Model Based Design Framework Development of a Hybrid Supervisory Controller for a P4 Parallel Hybrid Vehicle

by

Asad Bhatti

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
In
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The expected rise in the number of ECUs in an automotive based development environment, poses additional efficiency risk on developer time and code complexity. This thesis examines the design and validation of a Hybrid Supervisory Controller, developed for the University of Waterloo Alternative Fuels Team's (UWAFT) retrofitted P4 parallel Chevrolet Blazer, in the EcoCAR Mobility Challenge competition.

The controller, component models and I/O interaction layers are developed in a MathWorks Simulink environment. The framework discussed, is built to incorporate automation via a custom developed -*Model-Configurator* tool. Component models, and functional sub-systems are converted to masked library blocks within Simulink, that are populated via an object-oriented class in the MATLAB environment. This opens the possibility for custom environment data population, swapping of data for models while retaining underlying physics and setting up for SIL/HIL requirements testing without explicit/contemporary interaction with the Simulink environment. The advantages of this approach are discussed, along with explanation accompanying the software framework.

The HSC incorporates interaction models of 9 stock vehicle, and on-board GM ECUs. The model spans full chassis longitudinal, and powertrain components. The functional controller incorporates 4 powertrain control layers - fault detection, vehicle state control, torque strategy and component level execution layers. The test environment switching time is reduced by >50%, and 86 controls requirements are tested over the course of 3 years.

The test vehicle is tested at the Canadian Technical Center McLaughlin Advanced Technology Track (CTC MATT) where a non-standard drive cycle is used due to limitations posed by the COVID-19 pandemic. The vehicle robustly sustains a 91-minute city/highway drive, with a 24% improvement in fuel economy compared to stock. The vehicle however is short of its VTS targets which are attributed to the lack of engine start/stop functionality, and a thermally constrained battery pack. Those remain major design shortcomings and immediate powertrain improvements are proposed, and efficacy of a well-organized model are discussed.

.

Acknowledgements

My journey as a graduate student is a culmination of the endearing support that my family, friends, colleagues, and mentors have imparted over the last two years. I want to first congratulate the UWAFT team and Dr. Roydon Fraser for achieving the 25-year milestone in team history, I am glad to have played a role in making it memorable! At the same time, I want to thank Dr. Roydon Fraser, Dr. William Melek, Dr. Michael Fowler and Denise Porter for their enthusiastic guidance throughout my graduate studies. I would also like to thank Bronson Patychuk, my General Motors mentor – who kept me encouraged, and never hesitated to impart critical advice through the course of time as a Project Manager at the team. Finally, my sincerest gratitude goes out to all teammates, colleagues, and coops. I've had the pleasure of working with throughout my journey.

Abdul Haseeb	Jake McGrory	Paul Boctor
Aidan O'Gorman	Jaskaran Basson	Peter Alexis
Alex Matos	Jason Nguyen	Ryan Tanary
Amanda Forwell	Jerica Rattana	Sophy Li
Bade Akinsanya	Jillian Wong	Sumeet Kler
Cole Tofflemire	Joanna Diao	Teodor Crnobrnja
Daniel Montazeri	John Webster	Timothy Er
David Hernandez	Julian Lowery	Urban Pistek
Derrick Tan	Kevin Liao	Vaibhav Patel
Dinil Karunaratne	Kevin Rao	Vatsalya Saini
Dhruv Manani	Liam Cain	Winnie Wu
Ellin Liu	Lucille Huang	Yiwei Wang
Geoffrey Qin	Mahad Zaryab	Yulian Bagnel
Haider Mirza	Naim Suleman	Xiry Dong
Haocheng Zhang	Omar Shakir	Zhen Ye

It would be prudent of me to be exclusive of the friendship, comradery, and good times my friends Paul Boctor and Timothy Er have had, for having me in their lives. A bond that as Tim likes to say, "goes either way", that I whole heartedly believe will continue for many years to come. Lastly, I would like to thank Peter Teertstra, and Graeme Adair who's support and tolerance was instrumental in keeping UWAFT's doors open, throughout the course of the COVID-19 pandemic.

Dedication

To,

My stars above parents, Saira and Mahmood

My wonderful sister Amna

My lovely grandma Rifat

استاد محترم Fraser, Asif, Tahir, Mahar, Fahim, Imran, Atta and Keshwar *(among others)*

My brother Abuzar

Z.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AAM	American Axle Manufacturing
APP	Accelerator Pedal Position
AVTC	Advanced Vehicle Technology Competition
AWD	All-Wheel Drive
BEV	Battery Electric Vehicle
BMS	Battery Management System
BSFC	Brake Specific Fuel Consumption
BPP	Brake Pedal Position
BTE	Brake Pedal Position
CAN	Controller Area Network
CAV	Connected and Automated Vehicle
CI	Compression Ignition
CS	Charge Sustaining
CTC MATT	Canadian Technical Center McLaughlin Advanced Technology Track
DOE	Department of Energy
DP	Dynamic Programming
E&EC	EcoCAR Emissions & Energy Consumption
ECMS	Equivalent Consumption Minimization Strategy
ECU	Electronic Control Module
EMC	EcoCAR Mobility Challenge
ESS	Energy Storage System
EV	Electric Vehicle
FSC	Functional Supervisory Controller
GHG	Green House Gases
GM	General Motors
HC	Hydrocarbon
HDS	Hybrid Design Services
HV	High Voltage
HVIL	High Voltage Interlock Loop
ICE	Internal Combustion Engine
MAAS	Mobility As A Service
MABx	MicroAutoBox
MBD	Model Based Design
NOx	Nitrogenous Oxides
NVH	Noise Vibration and Harshness
NYSR	Non-Year Specific Rules
OEM	Original Equipment Manufactures
PHEV	Plug in Hybrid Electric Vehicles
PMSM	Permanent Magnet Synchronous Motor
RTM	Requirements Trace-ability Matrix
SI	Spark Ignition
SIL	Software In Loop
SOC	State of Charge
VIL	Vehicle In Loop

VSM	Vehicle State Machine
VTS	Vehicle Technical Specifications
WEF	World Economic Forum
WOT	Wide Open Throttle

Chapter 1

Introduction & Background

1.1 Introduction

The World Economic Forum (WEF) estimates that the world by 2040 compared to 2015, will see the total number of cars and trucks to grow by two folds [1]. This when paired with the statistics published by the U.S. Energy Information Administration annual energy report for 2021, estimates that the up to 25% of the world's energy is spent in the transportation of people and goods [2]. With over 65% of the energy being sourced directly from petroleum and natural gas alone, environmental concerns are on the rise with the energy consumption trends of the automobile.

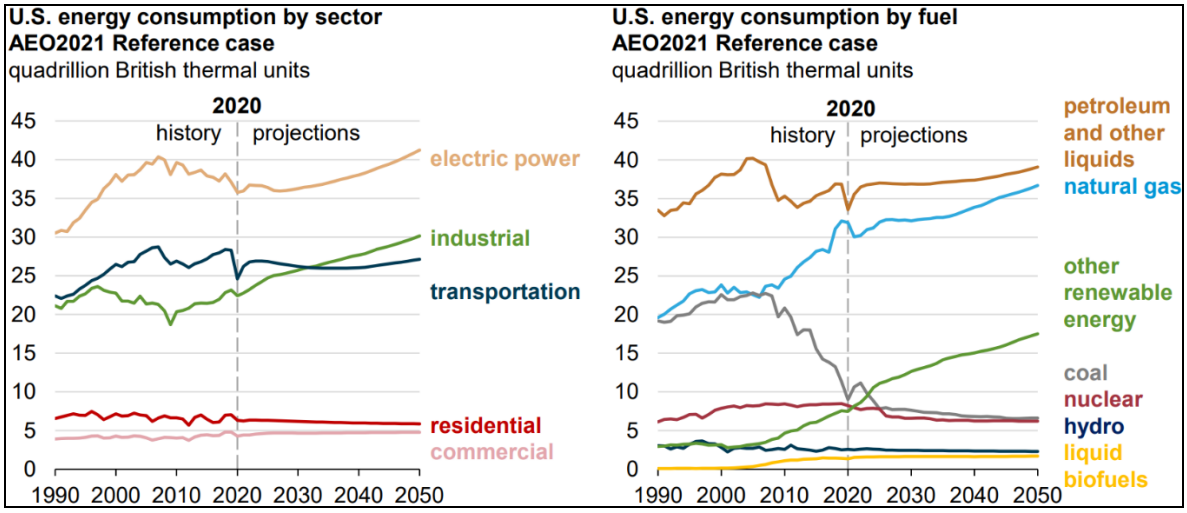


Figure 1: Total energy consumption by end-use sector [2]

Production of Green Houses Gases (GHG) can be broken down into various forms of pollutants including but not limited to CO & CO2 which are produced directly because of burning hydrocarbons. Hydrocarbon (HC) emissions lead to environmental degradation in both air and sea water. Symptoms include smog, rising sea levels and a reduction in ocean bio-diversity due to rise in temperatures. Nitrogenous oxides (NO_x) are formed when combustion occurs at high enough temperatures and pressures. NO_x compounds directly contribute in the depletion of the ozone layer. Increase in urban expansion, and industrialization is only going to

add to the contribution of GHGs as purchasing of locomotives that burn hydrocarbons increases, if other possible propulsive modes are not explored.

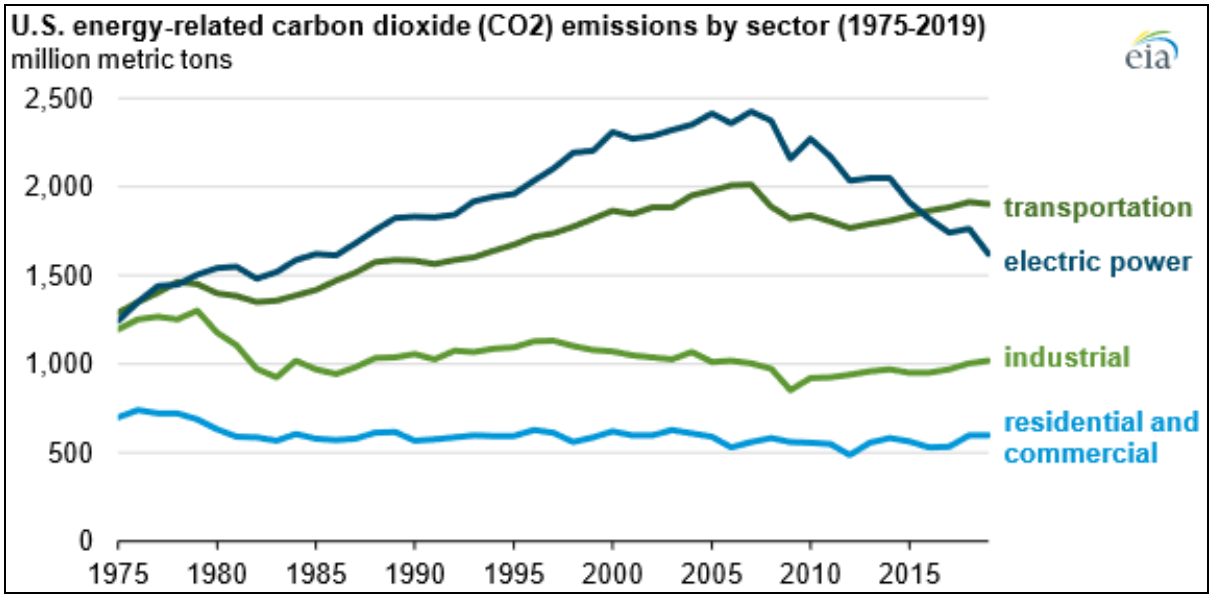


Figure 2: Global primary energy demand and energy-related CO2 emission, 1975-2019 [3]

Hybrid Electric Vehicles (HEV)s represent a steadily increasing portion of the Electric Vehicle (EV) market. The demand for global annual passenger car and light-duty hybrid, or electric vehicle sales is projected to be around 25% by the year 2030 [4]. In the pursuit of complete transition to EVs. There lie significant infrastructural, design, and political challenges, that warrant a focus on short term problem solving to not only expose the consumer market to the pros and cons of EVs, but to reap the benefits of the already existing electrification technologies. Especially in up and coming Asian, European, and African markets where adoption of full EVs is not yet feasible.

While Battery Electric Vehicles (BEV)s lead the charge in global electric automobile shares. Chinese, US, and European markets are all warming up to the idea of owning an electric vehicle but may not be ready to boot for the cost and "range anxiety" owing to a rising trend in sales of Plug-in Hybrid Electric Vehicles (PHEV)s and their less complex HEVs derivative.

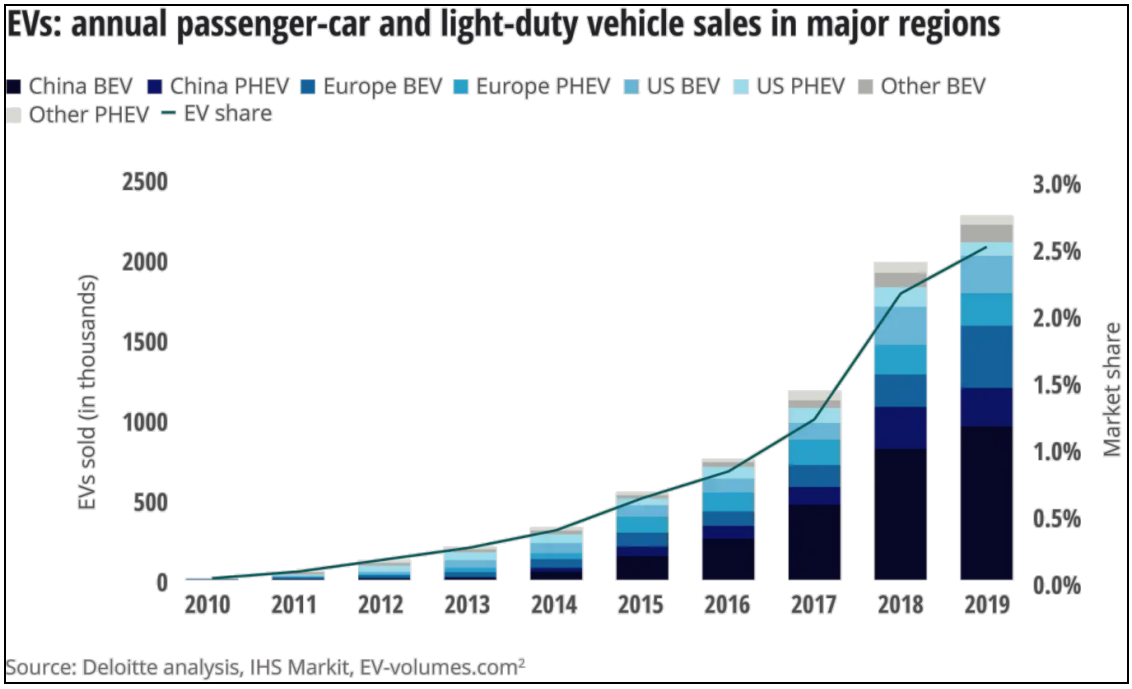


Figure 3: HEVs, PHEVs and EVs share consistent market growth [4]

Predominantly hybrid architectures utilize one or more electric motors, as the added mode of propulsive torque, in addition to the commonly used Internal Combustion Engine (ICE) for the purposes of hybridization. Arrangements of the hybridization i.e., the placement of the electric motor around the ICE dictates the viability of various hybridization architectures for instance series or parallel hybrids [5]. Moreover, the size of Energy Storage System (ESS) can dictate whether a hybrid vehicle is a PHEV or an HEV. PHEVs allow for a further reduction in GHGs emissions due to the ability to plug in to a charge network, allowing for sustained electric driving only, contributing to an overall more *efficient* drivetrain.

At the consumer level, the addition of the external ESS allows for the possibility to incorporate both fuel efficiency and drive quality-oriented features. These include limited range fully electric driving for daily commutes; higher vehicle acceleration control, regenerative braking, electric motor assist and possible reduction in size and power of the ICE [6]. Performance of the HEV hybridization is highly dependent on not only the efficiency of the added powertrain components, but also around the understanding developed around the use case of said architecture. For instance, the drive schedule in question. Increased utilization of the ESS is desirable for an HEV or PHEV, however for the ICE and electric motor to work

seamlessly a higher degree of control is needed over the propulsive units as well as the overall utilization of energy on-board the vehicle to result in the most energy efficient of outcomes.

1.2 Background

A significant engineering challenge in any an HEV is the optimal control of mechanical and electrical flow of power through the ICE, electric motor, and the various conversion/reduction devices. Typically, the added degrees of freedom give way to flexibility in driving modes, better utilization of the torque application between the ICE and electric motor. Resulting in reduction of GHG emissions and increase of the overall fuel economy [7]. Preservation of drive quality in terms of vehicle acceleration profile is another important aspect of the implemented torque management scheme [8].

This necessitates devising of a Hybrid Supervisory Controller (HSC), that interfaces with all vehicle level components' external Electronic Control Unit (ECU)s; performs state estimation; handles I/O; performs computation of the vehicle's torque strategy and executes on operating points for the component. The vehicle torque strategy is essentially a regimented series of rules that regulates the operation of the ICE and electric motor. This normally comprises of driver inputs in the form of accelerator pedal, vehicle level measurements such as speed, battery State of Charge (SOC) from the ESS, component operating conditions such as temperature, to output operating points for the propulsive systems or simply turn them On/Off based on the driving schedule.

The software development of University of Waterloo Alternative Fuel Team's (UWAFT) HSC follows a requirements-based software development process. This process is based on the Model Based Design (MBD) design process used for the creation, testing, and verification of software [9]. The requirements are developed at multiple levels and correspond to testing at their levels depending on the nature of the requirement. These can be as high level as a Vehicle Technical Specification (VTS) such 0-60 mph time or as low level as functional safety of maximum electric motor speed in rads/s. Due to reusability of the implemented models, the V-process is used such that the requirements can be frequently revised, and the development process is made iterative, based on the learning outcomes from the test results.

UWAFT is participating in the General Motors (GM) and U.S. Departments of Energy's (DOE) push to sustainable means of propulsions and advancement of electrification through the 4-year long EcoCAR Mobility Challenge (EMC) program [10]. This Advanced Vehicle Technology Competition (AVTC) is one of many in its 30+ years of history. Started in 2018 EMC is pushing the frontiers of transport in *electric utility* by providing a competitive landscape for 12 North American schools the support in hardware and training to produce more eco-friendly, SAE Level 2 autonomous enabled customer centric vehicles for the Mobility As A Service (MAAS) market. It is through the EMC's provided vehicle research platform, technical training of software and hardware; and the industry level sponsorships that enabled the development of the research content for this thesis. This year marks 25 years in the team's history of developing advanced technologies for advanced vehicles.

UWAFT's architecture of choice is the P4 Parallel through the road hybrid, shown in Figure 4. The front axle is powered using a 148 kW 2.5L inline GM LCV inline 4 that is mated to a GM 9-speed M3D transmission. The rear axle is driven by a 150 kW American Axle Manufacturing (AAM) electric EDU4 electric motor that is powered by a Semikron SKAI2HV inverter and a 360V 5.5 kWh (total) 1.5 kWh (nominal) Hybrid Design Services (HDS) Li-Ion 96S8P battery pack.

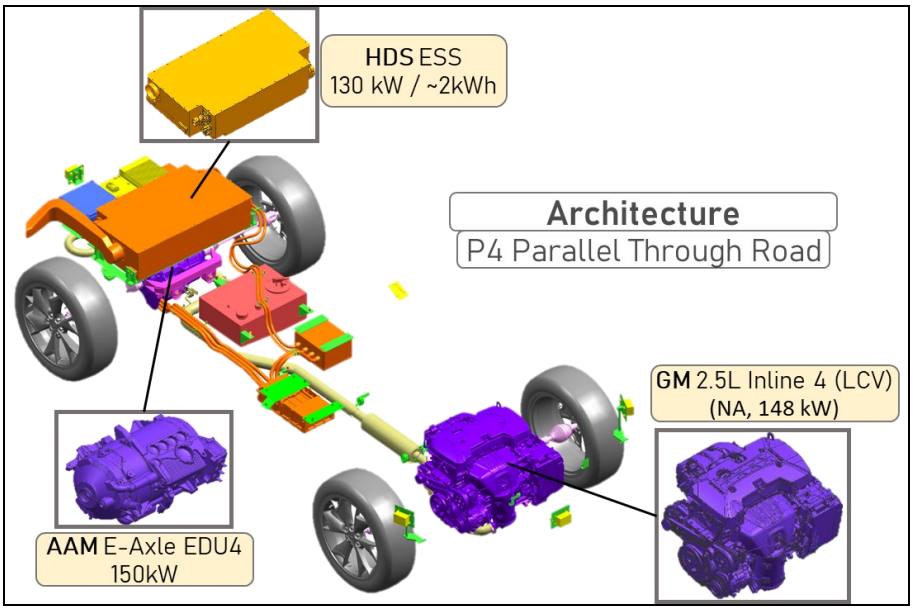


Figure 4: Main Powertrain Components of UWAFT Blazer

A P4 parallel through the road architecture connects the front axle and rear axle through the road, allowing for independent operating modes, reducing overall integration complexity, and allowing for a better front to rear dynamical vehicle weight distribution. The architecture allows for three main operating modes. ICE only, Parallel and EV only modes.

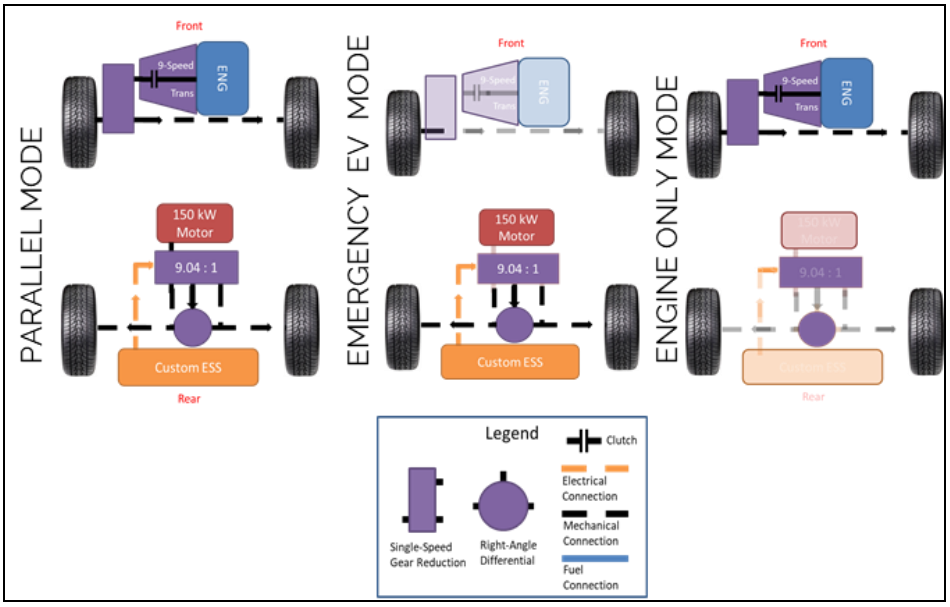


Figure 5: Operating Modes for the P4 parallel through road architecture

ICE only made is made possible by letting the stock GM ICE drive the front axle through the stock transmission while the electric motor in the rear is electrically disconnected from the ESS through opening of the internal pack contactors. This would result in no power produced or recovered at the rear axle.

Parallel mode which is the primary operating mode for the HEV operates in an All-Wheel Drive (AWD) fashion whereby the ICE powers the front axle, and the electric motor powers the rear axle. The strategy in place is a basic pedal based look up table that is tuned for team developed stop and go style drive cycle. The team is currently exploring development of a deterministic rule based continuous Charge Sustaining (CS) mode that aims to ensure the battery pack State of Charge (SOC) is maintained around a certain SOC level for a given drive cycle. Since the ESS is 5.5 kWh total, the 33% operating mode results in a 1.5 kWh of usable drive energy. This goal of the charge sustaining strategy is to maximize the usage of the usable SOC window in a combined city and highway-based drive cycle. Lastly EV mode, this mode

requires that the transmission is forced into neutral, and all torque is requested from the rear axle. At the current state of integration, due to a lack of DC-DC, the team only runs the vehicle in full hybrid or ICE only modes.

1.3 Objective

In the contemporary implementation of a Hybrid Supervisory Controller, focus is placed on modelling and simulation of reducing the energy consumption, and consequently minimizing the environmental impact of a Hybrid Electric Vehicle. In pursuit of this goal, there is generally a minor emphasis placed on the importance of the simulation framework setup, that ultimately supports the model testing activities. This can lead to a higher developer workflow inefficiency, resulting in an increase in repetitive modelling tasks such as initialization, porting to end hardware such as HIL and testing of requirements.

This primary objective of this thesis is to present the Model Based Design (MBD) framework implemented in a MATLAB/Simulink environment for the deployment of the Hybrid Supervisory Controller (HSC) of a P4 Parallel Hybrid Electric Vehicle. Functional and hardware interaction layers of the HSC are expanded on through a custom Model-Configurator tool that wraps the simulation model in a class object. Ultimately from a developer standpoint, this thesis serves as a reference and knowledge transfer document exemplifying the strengths and costs associated with development and maintenance of a highly organized framework for future powertrain-oriented HSC development.

The secondary objective of this thesis is to present in detail the roles of the 8 main sub-systems that form the Hybrid Supervisory Controller which incorporate masked library blocks populated with data through the configurator tool. These include the driver block, I/O, fault detection, vehicle state control, torque strategy, component level execution, plant model and soft-ECU. Testing, and validation of the team retrofitted powertrain are discussed in the final chapter as per the results collected at MATT CTC, with an in-depth retrospective of the team's established Vehicle Technical Specifications (VTS), and the future work needed to rectify current system limitations.

1.4 Thesis Outline

This thesis includes six chapters inclusive of this introduction. These chapters are categorized in sub-sections that are outlined in accordance with the development process of the Hybrid Supervisory Controller through use of Model Based Design, the results of which the developed controller then are tested and validated against the prototype vehicle.

Chapter 1 – Introduction & Background

This chapter provides insight into the motivation behind the thesis, outlines and objective and provides a brief overview of the stakeholders involved with the project. This chapter also introduces the project vehicle at a high level.

Chapter 2 – Literature Review

Outlines state of the art, pre-existing research on two main topics related to the work described in this thesis namely Hybrid Vehicle Architectures and Automotive Software.

Chapter 3 - Utilization of MBD in a Requirements Based Development Workflow

This chapter contributes uniquely to the expansion of MBD requirements-based workflow incorporating Object Oriented Programming principles to ease test environment switching. Additionally, the process surrounding development of requirements, and systems safety is outlined to set up the stage for the implementation of the Hybrid Supervisory Controller.

Chapter 4 – Hybrid Supervisory Controller

This chapter deep dives in the organization, and implementation of the HSC. The controller, plant model, I/O and utilization of the tester block is discussed that tie back the workflow and strategic decision made to test and verify functional and safety requirements mentioned in the chapter 3.

Chapter 5 – Model Validation Testing & Results

This chapter culminates validated results against the prototype vehicle of the supervisory controller and plant SIL model. This section also outlines overall vehicle VTS performance, and potential shortcomings.

Chapter 6 – Conclusions

This chapter is a summary of achievements, and shortcomings that concluded as a result of 2 years of pursuing the development process. This section also sheds lights on other research areas that benefitted from this approach.

Chapter 2
Literature Review

The advent of combining two types of propulsive units opens the possibility for a myriad of architectural topologies. The topologies or arrangements are normally centered around the placement of the electric motor with respect to the ICE. Depending on the size of the ESS, packaging constraints, and appetite for complexity. The electric motor can be strategically placed to allow for full EV mode; range extending; charging; and/or fully parallel drive modes [11]. The foremost function of the hybrid architecture is to capitalize on fuel economy and emissions, primarily through energy recuperation during deceleration events. The type of architecture deployed on the vehicle delegates how energy is sourced. For instance, an only a P0 electric motor is able convert energy directly from the engine in series or series-parallel powertrain. There exist however a band of hyper cars and motorsports examples that serve as the epitome of energy management, technical prowess, and optimization to serve a singular purpose which is to go fast around a racetrack [12].

The purpose of this section is outline on the complexities involved with various hybrid architecture topologies available, the key role of automotive software in the implementation of the hybrid supervisory controller, and an exploration of the main roles involved in the development of the HSC. The significance of the work done in said domains will help contrast the added organization, and workflow that UWAFT has incorporated in its implementation against the state of the art.

2.1 Hybrid Architecture Topologies

On a scale of ICE only to BEV there lies a degree to which a vehicle can be hybridized. As earlier mentioned, the architecture requirements stem from the placement of the electric motor with respect to the ICE. The placement brings along with it software, mechanical and electrical complexities that ultimately must be spec'd to serve the vehicle's VTS, derived from studying the end customer and project needs analysis.

As the degree of electrification increases, dependance on the ICE diminishes. This diminishing dependance on the continuous high power, and range available in an ICE only setup, is made possible through addition of a large ESS and electric motor to retain the vehicle's VTS performance in terms of acceleration, range, fuel efficiency measured in miles per gallon (mpg) as well as GHG emissions.

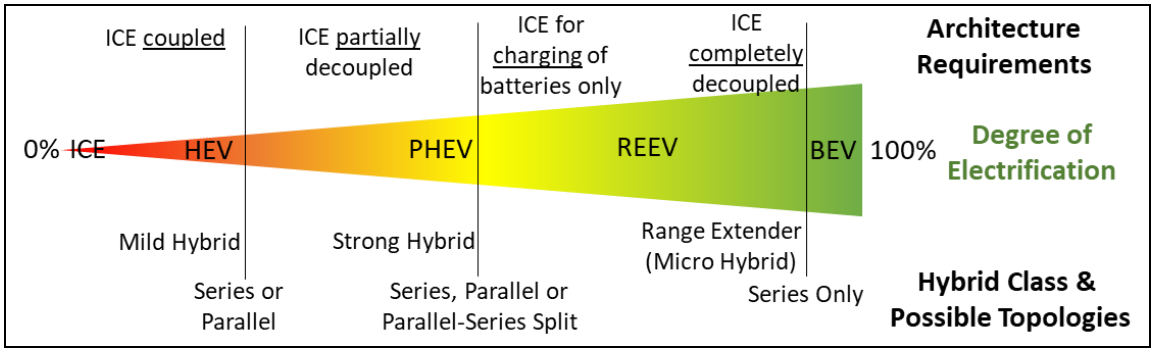


Figure 6: Degree of Electrification and Possible Architectural Topologies

To better understand state of the art of HEV technologies. We must look at the extensive research and industry applications that have taken place to categorize the specific features that are offered by the various HEV topologies. The topologies at their core can be categorized by the role the electric motor plays around the ICE. P0 is attached to the engine via belt/pulley; P1 spins directly with the engine through a shaft; P2 is post engine pre-transmission – disconnected through clutch; and P3 incorporates is a pre-differential electric motor; whereas in a P4 configuration the motor is integrated within the final rear drive ratio housing [13]. Take note however that the terms mild or strong are loose terms, and nod more to the size of ESS/electric motor pair than have any strong bearings on the overall topology of the hybrid vehicle. For instance, a mild hybrid can be a parallel with a small P0 or P1 assisting the engine or be a series, working through electric power on a different axle P3 or P4.

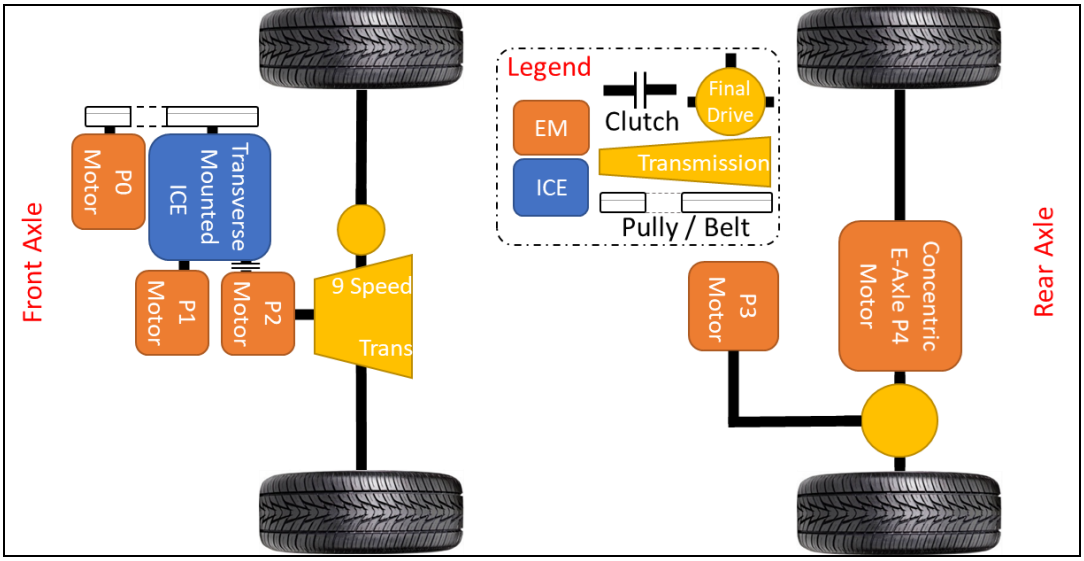


Figure 7: P0, P1, P2, P3 and P4 Hybrid Topologies Motor Placement

The split between series or parallel configurations comes down to how the electrical or mechanical power from the electric motor and ICE is delivered to the road. In any hybrid vehicle when the load is driven purely by the electric machine, irrespective of where the power (ESS or ICE) is sourced from, that is a series configuration. When the electric motor and ICE are providing the power in tandem, irrespective of whether they share the same axle or not, that is a parallel operation.

2.1.1 Series Hybrid

In a series operation, a generator spins directly with the engine, charging up the ESS such that this electric power in turn drives the electric motor. Spinning an engine to drive one electric motor, only to spin another electric motor may sound a bit ill-advised on paper, but this topology opens the door for pure electric driving given a large enough battery pack. This is especially true of the very early hybrid production vehicles such as the GM-EV-1 or the Fisker Karma, where in effect these vehicles were Range Extending Electric Vehicles (REEV), that were intended to be driven pure EV all the time. The addition of the ICE was intended to only extend the range, and not provide power to the wheels ultimately. The choice for series architectures provides key energy recuperation opportunities at RPMs where the engine is least efficient. [14] The small window approximately between 2000 and 3500 RPM in Figure 8 shows the region where the GM 2013 2.5 Ecotec LCV engine has the highest Brake Thermal Efficiency (BTE). This

region for instance of the % BTE, would for instance represent the speed-torque spectrum for designing a suitable P0/P1 generator for series configuration.

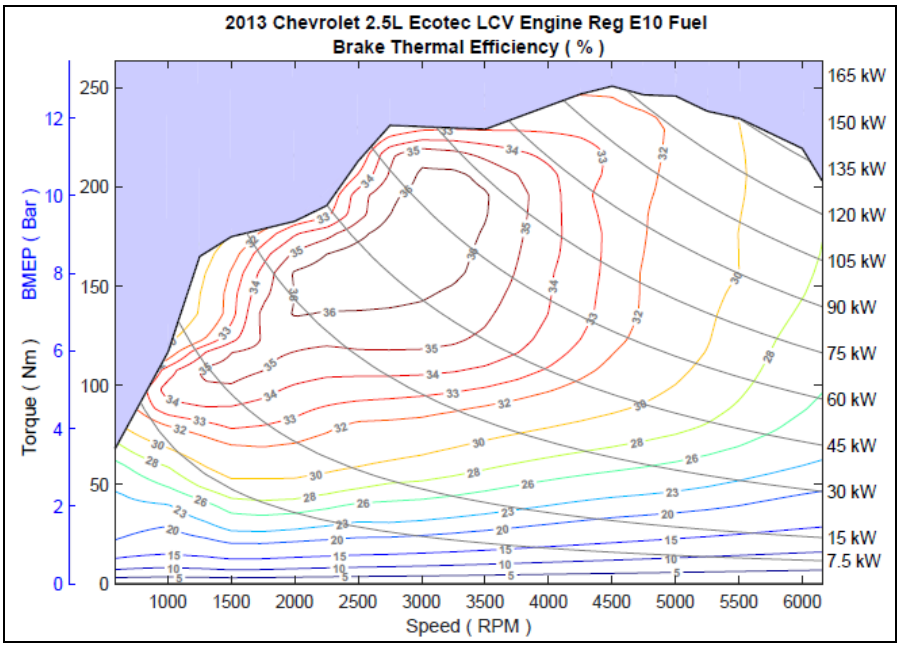


Figure 8: BTE Map shows higher efficiency regions [14]

2.1.1.1 Series Hybrid Pros and Cons

Since power from the fuel tank never meets the tarmac but only through the electric motor. Series powertrains are relatively easier to package compared to their parallel or parallel series split counterparts. The series or range extending ICE can be smaller in size; packaged more compact and can even be used as a damper for reduced Noise Vibration and Harshness (NVH) providing an improved drive quality experience [15]. Since an ICE is most efficient in a narrow RPM band, elimination of torque transferring/converting devices such as clutches and transmission, allow for purpose-built ICEs that are not expected to be high revving. Such is the example of the low compression Atkinson-cycle adapted ICE in the case of Toyota Prius [16], that can operate in both series and parallel modes.

The addition of an additional electric motor to drive the vehicle on top of the generator, penalizes the vehicle architecture in terms of weight, and purchase price. The major weakness of the series architecture is its predominant inefficiency that arises during sustained load driving. The coupled losses that are incurred during the mechanical to electrical power

conversion at the ICE/generator level, are much higher compared to driving the vehicle during ICE only, or ironically, EV only.

Upsizing the ICE, generator, electric motor and ESS are possible work arounds for allowing the vehicle to drive more in EV mode, however there are more gains to be had from upscaling the architecture altogether to make the vehicle more capable, before it is categorically a BEV, and this is where the parallel architecture comes in to play.

2.1.2 Parallel Hybrid

Parallel as the name suggests does not warrant routing of the mechanical power to electric before meeting the tarmac. It instead allows for tandem power delivery from the ICE and the electric motor. A parallel configuration is possible with all P0, P1, P2, P3 and P4 hybrid configurations. Parallel systems where the electric motor is directly coupled to the engine (P0, P1 or P2) are generally smaller as compared to when the electric motor is in either post transmission P3 or integrated as part of rear axle P4. This trade-off is driven due to limitations of peak torque that is seen on the engine crankshaft, and transmission.

2.1.2.1 Parallel Hybrid Pros and Cons

The parallel hybrid topology is a more efficient method of hybridization as it stands to benefit from no conversion losses i.e., ICE-generator. Some of the features of the series topology such as engine start/stop, is both possible and not, depending on the complexity of the topology. For instance, a P0, P1 or P2 parallel can crank the engine, but a p3 or P4 parallel cannot. More often than not Original Equipment Manufacturers (OEM)s develop parallel architectures due to the inherent simplicity of the architecture i.e. use of a simple clutch, over generator integration such as in the case of series. A similarly sized parallel configuration is generally more powerful as the electric motor and ICE do not need to share the same axle and can be appropriately sized larger.

Due to the larger role played by the ICE, a parallel hybrid can be conceptualized with a much smaller ESS. The smaller ESS size allows the architecture to be less dependent on a grid for charging, and makes it less complex since an additional onboard HV charger is not needed. Whereas in a series the ICE is merely a range extender for the unplanned miles and requires the additional plug-in capability to be considered a robust contender for all types of driving

conditions, unless the architecture is parallel-series split, which bring us to the parallel-series split architecture. Overall, the parallel architecture makes for a simpler, less expensive method to obtain a better fuel economy, faster acceleration, and lesser overall vehicle level emissions [17].

2.1.3 Parallel-Series Split

The parallel series split architecture combines best of both series and parallel architectures, allowing for mechanical and electro-mechanical paths for torque transmission to the tarmac. In this configuration, at least two electric motors are needed, series configuration warrants no mechanical path from engine to tarmac, which in the parallel-series architecture is generally supported by use of the two electric motors. This however is not always true such as in the case of the Toyota Prius which operates a single motor through a planetary gearset allowing for engine assist, battery charging and full EV driving depending on the driving situation.

2.1.3.1 Parallel-Series Split Hybrid Pros and Cons

Parallel-Series split architectures provide the best of both worlds, plug-in EV charging, energy recuperation as well as motor assist. Due to the additional capability of this architecture to displace most amount of fossil fuel through full EV operation - this architecture generally is the most adopted setup for PHEVs. Due to the higher degree of electrification (larger ESS/motor), and integration of 2 motors, or 1 motor + planetary gearset, the systems are generally more expensive to develop. Simultaneously allowing for a more fuel efficient, lower emission and much smoother ride quality product. This however comes at a higher up front purchase cost, and directly impacts the external grid system's ability to support charging loads.

2.2 Automotive Software

Evidence of the first piece of computer code on an automobile date back to 1957. Named Electrojector, the transistorized electronic fuel injection (EFI) system was designed for the American Motors Corporation's 1957 Rambler Rebel by Bendix [18]. Despite its promises on paper, the technology was only put on pre-production vehicles, of which none were sold as EFI variants. It is at this point in the history of the automobile that a piece of computer code was first used to track crank position to pulse fuel pre-ignition. Bosch would perfect their Jetronic

Fuel Ignition System in ’68, only seeding the basis of what would be a the most competitive aspect of the automotive industry in 2021, automotive software [18].

Initial software in the automotive industry were local implementations written in languages such as C; specific to obtaining functionality outcomes at the hardware level. Typically, these nodes were run-on low-level software on custom developed ECUs that would interface with dedicated sensors and actuators. In the late 80s the Controller Area Network (CAN) interface was introduced as a means for ECUs to communicate with other ECUs, acting as distributed localized work nodes [19]. This would allow for a bottom-up (build as you go) approach to ECU development. ECUs would be added as need arose on the pre-existing or additional vehicle CAN bus. It is estimated that a 2007 BMW 7 series implements some 270 functions, deployed over 65+ embedded platforms [20]. Today this number would exceed 100+ distinct embedded units [21].

Due to the rise in the number of software domains on a vehicle, spanning safety - both on a vehicle and user level; infotainment; and powertrain there existed a need for a development process that was repeatable, at lower cost. In automotive software it is a common practice during software development process to test software nodes tested against an analyzed set of requirements to maintain lineage, diagnostic characteristics and tracking of system improvements before the software would ever be tested on the end target vehicle platform.

2.2.1 Model Based Design

Model Based Design (MBD) is a math-based software development process, that makes it easier to develop code inside a virtual prototyping environment. This method facilitates visually understanding algorithm behavior before embedded code is written [22]. MathWorks MATLAB and Simulink are industry wide used programming tools that place MBD at the center of systems programming, more so in the case of Simulink than MATLAB [23]. The HSC sits at the center of all vehicle controls system responsible for estimating vehicle/component(s) state, monitoring of component thresholds, and deploying the torque strategy, among other things. The development of HSC in an MBD environment, allows for testing of requirements to occur at various levels such as SIL, HIL and VIL [24].

As the name suggests, in an MBD design process the model is at the center of the development workflow. In a real-time system, such as in the case of a hybrid vehicle, a model

can represent both the soft-ECU (software representation of a physical ECU), where the embedded states are simulated by the HSC [25], as well the plant model that represents the physics or data driven drivetrain components. The combined soft-ECU generally constitutes the plant model and is calibrated and improved over time to gain a higher fidelity representation of the physical system.

MBD systems need to incorporate the idiosyncrasies that stem from the vehicle architecture, combining the requirements that encapsulate software feature interaction. There exists the logical architecture which is based of decomposed component software interactions at the functional level, which would constitute the structure and layout of the HSC, and the technical architecture that defines the deployment of the basic software units, which constitutes the functionality of the HSC. [20]

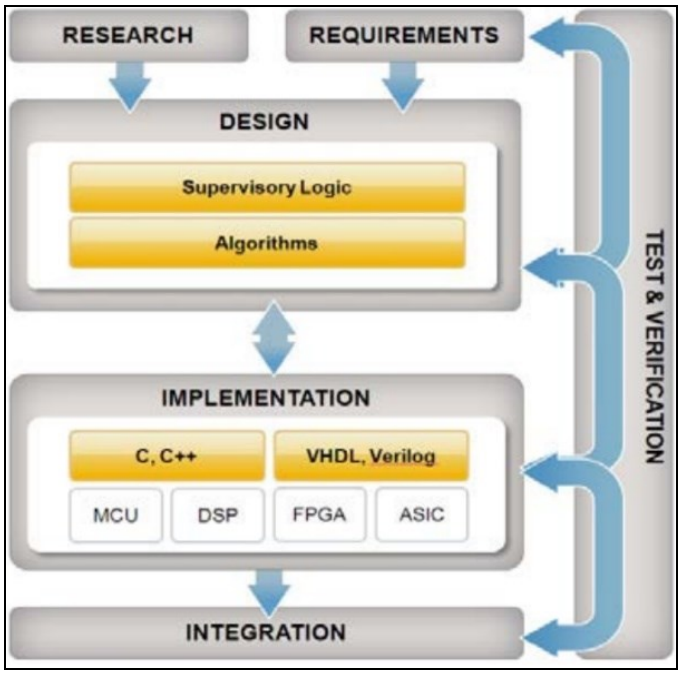


Figure 9: Research and requirements both feed in to the design of the HSC [26]

Over the span of the development process, research and logical requirements are generated at various levels of the V-diagram development process. These can range from high level requirements such as customer requirements (VTS targets) to component level requirements (safe torque request). The cascading nature of requirements in degree of fidelity, puts requirement traceability at the center of the MBD based implementation. These requirements

can be implemented for a variety of non-real time and real-time testing environments before end hardware integration.

The use of simulation to verify controller requirements proves extremely useful when replication of a certain state in a real environment is not desire-able or safe. For instance, applying exceedingly high torque on the electric motor for estimating thermal system limitations. Systems modelling based in first principles paired with an understanding of physicality of the system, enables pre-calibration tuning of the SIL model for performance estimation. Here the plant model plays a critical role in representing the physical system. For instance, estimating lowest possible fan speed and coolant pump to maintain electric motor operating temperature. Real life or VIL calibration plays a key role in improving the model, and thus reducing the controller effort to obtain a key outcome. Calibration in turn can have its own performance requirements such as the degree of accuracy required at various operating points of a physically actuated system. [27]

In the automotive space MBD is extensively utilized to accommodate requirements testing at the software, hardware, and vehicle levels. The results taken from the testing environments is fed back to either improve the SIL robustness or go back to the drawing board with the requirements itself. MBD facilities the incorporation of new code or requirements due to the ability of the design to accommodate software, interface, and execution segregation [28]. This is where the HSC plays a key role in the organization of all software code. HSC can be built as per needs basis without much thought to organization, but as we will see in Chapter 4 of this theses, the role of the HSC is extremely involved, and properly organizing the model-based design, becomes a necessity.

2.2.2 Hybrid Supervisory Control (HSC)

In a conventional vehicle, a driver requested vehicle torque command is honored through the ICE only. The ICE is a localized system that has stood the test of time, and the controls for which are localized and well understood. From a torque distribution standpoint, the HSC arbitrates with the up-stream controllers that can request torque such as an autonomous driving controller or driver pedal, to downstream components such as the high voltage inverter or the ICE ECU, to meet the acceleration request from the vehicle. The terms upstream and downstream are used to describe the signal path from the creation of a request to component

actuation. HSC decision making is feedback driven process, i.e. it is constantly monitoring hundreds if not thousands of signals coming from various ECUs onboard the vehicle. Typically, the HSC prototype controller has access to most if not all vehicle CAN communication networks in a hybrid architecture. [29]

The bottom-up approach of adding functionality as needed, owes to the availability of the diagnostics, safety, state, and health signals available on the CAN bus message frame that are leveraged for development of safe system operation for the HEV. The signals constitute basic feedback signals such as the ESS SOC reported by the Battery Management System (BMS), up to more advanced - such as triggering of a fault state, in case a component is operating out of it's safe boundary limits. The HSC is responsible for at least the control of vehicle (component) state estimation, housing of the plant model representation and execution of the torque strategy for the hybrid propulsion systems.

2.2.3 Vehicle (Component) State Estimation

The ECUs onboard the propulsive units of the hybrid vehicle, are all under a high-level management of the HSC. By design the HSC is developed to balance competing objectives e.g., fuel economy and driving performance. Monitoring system limitations such as protection of components at their limits, ensuring a healthy state of charge for ESS, while also honoring the driver inputs, often leads to the HSC operation to become exceedingly complex. It is the responsibility of the HSC to determine the state of a component through the available communication CAN, Digital or Analog network, to deem a control action safe to command. In an BEV, the HSC at the very least is arbitrating and keeping track of system states of at least the BMS, Body Control Module (BCM), the inverter, cooling fan, cooling pump and the motor controller/inverter [30]. In a hybrid vehicle however, the interaction is even more complex where the ICE ECU, the Transmission Control Unit (TCU), among other chassis, cabin components are tracked for purposes of state estimation.

There exists a desire for producing a control architecture that interprets incoming signals and groups their values to represent them as systems states. To implement systems that can operate in various states – a system state estimator is developed that allows the HSC to 1) be in the correct operating state and 2) protect the vehicle from harm through raising of flags. These systems state generally are non-linear and approaching them analytically is near impossible but

are required to drive a large amount of decision making. It is for this reason they are implemented in the form of finite state machines. [31] These finite state machines take in the messages from the vehicle CAN bus for allowing/disallowing for certain HSC control actions. From a calibration/testing point of view, the state machines when implemented in an MBD structure allows for all system states to take form of requirements of the form if x do y – allowing for implementation of the state estimation layer in the form of finite state machines. The implementation is more approachable from a viewing, maintenance and troubleshooting perspective.

The interacting ECUs on-board the component being controlled report among other things to the HSC the physical parameters such as power, temperature, voltage but also their system states such as ready, enable, fault, or off from and to each other. For instance, if the ESS HV DC Link bus is not energized, the internal ECU of the inverter is going to be in a ready, but not enabled state, thus signaling to the HSC, that honoring a torque request is not possible. Following is an example of an inverter state machine implementation, publicly available to view from Cascadia Motion inverter developer, that illustrates the ECU side state machine. [32]

VSM_State	Name
0	Start State
1	Pre-charge sequence initial state – Turn on the pre-charge relay
2	Pre-charge sequence active state – Waiting for capacitor to finish charging.
3	Pre-charge sequence finish state – Completes the final checks before proceeding to Wait State.
4	Wait State – waiting for activation of forward or reverse.
5	Ready State – Activates the inverter state machine to begin energizing the motor.
6	Motor Running State – Normal motor running
7	Fault State – The controller has faulted
14	Shutdown in Process – In key switch mode 1, user has turned key switch to off position.
15	Recycle Power State – This indicates that the power to the controller needs to be recycled after EEPROM Programming is complete.

Figure 10: ECU State Machines track OEM specified ECU states [32]

Pre-calibration tuning and developing an understanding of the physical makeover of components is an important aspect of state estimation. Fair amount of research has been conducted in implementing state estimation controllers, that are able to predict systems states that are not discretely reported as part of the messages on CAN signals [33]. This approach for

instance is widely used in safety critical active autonomous driving systems where the raw data from a perception stack may not always be operating in an ideal environment, such in precipitation or fog.

2.2.4 Plant Model Representation

The plant model represents the physicality of the system the HSC would interface with in a real time environment. The plant model is generally comprised of mathematical (analytical) models such as in the case of longitudinal body dynamics, as well as behavioral models whose parameters are populated through look up tables, that are made available by component suppliers or that are developed as part of extensive system characterization, such as in the form of soft-ECUs. The incorporation of plant model within the HSC can be broken in to two different approaches.

The backwards modelling approach, here the environment is pre-loaded with information such as vehicle weight, road gradient; and other vehicle characteristics such that the required tractive force at the wheel is calculated first. Then this force is equated as wheel torque, which is then propagated backwards through the drivetrain components and then to the engine. This approach is non-causal, as the static pre-determined efficiency maps are used to determine operating points for the powertrain components. This approach is also "quasi -static", since the speed demand is not propagated but 'applied' via the drive cycle through the drivetrain. Meaning the physical limits, unless captured within the models cannot be explored, and as such this approach is not scale-able for a Hardware in Loop bench setup. [34]

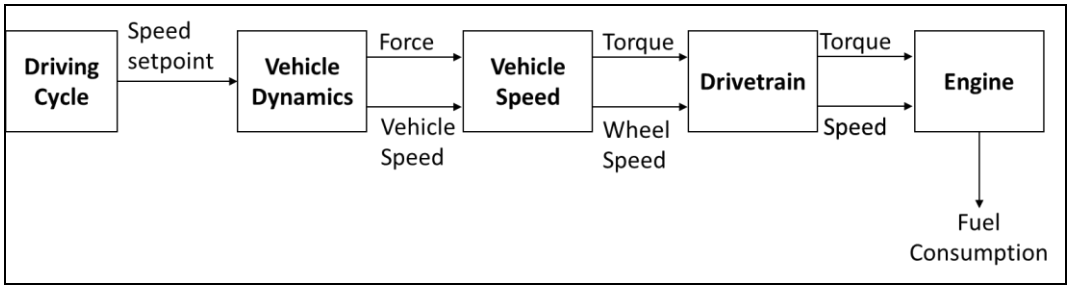


Figure 11: Backwards Modelling Approach [34]

The alternate and more commonly utilized modelling scheme is known as the forwards modelling approach. This approach uses a driver model, modelled as a PI or in some cases the Charles MacAdam Driver model [35] such as in the case of UWAFT, to emulate real world driver

input to system for following the speed-time drive cycle. Here the acceleration target is determined and is converted directly to a torque request that is then propagated through the engine, transmission, other transferring/reduction components - ultimately resulting in commanded torque at the wheels. Due to the closed loop nature of this modelling approach, the resulting vehicle speed post the plant, is fed back into the driver model, compared, and consequently nets in a higher or lower acceleration target.

The natural progression of signals is a much closer representation of real-life vehicle driving and is thus a much better suited environment for controls development and testing, thanks to its scalability in a HIL test environment. [36]

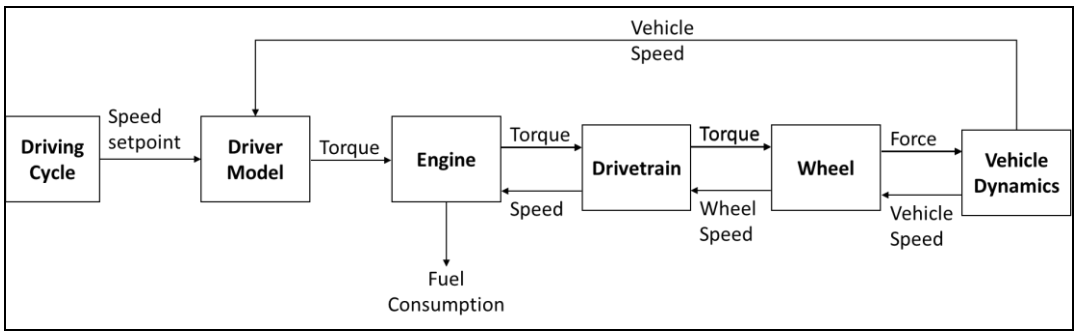


Figure 12: Forward Modelling Approach [34]

2.2.5 Hybrid Torque Strategy

At the very heart of the HSC, lies the energy management of the hybrid drivetrain. In a conventional vehicle, the acceleration/deceleration requests from the driver are directly translated to torque commands from the ICE. Hybrid electric vehicles are built different, accommodating one or more electric motors powered through an ESS. Due to the complexity involved with real time power delivery, as well as management of the battery SOC - naturally inheriting complexity from the energy management problem. An exhaustive amount of research and approaches have been developed in devising of a hybrid torque strategy. These approaches place emphasis on different aspects of hybrid energy management strategy such as fault detection for validation and testing [37], monitoring and control of battery degradation [38], purely optimality driven solution development for a cost function [39] and real time energy deployment [40] among many others.

However, the main objective of an energy management system is to minimize the overall energy consumption needed over defined drive cycle, while also satisfying the user's torque always demands. Energy management strategies can be split in to two main categories – optimal [7] [39] and rule based [41] [42]. Rule based control strategies comprise of deterministic and fuzzy logic rule-based methods, whereas optimal strategies utilize methods to globally optimize for the determination of a control strategy.

2.2.5.1 Rule Based Energy Management

Rule based energy management strategies are based on pre-defined understanding of system inefficiencies and are generally aimed at deploying heuristics to avoid operating in those scenarios as much as possible. These heuristics or rules are devised from the understanding developed around ICE fuel consumption, electric motor/ ESS efficiency maps, and human experiences. This allows definition of predefined points or threshold for when components will be used. These are generally implemented in the form of look up tables or in state machine style format.

Implementations of a rule-based energy management strategy includes fuzzy logic controllers, as well as deterministic controllers that utilize state machines. Both methods are equally robust however, computational complexity is higher with a fuzzy logic implementation. [43] Deterministic rule-based controllers include on/off look-up or finite state machine style control strategies. The state machine-based control transitions occur between modes based to primarily facilitate driver demanded torque while taking operating conditions and sub-system faults in to account. [44]

2.2.5.2 Optimization-Based Methods

There are a few different methods that stem from the *optimal control theory*, that work around the optimality criterion, aimed at finding a control law. While having a perfect understanding of the mathematical models of the system, and knowledge of the control horizon enables devising optimal control. In a real time, environment however, where the future control horizon is unknown, the solution is suboptimal. The works of G. Rizzoni et al. [45] [46] discuss various optimal energy minimization methods including Dynamic Programming (DP) and Equivalent Consumption Minimization Strategy (ECMS).

DP utilizes a numerical methods-based approach for solving multistage decision-making problems. The approach can produce optimal results of any complexity level, granted computational capabilities. [34] DP is a backwards-looking minimization technique that is only simulation implementable. This is dictated since the algorithm requires prior knowledge of the driving conditions such as the drive cycle, grade, driver model, etc. The entirety of the problem including the model, control and state variables are computed for minimization, at each time step for the drive cycle. DP can be used to minimize for multiple objectives such as energy consumption, fuel flow and emissions. The resulting calibration can be in-turn used to define heuristics, that form a rule-based controller for a practical real time implementation.

This is also true for the ECMS based approach which offers a real-time implementable optimization instantaneously taking in to account the energy consumption, while maintaining battery SOC around a reference point. The intuition behind equivalent fuel consumption stems from the fact that in a traditional HEV, the power within the drivetrain is sourced from the vehicle's fuel tank. This includes both the chemical energy sources from the fuel tank as well as the *equivalent* energy sourced in the form of electrical energy from the ESS. A cost is then assigned to the electrical energy, which allows saving (fuel), as part of the objective function. The approach enables instantaneous minimization to be performed at each time instant of the drive cycle, without prior knowledge of the drive cycle in its entirety.

Chapter 3

Utilization of MBD in a Requirements Based Development Workflow

The goal of chapter 3 is to expand on UWAFT's implementation of the Model Based Design framework. Here we start off by describing the broader strokes of the integration state of the Hybrid P4 UWAFT Blazer. Showcasing at a higher level, the multiple ECUs present within the vehicle's CAN architecture that the HSC interacts with. Then we will take an in-depth look at the workflow, and framework developed for UWAFT's requirements-based development. This section is also used to describe the role and organization of UWAFT's Requirements Traceability Matrix (RTM), in increasing cross-team transparency for both the HSC development for the PCM sub-team, but also from the viewpoint of other non-software sub-teams.

3.1 Hybrid Platform Conversion

The project 2019 Chevrolet Blazer RS from here on out referred to as the UWAFT Blazer, started out life as an AWD 3.6 L V6 vehicle. This vehicle from factory comes with the following VTS. [47]

Table 1: Stock 3.6L V6 Chevrolet Blazer VTS [47]

Vehicle Technical Specification	Value
Layout	Front Engine, AWD, 5 Door SUV
Engine / Transmission	3.6 L V6 LGX / 9T50 9-Speed
Curb weight (Front%/Rear%)	1985 KG (59%/41%)
0-60 MPH	6.1 sec
60-0 MPH	126 ft
Fuel Economy Combined	21 mpg

3.1.1 Market Definition - Mobility As A Service (MAAS)

The integration level modifications of the vehicle platform were driven in part by the team's research on customer discovery. Customer discovery was key in devising vehicle level requirements or VTS, for dictation of the research vehicle's performance targets. Car sharing is a sub-set of the MAAS market. Typically, the car sharing service is concentrated in denser populated urban areas, where a larger and more accessible customer base can be served. For the definition of the vehicle technical specifications, UWAFT ran an extensive survey with over 162 respondents with age groups ranging from 18 to 45 year(s). The goal of this survey was not only to develop insights into the sizing and layout of propulsive components that would be needed for achieving the VTS, but also the vehicle features that are to be achieved in the final version of the prototype vehicle in the last year of the competition.

Through the target market analysis, it was deduced that the vehicle be a traditional hybrid vehicle, with like stock cargo space, improved fuel economy/acceleration and have Connected and Automated Vehicle (CAV) safety features. While the CAV oriented features were developed alongside the conversion of the stock vehicle to hybrid, those active safety aspects of the vehicle however are only touched in this thesis to the extent in which they overlap with the focus of the development of the HSC. Some of the propulsion-oriented performance specs of the modified UWAFT Blazer include the following.

Table 2: UWAFT Vehicle Technical Specifications

Specifications	Units	UWAFT VTS
Layout	N/A	P4 Parallel Through Road
Engine / Transmission	ft	2.5L I4 NA LCV / M3D GF9
Curb weight	kg	2100
0-60 MPH	s	5.5
60-0 MPH	ft	158.2
Fuel Economy Combined	mpg	30.83 mpg

Note that weight distribution of the vehicle at the time of study was not determined, however the final weight change with the addition of the hybrid components such as motor, ESS and inverter were estimated and are discussed in the next section of this work.

3.1.2 Vehicle Modification Summary

Over the years 2019 to 2021 the stock GM Blazer underwent a fair degree of modifications. These modifications include addition of controllers, propulsive EV drivetrain, an energy storage system, and a thermal system for cooling of the EV components. The main propulsion components include the HDS 1.5 kWh Li-Ion capacity battery pack, Semikron SKAI2HV Inverter, and an AAM EDU4 motor. Table 2 summarize the technical specifications for the added components, and their location relative to the wheels in the vehicle.

Table 3: HEV Components

Component	Performance Specifications
HDS Custom ESS	Peak discharge power: 121 kW Continuous discharge: ~28 kW Total pack capacity: 5.5 kWh Pack nominal Voltage: 346V
AAM EDU4 E-Axle Motor	Peak power: 150 kW Final drive ratio: 9.04:1, Peak torque: 346 Nm
Semikron SKAI2HV Inverter	HV DC Link Voltage: 50-400V Peak Power: 150 kVA
EMP WP32 Brushless CAN enabled – Electric Water Pump	Operating voltage: 12 and 24 Volts

Through the integration process phase the necessary mounts, drive shafts, cooling, compute units, safety hardware, active safety sensors and thermal systems were integrated to develop the UWAFT P4 parallel hybrid aimed at the MAAS market. This conversion saw the vehicle's overall mass increase by 210 kg, with a 6% shift towards rear bias. This fell within UWAFT's earlier described curb mass goals in Table 2. The addition of mass towards the rear can be attributed to the addition of the motor on the rear axle, as well as the ESS.

Before the vehicle was integrated a series of regulations had to be abided by and followed to ensure the modified vehicle met the Non-Year Specific Rules (NYSR). These rules include restrictions around various aspects of the vehicle design. Some of the pertinent rules include addition of CAN enabled communication systems that interfaced with the stock vehicle body and driveline components, safety High Voltage Interlock Loop (HVIL). E-Stops and 12V Disconnect Switches that allow safe and immediate powering down of the ESS High Voltage

(HV) contactors and HSC controller. This required integration of a fair degree of electrical components and wiring, as shown below in Figure 13 electric systems integration.

The purpose of showing this image, is to highlight the brevity of ECUs, 12V and analog/digital components that all communicate with the HSC one way or another for the safe operation of the HEV. Note that the HSC, the CAV compute device, and the relay control module are all situated in the rear of the car. This allowed for easier debugging of harnesses and accessing the HSC, for software flashing as the trunk of the UWAFT Blazer SUV is a relatively large and accessible space.

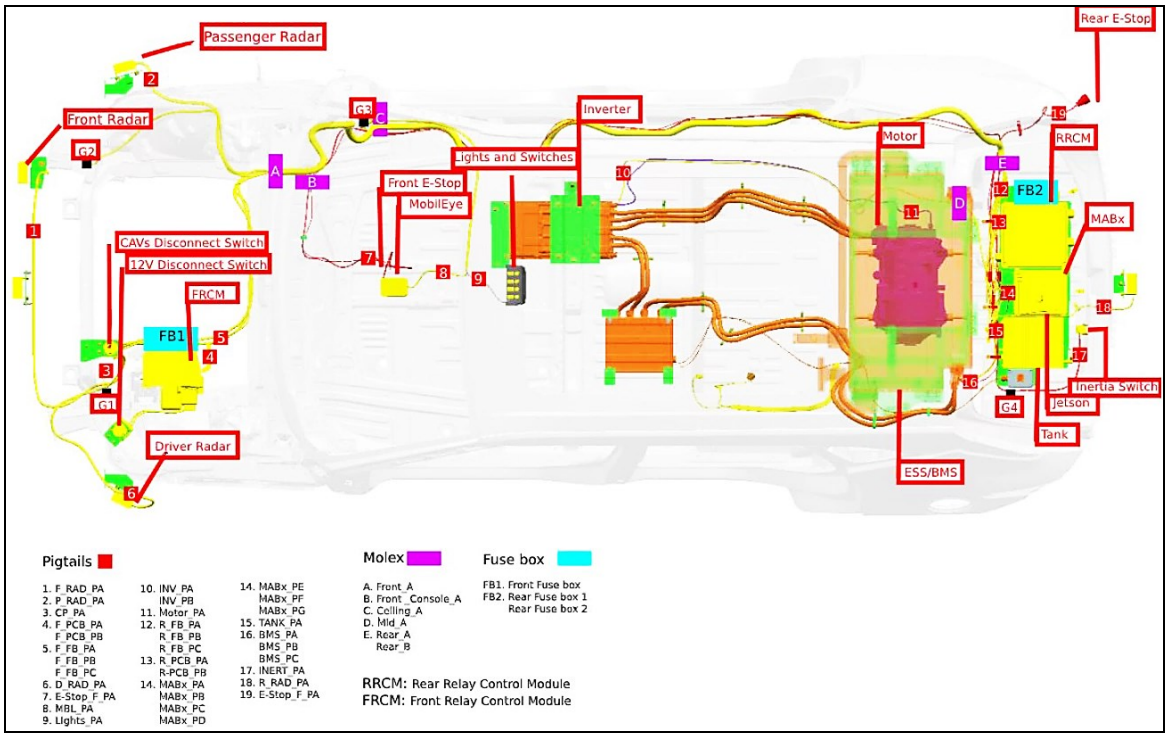


Figure 13: Vehicle Electrical Integration

3.1.3 ECU Layout & Interactions

The HSC interacts with both stock and team added vehicle components. The HSC is developed on the EMC sponsored dSPACE DS1401/1513 MicroAutoBox (MABx) II that serves as the central embedded prototype controller for software deployment. In total there are 5 major CAN buses on the vehicle. Three of which are stock to the vehicle - two of which are high speed (500 kbps) and one of which is low speed (<33.333 kbps) CAN. The two UWAFT CAN buses are both

high speeds. These are physically split due to both being responsible for different things. The CAV HS bus carries vehicle active safety/autonomy signals only that are processed and sent by the Intel Tank CAV compute unit. While the Prop HS bus interacts with the Relay Control Module (RCM) for component power toggling through the Relay Control Module (RCM), the Battery Management System (BMS) and the inverter also referred to as the Motor Control Unit (MCU). The UWAFT Blazer's propulsive units are housed on separate axles. The placement of the CAN buses is shown in Figure 14 which aims to highlight the segregation between stock system CAN and UWAFT added CAN. Note that the GM CAN buses are clumped and shown as one CAN bus.

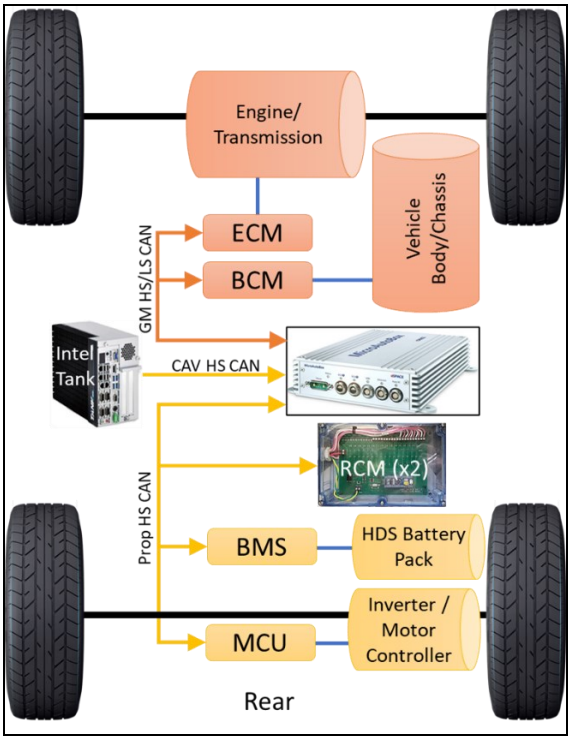


Figure 14: High level CAN only serial network diagram

This CAN segregation utilizes all 5 CAN bus ports on the MABx. Another important thing to note is that the MABx also interfaces with the components through Dig I/O – such as in the case of powering LEDs, and detecting safety switch status, as well as through voltage sensing on analog ports such as in the case of detecting keep alive circuits such as the HVIL safety loop that is sourced from the BMS and runs through all HV components.

3.2 Requirements Based Development Workflow

3.2.1 V-Model Development Process

UWAFT's software development process utilizes the V-model based development process extensively. This process discretizes the code development process to facilitate testing of developed requirements at multiple levels. Here progression of software development and testing can be tracked from conception to realization. The V-model of development allows developers, and requirements generators to incorporate feedback within the software development workflow, encouraging refinement of requirements and testing schemes through collaboration between project leads, and component level experts.

On the left side of the V-model, requirements are generated, and code is written to define system functionality. In the case of UWAFT, the highest level of requirement setting begins at the customer discovery level, where the Vehicle Technical Specification (VTS) are broken down into control system level requirements. For instance, in the case of UWAFT's P4 HEV, the need to have an HSC is a system level control requirement – this can be further broken down into smaller functionality level requirements features required for functionality of the HSC, for instance calculation of component torque limit-based on component temperature and battery SOC. As we traverse up the right-hand edge on the V-model, the testing moves from SIL to HIL to VIL - reflecting the target hardware and plant model validation at higher fidelity.

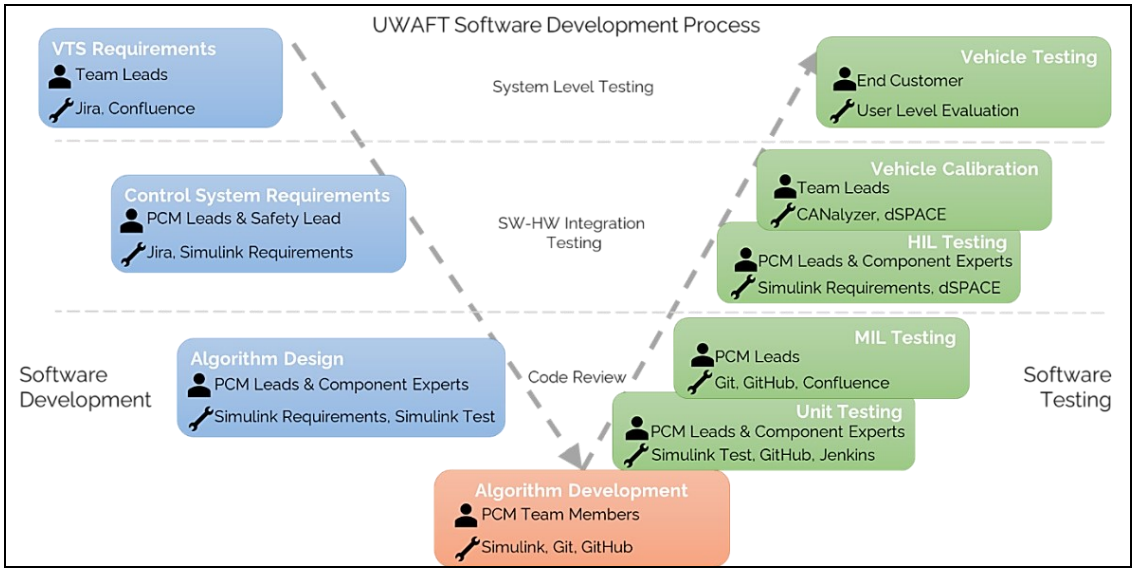


Figure 15: UWAFT V-Diagram Development Process for PCM sub-team

3.2.2 Requirements Development & Maintenance

Development of requirements is the first step in code development process, and serves as an important tool for outlining, and tracking of the development progress. There are two main types of requirements generated for the HSC development. System safety requirements and functional requirements. Systems safety requirements are generated through application of system safety analysis on the causal effects, interactions, and modes of operation a component or sub-system in a certain state. For instance, forced disengagement of team developed Adaptive Cruise Control (ACC) upon pressing of the brake pedal. Feature-based requirements are developed to drive performance and functionality-oriented aspects of the HSC. For instance, ensuring the actual battery SOC never dips below 30% for the duration of a drive cycle to prevent long term cell degradation. Interestingly feature based requirements often overlap with system safety requirements for instance protecting Li-Ion degradation from severe low and high SOC charging/discharging events. The generated requirements are documented, in the Requirements Trace-ability Matrix (RTM), which is a spreadsheet of requirements, categorized and organized based on the sub-team involved, and status of incorporation.

3.2.3 Incorporating Systems Safety into Requirements Development

Systems safety requirements are tracked and kept up to date in the centrally utilized, previously mentioned RTM. Figure 16 describes the system safety requirements development process that results in a robust system. Requirements are generated via study of competition safety requirements, team performance requirements and needs of the individual sub-systems. The task of reviewing, and developing systems based on these requirements lies with the developmental sub-teams. At least one member from each sub-team forms the Systems Safety Analysis Working Group - which is led by the Systems Safety Lead Engineer (SSLE). This ensures that the safety working group has up to date information about latest system level developments, and the RTM is updated based on the approval and review of the SSLE.

The role of the safety group is to work with developers on the individual sub-teams for development of requirements through careful study of component/sub-systems and the confidential EMC Non-Year Specific Rules (NYSR) to prioritize safe system operation. Once the requirement is documented, the sub-teams develop the sub-system/code and performs testing, the results of which are updated within the RTM. Upon verification and validation, a team lead initials the tested requirements, and the event is dated.

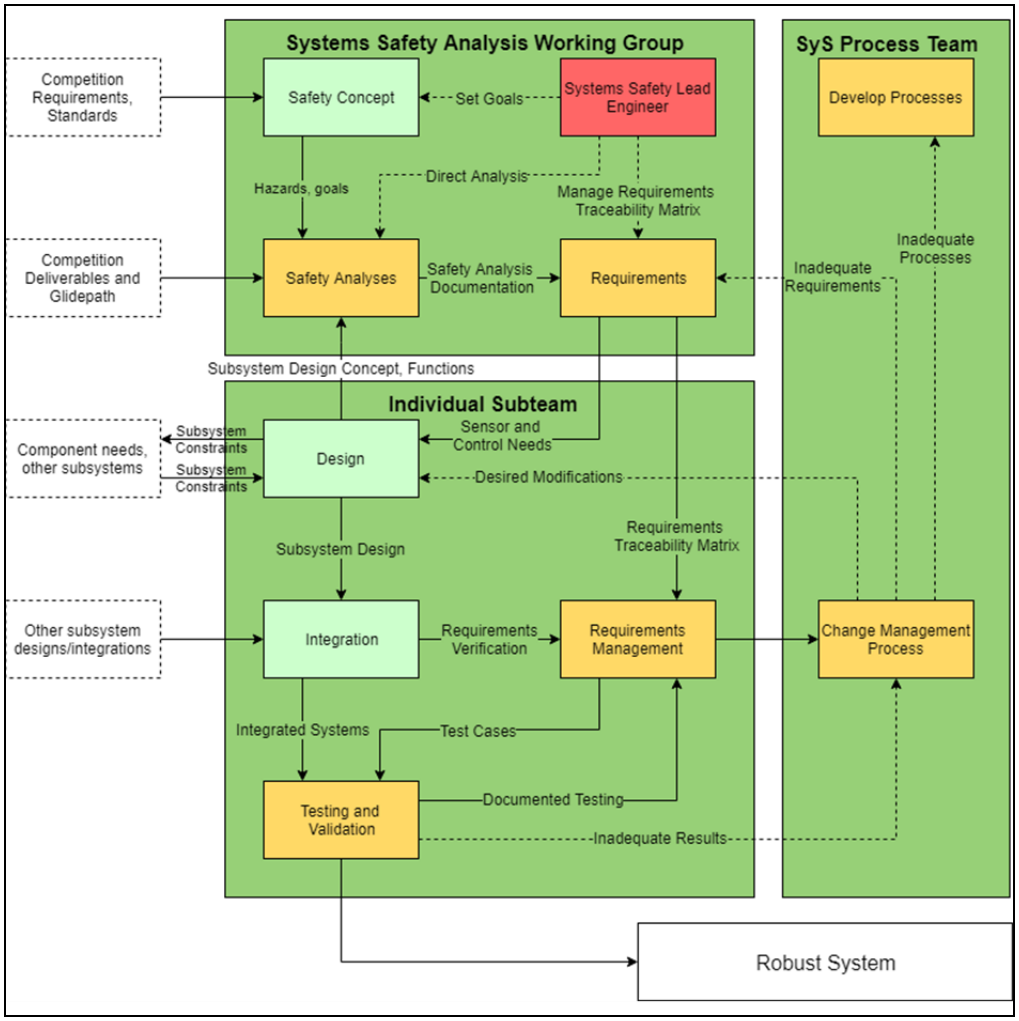


Figure 16: Individual Sub-Teams and Systems Safety Working Group co-develop Functional and Safety Requirements

3.2.4 Unintended Vehicle Acceleration System Level Requirement

EMC places an emphasis on team's ability to ensure safe vehicle operation during testing events. One of the key safety criteria for on-track testing is for teams to prove through analysis and system design that the vehicle is never able to accelerate without a user or an active safety-controlled request. Emphasis is placed on using the Accelerator Pedal Position (APP) and Brake Pedal Position (BPP) signals as the only means to request positive or negative acceleration. These signals are gate-wayed by the HSC to ensure it is the sole requester for all on-board ECUs.

An additional requirement is to ensure the EV systems are de-energized during a key off or E-Stop event.

Further for team added EV To ensure mitigation strategies are in place for such an event. Sub-systems are analyzed for unsafe actions using the Systems Theoretic Process Analysis (STPA). Single Element Fault Analysis (SEFA) as well as Design Failure Mode and Effect Analysis (DFMEA) are various types of systems safety analyses used for different sub-systems interactions. The team utilized SEFA extensively, which is a spreadsheet inspired take on the on the Fault Tree Analysis (FTA). [48] This analysis is conducted through deductive thinking. The idea behind the analysis is to identify any unsafe resulting states, because of an element (component) failure, and the diagnostic/mitigation actions that must be in place to avoid an unintended acceleration event. In the example below, shown is the analysis carried for one component. In case of a failure of the HSC, all team added EV components are at a risk of being impacted, as well as systems level interaction control with GM stock systems. This requires that no other components can request torque from the EV components, and risk of an energized ESS, is mitigated through the UWAFT-supplier agreed upon resulting state, which is to open contactors. This is possible as the ESS requires a voltage on the Dig I/O discharge_enable pin. This for instance is documented within the ALGO-BAT-9 within the RTM. The unintended acceleration analysis is applied to all ECUs that can either request (e.g., Intel Tank), command (e.g., HSC) or actuate (e.g., inverter) torque production.

Documentation ID in the Requirements Trace-ability Matrix

component analyzed | affected components | Impact, Resulting State, Hazard and Diagnostic Method

Item Number	*"Operating Scenario"*	HSC	CAV SC	EV Thermal	ESS Thermal	Engine	ESS	Inverter	Trans P	Trans R	Trans N	Trans D	Trans L	**Impact of Element Fault** (Impact prior to any remedial or mitigation actions)	**Immediate Resulting State** (State of the system prior to any remedial or mitigation actions)	**Potential Safety Hazard(s)**	**Diagnostic Method(s)**	**Mitigation Action(s)**	**System State After Mitigation**	**SAFETY ID**
	Normal Operation	1	1	1	1	1	1	1	1	1	1	1	1	NORMAL OPERATION		NONE	NONE	NONE	NONE	
HSC	HSC	0	0	0	1	1	0	0	1	1	1	1	1	No hybrid propulsive capability. No diagnostics feedback.	Complete EV (12V+HV) inoperable.	Loss or degradation of acceleration; loss or degradation of propulsion (e.g., stall)	No feedback from HSC through CAV/HMI system, (Physical) No power to team added components (lights etc)	HV will deenergize. System will default to engine-only, user will still have control of engine.	Engine-only, no team-added equipment available	USER-HSC-1, ALGO-BAT-9, ELEC-RCM-1

Figure 17: Sample Singular row of the Single Element Fault Analysis shows HSC Operating Scenario, Diagnostics Measure and resulting Safety IDs for the RTM

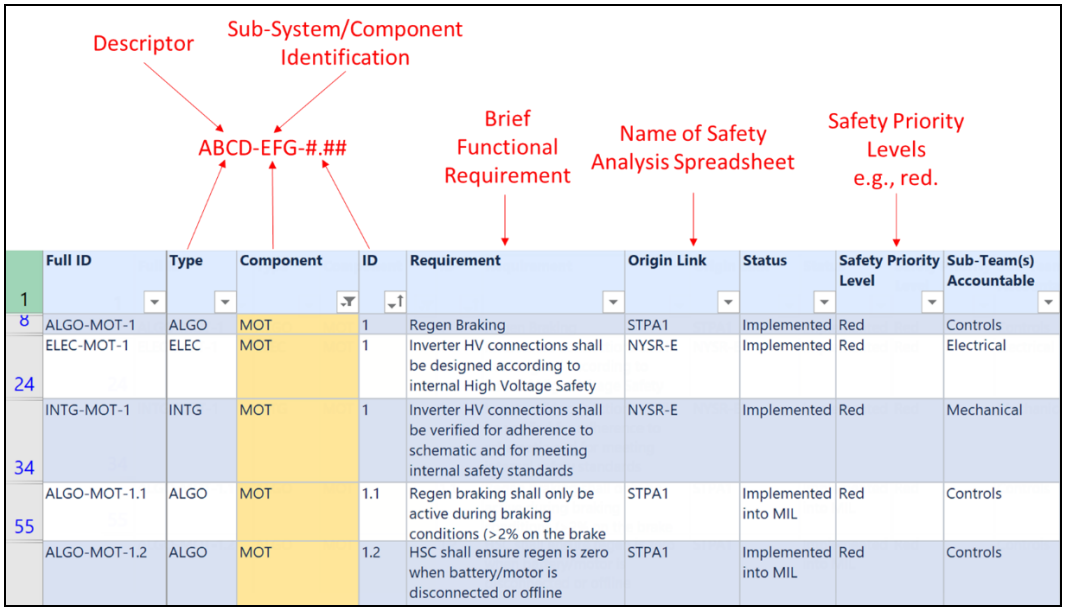
SEFA is applicable for development of safety requirements outside of software, such as in the development of the electrical or thermal systems in the vehicle. That appropriate diagnostics exist and resulting system states are understood, to mitigate from any unsafe system *ripple* effects. An example of this for instance is that, if two EV components share the same cooling loop such as in the case of UWAFT's inverter and motor. Then the peak temperature limit requirement of the cooling loop is dictated by the component with the lower of the upper limits of the two components. And as such SEFA must consider that upon failure of the coolant pump or overheating – how to go about determining safe thermal limits. The diagnosis of this is made possible through the understanding of the PCM team of the requirements laid out by the integration performed by the Propulsion Systems Integration (PSI) sub-team.

3.2.5 Requirements Trace-ability Matrix

The Requirements Trace-ability Matrix houses all requirements that are generated as part of the V-model development process. These include, the earlier described systems safety, as well as functional requirements developed by the sub-teams. The naming notation within the RTM, takes the form of a hyphenated compound "ABCD-EFG-#. ##". Here 'ABCD' represents system level types, such as Mech or Algo. 'EFG' identifies the component/subsystem for instance Mot or Eng, and the last third of the notation, comprises purely of digits. The digits denote IDs that add hierarchy between a functional requirement from the sub-system level/component level requirement. This allows sub-teams to easily differentiate between the type of interaction with the sub-system/component. This is important since a sub-system may have requirements outside of the software workflow, and those are important. Appendix A - RTM Types & Identifiers shows the RTM Descriptor Types and Sub-system Identifier's categories. In total around ~500 sub-system level requirements are developed for the project thus far, of which ~300 are software oriented, of which ~130 are powertrain HSC oriented.

The RTM remains at the center of the development process for the entire vehicle, as it serves as a singular document that all sub-teams collaborate and update frequently. Since the RTM is not limited to just software requirements, all sub-teams use it to determine the function level expectation of other sub-teams interacting with a certain sub-system. This enables faster identification of dependencies within the requirements development path, across the entirety of the team. For instance, a component like the EDU4 motor must be physically installed 'ELEC-

MOT-1.1', 'INTG-MOT-1' for the PCM team to verify feasibility of regenerative braking 'ALGO-MOT-1, 1.1, 1.2' as shown in Figure 18.



	Full ID	Type	Component	ID	Requirement	Origin Link	Status	Safety Priority Level	Sub-Team(s) Accountable
8	ALGO-MOT-1	ALGO	MOT	1	Regen Braking	STPA1	Implemented	Red	Controls
24	ELEC-MOT-1	ELEC	MOT	1	Inverter HV connections shall be designed according to internal High Voltage Safety	NYSR-E	Implemented	Red	Electrical
34	INTG-MOT-1	INTG	MOT	1	Inverter HV connections shall be verified for adherence to schematic and for meeting internal safety standards	NYSR-E	Implemented	Red	Mechanical
55	ALGO-MOT-1.1	ALGO	MOT	1.1	Regen braking shall only be active during braking conditions (>2% on the brake	STPA1	Implemented into MIL	Red	Controls
	ALGO-MOT-1.2	ALGO	MOT	1.2	HSC shall ensure regen is zero when battery/motor is disconnected or offline	STPA1	Implemented into MIL	Red	Controls

Figure 18: Requirement Transparency & Sub-team Dependency identified through an ID and Sub-team categorization setup

3.3 Expanding MBD for faster environment switching

3.3.1 High Level App Setup

At the highest level lies the project (.prj) file. This launches the MATLAB application which contains information about the included files/description and helps start/end the UWAFT HSC project. The data from the component suppliers are stored in spreadsheet (.csv) and (.mat) format files that are loaded directly into the workspace using getter style methods that are run as part of the model MATLAB scripts (.m). The plant model and controller constants are loaded into the Simulink environment and applied to the sub-systems via the masked library block setup. Simulink model properties such drive cycle, environment, grade, and tStop which is the length for which to execute the simulation are loaded as separate variables in the workspace accessible by the Simulink simulation model.

3.3.2 Model Configurator - MATLAB-Simulink task automation

Over the course of the development of the HSC, time was spent exploring methods to speed up model interaction tasks. The model configurator tool is developed for speeding certain simulation tasks and ensuring consistent build environments. The MATLAB scripting language and model is utilized to encapsulate data for component models, that are represented as masked library component blocks, described in more detail in the next section. The objects live in the MATLAB workspace that are accessed by the masked library blocks allowing simplification of component data and controller parameters initialization. For more procedural steps such as setting up of real-time/non-real time test environments, and I/O interactions, the MATLAB scripting language is leveraged and its powerful handle on properties within a Simulink environment are co-utilized. This approach significantly reduces the setup time needed for transitioning between non-real-time and real-time testing and enables a developer to replicate the simulation environment faster.

The Object-Oriented class structure allows UWAFT to define methods, and properties for the UWAFT simulation model, facilitating functional interaction with the simulation model as included in Appendix B - Model Configurator Script. This enables the team to configure testing environments for HIL, and MABx, provided the limitations of the MATLAB-dSPACE API exposed to developer. MathWorks provides comprehensive documentation on how to go about accessing Simulink model environment elements, without requiring the developer to open the Simulink model manually. Due to the object-oriented nature of the project setup, this setup is referred to internally as the Model Object.

The class contains three main types of methods as shows under the Class banner in Figure 19. Firstly, the object constructor that loads the model system, secondly the SIL simulation interaction methods (e.g., running the model) and thirdly the configuration methods that are used to initialize simulation data into the workspace, as well as allowing the simulation model variant to be pre-configured for target HIL hardware or MABx flashing. The drive cycle property of the model object is modified through the 'loadHighway()' method, which in this case would load the UDDS Highway Drive cycle. Similarly the initialization 'init()' method contains other getter style methods, that fetch the component model data, controller parameters and simulation time in order to prepare the simulation to be run. Appendix C- Model Based Design

Framework Overview provides a framework level illustration and interactions of the model configurator in the development process utilized by UWAFT.

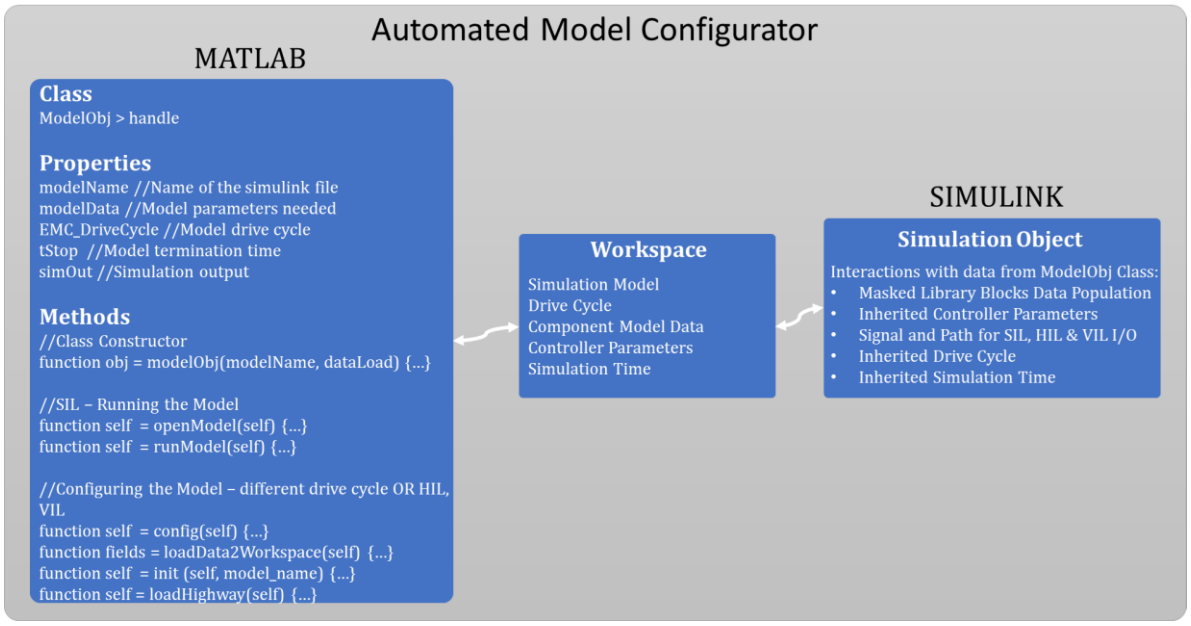


Figure 19: Automated Model Configurator Tool Setup through use of Object-Oriented Class

During year 1 of EMC, a conscious effort was made to in developing and setting up the masked library blocks [49] for component models, controllers, and the driver block among other subsystems within the modelling repository. The motivation behind this was twofold, firstly during year 1 of EMC the team needed a method whereby the component parameter data could be easily swapped for powertrain architecture selection studies and secondly, once the components were finalized the team would work with the component suppliers to obtain data surrounding various aspects of the component, allowing for retention of underlying physical model representation. Sub-systems that would operate using parameters, constants, efficiency maps and/or look up tables were all converted to masked sub-systems. This made the code base cleaner, as once the sub-system functionality was deemed satisfactory - the data within a sub-system could be lumped inside a single library workspace struct. The example shown in Figure 20 below is the EDU4 motor where the 'loadData2Workspace()' method from the Model Object class would call the getter style methods, to load the motor data in to the MATLAB workspace. In this case a pre-existing MathWorks motor model is re-assigned torque-speed,

and efficiency datas within the variable 'mot'. This of course makes it significantly easier to swap models for a component that shares similar physics in a prototype vehicle development environment.

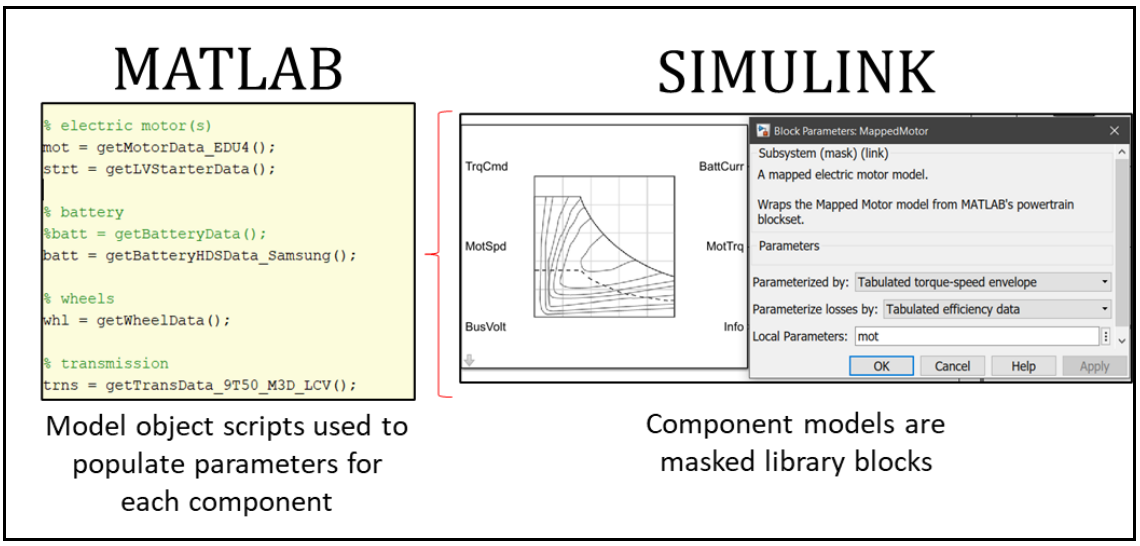


Figure 20: Populating parameters in a Simulink Masked Library Sub-system from MATLAB

MathWorks 'sim' command and 'getparam()' function are extensively used to manipulate the simulation model properties without requiring the developer to explicitly launch the simulation app. This is a unique but rarely used functionality available in the MATLAB-Simulink environment, that UWAFT heavily leveraged to automate set up of the model I/O for flashing code on MABx or for real-time testing in the HIL environment. I/O which is discussed in more detail in section 4.1.4 Hardware I/O setup of this work, utilizes the 'from' and 'goto' tags as sub-system input and output. These are accessible by name for removal and/or modification, using the 'getparam()' MATLAB function. This among other commands such as the RTI dSPACE interactions [50] are programmed within the 'config()' method as part of speeding up of environment setup. For example, to launch the HIL testing environment, a developer would simply pass the string HIL in to the 'config()' method. This would clean up the current working model, call the dSPACE RTI library, comment out the controller block to prepare for outputting

of the .sdf, file and make the necessary I/O changes for plant outputs to be redirected the HIL hardware. Once completed a new Simulink model variant would be loaded, which is pre-configured for HIL flashing. Shown in Figure 21. below is representation of the I/O blocks in the Simulink model for SIL, versus when setup for HIL. Lines 88 to 224 in Appendix B - Model Configurator Script shows the code written to achieve this.

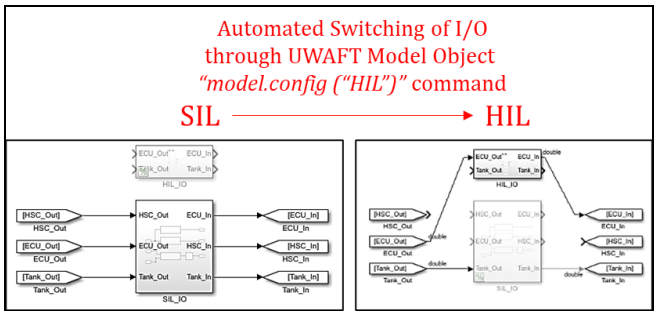


Figure 21: Switching of SIL to HIL I/O - automated for Target Hardware

3.3.3 Collaboration Through Atlassian Products & Version Control

The development of the UWAFT HSC modelling repository over the last 3 years of EMC has been a breadth heavy software endeavor. Multiple developers and component experts have worked collaboratively to develop and test a variety of software feature sets. From a project management perspective this poses a significant risk, warranting a need to ensure development remains organized, appropriate version control/approval mechanism are in place, and much importantly through the COVID-19 pandemic, that their remain transparency in workload assignment. Outside of purely the development environment UWAFT utilized Atlassian Jira [51] for developer ticket assigning and git [52] for version control.

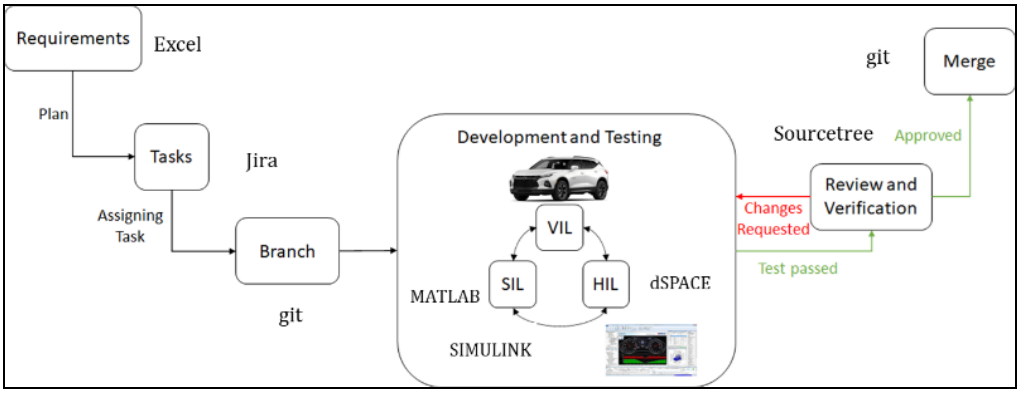


Figure 22: Feature Development, Testing and Merging Workflow

As described in earlier section requirements that are generated from the RTM need to be developed, tested, and merged in the master copy of the development repository. To facilitate this UWAFT's development process is supported by a Jira for task creation, assigning and linking with documentation. Whereas git is used as the main tool for version control, review, and verification process. To break the project in bite sized chunks throughout the year, UWAFT adopted the Agile Sprint methodology to track development progress, and burndown rates. The bite sized chunks are the 2-week Agile sprint, of which there are a total of 18 throughout the school calendar year excluding Winter and Spring breaks. The developers would use pre-set bi-weekly dates over the course of the year to determine workload in the form of JIRA tickets that would be placed on the KANBAN board for the sprint in question. The KANBAN board splits tickets into columns, that signify the status of the tickets. Any ticket created in the Jira system can be assigned directly to a developer. Developers can attach supplemental files, story points, and add descriptions to the ticket. This gives tasks lineage and can be brought up in the future for discussion.

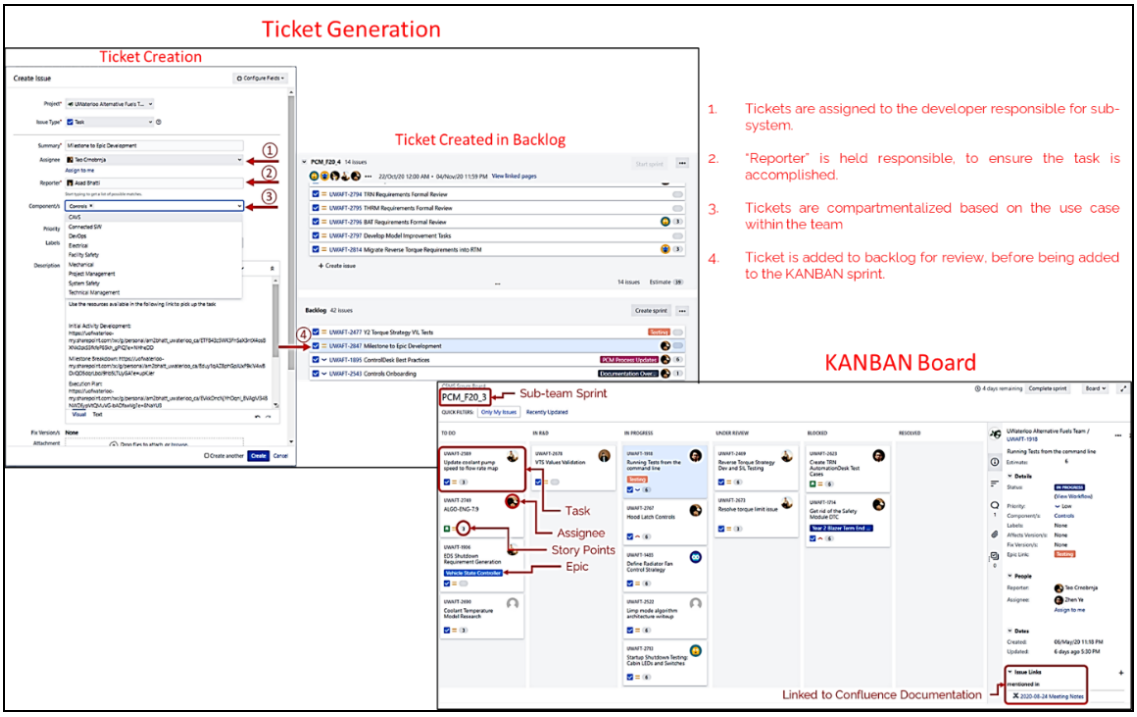


Figure 23: Ticket, KANBAN Based Development Workflow using Atlassian Jira

Not only do the senior developers need a mechanism to review code, but version control provides a mechanism to separate development and testing of feature requests through use of branches. Tickets created in the JIRA system are automatically assigned a unique ticket number. The ticket number is then used as the name of the feature branch, in which development occurs. This allows the reviewer developer to be able to go back to the requirements established within the original ticket and ensure the criteria for completion of the ticket are met.

Chapter 4
Hybrid Supervisory Controller

This chapter deep dives into the architecture of the software sub-systems that make up the Hybrid Supervisory Controller. The organization is systematically organized to facilitate testing of requirements, functional controller layer both on vehicle and sub-system level as well as plant modelling. Also described is the I/O layer that is strategically segregated depending on the target hardware for the simulation testing. In total there are 10 software sub-systems that include the driver block, I/O layer, tester block, the functional supervisory controller, the Soft-ECUs and the plant model that make up the HSC. The main hybrid controller in the HSC is called Functional Supervisory Controller (FSC), this comprises of the fault detection layer, vehicle state control, torque strategy block and the component level execution.

4.1 HSC Development

Historically teams that succeed in past AVTC offerings, are ones that heavily leverage their SIL, HIL environment workflow. This acts as a reliable surrogate to real vehicle development reducing time, and ensuring safety requirements are met, and no errors exist in the logic before VIL testing. The environment must be structured such that a beginner, and/or more experienced developers on the team, are able to get up to speed and configure with relative ease. In this sub-section of the work is discussed how model-based design was expanded on, to support easier SIL, HIL environmental configuration, requirement testing, version control and HSC role(s) segregation. The team's ability to leverage the MathWorks MATLAB/Simulink development environment was key in increasing time spent in development and testing in SIL and HIL over VIL. This makes sense from a cost perspective but was especially crucial for the team during the COVID-19 pandemic, for the duration of the EMC year 2, 3 (2019-2021) development phase, where access to the garage was limited for students working remotely. The developed framework encouraged the team to rely on tools for adding and testing functionality but also to streamline workflow around the requirements.

4.1.1 Structure of UWAFT Simulation Model in Simulink

In a team environment, reducing the time taken to train and bring new developers up to speed is critical. Just as important is the ability of a developer to understand code structure and roles of the various subsystems. There was a need to partition the codebase in smaller functional blocks that serve distinct functions. Simulink offers a visual MBD style of programming approach. Here, systems are essentially drawn on to the screen as block diagrams. These block diagrams are interconnected through signals that can be combined like a harness in signal buses and selectors. This approach makes it easy to monitor signals and backtrack any simulation level faults. Signal propagation can then be traced through use of Simulink library components such as the scope, and data inspector. This approach is a well understood, and popular approach to programming in the current age of automotive systems development, thanks to the ease of code generation for target embedded hardware [53].

Having developed 4 prototype vehicles using a combination of MathWorks Simulink and dSPACE products, the team has obtained valuable experience that was ported in to the development of the UWAFT's EMC HSC [54] [55] [56] [57]. Simulink as mentioned earlier is fully integrated with MATLAB and the data within the project's active workspace. For the latest AVTC offering, UWAFT has elected to segregate the I/O, FSC and plant model. Before we can breakdown individual sub-functions, it is important to discuss the roles of the various blocks shown in Figure 24.

The Simulink model at its root follows the forward modelling approach where acceleration commands are generated through the longitudinal driver, the controller and the plant that results in vehicle longitudinal velocity. The tester block serves as gateway point for controller and plant outputs such that overriding system signals and asserting requirement checks for various components and sub-systems is possible. This gives the team the ability to inject intended faults for assessment of controller, and soft ecu behavior. as part of the HSC feedback loop. The HSC handles state estimation, fault detection, vehicle state control, propulsion torque strategy, and component level execution. The GM Blazer block contains the soft-ecu representation of all ECUs for state estimation purposes, as well as the plant model of the various drive components such as the ICE, transmission, gear reduction units, motor, ESS as well as longitudinal dynamics of the vehicle body.

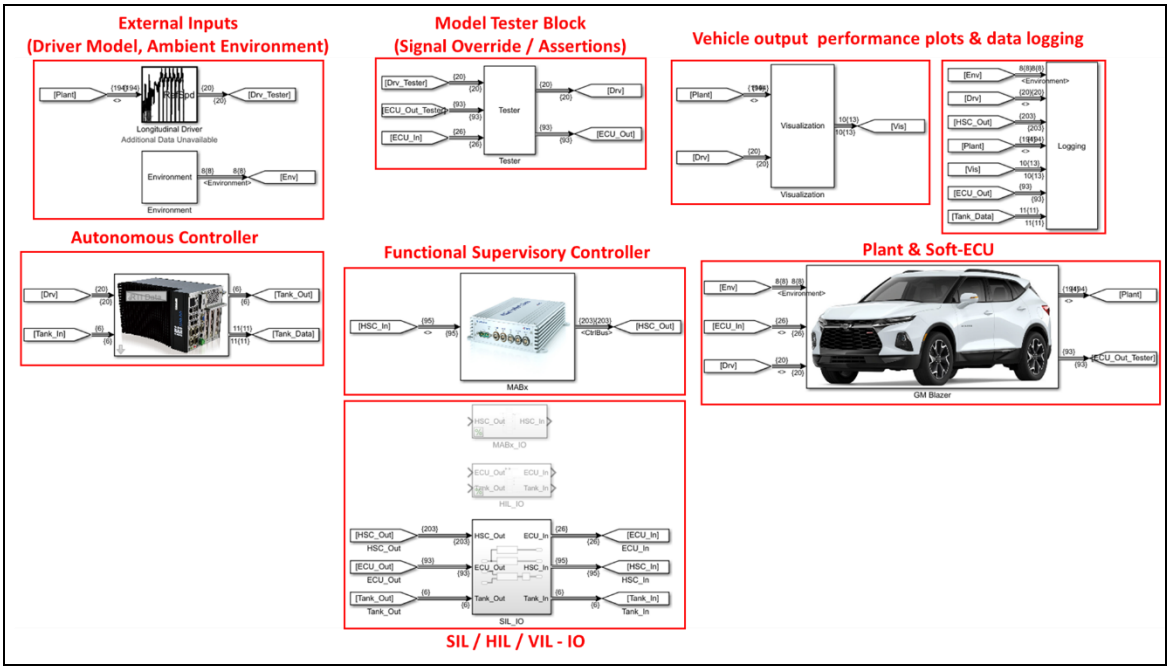


Figure 24: Root level Sub-Systems in the Hybrid Supervisory Controller Simulink Model

4.1.2 Model Tester Block & Simulink Test for Requirement Maintenance

In chapter 3 we discussed the significance of developing safe and functional requirements for vehicle controls development. However, as the code base grew, and more components/functionality were added to support new feature sets. The team felt a challenge in ensuring all requirements were still being met. The team was encouraged to find a method whereby requirements could be tested altogether within the same environment without the need to manually test for each. Regression options such as Jenkins - were explored however due to limitations in developer manpower, continuous changes being made to the model itself, and the limitations of the COVID-19 pandemic, this option was not chosen. By the end of Year 3 the team had amassed 132 requirements within the PCM swim lane of which some are safety critical, and others purely functional. To support simultaneous testing of said requirements, a new testing framework was introduced in the root directory of the UWAFT controls simulation model, that would check for all test cases during an SIL simulation. The tester sub-system shown in the top center of Figure 24 is made up of two parts. The testing override block which overrides signals for testing purposes and the test cases block, where the requirements have

been converted to test cases to verify that the testing requirements have been met. This is organized into various requirements based on the component being tested.

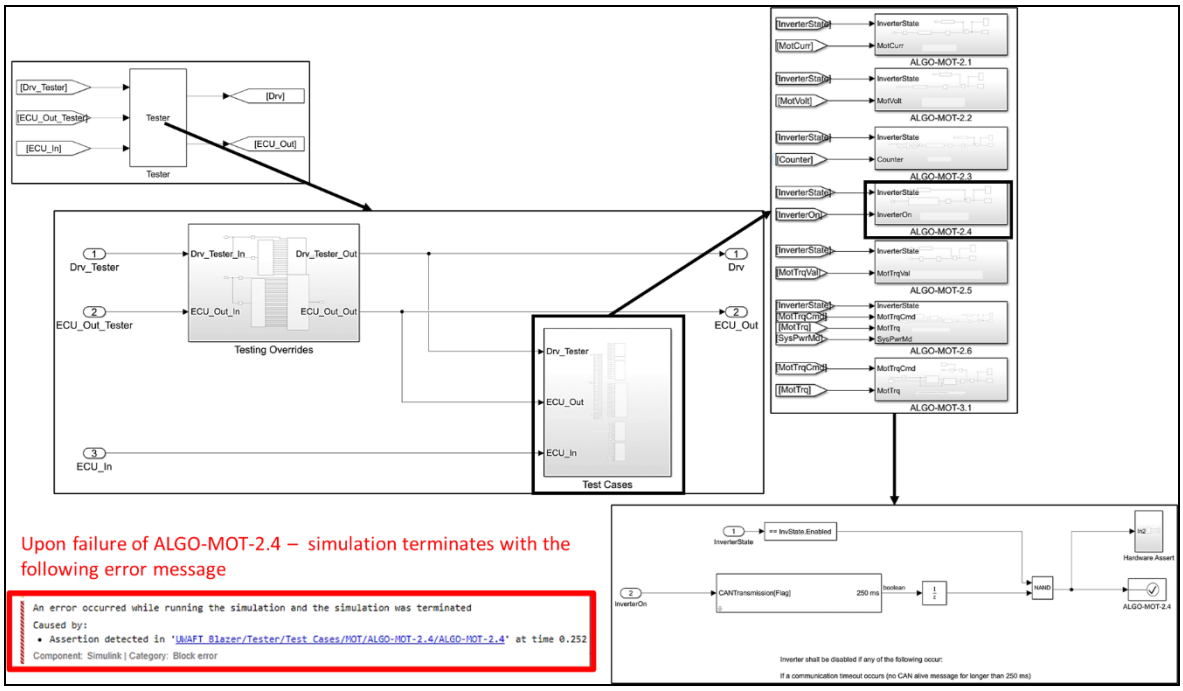


Figure 25: Tester Block & Testing Framework for requirements in the HSC

The inner structure, high level organization and working of this tester block is shown in Figure 25. For example, in the case of the motor controller - the requirements being tested shows that the inverter will be disabled if a CAN communication timeout occurs. During normal operation this should be caught by the Fault Detection layer in the HSC that is described in section 4.1.6.1 Fault Detection Layer. This is an example of a requirement that would ensure the vehicle does not accelerate unintended based on the safety analysis described in Chapter 3 of this work. Upon failure of any requirement, the simulation ends with the error message highlighted in red. The organization and testing of the testing requirements themselves is implemented within the Simulink environment and is organized through use of Simulink Test. [58] Here a singular requirement is linked to a singular test to ensure that each requirement is fully tested individually. This organizational decision allows the team to use the Simulink Test Pre-Load window where a generic test struct function is called and signal values are overridden, followed by the signals that need to be overridden for the test to fail and assertion to occur.

4.1.3 External Inputs (Driver Block & Ambient Environment)

During run-time of the SIL model, there exist two types of external inputs to the HSC and Plant. The driver input block, which uses the drive cycle as a target speed such that it is fed to the HSC as an acceleration request, and the ambient environment block. The driver input block utilizes the Charles McAdam [35] predictive driver block which takes in to account the road gradient and vehicle feedback speed to generate an acceleration or deceleration request. Driver cabin controls such as team developed safety switches, and active safety (autonomous) switches are also present for toggling. The cabin lights and switches are essential for immediate toggling of certain hybrid and active safety software features that can request longitudinal/lateral acceleration for safety purposes. These are documented within the RTM. The Simulink environment imports the drive cycle, and ambient environment parameters from the MATLAB environment details of which are discussed in Section 3.3.2 Model Configurator - MATLAB-Simulink . Shown below is the internal structure of the longitudinal driver model.

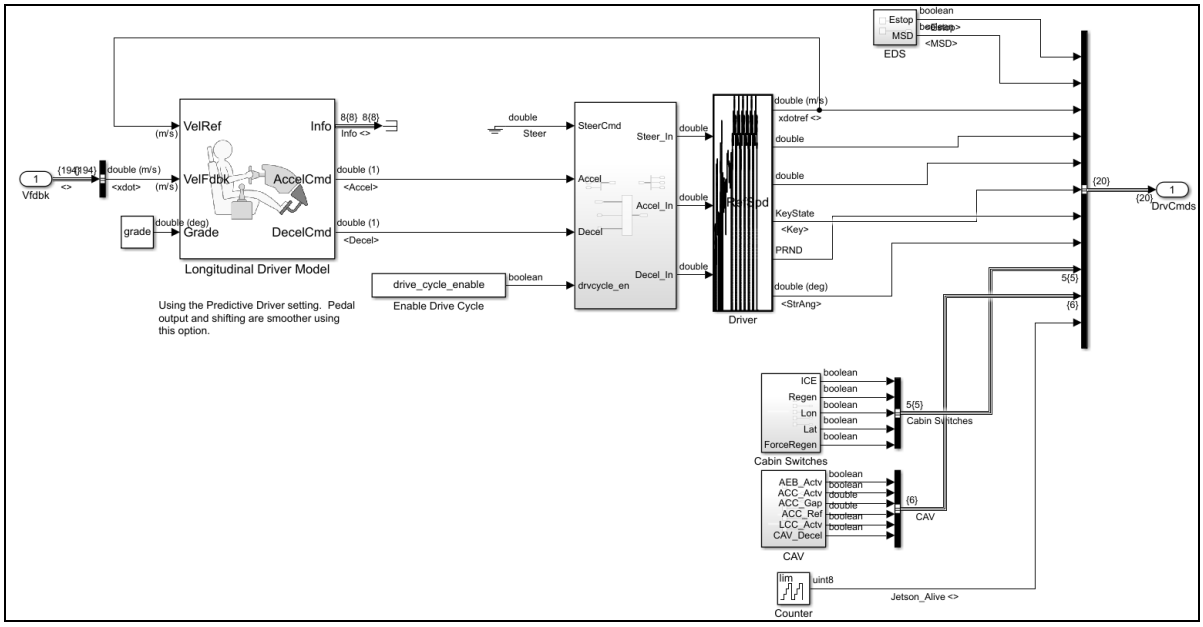


Figure 26: Longitudinal Driver & Team Added Safety Switches Block

The environment block contains ambient information such as grade, wind velocity, external outside temperature, and barometric pressure. These parameters are useful for certain simulation scenarios and are fed into the model where extreme external conditions are

required to find system limits – such as high temperatures and road grade for thermal system testing.

4.1.4 Hardware I/O setup

The I/O layer acts as a signal conditioning layer, where all the signals stemming from the HSC are adapted for the end hardware. In our discussion of the role of using Model Object in section 3.3.2 Model Configurator - MATLAB-Simulink . We have preliminarily touched on the advantages of initialization the model as an object. How that provides the ability of using the model to manipulate signal switching to adapt the simulation to be run in the SIL, HIL and VIL environments. From an MBD standpoint, while there are no explicit requirements for the implementation of the hardware I/O sub-system. The sub-system serves as the only place within the Simulink model, through which all signals must route in to and out of, before being sent to the plant model, HIL or VIL layer(s). Figure 24 shows the Hardware I/O sub-system at the bottom of the root of the model. Here currently only the SIL_IO block is active and uncommented. This is by default and is always the case when the model is initialized as can be seen in the MATLAB implementation in Appendix B - Model Configurator Script line 88 to 100.

From the perspective of the HSC subs-system all signals being received by the HSC from the ECM, BCM, Intel Tank, inverter and coolant pump are Rx signals. The vice versa is true for Tx signals. For the SIL simulation it is important to mimic the nature of feedback, from the real vehicle since the simulation model runs on non-real, CPU time. In that to simulate a unit delay, a propagation delay from the signal system buses, for every discrete time-step that the simulation is solved for, is made possible by adding 1/z discrete unit delay block [59]. The unit delay is only required as part of the Rx feedback, and generally is appended to the simulated plant outputs before being received by the HSC. The unit conversion or matching layer is where signals are first converted to the appropriate type such as double or Boolean, and then are assigned the appropriate signal name.

Signal names are importantly kept consistent throughout the model depending on the system being interacted with, and modifications made to the signal. In the case of SIL I/O, the signal names are represented, per the naming convention of the signal messages described within the CAN database (DBC) file. Due to confidentiality reasons, the message and signal name descriptions are not provided in this work.

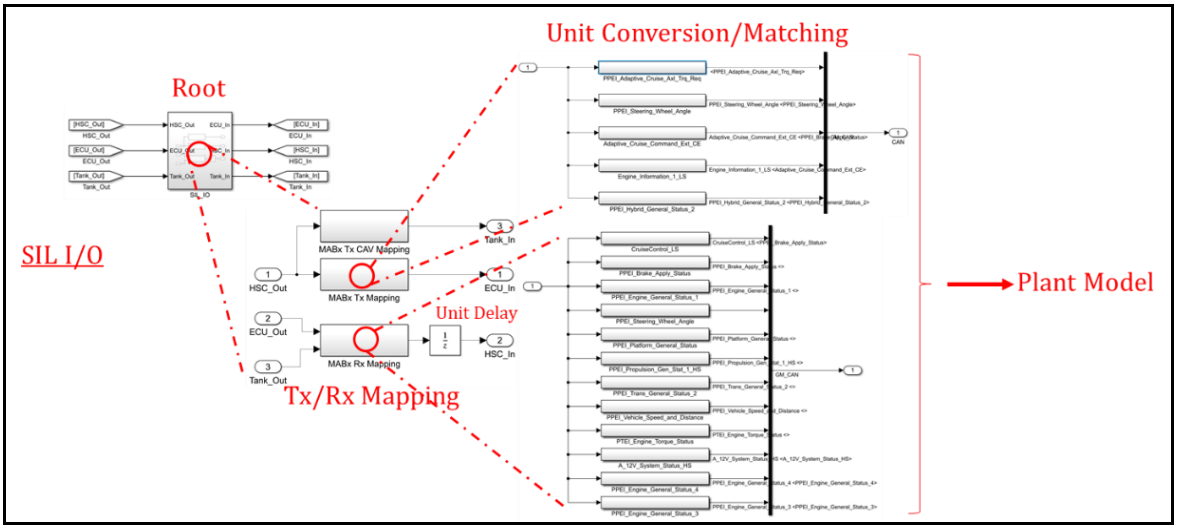


Figure 27: SIL I/O layer

Expanding the idea lines 89 to 250 in Appendix B - Model Configurator Script showcases how the Simulink model is configured for use with the HIL, and VIL environments. This is made possible through a series of scripted commands, that follow the manual setup pattern if a developer were to perform all steps manually. In HIL simulation the plant model block is commented out, as the HIL takes on the role of representing model for real-time simulation. Here the CAN, Analog and Digital outputs are physically mapped to the real HYP, and ZIP pins on the HIL and MABx respectively. The port location and harness development for the pins is described in dSPACE documentation that is provided as part of the purchase package of DS1401/1513 MABx II hardware. Working with hardware that is needed is simply uncommented and signals are re-routed, through use of the Model Object script. This is done through the command line, as part of a string parameter passed to function call. It should however be mentioned that there are other steps involved in setting up dSPACE hardware such as approving dSPACE licensing, using description files for CAN settings for signal population, and utilizing the CAN Multi-Message (CANMM) block-set [60] to generate the .sdf file type, such that it can be flashed on the target hardware. Shown below is the I/O setup for HIL testing established at UWAFT, that is an easily repeatable setup for UWAFT. Setup of the VIL layer is

very similar, except in that case ALL vehicle CAN signals are populated as part of the CANMM setup - to be received by the MABx.

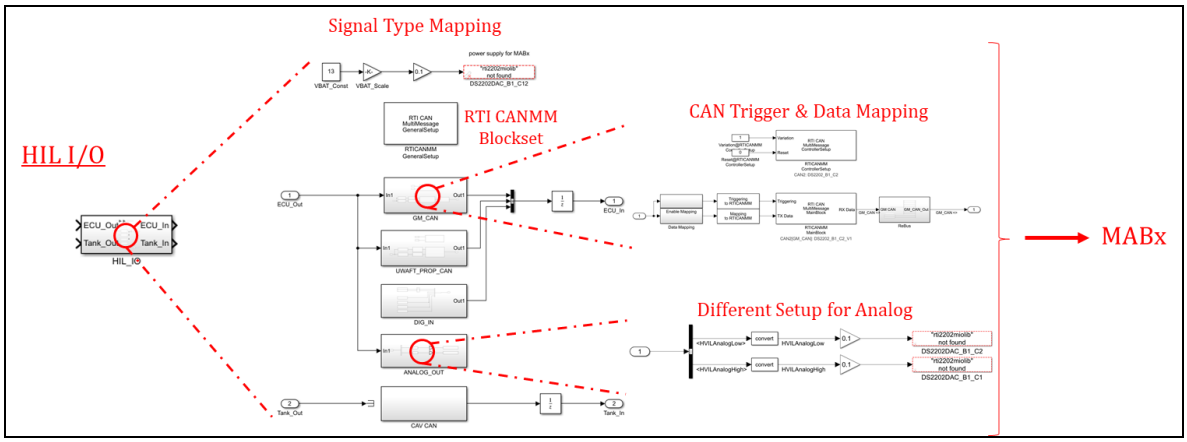


Figure 28: HIL I/O layer

4.1.5 Longitudinal Controller (Intel AIOT Tank)

Addition of a secondary torque requesting controller is a new AVTC requirement, that didn't exist explicitly pre-EMC. The Intel AIOT Tank is a physical onboard Linux compute systems that runs within the UWAFT Blazer as the primary CAV controller. This system runs the Robotics Operating System (ROS) written in C++ programming language that utilizes nodes to separate functionality. Functionality of the tank includes but is not limited to – performing CAN Tx/Rx, data filtering, sensor fusion, object association/detection, and updating safety counters within and external to the system. Representation of this controller in the HSC involves processing CAV-alive safety counters, driver cabin user selection for ACC modes/gap settings and using the pre-processed sensor inputs from the tank to generate a torque request.

The longitudinal controller incorporates an adaptive cruise control model that works based on its understanding of the lead car and the Ego vehicle. The HSC in the case is following acceleration commands that are generated from the CAV compute unit where the drive cycle acts as the lead vehicle, and Adaptive Cruise Control (ACC) ego-controller drives autonomously behind it. The lead car block generates the relative speed and relative position between the two vehicles, as inputs to the ACC block. The team is currently exploring other active safety features such Automatic Emergency Braking (AEB) and Lane Keep Assist (LKA).

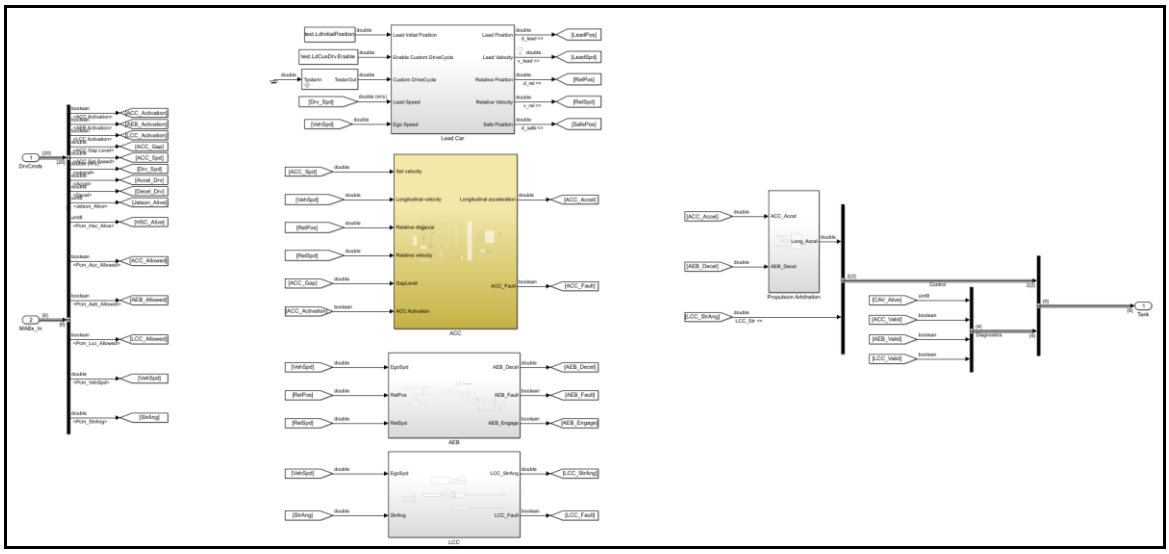


Figure 29: Longitudinal (Active Safety) Controller Layout

4.1.6 Functional Supervisory Controller

The Functional Supervisory Controller (FSC) is at the center of all vehicle operations and acts as the master arbitration controller for all team added components and ECUs. Before the FSC was developed, a fair bit of thought was put into the structure and layout of the controller. The UWAFT team wanted a controller structure that was agnostic to the plant model it was actuating on. Meaning that regardless of the hybrid architecture this structure could be re-used or expanded on to control more components if needed. The basic structure of the HSC is made up of 4 main layers. The State Estimation & Fault Detection layer, Vehicle State Control, Propulsion Strategy Optimization and Component Level Execution.

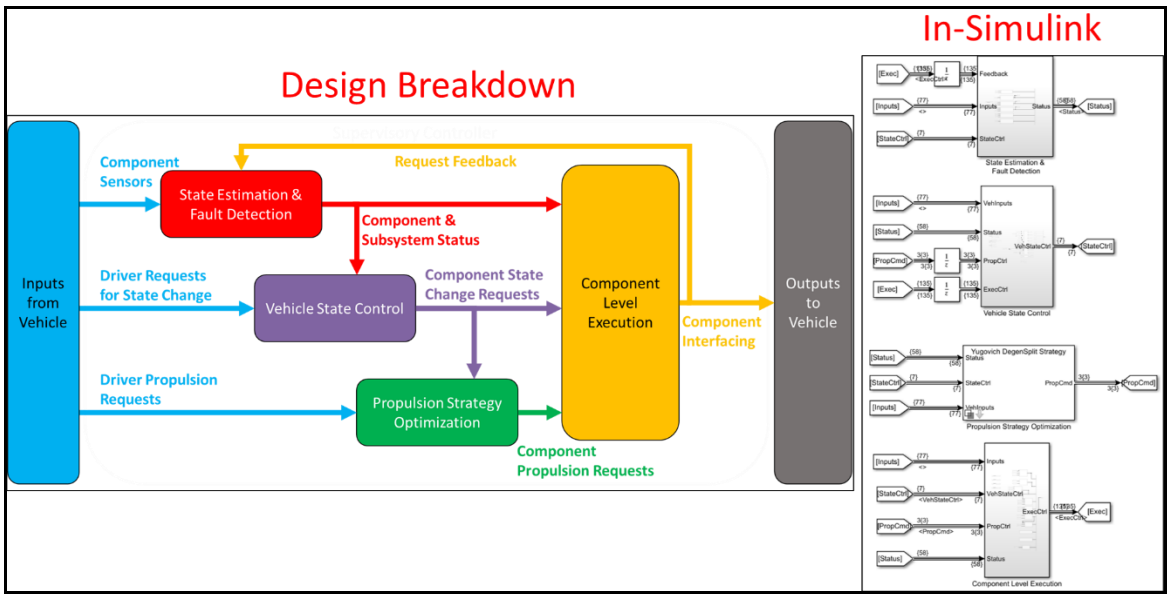


Figure 30: Functional Supervisory Controller Structure & Segregation of Roles

4.1.6.1 Fault Detection Layer

The state estimation and fault detection layer take inputs as feedback from the plant during a SIL simulation and, the CAN bus from the component ECUs. The inputs are measured component states, and physical parameters that are compared against physical limitations and unsafe component control combinations as deduced through systems safety analysis as well as data sheets provided by the component manufacturer. The goal of this layer is to produce Boolean flags that are evaluated at vehicle state control and soft-ecu level, to ensure vehicle operates in a correct safe state.

Faults are determined at this level to ensure two things. Firstly, that the vehicle is not requesting torque from a component that is operating near its limit, and needs to be disabled, but secondly to ensure the vehicle state control and component level execution subsystems are able to determine the appropriate operating state such as ICE only, HEV or fault mode. The table below summarizes the various checks performed at this level.

Table 4: Functional Requirements Tested at HSC Fault Detection Layer

	Component			
	Inverter	**Engine**	**ESS**	**Autonomy**
Functional Limit Requirement (Limit Unit) (Failure Evaluation)	HV Bus Active Discharge (V) (<50)	Coolant Temp (ºC) (>=110)	Minimum Voltage (V) (<=321)	Accel Override (Nm) (Driver>ACC)
	Running Counter (s) (>=0.5)	Low Voltage (V) (<=10.5)	ESS Discharge Enable (Bool) !(Discharge_Enable)	Accel Limit (m/s^2) (>0.3)
	CAN Signal Integrity (Bool) ! (CAN_VAL && CAN_MotSpd && CAN_PhaseCurrent && CAN_DcLinkVoltage && CAN_MotTemp)	Brake Press (%) (>=5)	BMS Internal Error (Bool) (Cell Over Voltage || Cell Under Voltage || Cell Over Temp || Cell Under Temp || Cell Over Current || Cell Under Current)	CAV Compute Alive (ms) (>250)
	Motor Current (A) (>=424)	CAN Timeout (ms) (>=170)	CAN Time Out (ms) (>1500)	CAV Switches (Bool) !(ACC Enable)
	Motor Voltage (V) (>=450 || <=321.6)	Over Speed (RPM) (>=6850)	Isolation Error (Bool) (False)	Vehicle Speed (kph) (<40 || >140)
	CAN Timeout (ms) (>=1250)	Max Torque (Nm) (MaxEngTrq -PlantTrq > 0)		
	Motor Speed (RPM) (>=13000)			
	Motor Temperature (ºC) (>=100)			
	Low Voltage (V) (<=10.5)			

Note that the failure evaluation condition for CAN timeout or signal integrity timers vary based on the criticality of the parameter being evaluated. For example, a motor over torque

signal is evaluated more frequently than a motor over temperature signal, because a motor casing may withstand a high temperature with active cooling for a slightly longer time, before permanent damage occurs, whereas a motor over torque condition may snap a torque rated half shaft immediately, and thus needs to be monitored at a higher rate.

4.1.6.2 Vehicle State Control

The primary function of the vehicle state control subsystem in the HSC is to ensure the vehicle is operating in the right state based on outputs from the fault detection layer, propulsion plant model and any executive requests made from the passenger cabin. The vehicle states include Accessory, Off, Startup, On, Fault and Shutdown states. The vehicle direction states determine driving direction primarily based on the position of the transmission shifter, given when vehicle in ON, and not in a fault state. Executive inputs into the state machine are used to determine state of actions that stem from the user cabin, such as state of the shifter and ignition button state. The vehicle state control is developed using Simulink Stateflow [61]. Stateflow as the name suggests, makes use of functional state transition diagrams, flow charts and truth tables for logical decision making within the Simulink model. The state machine can be further layered to accommodate any transitionary states within the main vehicle states. Figure 31 illustrates the structure of the vehicle state machine at a high level. Since the vehicle utilizes a stock GM ICE, the immediate fault state upon an EV related HEV failure is to shut off all EV component and continue operation in ICE only. Full shutdown occurs if the ICE is detected to not be running. In this case a full vehicle shutdown is required.

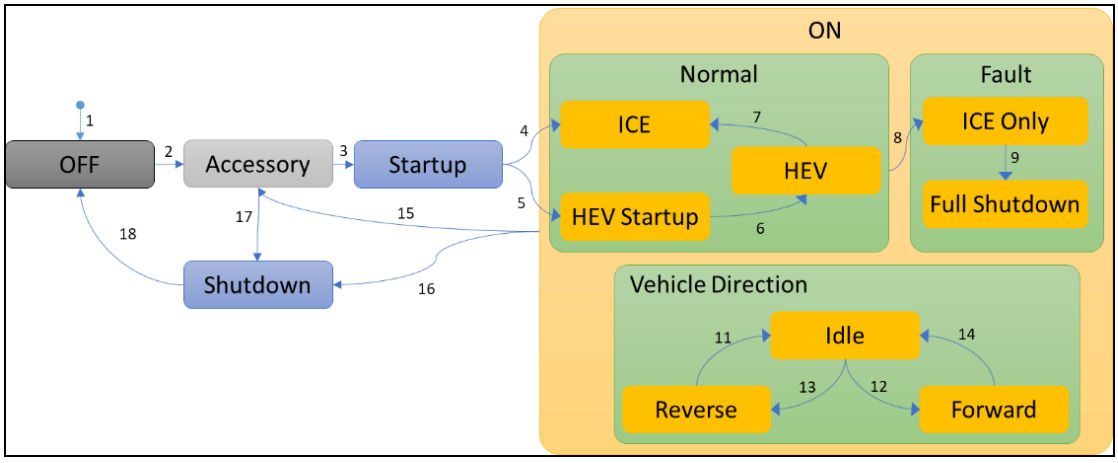


Figure 31: Vehicle Powertrain States

Table 5: Vehicle State Control state description & state transition condition

State	State Action	State Transition Condition
OFF	All Systems Off	1: Entry 2: System Power Mode: accessory
Accessory	System State: Accessory	3: Vehicle Start Request: true; Shift Lever: park 17: !(Vehicle Start Requested)
Startup	ESS Contactor Close: true Inverter On: true EV Cooling: true	4: ICE Request: true 5: HEV Request: true
Shutdown	ESS Close Contact: false Inverter On: false System State: Shutdown EV Cooling: false	18: Engine Ready: false; EV On: false
ON	Limp Mode: False Limp Shutdown Request: False	15: Engine Ready: false; Limp Mode: true 16: Vehicle Start Request: false ; Limp Mode: true
ICE	HEV State: ICE	**Fault state – leaving state unintended - except shutdown
HEV Startup	HEV State: ICE Inverter On: true ESS Close Contactor: true	6: EV Torque Ready
HEV	HEV State: Hybrid	7: ICE Request: true
ICE Only	HEV State: ICE Inverter On: false Limp Mode: true ESS Close Contact: false	8: is EV faulted: true
Full Shutdown	Vehicle State: Shutdown	9: Engine Ready: False
Idle	Vehicle Direction: Idle	12: Shift Lever: Drive 13: Shift Level: Reverse
Reverse	Vehicle Direction: Reverse	11: Shift Lever: Park
Forward	Vehicle Direction: Forward	14: Shift Lever: Park

Table 5 above describes the resulting output action within each state as well as the outgoing condition necessary for the states to transition. Note that state ICE is different from ICE Only state as the input required to be in the former state stems from user input of toggling one of the cabin safety switches to keep the HEV systems disengaged. This requirement originally stemmed from the NYSR and was incorporated here to differentiate from the ICE Only state, where the HEV system is faulted.

Another important thing to note is that in Simulink Stateflow implementation state actions can be programmatically prefixed with the en (enter) and du (during) operators that allow separation of one time and continuous actions, that are not shown in the state table to reduce unnecessary documentation complexity. Furthermore, the state action and state transition are enumerated, for ease of programming, which in this case are written in full words.

4.1.6.3 EV Torque Added Strategy

The HEV torque strategy developed for the UWAFT Blazer is a simple Charge Depleting (CD) strategy, where the front and rear axle operate independently of each other. Meaning that the torque request is not split between the two axles, instead the EV motor provides additional torque based on the accelerator pedal position of the driver. This classifies this torque strategy as a rule-based strategy, where the maps are developed through drive testing on a test track. The intent behind developing a simpler torque strategy was to help the team make the Blazer robust to drive and operate. Since all EV propulsion systems are team added, naturally a need was felt to first explore mechanical, electrical, and thermal systems boundary. Secondly since the inverter and inverter were paired by a third-party supplier, the team wanted to be confident of the torque application behavior of the rear axle.

4.1.6.4 Drive Cycle Requirements

The torque strategy was developed based on EMC's requirements for a drive cycle developed on the team's local track. The track local to UWAFT through the development of year 3 was the Waterloo Regional Emergency Services Training and Research Center (WRESTRC). The WRESTRC is a 1.2 km long oval complex, that the team mapped and adapted to simulate a city and highway drive cycle section. EMC's specific energy consumption course requirements were two-fold. Firstly, to develop a 36 miles long course that was split up in to 4 repeated city/highway profiles designed to fit the test location. Secondly to ensure that the city section above 50 kph with 3, 10 second stops, and highway with speeds above 70 kph and 2, 10 second stops. Shown below is a map view of the WRESTRC, and the points at which complete stops were made. Since the track was 1.2 km length, 48 loops of driving were required. The team decided to drive 12 loop cities, then 12 loop highway and repeat twice to fulfill the drive cycle requirements.

Figure 32: WRESTRC test track map view w/ slow down to stop points [62]

4.1.6.5 EV Torque Added Strategy

The torque strategy pedal maps for forward torque and regenerative braking were developed such that the Blazer could sustain battery SOC when driving the city profile and deplete battery SOC when driving in highway scenario. This would simulate a charge sustaining behavior below 55 kph and charge depletion above that speed for improvement in the overall fuel economy. The pedal maps were also tuned for a natural pedal feel that was subjectively developed based on inputs by multiple drivers.

During the development of the pedal maps, it was determined that upon stopping the vehicle, the regenerative braking map would cause the vehicle to drive backwards due to the negative torque application. A switch case was added to allow no EV below 7 kph and retain the stock vehicle's slow speed crawl feature. It must also be noted that since the UWAFT Blazer is an HEV and not a PHEV, charging on demand was deemed a critical feature. For this purpose, the team added a cabin switch that would request a safe amount of regenerative power for the duration of the drive, helping charge the ESS.

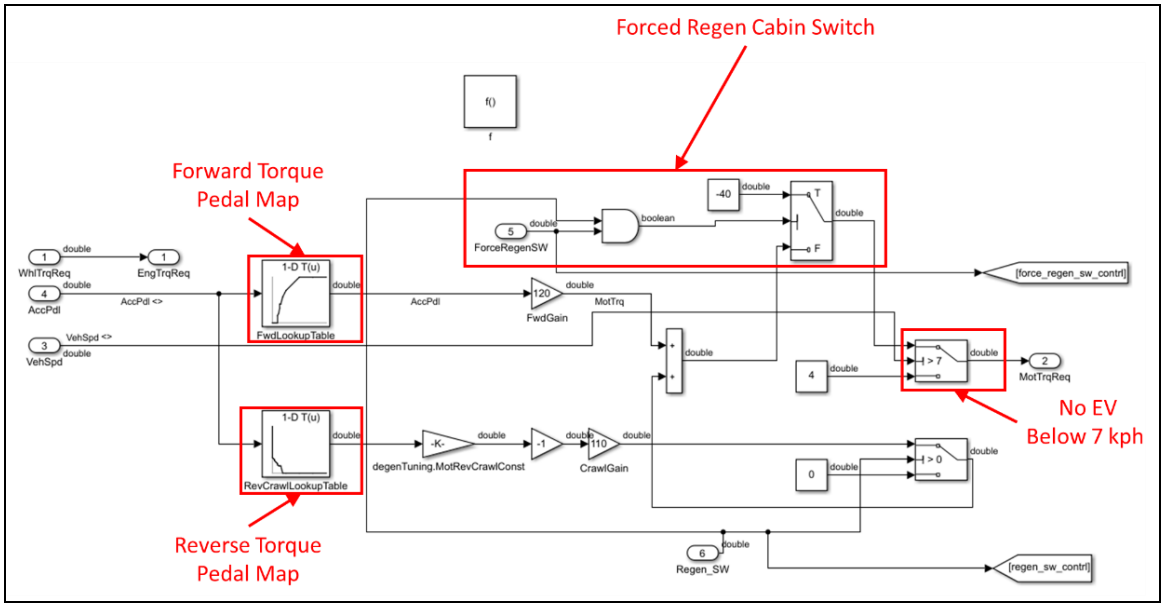


Figure 33: Torque added EV strategy

The tuned pedal map for the WRESTRC, resulted in the ESS being able to sustain SOC at city speeds and deplete SOC at highway speeds for 36-mile drive, as shown in Figure 34 below.

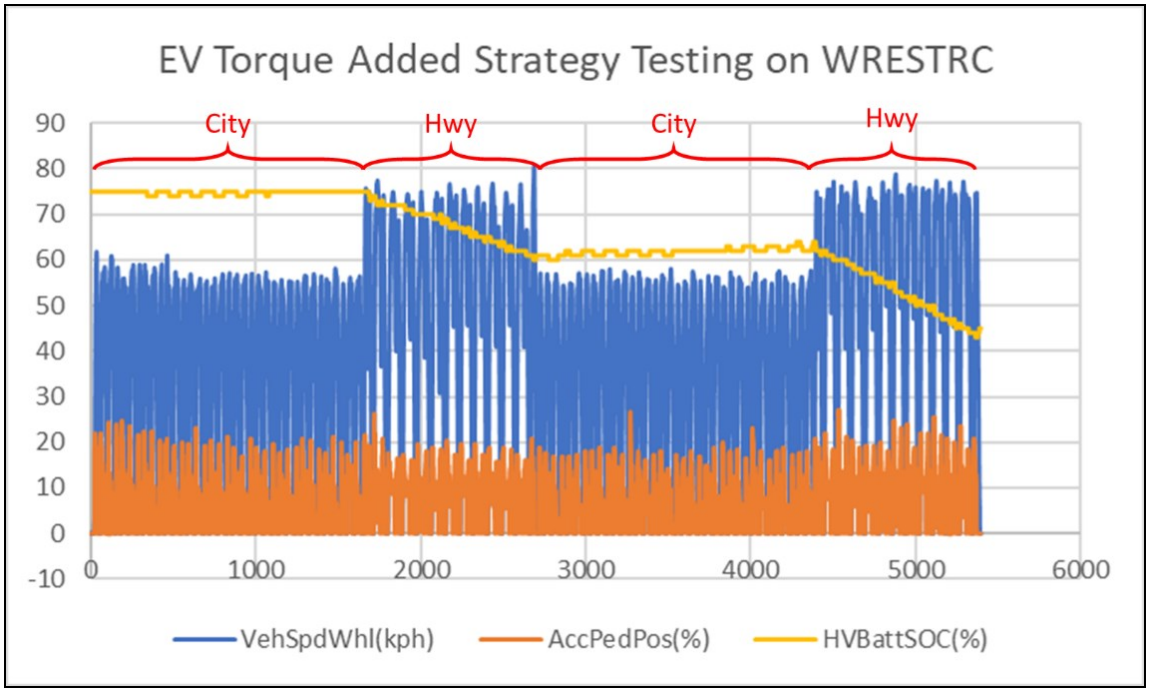


Figure 34: Initial SOC Safety Window Verification at WRESTRC

4.1.6.6 Component Level Execution

The last major sub-system that forms the functional supervisory controller is implemented within the Component Level Execution block. The primary function of this sub-system is tracking of ECU states for the various controllers onboard the vehicle interacting with the HSC interacts. These states are written in Stateflow similar to the vehicle state control layer, except in this case, all physical ECUs present on the vehicle are represented. This is important for a variety of reasons that includes but is not limited to - understanding of the operating states of the controller, determination of actions possible by the HSC, switching of states based on logic in the fault detection and vehicle state control layer, and lastly organizing ECU interaction for ease in troubleshooting in-vehicle. The state determination is made through a combination of in-HSC outputs as well as raw component CAN signals.

This layer interacts with ECUs of the following components – engine controller, body control module, inverter, BMS, in-cabin lights, coolant pump, the stock active safety control module, the rear differential control module, and the relay control module. It must be noted that control units such as the active safety control module, and rear differential control module were modified/removed during the vehicle retrofitting, and for the stock GM systems to perceive normal operation. This layer also populates. the expected GM CAN bus signals so the stock vehicle systems continue to operate as normal.

As an example of the inner working of this sub-system we can take the example of how the inverter soft ecu is implemented. Shown in the image below is the inner ECU states for the inverter, as documented and supplied by the component manufacturer followed by the team implemented representation. It may not come as a surprise that there is a striking resemblance, in the layout, names and direction of state transition arrows. During the development of this layer, the team worked closely with all component suppliers to ensure the inner workings of the third-party ECUs was well understood for safety reasons. It is also important to note that some of the default ECU states are clumped together for ease in troubleshooting of the component. For instance, the Initialization Failed, Error and Shutting down are grouped together in the team's Idle and Shutdown states.

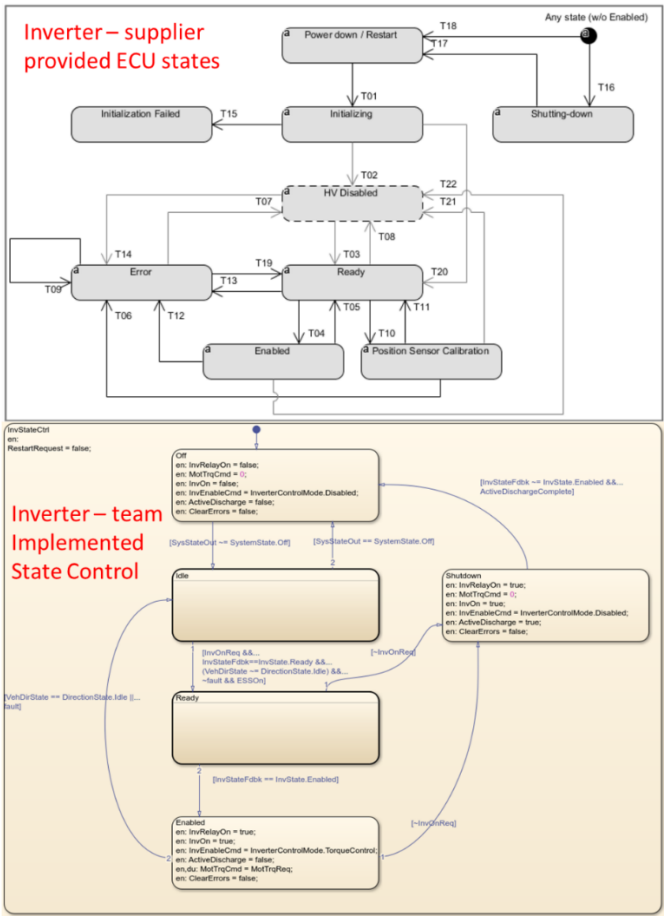


Figure 35: Inverter State Control - Analogous Implementation

4.2 Vehicle Plant Modelling

The plant model is used in conjunction with the functional supervisory controller for testing of all controller functions before the code is ever tested on the real vehicle. The plant model simulates the physics-based behavior of the drivetrain components as well as the longitudinal chassis dynamics. The ECU model adaptation are termed soft-ecu's as they are simplified software representation of the real ECUs operation, that are physically present within the vehicle. In the case of UWAFT the plant model fidelity, and requirements coverage spans the longitudinal dynamical, embedded, and thermal models of the drivetrain components. This is important for the SIL and HIL environment testing of the basic control algorithm, validation of systems diagnostics as well as refinement of the plant model itself based on calibration data collected from vehicle testing.

4.2.1 Soft-ECU Representation

In section 4.1.6.6 Component Level Execution of theses we discussed how component states are tracked and executed based on executive requests from the vehicle, within the functional supervisory controller. However, it is the soft-ECU implementation during a SIL/HIL simulation that mimics the functionally *complete* representation of inputs, outputs, and possible ECU states that the HSC needs to interact with. In some cases, such as in the case of the ICE, the soft-ECU contains the plant model – such that the executive requests can directly be translated to effective outputs. A well calibrated soft-ECU is necessary for scalability of SIL simulation in real-time HIL environment, as these ECU outputs are required for exploration of system boundaries – such as possible component faults, operating states, and physical constraints. The table included in Appendix D - Plant Soft-ECU Inputs/Outputs Summary summarizes the inputs and outputs involved with the soft-ECUs. The signals are further grouped, signifying the routing to the HSC in a VIL test environment, and to plant during a SIL/HIL simulation. The soft-ECUs that are implemented in the UWAFT Blazer plant model are a combination of soft-ECUs provided by the OEMs in the form of MathWorks block sets, and team developed soft-ECUs. OEM provided soft-ECUs include ICE, BCM and transmission soft-ECUs which were provided by GM and MathWorks and were left untouched. In some cases, such as the ICE, these were simplified to exclude the inner workings of ICE that were reported by the ECU such as crankshaft angle, and internal cylinder pressures – which were not useful for the team to keep for the purposes of the longitudinal simulations. The latter (team developed) soft-ECUs include the inverter, ESS, coolant fan, coolant pump, cabin switches and the RCM. These soft-ECUs were developed from ECU behavioral testing and component data sheets studies.

4.2.2 Powertrain Model

The Powertrain model is a single sub-system that contains torque producing and transferring components. UWAFT heavily utilized the MathWorks Powertrain Blockset [63] for the development of this sub-system. The Powertrain Blockset provides fully developed models of automotive powertrain components such as Compression Ignition (CI), Spark Ignition (SI), electric traction motor, shafts, battery packs and controller model reference applications. These were adapted to use the parameter data that was provided by the component manufacturers, through use of the masked library block setup described in section 3.3.2 Model Configurator -

MATLAB-Simulink of this thesis. The plant model for the powertrain comprises of the ICE, transmission, ESS, motor, differential, torque transferring shafts, wheel/tyres and body models all connected to represent the P4 parallel through road architecture as shown in Figure 36 below. Note that the term PowerCube is used by GM and is given to the ICE-transmission pair.

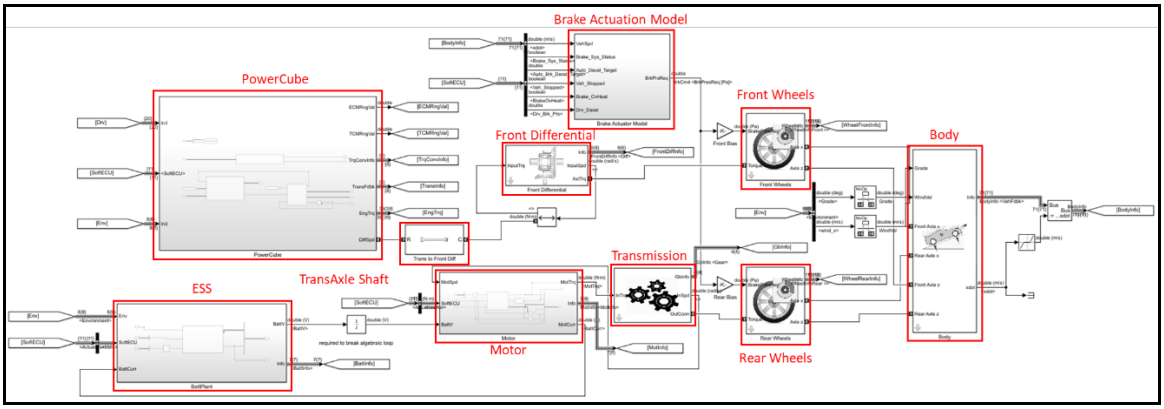


Figure 36: High Level Powertrain Plant Model Overview

4.2.3 ICE & Transmission Model

The ICE and transmission sub-systems are both adapted based on the confidential LCV and M3D data provided by GM respectively. The ICE sub-system is represented by the “Mapped SI” block, which is further expanded on by UWAFT to include the starter motor, as well as the catalytic converter. This data was produced through GM’s experimental results and was stored as 3D lookup tables within the model which are then imported as masked parameter to populate the ICE model. As described in the soft-ECU section above, the models are simplified to produce only the signals needed for UWAFT’s purposes. It’s important to note that since the mathematical representation of the engine is neither required not sought. The engine speed, engine torque, fuel flow, the Brake Specific Fuel Consumption (BSFC), and the exhaust gas composition are used instead in the simulation studies. The engine torque and speed are used to determine drivetrain and ultimately body states, whereas the fuel flow is used as a metric to calculate fuel consumption.

The transmission model utilizes a fixed-gear transmission model, that is connect to the ICE via a lock up type torque converter. The torque converter transfers power to the transmission model, which is populated via the gear ration, timing, gear efficiency, inertia, torque breakpoints and

speed breakpoints. Note that the torque converter block is unmodified, as calibration data for this device was unable during the development of the transmission sub-system.

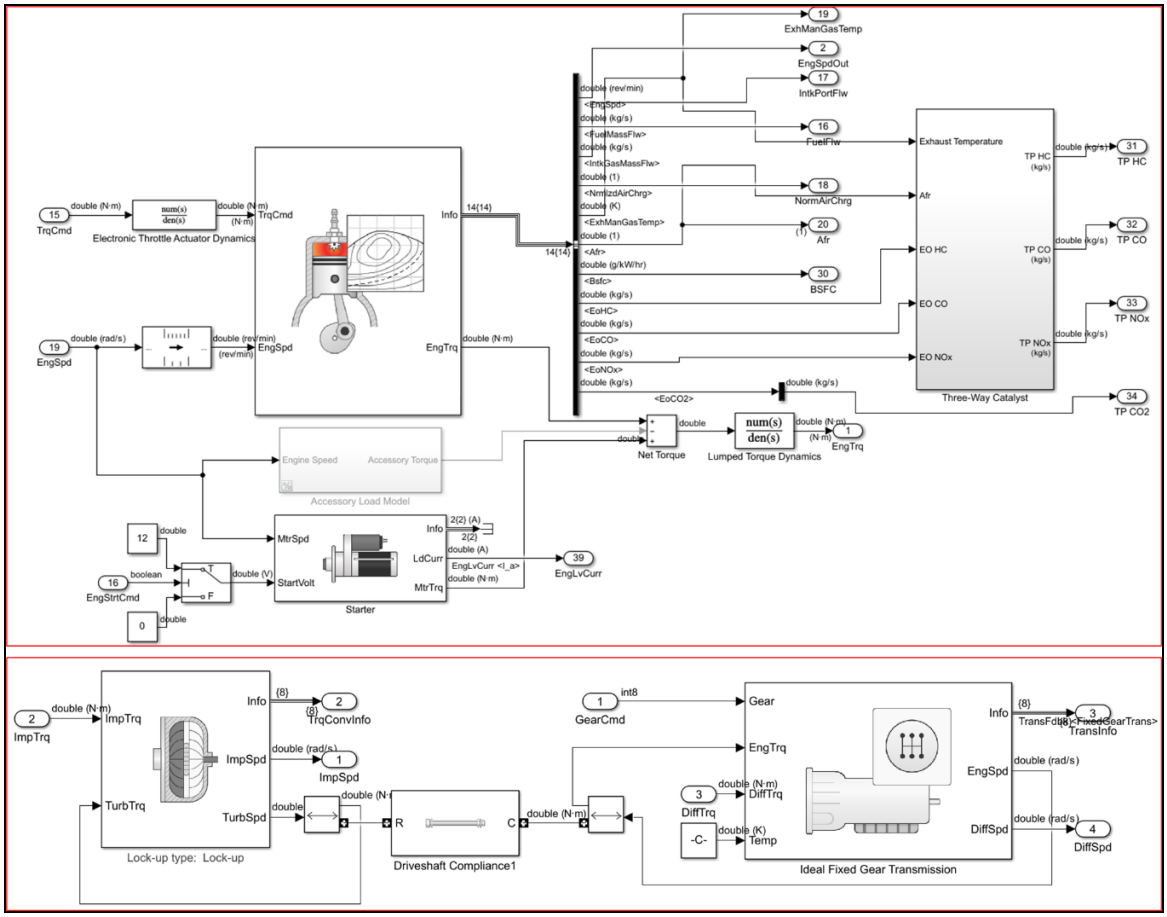


Figure 37: MathWorks Powertrain Blockset for ICE & Transmission Models

4.2.4 Energy Storage System

UWAFT worked closely with a Hybrid Design Services engineering consultancy firm for the development of its ESS. At its core the cell powering the battery pack is the Samsung INR18650-20S Lithium-Ion rechargeable cell. [64] The vehicle's VTS targets defined from the previous AVTC offerings – specifically results from the EcoCAR Emissions & Energy Consumption (E&EC) events were the primary driving factors for the power and electrical requirements of the ESS. The pack was delivered mid-way through Year 2 of the EMC competition, and the final specs of the resulting pack are as follows.

Table 6: HDS ESS Pack Model Characteristics

Requirement	Unit	Value
Pack Voltage	V	260-403.2
Total Pack Energy	kWh	5.5
Discharge Current Characteristics	A (s)	320 (2) 300 (10) 250 (30) 80 (∞)
Charge Current Characteristics	A (s)	160 (5) 100 (10) 32 (∞)

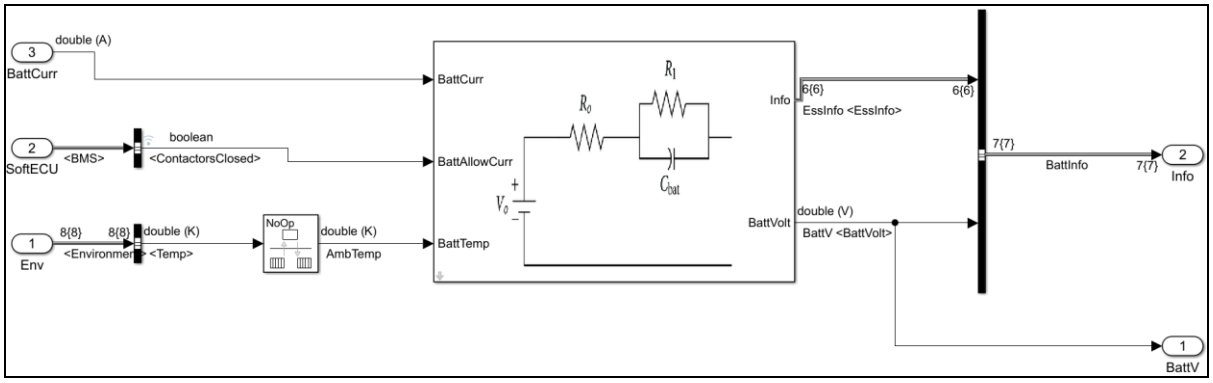


Figure 38: ESS 1-RC Model Representation [65]

The Simulink ESS model is represented using a 1-RC Thevenin Equivalent model, that consists of a an open-circuit voltage, a series resistor and 1 RC pair. The Samsung cells are configured in a 96S8P configuration. A consortium of tests is performed as a collaboration between the U.S DOE and HDS to ensure the pack chemistry met the performance targets that the team set out to achieve. These tests include the constant current constant voltage charge and constant current discharge for multiple batches of the Samsung cell. These experiments allowed characterization of the Li-Ion cell for the development of the Li-Bal BMS, as well as the UWAFT Simulink model ESS. Figure 39 below shows the results of the tests.

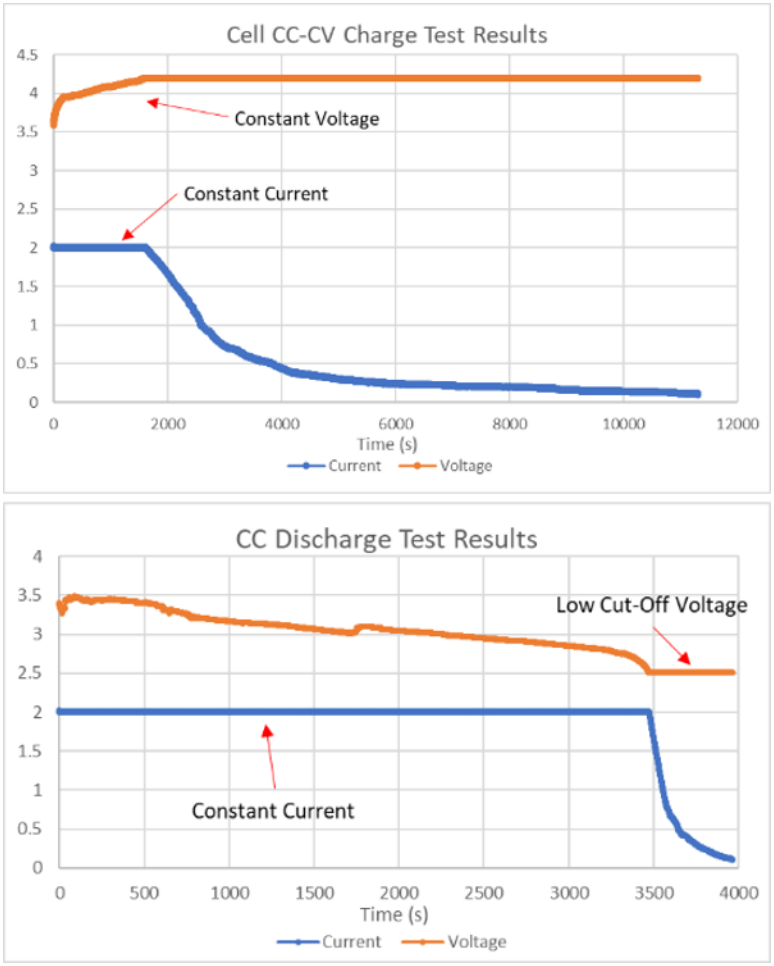


Figure 39: Charge & Discharge Characterization Tests for Samsung S20 Cells conducted in collaboration with HDS & U.S. DOE [64]

Further a long term 0.7C rated charge/discharge was applied to 8 cells in a parallel configuration, which would then be scaled up to 96 cells in series to form the overall energy capacity of the battery pack. These cells would be chosen at random over multiple batches to obtain the open circuit voltage (OCV) and SOC model curve, and then were averaged to obtain a lump sum model for the cell charge/discharge behavior. Figure 40 below shows a combined summary of all the tests conducted and the resulting pack models.

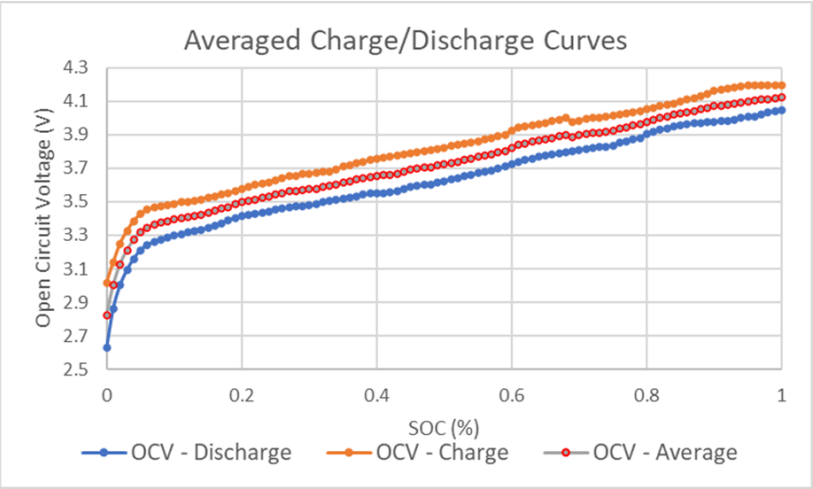


Figure 40: Averaged Charge/Discharge Curves for the HDS ESS

Data from Figure 39 and Figure 40 are used to establish the SOC vs OCV characteristics that are populated within the 1-RC battery model as look up tables. Note that the testing results are conducted for batches of 8 cells in parallel, this would allow for any cell degradation/health characteristics to be averaged out. For the Simulink model this is scaled up to represent 96 cells in series which is representative of the voltage, current and capacity characteristics of the full sized ESS.

4.2.5 AAM EDU4 Motor

The e-axle EDU4 is an liquid cooled 1-Spd open differential motor configuration, that is integrated concentric into the rear axle of the UWAFT Blazer. The internal electric configuration of this motor is of the Permanent Magnet Synchronous Motor (PMSM) type. The mapped motor in the Simulink model is an adaptation of the Flux-Based PMSM motor, where the electrical parameters are left the same, as they were not provided by the motor manufacturer, however the motor thermal, power and torque characteristics were provided and are adapted to match the Simulink Mapped Motor model [66]. The torque-speed efficiency curve was used to determine motor losses, and appropriately size the thermal system. Further the motor torque-speed power curves were used to determine longitudinal Blazer performance. It must be notes that the torque value indicated is torque at the e-machine, and not at the wheels which would be scaled by the internal differential ratio. The combined curves are shown in Figure 41 below.

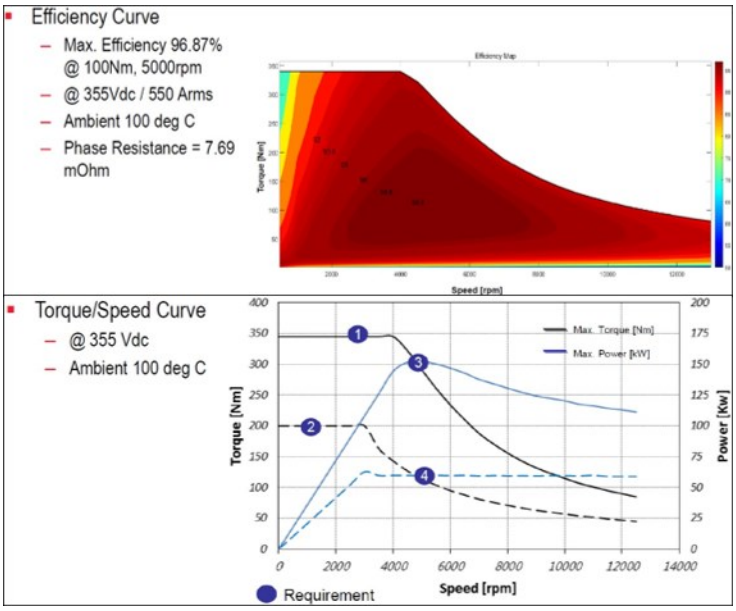


Figure 41: AAM EDU4 Torque/Speed/Efficiency Curve Data Incorporated into Simulink "Mapped Motor" sub-system [66]

The data curves provided by the AAM, where mapped into a look up table and represented in the mapped motor sub-system to represent the EDU4 motor as shown in the sub-system implementation below.

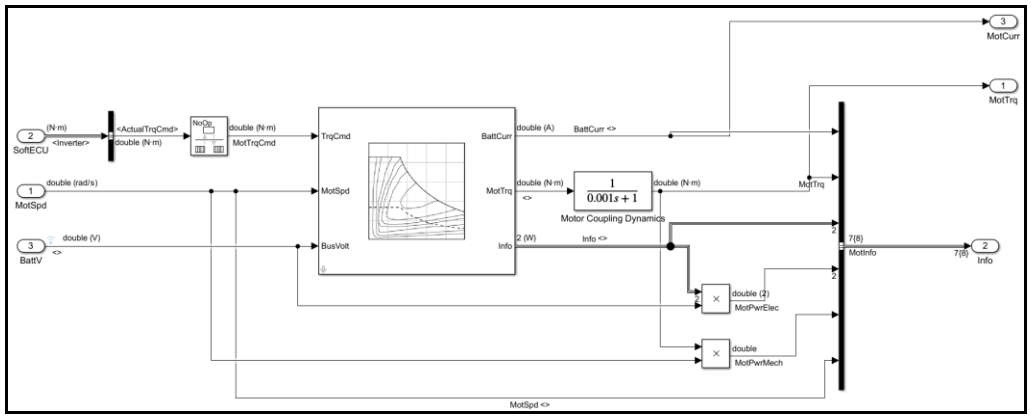


Figure 42: Motor Sub-system with Coupling Dynamics Implementation

Chapter 5
Model Validation, Testing & Results

In Chapter 3 we have taken a deep dive in to the framework that was utilized by UWAFT to develop and incorporate PCM oriented system requirements through use of the RTM. In Chapter 4 we described the various roles and organization of the HSC and interactions with the vehicle systems. Time in Year 2 of the competition was spent in development of requirements, and their programming in the MBD based workflow. Despite the limitation posed by the COVID-19 pandemic a significant amount of time and effort in validation of the HSC model, the requirements and testing of sub-systems integrated in to the UWAFT Blazer were undertaken.

5.1 Methodology

The plant model and functional requirements are tested in the SIL and VIL environments. Requirement from the RTM are programmed into the HSC and directly flashed on to the dSPACE MABX II hardware. Due to restrictions posed by the COVID-19 pandemic, and the targeted robustness validation of the vehicle required by end of Year 3 of EMC, priority was allotted to ensure safe and robust vehicle operation. This bottlenecked time available for HIL validation. The functionality is tested and validated on the UWAFT 2019 Blazer vehicle platform. While most of the safety testing for the unintended vehicle acceleration, and other HSC safety requirements were conducted at the University of Waterloo. The plots and graphs used for model validation, and operation of the HEV system were acquired at the 1.8 km Canadian Technical Center McLaughlin Advanced Technology Track (CTC MATT) [67] .

5.2 HSC Requirements Test Coverage for the AAM EDU4 & HDS ESS

In section 4.1.2 Model Tester Block & Simulink Test for Requirement Maintenance we have seen how the tester blocks is developed in path of the HSC feedback loop, and the setup of the test case assertion blocks to verify system level requirements. This sub-section describes the status of the critical requirements developed for the *team implemented* propulsion systems namely the EDU4 motor and the HDS ESS. Although many requirements exist for the ICE and transmission. Since the team does not currently have active control of the ECM/TCM - except state estimation

through feedback signals, validation of the motor and ESS requirement suffices for the level of functionality achieved leading up to year 3 of the competition. For requirements that are not possible to be tested in the SIL environment, such as a physical ground fault test - best efforts are made to describe the validation results gathered from logs gathered during VIL testing. Note that the RTM ID descriptors described here are kept consistent in the Simulink Environment, as described in section 3.2.5 Requirements Trace-ability Matrix of this thesis.

5.2.1 Test Coverage & Validation in Simulation Environment

All requirements that are developed in the simulation environments, end with assertion blocks. The assertion blocks are tied to logical unit test statements. Upon failure of any test statement, a non-zero value is generated that causes the assertion block to assert – halting the simulation. This process is shown in detail for a single test requirement in Figure 25. The following critical component level requirements taken from the RTM, are tested in the SIL environment. In total there are 7 ESS, and 17 motor/inverter related requirements that are tested within the tester block. The diagram below summarizes the requirements, and the status of validation. More than 80% of the requirements are tested and validated. The other 20% are imported from the RTM but are VIL level requirements, justifications for which are provided in the following VIL validation sections. Over the span of 3 years, the 132 requirements are developed, 65% of which have been developed, and validated.

Index	ID	Summary	Verified
UWAFT_Eco...			
1	BAT	BAT Component	
1.1	ALGO-BAT-11	The supervisory controller shall not command the pack contactors to open unless the pack current is less than 10A.	
1.2	ALGO-BAT-12	If ground fault isolation is less than 500 ohm times the instantaenous pack voltage, the battery pack contactors shall be opened.	
1.3	ALGO-BAT-14	If ground fault isolation is less than 500 ohm times the instantaenous pack voltage, the driver shall be notified via the ground fault cabin LED.	
1.4	ALGO-BAT-15	If the BMS opens contactors due to a fault, inverter torque production shall be disabled.	
1.5	ALGO-BAT-16	The supervisory controller shall not command contactors to close if the inverter reports measuring >=50V DC voltage while the BMS is reporting that contactors are open.	
1.6	ALGO-BAT-8	If the BMS opens contactors due to a fault, the HSC shall not attempt to close contactors again until the vehicle is turned off and on again.	
1.7	ALGO-BAT-9	The HSC shall not request torque from the inverter if: in The BMS is not the discharge state.	
2	MOT	MOT Component	
2.1	ALGO-MOT-2	Inverter shall be disabled if any of the following occur	
2.1.1	ALGO-MOT-2.1	If the motor phase current exceeds +/-424 Arms	
2.1.2	ALGO-MOT-2.2	If the DC link voltage is greater than 450 V or less than 321.6 V.	
2.1.3	ALGO-MOT-2.3	If a counter error is detected	
2.1.4	ALGO-MOT-2.5	If a zero torque command occurs (ALGO-MOT-3) due to torque feedback error (ALGO-MOT-3.1) twice before power cycle	
2.1.5	ALGO-MOT-2.6	If the measured motor torque, motor speed, DC Link voltage, phase current or motor temperature are no longer valid (based on validity signal from inverter)	
2.2	ALGO-MOT-3	Inverter shall command zero torque if any of the following occur	
2.2.1	ALGO-MOT-3.1	If the feedback torque is less than 90% or greater then 105% of the desired motor torque for over 1 second	
2.2.2	ALGO-MOT-3.2	If motor speed feedback exceeds +/-13000 rpm	
2.2.3	ALGO-MOT-3.5	If the motor temperature is less than -40 or greater than 150 degrees Celsius	
2.2.4	ALGO-MOT-3.7	If the engine is not running	
2.2.5	ALGO-MOT-3.9	The 12V battery voltage is below 10.5V	
2.3	ALGO-MOT-4	The HSC shall limit the motor torque command under the given conditions:	
2.3.1	ALGO-MOT-4.1	When phase current exceeds 320 A or 90% of ESS current limit (whichever value is lower)	
2.3.2	ALGO-MOT-4.3	When motor speed exceeds +/-12500 rpm	
2.3.3	ALGO-MOT-4.4	When motor temperature is greater than 140 degrees Celsius or less than -35 degrees Celsius	
2.3.4	ALGO-MOT-4.6	When ESS temperature is less than -25 or greater than 55 degrees Celsius	
2.4	ALGO-MOT-6	The inverter shall not enter the Enabled state until BMS is in Discharge State	
2.5	ALGO-MOT-7	Vehicle must not power motor if the vehicles gear selector is in neutral or park according to NYSR G.1.1.3	
2.6	ALGO-MOT-1.3	Regen braking shall not be active during reverse drive conditions	

Figure 43: Simulation Level Requirements Validation for AAM EDU4 Motor and HDS ESS

5.3 Model Validation

For plant model validation - the data acquired by the team during end of Year 3 testing is utilized. In particular, the drive cycle developed for vehicle testing followed the same guidelines as described in section 4.1.6.4 Drive Cycle Requirements except this one was developed for the CTC MATT facility. Due to the 1.8 km length of the CTC MATT facility – the 36-mile requirement of the drive cycle posed by EMC, required a total of exactly 32.2 or 33 total rounds of the test track. The driving was broken in 2 sets of continuous city and highway drives each. The city driving was limited to 55 kph with 4 full 10 second stops per lap, and the highway to 75 kph with 2 full 1 second stops per lap. The final drive cycle is shown below.

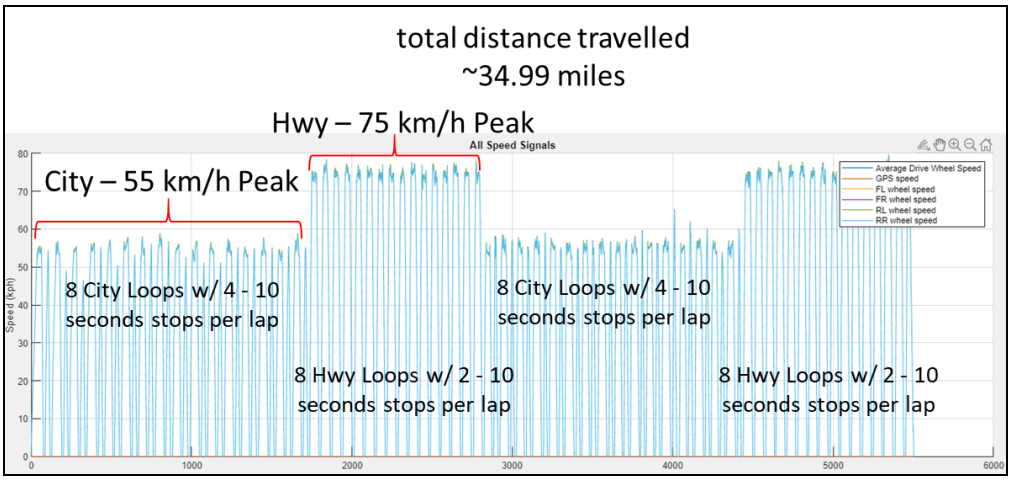


Figure 44: CTC MATT Energy Consumption Drive Cycle

5.3.1 ESS Model Validation

The ESS voltage data from the CTC MATT is used as the primary source of validation for the ESS plant model. To obtain this data, the drive cycle shown in Figure 44, is fed back into the HSC as drive cycle source, and the drive cycle is run as normal. The resulting pack voltage plot is obtained and is then compared to the real-world values obtained during the energy consumption testing event to produce the plot below.

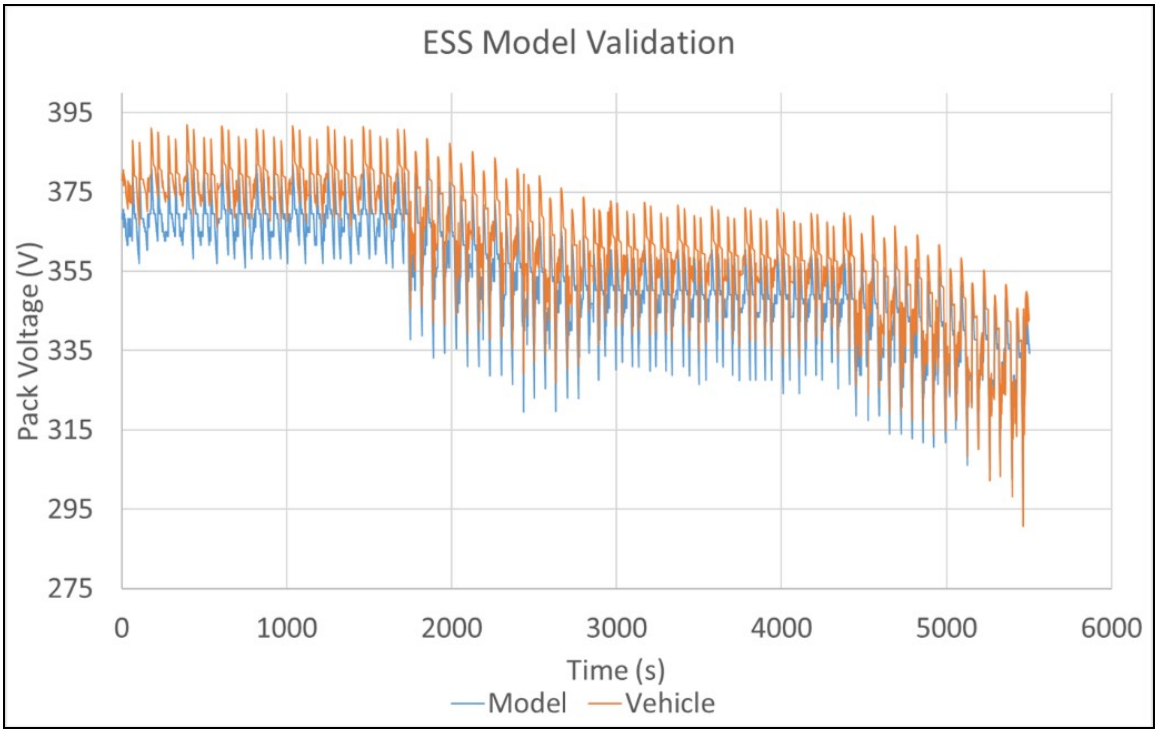


Figure 45: ESS Model Validation

Through the entirety of the model, there seems to an offset of -9.2V from the model when compared with the vehicle data acquired from the BMS. This offset is attributed to the starting voltage that corresponds to a certain SOC, and since the writing of this thesis has been corrected for. It may also be noted, the model follows the curve intricacies of the pack voltage quite well, and thus is a good representation of the overall SOC variation.

5.3.2 Motor Model Validation

The commanded motor torque signal is taken from the inverter data logs and is compared to the drive cycle input to the HSC. The commanded torque is relatively accurate as the motor must be paired to an inverter through a series of calibration activities before they are able to operate. This calibration was conducted on behalf of UWAFT by a third-party supplier FEV gmbh. Due to the enormity of the data and the high amount of overlap, the logged vehicle, and model data are separately plotted. A few key statistical comparison values are provided on the plots. It must also be noted that upon closer inspection the vehicle plot shows a much a higher level of noise as compared to the model.

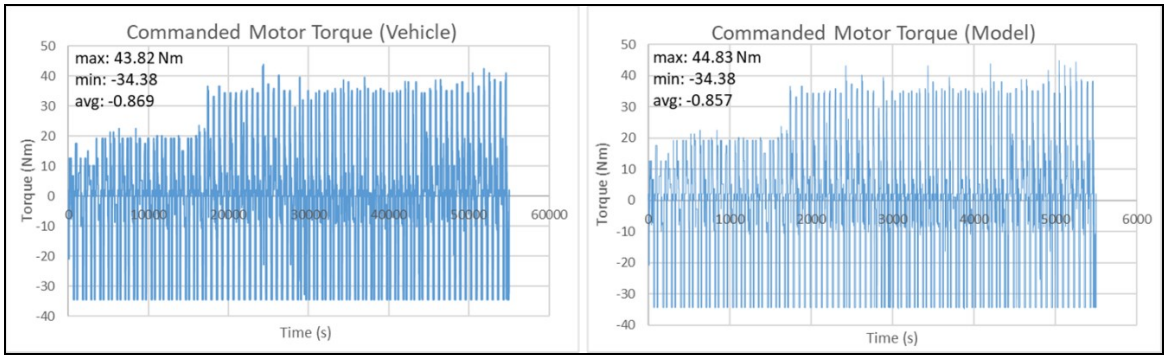


Figure 46: Motor Torque Validation

5.3.3 ICE Fuel Flow Model Validation

Validating the ICE is a much more involved process as compared to the motor due to the higher complexity. Since the purpose of the longitudinal simulation ultimately, is to measure fuel economy – fuel flow rate measurements are compared. An ICE due to its higher number of moving parts, differences in design, impacts due to ambient conditions and the type/quality of fuel used for the ICE testing is prone to a fair amount of variation in the results. It was therefore anticipated that a high degree of calibration was needed than just the rpm-torque and BSFC, data that was provided by GM to develop a proper ICE fuel consumption model. This was not possible due to shut down of campus dyno facilities and more importantly since the team was not able to travel to its usual full scaled testing events conducted at GM Proving Grounds in Yuma AZ – as it has been in past AVTC events. It is whence the team logged the fuel flow sensor on-board the UWFAT Blazer for fuel consumption data and compared with the fuel flow output from the ICE Simulink model. The ICE fuel flow validation plots are shown in figure xx below.

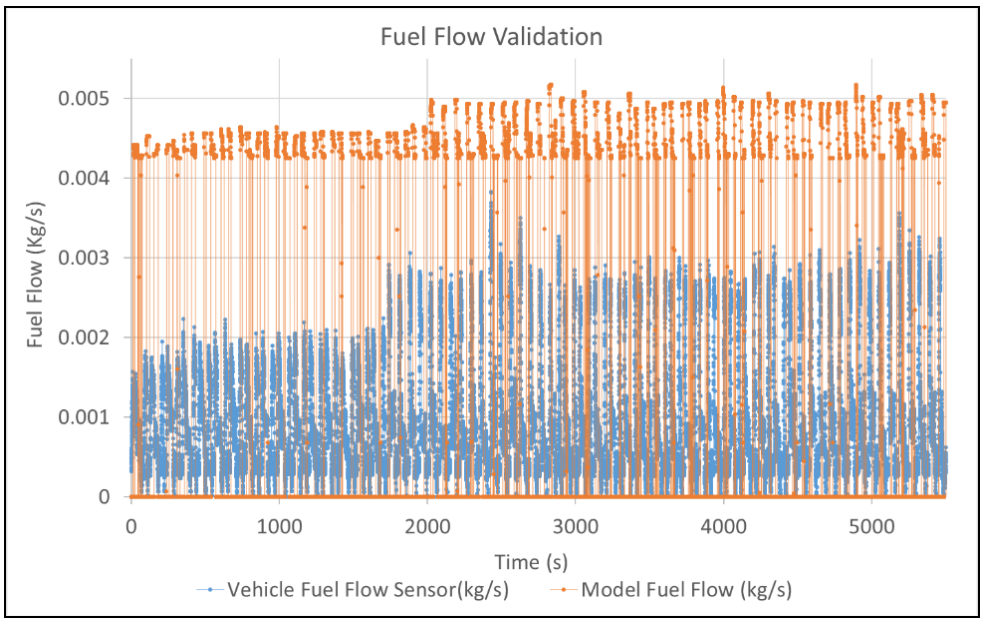


Figure 47: ICE Fuel Flow Validation

There are two main differences that can be noted about the status of ICE fuel flow model. Firstly, that the Simulink model data fuel flow values are much higher than the real-world data, and secondly that the real-world data is zero for the duration when the comes to a complete stop. The first difference is to the model over-estimating fuel consumption data through its BSFC calculation. The second is attributed to the fact that the when the vehicle comes to a stop in the simulation – the ICE spins to 0 RPM, which is a modelling shortcoming - such is not the case in real life. Discrepancy in data may also arises from fuel flow sensor itself which may not be entirely accurate. [68] It must also be noted that the fuel flow sensor on the vehicle measures fuel flow in l/s, whereas the model logs in kg/s, which is another source of error, among other such as the specific calorific value of fuel used in the model vs in real life. Therefore, a nominal E10 gasoline density of 0.74 kg/L [69] is used to convert the logged vehicle data to reflect the model.

5.3.4 Longitudinal Drive Trace & Executive APP, BPP Validation

The ability of the simulation model to follow a target vehicle speed is critical for the verification of performance of all other components. The speed trace plot affirms the simulated powertrain and body's ability to match real vehicle acceleration/speed performance for the driven drive cycle at MATT CTC. The Accelerator Pedal Position (APP) and Brake Pedal Position (BPP)

speed-trace validation graphs for the real vs simulated data reveals that there exist large and frequent high APP events, that are not present in real data. This is attributed to the large driver acceleration request because of the multiple start/stops that are present in the drive cycle. BPP curves for the model, and recorded data are similar, however the model uses a lesser magnitude. This is attributed to multiple things including but not limited to brake pad wear, damp track surface, but also that in real life, a high brake pedal value can be applied while the vehicle has already come to a stop by the driver, whereas the model only applies the minimal value needed to decelerate the vehicle.

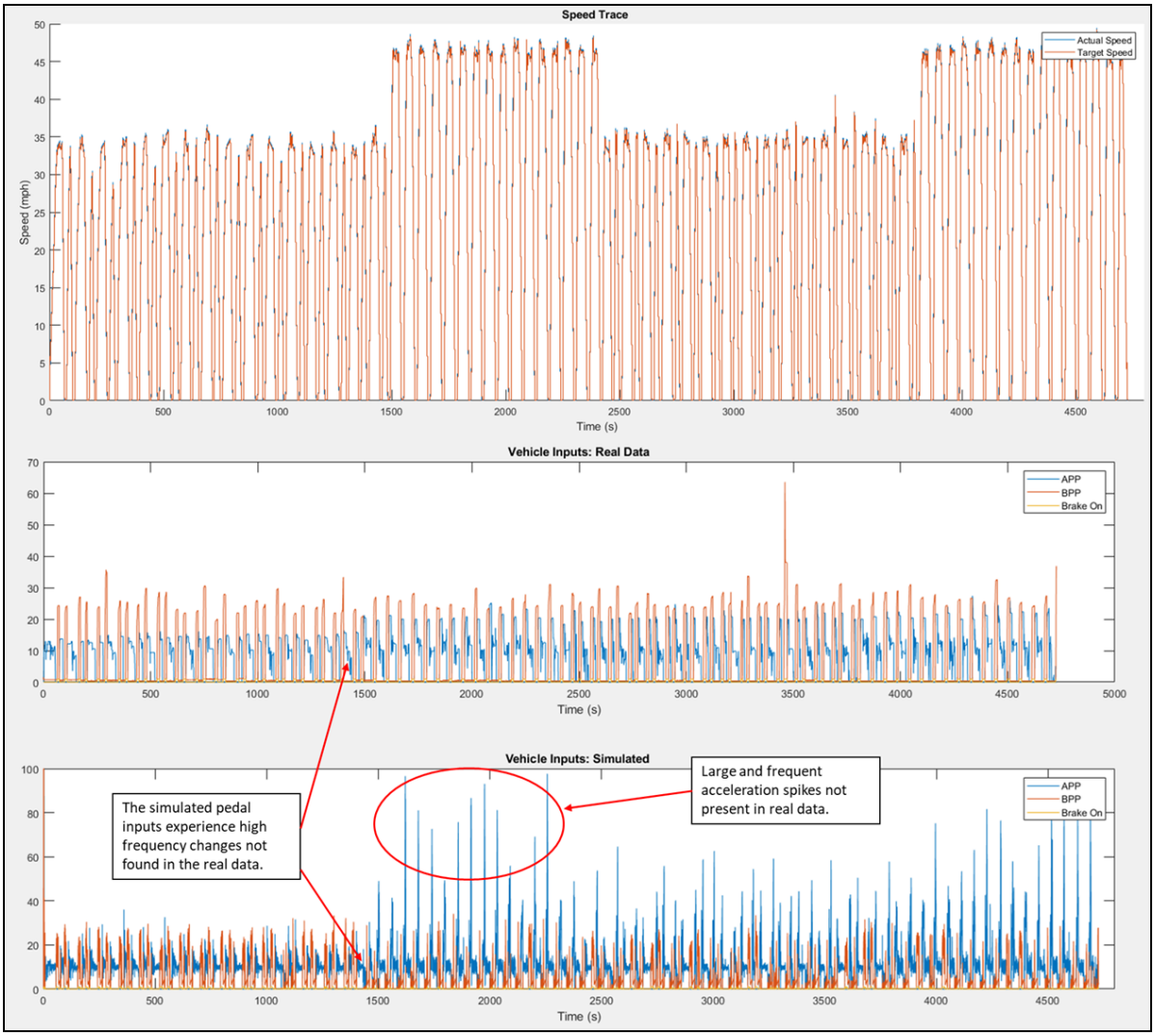


Figure 48: Vehicle Executive Inputs Validation

5.4 Vehicle On-Track Testing

At the MATT CTC facility, UWAFT conducted three major tests to verify systems integrity and validate vehicle performance targets. Note that due to the confidentially agreements with GM, the layout and setup at the test track are not shown for the CTC MATT facility. However, since the tests are regimented with specific requirements, and were conducted in preparation for the CTC MATT event, the detailed test plans with signal names setup requirements are shown in Appendix E - Acceleration 0-60 mph Test Plan, Appendix F- Braking 60-0 Test Plan and Appendix G - Energy Consumption Test Plan.

5.4.1 Unintended Acceleration Safety Evaluation

As described in section 3.2.4 Unintended Vehicle Acceleration System Level Requirement, the vehicle must not accelerate unintended without explicit system determination of a valid input signals. In UWAFT's P4 architecture there lie to propulsive systems – the stock LCV ICE and the EV propulsion systems powered by the ESS. The APP (Accelerator Pedal Position) is the signal that is either commanded by the throttle pedal OR is overridden by active safety systems such as ACC during autonomous driving. There are two main criteria to ensure the vehicle never accelerates unintended while the vehicle is on a lift. This was a competition pre-requisite before any of the vehicle performance testing was conducted. Firstly, that the team can prove the integrity of the APP signal, and that the HV systems are completely de-energized if the emergency E-Stop is pressed, and when the vehicle is switched out of the On state as described in Figure 31: . As shown in Figure 49 below, the HV system is de-energized (below 50V in under 10 seconds) when Key off or E-Stop event is triggered - to prevent any possible EV propulsion and the integrity of the APP signal during on-lift testing.

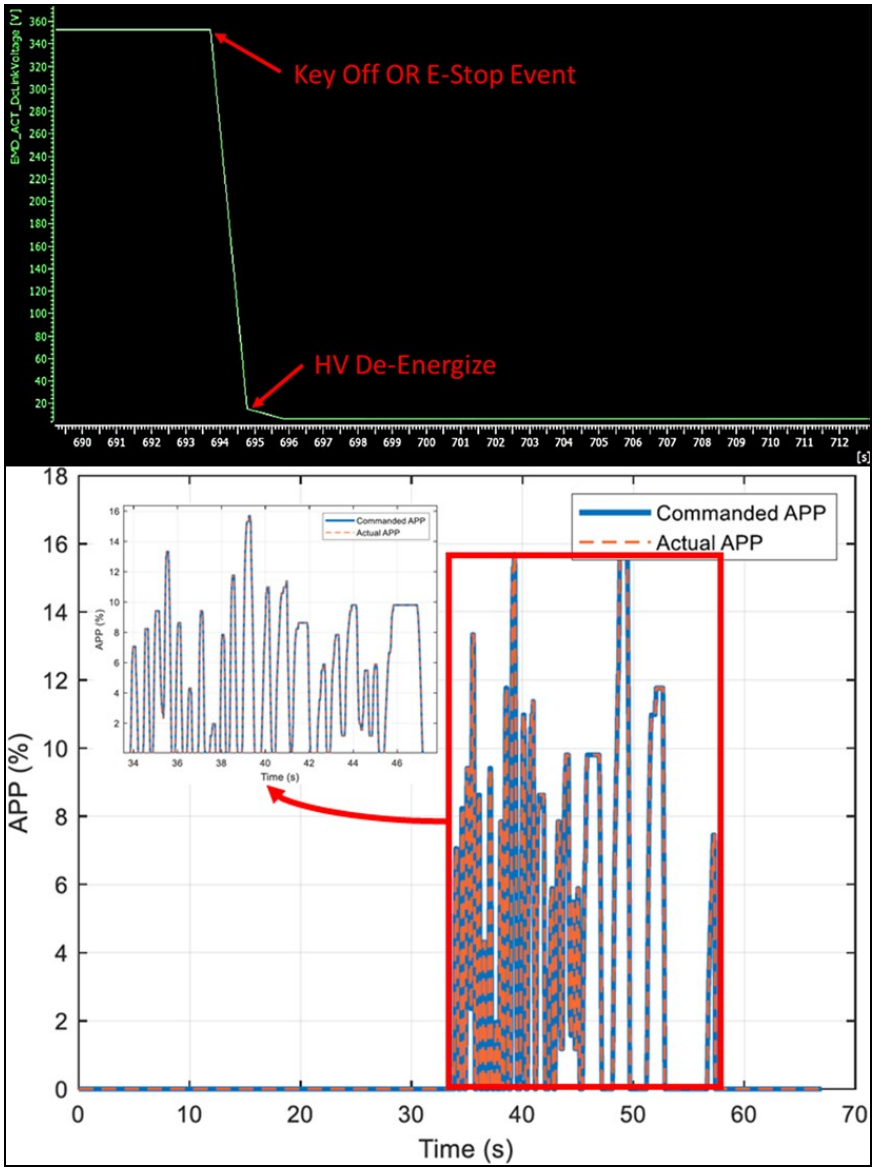


Figure 49:Unintended Vehicle Acceleration Validation

5.4.2 0-60 MPH Acceleration Performance Evaluation

The acceleration performance evaluation is an important VTS target that the team intended to meet as part of its architecture retrofitting for the UWAFT Blazer. The test involved accelerating the vehicle with Wide Open Throttle (WOT) over a straight-away and repeating the test both ways along the same strip to find an average 0-60 value. The test is conducted both ways to

mitigate any effects made to the test due to road gradient or wind. The figure below shows the annotated 750 ft straight-away at the WRESTRC test center that was replicated at MATT CTC.

Figure 50: 0-60 mph acceleration test [62]

Plots from both runs with averaged time is shown in Figure 51 below. The total time taken for the Blazer to begin rolling and hit 60 mph or 96 kph is averaged in the table below. The final average time for the 0-60 mph for the Blazer is 6.75s.

Table 7: 0-60 MPH Run Times

Time - Run 1 (s)	Time - Run 2 (s)
6.62	6.91

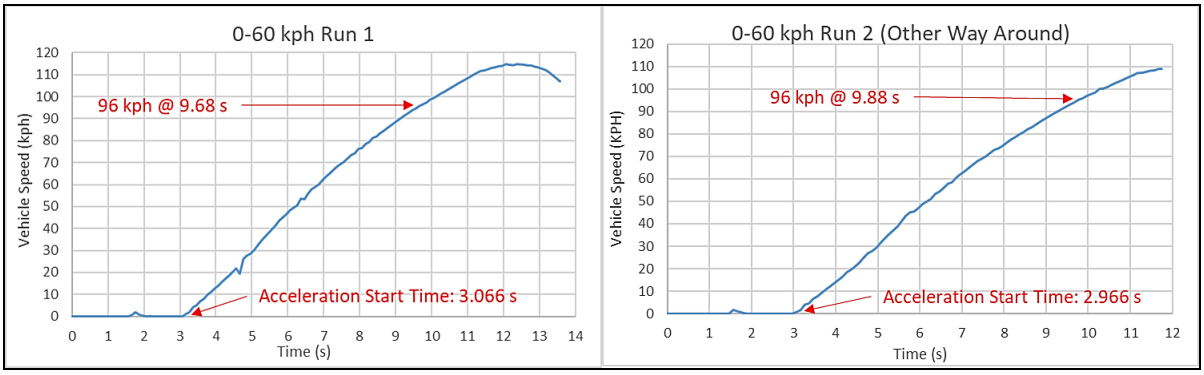


Figure 51: 0-60 Acceleration Runs Both Ways

5.4.3 60-0 MPH Braking Performance Evaluation

The braking performance evaluation is another important VTS target test for which was conducted at MATT CTC. The test requires the vehicle accelerates to 60 mph and comes to a complete through full use of vehicle baking, without any regenerative braking. The course is setup like the acceleration test, except, at the start line the vehicle is already travelling at the target speed. Two tests were conducted both ways.

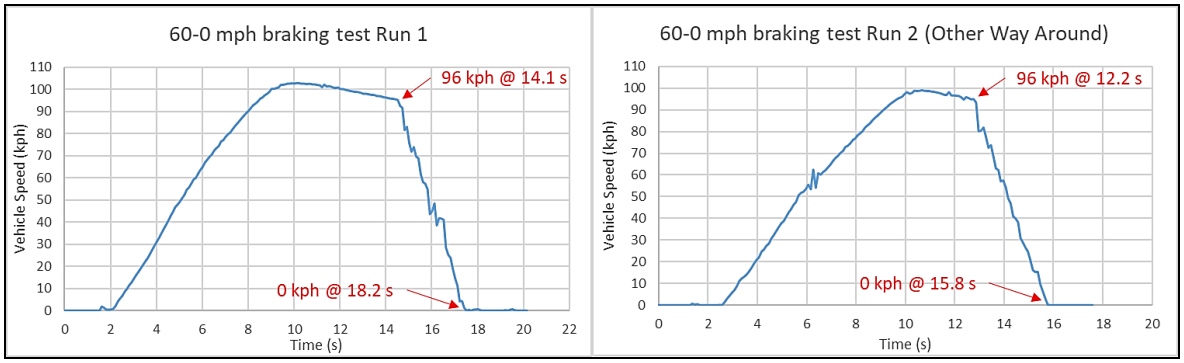


Figure 52: 60-0 Braking Runs Both Ways

The curves gathered from the braking test are integrated over the braking period, over the time and speed limits annotated in Figure 52. The results are then converted to feet and averaged. The vehicle can come to a complete stop with an average distance of 168.25 ft.

Table 8: 60-0 Braking Distance

Distance - Run 1 (ft)	Distance - Run 2 (ft)
164.9	171.6

5.4.4 Energy Consumption Testing

For the energy consumption test the CTC MATT energy consumption drive cycle shown in Figure 44 was used. UWAFT ran the vehicle for a total of 56.3 km through the CTC MATT drive cycle. The test run elapsed for 1 hour and 32 minutes, during which the ICE and EV systems were operational throughout the entirety of the run. The CD torque strategy tuned for the WRESTRC drive cycle worked well at CTC MATT and allowed the team to retain a safe amount of SOC of 34% at the end of the drive cycle run. Overall, the team used 6.48 L of fuel as measured by the competition required fuel flow sensor, and a measured amount of 4.873 L. The measured fuel amount was calculated by brimming the tank, before the run, and immediately after ending the run, brimming and weighing the filling cannister. A density value of 0.74 kg/L [69] for E10 gasoline was applied. Based on the measured data the team concluded testing with a fuel economy result of **27.6 mpg**.

Performing an integration on the amount of amount of net high voltage battery current, the battery pack depleted 6.391 Ah, which was integrated with respected to time to obtain ~2.25 kWh of energy - provided by the ESS alone. This makes sense as the ESS is ~5.5 kWh in size, and the SOC went from 80% to 36% which is 46% of depletion. 44% of 5.5 kWh is 2.42 kWh. The remaining 0.17kWh can be attributed to reported SOC measurement drift that may have resulted from the long running time of the ESS based on the BMS reported values.

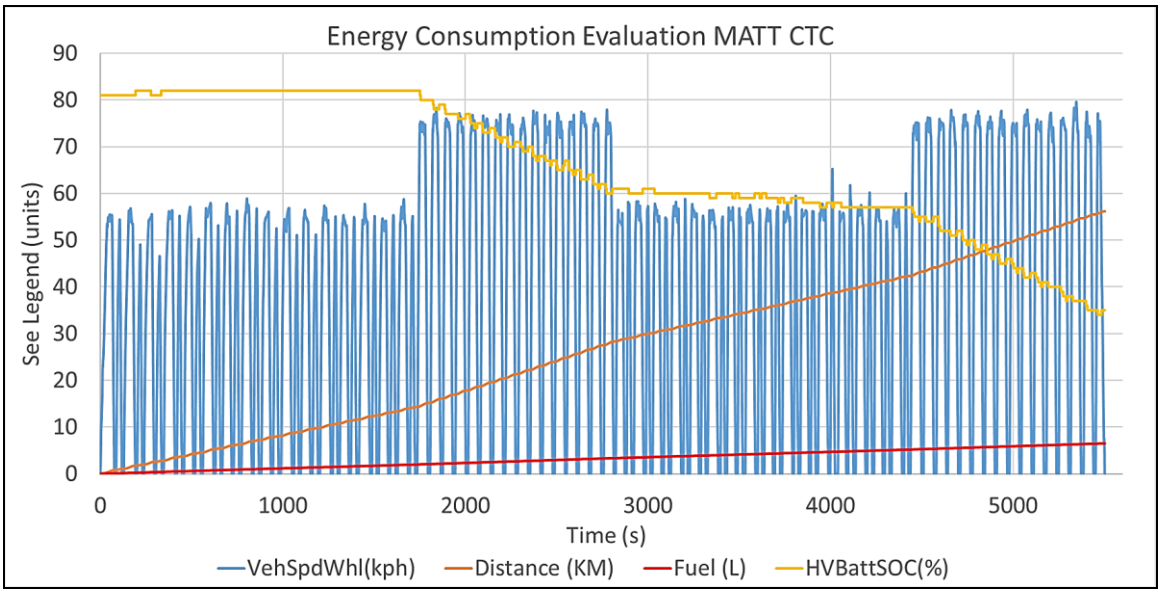


Figure 53: Energy Consumption Evaluation MATT CTC

5.5 VTS Recap

The initial UWAFT VTS that was established in year 1 of EMC and is shown in Table 2 of this work. Upon completion of all tests in year 3 of EMC, the VTS table can be updated as follows. The overall curb weight and braking distance are within 5% of each the team's original target and can be comfortably achieved through swapping of mounts with less denser materials, and possibly upgrading the vehicle brakes to larger ones. An important item to note is that UWAFT during its year 1 modelling used regenerative braking when modelling the braking results, where-as in year 3 testing, the team was not allowed to use regenerative braking during the braking distance tests. As such the weight and stopping distance are acceptable.

The area of a much higher concern are acceleration and fuel economy numbers. While the fuel economy figure is ~11% off, the current UWAFT Blazer's acceleration performance is over 20% off its anticipated target.

Table 9: Measured VTS Results

Specifications	Units	UWAFT VTS	Measured
Layout	N/A	P4 Parallel Through Road	N/A
Engine / Transmission	ft	2.5L I4 NA LCV / M3D GF9	N/A
Curb weight	kg	2100	2066
0-60 MPH	s	5.5	6.75
60-0 MPH	ft	158.2	168
Fuel Economy Combined	mpg	30.83	27.6

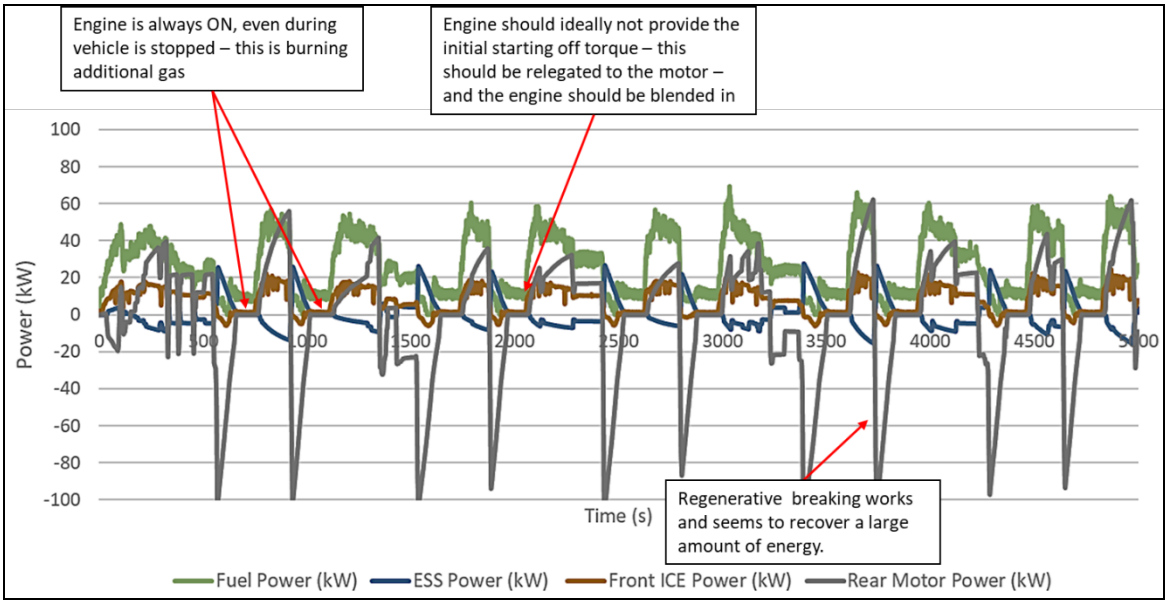
The following section is an in-depth analysis as to why the UWAFT Blazer is not able to currently meet its acceleration and fuel economy targets and what are the immediate areas of concern, requiring addressing for the team to succeed in Year 4 of EMC.

5.6 In-Depth Discussion of Testing & VTS Related Limitations

After testing concluded at MATT CTC, a fair amount of time was spent developing an assessment around the performance of the UWAFT Blazer, and where necessary comparing it to the model results. The plots included in this section are included as supplementary information for discussion of the overall results and areas of the UWAFT Blazer that are high priority to be resolved for improvement in overall fuel economy and system performance.

5.6.1 ICE & Motor Power Distribution

Another major contributing factor to loss of system performance is the current inability of the team to turn off the ICE at low power requirements. This especially affected the fuel economy number as the ICE always remained ON, even during the 10 second recurring vehicle stop events. Of the 91 minutes of total drive time, the vehicle was at a complete stop for a total of 16 minutes, this a large amount of time where the ICE should've been off. The ICE is also most inefficient at vehicle launch, having engine start/stop functionality would potentially help displace the initial acceleration to the EV motor, and would allow the team to make up the 10% off target fuel economy figure.



5.6.2 EV Thermal Systems

From a thermal systems standpoint the inverter and EDU4 motor share the same water-cooling loop. As it can be seen, the plateauing nature of the temperatures shows that both components inline of the thermal loop are well thermally controlled and within the operating spec.

One of the major system limitations, that became apparent during initial testing at Waterloo was the poor thermal systems design of the HDS battery pack. The HDS battery pack pulls air from the passenger cabin and exhausts it to the environment. Despite exposing the inlet to plenty of fresh air, the ESS internal temperature kept rising for the entirety of the test. HDS has

not provided the team with access to fan control, due to which additional cooling could not be commanded.

To mitigate this short term and fulfill the drive cycle requirements of the energy consumption test the team had to resort to restricting power draw from the motor. In effect reducing cell heating due to inefficiencies. This was achieved through trial and error at WRESTRC during development and testing for the energy consumption drive cycle. Despite limited power draw the pack temperatures continued to rise. This is a serious design limitation, that directly impacted the team's final fuel economy and acceleration figures, as the team ultimately could not request a higher forward or regenerative motor torque. The team must work closely with HDS to try and gain control of the internal fans and find an alternate way to force cooler air in to the ESS circumventing this issue.

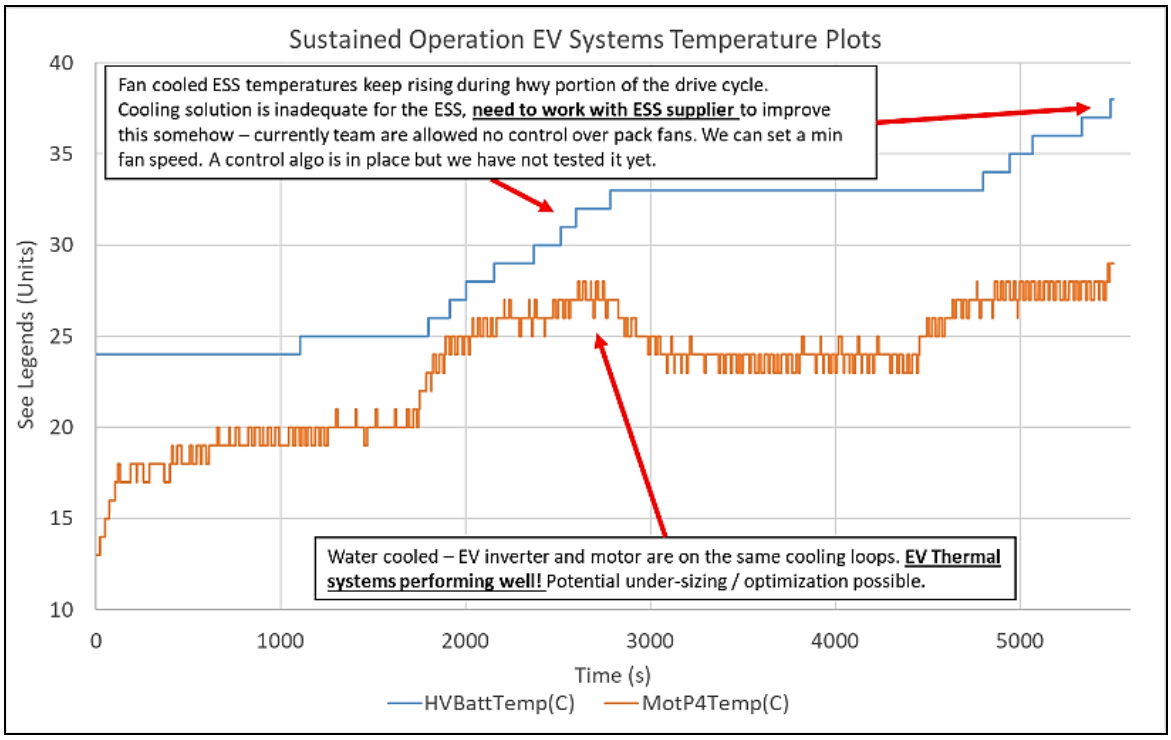


Figure 54: Sustained Operation EV System Temperature Plots

5.6.3 Limited EV Torque Application

The P4 motor is rated for 190 Nm of forward and regenerative torque application. A Torque-RPM plot shows that throughout the fuel economy testing, the motor is barely being utilized to

its full potential. As described in the sections above, this is due to the thermal limitations of the ESS and is not a result of thermal loop of motor itself. It may also be noted that the DP torque strategy ultimately is a primitive torque strategy and does not deploy the best possible torque blending with the ICE, resulting in less ideal drive quality characteristics. The team has yet to also receive the flashed ECM from GM that allows propagation of a torque command from the MABx, where the current torque strategy is purely torque additive.

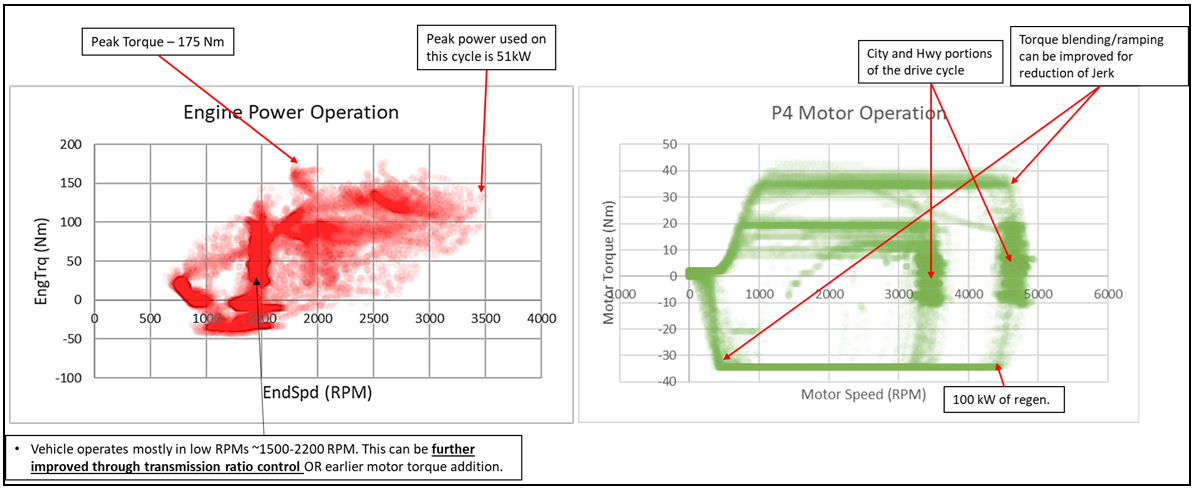


Figure 55: Underutilization of EV systems

Chapter 6
Conclusions

The presented research identifies the brevity involved in developing a real-time controller that is organized in such a way to reduce complexity associated with adding functional layers across the supervisor. Emphasis is placed on appropriate scoping of sub-system functions that are cascaded for distinct identification of roles, and introduction of new functionality. The decisions made over 3 years of developed are geared towards ease of feature development, incorporation of requirements-based testing methodology and a functional supervisory controller that is agnostic of powertrain architecture. The foundation upon which the simulation model stands is expanded on through use of object-oriented programming such that developer interactions facilitate a degree of automation in repetitive tasks such as initialization, test case running, and launching model variants for SIL, HIL or VIL target hardware flashing.

The design of the HSC, and the inline placement of the tester assertion blocks ties the RTM requirements right into the HSC allowing for test case to be automatically asserted when a requirement is not met. This ensures minimal logic errors or faults to propagate during HIL or VIL testing. Thus far 86 requirements out of a total of 132 controls requirements are validated, the other of which are pending development for year 4 of EMC. The structure and heavy use of the RTM centralizes development across all sub-teams ensuring team-wide transparency, and recognition of dependency. The COVID-19 pandemic limited use of the HIL in model validation, due to restriction in garage access.

New and more experienced developers benefit from the segregated roles and functions of the HSC, in a few different ways. Development efficiency is improved as a parallel version control-oriented development environment is made possible minimizing merge conflict possibilities when pull requests are generated to incorporate new features. A well thought out structure exists for adding future components, and/or component/vehicle functionality. Code ownership and sub-system specific testing is improved as developers take on smaller, more modular sub-systems to develop.

The expansion of Model Based Design with UWAFT's Model Configurator tool provides several benefits over manual Simulink based model interaction. The model initialization, and switch-over to HIL setup is improved from a ~9 minutes process to ~4 minutes process due to reduced manual dSPACE RTI hardware setup, and instantiation of a new model for HIL based I/O configuration. Component parameter population and modification is simplified through use of masked library blocks to encapsulate data and constants for 7 team added ECUs.

The rule-based torque strategy is successfully able to sustain robust HEV operation and safe SOC charge through the MATT CTC E&EC testing. The Blazer is currently off of its VTS acceleration target by 20%, and its fuel economy target by 11%. The team must work with the HDS ESS supplier to uncap the significant thermal constraint, for a possible improvement in higher electric regenerative braking capability and higher forward torque value. The 16-minute idle time in the MATT CTC fuel economy significantly contributes to a lower mpg value, as no miles are driven during that time.

One of the operationally viable – full EV mode, is currently not possible as the team does not have 1) a fully analyzed 12V system, that can support all vehicle ECUs, thermal, brake and steering systems to be fully ON. 2) development of TCM signal gate-waying through MABx, for forcing neutral when the shifter is in Drive.

Bibliography

[1] M. N. Smith, "The number of cars worldwide is set to double by 2040," 22 April 2016. [Online]. Available: https://www.weforum.org/agenda/2016/04/the-number-of-cars-worldwide-is-set-to-double-by-2040. [Accessed 3 July 2021].

[2] U.S. Energy Information Administration , "ANNUAL ENERGY OUTLOOK 2021," 3 February 2021. [Online]. Available: https://www.eia.gov/pressroom/presentations/AEO2021_Release_Presentation.pdf. [Accessed 03 07 2021].

[3] U.S. Energy Information Administration, "AIR QUALITYU.S. Energy-Related CO2 Emissions Fell By 2.8% In 2019," 04 06 2020. [Online]. Available: https://cleantechnica.com/2020/06/04/u-s-energy-related-co2-emissions-fell-by-2-8-in-2019/. [Accessed 08 08 2021].

[4] M. Woodward, D. B. Walton and D. J. Hamilton, "Setting a course for 2030," 28 July 2020. [Online]. Available: https://www2.deloitte.com/us/en/insights/focus/future-of-mobility/electric-vehicle-trends-2030.html. [Accessed 3rd July 2021].

[5] H. Yasar and G. Canbolat, "Performance Comparison for Series and Parallel Modes of a Hybrid Electric," *Sakarya University Jounral of Science,* vol. 23, no. 1, pp. 43-50, 2019.

[6] P. Savagian and E. Tate, "The Electrification of the Automobile: From Conventional Hybrid, to Plug-in Hybrids, to Extended-Range Electric Vehicles," *SAE International Journal of Passenger Cars - Electronic Journal Systems,* vol. 1, no. 1, pp. 156-166, 2009.

[7] T. Nüesch, M. Wang, C. Voser and L. Guzella, "Optimal Energy Management and Sizing for Hybrid Electric Vehicles Considering Transient Emissions," in *IFAC Proceedings Volume*, Zurich, 2012.

[8] M. A. Yard, "Control and Drive Quality Refinement of a Parallel-Series," 2014. [Online]. Available: https://etd.ohiolink.edu/apexprod/rws_etd/send_file/send?accession=osu1417695439&disposition=attachment. [Accessed 03 07 2021].

[9] G. B. Regulwar, P. M. Jawandhiya, V. S. Gulhane, R. M. Tugnayat, P. R. Deshmukh and , "Variations in V Model for Software Development," *International Journal of Advanced Research in Computer Science,* vol. 1, no. 2, 2010.

[10] Advanced Vehicle Technology Competition, "ECOCAR MOBILITY CHALLENGE," [Online]. Available: https://avtcseries.org/ecocar-mobility-challenge/. [Accessed 04 07 2021].

[11] C. Shen, P. Shan, T. Gao and , "A Comprehensive Overview of Hybrid Electric Vehicles," *International Journal of Vehicular Technology,* vol. 2011, 2011.

[12] M. YOSHIDA, M. KITA and H. ATARSHI, "https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/318437/1/phdthesis.pdf," Automobile R&D Center.

[13] B. R. Siddhart, V. S. Chandu, D. M. Babu, D. J. Pradeep and Y. V. Pavan Kumar, "Performance Analysis of Hybrid Electric Vehicle Architectures," *International Journal of Advanced Science and Technology,* vol. 29, no. 10S, pp. 4370-4390, 2020.

[14] United States Environmental Protection Agency, "Combining Data into Complete Engine ALPHA Maps," United States Environmental Protection Agency, March 2018. [Online]. Available: https://www.epa.gov/vehicle-and-fuel-emissions-testing/combining-data-complete-engine-alpha-maps. [Accessed Sunday July 2021].

[15] H.-Y. Hwang, T.-S. Lan and J.-S. Chen, "Control Strategy Development of Driveline Vibration Reduction for Power-Split Hybrid Vehicles," *MDPI Applied Sciences,* vol. 10, no. 5, 2020.

[16] B. Sendyka and A. Sochan, "INCREASE OF THE TOTAL EFFICIENCY USING THE ATKINSON CYCLE IN THE SPARK IGNITION ENGINE," *Journal of KONES Powertrain and Transport,* vol. 16, no. 2, 2009 .

[17] K. V. Singh, H. O. Bansal and D. Singh, "A comprehensive review on hybrid electric vehicles: architecturesand components," *J. Mod. Transport.,* vol. 27, no. 2, pp. 77-107, 2019.

[18] A. H. Winkler and R. W. Sutton, "Electrojector - Bendix Electronics," in *SAE Annual Meeting*, Detroit, 1957.

[19] Robert Bosch GmBH, "Embedded System Design: A Unified Hardware/Software

Introduction," 5 October 1991. [Online]. Available: http://esd.cs.ucr.edu/webres/can20.pdf. [Accessed 4 July 2021].

[20] A. Pretschner, M. Broy, I. H. Kruger and T. Stauner, "Software Engineering for Automotive Systems: A Roadmap," in *Future of Software Engineering (FOSE '07)*, Minneapolis, MN, USA, 2007.

[21] J. Yu and B. M. Wilamowski, "Recent advances in in-vehicle embedded systems," in *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, Melbourne, VIC, Australia, 2011.

[22] MathWorks, "Why Adopt Model-Based Design for Embedded Control Software Development," 2014. [Online]. Available: https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/campaigns/portals/files/adopting-model-based-design/why-adopt-model-based-design-for-embedded-control-software-development.pdf. [Accessed 4 July 2021].

[23] MathWorks, "Model-Based Design with Simulink," MathWorks, [Online]. Available: https://www.mathworks.com/help/simulink/gs/model-based-design.html. [Accessed 4th July 2021].

[24] P. J. Tulpule and A. Rezaeian, "Model Based Design (MBD) and Hardware In the Loop validation: Curriculum Development," in *2017 American Control Conference (ACC)*, Seattle, 2017.

[25] K. S. Gurumurthy and R. Hegde, "Model Based Approach for the Integration of ECUs," in *Proceedings of the World Congress on Engineering* , London, 2008.

[26] T. Kelemenova, M. Keleman, L. Mikova, V. Maxim, E. Prada, T. Liptak and F. Menda, "Model Based Design and HIL Simulations," *American Journal of Mechanical Engineering,* vol. 1, no. 7, pp. 276-281, 2013.

[27] C. Atkinson and G. Mott, "Dynamic Model-Based Calibration Optimization: An Introduction and Application to Diesel Engines," in *2005 SAE World Congress*, Detroit, Michigan, 2005.

[28] J. Nibert, M. E. Herniter and Z. Chambers, "Model-Based System Design for MIL, SIL, and HIL," in *World Electric Vehicle Journal Vol. 5* , Los Angeles, California, 2012.

[29] M. D. Russo, G. Zhu, K. Jerry and S. Balakrishnan, "Development of the Hybrid Supervisory

Controller for a Pre-Transmission Hybrid Electric Vehicle for Year 3 of the EcoCAR3 Competition," *SAE Technical Paper,* p. 16, 2018.

[30] J. L. Bossa, G. A. Magallan, H. C. De Angelo and G. O. Garcia, "Implementation of a Supervisory Control System for an Electric Vehicle," in *Industry Applications (INDUSCON), 2010 9th IEEE/IAS International Conference*, Sao Paulo, Brazil, 2010.

[31] B.-A. M. and M. F., "Finite State Machines," in *Finite State Machines. In: Elements of Robotics.*, Springer, Cham, 2018, pp. 55-61.

[32] Cascadia Motion, "Documentation Tools and Firmware," 30 June 2021. [Online]. Available: https://app.box.com/s/vf9259qlaadhzxqiqrt5cco8xpsn84hk/file/23982240179. [Accessed 05 July 2021].

[33] T. Liang, "An Approach for Vehicle State Estimation Using Extended Kalman Filter," in *Springer*, Berlin, Heidelberg, 2012.

[34] S. Onori, L. Serrao and G. Rizzoni, in *Hybrid Electric Vehicles - Energy Management Strategies*, London, Springer, 2016.

[35] C. C. MacAdam, "Understanding and Modeling the Human Driver," *Vehicle Systems Dynamics,* vol. 40, no. 1-3, pp. 101-134, 2003.

[36] R. A., S. P. and P. M., "Validation Process of an HEV System Analysis Model," *SAE Technical Paper,* vol. 16, no. 6, pp. 1242-1251, 2001.

[37] C. Sankavaram, B. Pattipati, K. M. Pattipati, Y. Zhang and M. Howell, "Fault Diagnosis in Hybrid Electric VehicleRegenerative Braking System," *IEEE Access,* vol. 2, no. 2169-3536, pp. 2169-3536, 2014.

[38] T.-K. Lee, Y. Kim, A. Stefanopoulou and Z. S. Filipi, "Hybrid Electric Vehicle Supervisory Control Design Reflecting Estimated Lithium-Ion Battery Electrochemical Dynamics," in *2011 American Control Conference*, O'Farrell Street, San Francisco, CA, USA, 2011.

[39] N. Kim, S. Cha and H. Peng, "Optimal Control of Hybrid Electric Vehicles Based on Pontryagin's Minimum Principle," *IEEE Transactions on Control Systems Technology,* vol. 19, no. 5, pp. 1279 - 1287, 2011.

[40] C. Arben, A. Reama, R. Haouche and A. Hrazdira, "Real time energy management algorithm for hybrid electric vehicle," in *ITSC 2011. 14th International IEEE Conference on Intelligent*

Transportation, Washington, DC, USA, 2011.

[41] T. Hofman, M. Steinbuch, A. Serrarens and D. v. RM , "Rule-based energy management strategies for hybrid vehicle drivetrains: A fundamental approach in reducing computation time," *IFAC Proceedings Volumes,* vol. 47, no. 3, pp. 740-745, 2006.

[42] T. Hofman, M. Steinbuch and A. Serrarens, "A Rule-based energy strategies for hybrid vehicles," *inInternational Journal of Electric and Hybrid Vehicles,* vol. 1, no. 1, pp. 71-94, 2007.

[43] F. R. Salmasi, "Control Strategies for Hybrid Electric Vehicles: Evolution, Classification, Comparison, and Future Trends," *IEEE Transactions on Vehicular Technology,* vol. 56, no. 5, pp. 2393-2404, 2007.

[44] M. J. Yatsko, "Development of a Hybrid Vehicle Control System," The Ohio State University, Ohio, 2016.

[45] P. Tulpule, V. Marano and G. Rizzoni, "Energy management for Plug-in Hybrid Electric Vehicles using Equivalent Consumption Minimisation Strategy," *International Journal of Electric and Hybrid Vehicles,* vol. 2, pp. 329-350, 2010.

[46] G. Rizzoni, L. Serrao and T. Donateo, "A two-step optimisation method for the preliminary design of a hybrid electric vehicle," *International Journal of Electric and Hybrid Vehicles,* vol. 1, no. 2, pp. 142-165, 2008.

[47] R. Trajano, "2019 Chevrolet Blazer RS First Test: The Camaro of Crossovers," MOTORTREND, 07 March 2019. [Online]. Available: https://www.motortrend.com/reviews/2019-chevrolet-blazer-rs-first-test-review/. [Accessed 15 July 2021].

[48] J. Glancey, "Failure Analysis Methods - What, Why and How," Spring 2006. [Online]. Available: http://research.me.udel.edu/~jglancey/FailureAnalysis.pdf. [Accessed 20 July 2021].

[49] MathWorks, "Create a Simple Mask," [Online]. Available: https://www.mathworks.com/help/simulink/ug/how-to-mask-a-block.html. [Accessed 29 07 2021].

[50] dSPACE, "Implementation software for running models on dSPACE hardware," [Online].

Available: https://www.dspace.com/en/inc/home/products/sw/impsw/real-time-interface.cfm#175_25036. [Accessed 29 07 2021].

[51] Atlassian, "Jira Software," [Online]. Available: https://www.atlassian.com/software/jira. [Accessed 29 07 2021].

[52] git, "git --distributed-even-if-your-workflow-isnt," [Online]. Available: https://git-scm.com/. [Accessed 29 07 2021].

[53] MathWorks, "Generate C Code from Simulink Model," MathWorks, [Online]. Available: https://www.mathworks.com/help/dsp/ug/generate-c-code-from-simulink-model.html. [Accessed 24 07 2021].

[54] M. B. Stevens, "Hybrid Fuel Cell Vehicle PowertrainDevelopment Considering Power Source Degradation," University of Waterloo, Waterloo ON, 2008.

[55] M. Giannikouris, "Design and Control of a Unique Hydrogen Fuel Cell Plug-In Hybrid Electric Vehicle," University of Waterloo, Waterloo, ON, 2013.

[56] G. P. Singh, "Experiential Learning withRespect toModel Based Design Applied to Advanced Vehicle Development," University of Waterloo, Waterloo, ON, 2014.

[57] P. Ellsworth, "Increasing Efficiency of Hybrid Electric Vehicles Using Advanced Controls," University of Waterloo, Waterloo, ON, 2016.

[58] MathWorks, "Simulink Test - Develop, manage, and execute simulation-based tests," [Online]. Available: https://www.mathworks.com/products/simulink-test.html. [Accessed 30 07 2021].

[59] MathWorks, "Unit Delay," [Online]. Available: https://www.mathworks.com/help/simulink/slref/unitdelay.html. [Accessed 05 08 2021].

[60] dSPACE, "RTI CAN MultiMessage Blockset," [Online]. Available: https://www.dspace.com/en/inc/home/products/sw/impsw/rti_can_multimessage_blockset.cfm. [Accessed 05 08 2021].

[61] MathWorks, "Stateflow - Model and simulate decision logic using state machines and flow charts," [Online]. Available: https://www.mathworks.com/products/stateflow.html. [Accessed 28 07 2021].

[62] Google Earth, "WRESTRC Test Track Map," [Online]. Available: https://earth.google.com/earth/d/1GWf91cEFt59qpFC8GAZ4_g39w0DyIs00?usp=sharing. [Accessed 29 August 2021].

[63] MathWorks, "Powertrain Blockset," [Online]. Available: https://www.mathworks.com/products/powertrain.html. [Accessed 09 08 2021].

[64] Samsung, "Specification of Product," 27 11 2014. [Online]. Available: https://sep.yimg.com/ty/cdn/theshorelinemarket/SAMSUNG-INR18650-20S-DataSheet.pdf. [Accessed 10 98 2021].

[65] MathWorks, "Equivalent Circuit Battery," MathWorks, [Online]. Available: https://www.mathworks.com/help/autoblks/ref/equivalentcircuitbattery.html. [Accessed 29 08 2021].

[66] MathWorks, "Mapped Motor," [Online]. Available: https://www.mathworks.com/help/autoblks/ref/mappedmotor.html. [Accessed 29 08 2021].

[67] General Motors, "Canadian Technical Centre McLaughlin Advanced Technology Track," 17 02 2021. [Online]. Available: https://www.gm.ca/en/home.detail.html/content/Pages/news/ca/en/2021/Feb/0217_canadian-engineers-can-now-test-vehicles.html. [Accessed 10 08 2021].

[68] P. J, F. G, B. S, C. B, K. M A and G. T, "How accurately can we measure vehicle fuel consumption in real world operation?," *Transportation Research Part D: Transport and Environment,* vol. 90, no. 0, 2021.

[69] D. Splitter and J. Szybist, "Intermediate Alcohol-Gasoline Blends, Fuels for Enabling Increased Engine Efficiency and Powertrain Possibilities," *SAE International,* vol. 1, no. 1231, pp. 29-47, 2014.

[70] P. K. Tarei, P. Chand and H. Gupta, "Barriers to the adoption of electric vehicles: Evidence from India," *Journal of Cleaner Production,* vol. 291, 2021.

[71] S&P Global, "Global Auto Sales Forecasts: The Recovery Gears Up," S&P Global, 11 05 2021. [Online]. Available: https://www.spglobal.com/ratings/en/research/articles/210511-global-auto-sales-

forecasts-the-recovery-gears-up-11952032#:~:text=Related%20Research-,Key%20Takeaways,and%20Europe%20likely%20lagging%20behind.. [Accessed 03 07 2021].

[72] S. Onori, L. Serrao and G. Rizzoni, "Forward and Backward Modelling Approaches," in *Hybrid Electric Vehicles - Energy Management Strategies*, London, Springer, 2016, pp. 10-13.

[73] MathWorks, "Why Use Model-Based Design?," MathWorks, [Online]. Available: https://www.mathworks.com/solutions/model-based-design.html. [Accessed 22 July 2021].

[74] dSPACE, "Compatibility of dSPACE products with MATLAB Releases," dSPACE, [Online]. Available: https://www.dspace.com/en/inc/home/support/supvers/supverscompm/mlcomp.cfm. [Accessed 22 July 2021].

[75] dSPACE, "Model Based Development (MBD)," [Online]. Available: https://www.dspace.com/en/inc/home/products/systems/mbd.cfm. [Accessed 04 08 2021].

Appendix A - RTM Types & Identifiers

Type Descriptor (ABCD-***-*.**)

ID	Type	Description
MECH	Mechanical design	Anything related to the mechanical team system design (ex fuel system, thermal systems)
ELEC	Electrical design	Anything related to the electrical team system design (schematics, fuses or system loops)
ALGO	Supervisory Controller, CAV	Algorithm requirements for controllers (includes CAV and HSC, can extend to additional controllers if they can be programmed)
USER	User Interaction	Requirements for physical components the driver interacts with – HMI, steering wheel, gas pedal etc.
INTG	Integration	Requirements for verification of tests completed, systems integrated according to best practices and standards
MAIN	Maintenance	Requirements for maintaining integrity of vehicle (calibration, etc)

Component/Subsystem Identification (****-EFG-*.**)

ID	Subsystem
MOT	Motor
ENG	Engine
FUEL	Fuel System
BAT	Battery
TRN	Transmission
CAV	Autonomy sensors
BDY	Vehicle features
DVLN	Driveline
DYN	Dynamics
AEB	Automatic emergency braking
ACC	Adaptive Cruise Control
LCC	Lane Centering Control
OBJ	Object identification
COM	Communications
HV	High Voltage
LV	Low Voltage
EXST	Exhaust
GEN	General
HIL	Hardware-in-the-Loop

HMI	Human Machine Interface
HSC	Hybrid Supervisory Controller
SSPN	Suspension
THRM	Thermal
PWTN	Powertrain
RDR	Rear Drivetrain

Appendix B - Model Configurator Script

```
1.
2. classdef modelObj < handle
3.    % modelClassBased
4.    % Model scripts rewritten using OOP. Initialize using modelClassBased
5.
6.    properties
7.       modelName % Name of the simulink file
8.       modelData % Model parameters needed
9.       EMC_DriveCycle % Model drive cycle
10.       tStop % Model termination time
11.       simOut % Simulation outputs
12.       initialized = false; % Has WheelInit been called?
13.       notes % Blank variable, use for whatever
14.    end
15.
16.    methods(Hidden)
17.       function fields = loadData2Workspace(self)
18.          % Puts model parameters in base workspace
19.          for i = 1:length(fieldnames(self.modelData))
20.             fields = fieldnames(self.modelData);
21.             assignin('base', fields{i,1}, ...
22.                   self.modelData.(fields{i,1}))
23.          end
24.       end
25.
26.       function loadDriveCycle2Workspace(self)
27.          % Puts Drive cycle parameters in base workspace
28.          assignin('base', 'tStop', self.tStop);
29.          assignin('base', 'EMC_DriveCycle', self.EMC_DriveCycle);
30.       end
31.
32.       function workspaceCleanup(self, fields)
33.          assignin('base', 'fields', fields);
34.          for i = 1:length(fieldnames(self.modelData))
35.             assignin('base', 'i', i);
36.             evalin('base', 'clearvars(fields{i,1})')
37.          end
38.          evalin('base', ...
39.             'clearvars(''fields'',''i'',''tStop'',''EMC_DriveCycle'')')
40.       end
41.    end
```

```
42.
43.    methods(Static)
44.      function cleanUpBuild()
45.      % Cleans up the build folder
46.         pattern = ('.sdf');
47.         dinfo = dir;
48.          for i = 1:length(dinfo)
49.            directory = dinfo(i).name;  %just the name
50.            if directory == "build"
51.               cd(directory)
52.               build_contents = dir;
53.               for idx = 1:length(build_contents)
54.                  if build_contents(idx).isdir == 0 &&
~endsWith(build_contents(idx).name,pattern)
55.                     delete(build_contents(idx).name)
56.                  end
57.               end
58.            end
59.          end
60.         cd('../')
61.      end
62.
63.      function buildCheck()
64.      % Used to Check if current directory is the build folder
65.      current_folder = pwd ;
66.         if regexp(current_folder, '.+?build')
67.            cd('../')
68.         end
69.      end
70.
71.   end
72.
73.    methods(Access = public)% Pre simulation, data processing, preparation
74.
75.       function obj = modelObj(modelName, dataLoad)
76.          % modelClassBased Construct an instance of this class
77.          % Inputs:
78.          %   modelName: string, name of model, eg.
79.          %            'UWAFT_Blazer_P4_4WD_Opt'
80.          %   dataLoad: struct, containing all necessary model parameters
81.          % Call using: objName = modelClassBased(modelName, dataLoad);
82.
83.          obj.modelData = dataLoad;
84.          obj.modelName = modelName;
85.          load_system(modelName);
86.       end
87.
```

```
88.     function self = init(self, model_name)
89.       % Compiles model to fix rolling resistance problem
90.       % called by object, but also provides public interface to use
91.       % WheelInit
92.       % in case runModel() was not called, load data to workspace
93.       self.loadData2Workspace();
94.       self.loadDriveCycle2Workspace();
95.       if nargin == 1
96.         model_name = self.modelName;
97.       end
98.       WheelInit(model_name);
99.       self.initialized = true;
100.     end
101.     function self = config(self, buildEnv)
102.       %Builds SIL, HIL or MABx model
103.
104.       switch buildEnv
105.         case 'HIL'
106.           rti1006;
107.           target_file = 'rti1006.tlc';
108.         case 'MABx'
109.           rti1401;
110.           target_file = 'rti1401.tlc';
111.         case 'SIL'
112.           target_file = 'grt.tlc';
113.       end
114.
115.       if buildEnv == "SIL"
116.         load_system(self.modelName);
117.       else
118.         open_models = get_param(Simulink.allBlockDiagrams(),'Name'); %checking for
any open models, to close them
119.         if ~isempty(open_models) == 1
120.           for i = 1:length(open_models)
121.             if regexp(open_models{i},'UWAFT')
122.               bdclose('all')
123.             end
124.           end
125.         end
126.         %modelObj.cleanUpBuild;
127.         open_system(self.modelName)
128.       end
129.       fields = loadData2Workspace(self);
130.
131.
132.       % find goto and from blocks
133.       HSC_In = find_block_by_type(self.modelName, 'Goto', 'HSC_In');
```

```
134.        ECU_In = find_block_by_type(self.modelName, 'Goto', 'ECU_In');
135.        HSC_Out = find_block_by_type(self.modelName, 'From', 'HSC_Out');
136.        ECU_Out = find_block_by_type(self.modelName, 'From', 'ECU_Out');
137.        % delete all line connections
138.        delete_line_connections(HSC_In);
139.        delete_line_connections(HSC_Out);
140.        delete_line_connections(ECU_In);
141.        delete_line_connections(ECU_Out);
142.
143.        % comment out all blocks
144.        comment_block_list(self.modelName, {'MABx_IO', 'HIL_IO', 'SIL_IO'}, 'on');
145.        set_param(sprintf('%s/MABx_IO', self.modelName),'Commented','on');
146.
147.        % connect lines
148.        switch buildEnv
149.          case 'SIL'
150.            comment_block_list(self.modelName, {'SIL_IO', 'MABx', 'GM Blazer', 'Tank',
'Longitudinal Driver', 'Environment', 'Visualization', 'Logging'}, 'off');
151.            add_line(self.modelName, 'ECU_Out/1', 'SIL_IO/2');
152.            add_line(self.modelName, 'HSC_Out/1', 'SIL_IO/1');
153.            add_line(self.modelName, 'SIL_IO/1', 'ECU_In/1');
154.            add_line(self.modelName, 'SIL_IO/2', 'HSC_In/1');
155.          case 'HIL'
156.            comment_block_list(self.modelName, {'HIL_IO', 'GM Blazer', 'Longitudinal
Driver', 'Environment'}, 'off');
157.            comment_block_list(self.modelName, {'Tank', 'MABx', 'Visualization',
'Logging'}, 'on');
158.            add_line(self.modelName, 'ECU_Out/1', 'HIL_IO/1');
159.            add_line(self.modelName, 'HIL_IO/1', 'ECU_In/1');
160.            build_file_name = 'UWAFT_Blazer_HIL';
161.            delete_block_name = 'MABx_IO';
162.            io_sys_name = 'HIL_IO';
163.          case 'MABx'
164.            comment_block_list(self.modelName, {'MABx_IO', 'MABx'}, 'off');
165.            comment_block_list(self.modelName, {'GM Blazer', 'Tank','Longitudinal
Driver', 'Environment', 'Visualization', 'Logging', 'Tester'}, 'on');
166.            add_line(self.modelName, 'HSC_Out/1', 'MABx_IO/1');
167.            add_line(self.modelName, 'MABx_IO/1', 'HSC_In/1');
168.            build_file_name = 'UWAFT_Blazer_MABx';
169.            delete_block_name = 'HIL_IO';
170.            io_sys_name = 'MABx_IO';
171.        end
172.
173.
174.        %Build Model
175.        if strcmp(buildEnv, 'SIL')
176.          set_param(self.modelName, 'SolverType', 'Variable-step')
```

```
177.            set_param(self.modelName, 'SystemTargetFile', target_file)
178.            loadDriveCycle2Workspace(self);
179.            return;
180.          else
181.            modelObj.buildCheck()
182.          end
183.
184.          cd build;
185.          save_system(self.modelName, build_file_name);
186.          delete_block(sprintf('%s/%s', build_file_name, delete_block_name));
187.          set_param(build_file_name, 'SystemTargetFile', target_file);
188.          set_param(build_file_name, 'TRCGenerateLabels', true)
189.          set_param(build_file_name, 'BlockReduction', false);
190.          set_param(build_file_name, 'LoadAfterBuild', false);
191.
192.          if strcmp(buildEnv, 'HIL')
193.            % fastest HIL can currently run is 2 ms, which is able to
194.            % satisfy all CAN message rate requirements
195.            % if you change the HIL step time you will have to manually
196.            % update the step time of the drive cycle block!
197.            step_time = 0.002; % 2 ms
198.          else
199.            % MABx is capable of running faster than HIL
200.            step_time = 0.001; % 1 ms
201.            set_param(build_file_name, 'StopTime', 'Inf'); % MABx runs infinitely
202.          end
203.
204.          set_param(build_file_name, 'SolverType', 'Fixed-step', 'FixedStep', sprintf('%s',
step_time));
205.          open_system(sprintf('%s/%s', build_file_name, io_sys_name), 'tab');
206.          waitfor(msgbox('Please manually build RTICANMM blocks and then click OK!',
'Manual Input Required!'));
207.
208.          if strcmp(buildEnv, 'HIL')
209.            if ~isempty(self.tStop)
210.              self.loadDriveCycle2Workspace();
211.            else
212.              self.tStop= 30;
213.              self.EMC_DriveCycle = zeros(30,2);
214.              self.EMC_DriveCycle(:,1) = [1:30]';
215.              self.loadDriveCycle2Workspace();
216.            end
217.            self.init(build_file_name); % force WheelInit
218.          end
219.
220.          rtwbuild(build_file_name);
221.          close_system(build_file_name, 0);
```

```
222.          workspaceCleanup(self,fields)
223.          delete(build_file_name);
224.          modelObj.buildCheck()
225.
226.          function comment_block_list(mdl, block_names, on_off)
227.            for n = 1 : length(block_names)
228.              set_param(sprintf('%s/%s', mdl, block_names{n}), 'Commented', on_off);
229.            end
230.          end
231.
232.          function delete_line_connections(block_handle)
233.            line_handles = get_param(block_handle, 'LineHandles');
234.            for l = line_handles.Inport
235.              if l > 0
236.                delete_line(l);
237.              end
238.            end
239.            for l = line_handles.Outport
240.              if l > 0
241.                delete_line(l);
242.              end
243.            end
244.          end
245.
246.          function h = find_block_by_type(mdl, type, name)
247.            h = find_system(mdl, 'SearchDepth', 1, 'FindAll', 'On', 'Type', 'block', 'BlockType',
type, 'Name', name);
248.          end
249.        end
250.
251.        function self = openModel(self)
252.          % public interface to open the underlying simulink model
253.          open_system(self.modelName);
254.        end
255.
256.        function self = runModel(self)
257.          % Runs the simulation
258.          % Outputs the updated object containing simulation results
259.          % Run using: objName = objName.runModel();
260.
261.          % setup model to run in SIL
262.          self.config('SIL');
263.
264.          % loads ModelData and EMC_DriveCycle to workspace
265.          fields = loadData2Workspace(self);
266.          loadDriveCycle2Workspace(self);
267.
```

```
        % initialize model if necessary (but don't waste time if not)
        if ~self.initialized
            self.init()
        end

        % Runs the simulation and stores simulation outputs into object
        % for logging
        self.simOut = sim(strcat(self.modelName), ...
                'StopTime', num2str(self.tStop));

        % Clean up workspace
        % workspaceCleanup(self, fields)
    end

    function runTests(self, test_file_path, test_suite_name, test_case_name)
        %loads and runs MABx tests

            if nargin == 4
            sltest.testmanager.view;
                                test_case =
sltest.testmanager.TestFile(test_file_path).getTestSuiteByName(test_suite_name).getTestCaseB
yName(test_case_name);
            if ~self.initialized
                self.init()
            end
            run(test_case)
                                elseif nargin == 1
            sltest.testmanager.view;
            %Get path to modelObj.m
            path_folder_arr = convertCharsToStrings(split(mfilename('fullpath'), filesep));
            %Remove last two sections of path to get project directory
            path_folder_arr = strjoin(path_folder_arr(1:end-2), filesep);
            %Get all .mldatx files in requirements/Main
            file_list = dir(fullfile(path_folder_arr, "requirements", "Main", "*.mldatx"));
            %fast restart allows for reuse of a single compiled model
                                        set_param(self.modelName, 'FastRestart', 'on')
            for i = 1:numel(file_list)
                                                sltest.testmanager.load(fullfile(file_list(i).folder,
file_list(i).name));
            end
            if ~self.initialized
                self.init()
            end
            sltest.testmanager.run;
            %need to turn off fast restart to avoid issues with running
            %the model
            set_param(self.modelName, 'FastRestart', 'off')
```

```
312.          else
313.                                        disp("runTests() in class modelObj takes either 3 or no
arguments.")
314.          end
315.      end
316.
317.      function self = clearLoggers(self)
318.          % Find all ports with data logging enabled
319.          ports = find_system(self.modelName, 'FindAll', 'on', 'Type', 'Port', 'DataLogging',
'on');
320.          % Disable all logging
321.          for x=1:length(ports)
322.            set_param(ports(x), 'DataLogging', 'off');
323.          end
324.      end
325.
326.      function self = rosBuild(self, CavType)
327.         %CavType: String of 'ACC', 'AEB', or 'LCC'
328.         %All CAV Controllers follow the same format:
329.         %1 publisher for their controller output
330.         %1 subscriber to target object for AEB and ACC
331.         %1 subscriber to raw lane data for LCC
332.         %1 subscriber from drive control inputs for ACC
333.
334.         folder = cd;
335.         if(any(folder(length(folder)-26+1:length(folder)) ~=
'ecmc_architecture_modeling'))
336.             cd ..;
337.         end
338.
339.         open_models = get_param(Simulink.allBlockDiagrams(),'Name'); %checking for
any open models, to close them
340.             if ~isempty(open_models) == 1
341.                for i = 1:length(open_models)
342.                   if regexp(open_models{i},'UWAFT')
343.                      bdclose('all')
344.                   end
345.                end
346.             end
347.         newsys = new_system(CavType);
348.         buildfilename = strcat(self.modelName, '/Tank/',CavType);
349.         ACC = add_block(buildfilename, strcat(CavType,'/',CavType)); %CAV File to new
system
350.
351.         Publish = 'Publish';
352.         add_block('robotlib/Publish', strcat(CavType,'/', Publish));
353.         set_param(strcat(CavType,'/', Publish), 'topicSource','Specify your own');
```

```
    set_param(strcat(CavType,'/', Publish),
'messageType',strcat('common/',lower(CavType),'_output_msg'));
    set_param(strcat(CavType,'/', Publish), 'topic',
strcat('/',lower(CavType),'_output_msg'));

    Blank = 'Blank';
    add_block('robotlib/Blank Message', strcat(CavType,'/', Blank));
    set_param(strcat(CavType,'/', Blank),
'entityType',strcat('common/',lower(CavType),'_output_msg'));

    BusAssign = 'BusAssign';
    add_block('simulink/Signal Routing/Bus Assignment',
strcat(CavType,'/',BusAssign));

    add_line(CavType, strcat(Blank, '/1'),'BusAssign/1');
    add_line(CavType, strcat(BusAssign,'/1'), 'Publish/1');
    switch CavType
      case 'ACC'
        TargetObj = 'target_output';
        add_block('robotlib/Subscribe', strcat(CavType,'/',TargetObj ));
        set_param(strcat(CavType,'/', TargetObj), 'topicSource','Specify your own');
        set_param(strcat(CavType,'/', TargetObj),
'messageType',strcat('common/',lower(TargetObj),'_msg'));
        set_param(strcat(CavType,'/', TargetObj), 'topic', strcat('/',TargetObj));

        DriveCtrl = 'drive_ctrl_input';
        add_block('robotlib/Subscribe', strcat(CavType,'/', DriveCtrl));
        set_param(strcat(CavType,'/',DriveCtrl), 'topicSource','Specify your own');
        set_param(strcat(CavType,'/',DriveCtrl),
'messageType',strcat('common/',lower(DriveCtrl),'_msg'));
        set_param(strcat(CavType,'/',DriveCtrl), 'topic', strcat('/', DriveCtrl));

        add_block('simulink/Commonly Used
Blocks/Terminator',strcat(CavType,'/Terminator1'));
        add_block('simulink/Commonly Used
Blocks/Terminator',strcat(CavType,'/Terminator2'));

        add_line(CavType,strcat(TargetObj,'/1'),'Terminator1/1');
        add_line(CavType,strcat(DriveCtrl, '/1'),'Terminator2/1');

        add_block('simulink/Signal Routing/Bus
Selector',strcat(CavType,'/BusSelector1'));
        add_block('simulink/Signal Routing/Bus
Selector',strcat(CavType,'/BusSelector2'));

        add_line(CavType, strcat(TargetObj,'/2'), 'BusSelector1/1');
        add_line(CavType, strcat(DriveCtrl,'/2'), 'BusSelector2/1');
```

```
391.
392.             case 'AEB'
393.               TargetObj = 'target_output';
394.               add_block('robotlib/Subscribe', strcat(CavType,'/',TargetObj ));
395.               set_param(strcat(CavType,'/', TargetObj), 'topicSource','Specify your own');
396.               set_param(strcat(CavType,'/', TargetObj),
'messageType',strcat('common/',lower(TargetObj),'_msg'));
397.               set_param(strcat(CavType,'/', TargetObj), 'topic', strcat('/',TargetObj));
398.
399.               add_block('simulink/Commonly Used
Blocks/Terminator',strcat(CavType,'/Terminator1'));
400.               add_line(CavType,strcat(TargetObj,'/1'),'Terminator1/1');
401.               add_block('simulink/Signal Routing/Bus
Selector',strcat(CavType,'/BusSelector1'));
402.               add_line(CavType, strcat(TargetObj,'/2'), 'BusSelector1/1');
403.
404.             case 'LCC'
405.               %Still need raw lane data msg
406.         end
407.
408.         open_system(CavType);
409.         waitfor(msgbox('Please link bus to controller with appropriate message signals',
'Manual Input Required!'));
410.
411.         cd build
412.
413.         rosinit;
414.         set_param(CavType,'SystemTargetFile', 'ert.tlc');
415.         set_param(CavType, 'HardwareBoard', 'Robot Operating System (ROS)');
416.
417.         loadData_UWAFT;
418.         rtwbuild(CavType);
419.         close_system(CavType, 0);
420.         delete(strcat(CavType,'.slx'));
421.         clearvars -except self
422.         rosshutdown;
423.         cd ..;
424.       end
425.
426.       function self = loadLog(self, path)
427.         % This function is used to load a log file thats been converted
428.         % via blf2mat (https://github.com/uwaft/blf2mat) tool. Giving the
429.         % channel which contains VehSpdAvgDrvn will load the drivecycle
430.         % that was driven in the log
431.
432.         if self.modelName == "UWAFT_Blazer_Stock_4WD"
433.           blockPath = strcat(self.modelName,'/Drive Cycle Source');
```

```
        else
          blockPath = strcat(self.modelName, '/Longitudinal Driver');
        end
        set_param(blockPath, 'cycleVar','Workspace variable')
        set_param(blockPath, 'wsVar', 'EMC_DriveCycle');

        timeseries_table = load(path);
        try
          self.EMC_DriveCycle =
timeseries_table.timetable.PPEI_Vehicle_Speed_and_Distance.VehSpdAvgDrvn;
          self.tStop = ((size(self.EMC_DriveCycle,1)) / 10.0) - 1 ;
        catch ME
          switch ME.identifier
            case 'MATLAB:nonExistentField'
              warning('Channel does not contain VehSpdAvgDrvn')
          end
          disp(ME)
        end
    end

    function self = loadDriveCycle(self, time, speed)
        % Two ways of running this method:
        %   Loads two m by 1 vectors containing time data and speed
        %   data
        %   Call using: objName = objName.loadDriveCycle(time, speed)
        % Or:
        %   Loads in a drive cycle saved in the drive cycle block
        %   Call using:
        %   objName = objName.loadDriveCycle('nameOfDriveCycle')

        if self.modelName == "UWAFT_Blazer_Stock_4WD"
          blockPath = strcat(self.modelName, '/Drive Cycle Source');
        else
          blockPath = strcat(self.modelName, '/Longitudinal Driver');
        end

        if nargin == 3 % Time and speed vectors

          try
            self.EMC_DriveCycle = [time, speed];
          catch
            % Something to try and fit the vectors together
            error('Time and speed vectors not identical')
          end
          self.tStop = time(end);

          set_param(blockPath, 'cycleVar','Workspace variable')
```

```
480.            set_param(blockPath, 'wsVar', 'EMC_DriveCycle');
481.
482.        elseif nargin == 2 % Drive cycle string
483.
484.            try
485.                set_param(blockPath, 'cycleVar' , time);
486.            catch
487.                error(strcat(time, ' is not a valid drive cycle'))
488.            end
489.
490.            if self.modelName == "UWAFT_Blazer_Stock_4WD"
491.                baseBlockPath = strcat(blockPath, ...
492.                'Driver/Enable Drive Cycle/Drive Cycle Source');
493.            else
494.                baseBlockPath = strcat(blockPath, ...
495.                    '/Driver/Enable Drive Cycle/Drive Cycle Source');
496.            end
497.            self.tStop = get_param(baseBlockPath, 'tfinal');
498.            self.tStop = split(self.tStop, ' ');
499.            self.tStop = str2double(self.tStop{1,1});
500.
501.        else
502.            error('Incorrect number of inputs')
503.        end
504.
505.        % Verifies what the drive cycle has been set to
506.        disp(newline)
507.        disp(strcat('Drive cycle block set to: ', ...
508.            get_param(blockPath, 'cycleVar')))
509.
510.    end
511.
512.    function self = loadVTS(self)
513.        % Loads the 0-60, 50-70, 60-0 drive cycle to the object
514.        % Call using: objName = objName.loadVTS();
515.
516.       % Ensures drive cycle block is set to workspace variable
517.       if self.modelName == "UWAFT_Blazer_Stock_4WD"
518.          blockPath = strcat(self.modelName,'/Drive Cycle Source');
519.       else
520.          blockPath = strcat(self.modelName, '/Longitudinal Driver');
521.       end
522.        set_param(blockPath, 'cycleVar','Workspace variable')
523.        set_param(blockPath, 'wsVar', 'EMC_DriveCycle');
524.
525.        FILE_NAME = 'Acceleration Test Cycle.xlsx';
526.        XL_RANGE = 'A1:B430';
```

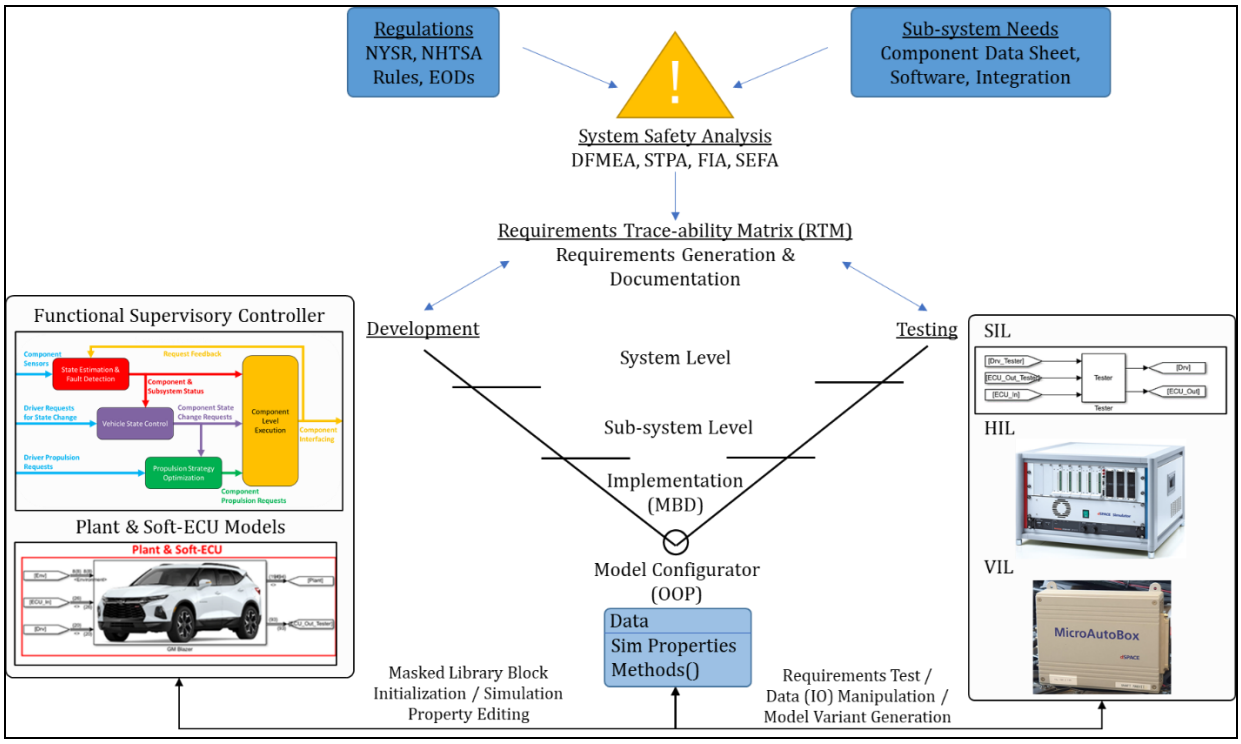
```
527.          self.EMC_DriveCycle = xlsread(FILE_NAME, XL_RANGE);
528.
529.          self.tStop = ((size(self.EMC_DriveCycle,1)) / 10.0) - 1 ;
530.       end
531.
532.       function self = loadTopSpeed(self)
533.          % Loads the max velocity drive cycle to the object
534.          % Call using: objName = objName.loadTopSpeed();
535.
536.          % Ensures drive cycle block is set to workspace variable
537.          if self.modelName == "UWAFT_Blazer_Stock_4WD"
538.             blockPath = strcat(self.modelName,'/Drive Cycle Source');
539.          else
540.             blockPath = strcat(self.modelName, '/Longitudinal Driver');
541.          end
542.          set_param(blockPath, 'cycleVar','Workspace variable')
543.          set_param(blockPath, 'wsVar', 'EMC_DriveCycle');
544.
545.          FILE_NAME = 'Top Speed Test Cycle.xlsx';
546.          XL_RANGE = 'A1:B1001';
547.          self.EMC_DriveCycle = xlsread(FILE_NAME, XL_RANGE);
548.
549.          self.tStop = ((size(self.EMC_DriveCycle,1)) / 10.0) - 1 ;
550.       end
551.
552.       function self = loadCity(self)
553.          % Loads the city drive cycle into the object
554.
555.          FILE_NAME = 'Offical EMC Drive Cycles.xlsx';
556.          SHEET = 1;
557.          XL_RANGE = 'A2:B7392';
558.          MPH_TO_SI = 0.44704; % mph to m/s
559.
560.          drive_cycle = xlsread(FILE_NAME, SHEET, XL_RANGE);
561.          % split into time and speed vectors and call loadDriveCycle
562.          self.loadDriveCycle(drive_cycle(:,1),...
563.                              drive_cycle(:,2) * MPH_TO_SI);
564.       end
565.
566.       function self = loadHighWay(self)
567.          % Loads the highway drive cycle into the object
568.
569.          FILE_NAME = 'Offical EMC Drive Cycles.xlsx';
570.          SHEET = 2;
571.          XL_RANGE = 'A2:B29622';
572.          MPH_TO_SI = 0.44704; % mph to m/s
573.
```

```
574.          drive_cycle = xlsread(FILE_NAME, SHEET, XL_RANGE);
575.          % split into time and speed vectors and call loadDriveCycle
576.          self.loadDriveCycle(drive_cycle(:,1),...
577.                        drive_cycle(:,2) * MPH_TO_SI);
578.       end
579.
580.       function self = loadDegenTuning(self)
581.          % Loads the highway drive cycle into the object
582.
583.          FILE_NAME = 'degenTuningDriveCycle.xlsx';
584.          SHEET = 1;
585.          XL_RANGE = 'A1:B37012';
586.          MPH_TO_SI = 0.44704; % mph to m/s
587.
588.          drive_cycle = xlsread(FILE_NAME, SHEET, XL_RANGE);
589.          % split into time and speed vectors and call loadDriveCycle
590.          self.loadDriveCycle(drive_cycle(:,1),...
591.                        drive_cycle(:,2) * MPH_TO_SI);
592.       end
593.
594.       function self = writeNote(self, note)
595.          self.notes = note;
596.       end
597.

598.    end

599. end
```

Appendix C- Model Based Design Framework Overview



Appendix D - Plant Soft-ECU Inputs/Outputs Summary

Component	Input	Output	To Plant / To HSC
Relays	LV Voltage Power Command 1 Power Command 2 Power Command 3 Power Command 4	Inverter Relay BMS Relay ECM Relay	To Plant
		N/A	To HSC
TCM	Effective Acc Pedal Position Vehicle Speed Clutch Locked TCM RNG Value Shift Pos	Gear State	To Plant
		Gear State TCM Fault TCM State PRNDL State	To HSC
BCM	Engine Running Key State	N/A	To Plant
		Power Mode Engine State Request ECU Power	To HSC
ECM	Engine Speed Engine Torque Command Ambient Pressure ECM Relay Engine On Request ECM Rng Value Engine Fault Accelerator Pedal In Brake Pedal In Vehicle Speed In Transmission Gear Engine Torque Feedback	Torque Command Out Starter Motor On Steering Wheel Angle	To Plant
		ECM Power Torque Command Out Engine Speed Out Engine Torque PRNDL State Effective Pedal Engine Running Accelerator Pedal Brake Pedal Steering Angle CAV Decel Vehicle Speed Out Engine Coolant Temp	To HSC
BMS	Low Voltage Discharge Enable Temperature Current In Voltage In SOC Minimum Fan Speed HVIL Analog Low HVIL Analog High	Contactors Closed	To Plant
		Contactors Closed BMS On System State Current Out Usable SOC Real SOC Voltage Out Charge Current Fault	To HSC

	Resistance to Ground	Discharge Current Fault Over Voltage Fault Under Voltage Fault Over Temperature Fault HVIL Low HVIL High Ground Isolation Fault Charge Current Available Discharge Current Available Pack Temperature High Pack Temperature Low	
Inverter	Inverter Relay Motor Speed Battery Voltage Battery Current Motor Torque Control Mode Clear Error Power Limit Torque Command Restart Request Active Discharge Ambient Temperature Coolant Speed Power Loss	Actual Torque Command	To Plant
		Inverter State Motor Speed Motor Voltage Motor Current Discharge State Inverter On Motor Temp Motor Phase Current ESS DC Link Voltage Motor Temperature Inverter Power Loss Max Junction Temp Max Junction Temperature	To HSC
Coolant Pump	Pump On/Off Direction Pump Power Hold Command Pump Motor Speed Ignition	N/A	To Plant
		Output Speed Controller Status Measured Power Motor Direction Coolant On	To HSC
Coolant Fan	Hybrid Fan Speed Request	N/A	To Plant
	Engine Coolant Fan On	Fan speed	To HSC

Appendix E - Acceleration 0-60 mph Test Plan

Test Name	
Author(s)	Timothy Er, Will Stairs, Masih Bourlayee
Test Summary	To evaluate the vehicle's ability to accelerate from IVM to 60 mph as quickly as possible
Date of Testing, Duration of Test	April 16/17. Test duration, 2 hours per day
Required Test Equipment/Facilities	Facilities: • Waterloo Region Emergency Services Training and Research Centre (WRESTC) Test Track Test Equipment: • Pylons • Long reel tape measure • Walkie talkies • Stopwatch • Data logging equipment
Required Conditions	Dry track (no rain or snow)
Personnel Required	• Driver • Passenger (Internal data logging, monitor track conditions and state of vehicle) • Observer (External data collection, break distance measurement, track setup)
Vehicle Check Procedure	**Note:** The 7 steps outlined below model a vehicle shakedown check that is listed in both acceleration and braking tests. These procedures only have to be checked once, so the vehicle is assumed good for both tests once the shakedown is passed. 1. Slowly drive at 15 mph for a quarter-mile. Maneuver the vehicle left and right to check lateral movement performance. a. Listen for rubbing noises. b. Check for restricted movement. 2. Ensure regen is on. Perform a quick acceleration to 15 mph and let off the throttle, the car should slow down with regen braking. 3. Ensure regen is off. Perform step 2, the car should have a deceleration rate that is much lower than before, indicating regen is off. 4. Perform a quick acceleration to 15 mph and proceed to a hard brake. 5. If no performance issues at 15 mph, perform additional shakedowns by repeating steps 1 to 4 for 30 mph, 45 mph, and 60 mph (or top acceleration required for testing). 6. Perform a brief visual inspection of the vehicle underbody, engine bay, and electronics in the rear. Ensure nothing is loose, out of place, and that no leaks are present. 7. If no issues found, proceed to testing.
Test Setup Procedure	1. Review Acceleration EOD. 2. Mark off the starting line (shown in the diagrams below) with pylons. 3. Secure any loose objects in the vehicle so that they can't become projectiles while the vehicle is moving 4. If the acceleration test is being done before the braking test, perform the shakedown check as outlined in "Vehicle Check Procedure" to ensure the vehicle is running as expected. Otherwise, this test should already be completed. 5. Determine that the testing area is clear. If the starting line is covered in dirt and debris, sweep the area as best as possible to maximize traction. 6. Note: Make sure to maximize the SOC before every run. Drive consistently for 30kph while the passenger keeps an eye on the SOC. Stop when maximized. 7. Drive to the starting line. 8. Set up datalogger and hit record. 9. Perform event per Event Procedure. 10. Save recorded CAN file. 11. Repeat steps 5 - 10 for a total of 6 test runs. 12. Power down datalogger.
Evaluation Procedure	**Note:** Drivers must minimize excessive tire slip on launch. In vehicles with automatic transmissions, drivers are not allowed to perform "neutral drops" but are allowed to "power brake" their vehicles, given their vehicle design and if it is safe to do so. No cool-down laps are allowed for this event. It is not recommended to drive this event on wet pavement due to the reduction in traction. 1. The driver will enter a predefined course (shown in the diagrams below) and come to a complete stop at a marked starting line. 2. The length of the track being used for the IVM-60 mph test should be verified clear of obstructions and of appropriate length for the test (see Test Setup Procedure section for details). 3. The passenger should begin recording data 4. When data recording has started and the track is clear of obstructions, the driver will accelerate as quickly as possible to 65 mph in order to capture the IVM-60 mph acceleration time. a. The driver is responsible to follow safety regulations and ensure safe use of the track even if the requested speed is not reached. b. Teams should consider IVM as the point where the first non-zero vehicle speed value occurs. 5. Once the first run is complete, the passenger should end the recording. a. NOTE: Do not end the recording once 60 mph is reached. Record for a few seconds extra to ensure the data from the run is not truncated. 6. Six runs of the acceleration test will be performed. They will all be driven in the same direction due to track restrictions.
CAN Networks	GMHS Prop
CAN Signals	**Note - All of these are from the spreadsheet on ANL Sharepoint (all logged at 100 Hz):** "Z:\Testing\V3\EMC_V3_RequiredDataSignals_Rev1.xlsx"

Appendix F- Braking 60-0 Test Plan

Test Name	
Author(s)	Timothy Er, Will Stairs, Masih Bourlayee
Test Summary	To evaluate the vehicle's ability to decelerate from 60 mph to 0 as quickly as possible
Date of Testing, Duration of Test	April 16/17. Test duration, 2 hours per day
Required Test Equipment/Facilities	Facilities: • Waterloo Region Emergency Services Training and Research Centre (WRESTC) Test Track Test Equipment: • Pylons • Long reel tape measure • Walkie talkies • Stopwatch • Data logging equipment
Required Conditions	Dry track (no rain or snow)
Personnel Required	• Driver • Passenger (Internal data logging, monitor track conditions and state of vehicle) • Observer (External data collection, break distance measurement, track setup)
Vehicle Check Procedure	**Note:** The 7 steps outlined below model a vehicle shakedown check that is listed in both acceleration and braking tests. These procedures only have to be checked once, so the vehicle is assumed good for both tests once the shakedown is passed. 1. Slowly drive at 15 mph for a quarter mile. Maneuver the vehicle left and right to check lateral movement performance. a. Listen for rubbing noises. b. Check for restricted movement. 2. Ensure regen is on. Perform a quick acceleration to 15 mph and let off the throttle, the car should slow down with regen braking. 3. Ensure regen is off. Perform step 2, the car should have a deceleration rate that is much lower than before, indicating regen is off. 4. Perform a quick acceleration to 15 mph and proceed to a hard brake. 5. If no performance issues at 15 mph, perform additional shakedowns by repeating steps 1 to 4 for 30 mph, 45 mph, and 60 mph (or top acceleration required for testing). 6. Perform a brief visual inspection of the vehicle underbody, engine bay, and electronics in the rear. Ensure nothing is loose, out of place, and that no leaks are present. 7. If no issues found, proceed to testing.
Setup Procedure	1. Review Braking EOD. 2. Mark off the starting line for test directions 1 and 2 (shown in the diagrams below) with pylons. 3. Mark off the braking line for test directions 1 and 2 (shown in the diagrams below) with pylons. 4. Secure any loose objects in the vehicle so that they can't become projectiles while the vehicle is moving. 5. If the braking test is being done before the acceleration test, perform the shakedown check as outlined in "Vehicle Check Procedure" to ensure the vehicle is running as expected. Otherwise, this test should already be completed. 6. Determine that the testing area is clear. If the braking zone is covered in dirt and debris, sweep the area as best as possible to maximize traction. 7. Make sure regenerative braking is disabled. 8. Drive to the starting line of test direction 1. 9. Set up datalogger and hit record. 10. Perform event per event procedure. 11. Save recorded CAN file. 12. Drive to the starting line of test direction 2. 13. Set up datalogger and hit record. 14. Perform event per event procedure. 15. Save recorded CAN file. 16. Repeat steps 8 - 15 for a total of 3 sets. 17. Power down datalogger.
Evaluation Procedure	1. The driver will enter a predefined course (shown in the diagrams below) and come to a complete stop at a marked starting line. 2. The length of track being used for the 60-0 mph test should be verified clear of obstructions and of appropriate length for the test (see Event Planning and Setup section for details). 3. The passenger should begin recording data. 4. When data recording has started and the track is clear of obstructions, the driver will accelerate to 60 mph and, once they achieve/stabilize that speed, will begin to brake. 5. Note: the track location for the application of the brakes must be clearly marked (cones preferred) and the driver is required to apply brakes at this marker. The driver will confirm whether ABS chatter occurs. If the vehicle tires lock, the vehicle does not stop within the required distance, or the vehicle displays any unfavorable dynamics, teams should abort further testing and evaluate the vehicle for repairs or modifications. 6. The driver is responsible to follow safety regulations, even if the braking run must be called off. 7. Three sets of the braking test will be performed. Each set consists of 2 test runs, one run in each direction of the track. a. This offsets the effects of varying track grades on the results.
CAN Networks	GMHS Prop
CAN Signals	**Note - All of these are from the spreadsheet on ANL Sharepoint (all logged at 100 Hz):** "Z:\Testing\Y3\EMC_Y3_RequiredDataSignals_Rev1.xlsx"

Appendix G - Energy Consumption Test Plan

Table 1: Test Plan

Test Name	
Author(s)	Asad, Haocheng, Teo
Test Summary	Energy Consumption Testing
Date of Testing, Duration of Test	
Required Test Equipment/Facilities	• GoPro or other camera with suction cup needed • Cones • Wrong • Vector CAN cases • Working logging setup (comp excel file +) • Need a scale to measure fuel added • Appendix C printed out on a piece of paper • Bring a clipboard + Pen
Required Conditions	• The test is 36 (58 kms) miles long - track is a 1.2 km oval (48th) • Ensure that team has geographic Google Earth Screenshots of the test track with cone placements, and measured distances • **Action camera with suction cup needs to be mounted** • **Ensure that the logging works** based on this file: Z:\Workshop & Webinars\Year 3\Winter Workshop\Energy Consumption\EMC_Y3_RequiredDataSignals_Rev0.xlsx • **Ensure that the thermal pump is operational** and repeat bleed for the thermal loop • **Ensure motor has lube** • **Ensure fuel tank is brimmed to the neck** before starting the test, so more can be measured and added • **Ensure number of people in-vehicle before test is noted**
Personnel Required	• Driver (@Asad Bhatti) • Logger (@Teo Crnobrnja @Haocheng Zhang) • One more guy for the Appendix B + C (@paul Boctor)

Setup Procedure	▪ Non-competition Vehicle
	  ▪ Drive for 3-5 continuous runs of both city & hwy each
	  ▪ Setup a data logger at 10 Hz
	  ▪ Review the drive cycle to ensure looks trapezoidal and consistent ····· adjust
	  ▪ Move Cones as needed and update the Google Earth to match
	▪ Documentation
	  ▪ Ensure the car has copy of Appendix C in a table form on a clip board
	  ▪ Ensure stop times are perfectly documented to be 10 seconds
	▪ Competition Vehicle
	  ▪ Ensure all steps in appendix B are completed
	  ▪ Ensure regen is working appropriately
	  ▪ Brim the vehicle to the top of its filler neck, till spill
	  ▪ Setup Logger and ensure all signals required by comp are being shown
	▪ Drive and good luck!
	▪ Get a guy to tally laps

Evaluation Procedure	• After a full 4x run
	  • weigh the gas canister
	  • add fuel till brim
	  • weigh gas canister again

CAN Networks	▪ GMHS ▪ PROP ▪ CAV
CAN Signals	Z:\Workshop & Webinars\Year 3\Winter Workshop\Energy Consumption\EMC_Y3_RequiredDataSignals_Rev0.xlsx On GMHS: • PPEI_Vehicle_Speed_and_Distance | PTEI_Engine_General_Status_3_HS | Driven_Wheel_Gmd_Velocity_HS | Driven_Wheel_Gmd_Velocity_HS | NonDriven_Wheel_Gmd_Velocity_HS • NonDriven_Wheel_Gmd_Velocity_HS | PPEI_Driven_Whl_Rotational_Stat | PPEI_Driven_Whl_Rotational_Stat | PPEI_NonDrivn_Whl_Rotationl_Stat | PPEI_NonDrivn_Whl_Rotationl_Stat • IMU_Protected_HS | ETEI_Engine_General_Status | PPEI_Brake_Apply_Status | Exterior_Lighting_HS | PPEI_Engine_General_Status_1 | PPEI_Engine_Torque_Status_2 | PPEI_Engine_Torque_Status_3 • PTEI_Engine_General_Status_2 | PPEI_Fuel_System_Request_3 | Secure_PPEI_Transmsn_Estmtd_Gear | PPEI_Trans_General_Status_2 | ETEI_Transmission_General_Status | PPEI_IBS_Battery_Status_1 • PPEI_IBS_Battery_Status_3 | PPEI_IBS_Battery_Status_6 | PTEI_Hybrid_Transmission_Req | Body_Information_HS | PPEI_IBS_Battery_Status_2 | PPEI_Engine_General_Status_5 • PPEI_Engine_Environmental_Status | PPEI_Trans_General_Status_1 On PROP • EMD_ACT_MotorSpeed • EMD_ACT_Torque • EMD_ACT_MotorTemperature