 1500\ mV \quad (offset)$
$\alpha_{CUR} = 400\ mV/°C \quad (teplotní\ koeficient)$

Údaje v datasheetu:
Typ: TMCS1100A4
$\alpha_{CUR}$ – strana 3 a 7
Rovnice samotného čidla – strana 15

**Magnetický enkodér pro setrvačník**

Rovnice pro pozici a rychlost setrvačníku

$$\boldsymbol{y} = \frac{\boldsymbol{u} \cdot 2\pi}{CPR \cdot K} \tag{4}$$

Hodnoty:
$CPR = 48 \quad (rozlišení\ enkodéru\ motoru)$
$K = 20{,}4 \quad (převodový\ poměr\ motoru)$

**Teplotní čidlo**

Teplota rezistorů 1 a 2, teplota okolí
Rovnice

$$\boldsymbol{y} = \frac{\frac{\boldsymbol{u}}{R_{ADC}} \cdot V_{REF} - V_{TEMP0}}{\alpha_{TEMP}} \tag{5}$$

Hodnoty:
$V_{REF} = 3000\ mV \quad (referenční\ napětí)$
$R_{ADC} = 4096 \quad (rozlišení\ ADC)$
$V_{TEMP0} = 400\ mV \quad (offset\ pro\ 0°C)$
$\alpha_{TEMP} = 10\ mV/°C \quad (teplotní\ koeficient)$

Údaje v datasheetu:
Typ: MCP9700T-E/LT
$\alpha_{TEMP}$ – strana 2
$V_{TEMP0}$ – strana 2
Rovnice samotného čidla – strana 8

**Akcelerometr, gyroskop**

Rovnice pro akcelerometr

$$\boldsymbol{y} = \frac{(\boldsymbol{u} - O_{IMU})}{R_{IMU}} \cdot C_{ACC} \tag{6}$$

Rovnice pro gyroskop

$$\boldsymbol{y} = (\boldsymbol{u} - O_{IMU}) \cdot C_{GYR} \tag{7}$$

Hodnoty:
$O_{IMU} = 32\,768 \quad (IMU\ offset)$
$R_{IMU} = 32\,768 \quad (rozlišení\ IMU)$
$C_{ACC} = 2\ g \quad (rozlišení\ akcelerometru)$
$C_{GYR} = 61\ m°/s/LSB \quad (rozlišení\ gyroskopu)$

Údaje v datasheetu:
$C_{ACC}$ – strana 8
$C_{GYR}$ – strana 9
Registry pro nastavení citlivostí – strany 64-66

**Napětí baterie**

Rovnice

$$\boldsymbol{y} = \frac{\boldsymbol{u}}{R_{ADC}} \cdot V_{REF} \cdot G_{BAT} \tag{8}$$

Hodnoty:
$V_{REF} = 3000\ mV \quad (referenční\ napětí)$
$R_{ADC} = 4096 \quad (rozlišení\ ADC)$
$G_{BAT} = 2 \quad (poměr\ děliče\ napětí\ baterie)$

**Napětí kondenzátorů**

Rovnice

$$\boldsymbol{y} = \frac{\boldsymbol{u}}{R_{ADC}} \cdot V_{REF} \tag{9}$$

Hodnoty:
$V_{REF} = 3000\ mV$ $(referenční\ napětí)$
$R_{ADC} = 4096$ $(rozlišení\ ADC)$