
# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

## ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION


# AURIX TARGET V SYSTÉMU MATLAB SIMULINK
AURIX TARGET IN MATLAB SIMULINK

**DIPLOMOVÁ PRÁCE**
MASTER'S THESIS


**AUTOR PRÁCE** — Bc. Michael Chromiak
AUTHOR

**VEDOUCÍ PRÁCE** — doc. Ing. Petr Blaha, Ph.D.
SUPERVISOR



**BRNO 2020**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

***Student:*** Bc. Michael Chromiak
***Ročník:*** 2

***ID*:** 186093
***Akademický rok:*** 2019/20

**NÁZEV TÉMATU:**

## AURIX target v systému MATLAB Simulink

**POKYNY PRO VYPRACOVÁNÍ:**

1. Seznamte se s automatickým generováním jazyku C/C++ z prostředí MATLAB Simulink a možnostmi jeho nastavení. Nastavte konfiguraci pro procesor AURIX od firmy Infineon.
2. Nakonfigurujte vývojové prostředí od firmy Hightec pro programování a ladění procesorů AURIX s využitím knihoven iLLD.
3. Seznamte se s metodami Processor In the Loop (PIL), Software In the Loop (SIL) a Hardware in the Loop (HIL) a jejich podporou v prostředí MATLAB Simulink.
4. Zprovozněte SIL simulaci na zvoleném příkladu v Simulinku.
5. Realizujte PIL simulaci mezi PC a AURIX procesorem TC277 přes sériové rozhraní, případně přes Ethernet.
6. Proveďte srovnání vyzkoušených metod simulací.

**DOPORUČENÁ LITERATURA:**

[1] P. Karban: Výpočty a simulace v programech MATLAB a Simulink. Computer Press, a.s., ISBN 978-80-2-1-1448-3, 2006.

Uživatelské manuály k programu MATLAB a jeho nástrojům (Simulink, Simulink Coder, ...).

***Termín zadání:*** 3.2.2020 ***Termín odevzdání:*** 1.6.2020

***Vedoucí práce:*** doc. Ing. Petr Blaha, Ph.D.

**doc. Ing. Václav Jirsík, CSc.**
*předseda oborové rady*

# Abstrakt

Táto diplomová práca sa zaoberá realizáciou SIL a PIL simulácie pre mikrokontrolér Aurix TriCore TC277D vykonávanou v prostredí Matlab/Simulink.
Realizácia simulácie predstavuje okrem iného aj implementáciu simulovaného modelu do mikrokontroléra, ako aj tvorbu rozhrania pre komunikáciu mikrokontroléra s programom Matlab Simulink. Správnosť SIL a PIL simulácie bola overená porovnaním simulovaných priebehov zo simulácie v Simulinku. Simulovaným modelom bol použitý tepelný model kabíny automobilu vytváraný v predchádzajúcej bakalárskej práci autora tejto diplomovej práce. Model v TC277D, ako aj konfigurácia pre SIL/PIL je vytvorená pre použitie s programovacím jazykom C. Súčasťou práce je taktiež návod, podľa ktorého je možné model a konfiguráciu upraviť tak, aby bolo možné uvedené simulácie vykonávať na akomkoľvek zariadení obsahujúcom potrebný softvér.
Z porovnania odsimulovaných údajov jednotlivých simulácii je jasné, že vytvorenú konfiguráciu je možné použiť pre SIL alebo PIL simuláciu.

# Klíčová slova

Matlab, Simulink, SIL/PIL Simulácia, Aurix, TriCore, TC277D

# Abstract

This diploma thesis deals with the implementation of SIL and PIL simulation for the microcontroller Aurix TriCore TC277D performed in the Matlab Simulink. The realization of the simulation represents, among other things, the implementation of the simulated model into the microcontroller, as well as the creation of an interface for the communication of the microcontroller with the Matlab Simulink. The accuracy of SIL and PIL simulations was verified by comparing the simulated waveforms from the simulation in Simulink. The simulated model used a thermal model of a car cabin created in the previous bachelor's thesis of the author of this diploma thesis. The model in TC277D, as well as the configuration for SIL / PIL is created for use with the C programming language. The work also includes instructions according to which the model and configuration can be modified so that the simulations can be performed on any device containing the necessary software. From the comparison of the simulated data is it clear that the created configuration can be used for SIL or PIL simulation.

# Keywords

Matlab, Simulink, SIL/PIL Simulation, Aurix, TriCore, TC277D

## Bibliografická citace:

CHROMIAK, Michael. *AURIX target v systému MATLAB Simulink.* Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/126885. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Petr Blaha.

## Prohlášení

„Prohlašuji, že svou diplomovou práci na téma AURIX target v systému Matlab Simulink jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 23. května 2020 ……

podpis autora

## Poděkování

Ďakujem vedúcemu diplomovej práce doc. Ing. Petrovi Blahovi Phd. za účinnú metodickú, pedagogickú a odbornú pomoc, ako aj za všetky cenné rady pri spracovaní mojej diplomovej práce. Rád by som môjmu vedúcemu práce taktiež poďakoval za umožnené konzultácie, ako aj za čas ktorý mi bol ochotný venovať. V neposlednej rade patrí taktiež poďakovanie mojej priateľke a rodine, bez ktorých by som túto prácu nemohol dokončiť.

V Brně dne: 23. května 2020

…….
podpis autora

# Obsah


1 Úvod ..... 1
2 MATLAB SIMULINK ..... 2
2.1 TOOLBOXY PROGRAMU MATLAB-SIMULINK ..... 2
2.2 MATLAB CODER ..... 5
2.2.1 Embedded Coder ..... 5
2.2.2 Automatické generovanie kódu v jazyku C pre Matlab-Simulink ..... 6
2.2.3 Nové rysy prostredia Matlab-Simulink pre generovanie kódu ..... 8
2.3 „IN THE LOOP“ SIMULÁCIE ..... 9
2.3.1 MIL ..... 10
2.3.2 SIL ..... 11
2.3.3 PIL ..... 11
2.3.4 HIL ..... 12
2.4 PODPORA SIL/PIL SIMULÁCII V PROSTREDÍ MATLAB SIMULINK ..... 13
2.4.1 Podporované platformy Matlabu ..... 14
3 AURIX TARGET ..... 14
3.1 AURIX TRICORE TC277D ..... 15
3.2 KOMUNIKÁCIA MIKROKONTROLÉRA A PC ..... 16
3.2.1 Sériová komunikácia ..... 16
3.2.2 CAN ..... 19
3.2.3 LIN ..... 19
3.2.4 ETHERNET ..... 19
3.3 DÁTOVÁ INTEGRITA ..... 20
3.3.1 Kontrolný súčet – checksum ..... 20
3.3.2 Handshake a Potvrdzovanie správ ..... 21
3.3.3 Zmena baudrate ..... 21
3.4 HW a SW KONFIGURÁCIA PRE AURIX ..... 22
3.4.1 Hightec Free TriCore Entry Toolchain ..... 22
3.4.2 BIFACES ..... 23
3.4.3 Infineon Memtool ..... 23
3.4.4 DAS – Device Access Server ..... 24
3.4.5 Dávkové súbory ..... 25
3.4.6 Debug konfigurácia ..... 26
3.5 PROCES KOMPILÁCIE ..... 27
3.5.1 Nastavenia pre GNU Compiler Collection ..... 29
3.5.2 Makefile ..... 31
3.5.3 Toolchain ..... 32

4 PIL SIMULÁCIA MEDZI PC A AURIX ..... 32
4.1 ŠTRUKTÚRA BuildTool Class ..... 32
4.1.1 Popis jednotlivých konfiguračných skriptov v Matlabe ..... 33
4.1.2 Súbory v zložke +AurixTriCorePIL ..... 34
4.1.3 Súbory v zložke AurixTriCore_tc_setup ..... 34
4.2 PRISPÔSOBENIE „Build“ PROCESU V MATLABE ..... 35
4.2.1 Programové utility ..... 36
4.3 TESTOVANIE FUNKČNOSTI ..... 38
4.3.1 Test Recv-Send ..... 38
4.3.2 Test rtiostreamtest ..... 39
4.3.3 Test Toolchain ..... 40
4.4 SPUSTENIE SIL/PIL SIMULÁCIE ..... 40
4.4.1 Simulačná schéma ..... 41
4.5 POROVNANIE SIMULÁCII ..... 43
4.5.1 Diskrétny integrátor ..... 43
4.5.2 Generátor obdĺžnikového signálu ..... 46
4.5.3 Sínusoida ..... 49
4.5.4 Generátor náhodných čísiel ..... 51
4.6 ZÁVEREČNÉ USTANOVENIA ..... 53
4.7 NÁVOD NA POUŽITIE ..... 54
Záver ..... 56
Literatúra ..... 57
Zoznam symbolov, veličín a skratiek ..... 61

# Zoznam obrázkov a grafov

Obr. 1: Aurix Tricore TC 277D 15
Obr. 2: Aurix Tricore TC 277D 15
Obr. 3: Ukážka dátového rámcu 18
Obr. 4: Vzhľad programu Infineon Memtool 24
Obr. 5: Okno programu DAS Device Info s pripojeným TriCore TC 277D 25
Obr. 6: Znázornenie procesu kompilácie [25] 28
Obr. 7: Štruktúra BuildTool class použitá pre PIL simuláciu[37] 33
Obr. 8: Porovnanie implementovanej sériovej komunikácie 40
Obr. 9: Ukážka simulovaných modelov 41
Obr. 10: Ukážka PIL simulácie subsystému „model PIL“ 42
Obr. 11: Ukážka simulovaného modelu diskrétneho integrátoru 43
Obr. 12: Výstupné porovnanie signálov SIL simulácie 44
Obr. 13: Výstupné porovnanie signálov PIL simulácie 45
Obr. 14: Ukážka simulovaného modelu generátoru obdĺžnikového signálu 46
Obr. 15: Výstupné porovnanie signálov SIL simulácie 47
Obr. 16: Výstupné porovnanie signálov PIL simulácie 48
Obr. 17: Ukážka simulovaného modelu sínusového signálu 49
Obr. 18: Výstupné porovnanie signálov SIL simulácie 50
Obr. 19: Výstupné porovnanie signálov PIL simulácie 51
Obr. 20: Ukážka simulovaného modelu generátoru náhodných čísel 51
Obr. 21: Výstupné porovnanie signálov PIL simulácie 52

# Zoznam tabuliek

Tab.1: Súhrn toolboxov v Matlabe [7] 3
Tab.2: Súhrn toolboxov v Simulinku [7] 4
Tab.3: Podporované typy Infineon hardvéru [24] 6

# 1 ÚVOD

Pre verifikáciu modelovaných systémov existuje niekoľko typov simulácii. Jedným z nich je aj metóda simulácie Processor In the Loop spolu so Software in the Loop, o ktorých pojednáva táto práca.

Pred začiatkom riešenia zadania samotnej práce je potrebné vytvoriť prehľad teoretických znalostí, ktoré budú potrebné pre správne riešenie zadanej diplomovej práce.

Hlavným cieľom diplomovej práce je spustenie SIL simulácie a vytvorenie funkčnej PIL simulácie, ktorá bude prebiehať pomocou sériového rozhrania medzi mikrokontrolérom Aurix Tricore TC277D a programom Matlab, ktorý bude spustený na počítači.

Program vytváraný pre mikrokontrolér bude automaticky generovaný v jazyku C, preto je potrebné zoznámiť sa nielen so širokými možnosťami konfigurácie automatického generovania kódu v Matlabe a s technickými prostriedkami pre mikrokontrolér Aurix, ale aj s celkovou konfiguráciou potrebnou pre spustenie PIL simulácie, ktorej popis bude uvedený v ďalších častiach tejto diplomovej práce.

Samotná práca nebude predstavovať iba teoretický základ pre vytvorenie PIL simulácie, ale bude predstavovať aj stručný návod, pomocou ktorého bude možné tento postup zopakovať pre ďalšie mikrokontroléri Aurix.

# 2 MATLAB SIMULINK

Jedná sa o jeden z najpoužívanejších softvérových vybavení pre technické výpočty vo viacerých technických oboroch. [1] Do jedného programovacieho prostredia spája možnosť technických výpočtov vizualizácie rôznych foriem dát, ako aj vlastný programovací jazyk. Samotné vývojové prostredie Matlab pre výpočty obsahuje pomerne veľké množstvo takzvaných toolboxov, čo sú vlastne rozšírenia pôvodného programového balíku pre špecifické použitie.
Takýmto najrozšírenejším toolboxom je Simulink.
Simulink je určený pre modelovanie, simuláciu a analýzu dynamických systémov. S daným toolboxom sa priamo nepracuje ako s klasickými programovacími jazykmi, alebo ako aj s jazykom Matlab formou textového programovania, ale jedná sa o grafické programovacie prostredie. Samotný Simulink využíva pre riešenie systémov algoritmy z Matlabu. Vytvárané systémy je možné inicializovať a aj ovládať pomocou skriptov z Matlabu, do ktorého je taktiež možné ukladať dáta zo simulácii v Simulinku.
Programovací jazyk Matlabu má množstvo výhod, ktoré uľahčujú prácu pri riešení technických problémov. Medzi tieto výhody patrí najmä optimalizácia maticových výpočtov, ako aj podpora viacrozmerných polí a dátových štruktúr.
Veľkou výhodou Matlabu je aj široká možnosť spracovania a vizualizácie dát. Obsahuje množstvo pred vytvorených funkcii, ktorými je možné veľmi jednoducho vytvoriť grafické rozhranie a zobrazovať nielen obrazové dáta. Samotné dáta je možné importovať, ale aj exportovať v rôznych formátoch, čo taktiež uľahčuje prácu s externými zariadeniami, ktoré sú schopné s Matlabom spolupracovať.
Matlab je licencovaný rozšíriteľný systém s veľkým množstvom aplikačných knižníc pomocou ktorých je možná komunikácia s externými zariadeniami, a to dokonca aj v reálnom čase. [1] Pre takéto spojenie sú ale potrebné jednotlivé toolboxy Matlabu, ktoré budú opísané v nasledujúcej kapitole.

## 2.1 TOOLBOXY PROGRAMU MATLAB-SIMULINK

Ako už bolo spomenuté, existuje veľké množstvo toolboxov uľahčujúcich prácu pri práci so softvérom Matlab Simulink. Tieto toolboxy sú rozdelené do jednotlivých kategórii, ktoré znázorňujú nasledujúce tabuľky Tab.1 a Tab.2. Spoločnosť Mathworks ponúka aj ďalšie aplikačné produkty, ktoré rozširujú použiteľnosť jednotlivých toolboxov.
Pre použitie jednotlivých toolboxov je potrebné licencovať každý typ toolboxu samostatne. Simulink obsahuje okrem toolboxov aj SystemComposer, ktorý sa používa pre návrh a automatické generovanie správ pre Matlab aplikácie.

Tab.1: Súhrn toolboxov v Matlabe [7]

| Toolboxy Matlabu | | |
|---|---|---|
| **Typ toolboxu** | **Názov toolboxu** | **Všeobecný opis toolboxu** |
| **Parallel Computing** | Parallel Computing Toolbox | Umožňuje paralelný výpočet |
| | MATLAB Parallel Server | Spojenie paralelných výpočtov s cloudovými službami |
| **Math and Optimization** | Curve Fitting Toolbox | Grafická tvorba a úprava kriviek a plôch |
| | Optimization Toolbox | Funkcie pre hľadanie parametrov pre min./max. pri riešení |
| | Global Optimization Toolbox | Funkcie hľadanie globálnych riešení obsahujúcich minimá/maximá |
| | Symbolic Math Toolbox | Vykonávanie symbolických mat.výpočtov |
| | Mapping Toolbox | Práca s geografickými informáciami |
| | Partial Differential Equation Toolbox | Riešenie parciálnych dif.rovníc |
| **AI, Data Science, Statistics** | Statistics, Machine Learning Toolbox | Analýza a modelovanie dát |
| | Deep Learning Toolbox | Tréning, tvorba a analýza sietí hlbokého učenia |
| | Reinforcement Learning Toolbox | Tvorba rozhodovacích algoritmov |
| | Text Analytics Toolbox | Analýza a modelovanie textových dát |
| | Predictive Maintenance Toolbox | Nástroje pre prediktívnu údržbu |
| **Code Generation** | MATLAB Coder | Generovanie C,C++ kódu z Matlab kódu |
| | Embedded Coder | Generovanie C,C++ kódu z Matlab kódu pre vstavané systémy |
| | HDL Coder | Generovanie VHDL, Verilog kódu z Matlab kódu |
| | HDL Verifier | HDL a FPGA simulátor |
| | Filter Design HDL Coder | Tvorba VHDL a Verilog kódu pre filtre s FP |
| | Fixed-Point Designer | MaO algoritmov s FP a pohyblivou r.č. |
| | GPU Coder | Generovanie CUDA kódu z Matlab kódu |
| **Application Deployment** | MATLAB Compiler | Tvorba spustiteľných a webových aplikácii |
| | MATLAB Compiler SDK | Rozšírenie funkcionality MATLAB Compiler |
| | MATLAB Production Server | Tvorba vlastných analýz |
| **Application Deployment** | Database Toolbox | Výmena dát medzi databázami |
| | MATLAB Report Generator | Tvorba reportov z Matlab aplikácii |

Súčasťou Matlab a Simulink sú aj produkty softvéru Polyspace, ktorý ale nebude použitý pre túto prácu, preto ani nebudú viac spomínané.
Ako znázorňuje popis jednotlivých toolboxov v Matlabe, ako aj Simulinku, bude kľúčovým toolboxom pre túto prácu práve toolbox Code Generation, ktorý bude podrobnejšie opísaný v ďalších častiach tejto práce. Bude sa jednať o Matlab, Simulink a Embedded coder.

Tab.2: Súhrn toolboxov v Simulinku [7]

| Toolboxy Simulinku | | |
|---|---|---|
| **Typ toolboxu** | **Názov toolboxu** | **Všeobecný opis toolboxu** |
| **Event-Based Modeling** | Stateflow | Využitie stavových a vývojových diagramov |
| | SimEvents | MaS systémov diskrétnych udalostí |
| **Physical Modeling** | Simscape | MaS viacdoménových fyzikálnych systémov |
| | Simscape Driveline | MaS rotačných/translačných m.sys. |
| | Simscape Electrical | MaS elektrických,mechatronických a EE sys. |
| | Simscape Fluids | MaS hydraulických systémov |
| | Simscape Multibody | MaS 3D mechanických systémov |
| **Real-Time Simulation and Testing** | Simulink Real-Time | Tvorba aplikácii reálneho času |
| | Simulink Desktop Real-Time | Tvorba Simulink modelov pre reálny čas |
| **Code Generation** | Simulink Coder | Generovanie C,C++ kódu zo Simulinku |
| | Embedded Coder | Generovanie C,C++ kódu zo Simulinku pre vstavané systémy |
| | AUTOSAR Blockset | Tvorba AUTOSAR softwaru |
| | Fixed-Point Designer | MaO algoritmov s FP a pohyblivou r.č. |
| | Simulink PLC Coder | Generovanie STL a LAD kódu |
| | Simulink Code Inspector | Overovanie bezpečnostných noriem pre generovaný kód |
| | DO Qualification Kit (for DO-178) | Zjednodušenie certifikácie generovaných kódov zo Simulinku a Polyspace produktov |
| | IEC Certification Kit (for ISO 26262 and IEC 61508) | Zefektívnenie certifikácie zabudovaných systémov |
| | HDL Coder | Generovanie VHDL, Verilog kódu zo Simulinku |
| | HDL Verifier | HDL a FPGA simulátor |
| **Verification, Validation, and Test** | Simulink Requirements | Správa a analýza generovaného kódu |
| | Simulink Check | Identifikácia chýb vývoja modelu podľa definovaných štandardov |
| | Simulink Coverage | Analýza testovania v modeloch a generovaných kódoch |
| | Simulink Design Verifier | Tvorba testov a identifikácia chýb v modeloch |
| | Simulink Test | Vývoj, správa a vykonávanie simulačných testov |
| **Simulation Graphics and Reporting** | Simulink 3D Animation | Vizualizácia dynamických systémov |
| | Simulink Report Generator | Tvorba reportov zo Simulink aplikácii |

## 2.2 MATLAB CODER

Pomocou toolboxu Matlab Coder je možné automaticky generovať kód v jazyku C a C++ zo skriptov a funkcii vytvorených v jazyku Matlab. [3] Samotný automaticky generovaný kód je vytváraný podľa normy, ktorú si je možné ľubovoľne zvoliť z nasledujúcich možností: ANSI C, ISO C, ANSI C++, ISO C++. [24]

### 2.2.1 Embedded Coder

Jedná sa o toolbox, ktorý je široko využívaný a nenahraditeľný pri vytváraní SIL/PIL simulácie.
Pomocou Embedded Coderu je možné vytvoriť automaticky generovaný zdrojový kód, ktorý je možné používať pre vstavané embedded systémy.
Ako sme už spomenuli, jedná sa o toolbox a tie rozširujú možnosti využitia Matlabu a Simulinku. Tento toolbox dokonca rozširuje aj ďalšie toolboxy, a to konkrétne funkcionalitu toolboxov Matlab Coder a Simulink Coder.
K vylepšeniam, ktoré tento toolbox umožňuje patrí napríklad pokročilá optimalizácia pre presnú kontrolu automaticky generovaných funkcii, ako aj kontrolu súborov a dát pre generovanie kódu.
Pomocou týchto optimalizácii je taktiež umožnená možnosť zlepšenie efektivity generovaného kódu, ako aj možnosť použitia vývojových nástrojov pre generovanie kódu takzvane tretích strán, ako je napríklad aj v našom prípade Infineon. Pomocou tejto možnosti je možné vytvoriť spustiteľné súbory pre vstavané systémy.
Taktiež je pri generovaní kódu možné vytvoriť automaticky okrem samotného zdrojového kódu, ktorý je vytváraný tak, že je ho možné preniesť a skompilovať na akomkoľvek procesore, aj automatické generovanie komentárov a dokumentácie vytvoreného zdrojového kódu.
V samotnom toolboxe sa taktiež nachádzajú podporné balíčky s ovládačmi viacerých hardvérových zariadení, pre ktoré je generovanie kódu určené. [23] Medzi nimi je rovnako možnosť generovania kódu pre zariadenia od výrobcu Infineon. Konkrétne typy, ktoré je možné pre automatické generovanie kódu od uvedeného výrobcu použiť znázorňuje nasledujúca tabuľka Tab. 3.

Tab.3: Podporované typy Infineon hardvéru [24]

| Infineon | | | |
|---|---|---|---|
| **XC800** | **XC2000/XE166** | **XMC Family** | **TriCore** |
| XC800 A | XC2000 | XMC1000 | AURIX™ |
| XC800 I | XE166, XC16x, C166 | XMC4000 | AUDO |
| | | | TC1100 |

Ako z hore uvedenej tabuľky Tab. 3 vyplýva, tak bude možné nami používaný mikrokontrolér využiť pre automatické generovanie kódu pomocou Embedded Coder toolboxu v spolupráci s toolboxami Matlab Coder, Simulink Coder a Simulink Real-Time.
Pomocou nich vygenerovaný kód bude možné daný zdrojový kód skompilovať a spustiť na danom mikrokontroléri.
Embedded Coder síce umožňuje vygenerovanie kódu, ktorý bude možné skompilovať a spustiť na nami využívanom Aurix TriCore, avšak neponúka podporu samotnej kompilácie a nahrávania kódu, ktorú bude potrebné zabezpečiť za pomoci ďalších externých softvérových prostriedkov. [24]

### 2.2.2 Automatické generovanie kódu v jazyku C pre Matlab-Simulink

Automatické generovanie kódu v skripte Matlabu je umožnené spustiť pomocou príkazu „codegen“.
Automatické generovanie kódu v Matlabe je možné vykonať pre viacero programovacích jazykov, avšak pre náš účel poslúži iba jazyk C.
Samotné generovanie kódu je uskutočniteľné buď z Matlabu použitím „codegen“ v skripte alebo stlačením tlačidla Build model v prostredí Simulinku.
Generovanie kódu je ale potrebné vždy nastaviť tak, aby prebehlo správne a dosiahol sa čo najlepší výsledok, nakoľko generovaný kód, ktorý je vytvorený automaticky je bez potrebných nastavení pomerne ťažko čitateľný.

#### 2.2.2.1 Nastavenie generovania kódu

V softvére Matlab Simulink je možné pre automatické generovanie kódu nastaviť veľké množstvo parametrov.
Na zistenie o ktoré sa konkrétne jedná postačuje spustiť v Simulinku konfiguračné parametre a rozkliknúť si záložku „Code Generation“.

Následne uvidíme všetky možnosti, ktorými je možné nastaviť generovanie kódu pre Matlab a Simulink.
Rozdielny je medzi nimi iba spôsob, v ktorom u Simulinku stačí označiť jednotlivé parametre v okne konfiguračných parametrov a pre Matlab je potrebné vytvoriť skript, ktorý bude definovať dané nastavenia.
Nastavenia v skripte Matlabu by mali v podstate iba kopírovať nastavenia z okna konfiguračných parametrov v Simulinku.
Týmito nastaveniami je možné určiť výber cieľového zariadenia pre ktoré bude kód generovaný. Táto časť sa určuje pomocou „ .tlc “ súboru, ktorý predstavuje smernicu prekladača pre vybraný programovací jazyk, ktorý je v prípade výberu pre „embedded“ systémy možné vybrať iba medzi C a C++.
Pomocou „ .tlc “ súborov je možné určovať, akým spôsobom sa bude generovať kód z modelov v Simulinku. [27]
Ďalej je možné vybrať možnosť, či bude generovaný iba samotný kód, alebo či má byť generovaný kód zabalený do archívu.
Ďalším veľmi dôležitým nastavením je výber takzvaného Toolchain, ktorým sa definuje kompilátor, linker, archiver a ostatné súčasti potrebné pre generovanie kódu.

Taktiež je možné nastaviť optimalizáciu generovania kódu.
Je možné si zvoliť, či budú premenné vytvárané ako štruktúry alebo iba ako samostatné premenné, či môžu byť vytvárané bitové polia alebo paralelné „for“ cykly. Samozrejme týchto parametrov je ďaleko viac.

V nastaveniach generovania kódu je možné taktiež zvoliť možnosť tvorby reportu o generovaní kódu, ako aj možnosť zahrnúť do vygenerovaného kódu aj komentáre z popisu blokov Simulinku alebo zo skriptov v Matlabe.

Ďalšou možnosťou na výber je, či budú podporované pre generovanie kódu aj čísla s pohyblivou desatinnou čiarkou, komplexné alebo aj nekonečné čísla.
Taktiež je možné zvoliť podporu pre spojitý čas, signály s variabilnou veľkosťou alebo aj absolútny čas.

Je možné taktiež optimalizovať štýl písania automaticky generovaného kódu. Je možné zvoliť úroveň ozátvorkovania kódu, možnosť nahradenia násobenia, exponentov na druhú pomocou bitových posunov alebo možnosť voľby podmienok pre prípad, či chceme využívať switch-case príkazy alebo if-else.

Taktiež je možné verifikovať kód pre SIL a PIL simuláciu pomocou voľby programov tretích strán. Ponúka sa taktiež možnosť voľby debugovania pre SIL simuláciu.
Z radu jednotlivých nastavení je dobré spomenúť rovnako možnosť zmeny názvov jednotlivých dátových typov.

### 2.2.3 Nové rysy prostredia Matlab-Simulink pre generovanie kódu

Od prvej verzie toolboxu pre automatické generovanie kódu, ktorá bola uvedená na trh v roku 1999 uplynulo už niekoľko rokov, počas ktorých došlo ku viacero rozšíreniam a vylepšeniam. [15]
Najnovšia verzia Matlabu v čase písania tejto práce je verzia R2020a.
Samotná práca bola v počiatku vytváraná za pomoci študentskej verzie Matlabu R2015b. V priebehu tvorby bola ale sprístupnená licencia pre študentskú verziu R2019b. Pre tvorbu tejto diplomovej práce sa začala používať daná verzia R2019b. Z tohto dôvodu budú opisované novinky z verzie R2015b, ako aj R2019b, ktoré majú vplyv na vykonávanie a tvorbu PIL simulácie.

#### 2.2.3.1 Zmeny v generovaní kódu podľa verzie Matlabu R2015b

Ako sa dá poznať z webovej stránky Mathworks, tak každá verzia Matlabu prešla veľkými zmenami a ani táto nebola výnimkou.
Pre verziu R2015b prišli úpravy pre viaceré toolboxy, avšak zmien týkajúcich sa generovania kódu v jazyku C prišlo iba u toolboxov s názvom Matlab Coder, Embedded Coder a Simulink Coder.
Všeobecne najvýznamnejšou zmenou bola možnosť začať využívať voľne dostupný kompilátor pre generovanie kódu. Jedná sa o kompilátor MinGW-w64.[16]

Čo sa týka zmien u Matlab Coderu, tak pre túto verziu bolo novinkou možnosť generovania kódu v jazyku C aj s podporou bunkových polí. [17]

Pre Embedded Coder prišla zmena pri konfigurácii modelov pri generovaní kódu, ktorá umožňovala rýchlejšiu konfiguráciu. [17]

V toolboxe Simulink Coder bolo možné začať používať viacjazyčné názvy jednotlivých blokov a signálov v modeli alebo aj komentáre funkcii, táto funkcia bola sprístupnená taktiež aj pre ďalšie toolboxy v Simulinku. [17]

#### 2.2.3.2 Zmeny v generovaní kódu podľa verzie Matlabu R2019b

V uvedenej verzii softvéru Matlab SImulink prišlo viacero vylepšení.
Pomerne výraznými vzhľadovými zmenami prešiel Simulink. Klasický hlavný panel so všetkými nástrojmi nahradila možnosť výberu rôznych aplikácii, podľa ktorého sa hlavný panel upraví. Jednou z takýchto aplikácii, ktorá je pre SIL alebo PIL simuláciu veľmi potrebná je SIL/PIL Manager. V tejto verzii oproti staršej, nami používanej verzie, už bolo možné využiť aj práve uvedený SIL/PIL Manager. Jedná sa o komplexný nástroj, ktorý výrazne zjednodušuje verifikáciu modelu, z ktorého sa automaticky vygeneroval zdrojový kód pre mikrokontrolér. Samotné spustenie SIL alebo PIL simulácie je možné po výbere konkrétneho typu simulácie spustiť kliknutím na zelené tlačidlo „Run verification“. [38]
Veľkou pomôckou je aj použitie Simulation Data Inspector, ktorým je možné taktiež veľmi ľahko kontrolovať a porovnávať simulované údaje získané pomocou SIL/PIL Manager. [40]
V tejto verzii Simulinku je taktiež možné vytvárať Subsystémové referencie z modelov, ktoré je možné použiť pre SIL/PIL simulácie, takže je možné vykonávať PIL simuláciu aj pre jednotlivé subsystémy modelu, ako aj pre celý model. [39]

## 2.3 „IN THE LOOP“ SIMULÁCIE

Názov tejto kapitoly je odvodený od názvu typov simulácii, ktoré sú potrebné pri návrhu softvéru pre vstavané aplikácie.
Jedná sa o pojem pochádzajúci z modelovania systémov určených pre vstavané riadiace systémy.
V prípade vytvárania kódu pre takéto systémy, je bez ich simulácie možné prvotné testovanie daného kódu až na fyzickom prototype.
V prípade, že sa zmenia požiadavky alebo sa objaví chyba v kóde, tak sa potrebné zmeny budú vykonávať až na fyzickom prototype, keď bude už celý systém naprogramovaný a bude ho potrebné vo veľkej miere prerobiť.
Samotné testovanie na fyzickom hardvéri býva často taktiež cenovo veľmi náročné.
Pomocou tvorby kódu založeného na dizajnovaní modelov v Matlabe a Simulinku, je možné kód vytvoriť automatickým generovaním kódu miesto ručného písania a taktiež je možné vytvoriť systémový model, ktorý je ľahšie modifikovateľný no hlavne je ho možné simulovať aj bez použitia koncového hardvéru.
Vďaka tomu, je možné získať dôležité informácie týkajúce sa správanie systému vo viacerých situáciách, ktoré sa ťažko na reálnom hardvéri testujú alebo sú dokonca nebezpečné. [8]

Je taktiež možné overiť, že naprogramovaný ovládací softvér bude správne fungovať ešte skôr, než bude použitý v hardvéri, pre ktorý bude vytváraný. Pomocou týchto simulácii je možné nahradiť akúkoľvek časť alebo aj úplne celý hardvér, na ktorý bude vytváraný jeho ovládací softvér.
Takýchto modelovaní je niekoľko typov a budú popísané v nasledujúcich podkapitolách.

Pre samotné modelovanie systému je v základe potrebné pomocou softvéru Matlab Simulink využitie toolboxov Simulink Control design, Stateflow alebo Simscape. Následne bude pre vývoj koncovej aplikácie potrebné využiť automatické generovanie kódu, ktoré bude zabezpečovať pre jazyk C toolbox Simulink coder a Embedded coder. Pre záverečnú verifikáciu a testovanie vytvoreného kódu je možné zo Simulinku použiť toolboxy Test, Check, Coverage, Real-Time. Samozrejme, ktorý konkrétne závisí už od samotnej aplikácie a druhu simulácie.

### 2.3.1 MIL

Jedná sa o simuláciu vstavaných systémov v počiatočných častiach vývoja modelovania. V tejto fázy vývoja sa overujú algoritmy iba v simulačnom prostredí. [20]
Pri tomto type simulácie je potrebné vytvoriť nielen samotný model, ale aj nasimulovať podmienky jeho používania.
V prípade, že by sa napríklad jednalo o prístroj, ktorý by obsahoval rôzne druhy snímačov pre získavanie dát, tak je potrebné nasimulovať aj charakteristiky daných snímačov, ako sú ich časové konštanty.
Samotná simulácia prebieha v systémoch s reálnym časom.
Takéto modelovanie nesie svoje výhody najmä v tom, že je možné odsimulovať aj systémy, ktoré ešte neboli fyzicky vyrobené, avšak rovnako vzniká problém v nasimulovaní prevádzkových podmienok daných systémov. A to hlavne v prípade, ak sa jedná o prírodné podmienky. [18]
Pri MIL simulácii je v jednom počítači využité simulačné prostredie pre simuláciu matematického modelu, ako aj systému využívaného pre riadenie daného modelu. [19]

### 2.3.2 SIL

Ďalšou fázou vývoja modelovania systémov je takzvaná SIL simulácia.
Software in the loop simulácia slúži najmä ku vyhodnocovaniu správania vygenerovaného zdrojového kódu, ktorý bude používaný v mikrokontroléri.
V tomto druhu simulácie je vytváraný model ďalej upravovaný pre použitie koncového vstavaného systému.
Model pre kontrolér je buď vytváraný alebo aj automaticky generovaný v jazyku C alebo C++ a následne je vrátený späť do simulačného prostredia v počítači, ktoré v našom prípade predstavuje softvér Matlab Simulink.
Daný kód bude vyhodnocovaný spolu so simuláciou vytváraného modelu.
Tento druh overovania je veľmi dôležitý pre vyvíjaný softvér, ktorý je potrebné spustiť na koncovej platforme, ktorá môže predstavovať napríklad mikrokontrolér.
Tento druh testovania využíva dátové a modelové štruktúry z predchádzajúceho kroku simulácie, a to konkrétne MIL simulácie.
Simulovaný model a softvér, ktorý by mal následne pracovať napríklad v mikrokontroléri u tohto druhu simulácie prebiehajú na počítači za pomoci softvéru, ktorý simuluje správanie daného mikrokontroléra. [20]

### 2.3.3 PIL

Skratka PIL simulácie predstavuje v preklade procesorovú simuláciu v slučke. PIL simulácia využíva krížovú kompiláciu vygenerovaného zdrojového kódu, po ktorej objektový kód stiahne a spustí na cieľovom hardvéri. [4]
Krížovú kompiláciu, z anglického cross-compile, špecifikuje práve to, že na strane počítača vykoná kompiláciu zdrojového kódu na binárne súbory, ktoré budú spustené v cieľovom hardvéri.
Pri krížovej kompilácii je dôležité rozlišovať medzi platformou, na ktorej sa bude vykonávať kompilácia kódu a platformou, na ktorej tento kód bude spustený. [5]

PIL simulácia v podstate poskytuje možnosť na overenie skutočného kódu, ktorý bude v mikrokontroléri, a ten spolupracuje so simuláciou prebiehajúcou vo vývojovom prostredí v počítači.
Hlavným účelom tejto simulácie je navrhovaný model v simulačnom softvéry, akým je Matlab vložiť do mikrokontroléra, v ktorom sa bude vykonávať daný kód.
Úlohou PIL je identifikácia potencionálnych chýb v generovanom kóde, ktoré môžu vzniknúť napríklad v kompilátore.

Taktiež je možné vyhodnotiť výkonnosť mikrokontroléra u výpočtov simulácie, ako aj ladiť funkčnosť algoritmov.
Ďalšou užitočnou možnosťou, ktorú umožňuje PIL simulácia je zaznamenávanie využitia pamäte, ako aj čas vykonávania kódu, načo je napríklad potrebné vytvoriť časovač v mikrokontroléri.
Pomocou týchto dát je možné upravovať potrebné časti kódu v zabudovanom hardvéri už v počiatočných fázach vývoja. [20]
Porovnaním výsledkov normálnej simulácie prebiehajúcej na počítači v Matlabe a simulácie PIL je možné otestovať numerickú rovnocennosť simulačného modelu a vygenerovaného kódu pre cieľový hardvér.
Výhodou PIL simulácie je taktiež možnosť získavania pokryteľnosti kódu a dĺžky vykonávania pre generovaný kód.
Pre potreby správneho nakonfigurovania krížovej kompilácie je veľmi dôležité vytvoriť konfiguráciu cieľového pripojenia. [4]
Samotný Simulink obsahuje aplikáciu pre verifikáciu kódu, ktorý bol vygenerovaný z modelu vytvorenom v Simulinku. Jej názov je SIL/PIL Manager. [6]
Pomocou ponúkaných možností je možné v Simulinku vykonávať PIL simuláciu na úrovni Top modelu, ako aj na jednotlivých blokoch simulačnej schémy.

### 2.3.4 HIL

Posledným krokom v oblasti verifikácie systémov pre vývoj je HIL simulácia.
Pod pojmom HIL simulácie je možné si predstaviť druh simulácie, ktorá je vykonávaná v reálnom čase.
Je možné pomocou nej testovať vytváraný model na základe reálnych podnetov, ako aj otestovať správnosť daného fyzikálneho modelu.
V podstate sa jedná o súbežnú simuláciu reálneho modelu, ako aj virtuálneho modelu, ktorý je vytvorený v počítači a je realizovaný vo vývojovom hardvéri predstavujúci mikrokontrolér, ktorý má k dispozícii aj rozhranie pre ovládanie modelu [13].
Ak je použitý mikrokontrolér ako vývojový hardvér pre HIL simuláciu, tak je potrebné z vytvoreného modelu vygenerovať kód v jazyku pre daný mikrokontrolér, čo je vo väčšine prípadov programovací jazyk C, ktorý je možné do mikrokontroléra pomerne jednoducho naprogramovať.
Pre automatické generovanie zdrojového kódu v jazyku C slúži pre Simulink toolbox s názvom Simulink Coder [14].

## 2.4 PODPORA SIL/PIL SIMULÁCII V PROSTREDÍ MATLAB SIMULINK

Prostredie Matlab Simulink ponúka pomerne veľkú podporu pre SIL a PIL simulácie.
Pre ich využitie je potrebné mať licencovaný toolbox s názvom Embedded Coder hlavne kvôli možnosti automatického generovania kódu.
Pre prostredie Matlab a aj Simulink existujú už pred vytvorené vzorové modely a skripty pre ich konfiguráciu.
Tieto vzorové príklady je možné pomerne rýchlo dohľadať v inštalačnej zložke Matlabu pomocou kľúčových slov SIL alebo PIL.
Pre SIL simuláciu postačuje tieto pred vytvorené modely a skripty iba spustiť.
V prípade PIL simulácie je ale potrebné vlastniť hardvér, pre ktoré je daná PIL simulácia vytváraná.
Naša verzia Matlabu obsahuje vytvorené súbory potrebné pre spustenie PIL simulácie pre mikrokontroléri od výrobcov ARM a Texas Instruments.
V samotnom vývojovom prostredí Matlabu a Simulinku je ale v nastaveniach konfiguračných parametrov možné nastaviť aj množstvo ďalších hardvérových dosiek mikrokontrolérov, ktoré je možné pre SIL a PIL simuláciu použiť.
Nakoľko je ale pre našu prácu použitý mikrokontrolér Aurix TriCore TC277D od výrobcu Infineon, tak tieto vzorové riešenia nie je možné pre naše potreby PIL simulácie použiť.
Z tohto dôvodu je teda potrebné vytvoriť úplne nové konfiguračné súbory pre všetky súčasti PIL simulácie.
Jedná sa taktiež o konfiguráciu komunikácie medzi mikrokontrolérom a Matlabom v počítači, ako aj súbory pre konfiguráciu hardvéru, ktorým je mikrokontrolér Aurix. Rovnako bude potrebné nastaviť spúšťanie a ovládanie samotnej PIL simulácie.

Aby dokázal Matlab generovať kód pre PIL simuláciu, je taktiež potrebné v premenných prostredia Windows v premennej s názvom PATH definovať nasledujúcu cestu k spustiteľnému súboru cmd.exe, čiže príkazovému riadku, ktorá je v našom prípade:

C:\Windows\System32

### 2.4.1 Podporované platformy Matlabu

Ako už bolo spomenuté, tak v dokumentácii Matlabu môžeme nájsť niekoľko vzorových riešení pre PIL simuláciu, avšak len pre niekoľko vybraných platforiem. Konfigurácia pre každú z nich je odlišná a vyžaduje si špecifické časti.
V tejto časti kapitoly diplomovej práce budú v základoch zhrnuté platformy, pre ktoré už je vytvorená konfigurácia a potrebné súbory pre plne funkčnú PIL simuláciu.

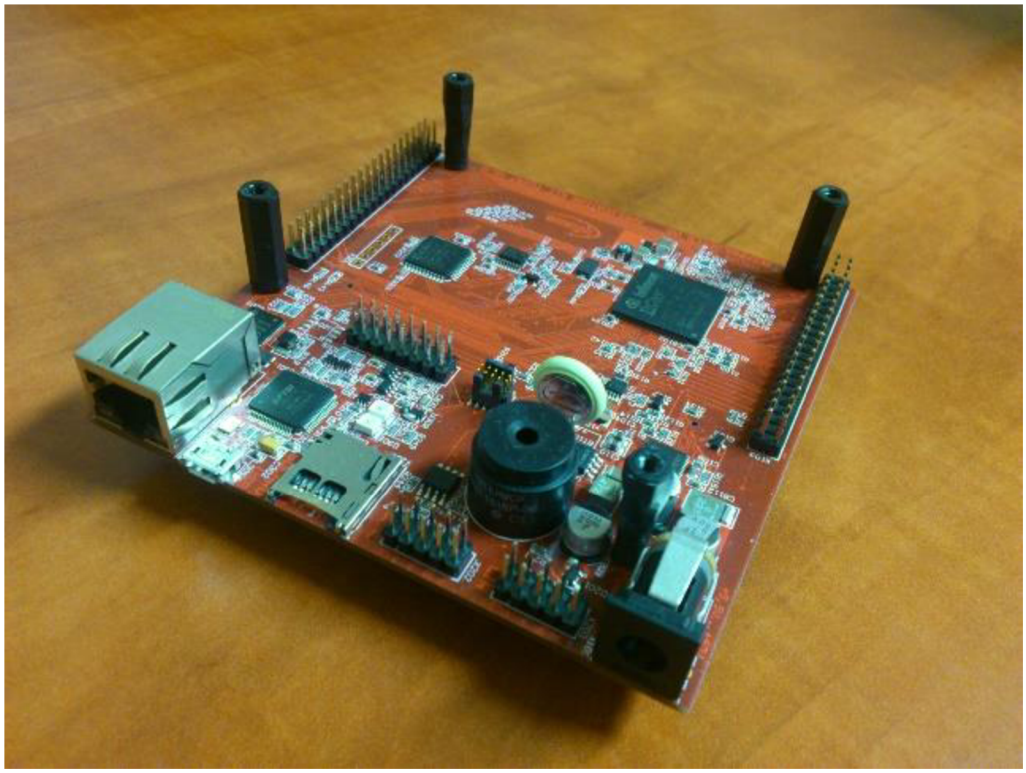
Podporovanými platformami, ktoré obsahujú už vytvorený target systém sú napríklad od výrobcov Texas Instrumens, STMicroelectronics, Raspberry a ARM, pre ktorý je aj vytvorený takzvaný QEMU emulátor.
QEMU emulátor je simulačný softvér, ktorý umožňuje beh programu z iných platforiem na počítači. Jedná sa ale iba o platformy ARM.
Na tento typ platforiem je možné pomocou QEMU emulátoru vykonávať overovanie vygenerovaného kódu bez toho, aby bol ku počítaču pripojený akýkoľvek fyzický hardvér.
Tento emulátor je ale funkčný, iba ak je vo vývojovom prostredí Matlab Simulink nainštalovaný Embedded Coder toolbox a verzia samotného Matlabu musí byť R2016a a novšia.
Samotný QEMU emulátor teda neumožňuje generovanie kódu ale slúži pre virtualizáciu fyzického hardvéru. [29]
Keďže podpora pre mikrokontroléri Aurix v tomto smere v čase vytvárania tejto diplomovej práce nebola dostupná, tak bolo potrebné využiť poznatky z vytvorených vzorových príkladov od ostatných výrobcov.
Základom pre tvorbu konfigurácie pre PIL simuláciu bol vzorový príklad od Raspberry Pi, ktorý je dostupný z webových stránok Mathworks. [36]

# 3 AURIX TARGET

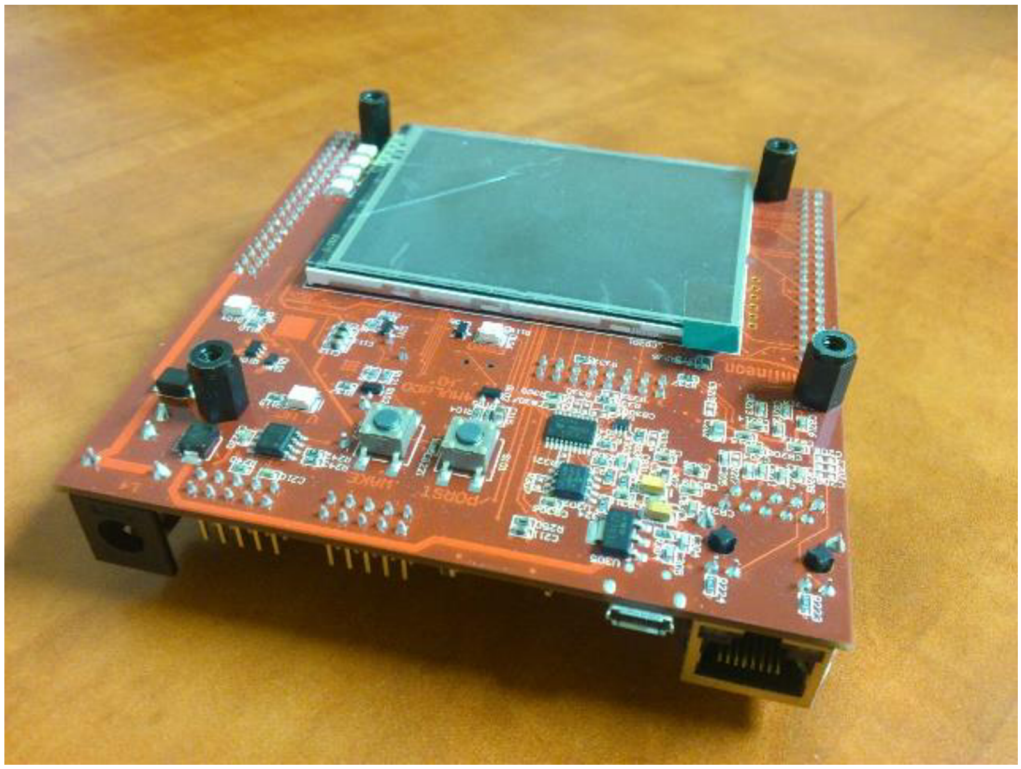
Výsledkom automatického generovania kódu budú spustiteľné súbory, ktoré budú spúšťané na cieľovej platforme.
Táto cieľová platforma sa nazýva Target a v našom prípade je týmto Targetom práve 32 bitový mikrokontrolér Aurix TriCore TC277D, ktorý bude opísaný v nasledujúcej kapitole.

## 3.1 AURIX TRICORE TC277D

Pre PIL Simuláciu bude využitý mikrokontrolér Aurix Tricore ApplicationKit TC277D, ktorý sa nachádza na nasledujúcom obrázku Obr. 1 a Obr.2.



Obr.1: Aurix Tricore TC 277D [28]



Obr.2: Aurix Tricore TC 277D [28]

## 3.2 KOMUNIKÁCIA MIKROKONTROLÉRA A PC

Nami používaný mikrokontrolér obsahuje množstvo periférii a viacero komunikačných možností.
Tie budú spomínané v nasledujúcich častiach tejto kapitoly.
Samotný mikrokontrolér je programovaný cez kábel, ktorý je pripojený do microUSB portu z USB portu počítača.
Ako udáva dokumentácia k mikrokontroléra, tak môže byť jeho prúdová spotreba väčšia než 400mA. Preto nemusí pripojenie do portu USB 2.0 postačovať.
Z toho dôvodu je výhodnejšie používať pripojenie do USB 3.0, ktoré je schopné napájať pripojené zariadenia až do 900mA. Rovnako výrobca nedoporučuje napájanie cez USB HUB.
V prípade, že by dostupný prúd nepostačoval je možné taktiež využiť externé napájanie.

### 3.2.1 Sériová komunikácia

Sériovú komunikáciu medzi mikrokontrolérom Aurix TriCore TC277D bolo potrebné zabezpečiť tak na strane mikrokontroléra, ako aj Matlabu.

Sériová komunikácia je jedna z najpoužívanejších typov komunikácii v embedded systémoch pre dátovú komunikáciu. Existujú dva základné druhy sériovej komunikácie.
Jedná sa o synchrónnu sériovú komunikáciu a asynchrónnu sériovú komunikáciu.
Rozdiel medzi synchrónnou sériovou komunikáciou a asynchrónnou sériovou komunikáciou je ten, že synchrónna sériová komunikácia používa pri prenose dát synchronizačný signál, ktorý je používaný pre synchronizáciu prenášaných dát.
Ako príklad pre synchrónnu sériovú komunikáciu je možné uviezť rozhranie SPI alebo protokol I2C.
Na druhú stranu asynchrónna sériová komunikácia, ako už z názvu vyplýva synchronizačný signál nepoužíva.
Asynchrónna sériová komunikácia používa v komunikácii Start a Stop bity, ktoré sú používané pre oddeľovanie dátových bajtov.
Ako príklad pre tento typ komunikácie je možné uviezť napríklad protokoly RS232 alebo aj RS485. [30]

Mikrokontrolér Aurix využíva pre sériovú komunikáciu modul ASCLIN.
Pomocou neho je vykonávaná asynchrónna sériová komunikácia s externými zariadeniami, ktoré v našom prípade predstavujú počítač s Matlabom.

Na komunikáciu používa signály pre odosielanie a prijímanie dát. [10]
Modul ASCLIN je možné využiť aj na iné typy komunikácii, nielen na tú sériovú.
Pre sériovú komunikáciu je využívaný konkrétne modul ASCLIN0, ktorý je možné s počítačom pre sériovú komunikáciu spojiť pomocou USB konektoru.
Pre komunikáciu po sériovej linke je potrebné mať v počítači nainštalovaný softvér DAS.
Po pripojení mikrokontroléra sa v počítači automaticky vytvorí virtuálny sériový COM port, ktorý bude možné využiť na už spomínanú sériovú komunikáciu. [22]

Sériovú linku charakterizuje viacero parametrov. Jedným z nich je takzvaný baudrate, ktorý predstavuje rýchlosť komunikácie, ktorá je vyjadrovaná v počte bitov za jednu sekundu.
Jednotlivé parametre je možné v uvádzanom mikrokontroléri softvérovo nastavovať.

Štandardnými základnými parametrami tohto typu komunikácie u popisovaného mikrokontroléra sú nasledujúce:

- Dátový rámec je možné nastaviť na 7, 8, 9 alebo aj 16 bitov.
- Dátová komunikácia môže obsahovať aj paritný bit.
- Je možné nastaviť jeden alebo dva stop bity.
- Ďalším voliteľným nastavením komunikácie je možnosť takzvaného „handshakingu“ pomocou RTS/CTS.
- Rýchlosť komunikácie je špecifikovaná pomocou baudrate, ktorú je možné taktiež voliteľne nastavovať. Pre našu komunikáciu sme ponechali prednastavenú rýchlosť 115200 bps.

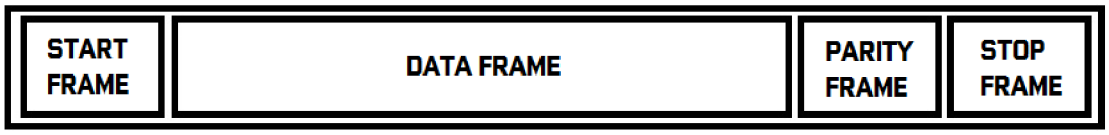
Pre komunikáciu po sériovej linke pomocou modulu ASCLIN0 je nutné využívať iLLD súbory dodávané výrobcom mikrokontroléra.
V nich sú pre komunikáciu uvedené základné nastavenia.
Pre správnu funkčnosť s našim typom mikrokontroléra bolo potrebné prekontrolovať pripojenie pinov modulu ASCLIN s USB konektorom a bolo nutné po danej kontrole zmeniť v zdrojovom kóde označenie pinov.
Tie sme pre správne nasmerovanie komunikácie zmenili na piny P14.1 pre príjem dát a P14.0 pre odosielanie dát.

Dáta sú pri tejto komunikácii odosielané a prijímané v podobe ASCII znakov.
Pre vytvorenie sériovej komunikácie bolo potrebné pripraviť funkcie pre otvorenie sériového portu, prijímanie dát a odosielanie dát.

Samotný zdrojový kód je uvedený v prílohe číslo 1.

#### 3.2.1.1 Dátový rámec sériovej komunikácie

Pri komunikácii po sériovej linke je potrebné okrem rýchlosti prenosu dát nastaviť aj ďalšie veľmi významné parametre určujúce prenos dát. Jedná sa o parametre, ktoré určujú dátový rámec, ktorým sa po sériovej linke prenášajú dáta.
Na nasledujúcom obrázku Obr. 3 je znázornený všeobecne používaný rámec sériovej komunikácie používaný pri prenose dát.




Obr.3: Ukážka dátového rámcu

Ako je znázornené na predchádzajúcom obrázku Obr. 3, tak rovnaký dátový rámec je použitý v nami používanej sériovej komunikácii.
Na začiatku prenášaných dát je takzvaný Start Frame, teda štartový rámec, ktorý určuje začiatok rámca.
V našom prípade je veľkosť štartovacieho rámca jeden bit, ktorý má hodnotu nula.
Bezprostredne za ním nasledujú dátové bity, ktoré sa odosielajú štýlom little endian. V nami využívanej sériovej komunikácii je nastavených presne osem dátových bitov.
Za požadovaným počtom dátových bitov nenasleduje paritný bit. Ten je ale možné nastaviť na párnu paritu, teda v prípade párneho počtu jednotiek v dátovom rámci by bola hodnota parity nula.
V mikrokontroléri je možné paritu taktiež nastaviť na nepárnu paritu.
Ďalej nasleduje rámec, ktorý zastavuje odosielanie aktuálneho dátového rámcu a to konkrétne takzvaný Stop Frame, ktorého veľkosť je taktiež, ako u štartovacieho bitu iba práve jeden bit. Pomocou tohto zastavovacieho bitu sa ukončí odoslanie jedného dátového rámca a po jeho spracovaní prijímaným zariadením môže začať odosielanie ďalšieho.
Jednotlivé prijímané a odosielané dáta sú ukladané v bufferu, ktorý je riešený ako FIFO. Veľkosť daného bufferu je 128 bajtov.

Nastavenia sériovej komunikácie vytvárané v mikrokontroléri Aurix TriCore TC277D je potrebné dodržať taktiež v nastaveniach komunikácie v Matlabe.
V prípade akejkoľvek zmeny by bolo potrebné tieto zmeny vykonať na oboch stranách. Ako bolo už vyššie spomenuté, tak rýchlosť komunikácie určuje takzvaný baudrate.

### 3.2.2 CAN

Jedná sa o ďalšiu možnosť využitia komunikácie s externými zariadeniami.
V samotnej dokumentácii je zbernica CAN nazývaná ako MultiCAN. [22]
Komunikácia pomocou zbernice CAN je využívaná najmä v automobilovom priemysle. Pre našu potrebu PIL simulácie bude ale nevyužitá, preto ju nebudeme naďalej opisovať.

### 3.2.3 LIN

Zbernica LIN poskytuje ďalšiu možnosť komunikácie pomocou mikrokontroléra.
Funkcionalitu LIN zbernice zabezpečuje modul ASCLIN1. [22]
Táto zbernica má podobné využitie, ako už hore uvádzaná zbernica CAN, najmä v automobilovom priemysle. Pre pripojenie s Matlabom pre vykonanie PIL simulácie je ale rovnako, ako CAN zbernica pre naše potreby nepoužiteľná, preto nebude taktiež ďalej v tejto práci popisovaná.

### 3.2.4 ETHERNET

Samozrejmosťou pre využívaný mikrokontrolér je aj možnosť pripojenia RJ45 konektoru pre komunikáciu pomocou Ethernet.
Zadanie tejto práce požaduje pre PIL simuláciu využitie najmä sériovej komunikácie. Pre PIL simuláciu by ale bolo samozrejme možné využiť aj pripojenie pomocou RJ45 konektoru, nakoľko ním mikrokontrolér a aj nami využívaný počítač disponuje.
Vo všeobecnosti by sa malo jednať o rýchlejšiu komunikáciu, ako so sériovou linkou.

## 3.3 DÁTOVÁ INTEGRITA

Dátová integrita alebo aj inak nazývaná signálová integrita predstavuje pojem, ktorý označuje správnosť odosielaných a prijímaných údajov a to nielen po nami využívanej asynchrónnej sériovej komunikácii.
Aj keď pri použití asynchrónnej sériovej komunikácie nie je použitý synchronizačný signál, ktorý by zabezpečoval správne načasovanie prenášaných údajov, tak je aj napriek tomu potrebné aby sa údaje po asynchrónnej sériovej komunikácii prenášali presne.

V prípade nesprávnej integrity dát je možné pri komunikácii úplne prísť o dáta alebo môže byť pozmenený aj ich obsah.
Aj napriek tomu, že sériová komunikácia, či už synchrónna alebo asynchrónna je digitálna komunikácia používajúca binárnu logiku, tak úrovne 1 a 0 ktoré predstavujú binárne hodnoty sú zastúpené napäťovými úrovňami, ktoré majú analógový priebeh. Tento priebeh je často skreslený, čo spôsobuje viacero faktorov.
Prenos údajov môže ovplyvniť elektromagnetické rušenie ale aj kvalita a dĺžka použitého káblu pre komunikáciu.
Z toho dôvodu je potrebné v zdrojovom kóde zabezpečujúcom prenos dát po sériovej linke vytvoriť taktiež systém pre kontrolu prípadných chýb. [30]
Existuje niekoľko spôsobov ako zlepšiť dátovú integritu pri prenose údajov, tie je ale potrebné zabezpečiť nielen na softvérovej, ale aj na hardvérovej úrovni. Spôsoby, ktorými je možné zabezpečiť dátovú integritu budú opisované v nasledujúcich častiach tejto kapitoly.

### 3.3.1 Kontrolný súčet – checksum

Pre účel zvýšenia integrity dát je možné využiť takzvaný CRC kontrolný súčet alebo použiť jednoduchšiu možnosť, a to napríklad vykonávať bitovú operáciu XOR na všetkých bajtoch v pakete. Vďaka jednej z týchto dvoch možností je možné zistiť, či bola prijatá správa poškodená. [30]
Kontrolný súčet je potrebné vykonávať na strane targetu, ako aj Matlabu, nakoľko obe strany prijímajú dáta. V nami použitom target systému Aurix TriCore TC277D, ako aj v Matlabe je pre výpočet kontrolného súčtu použitá práve metóda výpočtu checksum pomocou bitového XOR.

### 3.3.2 Handshake a Potvrdzovanie správ

Ak napríklad počas PIL simulácie Matlab vyšle nejakú správu, tak mikrokontrolér odošle cez vytvorený komunikačný kanál správu o tom, že správu úspešne prijal alebo v prípade nesprávneho checksum, ktorý by indikoval prijatie poškodenej správy vyšle signál o neprijatí správy a Matlab odošle správu znovu. Pri prípadnom ďalšom neúspechu bude komunikačný kanál uzavretý a prenos údajov rovnako ako aj samotná PIL simulácia skončí chybou.
Jedná sa o funkcionalitu potvrdzovania správ. Ďalšou potrebnou funkcionalitou pre zabezpečenie potrebnej úrovne integrity dát je zabezpečená pomocou takzvaného handshake. Jedná sa o signály, ktorými sa komunikujúce zariadenia informujú, či sú pripravené prijať dáta. [35]

### 3.3.3 Zmena baudrate

V prípade použitia sériovej komunikácie po linke RS232 je napríklad možné pri prenose dát rýchlosťou 9600bps použiť komunikačný kábel maximálnej dĺžky 15m. [31]
V prípade použitia sériovej komunikácie cez USB 3.0, ktoré bolo použité v našom prípade je dĺžka kábla obmedzená na 3 metre. [32]
Znehodnotenie kvality komunikácie môže byť spôsobené použitím nevhodnej nestienenej alebo aj poškodenej kabeláže alebo konektorov. [33]
Ako je vyššie uvedená teoretická hodnota rýchlosti prenosu dát 9600bps pri maximálnej vzdialenosti 15m v prípade použitia RS232.
Táto maximálna vzdialenosť môže byť z dôvodu nesprávnych pripojovacích komponentov znížená.
Ak ale nie je možné znížiť dĺžku použitého káblu pre prenos údajov, tak je možné znížiť prenosovú rýchlosť baudrate, nakoľko sa v takom prípade môžu pri vyššej prenosovej rýchlosti poškodiť jednotlivé dáta.
Maximálna možná rýchlosť baudrate môže byť taktiež ovplyvnená možnosťou mikrokontroléra spracovávať prijímané a odosielané dáta popri vykonávaní ďalších programových úloh. [30]

## 3.4 HW a SW KONFIGURÁCIA PRE AURIX

Pre plnohodnotné využívanie mikrokontroléra je potrebné okrem hardvérových nastavení, akými sú prepojenie jumperov na doske mikrokontroléra aj ďalšie, už komplikovanejšie nastavenia, ktoré budú opísané práve v tejto časti dokumentu diplomovej práce.

### 3.4.1 Hightec Free TriCore Entry Toolchain

Programovacie prostredie Hightecu využíva pre svoju funkciu Eclipse. Z tohto dôvodu je nutné mať na počítači nainštalovaný balík JAVA.
Pre správne nakonfigurovanie vývojového prostredia je potrebné vykonať v počítači zásah do premenných prostredia operačného systému Windows.
Do premennej s názvom PATH bolo v našom prípade nevyhnutné pridať nasledujúce cesty a oddeliť ich bodkočiarkou:

C:\HighTecX\toolchains\tricore\v4.9.3.0-infineon-1.0\bin;
C:\HighTecX\licensemanager;
C:\ProgramData\Oracle\Java\javapath;
C:\Program Files (x86)\Java\jdk1.8.0_151\bin;

Inštalačný priečinok tohto programu v operačnom systéme Windows je možné zmeniť na iný, ako je prednastavená možnosť, avšak je potrebné aby systémová cesta k danému priečinku neobsahovala medzery alebo diakritiku.

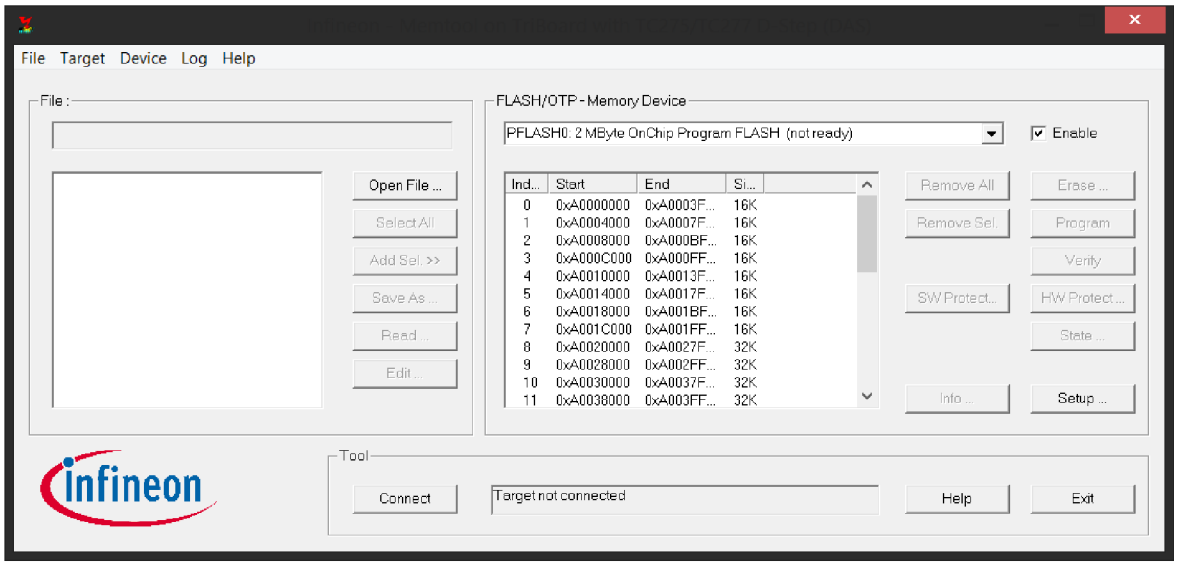
Užitočnú dokumentáciu je možné získať iba priamo z webových stránok výrobcu, kde je nevyhnutné sa registrovať.
Pomocou registrácie je možné získať ročnú skúšobnú licenciu na daný program. Registrácia je teda taktiež potrebná nielen pre stiahnutie inštalácie vývojového prostredia, ale aj pre získanie potrebných dokumentácii, pre ktoré je ale potrebné sa registrovať ešte zvlášť aj do ich informačného portálu MyICP pre získanie kompletnej dokumentácie, ako aj iLLD súborom a vzorovým zdrojovým kódom, pomocou ktorých je možné vytvárať projekty pre použitý mikrokontrolér.

### 3.4.2 BIFACES

Bifaces je vlastne aplikačný spôsob, ktorým je možné využívať iLLD ovládače a demoverzie kódov pre programovanie v Hightec.
Je možné ich získať po registrácii na webových stránkach výrobcu mikrokontroléra.
Okrem toho program Bifaces poskytuje jednotnú platformu pre kompletný softvérový balíček pre programovanie Aurix mikrokontrolérov.
Samotné demoverzie projektov iLLD sa skladajú zo súborov vygenerovaných makefilom, ale bez akejkoľvek kontroly kompilátora alebo linkeru.
Úlohou Bifaces je práve prevedenie týchto súborov do projektu v HighTec, čím je ich následne možné používať ako klasické .c/.h súbory. [21]
Uvedené iLLD ovládače obsahujú pred vytvorené časti kódu spolu s Demo verziami ich použití v podstate pre všetky periférie mikrokontrolérov Aurix TriCore. Pre ich použitie je ale potrebná správna konfigurácia a poprepojovanie.
Z uvedených iLLD driverov budú použité pre vytvorenie sériovej komunikácie časti modulu ASCLIN. Ten je používaný pre vytváranie asynchrónnej sériovej komunikácie v 8 bitovom móde, ako naznačuje používateľská dokumentácia k iLLD, ktorú je možné dostať po schválenej registrácii na stránkach Infineon.

### 3.4.3 Infineon Memtool

Jedná sa o program, pomocou ktorého sa ukladá skompilovaný program do pamäti mikrokontroléra. Tento program je pre nahranie programovaného kódu do mikrokontroléra nutné spúšťat zvlášť, čo z časti sťažuje PIL simuláciu nakoľko by nahratie kódu do mikrokontroléra malo prebiehať automaticky.
Zakúpená plnohodnotná verzia Infineon Memtoolu umožňuje využívanie príkazmi z príkazovej riadky cmd.exe v operačnom systéme Windows, avšak počas tvorby tejto práce sme plnou verziou nedisponovali. V opačnom prípade by bolo možné nahrávanie kódu do mikrokontroléra bez akýchkoľvek problémov zautomatizovať. V našom prípade to ale nie je možné, nakoľko nie je iná cesta ako vložiť vytvorený kód do mikrokontroléra a z tohto dôvodu bolo potrebné vytvoriť v PIL simulácii časové oneskorenie, ktorým sa čakalo, kým sa podarí nahrať kód do mikrokontroléra Aurix TriCore TC277D. Na nasledujúcom obrázku Obr. 4 bude znázornené okno opisovaného softvéru Infineon Memtool.

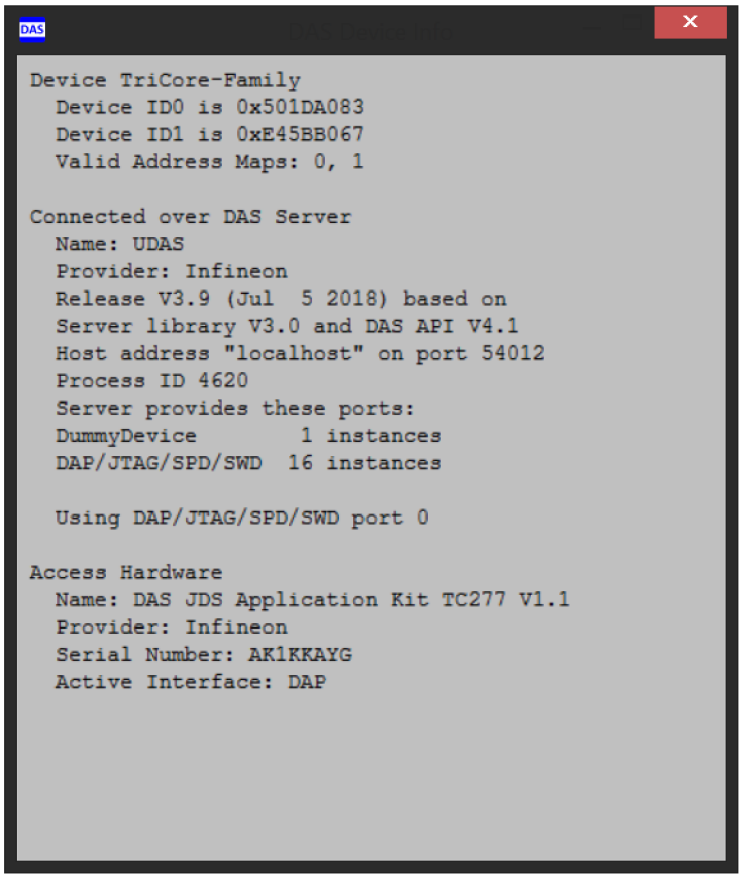


Obr.4: Vzhľad programu Infineon Memtool

Okno programu Infineon Memtool znázorňuje predošlý obrázok Obr.4.
Pre naprogramovanie kódu do mikrokontroléra je najskôr potrebné sa s ním spojiť pomocou kliknutia na tlačidlo „Connect“ a v prípade úspešného spojenia stlačiť tlačidlo „Program“.
Pomocou Memtoolu je taktiež možné vykonať test pamäte mikrokontroléra, ako aj jeho celkovú funkčnosť pomocou EOL Testu, ktorý je bližšie popísaný v dokumentácii nami použitého mikrokontroléra. Ako už bolo spomenuté, tak uvedená dokumentácia je prístupná až po registrácii na stránkach Infineonu.

### 3.4.4 DAS – Device Access Server

Jedná sa o program používaný ako prístupový server pre mikrokontrolér Aurix. Je možné ho využiť aj ako abstrakciu fyzického pripojenia mikrokontroléra k počítaču, alebo aj pre kontrolu správneho pripojenia mikrokontroléra ku počítaču.
Pomocou DAS je možné pripojiť mikrokontroléri, ktoré môžu byť aj iného typu, ako len TriCore, avšak musí sa jednať o mikrokontrolér od spoločnosti Infineon.
O pripojených mikrokontroléroch je pomocou tohto programu možné zistiť viaceré informácie týkajúce sa pripojenia, pamäte mikrokontoléra alebo vykonávať analýzu pre debugovanie.
Na nasledujúcom obrázku Obr.5 sú znázornené informácie, ktoré je možné získať pomocou jednej z niekoľkých súčastí DAS programu.



Obr.5: Okno programu DAS Device Info s pripojeným TriCore TC 277D

Aby bolo možné využívať tento program je potrebné, aby premenné prostredia Windowsu obsahovali premennú s názvom DAS_HOME a táto premenná musí obsahovať systémovú cestu, kde je DAS nainštalovaný.

### 3.4.5 Dávkové súbory

Dávkové súbory sú textové súbory, ktoré ale majú príponu .bat a nie .txt, ako je tomu v operačnom systéme Windows zvykom.
Tieto dávkové súbory môžu obsahovať riadkové príkazy príkazového riadku, DOS operačného systému alebo aj iné, ktoré sa po spustení súboru vykonajú. [9]
Pomocou jedného dávkového súboru s príponou .bat je možné v operačnom systéme Windows napríklad spustiť niekoľko programov naraz, alebo vykonať potrebnú konfiguráciu.
Nami použitý dávkový súbor pre spúšťanie správne nakonfigurovaného vývojového prostredia pre programovanie mikrokontroléra Aurix TriCore TC277D od výrobcu Infineon je možné nájsť na nasledujúcich riadkoch:

```
@SET BINUTILS_PATH=C:\Tools\BifacesTools\bin
@SET MAKECMD_WITH_APP=C:\HighTecX\ide\htc-ide-v2.2.4\htc-ide.exe

::-----------------------------------------------------
@SET PATH=%BINUTILS_PATH%;%PATH%
@START %MAKECMD_WITH_APP%
@EXIT
```

### 3.4.6 Debug konfigurácia

Vývojové prostredie Hightec využíva Universal Debug Engine, u ktorého je nevyhnutné vytvoriť pre „Debug“ kódu samostatnú konfiguráciu pre každý projekt samostatne.
Túto konfiguráciu je možné vytvoriť v okne Debug Configurations programu Hightec.
Konfigurácia obsahuje názov projektu, ako aj súbor v časti „C/C++ application“, ktorý je potrebné vybrať ručne. Príklad systémovej cesty súboru, ktorý je potrebné vybrať do danej časti konfigurácie v prípade použitia Bifaces je v projekte na našom počítači nasledujúca:

C:\Users\Michael\htc-workspace\BaseFramework_TC27D\2_Out\Tricore_Gnuc\BaseFramework_TC27D_Tc.elf

Uvedená cesta k danému súboru bude na inom počítači alebo v projekte s iným názvom pochopiteľne odlišná.
Ďalej je nutné definovať súbor UDE workspace, ktorý je typu „.wsx“ a ďalej je potrebné vybrať v časti UDE Target Configuration konkrétny typ mikrokontroléra. V našom prípade sa jednalo o súbor „AppKit_TC277D.cfg“.
Po uložení vytvorenej konfigurácie je možné spustiť „Debug“ kódu.

#### 3.4.6.1 CONFIG TRICORE GNUC

Bez nastavenia správnej cesty kompilátoru však „Debug“ kódu pri použití Bifaces vyhlási chybu a žiaden kód sa neskompiluje.
V našom prípade bol využitý kompilátor GNUC spolu s Bifaces.
Cesta k súboru, ktorý je potrebné editovať bola v našom prípade:

C:\Users\Michael\htc-workspace\BaseFramework_TC27D\1_ToolEnv\0_Build\1_Config\Config_Tricore_Gnuc\Config_Gnuc.mk

Ako z uvedenej cesty vyplíva, jednalo sa o makefile.
V danom makefile bolo potrebné nastaviť správnu systémovú cestu v nasledujúcej premennej:

B_GNUC_TRICORE_PATH

Do hore uvedenej premennej sme museli pre správnu funkciu kompilátoru vložiť Nasledujúcu systémovú cestu:

C:\HighTecX\toolchains\tricore\v4.9.3.0-infineon-1.0

Až po vykonaní tohto kroku bolo možné vykonať kompiláciu a nahranie kódu do mikrokontroléra Aurix TriCore TC277D.

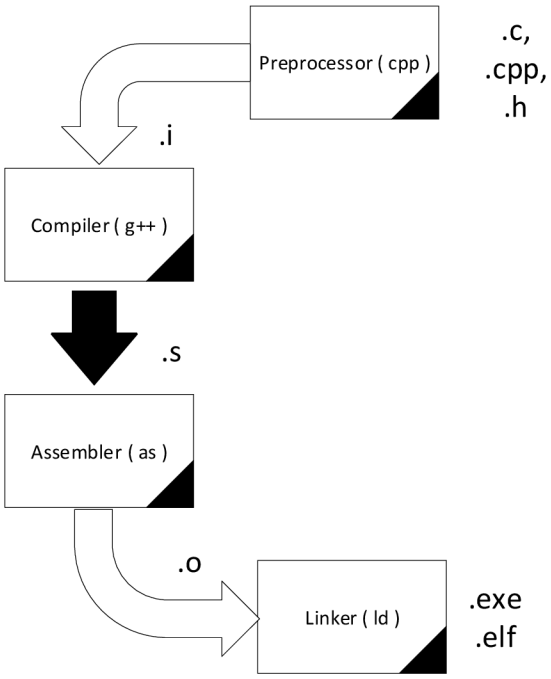
## 3.5 PROCES KOMPILÁCIE

Jedná sa o ďalšiu podstatnú časť nastavenia procesu PIL simulácie.
Proces kompilácie kódu zahŕňa niekoľko krokov, ktoré budú opísané práve v tejto kapitole.
Pre potreby PIL simulácie prebiehajúcej v prostredí Matlab Simulink bude potrebné vykonať nastavenia jednotlivých častí procesu kompilácie kódu, ktorá je taktiež znázornená na nasledujúcom obrázku Obr.2.
Dané nastavenia bude nutné vytvoriť v skripte Matlabu.

Prvou časťou je spracovanie zdrojových súborov, ktoré majú nasledovné súborové prípony:
„ .c “, „ .cpp “, „ .h “

Tie sú spracovávané preprocesorom, ktorý je označovaný ako cpp, v našom prípade s použitím GNU kompilátoru HighTec sa jedná o spustiteľný súbor „ tricore-cpp “, ku ktorému bude potrebné nastaviť cestu a parametre nevyhnutné pre jeho použitie. Predspracovaním zdrojových súborov sa zahrnú ďalšie hlavičkové súbory a rozbalia prípadné makro súbory a z celého procesu vzniknú súbory s príponou „ .i “, ktoré sú vstupom pre kompilátor. [25]
Pre zahrnutie procesu krížovej kompilácie kódu bolo potrebné zahrnúť spustiteľný súbor „ tricore-gcc “, ktorý predstavuje samotný kompilátor pre mikrokontrolér Aurix TriCore.

Výstupom kompilátoru je takzvaný assembler kód s príponou „ .s “, ktorý je vstupom pre Assembler „ tricore-as “, z ktorého vychádza strojový kód „ .o “. Daný strojový kód je prevzatý Linkerom „ tricore-ld “. Okrem strojového kódu je možné Linkerom spracovať aj statické knižnice s príponami „ .lib “ a „ .a “, tie však v našom prípade neboli použité, preto nie sú ani znázornené na obrázku Obr.6. Výstupom z Linkeru je už nami požadovaný spustiteľný súbor, ktorý môže byť buď „ .exe “ alebo „ .elf “, ktorý bude nahrávaný do pamäte mikrokontroléra za pomoci programu Infineon MemTool alebo z prostredia HighTec IDE.




Obr. 6: Znázornenie procesu kompilácie [25]

Ako z predchádzajúceho obrázku Obr. 6 vyplýva je potrebné vykonať niekoľko úkonov pre spracovanie zdrojového kódu k získaniu potrebného „ .elf “ súboru, ktorý bude možné prekonvertovať na „.hex“ súbor a uložiť do pamäte mikrokontroléra.
Tieto jednotlivé úkony sa ale nedejú samé od seba.
Na ich postupné vykonanie je potrebné vytvoriť ďalší súbor, ktorý jednotlivé akcie bude postupne spúšťať a obsluhovať. Pre túto úlohu je určený makefile súbor alebo ďalšia možnosť, ktorá je umožnená Matlabom, ktorou je toolchain.

### 3.5.1 Nastavenia pre GNU Compiler Collection

V prípade krížovej kompilácie, ktorú bude nutné použiť pre vykonávanie PIL simulácie bude potrebné použiť súčasti vývojového prostredia HighTec IDE z prostredia Matlabu. Ako bolo avizované už v predchádzajúcej časti kapitoly, bude sa jednať konkrétne o spustiteľné súbory, ktoré budú predstavovať základné programové prostriedky pre zostavovanie kódu.
Bude sa jednať o kompilátor, linker, archiver a prekladač z „ .elf “ súboru do „ .hex “ súboru.

#### 3.5.1.1 Nastavenia kontrolujúce generovanie kódu

Pre nastavenia generovania kódu, ktorý je možné nahrať do mikrokontroléra pomocou GNU kompilátoru, ktorý je využívaný vývojovým prostredím HighTec IDE je potrebné najskôr pochopiť jednotlivé nastavenia kompilátoru.
Uvedené nastavenia bolo potrebné upraviť, aby bolo do projektu v HighTec IDE možné zahrnúť zdrojový kód z Matlabu.
Na druhej strane v Matlabe bolo potrebné vytvoriť a upraviť taktiež nastavenia pre kompilátor aby bolo možné vygenerovať správny zdrojový kód, ktorý bude možné spracovať GNU kompilátor používaný v HighTec IDE.
Význam jednotlivých nastavení je možné sa dočítať z návodu, ktorý je dostupný po inštalácii vývojového prostredia HighTec IDE [41] alebo z návodu pre GNU kompilátor [42].
Kompletné konfigurácie nastavení sa nachádzajú v prílohe v jednotlivých súboroch, ktoré sa používajú pre nastavenia kompilátoru.
Z dôvodu priveľkého množstva možných nastavení, budú v tejto kapitole opísané práve tie, ktoré bolo nutné použiť pri tvorbe tejto diplomovej práce a ich zdrojom popisu sú práve dokumentácie [41] a [42].
Uvedené nastavenia budú označované z dôvodu prehľadnosti textu v úvodzovkách.

Pomocou nastavenia „-I“ sa pridáva do procesu kompilácie zložka hlavičkových súborov, ktoré majú byť zahrnuté do procesu kompilácie kódu.

Nastavením „-g“ kompilátor vytvára debugovacie informácie o prebiehajúcich procesoch

Pomocou nastavenia „-v“ je možné zobraziť číslo verzie súboru „gcov“.Súbor „gcov“ je súbor používaný pre testovanie kódového pokrytiavo vytváranom programe.

Použitím nastavenia „-mtc161“ sa vykoná časť nastavenia určujúca požiadavka pre uloženie programu do pamäte ROM. Jedná sa o označenie pamäťového súboru

tc161, ktorý je využívaný nami používaným typom mikrokontroléra. Definovaním takéhoto súboru sú adresy v pamäti RAM pre kód a dáta adaptované do pamäti určenej práve uvedeným súborom tc161.

V prípade použitia nastavenia „-O0“ kompilátor pridelí pamäť zásobníku pre premenné, u ktorých nie je definované uloženie v registroch. V prípade, že má premenná špecifikované úložisko v registroch, tak do zásobníku uložená nebude.

Voľba nastavenia „-fno-common“ je používaná pre globálne premenné, u ktorých nebola vykonaná inicializácia a nemajú pridelené úložisko. Vďaka tomuto nastaveniu je umožnené kompilátoru prideliť pamäťový priestor aj neinicializovaným globálnym premenným, vďaka čomu dokáže linker vyriešiť definície daných premenných.

Použitie „-fstrict-volatile-bitfields“ je potrebné v prípade, keď je vyžadovaný presne stanovený prístup k volatilným bitovým poliam na jediný prístup k celej šírke poľa.

Pomocou nastavení „-ffunction-sections“ a „-fdata-sections“ je možné umiestniť každú funkciu do vlastnej sekcie vo výstupnom súbore, kde táto sekcia bude niesť názov uvedenej funkcie. Tieto dve nastavenia je nutné použiť pre linker generujúci „.elf“ súbor, ktorý vykonáva optimalizácie pre zlepšenie lokality referencii v inštrukčnom priestore. V prípade použitia týchto nastavení je ale nevýhoda, že linker vytvorí väčší objektový súbor a takto vygenerované spustiteľné súbory sú tiež pomalšie.

Nastavenie „-Wall“ povoluje zobrazenie všetkých varovaní, ktoré nastanú pri procese kompilácie.

Pomocou nastavenia „-std=c99“ je možné definovať normu programovacieho jazyku, ktorý sa bude používať.

Nastavenie „-D“ definuje makro pre preprocesor v spojení s názvom makra, ako je tomu uvedené na nasledujúcich príkladoch:

```
-DONESTEPFCN=1
-DRTIOSTREAM_RX_BUFFER_BYTE_SIZE=128
-DRTIOSTREAM_TX_BUFFER_BYTE_SIZE=128
-DMEM_UNIT_BYTES=1
-DCA_CHECK_DAZ_ENABLED=1
-DMODEL=pilmodelskuska
-DNUMST=3
-DCA_CHECK_LONG_LONG_ENABLED=1
-DCA_CHECK_FLOATING_POINT_ENABLED=1
-DTID01EQ=1
-DMemUnit_T=uint8_T
-DCODER_ASSUMPTIONS_ENABLED=1
```

Hore uvedené makrá boli pridané do nastavení v súbore „Config_Gnuc.mk“.
V našom prípade nebol pre spustenie linkeru volaný priamo spustiteľný súbor linkeru ale kompilátor, ktorý si sám spúšťal linker. Vďaka tomu bolo potrebné príkazy pre linker používať vždy s nastavením „-W1“.
Nastavenie „-W1“ je vždy použité s iným nastavením, ktoré je v spojení s „-W1“ možné uviezť ako nastavenie pre linker. Napríklad v nastavení linkeru v prostredí HighTec IDE je použité nastavenie“-Wl,--gc-sections“, ktoré je použité pre určovanie nepoužitých vstupných sekcii.
Pomocou nastavenia „-nostartfiles“ je určené, že linker nebude používať žiadne štartovacie súbory, ktoré sú pridávané ako ďalšie linker skripty.
Ďalším nastavením pre linker je „-Wl,-n“, ktoré spôsobí, že nebude vytvorený „gcov“ výstupný súbor.

Ako je napríklad použité pre nastavenia linkeru v prostredí Matlabu:
„-Wl -T $(LINKER_SKRIPT)“, tak použitím uvedeného spojenia príkazov sa pomocou príkazu -T prevedie skript na linker skript. Daný skript je v uvedenom príklade označený ako premenná s názvom LINKER_SKRIPT. Premenná sa označuje pomocou znaku $. Obdobne použitím príkazov „-Wl,-Map=$(MAP_FILE)“ sa do linkeru prevedie mapovací súbor označený premennou MAP_FILE. Taktiež pomocou príkazu „-Wl,--extmap=a“ sa vygeneruje rozšírený mapovací súbor s prídavnými informáciami. Uvedená séria príkazov musí byť použitá spolu s predchádzajúcou časťou, v ktorej sa definoval mapovací súbor. Uvedené prídavné informácie v prípade použitia označenia „a“ znamená, že budú obsahovať údaje o hlavičkových súboroch, ich verziách a dátumu tvorby a zoznam premenných zoradených podľa adries. Pomocou príkazu „-o“ je určený výstupný súbor, ktorý má byť výsledkom linkovania.

### 3.5.2 Makefile

Samotné generovanie kódu pre PIL simuláciu mikrokontroléra Aurix TriCore TC277D je potrebné spracovať pomocou takzvaného makefile súboru, ktorý bude pri generovaní kódu v Matlabe slúžiť pre správnu konfiguráciu krížovej kompilácie kódu. Taktiež okrem využitia makefile súboru existuje ešte možnosť definovania takzvaného vlastného toolchainu.
Matlab pre potreby PIL simulácie vo svojich vzorových riešeniach ponúka template makefile súbory, ktoré sú už vytvorené a nachystané pre použitie.
Sú však vytvorené pre konkrétne druhy hardvéru a ich úprava pre náš mikrokontrolér by bola príliš komplikovaná.

### 3.5.3 Toolchain

Toolchain je spôsob, ktorým je možné v Matlabe pomerne jednoducho zabezpečiť krížovú kompiláciu kódu.
V podstate sa jedná o zjednodušenú tvorbu makefile súboru avšak s názorným postupným nastavením jednotlivých komponentov.
Samotný makefile sa automaticky vygeneruje po registrácii Toolchainu. [26]
Tvorba Toolchain prebieha v skripte Matlabu a obsahuje všetky potrebné informácie o archivátore, preprocesore, kompilátore, assemblery a linkeru spolu s informáciami o konkrétnom hardvéri, pre ktorý má byť použitý.
Vytvorený toolchain, ako aj súbory potrebné pre registráciu Toolchainu sa nachádzajú v prílohe.

# 4 PIL SIMULÁCIA MEDZI PC A AURIX

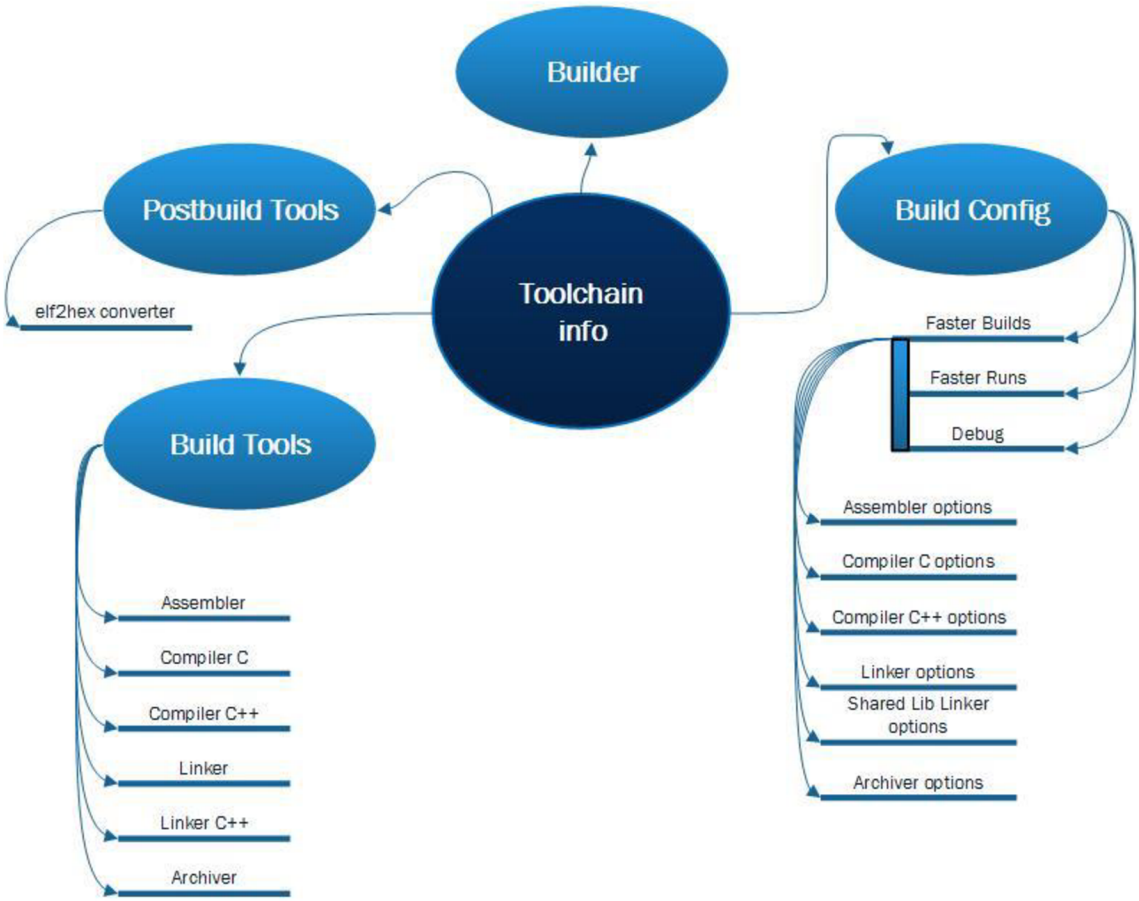
Pre spojenie mikrokontroléra s Matlabom cez USB port bolo potrebné vytvoriť sériovú komunikáciu pre samotný mikrokontrolér, ako aj pre samotný Matlab, aby medzi nimi mohla prebiehať komunikácia s potrebnými údajmi.

V našom prípade sú nastavenia automatického generovania kódu vykonávané práve pomocou skriptov z Matlabu.
Všetky súbory potrebné pre PIL simuláciu sú rozdelené do jednotlivých zložiek. Jedná sa konkrétne o súbory „rtwTargetInfo.m“ pre registráciu Toolchain a „AurixTriCore_tc.m“ pre nastavenie kompilátoru a ďalších nastavení potrebné pre správne generovanie kódu krížovou kompiláciou a „getTargetConfiguration .m“ pre nastavenie cesty ku HighTec kompilátoru spolu s nastavením sériovej komunikácie a „sl_customiozation.m“ pre registráciu všetkých nastavení do Simulinku. Tento skript bude opísaný v nasledujúcich častiach o prispôsobení Build procesu.

## 4.1 ŠTRUKTÚRA BuildTool Class

Na nasledujúcom obrázku Obr. 7 je znázornená štruktúra jednotlivých použitých nástrojov pre proces zostavovania kódu.
Jeho základom je objekt Toolchain info označovaný v kóde ako „tc“. [37]
Jedná sa o štruktúru, ktorú je nutné dodržať, pretože pri spustení Build procesu je uvedená štruktúra v presnom znení vyžadovaná.

Daná štruktúra poskytuje viacero možností, ktoré plne postačujú pre správne nastavenie jednotlivých súčastí procesu zostavovania kódu.




Obr. 7: Štruktúra BuildTool class použitá pre PIL simuláciu [37]

### 4.1.1 Popis jednotlivých konfiguračných skriptov v Matlabe

Pre prvotné nastavenie vyhľadávacích ciest Matlabu slúži skript „startup.m“ nachádzajúci sa v zložke „AurixTricorePIL“.
Vo vnútri tejto zložky sa nachádza zložka s názvom „SimulinkDevEnv“, ktorá obsahuje všetky potrebné konfiguračné skripty a funkcie pre správne vykonávanie PIL simulácie v prostredí Matlab Simulink v spolupráci s mikrokontrolérom Aurix TriCore TC277D. V tejto zložke sa nachádza už spomínaný „sl_customization.m“ a rovnako aj skript „getTargetConfiguration.m“.
Ďalší opis skriptov a funkcii v jednotlivých zložkách bude opísaný pre každú zložku jednotlivo.

### 4.1.2 Súbory v zložke +AurixTriCorePIL

Skripty a funkcie v tejto zložke zabezpečujú ovládanie PIL simulácie v prostredí Matlab-Simulink.

#### 4.1.2.1 ConnectivityConfig

V popisovanom „ .m “ súbore sa vytvára aplikačné rozhranie pre sériovú komunikáciu, ktorá spája náš mikrokontrolér s Matlabom v počítači za pomoci predvytvorených knižníc „rtiostream“ a „rtiostreamserial“.

#### 4.1.2.2 Launcher

V „.m“ súbore Launcher sa nachádzajú funkcie pre stiahnutie súborov PIL simulácie do mikrokontroléra a rovnako aj spustenie a zastavenie samotnej simulácie. Taktiež obsahuje funkciu pre prekopírovanie potrebných súborov a pozastavenie simulácie kvôli naprogramovaniu mikrokontroléra.

#### 4.1.2.3 MakefileBuilder

V tomto súbore sa vykonáva nastavenie procesu zostavovania kódu za použitia súboru TargetApplicationFramework, ktorý bude opísaný na nasledujúcich riadkoch.

#### 4.1.2.4 TargetApplicationFramework

Vo vnútri tohto súboru je za pomoci rovnomennej funkcie „TargetApplicationFramework“ nastavená cesta k takzvanému main súboru, aby bolo možné vykonať proces zostavovania kódu určeného pre hardvérovú platformu nami použitého mikrokontroléra.

### 4.1.3 Súbory v zložke AurixTriCore_tc_setup

Táto zložka obsahuje súbory potrebné pre správne nastavenie a registráciu toolchain.

#### 4.1.3.1 update_mat_file

V uvedenom skripte je volaná funkcia „AurixTriCore_tc()“ pre nastavenie toolchain a dané nastavenia toolchain sa uložena do súboru „AurixTriCore_tc.mat“, z ktorého sa budú načítavať nastavené parametre.

#### 4.1.3.2 rtwTargetInfo

Tento skript obsahuje 2 funkcie pre vytvorenie a registráciu toolchain pre mikrokontrolér Aurix TriCore TC277D pomocou dát získaných z funkcie „AurixTriCore_tc()“ volanej v skripte „update_mat_file“.

#### 4.1.3.3 AurixTriCore_tc

Vo funkcii AurixTriCore_tc sa vykonávajú všetky potrebné nastavenia toolchain pre krížovú kompiláciu kódu pomocou nástrojov používaných pre zostavovanie kódu pre mikrokontrolér Aurix TriCore TC277D.
Sú tu vykonávané nastavenia pre kompilátor, linker a archiver.

## 4.2 PRISPÔSOBENIE „Build“ PROCESU V MATLABE

Prispôsobenie kompilácie kódu v Matlabe alebo Simulinku je možné pomocou súboru „sl_customization.m“.
Jedná sa o súbor, ktorým je možné nastaviť rozhranie procesu kompilácie kódu v Simulinku, avšak nastavenie je nutné z Matlabu.
Daný súbor musí obsahovať volanie funkcie sl_customization, samotný súbor musí byť taktiež obsiahnutý vo vyhľadávacej ceste Matlabu.
Tento súbor je zo Simulinku načítaný pri spustení samotného Simulinku.
V prípade zmeny v danom súbore je potrebné vypnúť a zapnúť Simulink, aby dané zmeny opätovne načítal. Taktiež je ale možné využiť príkazy z Matlabu. [12]
Jedná sa o prikaz: „sl_refresh_customizations“. Tento prikaz spustí všetky „sl_customization“ súbory v MATLAB ceste, ako aj v aktuálnej zložke.
Je ho potrebné spustiť pri akejkoľvek zmene „sl_customization súboru“, inak sa neprevedú požadované zmeny, nakoľko by sa tento súbor znovu načítal až pri opätovnom spustení Simulinku, ako už bolo vyššie spomenuté. [11]

Problémom však je, že tento súbor môže byť aktívny iba jeden. Pomocou príkazu: „which -all sl_customization.m“ je možné zobraziť všetky „sl_customization“ súbory, s ktorými môže Matlab pracovať.
Ak nastane pokus o konfiguráciu viacerých „sl_customization“ súborov, tak ostatné budú označené za „Shadowed“ a nebude umožnené ich použitie.
Aby sme mali nastavený práve nami požadovaný súbor, je potrebné vyhľadať všetky ostatné a odstrániť ich z vyhľadávacej cesty Matlabu pomocou príkazu „rmpath“.
Taktiež je možné v krajnom prípade zvoliť možnosť, ktorou dané konfliktné súbory úplne odstránime pomocou použitia príkazu „delete('sl_customization.m')“.
Pre opätovný reštart nastavenia bude taktiež potrebné spustiť príkaz: „sl_refresh_customizations“.

Výsledná časť pre nastavenie mikrokontroléra v tomto súbore pre náš prípad vyzeral nasledovne:

```
config = rtw.connectivity.ConfigRegistry;
config.ConfigName = 'MyAurixPILExampleTC277D';
config.ConfigClass = 'AurixTriCorePIL.ConnectivityConfig';

config.SystemTargetFile = {'ert.tlc'};
config.Toolchain =  {'AurixTriCore | gmake makefile (64-bit
Windows)'};
config.TargetHWDeviceType = {'Infineon->TriCore'};
```

Pomocou vlastnej definície Toolchain nebolo potrebné vytvárať makefile.
Všetky súbory, ktoré nastavujú PIL simuláciu sa nachádzajú v prílohe.
Pre overenie správnosti vytvoreného Toolchain je možné v konfiguračných parametroch modelu Simulinku v záložke Code Generation spustiť proces validácie Toolchain pre overenie správnosti nastavení. Výsledok validačného procesu sa nachádza v prílohe. Následne je potrebné pre overenie spustiť SIL simuláciu.

### 4.2.1 Programové utility

Pre zjednodušené používanie a nastavovanie PIL simulácie bolo vytvorených niekoľko programových pomôcok v Matlabe, ktoré taktiež zabezpečia jednoduchšiu konfiguráciu v prípade použitia na inom počítači.

Ako prvý bude v krátkosti spomenutý Matlab skript vytvorený pre testovanie funkčnosti sériovej komunikácie s názvom „communicationTest.m“, čo vyplýva už z názvu samotného Matlab skriptu.
Tento skript a jeho použitie bude bližšie opísané v nasledujúcej kapitole o testovaní funkčnosti.
Daný skript obsahuje nastavenie a spustenie Matlab funkcie rtiostreamtest a za komentovanú časť kódu, ktorá slúžila v prvotných fázach testovania aktuálne používanej sériovej komunikácie.

Ďalším Matlab skriptom, ktorý bol vytvorený pre zjednodušenie používania PIL simulácie je skript s názvom „copyMatlabFilesXIL.m“.
Tento skript slúži na prekopírovanie všetkých potrebných obslužných súborov potrebných pre chod PIL alebo SIL simulácie.
Uvedený skript je postačujúce spustiť iba raz pre každý projekt, nakoľko kopíruje súbory, ktoré sa pri akejkoľvek zmene simulačnej schémy alebo konfigurácie PIL simulácie nijako nemenia.

Kopírované súbory pomocou opisovaného skriptu s názvom „copyMatlabFilesXIL.m“ sa všetky nachádzajú v inštalačnej zložke Matlabu. Bez týchto súborov nie je možné SIL a ani PIL simuláciu spustiť.
Ako bude aj ďalej spomenuté, tieto súbory, ktoré uvedený skript nakopíruje do zložky projektu pre mikrokontrolér Aurix TriCore TC277D nie sú súčasťou prílohy uvedenej diplomovej práce, nakoľko autor tejto práce uvedené súbory nevytvoril a ani ich nebolo potrebné nijako editovať.

Nasledujúcou takzvanou programovou utilitou, ktorá bola vytvorená v rámci uľahčenia obsluhy PIL simulácie je ďalší Matlab „m“ súbor, ktorý nesie názov „copyFilesDemoiLLD.m“.
V tomto prípade sa už ale nejedná o skript ale o funkciu Matlabu.
Tá je spúšťaná automaticky pri každej PIL simulácii bezprostredne po vygenerovaní zdrojových súborov z modelu v Simulinku. Uvedená funkcia prekopíruje všetky potrebné zdrojové súbory rovnako, ako aj predošlý Matlab skript do projektovej zložky pre používaný mikrokontrolér.
Daná funkcia je spúšťaná vo vnútri ďalšej funkcie, ktorá nesie názov „postGenFunc.m“.
Účelom tejto funkcie je sprehľadnenie kódu, podľa ktorého sa obsluhuje PIL simulácia. V danej funkcii je taktiež veľmi potrebná pauza slúžiaca na nahranie kódu do mikrokontroléra.

Taktiež ako aj v predošlej situácii v prípade použitia na inom počítači je potrebné pred prvým použitím časť súboru editovať. Jedná sa o časť súboru, v ktorom je určené, kde sa uvedené zdrojové súbory nachádzajú a kam sa majú prekopírovať.
Nakoľko je vysoko pravdepodobné, že uvedené zložky budú v každom počítači na inom mieste je potrebné pred prvým použitím tieto cesty prekontrolovať.
U ktorých riadkov bude potrebné vykonať prípadnú úpravu súborových ciest bude opísané v návode na použitie, ktorý sa bude nachádzať na konci tejto diplomovej práce.
Uvedené riadky budú taktiež v konkrétnych častiach zvýraznené pomocou komentárov. Dané súborové cesty sú vstupom oboch funkcii pre kopírovanie súborov.

## 4.3 TESTOVANIE FUNKČNOSTI

Pre testovanie funkčnosti vytvorenej sériovej komunikácie za pomoci iLLD driverov a dostupných komunikačných súborov Matlabu bolo pre zabezpečenie správneho použitia komunikácie pre PIL simuláciu vytváranú komunikáciu najskôr otestovať.
V hlavnom zdrojovom súbore Cpu0_main v mikrokontroléri Aurix sú predpripravené tri makrá, z ktorých sú dve zakomentované.
Podľa ich popisu v komentári je jasné, k čomu slúžia. Ďalšie vysvetlenie bude uvedené nižšie.

### 4.3.1 Test Recv-Send

Jedná sa o makro s názvom test_RecvSend. Jeho odkomentovaním sa v zdrojovom kóde mikrokontroléra povolia častí kódu určené pre jednoduchú komunikáciu, ktorá obsahuje cyklické prijímanie a odosielanie dát po sériovej linke.
Pre prijímanie a odosielanie dát po sériovej linke na strane počítača je možné použiť v Matlabe predpripravený skript s názvom „ communicationTest.m “, ktorý obsahuje taktiež predpripravený kód pre prijímanie a odosielanie dát.
V danom skripte sa nachádza aj ďalší test, ktorý bude popísané v nasledujúcej časti kapitoly.
Nakoľko je možné používať súčasne iba jeden komunikačný kanál, tak je potrebné vykonávať iba jeden test.
V tomto prípade ale Matlab nie je nutnosťou. Pre uvedené testovanie je možné použiť napríklad aj program RealTerm alebo akýkoľvek program, ktorý je schopný otvoriť komunikačný kanál a spracovávať dáta po sériovej linke.
V prípade využitia Matlab skriptu „ communicationTest.m “ je okrem zakomentovania časti kódu pre rtiostreamtest a odkomentovania predtým zakomentovanej časti potrebné taktiež overiť pripojenie mikrokontroléra na správny COM port.
Nami použitý mikrokontrolér sa pripájal v počítači s operačným systémom Windows 8.1 na COM port 4.
Uvedený fakt bolo možné zistiť buď za použitia softvéru RealTerm, ktorý zobrazil aktuálne pripojené zariadenie na COM porte, ktorý predstavuje v počítači virtuálny sériový port.
Na ktorý port sa mikrokontrolér v počítači pripojil je taktiež možné zistiť bez akéhokoľvek doinštalovaného softvéru.

Túto informáciu je možné zistiť taktiež pomocou už predinštalovaného Správcu zariadenia Windows, ktorý je možné poľahky nájsť pod názvom „devmgmt.exe“.

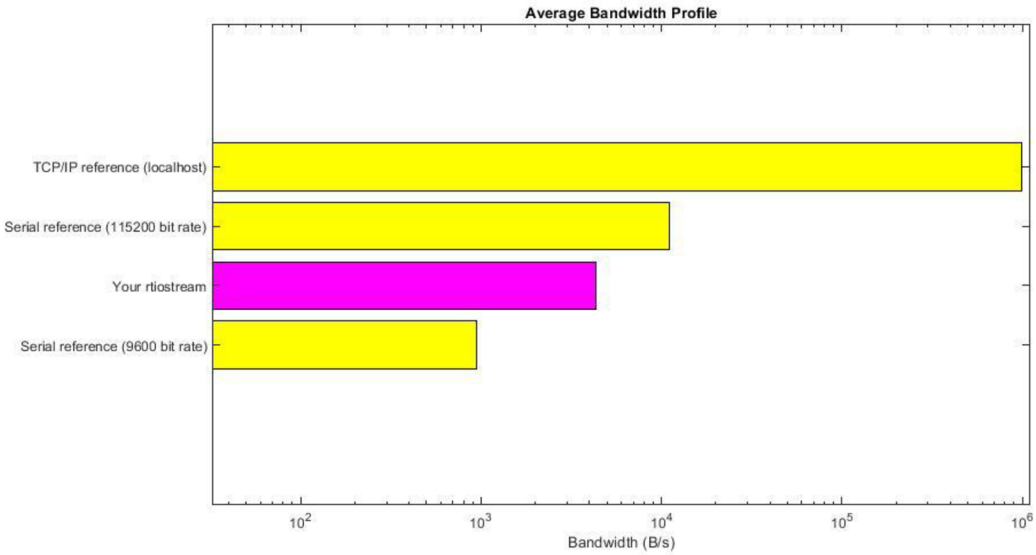
### 4.3.2 Test rtiostreamtest

Po overení funkčnosti otvorenia komunikačného portu, úspešného prijatia a odoslania správy predchádzajúcim spôsobom je následne potrebné spustiť test vytvorený spoločnosťou Mathworks, ktorým sa potvrdí správnosť vytvorenej sériovej komunikácie, ktorá bude môcť byť následne použitá pre vykonávanie PIL simulácie. Tento test sa spustí v mikrokontroléri odkomentovaním makra test_rtiostream.

Je taktiež nutné aby ostatné makrá pre test boli zakomentované.
Po nahratí programu do mikrokontroléra a jeho spustení, je potrebné pre začatie testu spustiť taktiež v Matlabu skript s názvom „ communicationTest.m “, v ktorom je predchystané nastavenie a volanie funkcie rtiostreamtest.
Jedná sa o funkciu, ktorá je špeciálne vytvorená pre testovanie správania implementácie používateľského rozhrania rtiostream, teda v našom prípade sériovej komunikácie. [35]

Aby bolo možné zabezpečiť správnu funkčnosť PIL simulácie je potrebné implementovať komunikáciu, ktorá úspešne prejde uvedeným testom.
Po spustení testu sa najskôr inicializuje spojenie s mikrokontrolérom a začnú sa vykonávať 3 testy.
V Command Window Matlabu pritom vypisuje informácie o priebehu testu, kde v prípade chýb v komunikácii taktiež vypíše chybu s predpokladaným dôvodom vzniknutej chyby.

Výpis úspešne vykonaného testu je uložený v textovom súbore v prílohe spolu s grafom, kde je porovnaná nami implementovaná sériová komunikácia s referenciou.
Uvedený graf získaný ako výstup z testu je taktiež uvedený na nasledujúcom obrázku Obr. 8.




Obr. 8: Porovnanie implementovanej sériovej komunikácie

Tak, ako aj v predošlom prípade je potrebné pred použitím tejto funkcie overiť správnosť nastavenia COM portu.
Taktiež je možné upraviť rýchlosť komunikácie pomocou baudrate.

### 4.3.3 Test Toolchain

Pomocou nastavenia makra test_PIL v zdrojovom súbore Cpu0_main je možné nahrať do mikrokontroléra kód, ktorý bude vykonávať PIL alebo SIL simuláciu.
Taktiež je potrebné aby ostatné makrá boli zakomentované.
Testovať funkčnosť vytváraného toolchain nebolo možné tak jednoduchým spôsobom, ako v prípade použitia funkcie rtiostreamtest, ktorá bola pred chystaná z Matlabu.
Overiť funkčnosť vytvoreného toolchain bolo možné iba spustením SIL alebo PIL simulácie a následne sledovať výpis z build procesu a opravovať jednotlivé chyby.
Postupnými krokmi sa bolo možné následne dostať až ku spusteniu samotnej SIL simulácie a po sfunkčnení sériovej komunikácie aj PIL simulácie.

## 4.4 SPUSTENIE SIL/PIL SIMULÁCIE

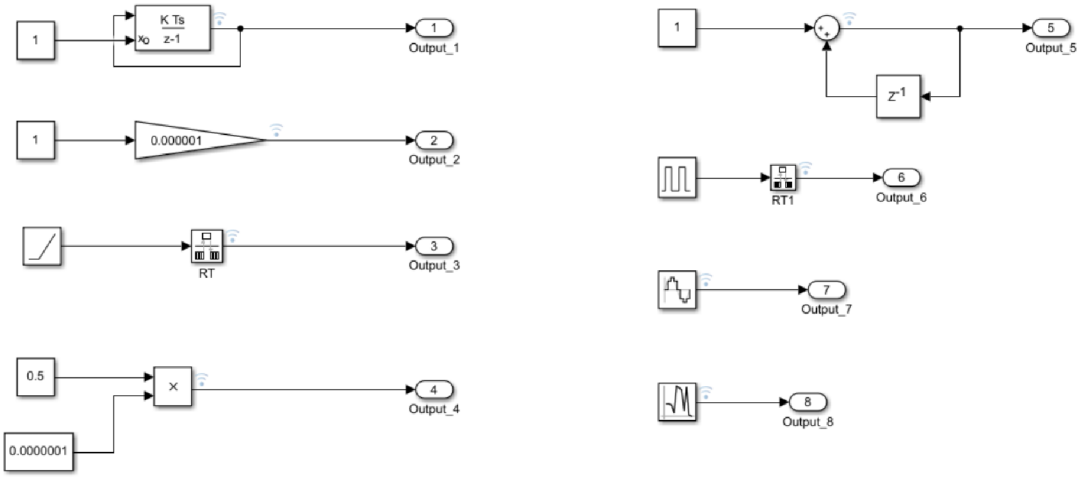
Najskôr je potrebné v Matlabe spustiť skript s názvom „startup.m“, pomocou ktorého sa nastavia potrebné cesty k jednotlivým súborom.
Samotné spustenie SIL alebo PIL simulácie sa vykonáva v Simulinku.
Pre potreby overenia správnosti daných simulácii je použitý model „pilmodelskuska.slx“, ktorý sa nachádza rovnako ako aj všetky ostatné potrebné

súbory v prílohe. Uvedený model je možné spustiť pre Top model simuláciu alebo pomocou modelovej referencie zo súboru „pilmodelskuskaX.slx“ aj pre SIL alebo PIL simuláciu subsystémov.
Pre spustenie modelu v PIL alebo SIL simulácii je potrebné prekliknúť v stavovej lište programu Simulink zo simulačného módu „Normal“ na mód „Processor-in-the-Loop (PIL)“ alebo „Software-in-the-Loop (SIL)“ a spustiť simuláciu modelu, akoby sa jednalo o obyčajnú simuláciu. Následne sa začne proces zostavovania kódu.

Ako už bolo v predchádzajúcich častiach naznačené, tak je taktiež možné si vybrať, či budeme vykonávať top level PIL simuláciu alebo PIL simuláciu iba pre jeden subsystém modelu. V ovládacom panely Simulinku je možné v časti daného okna s názvom System Under Test možné vybrať možnosť „Model blocks in SIL/PIL mode“ pre SIL/PIL simuláciu subsystému alebo „Top model“ pre SIL/PIL simuláciu celého modelu.

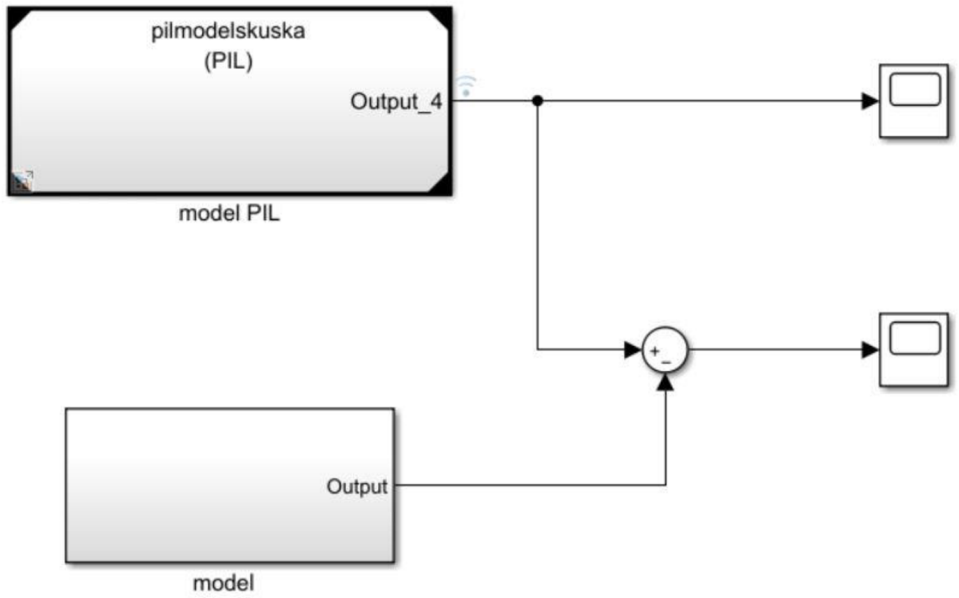
### 4.4.1 Simulačná schéma

Pre čo najväčšiu názornosť a pochopenie automaticky generovaného kódu bola ako simulačná schéma použitá čo najjednoduchšia simulačná schéma v rôznych variantoch. Uvedené jednoduché modelovacie schémy, na ktorých bola overená funkčnosť SIL/PIL simulácii sú znázornené na nasledujúcom obrázku Obr. 9.




Obr. 9: Ukážka simulovaných modelov

Jednotlivé modely, ako aj výsledky získané v jednotlivých simuláciách budú opísané v nasledujúcich častiach. Na predošlom obrázku Obr. 9 sa nachádza simulačná schéma pre Top model SIL/PIL simuláciu. Pre simuláciu System Under Test bola použitá simulačná schéma na nasledujúcom obrázku Obr. 10.




Obr. 10: Ukážka PIL simulácie subsystému „model PIL“

Na hore uvedenej simulačnej schéme sa nachádzajú dva subsystémy, ktoré v sebe obsahujú rovnaké časti z Top model schémy uvedenej na obrázku Obr. 9.
Ako subsystém pre PIL simuláciu je použitá modelová referencia ukazujúca na súbor „pilmodelskuska.slx“. Použitie modelovej referencie pre PIL simulácie je možné zmenou hodnoty v prepínači vytvorenom v skripte „startup.m“.

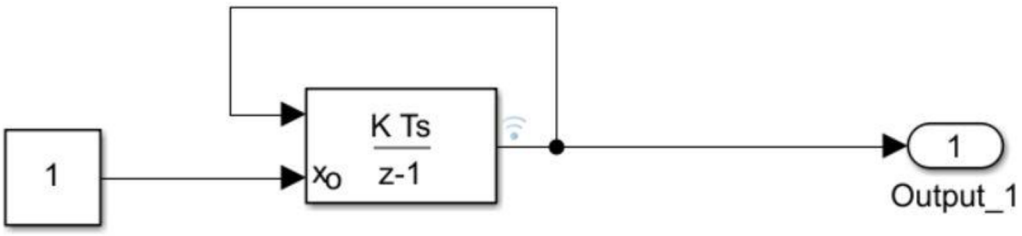
Pre uvádzané časti modelu sa v prílohe nachádzajú aj jednotlivé vygenerované zdrojové kódy a súbory pre SIL a PIL simuláciu.
Pre lepšiu názornosť automaticky generovaného kódu bol aktívny vždy len jeden výstup simulačnej schémy aby sa v zdrojovom kóde modelu nachádzala iba časť patriaca danej schéme.

## 4.5 POROVNANIE SIMULÁCII

Pre porovnanie simulácii, ako aj pre ich spúšťanie bol využitý SIL/PIL Manager spolu so Simulation Data Inspector, ktorým sa určovali zaznamenávané signáli.

### 4.5.1 Diskrétny integrátor

Na nasledujúcom obrázku Obr. 11 je znázornená jednoduchá schéma s použitím diskrétneho integrátoru. Ako je možné z uvedenej schémy poznať, počiatočná podmienka integrátoru mala veľkosť 1 a rovnakú hodnotu mala aj použitá vzorkovacia perióda.
Na vstup integrátoru je napojený vlastný výstup daného integrátoru.
Na výstupe diskrétneho integrátoru je možné poznať značku pripomínajúcu WI-FI signál, jedná sa však o označenie logovania signálu pomocou Simulation Data Inspector, ktoré je potrebné vykonať.




Obr. 11: Ukážka simulovaného modelu diskrétneho integrátoru

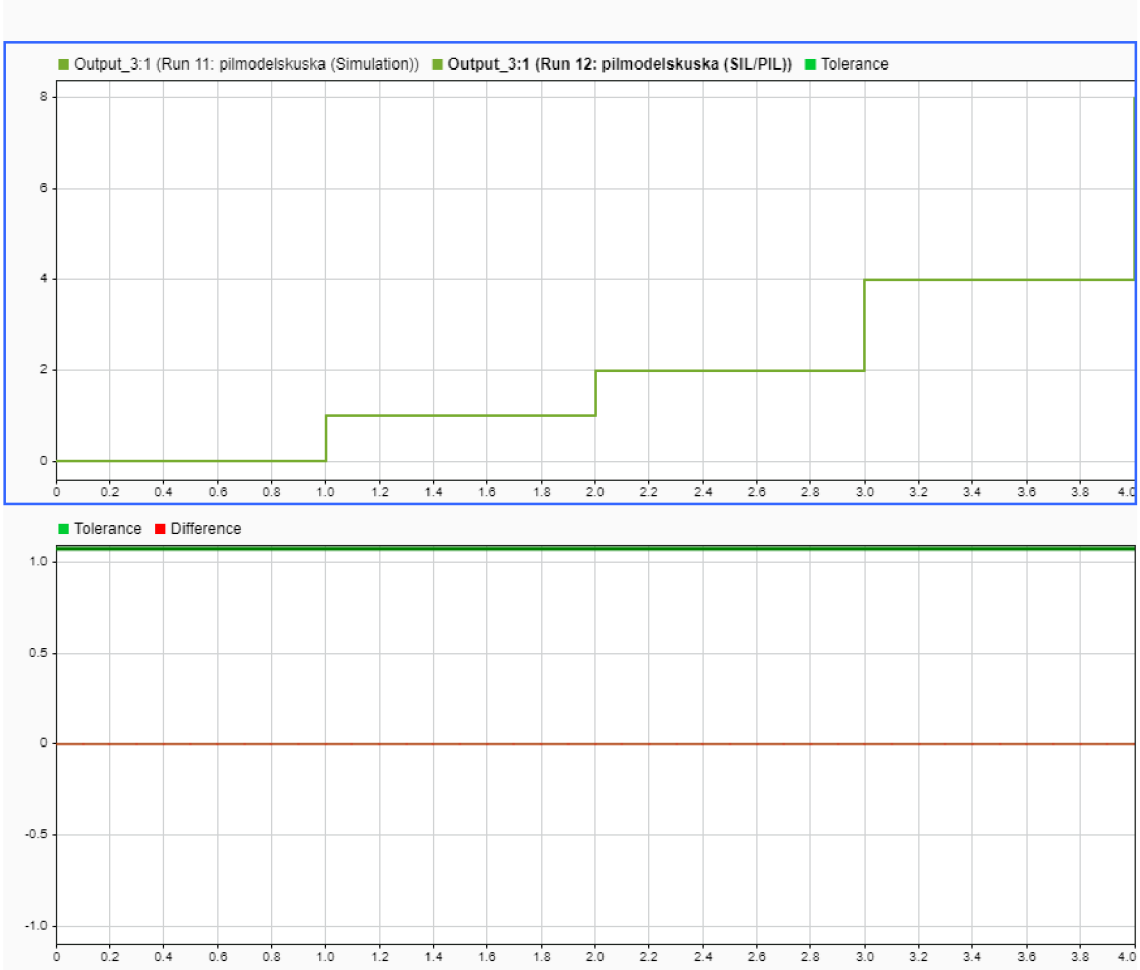
Na nasledujúcich riadkoch je znázornená funkcia pre výpočet jedného kroku simulácie. Uvedená funkcia bola ako aj zvyšok zdrojového kódu automaticky vygenerovaná a nachádza sa spolu so všetkými vygenerovanými súbormi v prílohe tejto práce.

```
void pilmodelskuska_step1(void)
{
  pilmodelskuska_Y.Output_1 =
pilmodelskuska_DW.DiscreteTimeIntegrator3_DSTATE;
  pilmodelskuska_DW.DiscreteTimeIntegrator3_DSTATE +=
    pilmodelskuska_DW.DiscreteTimeIntegrator3_DSTATE;
}
```

Ako je možné z predchádzajúcich riadkov zdrojového kódu poznať, nikde nie je uvedená perióda vzorkovania a rovnako je tomu aj v zbytku zdrojového kódu a to nielen u tejto simulovanej schémy. Generované zdrojové súbory pre mikrokontrolér nie sú nijako spojené s časom simulácie. Čas simulácie sa nastavuje iba v Simulinku.

#### 4.5.1.1 SIL Simulácia

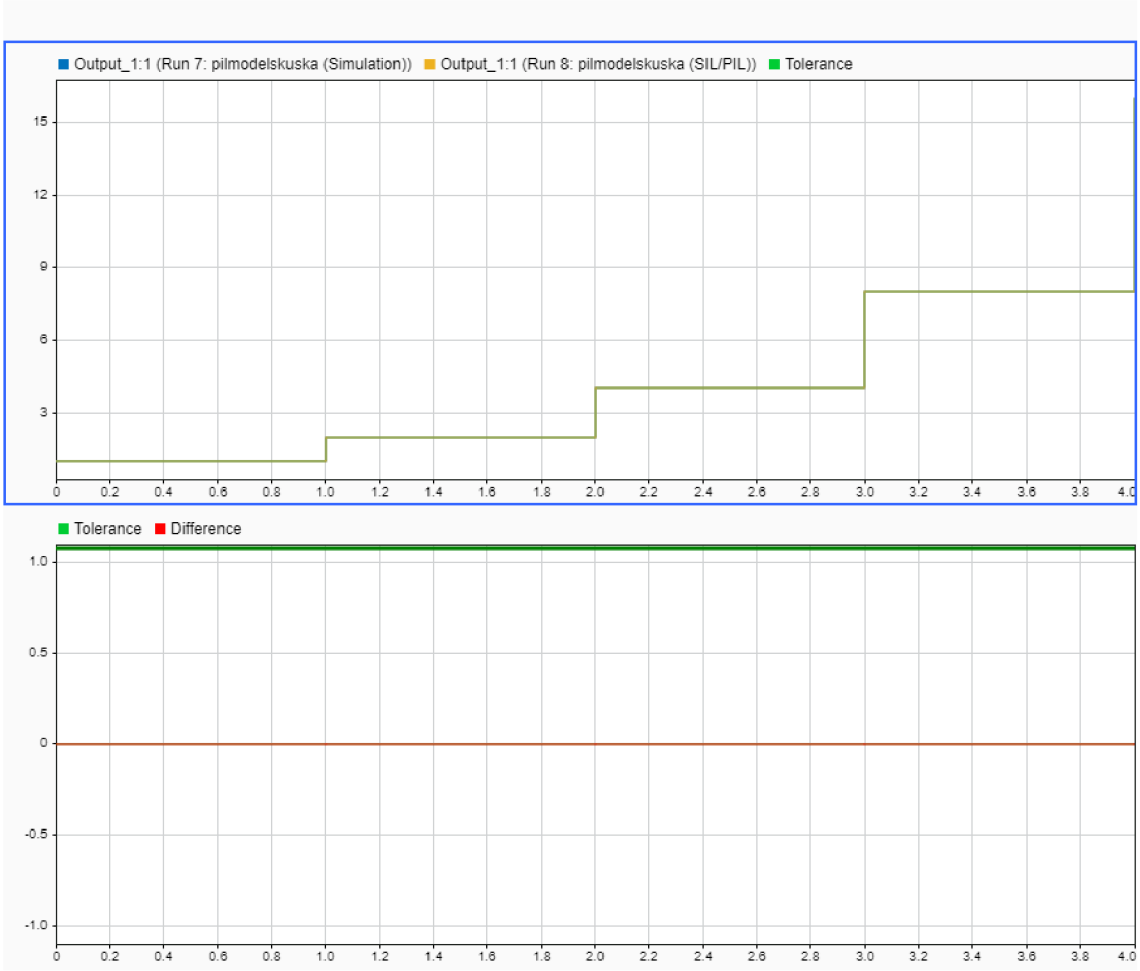
Pre vykonanie SIL simulácie nie je potrebné mať fyzicky pripojený hardvér predstavujúci nami používaný mikrokontrolér Aurix TriCore TC277D.
V SIL simulácii sa simuluje na základe nakonfigurovaného toolchain.
Na nasledujúcom obrázku Obr. 12 sa nachádza graf, ktorý porovnáva simulovaný priebeh diskrétneho integrátoru po dobu 4 sekúnd. Vo vrchnom okne sa nachádza priebeh spočítaný priamo v Simulinku a aj priebeh zo SIL simulácie mikrokontroléra. V spodnom okne sa nachádza rozdiel medzi uvedenými dvoma signálmi.




Obr. 12: Výstupné porovnanie signálov SIL simulácie

Ako je možné z predchádzajúceho grafu na obrázku Obr. 12 spoznať, tak rozdiel medzi výpočtom Simulinku a SIL simulácie nie je a ich výpočet je totožný.

#### 4.5.1.2 PIL Simulácia



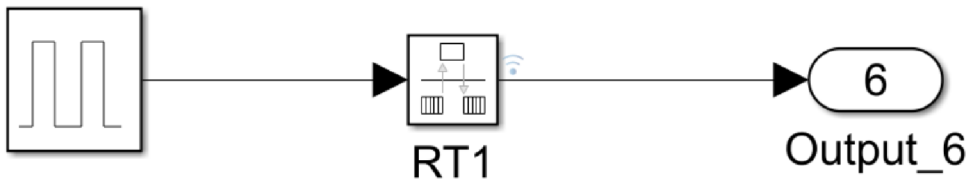

Obr. 13: Výstupné porovnanie signálov PIL simulácie

Tak ako aj v predošlom prípade SIL simulácie, tak aj v tomto prípade je rozdiel medzi výpočtom Simulinku a výpočtom, ktorý prebehol v mikrokontroléri nulový. Ako je možné poznať z oboch predchádzajúcich grafov, tak výpočet SIL a PIL simulácie je taktiež totožný, čo potvrdzuje správnosť nastavenia toolchain. Rovnako, ako sa dalo očakávať zo znalosti vygenerovaného zdrojového kódu zo simulačnej schémy, tak diskrétny integrátor integruje svoj výstup v danej vzorkovacej perióde.

### 4.5.2 Generátor obdĺžnikového signálu

Na nasledujúcom obrázku Obr. 14 je znázornená jednoduchá schéma s použitím generátoru obdĺžnikového signálu.

V schéme sa opäť vyskytuje označenie logovania signálu pomocou Simulation Data Inspector, ktoré je potrebné rovnako ako v predošlom prípade vykonať.
Taktiež je v schéme vložený blok s názvom „RateTransition“, ktorý bol do uvedenej schémy vložený automaticky Simulinkom.
Jedná sa o blok, ktorý slúži ako pomocný buffer a je schopný zaistiť dátovú integritu počas prenosu dát v prípade použitia rôznych vzorkovacích periód.
Pre overenie možností SIL/PIL simulácie bola nastavená perióda výstupného portu na jednu desatinu sekundy zatiaľ čo perióda vzorkovania generátoru obdĺžnikového signálu bola 1 sekunda. Vďaka automaticky vloženému bloku s označením RT1 to ale nebol žiaden problém.




Obr. 14: Ukážka simulovaného modelu generátoru obdĺžnikového signálu

Na nasledujúcich riadkoch je taktiež znázornená funkcia určená pre výpočet jedného kroku simulácie. Uvedená funkcia bola opäť, ako aj zvyšok zdrojového kódu automaticky vygenerovaná a nachádza sa rovnako spolu so všetkými vygenerovanými súbormi v prílohe tejto práce.

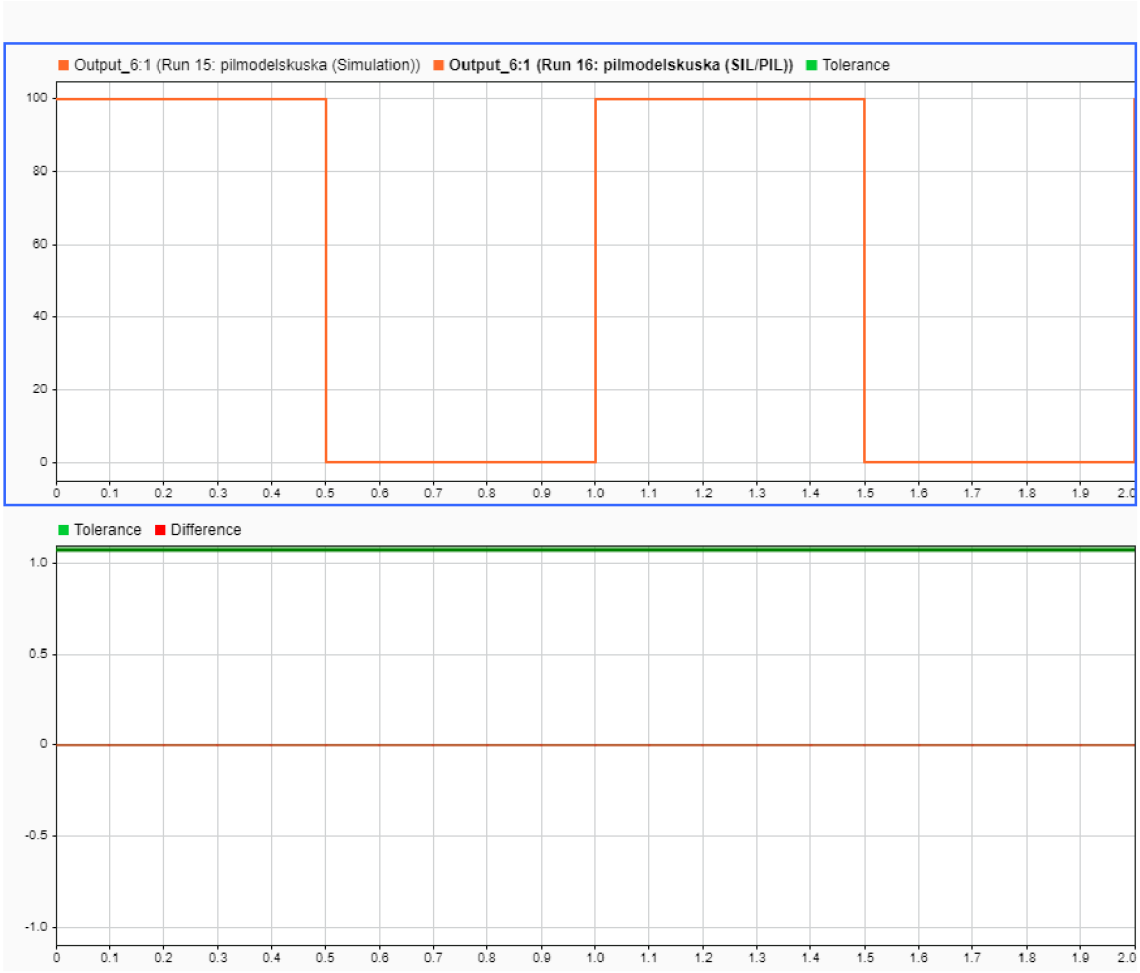
```
void pilmodelskuska_step(void)
{
  pilmodelskuska_Y.Output_6 = pilmodelskuska_DW.clockTickCounter
< 5 &&
    pilmodelskuska_DW.clockTickCounter >= 0 ? 100.0 : 0.0;
  if (pilmodelskuska_DW.clockTickCounter >= 9) {
    pilmodelskuska_DW.clockTickCounter = 0;
  } else {
    pilmodelskuska_DW.clockTickCounter++;
  }
}
```

Tak ako aj v predošlom prípade, tak aj v tomto nie je generovaný kód spätý so simulačným časom. Jedná sa o vlastnosť SIL a PIL simulácie,

kedy sa mikrokontrolér podieľa na simulácii vždy iba výpočtom jedného kroku simulácie, ktorej výsledok cez komunikačný kanál odošle do Simulinku.

#### 4.5.2.1 SIL Simulácia

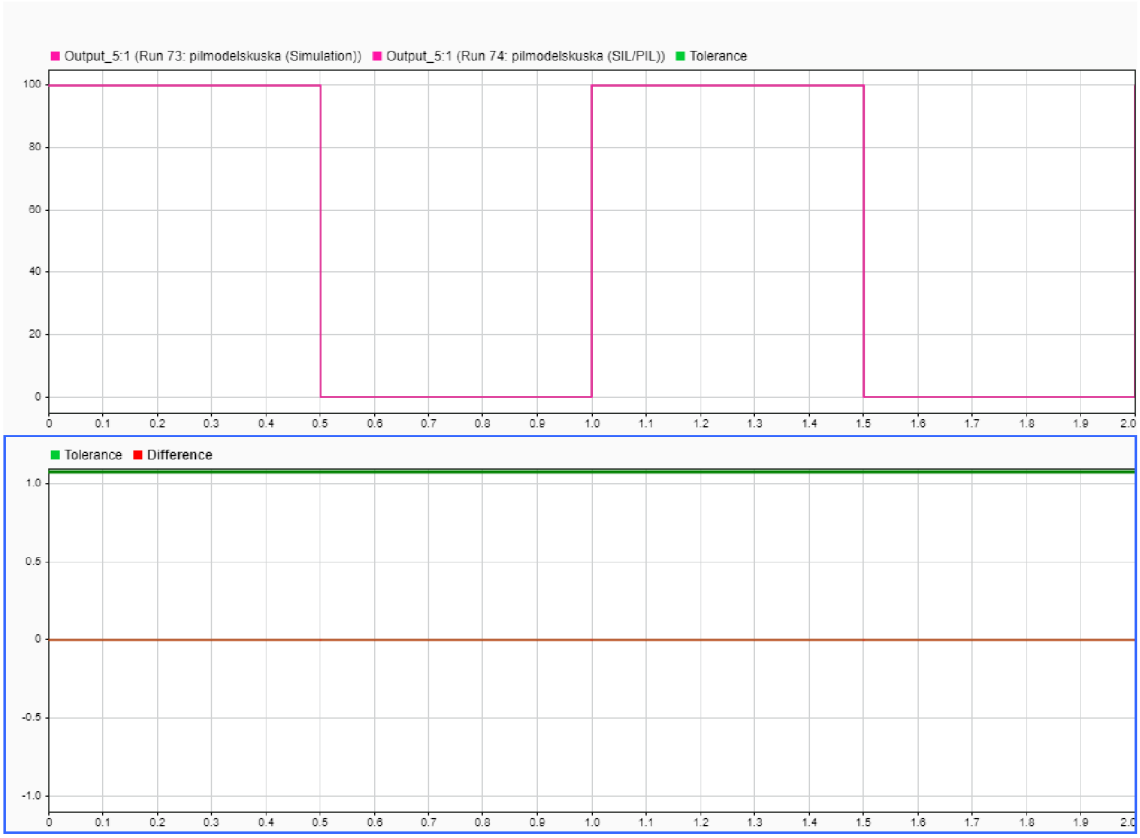
Ako je možné poznať z nasledujúceho priebehu, tak generátor obdĺžnikového signálu mal nastavenú amplitúdu s hodnotou 100 a periódu 1 sekundu a pulznú šírku 50%. Na nasledujúcom obrázku Obr. 15 sa taktiež nachádza graf získaný pomocou SIL/PIL Managera a súčasti Simulation Data Inspector, ktorý porovnáva simulovaný priebeh generátoru obdĺžnikového signálu. Vo vrchnom okne sa nachádza priebeh spočítaný priamo v Simulinku a aj priebeh zo SIL simulácie mikrokontroléra. V spodnom okne sa nachádza rozdiel medzi uvedenými dvoma signálmi.




Obr. 15: Výstupné porovnanie signálov SIL simulácie

#### 4.5.2.2 PIL Simulácia

Na nasledujúcom obrázku Obr. 16 sa nachádza rovnako ako v predchádzajúcich prípadoch graf získaný pomocou SIL/PIL Managera a Simulation Data Inspector. Ten porovnáva simulovaný priebeh generátoru obdĺžnikového signálu.
V hornom okne sa nachádza priebeh spočítaný priamo v Simulinku spolu s priebehom z PIL simulácie mikrokontroléra.
V dolnom okne sa nachádza rozdiel medzi dvomi porovnávanými signálmi.




Obr. 16: Výstupné porovnanie signálov PIL simulácie

Tak ako aj v predošlých prípadoch SIL simulácie, tak aj v tomto prípade je rozdiel medzi výpočtom programu Simulink a výpočtom, ktorý prebiehal v mikrokontroléri je taktiež nulový.
Ako je možné poznať z oboch predchádzajúcich grafov, tak výpočet SIL a PIL simulácie je opäť totožný.

### 4.5.3 Sínusoida

Pre vygenerovanie zdrojového kódu bola použitá najjednoduchšia možná modelovacia schéma sínusoidy, ktorej výstup sa odošle priamo na výstupný port.


Obr. 17: Ukážka simulovaného modelu sínusového signálu

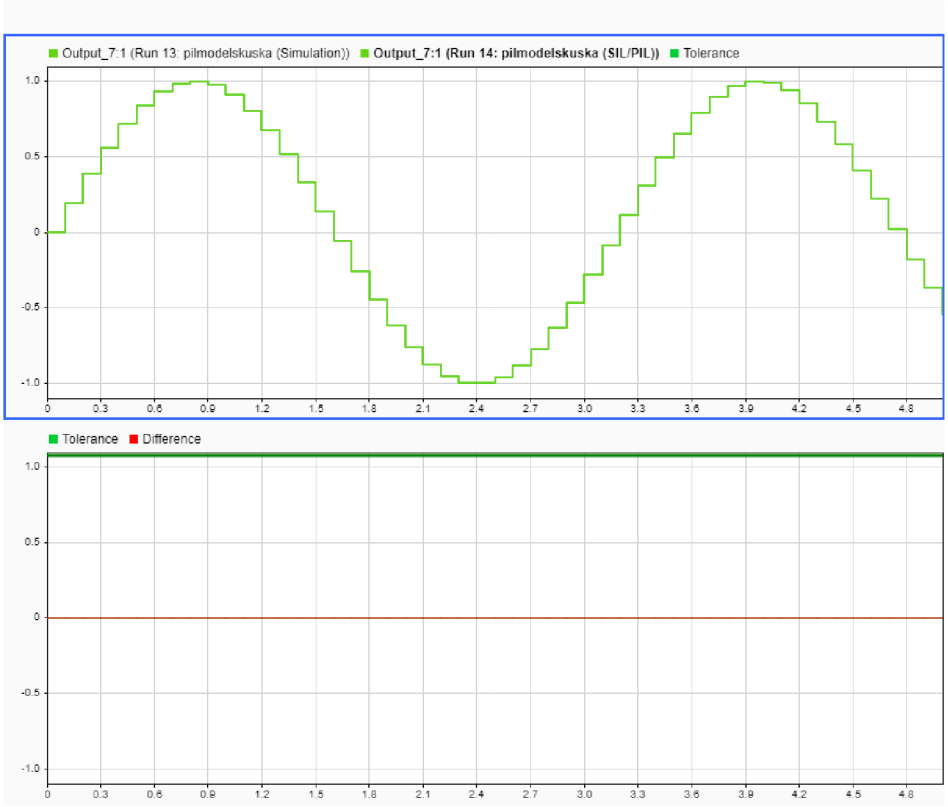
Na nasledujúcich riadkoch bude uvedená funkcia pre výpočet jedného kroku simulácie s názvom „pilmodelskuska_step“.
Z pohľadu názornosti a čitateľnosti automaticky generovaného zdrojového kódu je na tomto prípade možné asi najlepšie poznať dôvod, prečo boli pre SIL/PIL simulácie použité práve tie najjednoduchšie modelové schémy.

```
void pilmodelskuska_step(void)
{
  real_T lastSin_tmp;
  if (pilmodelskuska_DW.systemEnable != 0) {
    lastSin_tmp = 5.0 * ((pilmodelskuska_M->Timing.clockTick0) *
0.1);
    pilmodelskuska_DW.lastSin = sin(lastSin_tmp);
    pilmodelskuska_DW.lastCos = cos(lastSin_tmp);
    pilmodelskuska_DW.systemEnable = 0;
  }

  pilmodelskuska_Y.Output_5 = (pilmodelskuska_DW.lastSin *
0.87758256189037276 +
    pilmodelskuska_DW.lastCos * -0.479425538604203) *
0.87758256189037276 +
    (pilmodelskuska_DW.lastCos * 0.87758256189037276 -
pilmodelskuska_DW.lastSin
     * -0.479425538604203) * 0.479425538604203;
  lastSin_tmp = pilmodelskuska_DW.lastSin;
  pilmodelskuska_DW.lastSin = pilmodelskuska_DW.lastSin *
0.87758256189037276 +
    pilmodelskuska_DW.lastCos * 0.479425538604203;
  pilmodelskuska_DW.lastCos = pilmodelskuska_DW.lastCos *
0.87758256189037276 -
    lastSin_tmp * 0.479425538604203;
  pilmodelskuska_M->Timing.clockTick0++;
}
```

#### 4.5.3.1 SIL Simulácia

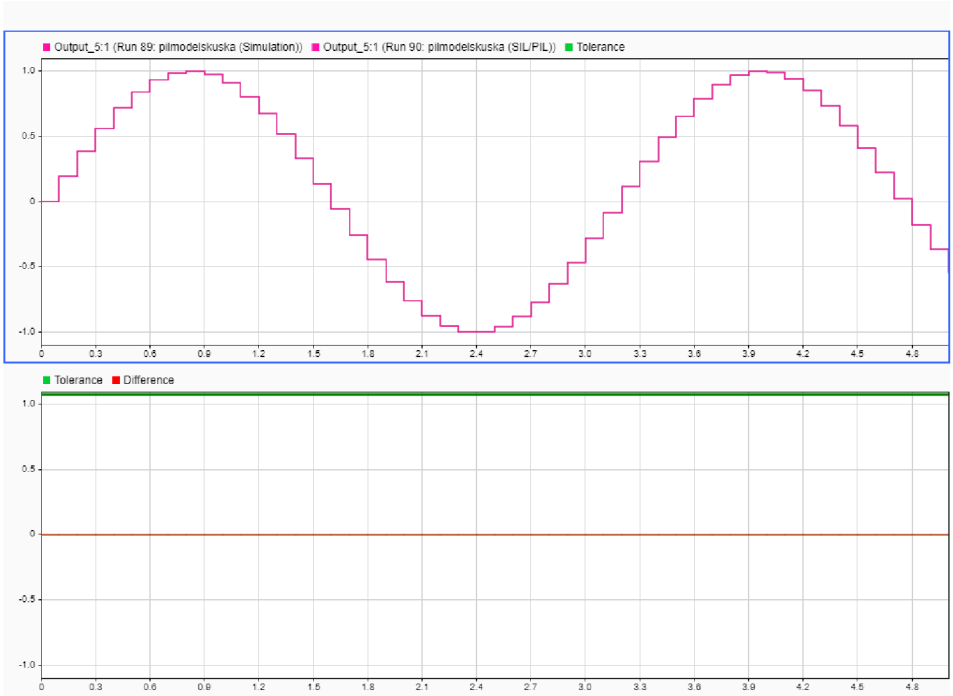
Pre SIL a rovnako aj PIL simuláciu bol použitý sínusový signál s amplitúdou 1 a s frekvenciou 2Hz pre perióde vzorkovania 0,1s.




Obr. 18: Výstupné porovnanie signálov SIL simulácie

Ako je možné z uvedených grafov SIL a aj PIL simulácie uvedenej na ďalšom grafe nachádzajúcom sa na obrázku Obr. 19 je simulácia prebiehajúca v Simulinku totožná s výpočtom v mikrokontroléri.

#### 4.5.3.2 PIL Simulácia




Obr. 19: Výstupné porovnanie signálov PIL simulácie

### 4.5.4 Generátor náhodných čísiel

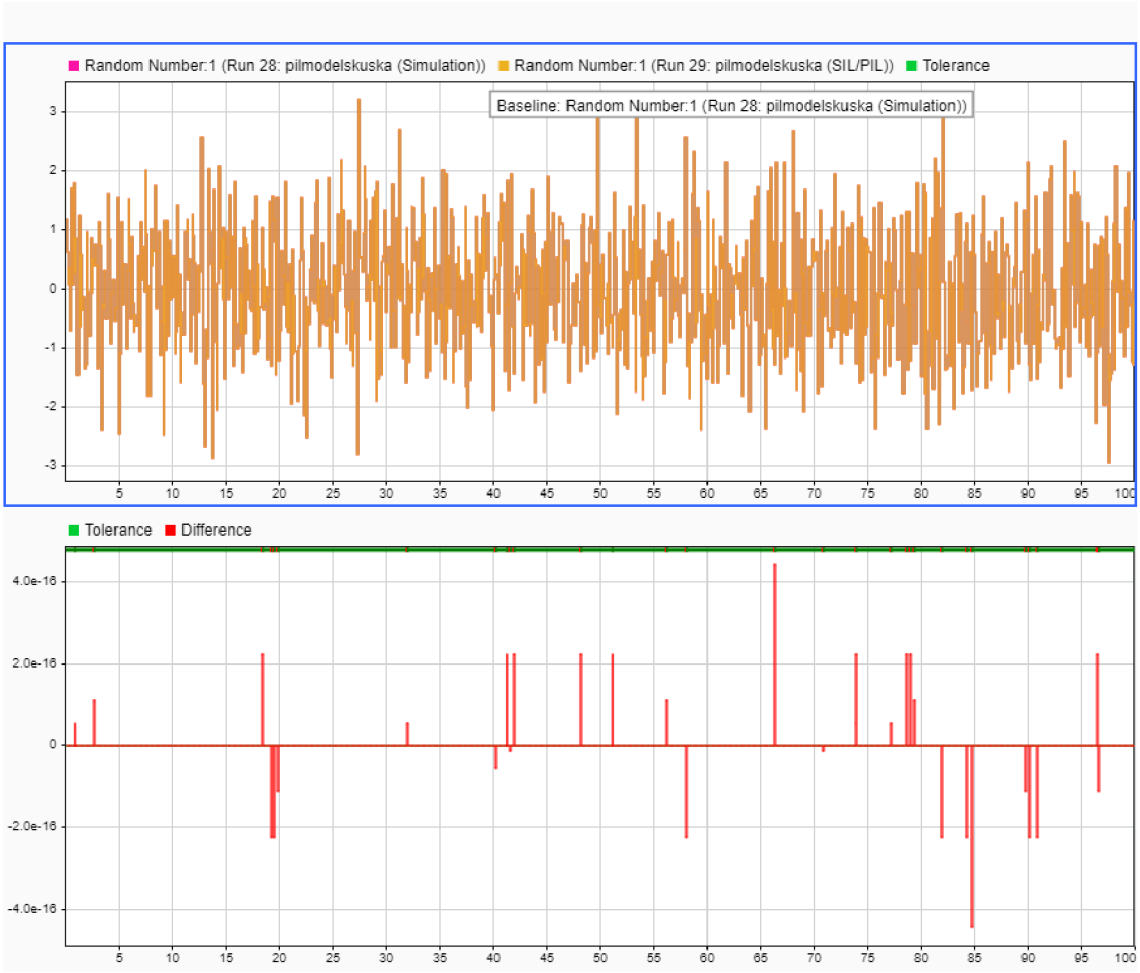
Nakoľko by mala SIL/PIL simulácia slúžiť pre overovanie funkčnosti programu v embedded systémoch a hľadania prípadných limitov funkčnosti a nie len overovania prípadnej funkčnosti komunikácie a nastavení, hľadali sme taktiež možné hranice, ktoré je možné prekonať a na prípadné problémy pri vykonávaní kódu naraziť. Zo všetkých vyskúšaných riešení sa nám podarilo zachytiť ale iba jednu, a to konkrétne s použitím generátoru náhodných čísiel s normálnym Gaussovským rozložením.


Obr. 20: Ukážka simulovaného modelu generátoru náhodných čísel

Ako je možné z uvedenej schémy poznať opäť bola použitá veľmi jednoduchá schéma.

Uvedený signál náhodných hodnôt s normálnym Gaussovským rozložením mal nastavené nasledujúce parametre, ktoré bolo možné nastaviť.
Uvedený generovaný signál náhodných hodnôt mal počiatočnú hodnotu nula a priemer signálu bola nastavená taktiež nula.
Rozptyl hodnôt 1 a vzorkovací čas opäť ako v predchádzajúcom prípade 0,1 sekundy.
Na nasledujúcom riadku je možné nájsť funkciu pre výpočet jedného kroku simulácie. V nej je volaná funkcia pre generovanie náhodných čísel, ktorá je pomerne rozsiahla. Celá zložka so všetkými vygenerovanými súbormi sa taktiež nachádza v prílohe tejto práce.

```
void pilmodelskuska_step(void)
{
  pilmodelskuska_Y.Output_5 = pilmodelskuska_DW.NextOutput;
  pilmodelskuska_DW.NextOutput = rt_nrand_Upu32_Yd_f_pw
    (&pilmodelskuska_DW.RandSeed);
}
```



Obr. 21: Výstupné porovnanie signálov PIL simulácie

Ako je možné z predchádzajúceho obrázku poznať, tak na prvom grafe, v ktorom je použité porovnanie signálov, ktorých jednotlivé hodnoty boli vypočítané v Matlabe a v mikrokontroléri nie je možné nájsť rozdiel ale v spodnom grafe, ktorý určuje ich odchýlku je možné nájsť viaceré rozdiely ale rádovo na veľmi nízkej hodnote, čo môže byť spôsobené chybou zaokrúhľovania alebo sa môže taktiež jednať o rozdiel vo výpočte mikrokontroléra medzi Matlabom z toho dôvodu, že nami používaný mikrokontrolér využíva pre výpočet čísiel s plávajúcou desatinnou čiarkou takzvanú „single precision“ zatiaľ čo Matlab počíta v „double precision“.

## 4.6 ZÁVEREČNÉ USTANOVENIA

Pred samotným návodom na použitie, ktorý bude štruktúrovaný v jednotlivých bodoch je potrebné spomenúť a zhrnúť niekoľko veľmi dôležitých poznatkov, a to konkrétne:

PIL simulácia je síce funkčná ale na jej spustenie na inom počítači nestačí iba stiahnuť súbory z prílohy tejto práce a spustiť SIL/PIL simuláciu v Simulinku ale je potrebné niektoré časti konfigurácie upraviť, nakoľko inštalačné zložky niektorých požadovaných súborov budú mať rozdielne úložisko.

Taktiež sa v prílohe tejto práce nenachádzajú všetky potrebné súbory pre vykonávanie PIL simulácie nakoľko neboli vytvorené ani editované autorom tejto práce ale boli vytvorené spoločnosťou Mathworks.

Z tohto dôvodu boli vytvorené Matlabovské skripty opisované v predchádzajúcej kapitole o utilitách pre nakopírovanie požadovaných súborov do potrebných súborových zložiek.

Rovnako je potrebné spomenúť, že je nutné uvedené názvy skriptov, súborov a zložiek ponechať pôvodné. Zmena ich názvov je síce možná ale nie je veľmi výhodná, nakoľko by následne bolo nevyhnutné vykonať úpravy vo viacerých súboroch.

Je vhodné taktiež spomenúť, že uvedenú PIL simuláciu je možné vykonávať na akomkoľvek mikrokontroléri Aurix TriCore.
Na to je ale potrebné vykonať zmeny v niektorých nastaveniach, ktoré určujú konkrétny typ mikrokontroléra a overiť správnosť označenia portov pre prijímanie a odosielanie dát po sériovej linke pri ich prípadnej zmene.

Tak, ako už bolo spomenuté v časti tejto diplomovej práce o programových utilitách, je potrebné taktiež overiť, na ktorý COM port sa používaný mikrokontrolér v počítači pripája a prípadný rozdielny COM port prepísať v súboroch, ktoré to vyžadujú.

Ako je z predchádzajúcich častí tejto práce jasné, je najskôr potrebné pred začatím používania PIL simulácie pre verifikáciu najskôr nainštalovať všetok potrebný softvér HighTec IDE spolu s iLLD ovládačmi a ďalším podporným softvérom, ktorý v prípade vlastnenia licencie bude automaticky nainštalovaný s vývojovým prostredím HighTec IDE, aby bolo možné naprogramovať mikrokontrolér Aurix TriCore TC 277D, ktorý je taktiež potrebné mať pripojený k počítaču s Matlabom, ktorý obsahuje potrebné toolboxy opisované v predošlých častiach tejto diplomovej práce. Samotný Matlab musí byť vo verzii R2019b alebo novšej. Taktiež je nutné aby Matlab používal kompilátor Microsoft Visual C++ 2017 (C) pre generovanie MEX súborov.
Či je uvedený kompilátor v Matlabe nastavený je možné zistiť pomocou príkazu „ mex -setup “, ktorý sa zadá do Command Window Matlabu.

Dole uvedený návod na použitie nebude zahŕňať časti týkajúce sa inštalácie potrebného softvéru ale bude predpokladať jeho úplnú funkčnosť. Taktiež nebude opisovať stiahnutie a rozbalenie jednotlivých príloh práce.

## 4.7 NÁVOD NA POUŽITIE

1.) Vývojové prostredie HighTec IDE je potrebné spúšťať pomocou dávkového súboru, ktorý sa nachádza v prílohe číslo 3. Pred použitím daného dávkového súboru je ho potrebné otvoriť napríklad v poznámkovom bloku a upraviť cesty s jednotlivými súbormi podľa toho, kde sa v používanom počítači spúšťané súbory nachádzajú.

2.) Po spustení HighTec IDE je potrebné importovať projekt BaseFramework_TC27D do vývojového prostredia HighTec IDE, v ktorom je potrebné vykonať úpravy podľa predchádzajúcich častí tejto práce týkajúce sa nastavenia cesty ku kompilátoru v súbore „Config_Gnuc.mk“ a následne upraviť súbory pre Debug konfiguráciu podľa kapitoly 3.4.6, alebo podľa návodu k prostrediu HighTec IDE . Taktiež je vhodné overiť v súbore „Config.mk“ správnosť nastavenia cesty k toolchain, ktorá môže byť v prípade rozdielnej inštalačnej zložky HighTec taktiež rozdielna.
Po daných úpravách je potrebné vykonať rebuild projektu.

3.) Po spustení programu Matlab je potrebné v skripte "startup.m" upraviť súborové cesty prislúchajúce globálnym premenným, ktoré sa v danom skripte nastavujú.

4.) Spustiť skript "startup.m" pri každom spustení Matlabu, v prípade konfliktných sl_customization, je potrebné ich pomocou príkazu rmpath odstrániť z vyhľadávacej cesty Matlabu.

5.) Spustiť skript“ copyMatlabFilesXIL.m“, ktorý ale postačuje spustiť iba pri prvej SIL/PIL simulácii.

6.) Spustiť skript "update_mat_file.m" – pri prvej SIL/PIL simulácii.

7.) V otvorenom Simulinkovom modely spustiť verifikáciu modelu pre PIL alebo SIL simuláciu.

8.) Po pozastavení simulácie je potrebné nahrať kód do mikrokontroléra.

9.) Po naprogramovaní mikrokontroléra a spustení jeho programu sa stlačením akéhokoľvek tlačidla v Command Window Matlabu bude simulácia pokračovať a po jej skončení sa zobrazí okno Data Inspectoru.

# ZÁVER

Tvorbu PIL simulácie sprevádzalo množstvo problémov, či už softvérových ale aj hardvérových. Z hardvérových je možné spomenúť nefunkčnú dosku prvého mikrokontroléra, alebo aj nepoužiteľné USB porty na mojom notebooku, na ktorom bola vytváraná táto práca, nakoľko bez USB HUBu nebolo možné ani len spustiť sériovú komunikáciu medzi počítačom a Aurixom a nahrávanie kódu prebiehalo taktiež na niekoľko pokusov kvôli neustálym chybám licenseru vývojového prostredia HighTec IDE.
Samozrejme bolo strateného veľa času, kým sa prišlo na to, že chyba je iba v USB portoch a nie vo vytváranom kóde pre komunikáciu. Taktiež na túto prácu boli použité 2 USB HUBy, nakoľko sa prvý z nich zrejme z dôvodu dlhodobého zaťaženia mikrokontrolérom odpálil.
Ako vyplíva zo zadania diplomovej práce, tak sme sa oboznámili s automatickým generovaním kódu v jazyku C z prostredia Matlab a Simulink, ako aj so všetkými rozsiahlimi spôsobmi nastavenia parametrov generovania kódu. Taktiež sme sa zoznámili s novinkami nami používanej verzie Matlabu pre vylepšenie generovania kódu. Všetky potrebné informácie sa nachádzajú v jednotlivých častiach práce, rovnako ako aj opis metód SIL a PIL.
Podarilo sa nám úspešne vytvoriť komunikáciu cez sériové rozhranie medzi počítačom s Matlabom a mikrokontrolérom Aurix za pomoci iLLD ovládačov. Správnosť komunikácie bola otestovaná pomocou funkcie Matlabu s názvom „rtiostreamtest“, ktorej výstup je možné nájsť na obrázku Obr. 8.
Pre komunikáciu mikrokontroléra s Matlabom nebolo použité rozhranie ethernet nakoľko sériová rozhranie plne postačovalo.
Metódu SIL a PIL simulácie sme vykonávali pomocou vytvoreného target systému Aurix TriCore TC277D, z ktorého získané výsledky simulácii je možné nájsť v kapitole o porovnaní simulácii. Pre naprogramovanie mikrokontroléra je ale potrebné využiť vývojové prostredie HighTec IDE, nakoľko automaticky generovaný „.hex“ súbor z prostredia Matlabu nebolo možné použiť.
Porovnaním priebehov zo SIL a PIL simulácie nebol zistený rozdiel.
Samotný proces zostavovania kódu pre SIL a PIL simuláciu je ale úspešný a vytvorenú konfiguráciu, ako aj nakonfigurovaný projekt v HighTec IDE je možné používať pre verifikáciu modelov aj na iných počítačoch ako bola táto práca vytvorená.

# LITERATÚRA


[1] P. Karban: Výpočty a simulace v programech MATLAB a Simulink. Computer Press, a.s., ISBN 978-80-251-1448-3, 2006.

[2] Matlab Coder [online]. [cit. 11.11.2019]. Dostupné z: https://www.mathworks.com/products/matlab-coder.html

[3] Matlab Coder [online]. [cit. 12.11.2019]. Dostupné z: https://www.humusoft.cz/matlab/matlab-coder/

[4] Processor-in-the-Loop Simulation [online]. [cit. 11.11.2019]. Dostupné z: https://www.mathworks.com/help/ecoder/processor-in-the-loop.html?s_tid=srchtitle

[5] Cross compilation [online]. [cit. 11.11.2019]. Dostupné z: https://www.gnu.org/software/automake/manual/html_node/Cross_002dCompilation.html

[6] SIL/PIL Manager [online]. [cit. 12.11.2019]. Dostupné z: https://www.mathworks.com/help/ecoder/ref/silpilmanager-app.html

[7] Products and Services [online]. [cit. 13.11.2019]. Dostupné z: https://www.mathworks.com/products.html?s_tid=gn_ps

[8] Model-Based Design for Embedded Control Systems [online]. [cit. 11.11.2019]. Dostupné z: https://www.mathworks.com/content/dam/mathworks/white-paper/gated/model-based-design-with-simulation-white-paper.pdf

[9] Dávkové soubory [online]. [cit. 19.11.2019]. Dostupné z: https://www.gvp.cz/ucebnice/Vyptech/windows/davky.htm

[10] ASCLIN [online]. [cit. 19.11.2019]. Dostupné z: https://www.infineon.com/dgdl/Infineon-AURIX_Asynchronous+Synchronous+Interface-TR-v01_00-EN.pdf?fileId=5546d46269bda8df0169ca52fd3e251f

[11] Registering Customizations [online]. [cit. 29.11.2019]. Dostupné z: https://www.mathworks.com/help/simulink/ug/registering-customizations.html

[12] Customize Build Process with sl_customization.m [online]. [cit. 29.11.2019]. Dostupné z: https://www.mathworks.com/help/rtw/ug/customizing-the-target-build-process-with-sl-customization-m.html?s_tid=srchtitle

[13] What is Hardware-In-the-Loop Simulation?. [online]. [cit. 21.12.2019]. Dostupné z: https://www.mathworks.com/help/physmod/simscape/ug/what-is-hardware-in-the-loop-simulation.html

[14] Simulink Coder - MATLAB & Simulink. *MathWorks - Makers of MATLAB and Simulink* [online]. Copyright © 1994 [cit. 21.12.2019]. Dostupné z: https://www.mathworks.com/products/simulink-coder.html

[15] Production Code Generation Introduction and New Technologies [online]. [cit. 22.12.2019]. Dostupné z: https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/automotive/files/mac2007/4_pcgintro.pdf

[16] Release notes R2015b [online]. [cit. 28.11.2019]. Dostupné z: https://www.mathworks.com/help/matlab/release-notes-R2015b.html

[17] R2015b Release Highlights [online]. [cit. 28.11.2019]. Dostupné z: https://www.mathworks.com/products/new_products/release2015b.html

[18] A.R:Plummer: Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering. Thousand Oaks, CA: Sage. ISSN 2041-3041.

[19] MIL simulation – Model in the Loop [online]. [cit. 28.11.2019]. Dostupné z: https://www.rexcontrols.com/mil-simulation

[20] Processor-In-Loop Simulation [online]. [cit. 28.11.2019]. Dostupné z: https://www.design-reuse.com/articles/42548/embedded-software-verification-validation-in-model-based-development.html

[21] Infineon BIFACES: Migrate iLLD demos to managed HighTec projects, version 2.4, May 2019 [cit. 28.11.2019]. Dostupné po schválení registrácie z: https://sso.infineon.com/idp/prp.wsf?wa=wsignin1.0&wtrealm=urn%3amyicp_SSO&wctx=https%3a%2f%2fmyicp.infineon.com%2fsites%2fmicrocontrollers-tools_connection%2f_layouts%2fAuthenticate.aspx%3fSource%3d%252Fsites%252Fmicrocontrollers%252Dtools%255Fconnection%252F%255Flayouts%252FOSSSearchResults%252Easpx%253Fk%253DBIFACES%255FV1%255F0%255F2%255FWin32%252Ezip

[22] 32-Bit Microcontroller Application Kit TC2X7 [cit. 29.11.2019]. Dostupné po schválení registrácie z:
https://sso.infineon.com/idp/prp.wsf?wa=wsignin1.0&wtrealm=urn%3amyicp_SSO&wctx=https%3a%2f%2fmyicp.infineon.com%2fsites%2fmicrocontrollers-tools_connection%2f_layouts%2fAuthenticate.aspx%3fSource%3d%252Fsites%252Fmicrocontrollers%252Dtools%255Fconnection%252F%255Flayouts%252FOSSSearchResults%252Easpx%253Fk%253DBIFACES%255FV1%255F0%255F2%255FWin32%252Ezip

[23] Generate C and C++ code optimized for embedded systems [online]. [cit. 8.12.2019]. Dostupné z:
https://www.mathworks.com/products/embedded-coder.html?s_tid=srchtitle

[24] Generate C and C++ code optimized for embedded systems [online]. [cit. 8.12.2019]. Dostupné z:
https://www.mathworks.com/hardware-support/infineon.html

[25] GCC and Make[online]. [cit. 9.12.2019]. Dostupné z:
https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html

[26] Adding a Custom Toolchain [online]. [cit. 9.12.2019]. Dostupné z:
https://www.mathworks.com/help/coder/ug/adding-a-custom-toolchain.html

[27] Target Language Compiler Directives [online]. [cit. 9.12.2019]. Dostupné z:
https://www.mathworks.com/help/rtw/tlc/target-language-compiler-directives.html

[28] CHROMIAK, Michael. *Tepelný model kabiny automobilu pro HIL simulaci* [online]. Brno, 2018 [cit. 2020-03-22]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/110959. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Jan Glos.

[29] Embedded Coder Interface to QEMU Emulator [online]. [cit. 24.03.2020]. Dostupné z:
https://www.mathworks.com/matlabcentral/fileexchange/57465-embedded-coder-interface-to-qemu-emulator

[30] How to guarantee signal integrity [online]. [cit. 03.05.2020]. Dostupné z:
https://resources.altium.com/p/how-to-guarantee-signal-integrity-in-your-pcb-design-for-serial-communication-protocols

[31] Lengths of serial cables [online]. [cit. 03.05.2020]. Dostupné z: https://www.tldp.org/HOWTO/Remote-Serial-Console-HOWTO/serial-distance.html

[32] J. Andrews, J. Dark, J.West: CompTIA A+ Guide to IT technical support., (10th ed.). Cengage Learning, ISBN 978-0-357-10829-1, 2020.

[33] How to improve signal integrity in Rapid IO serial interface design [online]. [cit. 03.05.2020]. Dostupné z: https://www.embedded.com/how-to-improve-signal-integrity-in-rapid-io-serial-interface-designs/

[34] Serial port Basics [online]. [cit. 03.05.2020]. Dostupné z: http://www.acumeninstruments.com/Support/documentation/SerialPortBasics/index_pg5.shtml

[35] rtiostreamtest[online]. [cit. 03.05.2020]. Dostupné z: https://www.mathworks.com/help/ecoder/ref/rtiostreamtest.html

[36] Processor-In-the-Loop Simulation on Embedded Linux Boards [online]. [cit. 04.05.2020]. Dostupné z: https://www.mathworks.com/company/newsletters/articles/processor-in-the-loop-simulation-on-embedded-linux-boards.html

[37] coder.make.BuildTool class [online]. [cit. 04.05.2020]. Dostupné z: https://www.mathworks.com/help/coder/ref/coder.make.buildtool-class.html

[38] SIL/PIL Manager [online]. [cit. 04.05.2020]. Dostupné z: https://www.mathworks.com/help/ecoder/ref/silpilmanager-app.html?s_tid=srchtitle

[39] What's New in Simulink R2019b [online]. [cit. 04.05.2020]. Dostupné z: https://www.mathworks.com/videos/whats-new-in-simulink-r2019b-1567201554373.html?s_tid=srchtitle

[40] Simulation Data Inspector [online]. [cit. 04.05.2020]. Dostupné z: https://www.mathworks.com/help/simulink/slref/simulationdatainspector.html

[41] Steve Chamberlain, Ian Lance Taylor: The GNU linker, Red Hat Inc, GNU Binutils documentation, 2009.

[42] Richard M. Stallman and the GCC Developer Community: Using the GNU Compiler Collection[online]. [cit. 04.05.2020]. Dostupné z: https://gcc.gnu.org/onlinedocs/gcc-4.6.1/gcc.pdf

# Zoznam symbolov, veličín a skratiek

| | | |
|---|---|---|
| FEKT | - | Fakulta elektrotechniky a komunikačních technologií |
| VUT | - | Vysoké učení technické v Brně |
| HIL | - | Hardware in the Loop |
| Obr. | - | Obrázok |
| Tab. | - | Tabuľka |
| cit. | - | Citované |
| č. | - | Číslo |
| min. | - | Minimalizácia |
| max. | - | Maximalizácia |
| mat. | - | Matematických |
| dif. | - | Diferenciálnych |
| FP | - | Fixed-Point ( pevná rádová čiarka ) |
| MaO | - | Modelovanie a optimlizácia |
| MaS | - | Modelovanie a simulácia |
| r.č. | - | rádová čiarka |
| m.sys. | - | mechanické systémy |
| EE | - | Elektro energetické |
| sys. | - | Systémy |
| 3D | - | Trojdimenzionálny |
| MIL | - | Model in the Loop |
| SIL | - | Software in the Loop |
| PIL | - | Processor in the Loop |
| HIL | - | Hardware in the Loop |
| CAN | - | Controller Area Network |
| LIN | - | Local Interconnect Network |
| iLLD | - | IFX Low Level Drivers Library |

# Zoznam príloh

Príloha 1. Zdrojový kód pre mikrokontrolér Aurix TriCore TC277D
Príloha 2. Matlab-Simulink súbory pre PIL simuláciu
Príloha 3. Dávkový súbor pre spustenie Bifaces HighTec IDE
Príloha 4. Výsledné vygenerované súbory