User Guide for Vehicle Controls System of McMaster Formula Electric Car

Team 28, Controls Freaks
Abhishek Magdum
Dharak Verma
Jason Surendran
Laura Yang
Derek Paylor

April 5, 2023

Revision History

Date	Version	Notes
April 7, 2023	1.0	Initial Revision

Contents

1 Symbols, Abbreviations and Acronyms **1**

2 Installation **2**

3 Simulink Project Layout **2**
3.1 Library blocks 2

4 Running Simulations **3**
4.1 Unit testing 3
4.2 System testing 4

5 Code Deployment **4**

List of Tables

List of Figures

1 Simulation workflow 3
2 Embedded Coder Tool Location 5

1 Symbols, Abbreviations and Acronyms

See the below table for abbreviations and technical lingo relevant to this project and document.

MFE	Mac Formula Electric
ECU	Electronic Control Unit
BMS	Battery Management System
CAN	Controller Area Network
EV	Electric Vehicle
SAE	Society of Automotive Engineers
PCB	Printed Circuit Board
SoC	State of Charge
TMS	Thermal Monitoring System
AMK	Arnold Müller Kirchheim GmbH; supplier of the vehicle's motor, inverter, controller kit
APPS	Accelerator Pedal Position Sensor
AMS	Accumulator Management System - FSAE lingo, alias for BMS
IMD	Insulation Monitoring Device, installed in the Tractive system.
BSPD	Brake System Plausibility Device, nonprogrammable circuit to check for simultaneous braking and high power output.
PWM	Pulse Width Modulation
G/Gov	Governor module
BM	Battery Monitor module
MI	Motor Interface module
VD	Vehicle Dynamics module

2 Installation

To open the FSAE Vehicle Control System to modify and deploy code, ensure that you download and install MATLAB version R2022b with McMaster's academic license. Ensure that during installation, all packages are installed, as many are required for full Simulink compatibility.

Clone the control system repository to your local drive via the link provided by MFE.

For bench-testing work, you will also need to install Kvaser CAN drivers, as well as the STM32 Nucleo Simulink support package (and all dependencies),

3 Simulink Project Layout

The entire vehicle control system is built within a Simulink project. To familiarize yourselves with the function of each control system module, **read Sections 3 and 6 of the SRS**.

Simulink was chosen for this project, as it allows continuous iteration of our control system for years to come. It features an easy to use interface and built-in workflow to turn model-based logic into usable, robust, and modular embedded code to run on the STM32F7 series of micro-controllers featured on the vehicle.

With your local copy of the project repo, open the project file (.prj). **View the readme on the root level of the project** for a description of the file structure.

3.1 Library blocks

Library blocks are Simulink's way of allowing you to use the same block in multiple instances, where updating the library model updates every instance of the model elsewhere. Our control system takes advantage of this by using the same basic modules in every environment (eg. system testing, unit testing, bench testing).

Library blocks can seem frustrating to work with at first, since they are 'locked' (un-editable) by default, and do not support some diagnostic overlays

that aid in development. But they isolate module implementation changes from each other, and enforce changes to be deliberate.

Some helpful settings when working in this system are toggling 'Show all links' (Debug tab > Information overlays > Show all links). If you are in a library block and need to edit something or inspect it in greater detail, open the corresponding library using ctrl+L.

4 Running Simulations

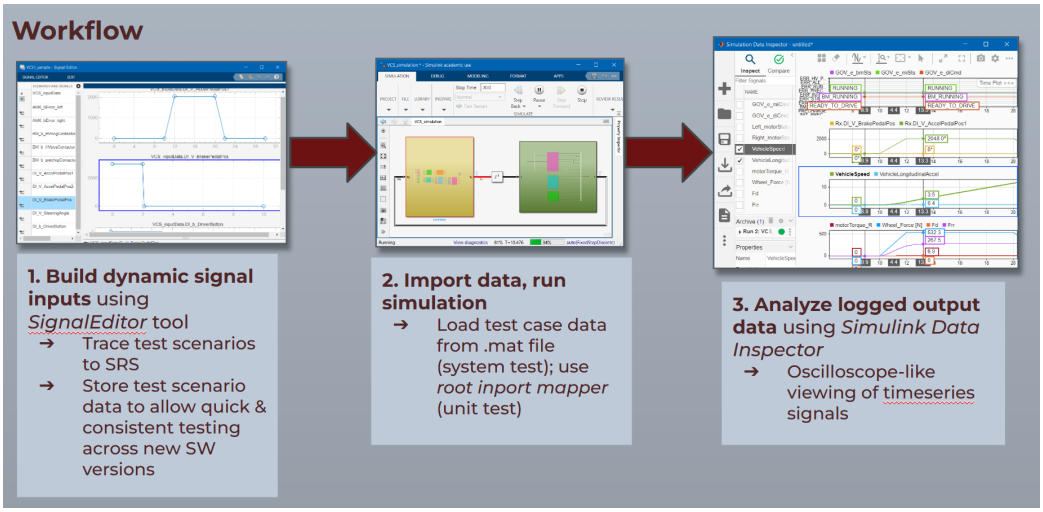

Figure 1: Simulation workflow

4.1 Unit testing

Each subsystem/module has its own unit testing environment. To run a unit test, open the corresponding *.slx Simulink model under the unit testing folder. Double click on any of the input ports to open the input mapper through the "Connect Inputs" button. In the input mapper, load the *.mat file for the respective unit test, and make changes to the input signals through the signal editor. After completing the mapping of input signals, in the Root Inport Mapper window, check map readiness by pressing the green play button. Ensure that all signals are mapped correctly and the scenario will simulate. Once this is finished, return to the Simulink window and press

the green run button. After the simulation has run, open the data inspector to view any of the desired signals. Ensure that all signals you wish to view are being logged.

You can add new logged signals by highlighting them in the .slx model and selecting 'log for simulation', though be careful not to dump too many signals into your Simulink Data Inspector, as it can be hard to find a signal in a big list.

To add a new unit test, copy and paste one of the .mat files, open the Signal Editor (run *signalEditor* in the matlab command line) and edit the signals within the .mat file. Save and rename the new .mat file according to your new test case.

4.2 System testing

System/Integration testing follows a similar workflow to unit testing. Loading test inputs simply involves loading a .mat file. There is no need for root inport mapping. Open the .slx model and run the simulation. View the logged signals in the data inspector.

5 Code Deployment

To deploy code to the front controller STM32F7 series board, click on the tools tab and launch the embedded coder tool. This tool allows users to take Simulink model based code, and convert it to embedded C code to be used on any board that supports real time components like interrupts, which is what is required to run the logic. After launching the tool, hit autogenerate code. This will take the entire model, and generate the relevant C files to run on any STM32F7 series microprocessor. From then on, it is in the scope of the Formula Software Team to deploy the new compiled files onto boards, with their pre-existing workflow.

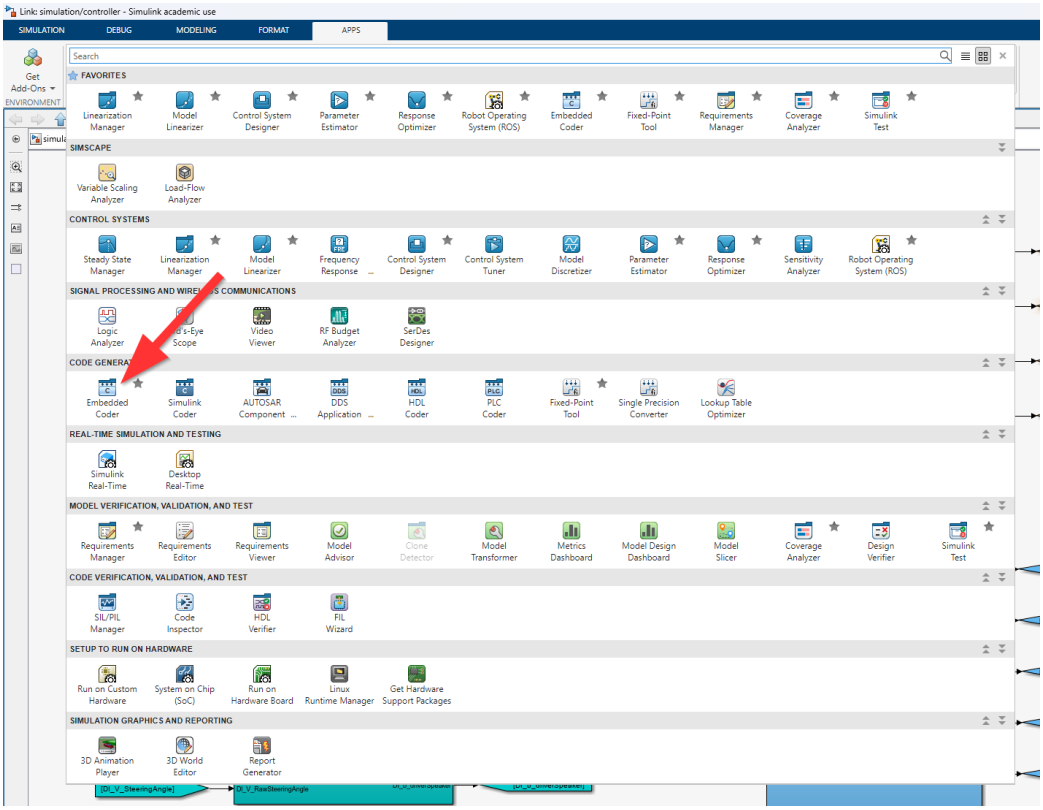

Figure 2: Embedded Coder Tool Location