2009-01-1021

Real Time Simulation Using Non-causal Physical Models

Tom Egel
The MathWorks, Inc.

ABSTRACT

As automotive electronics become more complex and more distributed, hardware in-the-loop simulation is now a widely adopted technique for performing controller software/hardware integration testing as well as controller/controller integration testing. Having real-time capable models that are correlated to physical hardware being controlled is key to successful implementation of hardware in-the-loop testing. Because models for hardware in-the-loop must be developed in a short amount of time and then stay in sync with the design through design changes, a best practice is to obtain such models from the system-level model used for requirements analysis and design trade offs. This way, one model can address the need of both requirements analysis and integration testing, reducing re-development of models and ensuring consistency between two process steps. While there has been significant progress made in recent years on real-time simulator technologies, including I/O accuracy, use of off-the-shelf hardware, acceleration using parallel processing, the process by which a system level simulation model is to be reused for hardware in-the-loop testing is not very well understood. This paper starts by examining options for developing a system level simulation model. When limited to causal modeling techniques, the process of creating models is often cumbersome and time-consuming. Many engineers find non-causal (or acausal) modeling methods to be much more intuitive. However, getting acausal models to run in real-time requires careful upfront planning and, when required, methodical reduction. The remaining sections of the paper deal with effective techniques for physical model development and reduction.

INTRODUCTION

Mechatronics[1] is a commonly used term for describing the combination of electromechanical physical systems with computer controls. Designers of embedded controls for mechatronic systems face difficult challenges. As illustrated in Figure 1, completion of a successful mechatronic system design requires the integration of multiple engineering domains and collaboration between the engineering teams. For example, in order to exhaustively test the software control algorithm for an ABS system requires accurately representing the physics of the electronics, hydraulics and mechanics.

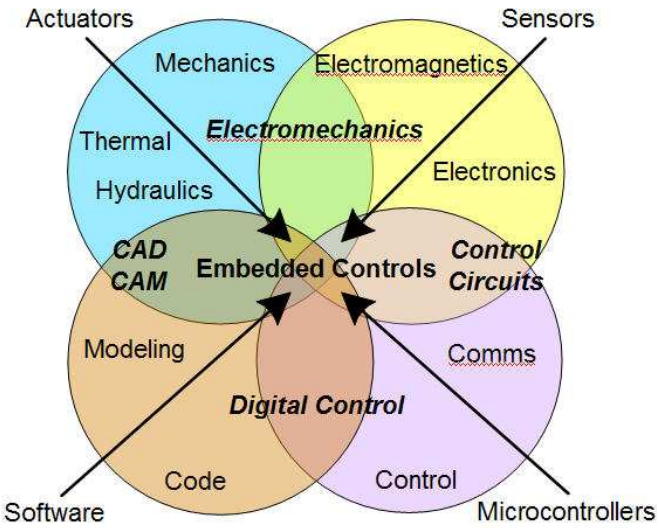

Figure 1 - Mechatronics Venn Diagram

In addition, as embedded controls continue to become more and more part of the core functionality of the modern automobile, time-to-market pressures, cost sensitivity, and quality expectations all contribute to the design challenge. Traditional methods of designing, testing, and implementing mechatronic systems cause designers to wait until late in the design effort, when actual or prototype products and real-time embedded targets become available, to find out if system actually meets the performance requirements. Only then, as system integration occurs, can the designer uncover the errors that may have found their way into the product during the early design stages.

The principles of Model-Based Design as a proven technique for creating embedded control systems[2,3], and apply equally as well when designing mechatronic systems. Using Model-Based Design, the various design teams can evaluate design alternatives without relying solely on expensive prototypes. A Model-Based Design environment allows engineers to mathematically model the behavior of the physical system, design the software and model its behavior, and then simulate the entire

system model to accurately predict and optimize performance.

Once the design performance has been verified with simulation, the next step in the process is to test the controller in real time using Hardware-in-the-Loop (HIL) simulation[4]. For the this discussion, HIL simulation will be defined as testing the control algorithm in real time by deploying it to a controller (hardware) and connecting to a model of the physical system (plant) running in real time. In order to achieve this, the physical model must be real-time capable. The main focus of this paper will be discussing the process of creating models of the physical system and deploying them to real time for HIL testing.

MODELING THE PHYSICAL SYSTEM

Model-Based Design is widely used to develop software algorithms for deployment onto an embedded controller. In order to perform closed-loop tests on the control algorithm, the first thing that is needed is a representation of the plant. There are no shortages of techniques for modeling physical systems. Some commonly used methods include signal flow diagrams[5], bond graphs[6] and even manually coding the system equations in C or Fortran. Since a mechatronic design relies on collaboration between engineering teams, it is imperative that the model can be easily shared and understood by the various stakeholders. While the methods above are perfectly valid for accurately modeling the physics, none of these are particularly well-suited for meeting the collaboration and integration needs of a multi-domain mechatronic system design. As a simple illustration, consider the problem of modeling a DC motor with speed and current control.

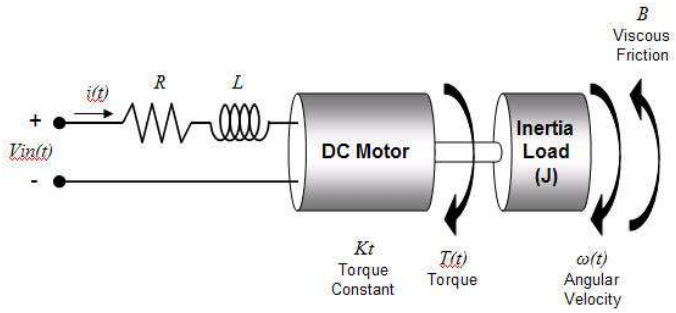

Figure 2 – DC Motor Architecture

SIGNAL FLOW APPROACH **-** Simulink by The MathWorks is widely used to design control algorithms using the signal flow approach. Once implemented in Simulink®[7], Model-Based Design methods are commonly used to verify the controller design and automatically generate the code for deployment onto the microcontroller for rapid prototyping and production. As a result, the signal flow method has also historically been used to model the plant in Simulink to test the controller in simulation and with a real-time hardware-in-the-loop system.

A common representation of a DC motor is shown in **Figure 2**. The signal flow modeling approach is a multi-step process that first requires deriving the motor equations:

$$T = K \cdot i - B \cdot \omega - J \cdot \frac{d\omega}{dt} \quad (1)$$

$$V = K \cdot \omega + i \cdot R + L \cdot \frac{di}{dt} \quad (2)$$

The next step is to graphically model these equations in a signal flow diagram, but this often requires reformulating the equations to support this approach:

$$\frac{di}{dt} = \frac{1}{L}\left(V - i \cdot R - K \cdot \frac{d\theta}{dt}\right) \quad (3)$$

$$\frac{d\omega}{dt} = \frac{1}{J}\left(K \cdot i - B \cdot \frac{d\theta}{dt}\right) \quad (4)$$

Finally, a signal flow model of the equations can be created:

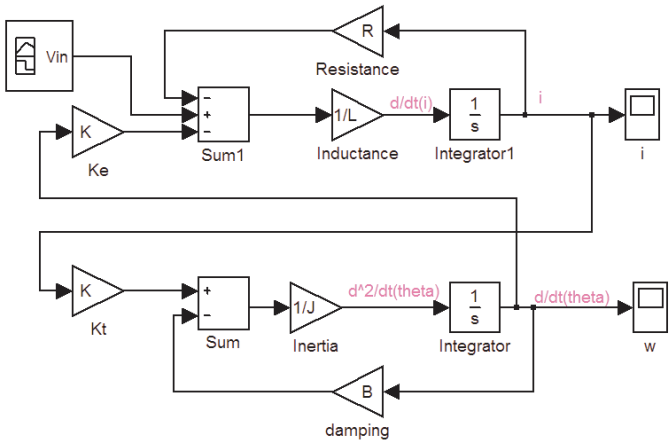

Figure 3 - DC Motor Signal Flow Model

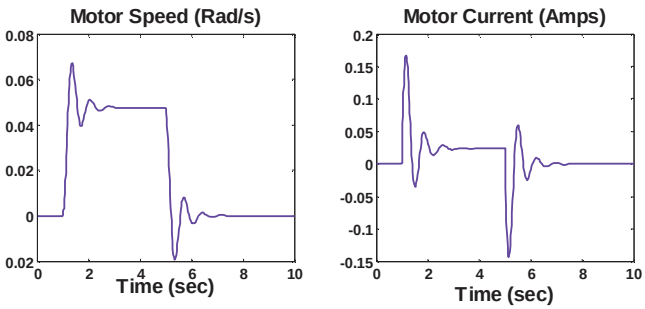

Figure 4 - DC Motor Model Simulation Results

Simulating this model yields the expected results (see Figure 4), but the multi-step modeling process results in a model that is unrecognizable when compared to the original diagram in **Figure 2**. Even for this simple model, in order to share with others it would require significant explanation and documentation.

NETWORK APPROACH - A more universal method for modeling multi-domain physical systems is often referred to as the network modeling approach[8]. Its origins come from the method of network analysis for electrical systems, and can been extended to also model systems consisting of mechanical, hydraulic, thermal and magnetic components. The main advantage of a network model over a signal flow model is the non-causal or sometimes called acausal[9] nature of the connection ports. In signal flow diagrams, the connections are causal. That is, every block is a transfer function with a signal on the input causing the output to behave according to the defined transfer function. A quick look at the model in Figure 3 illustrates how data flows through the model. Any interaction between blocks must be explicitly modeled by creating feedback loops. As the interactions become more complex and commonplace as with a mechatronic system, the signal flow method quickly becomes untenable for all but the most expert users. For example, if additional effects like damping, friction or hard stop limits are desired, the system equations (1) and (2) would need to be reformulated and the model recreated, resulting in an even more complicated model that is more difficult to interpret.

To illustrate, let's look at the same DC motor model using the network approach modeled using foundation blocks from the Simscape™[10] multi-domain physical modeling environment within Simulink:

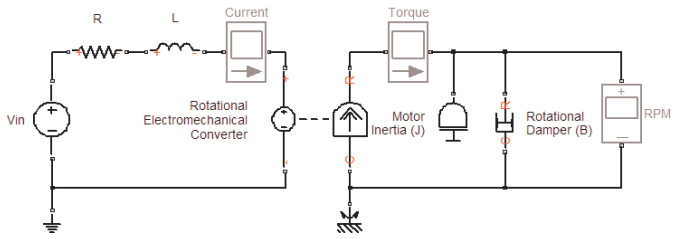

Figure 5 - DC Motor Network Model

As you can see, the model bears a close resemblance to the original diagram in **Figure 2**. This is an important benefit of the network approach making it much easier for others to understand and interpret thus fostering the collaboration needed for designing mechatronic systems. The electrical side of the model solves for the quantities current and voltage while the mechanical side solves for torque and angular velocity resulting in identical simulation results to the signal flow model. In the network terminology, these quantities are commonly referred to as "through" and "across" variables. Notice how current and torque "scopes" are placed in the network to measure the through variables and the RPM scope to measure the across variable of motor shaft speed. These measured quantities can also be easily fed back to the control algorithm modeled in Simulink for closed-loop system analysis (more on this later).

A major advantage of the network approach is the ability to quickly modify the system model without the need to derive the system equations. Here, the individual "blocks" contain the fundamental component equations defining the relationship between the through and across variables. The system equations (1) and (2) are then automatically formulated by interconnecting the components into the desired topology. For example the rotational damper component contains the equation:

$$T = B \cdot \omega \quad (5)$$

Defining the relationship between the through variable (torque) and the across variable (angular velocity) as a linear relationship with the damping coefficient (B) as a constant of proportionality. This method of embedding the first principle equations into the component models allows additional physical effects to be easily added to the system model without needing to worry about reformulating the overall system equations. For example, let's say we want to add limits to the angle of rotation. Using the network approach, you can simply connect a rotational hardstop to the motor shaft as shown in Figure 6.

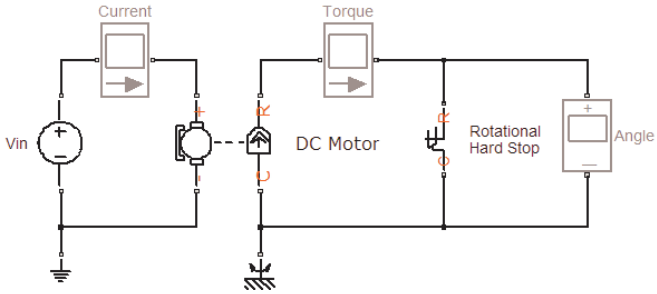

Figure 6 - DC Motor Model with Hardstop as Load

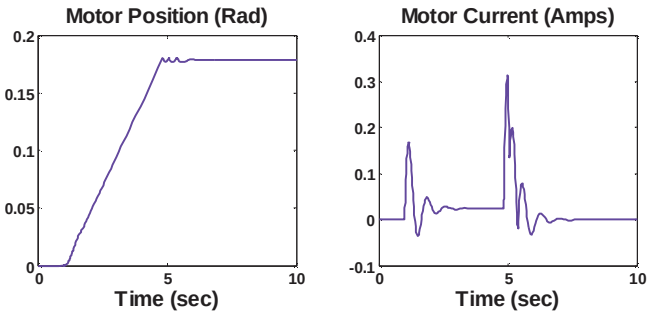

Figure 7 - Motor Position and Current with Hardstop

Here we also used hierarchy to group the previous motor model into a subsystem and added the rotational hardstop as an external load. An angular position scope is used to measure the angle of the motor shaft. As you

can see the network approach makes it easy to quickly add additional effects and immediately see the effects in simulation.

In Figure 7 you can clearly see the results of introducing angular travel limits. The motor shaft reaches the hardstop at about 5 sec resulting in an increase in the motor current as it works harder to overcome the obstacle. The motor angular position like angular velocity is an across variable that can also be fed back to the controller if desired.

MODELING LANGUAGE - The enabling technology for the network approach is a modeling language for formulating the component's characteristics equations relating the through and across variables in the various domains. The Simscape language, based on MATLAB®[11], provides the necessary constructs for modeling the multi-domain aspects of mechatronic systems. To revisit the DC motor example, the motor equations can be directly modeled using the Simscape language as shown below in Figure 8.

```
component dc_pm
  nodes
    p = electrical; % p:left
    n = electrical; % n:left
    r = rotational; % r:right
    c = rotational; % c:right
  end
  parameters
    Kt = {10 'N*m/A'};      % Torque constant
    Ke = {10 'V/(rad/s)'}; % Back EMF Constant
    Rwind = {1 'Ohm'};      % Winding Res
    Lwind = {1e-3 'H'};     % Winding Ind
    J = {1 'kg*m^2'};       % Motor Inertia
    B = {1 'N*m/(rad/s)'}; % Motor Damping
  end
  variables
    theta = {0,'rad'}; % Angular Displacement
    tq = {0,'N*m'};    % Torque thru variable
    w = {0,'rad/s'};   % AngVel across var.
    i = {0,'A'};       % Current thru var.
    v = {0,'V'};       % Voltage across var.
  end
  function setup
    through(tq,r.t,c.t); % thru variable tq
    across(w,r.w,c.w);   % across variable w
    through(i,p.i,n.i);  % through variable i
    across(v,p.v,n.v );  % across variable v
  end
  equation
      w == theta.der;
      v == Ke*w+i*Rwind+Lwind*i.der; % Motor
      tq == -Kt*i+B*w+J*w.der;       % Eq'ns
  end
end
```

Figure 8 – DC Motor model using Simscape language

This modeling method creates a new foundation component which can then be easily integrated into a larger system model by inserting it between the electrical controls and mechanical loads.

TUNING PARAMETERS - One of the challenges of any physical model is validating that the simulation results are accurate and represent reality. With the network approach to modeling, the model parameters are the degrees of freedom for adjusting the model performance. In some cases these parameters can be populated directly from the datasheet of a component manufacturer. For example, the stall torque and no-load speed curves on the motor datasheet could be used to parameterize the motor model. Many times, however, good data is not available and the model parameters must be manually adjusted. This is typically a tedious trial and error adjust-simulate-repeat process until a reasonable result is obtained. Optimization tools like Simulink Parameter Estimation™[12] can be used to automate this process by automatically tuning the model parameters by comparing the simulation results to measured lab data until a satisfactory parameter set is obtained. Figure 9 shows the results from using Simulink Parameter Estimation to automatically tune parameters of the model in Figure 5.

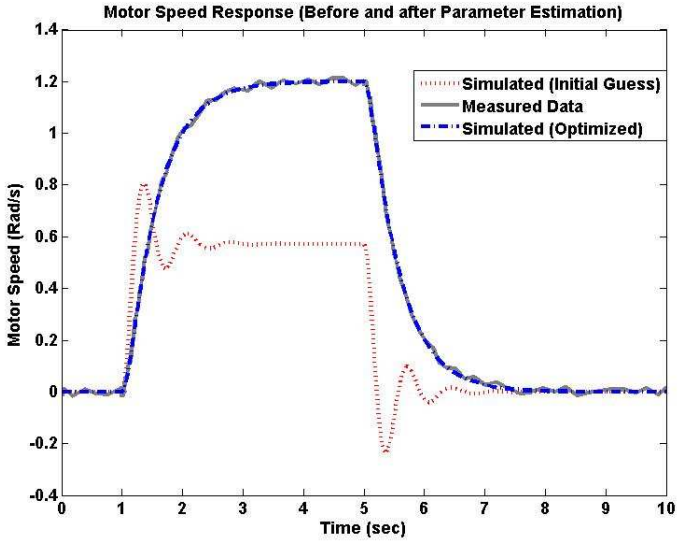

Figure 9 – Motor Speed Response (Measured and Simulated)

MECHATRONIC SYSTEM INTEGRATION

Using the network approach a complex system model can be quickly constructed by interconnecting the individual component models. The resulting model representation is intuitive and easily interpreted due to the physical connection ports. The overall system equations are automatically formulated from the individual component equations along with how they are interconnected.

As an example, consider the system modeled in Figure 10. Here the DC motor is integrated into a linear actuation system with speed control. The speed and current control subsystems are PI controllers made from

standard Simulink blocks while the rest of the model is constructed with physical blocks. The DC motor is driven by electrical PWM and H-Bridge block from the SimElectronics™[13] library.

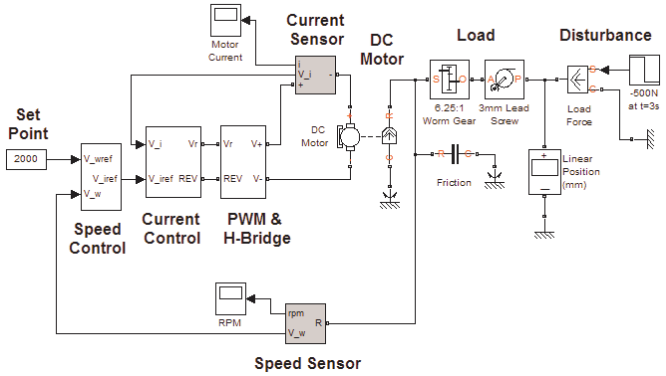

Figure 10 – Linear Actuator with Speed Control

The mechanical load includes a worm gear with friction connected to a lead screw to convert rotational to linear motion modeled in Simscape. The two inputs to the system are the constant set point signal of 2000 RPM for the desired motor speed into the controller and a disturbance force of -500N occurring at 3 seconds on the mechanical load. The goal of the simulation is to see how well the controller responds to the disturbance in closed-loop system. The network approach allows additional effects (like friction) to be easily inserted.

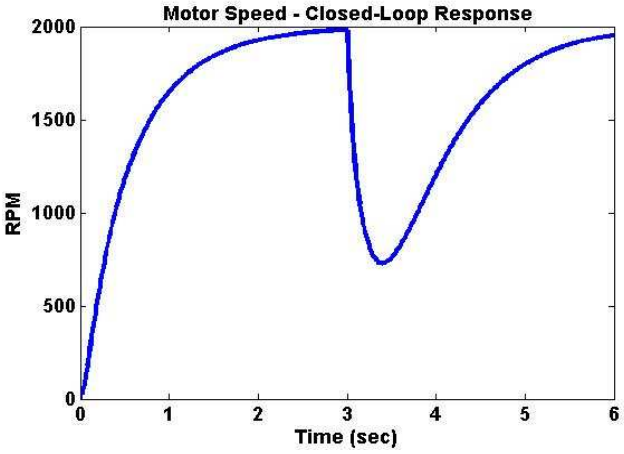

Figure 11 - Motor Speed - Closed-Loop Response

This formulation of the system equations is enabled by applying the through and across variable concept discussed earlier to the different physical domains. Table 1 shows the through and across variables used in this system.

Table 1 - Through and Across Variables

Domain	Through	Across
Electrical	Current	Voltage
Rotational	Torque	Angular velocity or position
Translational	Force	Velocity or position

The simulation results are shown in Figure 11. Here we see the motor speed increase to 2000RPM, droop to about 750RPM when the external force disturbance of 500N is applied at 3 sec, then recovers to desired set point.

CONTROL DESIGN - The risetime and recovery time are key performance attributes that can be used to determine if the design meets the requirements. The PI gains of the controller can be adjusted manually to improve the performance or automatically tuned using Simulink Response Optimization[14] as shown in Figure 12. Here we can graphically specify the response constraints (risetime, overshoot, settling time, etc…) for the optimization routine to enforce during automatic gain tuning. The result is a set of optimized gains assigned to the MATLAB workspace variables for storage and future use.

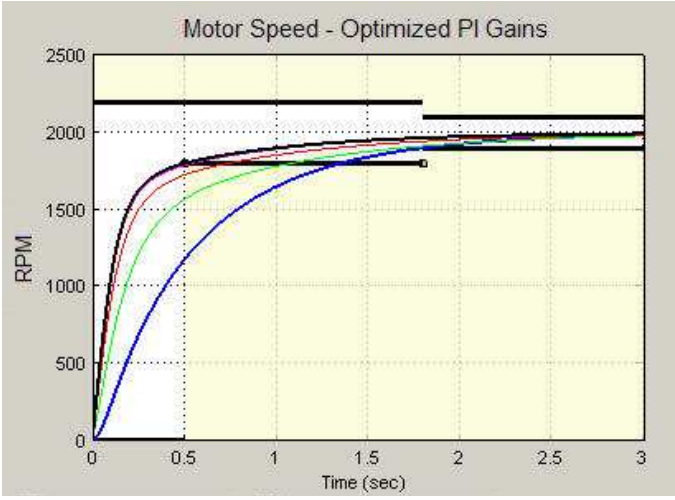

Figure 12 – Results from PI Gain Optimization

Alternatively, classic (linear) control techniques can also be applied by linearizing the physical model and using Simulink Control Design[15] to tune gains with a linear system model. The advantage of this approach is the ability to represent the physical system as an s-domain or z-domain transfer function enabling quick tuning of the gains. For example, linearizing the physical system in Figure 10 yields the transfer function in equation 6:

$$T(s) = \frac{a_6 s^6 + a_5 s^5 + a_4 s^4 + a_3 s^3 + a_2 s^2 + a_1 s + a_0}{b_7 s^7 + b_6 s^6 + b_5 s^5 + b_4 s^4 + b_3 s^3 + b_2 s^2 + b_1 s + b_0} \quad (6)$$

Where:

$a_6 = 0.0001168$ $\quad b_7 = 1$

$a_5 = 7.299e7$ $\quad b_6 = 1.532e4$

$a_4 = 2.321e10$ $\quad b_5 = 4.128e7$

$a_3 = 2.376e12$ $\quad b_4 = 1.657e10$

$a_2 = 7.685e13$ $b_3 = 2.202e12$
$a_1 = -7.316$ $b_2 = 1.021e14$
$a_0 = 0.005892$ $b_1 = 1.138e15$
$b_0 = 8.645e11$

The frequency response in the form of a bode plot is shown in Figure 13. Complete control system design and analysis is beyond the scope of this paper. The intent here is to show that all the advanced capabilities of Simulink are still available when using the network approach to represent the physical system.

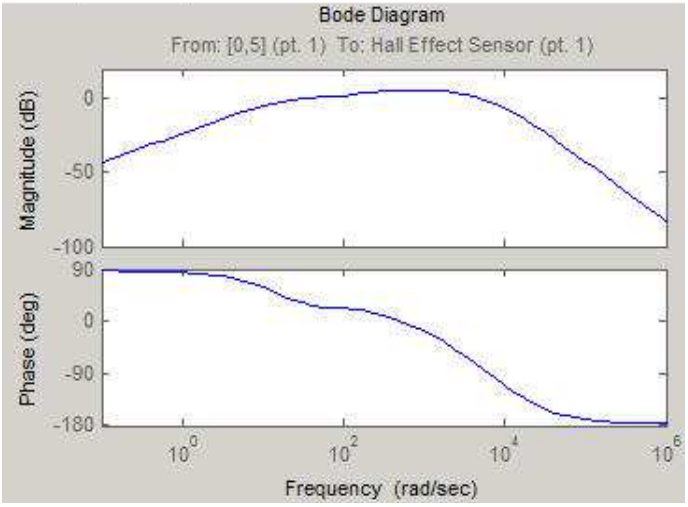

Figure 13 - Frequency Response of Linearized Model

REAL-TIME SIMULATION

Once the controller has been designed and tested in simulation, the next step in the Model-Based Design process is to deploy the control algorithm to the microprocessor for real-time testing and verification. Modern production code generation technology is typically used to create an on-target rapid prototyping environment where the algorithm code can be quickly deployed to the target microprocessor which is inserted into a physical prototype of the overall system. When a prototype of the physical system is not available, an alternative is to use Hardware-on-the-Loop (HIL) testing. For our purposes, HIL is defined as a microprocessor (the hardware) communicating (in-the-Loop) with a plant model that is running in real time (see Figure 14). This configuration allows you to exhaustively test and debug the controller prior to building a prototype of the physical system. The cost savings from reducing prototypes can be tremendous and there are many HIL systems available in the marketplace. An exhaustive discussion on HIL simulation in beyond the scope of his paper, rather the intent is to illustrate that it is not limited to the traditional causal modeling techniques typically employed by control system engineers.

In order to perform HIL testing, the physical model must be real-time capable. This presents a challenge when modeling physical systems as there is always a tradeoff between simulation speed and model complexity. The engineer must ensure that the model contains enough of the physics to provide an accurate representation for the controller, while avoiding overly complex models that slow down the simulation with no added benefit. The following discussion on model reduction explores this in more detail.

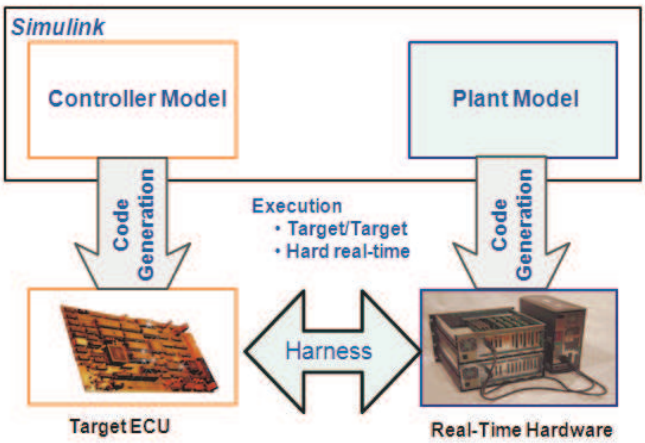

Figure 14 - Hardware-in-the-Loop Setup

MODEL REDUCTION - The motor control system in Figure 10 utilizes pulsewidth modulation (PWM) to control the switches in an H-Bridge connected to the DC motor. The resulting motor current is shown in Figure 15. Varying the duty cycle of the PWM signal is a common control technique, but the high switching frequency (10kHz) is computationally intensive for any numerical simulation engine and certainly not conducive to real-time simulation on any reasonably fast HIL hardware. What is needed is a "reduced" model that outputs the average value of the switching waveforms used to energize the motor. For our system we utilized the PWM and H-Bridge blocks from the SimElectronics library that allow you to quickly interchange the switching and averaged models. The removal of the switching waveforms increases the simulation speed by many orders of magnitude while maintaining the overall system performance. The Simscape modeling language discussed earlier could also be leveraged to create averaged models for real-time execution.

FIXED-STEP SOLVERS - In order to execute in real time, the model needs to be simulated with a fixed-step solver. When simulating a physical system, variable-step solvers are generally used due to the increased numerical efficiency required to capture the wide dynamic range typically present. However, since the microprocessor that will be connected to the real-time plant model executes in fixed time steps, the plant model must also execute in fixed steps. For fixed-step simulation, the size of the time step chosen will have a direct impact on the simulation speed and accuracy. A

smaller time step will provide greater accuracy, but will be numerically more expensive for the simulator. Selecting too large of a time step will speed up the simulation, but sacrifice accuracy. For this system we chose a time step of 100ms (as a comparison, for the PWM version of the model with a 10kHz switching frequency, the time step would have needed to be at least 100us!)

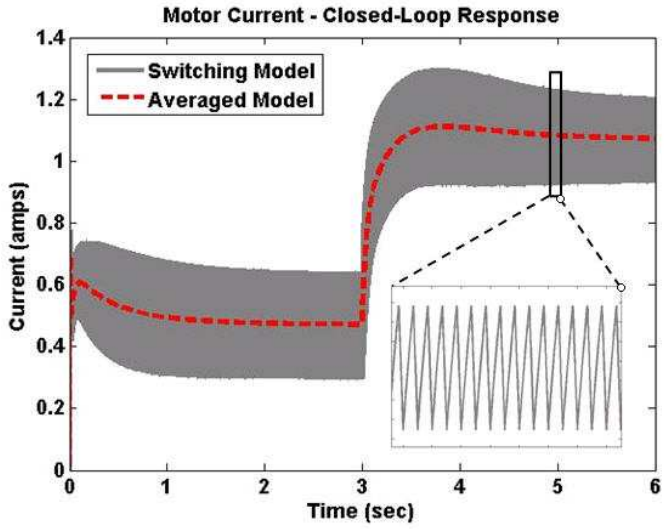

Figure 15 - Closed-Loop Response (switching and averaged models)

The motor current when using the fixed-step simulation with the averaged models is shown in Figure 15. As you can see, the averaged model provides a sufficient approximation of the motor current. The resulting motor speed is identical to the previous results shown in Figure 11.

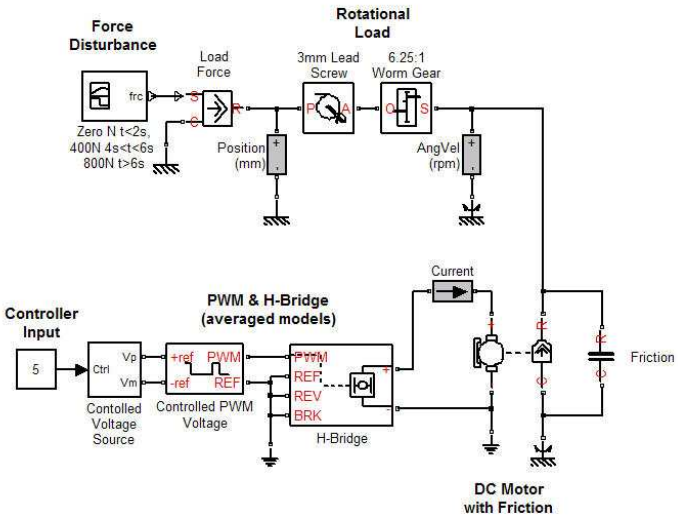

Figure 16 - Open-loop DC Motor Model

DEPLOYING TO REAL-TIME - Once the system model has been sufficiently reduced and simulated with a fixed-time solver, it is then ready to be deployed to real-time. The first step is to separate the controller and plant models so that code for each can be generated and deployed independently. Our focus here will be to deploy the plant model to real-time. Figure 16 shows the plant model without the controller. Here we are testing it in an open-loop configuration with a control signal input and an external force disturbance.

To deploy this model to real-time, we used xPC Target™[16] from the MathWorks. xPC Target leverages the code generation technology from Real-Time Workshop[17] to deploy the Simulink model to a real-time operating system capable of running on standard PC hardware.

With xPC Target you can use any PC with Intel or AMD 32-bit processors as your real-time target. The target PC can be a desktop computer, an industrial computer, PC/104, PC/104+, CompactPCI, all-in-one embedded PC, or any other PC-compatible form factor. The target PC for this system was a Pentium 4 with 768K RAM.

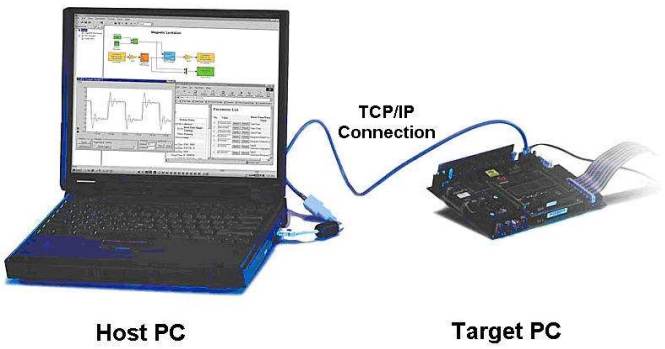

Figure 17 - xPC Target Setup

As shown in Figure 17, a single communications link connects the host and target computers. You design your application on the host computer in Simulink and download the real-time application to the target PC. The same communications interface is also used to pass commands and parameter changes to the target PC. You can choose either RS-232 or TCP/IP communications.

With a host computer running MATLAB, Simulink, Real-Time Workshop, xPC Target, and a C compiler as your development environment, you can create real-time applications and run them on a target PC using the xPC Target real-time kernel.

Once deployed to the target PC, the model can then be simulated in real time. The results of this simulation are shown in Figure 18. A control signal equivalent to 5V is supplied to the input of the averaged PWM and H-Bridge model. Initially, there is no load and the motor spins at

about 1500 RPM with a motor current of about 0.5 Amps. At 2sec, an opposing force of -400N is applied resulting in a decrease in speed to about 1100 RPM with an increase in current to about 1 amp. At 6 sec the opposing force is increased to -800N resulting in a further decrease in speed to about 750RPM an increase in current to about 1.5 amps.

The xPC Target scope also indicates an important metric called Average Task Execution Time (TET) which for our system is just under 2ms. This value is an average of the measured CPU times to run the model equations and post outputs during each sample interval. The average TET is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution. The average TET is the main factor to consider when determining the minimum achievable sample time. It the TET for a given interval exceeds the sample time, the result will be an overflow error. If this occurs you need to either further reduce your model or increase the processing power of the target PC.

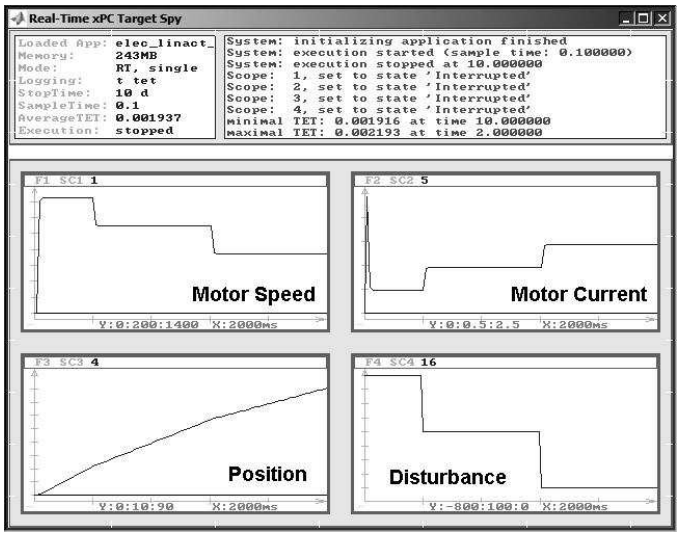

Figure 18 - Real-Time Simulation Results

CONCLUSION

Hardware-in-the-Loop simulation is a powerful tool for testing embedded control systems. In order to test the controller using HIL, a real-time plant model must be created. Signal flow (causal) modeling techniques have been traditionally used by control engineers to model the plant and deploy to real-time. The domain expertise required to derive the mechatronic system equations typically results prolonged development time or over-simplified models. The network approach to physical modeling uses non-causal connections between the physical component models. The benefit to this approach is that the system equations are automatically derived when the interconnecting components into a complete system. The existence of code generation technology for these non-causal models means that the time and effort required to create multi-domain real-time capable physical models is significantly reduced. Care must be taken to ensure that these network models have the appropriate amount of fidelity to provide a realistic dynamics for the controller. This typically means some amount of model reduction must occur to remove unnecessary detail from the physical model. The benefit of using a physical modeling language is that the user has direct control over this fidelity. Once the plant models have been simulated in real time with acceptable simulation results, they can be deployed to hardware running a real time operating system connected to the target microprocessor for HIL testing.

ACKNOWLEDGMENTS

I would like to thank my colleagues at The MathWorks Steve Miller, Jeff Wendlandt, Nathan Brewton and Rick Hyde for their support with the Simscape models and language. Additionally, I'd like to thank Ethan Woodruff, Sam Mirsky and Terry Denery for their support with real-time simulation using xPC Target.

REFERENCES

1. http://en.wikipedia.org/wiki/Mechatronics
2. Paul F. Smith, Sameer M. Prabhu, Jonathan H. Friedman, "Best Practices for Establishing a Model-Based Design Culture," SAE Paper 2007-01-0777.
3. Jeff Thate, Larry Kendrick, and Siva Nadarajah, "Caterpillar Automatic Code Generation," SAE Paper 2004-01-0894.
4. http://en.wikipedia.org/wiki/Hardware-in-the-loop
5. Feedback Control of Dynamic Systems, Gene F. Franklin, J. David Powell, Abbas Emami-Naeini, Prentice Hall, ISBN 0-13-032393
6. http://en.wikipedia.org/wiki/Bond_graph
7. http://www.mathworks.com/products/simulink
8. Mechatronics: An Integrated Approach, Clarence W. De Silva, CRC Press, 2005 ISBN 0849312744
9. http://en.wikipedia.org/wiki/Acausal
10. http://www.mathworks.com/products/simscape
11. http://www.mathworks.com/products/matlab
12. http://www.mathworks.com/products/simparameter
13. http://www.mathworks.com/products/simelectronics
14. http://www.mathworks.com/products/simresponse
15. http://www.mathworks.com/products/simcontrol
16. http://www.mathworks.com/products/xpctarget
17. http://www.mathworks.com/products/rtw

CONTACT

Tom Egel
Principal Application Engineer
The MathWorks Inc.
tom.egel@mathworks.com