Verification and Validation Report: Mechatronics

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

March 9, 2023

Revision History

Date	Version	Notes
03/08/23	1.0	

Symbols, Abbreviations and Acronyms

See SRS Documentation here.

Contents

List of Figures

This document provides the results of unit tests and integration tests the system underwent according to the VnV Plan documentation and to ensure requirements layed out in the SRS are met and traceable to tests performed.

1 Functional Requirements Evaluation

Based on the results in Sections 3 & 4, our control system meets basic functionality requirements. This VnV Report has prompted the team to generate a more comprehensive selection of requirements and tests, on the unit and system level, which will be reflected in the final documentation.

2 Nonfunctional Requirements Evaluation

All non-functional requirements, although subjecive in certain cases, have been met to some degree.

2.1 Look and Feel

NFR1 - (SRS: The control system Simulink models shall follow a standardized format that facilitates readability and encapsulation)

The control models followed the following variable naming scheme:
OriginNode_QuantityType_VariableDescription

Example: DI_p_BrakePedalPosition

- OriginNode - Driver Interface
- QuantityType - 'p' denotes percentage (0-100). 'V' is used for voltage, 'T' for a torque value, etc.

Example: VD_n_LeftMotorSpeedRequest

- OriginNode - Vehicle Dynamics
- QuantityType - 'n' denotes speed

This naming scheme made development, debugging, and maintaining the model much easier. Since finding the exact reference to a signal can be cumbersome in Simulink, our naming scheme greatly decreased the time and effort required to understand a model or set of blocks. On top of that, when the control system was converted to C code, the naming scheme was a great aid in integrating the relevant input and output variables into the embedded C layer, as we knew exactly where the signal came from, where it was going, and what its purpose was.

2.2 Usability

NFR2 - (SRS: The control system build target shall be the team-selected microcontroller platform, STM32F7)

The Simulink model was successfully converted into C code using the Simulink code generation toolkit. The autogenerated code was then integrated into our embedded C wrapped and successfully flashed onto MFE's STM32F767ZI Front Controller ECU. The Front Controller was able to step through the control system at its required periodicity, supply all required inputs via CAN, SPI, UART, and I2C communication protocols, and successfully take the control system outputs and propagate them to the motor controllers.

2.3 Performance

NFR3 - (SRS: Running the control system on the embedded controller shall not exceed the computational throughput of the controller.)

After utilizing the C Code Generation tool in Simulink, we were prompted to run the "step" function for the model at a certain periodicity. This periodicity, which Simulink had calculated internally and requested us to set, was the speed at which the control system would be refreshed and updated with new inputs and outputs. For our model, Simulink deemed that it must be refreshed a minimum of every 200ms for the C code to function exactly as the Simulink models did during our testing. This 200ms scan periodicity is used by the Simulink-generated C code to control timers, counters, delays, etc. (anything time-based), and any deviations from the 200ms periodicity of the step function would break the internal timing logic of the system.

Due to the above reasons, we decided it was wise to avoid messing with the scan rate of our control system and instead left it up to Simulink's discretion.

2.4 Maintainability and Support

NFR4 - (SRS: The control system shall make use of hardware abstraction modules, that have consistent outputs to our system regardless of changing the underlying hardware)

Hardware hiding in our build-model has been achieved by using our CAN hardware supplier's Simulink blockset, which allows our models to write signals to the vehicle's CAN bus simply by sending it to another block. The underlying implementation of the CAN transmission and reception is unknown to the control system. Similarly, the potentometer inputs (pedals, steering) have been hidden under the Formula team's embedded C GPIO layer.

NFR5 - (SRS: The control system modules shall incorporate modularization principles, including encapsulation and information hiding.)

The control system's architecture, as explained in Design Documents, has been developed from the start as a modularized system, with each subsystem handling an unique category of functionality in which the implementation is unknown to the other modules.

2.5 Security and Support

The non-functional requirements listed below are self evident and do not require formalized verification.

- NFR7 - (The control system repository shall be public for the duration of the capstone project.)
- NFR8 - (The control system repository shall be made private upon completion of the project.)

3 Unit Testing

Unit testing was carried out by creating single-module environments in which module inputs are supplied via manually-created timeseries signals (created via Simulink's Root-level importer and Signal Editor), and outputs are logged during simulation for inspection with a scope-like interface (Simulink Data Inspector). These environments are shown for each module below.

3.1 Driver Interface Module

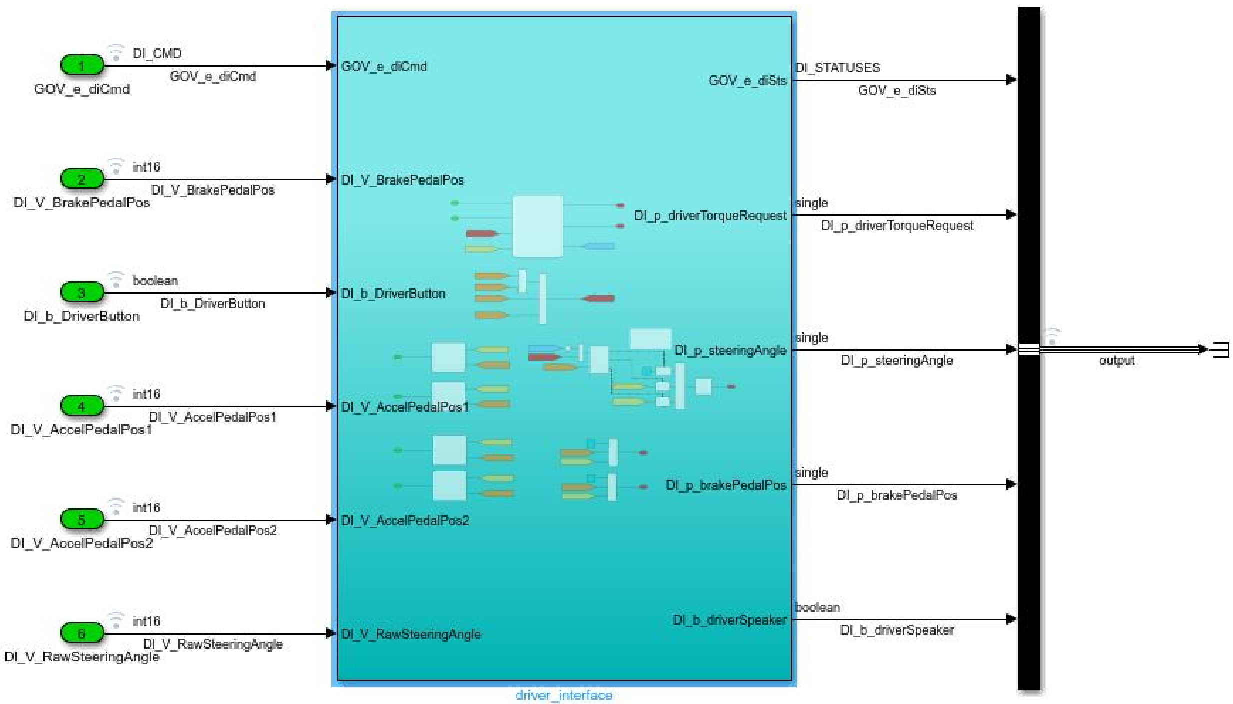

Figure 1: Driver Interface - Unit Test Simulation Environment

TID	Requirement	Test Input	Expected Result	Actual Result	Result
DI1.1	FR-CRIT-FAIL-VDR1 (VnV Plan rev0)	APPS1 Pot signal > 4096	APPS1 internal fault is set; APPS2 is used for torque request	APPS1 internal fault is set; APPS2 is used for torque request	Pass

TID	Requirement	Test Input	Expected Result	Actual Result	Result
DI1.2	FR-CRIT-FAIL-VDR1 (VnV Plan rev0)	APPS2 Pot signal > 4096	APPS2 internal fault is set; APPS1 is used for torque request	APPS2 internal fault is set; APPS1 is used for torque request	Pass

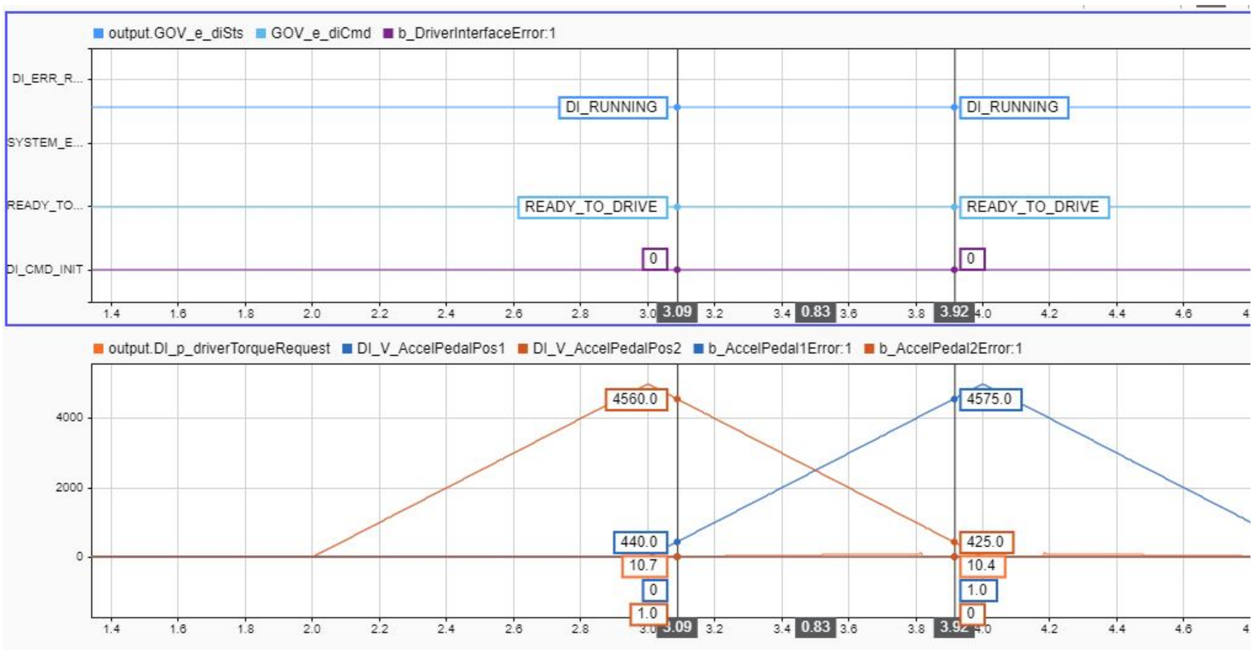

Figure 2: DI1.1 and DI1.2 Simulation Results

TID	Requirement	Test Input	Expected Result	Actual Result	Result
DI1.3	FR-CRIT-FAIL-VDR1 (VnV Plan rev0)	BPPS Pot signal > 4096	Internal fault is set; Driver Interface reports Error status	Internal fault is set; Driver Interface reports Error status	Pass

TID	Requirement	Test Input	Expected Result	Actual Result	Result
DI1.4	FR-CRIT-FAIL-VDR1 (VnV Plan rev0)	Steering Pot signal > 4096	Steering internal fault is set; Driver Interface reports Error status	Steering internal fault is set; Driver Interface reports Error status	Pass

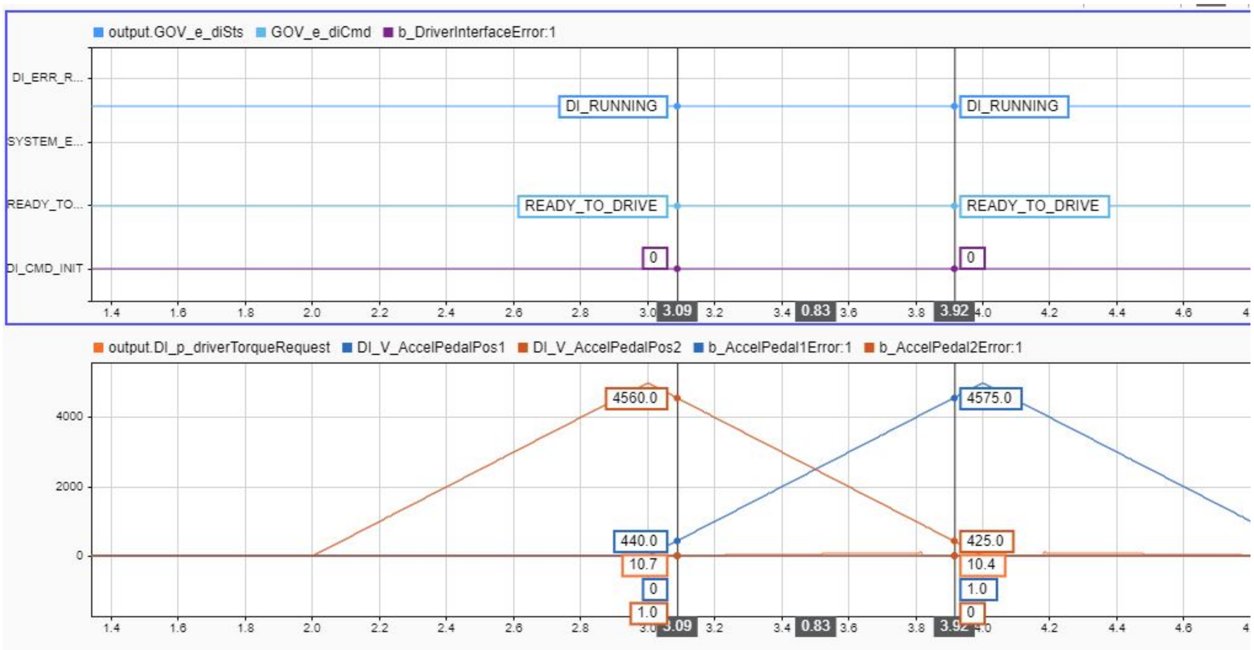

Figure 3: DI1.3 and DI1.4 Simulation Results

3.2 Vehicle Dynamics Module

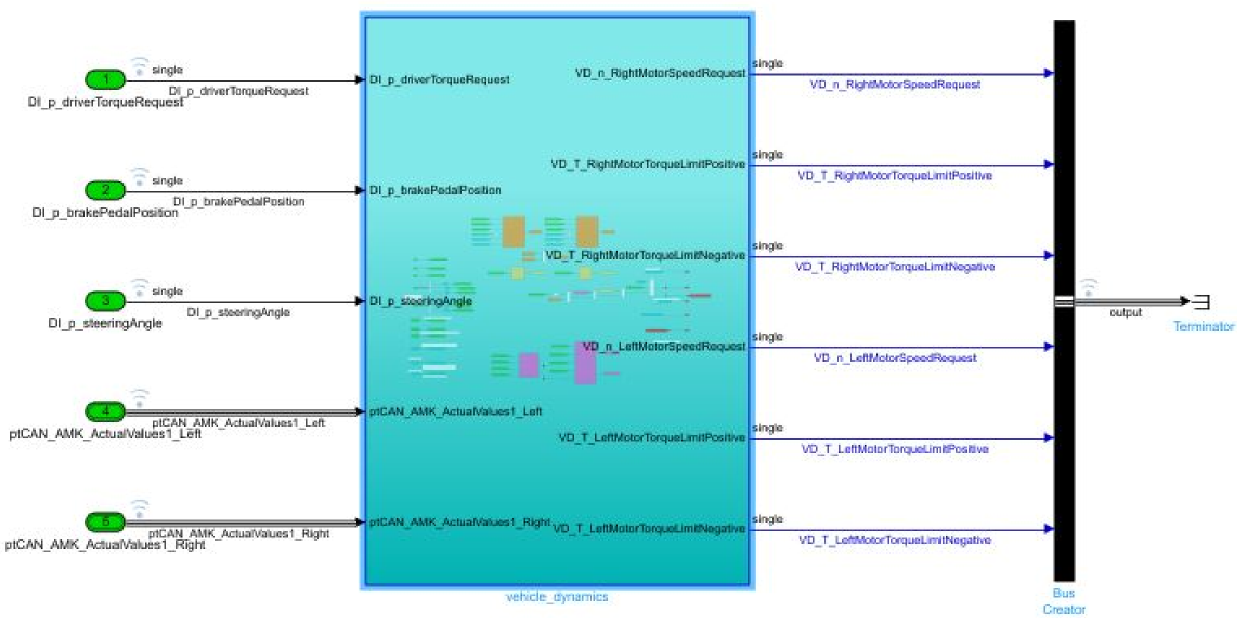

Figure 4: Vehicle Dynamics - Unit Test Simulation Environment

TID	Requirement	Test Input	Expected Result	Actual Result	Result
VD1	FR-WARN-VDR1 (VnV Plan rev0)	Torque Request & Brake Pedal Position > 0	Motor positive torque limits = 0	Motor positive torque limits = 0	Pass

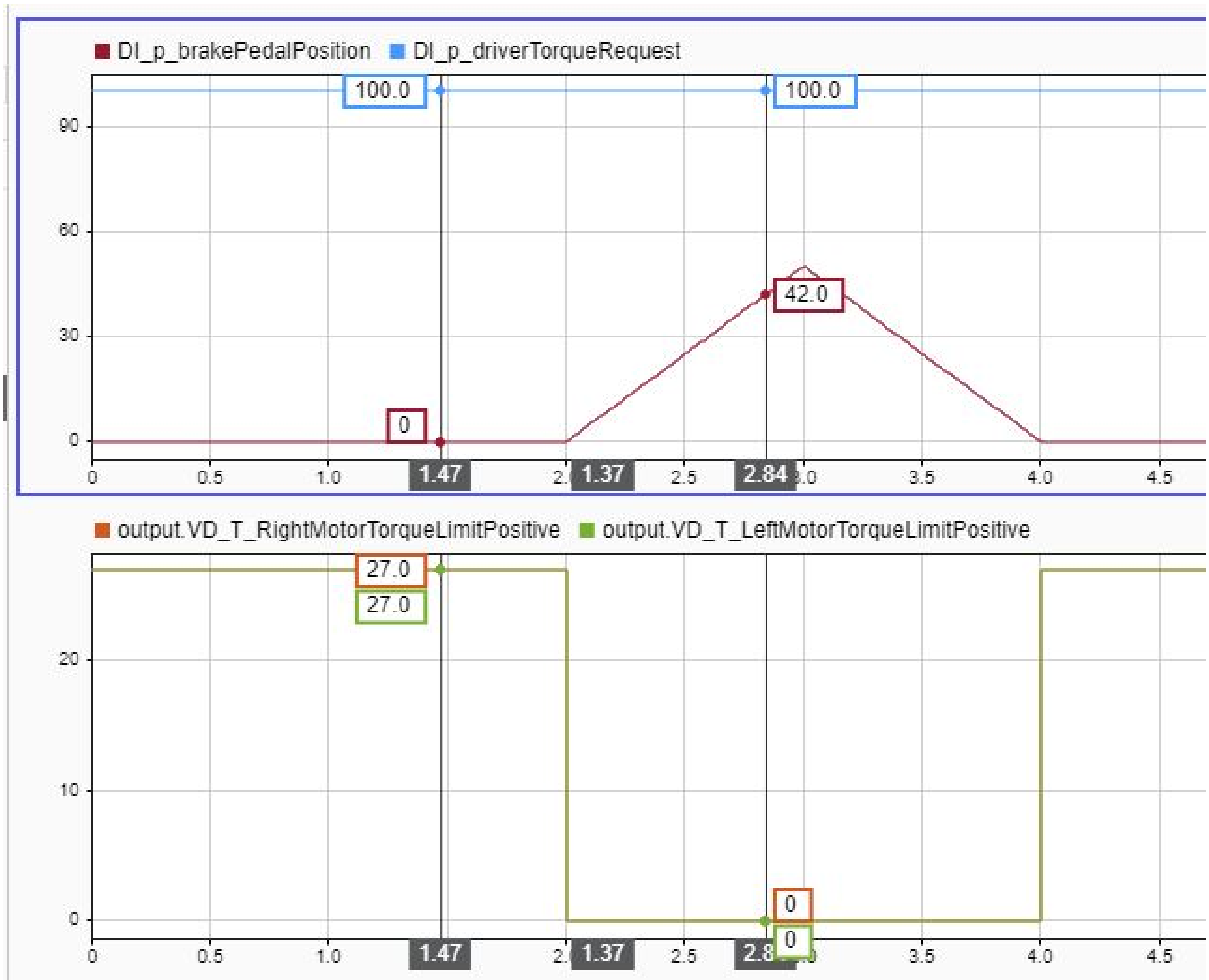

Figure 5: VD1 Simulation Results

3.3 Motor Interface Module

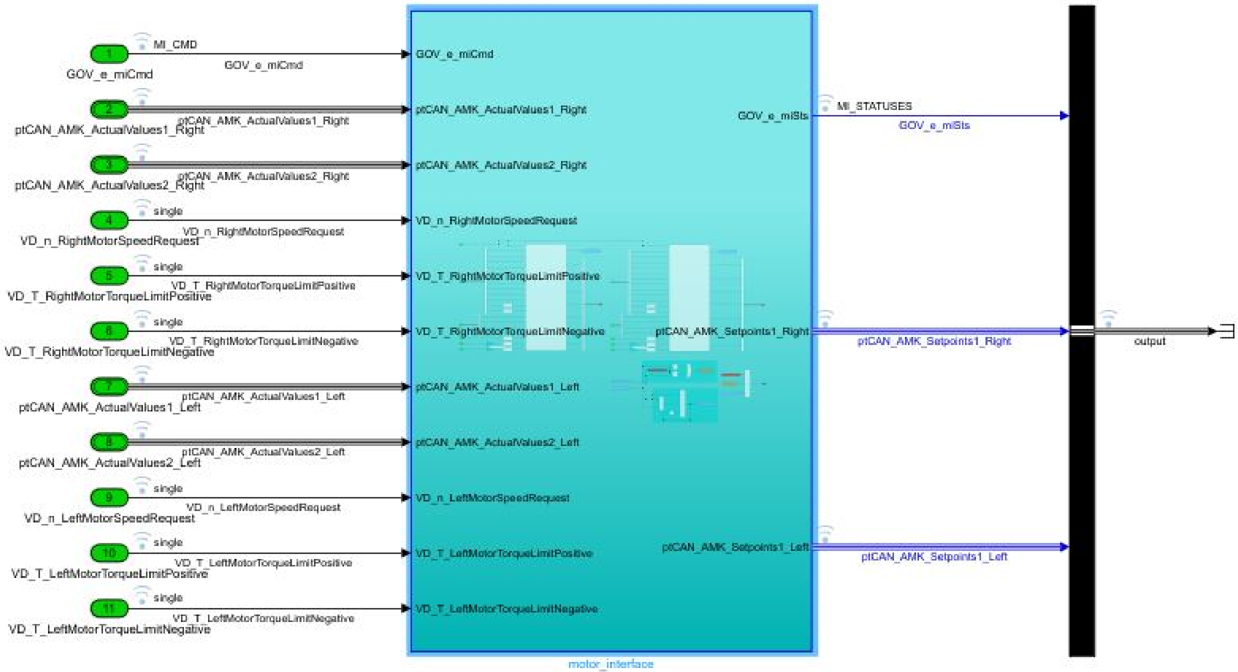

Figure 6: Motor Interface - Unit Test Simulation Environment

TID	Requirement	Test Input	Expected Result	Actual Result	Result
MI1	TMR1 TMR2 TMR3 TMR4 (SRS rev0)	AMK startup feedback signals (AMK documentation 8.2.6)	Control signals sent according to AMK 8.2.6; Motor interface reports 'Running' state	Control signals sent according to AMK 8.2.6; Motor interface reports 'Running' state	Pass

Note: TMR6 and TMR7 from the SRS are validated under test case VD1.

3.4 Battery Monitor Module

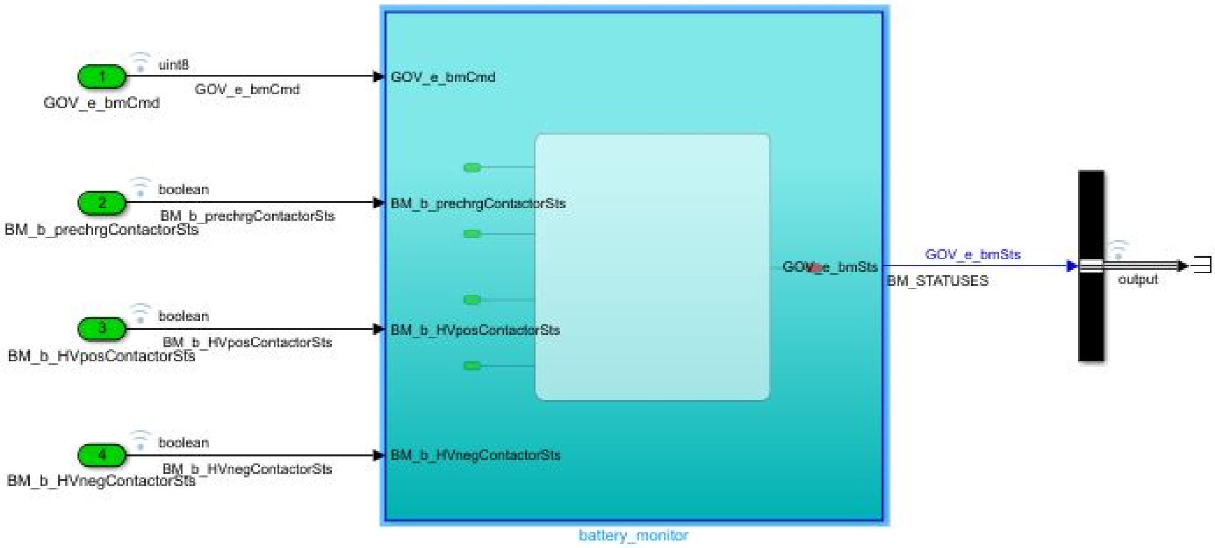

Figure 7: Battery Monitor - Unit Test Simulation Environment

TID	Requirement	Test Input	Expected Result	Actual Result	Result
BM1	FR-CRIT-FAIL-AMR1 (VnV Plan rev0)	Contactors: HV positive = 0; HV negative = 0; precharge = 1	Battery Monitor reports Error state on startup	Battery Monitor reports Error state on startup	Pass

Note: AMR2-9 are no longer applicable due to scope reduction; the Formula team now intends to use an off-the-shelf BMS, so a full battery management system is no longer required for this system.

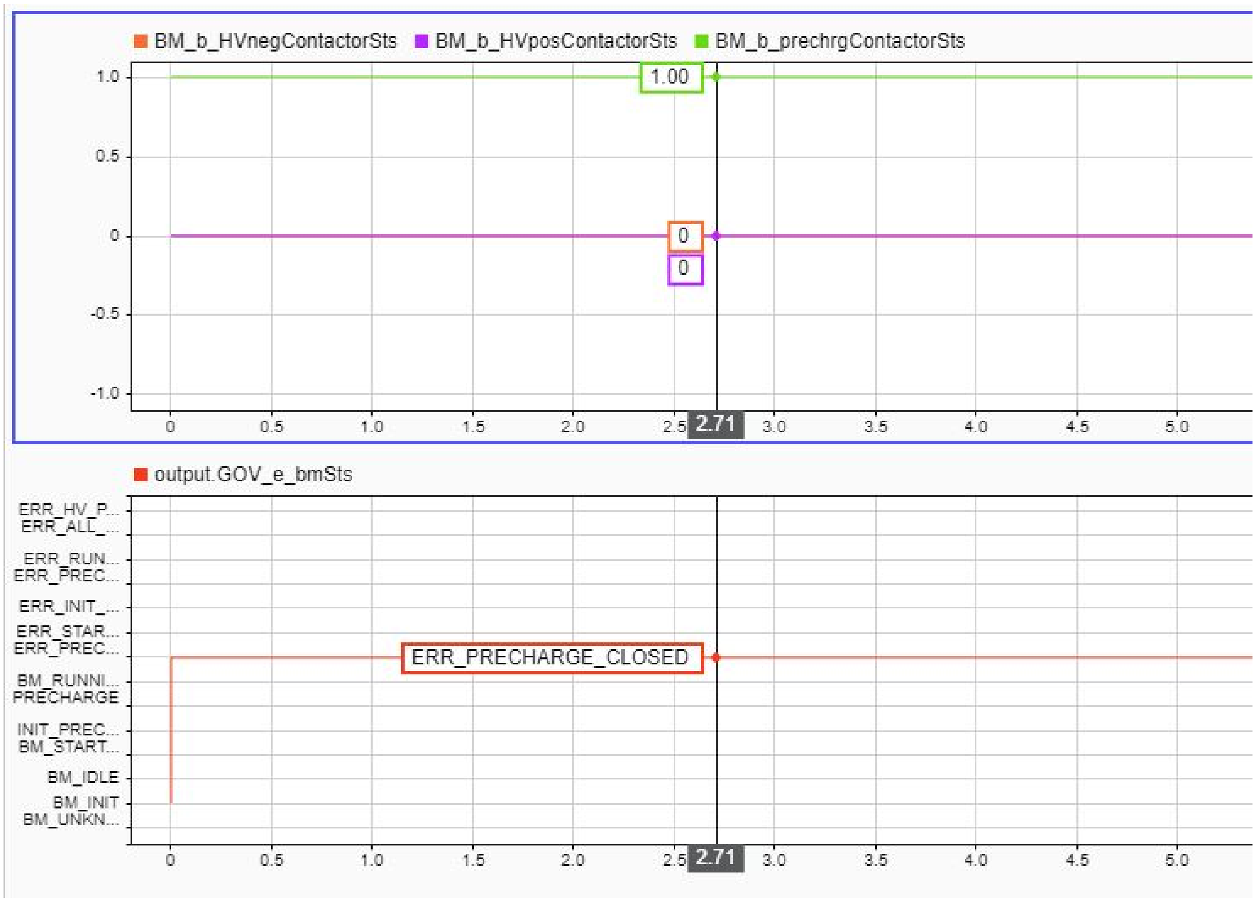

Figure 8: BM1 Simulation Results

4 Integration Testing

Shown below is the simulation environment for our control system. The Controller model (on the left), containing all the subsystems (tested in Section 3), all running simultaneously and supplying signals to a Plant model (on the right). The Plant model was designed by our team to simulate feedback from the vehicle's sensors and other controllers. AMK motor feedback, vehicle speed and acceleration are simulated using a physics model. Driver Inputs and Battery Contactor states are modelled simply by manually-created timeseries signals.

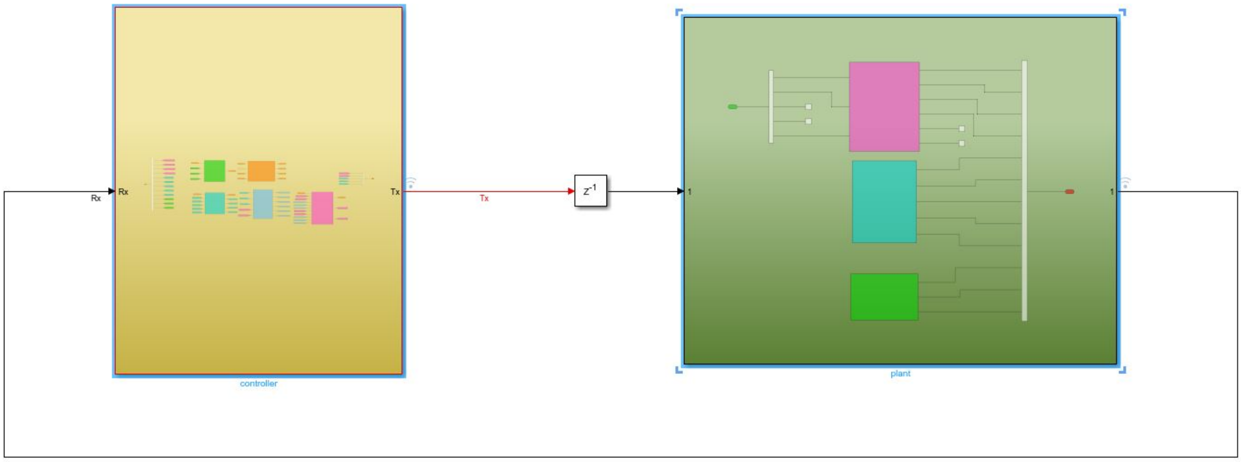

Figure 9: Vehicle Control System - System Test Simulation Environment

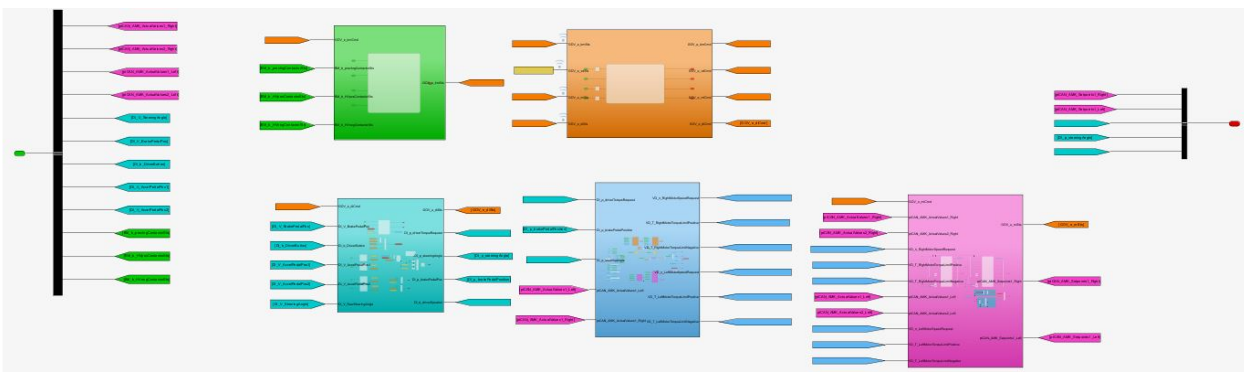

Figure 10: Vehicle Control System - Controller model

TID	Requirement	Test Input	Expected Result	Actual Result	Result
VCS1	MSR1 MSR2 MSR5 (SRS rev0)	Contactors in expected startup sequence; Driver inputs prompt vehicle start (button and brakes engaged), followed by throttle inputs	Governor issues ready-to-drive once BM, DI and MI report 'running' state; vehicle responds to throttle input	Governor issues ready-to-drive once BM, DI and MI report 'running' state; vehicle responds to throttle inputs	Pass

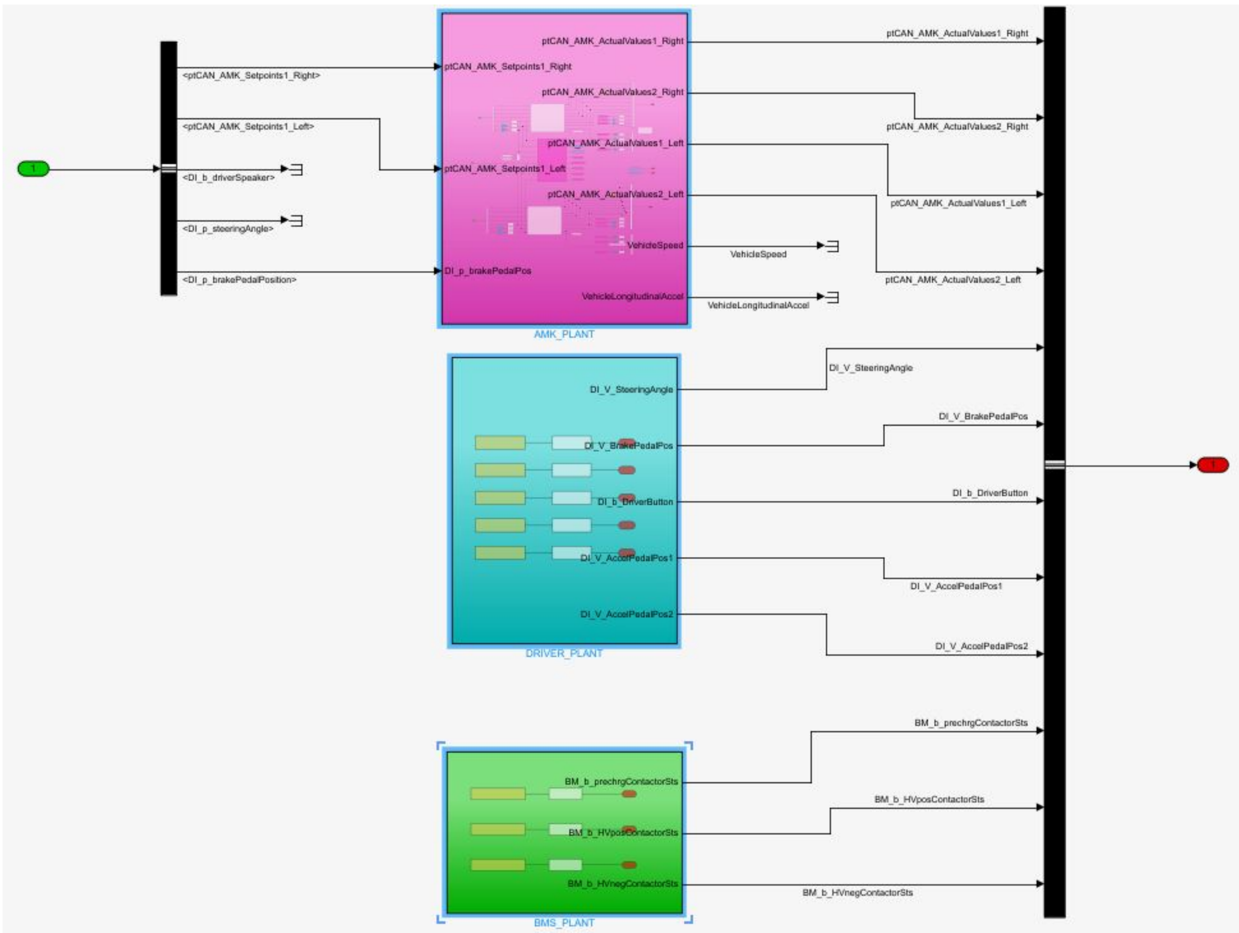

Figure 11: Vehicle Control System - Plant model

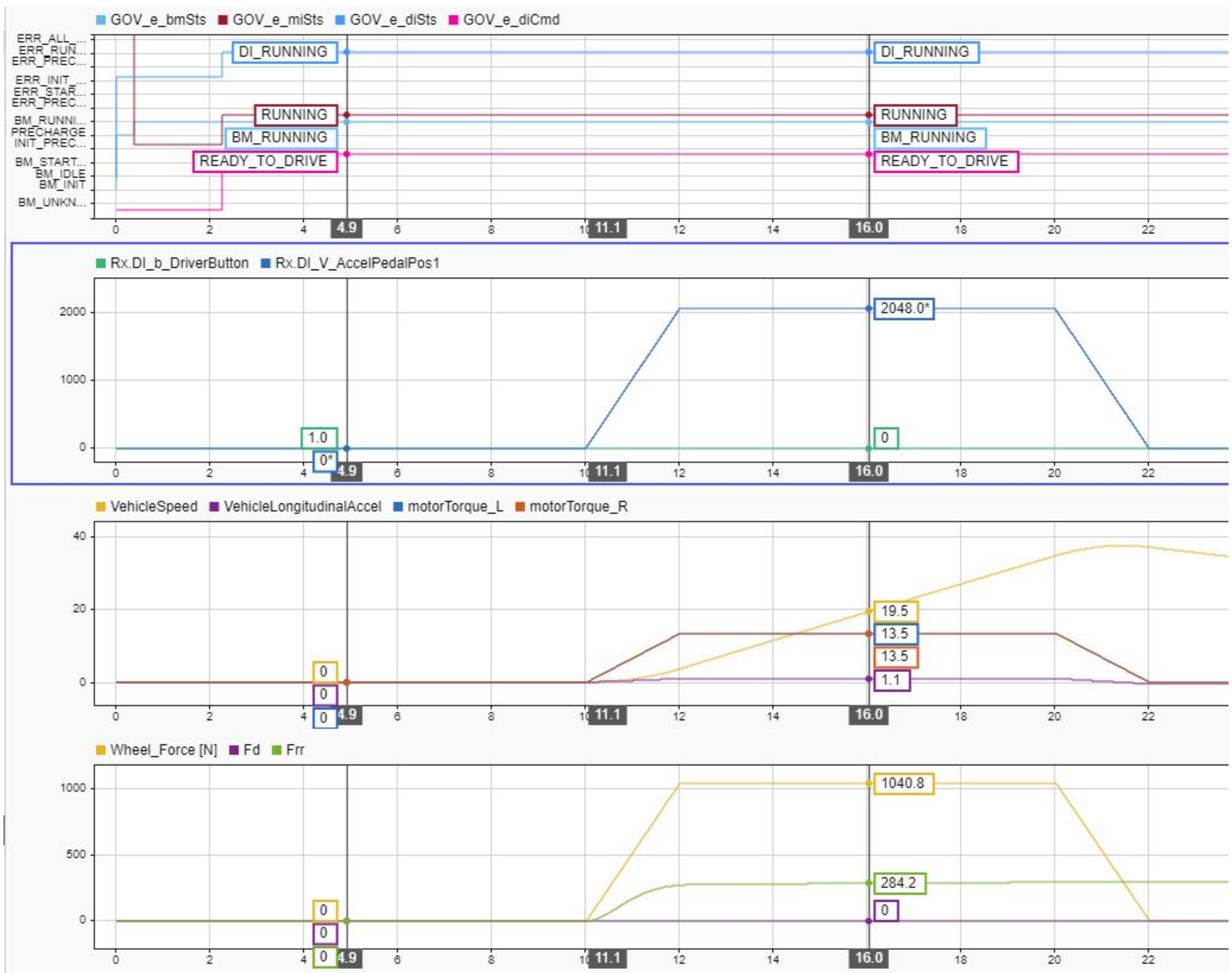

Figure 12: VCS1 Simulation Results

5 Changes Due to Testing

The process of validation (as well as instructor questioning during the PoC and rev0 demos) has highlighted the need to add control logic to handle various edge cases, such as errors in driver input sensors. This was implemented in the Driver Interface module, in the form of sub-modules which monitor each potentiometer reading for error states. When the Driver Interface detects any such errors, this error state is reported to the rest of the control system, and the motor torque request is set to 0 for the vehicle to coast down.

6 Automated Testing

It was not in the project scope or timeline to develop automated test tools for model-based development. Testing was carried out by building manual simulation environments as described in the VnV Plan.

7 Trace to Requirements

Corresponding requirements are mentioned in-line with validation summaries throughout Sections 2, 3 and 4.

8 Code Coverage Metrics

The concept of code coverage is not applicable in model-based design.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection. Please answer the following question:

1. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)

During activities conducted for VnV, it was decided that a simplification in architecture was required. This was due to having an off the shelf BMS being used in our vehicle, which negated the need to have an entire battery monitoring block. Since we are only monitoring the state of contactors to ensure we are in a safe running state, we decided to make our entire vehicle control system's logic only revolve around vehicle propulsion. Anything regarding vehicle dynamics will be tested, and any battery tests can be neglected.