EVS26
Los Angeles, California, May 6-9, 2012

Model-Based System Design for MIL, SIL, and HIL

Jonathan Nibert[1], Marc E. Herniter[2], Zachariah Chambers[3]
[1]*Rose-Hulman Institute of Technology, 5500 Wabash Ave, Terre Haute, IN 47803, nibertjw@rose-hulman.edu*
[2]*marc.herniter@ieee.org*
[3]*chambez@rose-hulman.edu*

Abstract

Rose-Hulman is competing in EcoCAR2, a three year competition where teams design, build, and test a hybrid-vehicle architecture. Teams are required to generate vehicle models that will be used throughout the life of the competition. The model is used to choose a hybrid architecture, design a robust control scheme, implement fault mitigation strategies, and optimize vehicle performance. Modelling techniques include Model-in-the-Loop, Software-in-the-Loop, and Hardware-in-the-Loop. This paper will discuss the techniques developed to build a model that can be actively used for the life of the three year competition and maintained across the MIL, SIL, and HIL modelling levels.

Contryl system, controller, hardware-in-the-loop (HIL), simulation, modeling,

1 Introduction

Rose-Hulman Institute of Technology is one of 16 universities competing in EcoCAR2: Plugging in to the Future [1], a three year international competition where teams are challenged to design, build, and test a hybrid vehicle architecture utilizing alternative fuels to reduce the energy consumption and emissions production of a 2013 Chevrolet Malibu. Teams are presently in year one of the competition where students choose an architecture, specify components, and design the vehicle. Design includes both the mechanical integration of the parts as well as design of the supervisory control system for the hybrid system. Year two of the competition is the build phase, and year three is the optimization and refinement phase.

To facilitate the process, teams are required to generate vehicle models that will be used throughout the life of the competition. In year one, the model is used to model stock vehicle performance, choose a hybrid architecture, and come up with a basic control scheme. In year two, the model is used to develop a robust hybrid control scheme, implement fault mitigation strategies, and gracefully start up and shut down the vehicle. In year three the model is used to optimize vehicle performance and increase the amount of fault detection and mitigation. As a result, the model is used throughout the life of the competition, from initial design concepts to final enhancements. Safety in the competition is paramount. Any changes to the control systems of the vehicle must be tested extensively before the vehicle is given clearance for open road testing.

Teams are required to use several modeling techniques including Model-in-the-Loop (MIL), Software-in-the-Loop (SIL), and Hardware-in-the-Loop (HIL) [2, 3]. Changes to the controller require that it be subjected to a prescribed battery of tests using all three levels of simulation before the

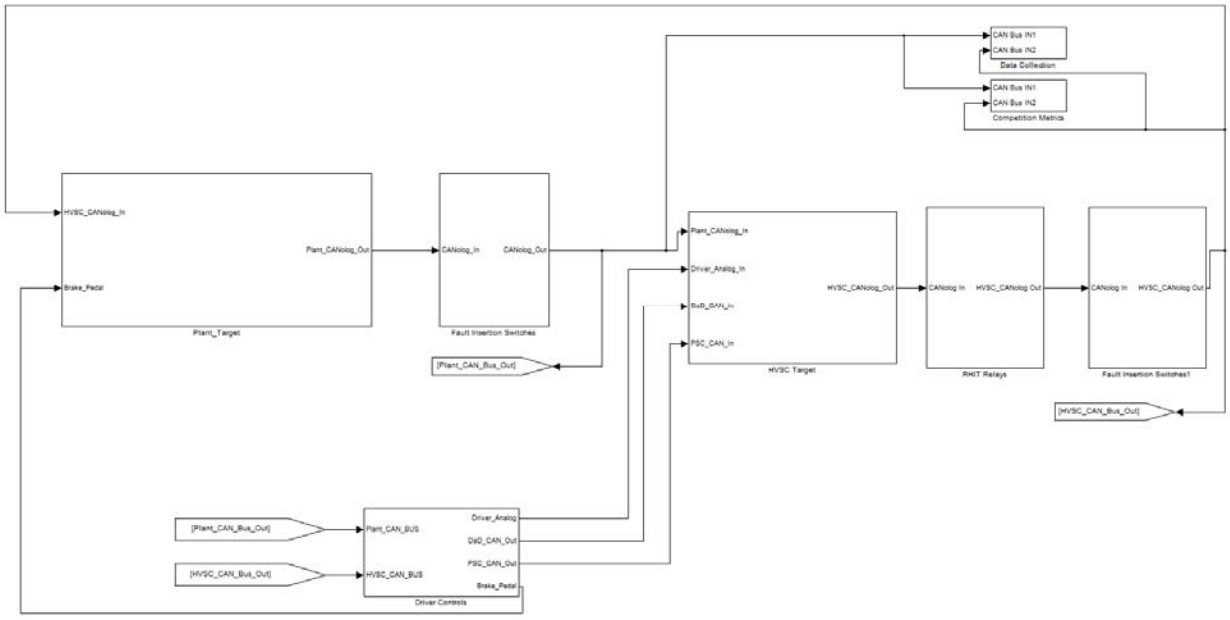

Figure 1: Top-level system model

modified control algorithm is deployed in the vehicle. Thus, the three models are to be used throughout the life of the competition. This will be the third advanced vehicle technology competition in which Rose-Hulman has participated. Experience has shown that it is very difficult to keep three different models consistent, especially when one model involves hardware and introduces signals required in the hardware realization but not required to simulate vehicle performance. Example signals are a heartbeat signal to enable the power steering controller or a signal to enable the instrument cluster. These signals are needed in the hardware realization of the controller deployed in the vehicle, but not necessary in the MIL and SIL simulations where vehicle performance is modeled. Typically, these signals are added at a later time, and may not be added to all three models. If care is not taken, the supervisory control blocks in the MIL, SIL, and HIL models diverge over time and it becomes difficult to test controller charges at all three levels.

To avoid this problem, the same control block must be used for all three models. Thus, if the controller is changed in one model, the entire controller can/should be copied and pasted into the other two models. (Or, the controller can be maintained in a library and changes in the controller are reflected in all models that reference the library.) Implementing a model that can be used for MIL, SIL, and HIL requires a certain structure in the model layout, careful choice in selecting the boundary between subsystem components, and discipline in choosing and routing signals.

This paper will discuss the model architecture and methodology used by Rose-Hulman to build a model that can be actively used for the life of a three year competition and maintained across the MIL, SIL, and HIL modeling levels. The three levels of modeling and rules for maintaining the models will be discussed.

The top-level layout of the model is shown above in in Fig. 1. For purposes of discussion in this paper, we will discuss the Hybrid Vehicle Supervisory Controller (HVSC) block. However, the layout discussed applies to the plant and two other controllers contained in the model. Note that the HVSC block is labeled as “HVSC Target” meaning that everything contained within this block will be deployed or realized on a microcontroller target. The contents of the HVSC Target block are shown below in Fig. 2.

The HVSC Logic block is the same for the MIL, SIL, and HIL models. In the MIL model, the block contains only MathWorks Simulink blocks. To impose that the HVSC Logic subsystem use only non-continuous blocks, the subsystem is declared as atomic and given a fixed time step. In the SIL Level,

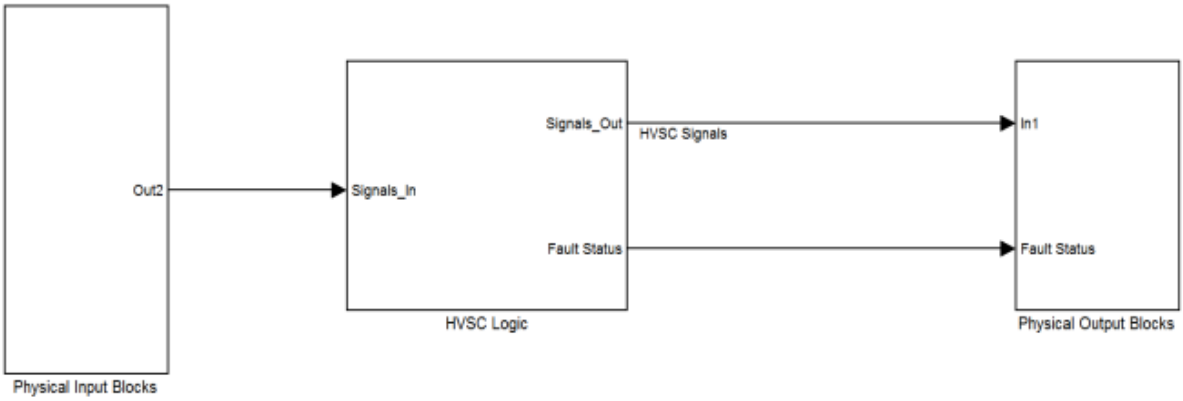

Figure 2: HVSC target block

the HVSC Logic block is compiled into C code, the same code that will be deployed on the target. At the MIL and SIL levels, the Physical Input and output subsystems are pass-through blocks. At the HIL Level, these blocks are replaced by Analog I/O and CAN communication blocks. In all cases (MIL, SIL, and HIL), the logic for the HVSC Logic block and the signals are the same. Thus a change in the HVSC Logic in one model is automatically changed for the other models as the blocks are logically identical.

2 Development Model Flow

Throughout the controller development process, the model moves through the MIL, SIL, and HIL testing stages in that respective order. The overall flow is linear but highly iterative since any changes made due to results in later stages can force the model back to earlier stages for retesting, depending on the extent of the alteration. Figure 3 below illustrates the model flow, including deployment.

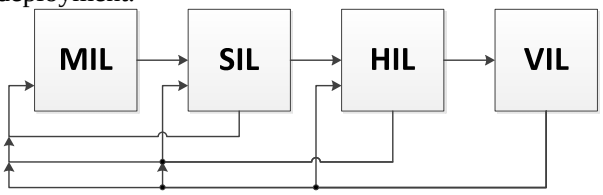

Figure 3: Model Development Stages

At the initial MIL stage, the model is run in the native development environment, such as Simulink in the case of this writing. The connection between subsystem components takes the form of a signal bus, regardless of how the signals may be implemented physically and the model runs in processor-time, that is to say often much faster than real-time and not necessarily at a fixed time-step. At the SIL stage the nature of the connections between subsystem components remains the same as in MIL, but the model is now compiled into C code and run with a fixed time-step. At the HIL stage, the model is separated into two distinct subsections; a plant model and a controlled. The plant model is compiled and run on a real-time simulation platform, while the controller is compiled and programmed to the target hardware that will be used in deployment. At this stage, the physical plant and controller subsystem component connections changes to reflect the physical implementation of the signals as will appear in the deployment hardware. Instead of a single signal bus passing all data between components, the signals are now broken apart and communicated in different way such as serial busses, discrete digital lines, analog signals, and potentially other methods depending on the system being developed. For deployment, the plant model is abandoned, and the compiled code on the target hardware is connected to the physical system. The changes the model must go through in the development flow are mapped out below in Fig. 4.

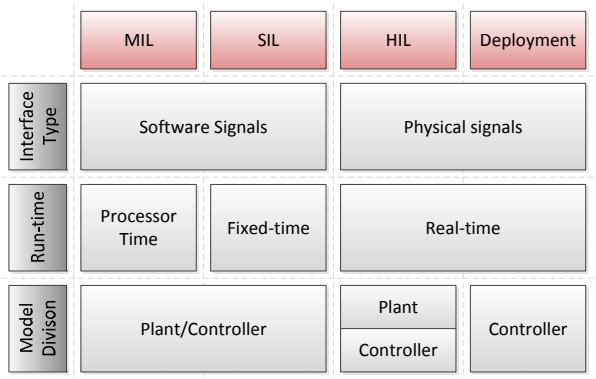

Figure 4: Model Variations Throughout Development

The figure illustrates the numerous changes a model must go through when progressing through the development cycle, as each break between blocks denotes a fundamental alteration to the model in terms of run-time, interfacing, or subdivision. The progressive alteration of the model to meet the changing requirements at each stage of development is a core concern in system modeling since it impacts development overhead, reusability, and model fidelity.

3 Model Concurrency

As different subsystem communication types and compilation requirements are present at each of the different stages of development, a single model is not sufficient to serve all the development purposes. The initial approach to solving this problem was to develop multiple models and maintain them concurrently. In this way, a model with pass-through oriented subsystem communication as used by the SIL and MIL stages was developed, and then an implementation-oriented model was developed for use in the HIL and deployment stages. The advantage to this approach was that the models could be heavily integrated in nature, thereby making initial development simpler. The cost of the approach was in the difficulty of maintaining multiple models concurrently. Since two full system models were being kept and maintained, any functional alteration had to be made twice. This lead to an eventual abandonment of the MIL/SIL stage model as the overall project moved into the HIL/Deployment stages, thereby crippling the ability to go back and do full MIL-SIL-HIL testing of functional changes.

An additional challenge was creep in signal addition at the deployment stage. When the actual physical system deviated from the physical plant model, appropriate functionality was added to the controller. The problem that arose with this is that the signals needed for the functionality were added to the controller, but not documented, and therefore not also updated in the physical plant model to account for the new understanding of the functionality. This unchecked addition of signals caused the physical plant model to deviate widely enough from the actual physical system that interfacing the controller to the HIL became difficult and also resulted in a later abandonment of the HIL system.

The heavy integration in each of the models was also problematic when dealing with maintaining and debugging. In many cases the models were heavily integrated, causing functional and interface components would be blended together, making it difficult to tell if the problem was in the function or how it was being communicated.

In an effort to resolve the problems with concurrent model maintenance, a new approach has been developed for the EcoCAR2 modeling process. In this new approach, a high level of modularity is being used to decouple the functional and interface components of subsystems from each other. In this way, the functional components will remain the same at each of the MIL/SIL/HIL/Deployment stages, with only the interface components changing to suit the needs at the specific development stage. In this way, only one functional model will need to be maintained, and whether errors are functional or interface-based will be more explicit.

4 Subsystem Component Layering

The core idea behind the modular approach to model layout deals with the decoupling of the functional and interface parts of subsystem components. In order to accomplish this, the system was examined and a set of layers defined. Figure 5 shows the breakdown of these layers, and highlights that the primary concern of the modeling lies in the software breakdown into functional and interface layers atop the hardware.

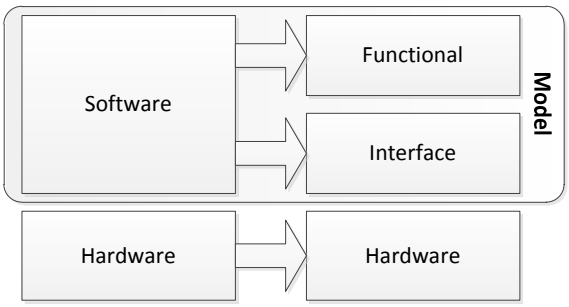

Figure 5 - Model Layering Breakdown

The hardware is the target hardware being used to run the model, and includes the processor, memory, interfacing, and any other physical components of the system. The software is either the control code or physical plant simulation code. The division of the software into a functional layer and interface layer serves to allow development of a single functional layer that can then be united with different functional layers to allow use in the different development stages without having to alter the functional logic or maintain multiple copies of it across different model variants.

The functional layer is defined as any portion of the model which is responsible for making decisions based on data, or performing any dependent processing such as combining data or performing dynamic scaling. The interface layer is defined as any portion of the model which is responsible for data entering or leaving a subsystem. Some data operations are permissible within the interface blocks, but must be static in nature, such as a fixed scaling or offset.

Centralizing the functional logic will eliminate the variants used in the prior multi-model approach, and thereby decrease the work required to update more than one model when functional changes are made. More importantly, the centralization aids with model integrity. When multiple variants of a model must be maintained, there is an increased risk of model drift, whereby the variants may become differentiated in unintended ways, causing undesirable behavior, and increasing the difficulty of debugging.

The separation of the functional and interface layers means that when moving between development stages, only the interface layer must be changed if the model is correctly designed using the layering approach. If the interaction between the layers is standardized, which will be discussed later in this paper, then a socket-style approach may be used. In this approach, a single functional block for a subsystem may be developed, and then connected to any different interface layer "socket" within a development stage-specific system-level model that will appropriately alter the signals used by the functional block to exist in the that development stage. This allows a specific system skeleton to be developed for each development stage, one each for MIL, SIL, and HIL. Deployment does not require a system skeleton, is reuses the HIL variant without the physical plant model. Using the socket approach, multiple variants still have to be maintained, but they are lighter-weight models, and do not contain any functional logic, only interfacing layers.

An example of layered model implementation is given below in Figure 6 6. The example is analogous to **Error! Reference source not found.**, but is simplified and shows to which layer the model blocks would belong. The figure clearly shows that any data coming into or going out of the HVSC subsystem must pass through an interface layer block before being processed by the functional layer block.

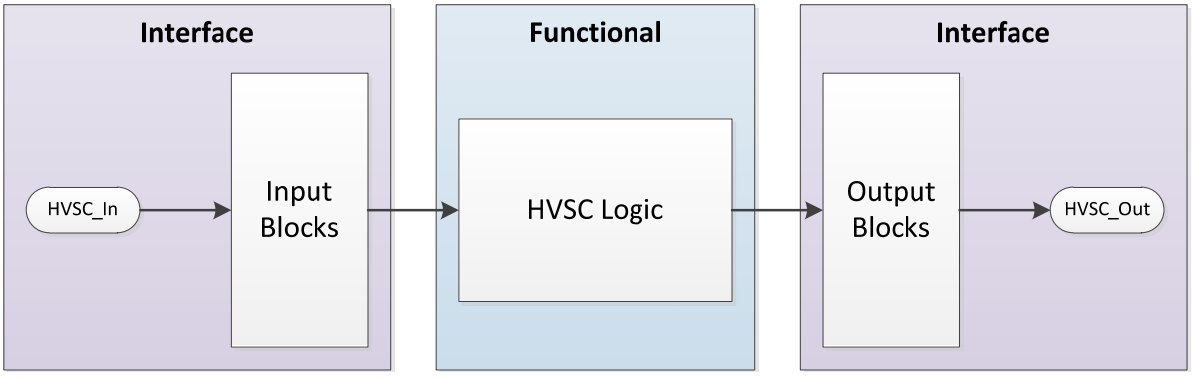

Figure 6 - Example Model Layering Implementation

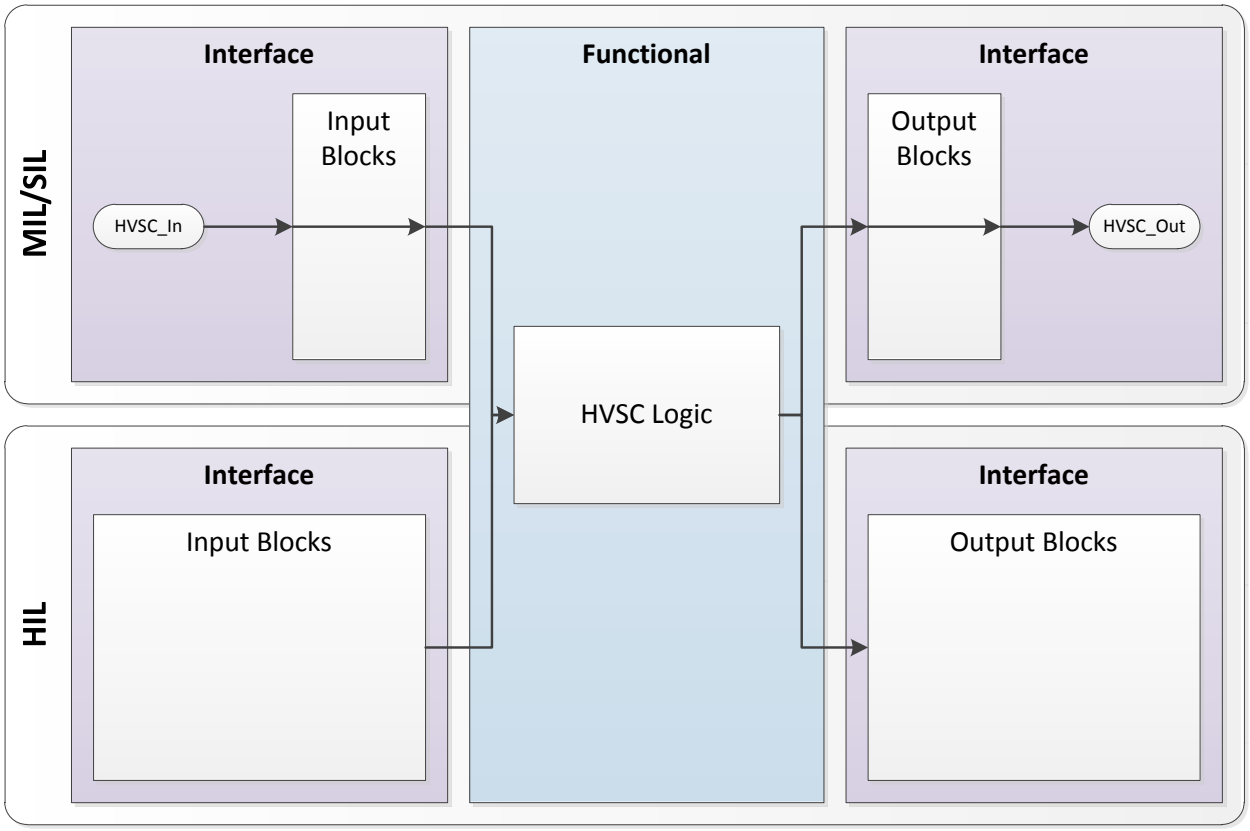

Figure 7 - Multi-Stage Model Laying Implementation

Figure 7 7 shows an expanded version of the same HVSC subsystem model, but this time illustrates the socketing idea.

In the interface layer of the MIL/SIL variant the input and output blocks are simply pass-throughs that allow a signal bus to be feed directly in from another block in the model, and directly out in the same fashion. At the HIL stage though, the signals must be input and output over physical connections, and so the blocks contain appropriate components to read and write data physically, and the direct data connection to the rest of the system is done away with. The importance is that in both cases, the same functional block is used regardless of how data is transferred. Following this principle, so long as the functional block receives and generates the appropriate raw signals, any number of interface blocks can be mated with it to alter how the data is communicated to the rest of the system.

5 Layer Interfacing Standards

In decoupling the functional and interface layers within the model, care must be taken to maintain a high level of cohesion between the layers as well. Maintaining high cohesion becomes additionally difficult as more interface sockets are generated, since the functional layer must connect to all of the matched interfaced layers. Because of this need for cohesion with multiple variants, standardization of the layer interaction is important. To establish these standards, the general nature of the interaction between layers must first be examined. Figure 8 8 shows the model layers again, and shows the general nature of what is passed between the layers.

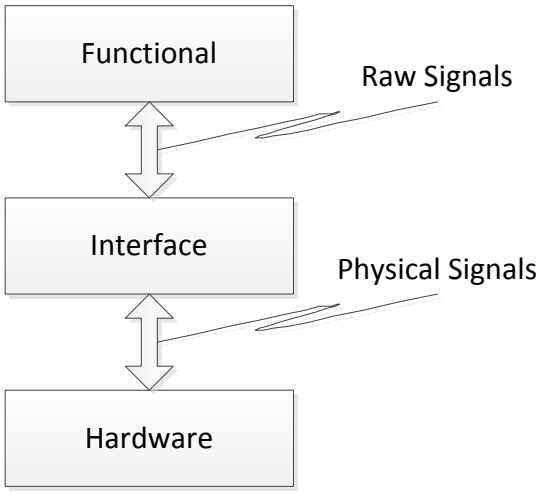

Figure 8 - Layer Interfaces

The communication between the functional and interface layers takes the form of raw signals. These are comprised of named data and the values therein. Being raw, any specifics of transmission such as scaling or offsets do not need to be applied to the data values. Between the interface layer and the hardware the interaction is denoted as physical signals. At this point, the raw signals have been processed by the interface layer, and are have been appropriately transformed for transmission between subsystem blocks.

The standards applied for the physical signals will be dictated by the specific nature of the hardware used at each of the development stages, such as processor architectures, communication protocols, and other hardware standards. As such, the interface-hardware layer interactions are beyond the scope of this writing, and instead the raw signals will be closely examined.

The requirements placed upon the raw signals are that all physical signals expected by the system from the subsystem must be produced by the functional layer, and all physical signals provided to the subsystem must be provided to the functional layer. Additionally, all raw signals produced by the functional or interface layer must be within a given bound for the particular signal, as dictated by the signal documentation. The importance herein is that all signals must exist and have a specific range. In this way, multiple layers may be produced, and so long as they provide all existing signals within bounds, they will be able to communicate.

In the case of this writing, since Simulink is being utilized for model development the standardization of communication between the layers is accomplished by requiring that the blocks representing each layer, as shown previously in Figure 6 6, may only communicate via a single bus structure. A portion of this bus structure for the communication between the input and logic blocks of the HVSC example is shown in Figure 9. The bus structure allows each signal to be checked; therefore helping ensure the aforementioned existence requirement is met for each signal expected by the HVSC logic block this bus is connected with.

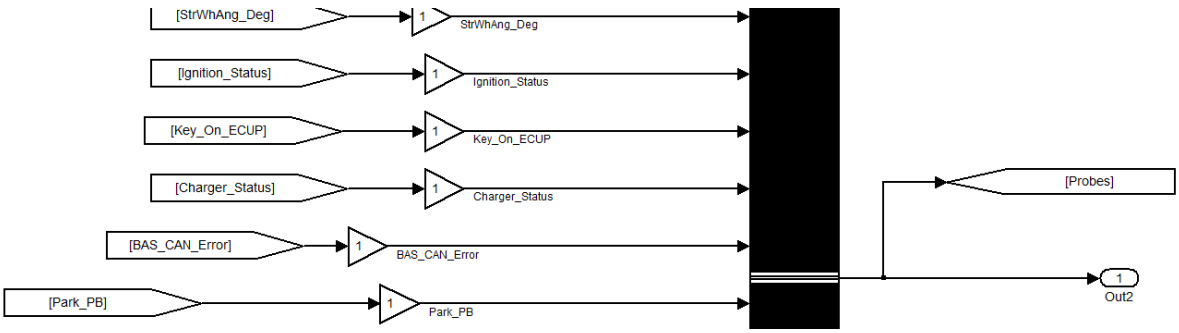

Figure 9 - HVSC Input Block Bus Structure Example

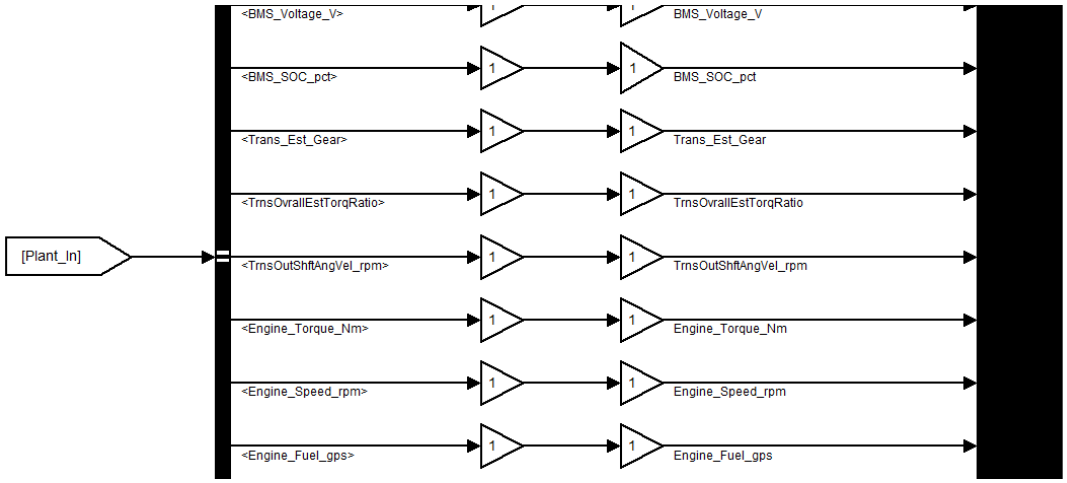

Figure 11 - MIL-SIL HVSC Input Block Example

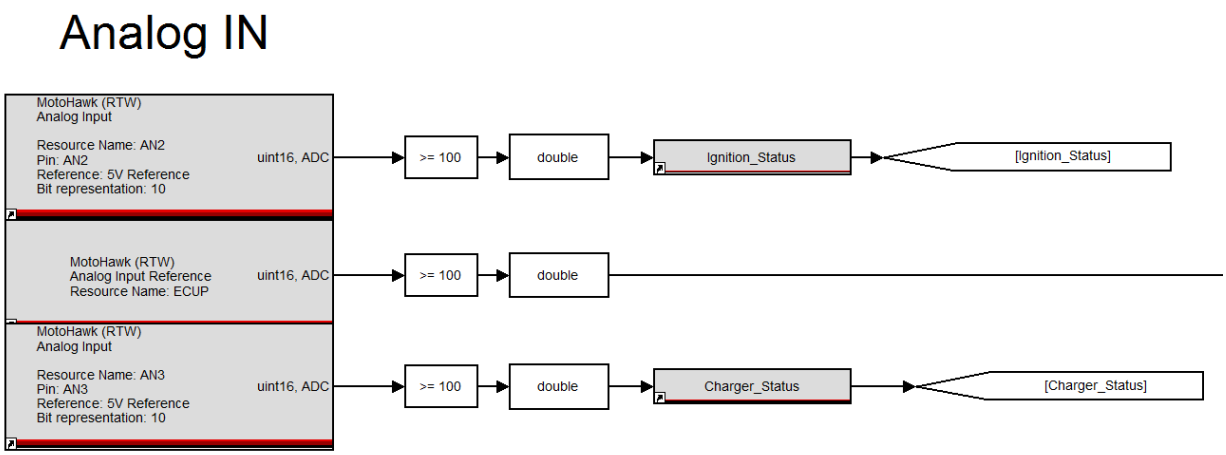

Figure 10 - HIL-Deployment HVSC Input Block Example

Based on this bus structure, how the different versions of the HVSC input blocks are created can be examined to show how both the pass-through approach of MIL-SIL and the physical communication blocks for HIL-Deployment can be applied to generate the same bus. The MIL-SIL version is shown below in Figure 11, and shows a single bus structure input from the plant model that is then expanded, buffered, and connected to the bus generation structure from Figure 9. Alternatively, the HIL-Deployment version in Figure 10 shows that no external inputs from the plant model exist, rather physical communication blocks supply the data values, which is then again connected to the bus generation structure from Figure 9.

6 Signal Documentation

In order to maintain the model's integrity and ensure that layer standards are being followed, a strict signal documentation system has been developed to track the addition and deletion of signals within the system as the model changes. The documentation ensures that the name and bounds of signals being produced by functional layers are known for consumption by interface layers, and vice-versa. The same documentation can later be utilized to specify the physical implementation of the signal, and to track the different consumers.

When a new signal is needed, a basic set of information must be compiled describing it to begin the signal addition process. The initial documentation must provide the high-level name of the signal as it will appear in the model, and bounds

and/or states, what subsystem component will be producing it, and what subsystem will be the primary consumer. Once this information has been established, the signal logic may be added to the functional layer block(s) while the interface layer blocks for both the producer and primary consumer are modified to handle the new signal. If an HIL or equivalent development stage exists where the signal must be physically transmitted, the documentation should be modified to define the method of transmission, and all pertinent data associated with that method such as scaled ranges, addresses, or other information unique to that signal. If additional subsystem components should need to consume the signal as well, they should be added to the documentation as secondary consumers, at which point their interface layers will be modified to handle the signal.

In order to avoid signal creep or accidental addition/deletion, the functional and interface layers will be controlled by different developers, or else by a single developer, but shall require the system architect's authorization to alter the functional layer. In all cases, the system architect shall serve as the holder of the signal documentation, and is responsible for authorizing and alterations outside of a layer boundary. In this way, a signal cannot be added without correctly progressing though the layering correctly, causing dependency problems between layers by coupling them.

In the specific setup being used in the EcoCAR2 model development, the system architect is responsible for assembling the top-level model and maintaining all signal documentation. The top-level model contain references the appropriate interface and functional layers for each subsystem. The different layers are in turn developed separately based on the signal documentation. The stop-level model is provided in a locked form that can be run by the developers, but not modified.

7 Future Work

Future development will focus on generating an automated test procedure for checking layer interaction requirements. Existing tests must be done by hand, with a check by developers that all documented signals are in fact produced by the appropriate layer in each subsystem. An automated system would rely on the signal documentation being held in a database which could be queried by a requirements-check script that would automatically compare the raw signals being produced against the existence requirements and notify developers if a required signal was missing, or an undocumented signal was being produced. Additional work is also planned on researching automation of system-level testing to check model integrity, signal bounds checks, external signal dependencies.

8 Conclusion

By defining the model breakdown in terms of functional and interface layers, and ensuring a standard of communication between the two layers, the layers may be successfully be decoupled while maintaining a high level of cohesion. In this way, a model can be moved through the different development stages of MIL, SIL, and HIL without having to maintain multiple variants of the full model. Instead, only lightweight variants containing just the interface layers need to be differentiated. In this way, maintaining variants is made easier since only a small subsection of the model needs to be altered between variants. More importantly, it ensures integrity of the control logic since the functional layer can be decoupled from the interface layer and therefore be exactly the same in all variants of the model, regardless of hardware dependencies related to interfacing. With the future work in test automation for layer compliancy and additional testing on bounds-checking and external dependencies, this methodology should be able to provide a managed, integrity-focused approach to multi-stage model development.

Acknowledgements

The authors would like to thank The Mathworks for providing state-of-the-art modeling tools and the ongoing training and support to use them to their fullest extent. They would also like to thank Dr. Shawn Bohner for his insight into software model development and Mr. Bill Schindel for his discussions concerning system modeling. The authors would also like to thank General Motors and the Department of Energy for supporting the EcoCAR2 competition and the past competitions.

References

[1] http://www.ecocar2.org/. Date accessed 7/20/2011.

[2] "Model-based ECU development – An Integrated MiL-SiL-HiL Approach," Vivek Jaikamal, SAE World Congress & Exhibition, April 2009, Detroit, MI, USA, Paper Number 2009-01-0153.

[3] "Maximizing Test Asset Re-Use across MiL, SiL, and HiL Development Platforms," Pjeter Vuli and Michael Badalament, SAE World Congress & Exhibition, April 2010, Detroit, MI, USA, Paper Number 2010-01-0660.

[4] Verification, Validation, and Test with Model-Based Design," Tom Erkkinen and Mirko Conrad, SAE World Congress & Exhibition, April 2008, Detroit, MI, USA, Paper Number 2008-01-2709.

Authors

Jonathan Nibert is a M.S. in Engineering Management candidate at Rose-Hulman Institute of Technology, where he earned his B.S. in Computer Engineering. Jonathan's interests are in systems engineering and electrical systems integration. He is the current engineering manager for the Rose-Hulman EcoCAR Hybrid-Electric vehicle team

Marc Herniter is a Professor at Rose-Hulman Institute of Technology (Ph.D., Electrical Engineering, University of Michigan, Ann Arbor, 1989); Dr. Herniter's primary research interests are in the fields of modeling of complex systems, power electronics, hybrid vehicles, and alternative energy systems. He has worked on power electronic systems that range in power levels from 1500 W to 200 KW. He is the author of several text books on simulation software including PSpice, Multisim, and Matlab. He is currently the co-advisor for the Rose-Hulman EcoCAR Hybrid-Electric vehicle team and the faculty supervisor of Rose-Hulman's Model-Based-Systems Design laboratory.

Zac Chambers has his BS and MS in Mechanical Engineering from Rose-Hulman Institute of Technology and his PhD in Engineering Science and Mechanics from the University of Tennessee, Knoxville. He has been on the ME faculty for Rose-Hulman since 2000 and has worked with Marc Herniter as Mechanical Advisor for ChallengeX, EcoCAR, and now EcoCAR2 racking up 7 years of experience in hybrid vehicle powertrain design and execution. He additionally serves as the Program Director for the Advanced Transportation System Program at Rose.