Model-Based Design of a Power Window System: Modeling, Simulation, and Validation

Sameer M. Prabhu and Pieter J. Mosterman
The MathWorks, Inc.
3 Apple Hill Dr.
Natick, MA 01760

Abstract

The need to bring innovative, high-quality products to market faster is driving the use of models during the design and realization process. Model-based design provides efficiencies in product development that enable companies to deliver products on time, remain within budget, and fulfill initial requirements. The latest model-based design tools can also generate prototype and production code from a model automatically, significantly decreasing development time. This paper applies the model-based design process to the design of a power window control system and considers various aspects of the validation process via testing both during simulation and physical realization.

1. Introduction

Given competitive temporal and cost constraints, developing a product on time and within budget requires a systematic approach to design and realization. A systematic approach ensures that the final product meets the initial requirements and lets engineering teams with different specializations work together and communicate between stages in the overall process. In addition, this approach also ensures that the design process and the final product are documented for maintenance and future development.

The systematic design and realization process in the aerospace and automotive industries is typically represented by a V diagram as shown in Figure 1 (e.g., see [3,4]). Each of the two branches of the V corresponds to distinctly different activities:

1. The left branch captures the decomposition of the initial system requirements into subsystems and components that are specified and implemented at a detailed level.
2. The right branch represents the realization of these subsystems and components and their integration.

In the traditional approach, engineering teams observe strict boundaries between their design activities and they communicate by passing design documents back and forth. This approach has the following drawbacks:

1. Documents can be unwieldy and unsuitable for recording functionality.
2. It is difficult to keep the documentation synchronized with the current state of the design.
3. Once the design is approved, coding the application becomes a separate, manual activity.
4. When documents are used as deliverables and shared electronically, engineers often duplicate efforts. It is difficult to trace the source of errors along a paper trail.

Engineering teams have turned to model-based design and realization to address these problems. The model-based approach lets them address increasing product complexity, more stringent performance requirements, and shorter product development cycles. By using models in the early design stages, engineers can create what are known as "executable specifications" that enable them to immediately validate and verify specifications against the requirements. Validation ensures that the requirements are correct and that they represent the intended behavior. Verification ensures that the outputs of each step satisfy the step's inputs (i.e., the system satisfies its requirements). Less formally, verification checks whether the model is built correctly and validation checks

whether the correct model is built. This model-based design approach allows engineers to detect errors earlier when the cost to fix them is less.

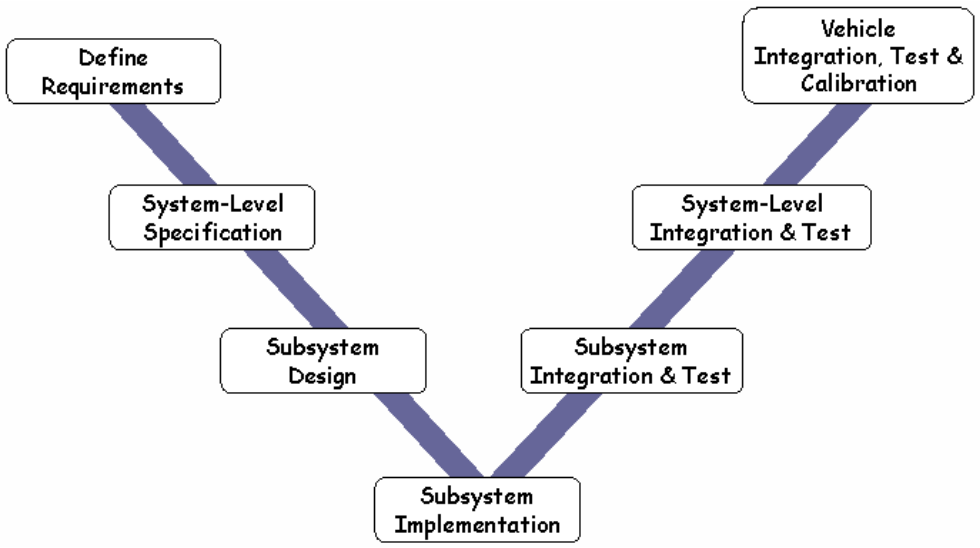

Figure 1: A V Diagram of the System Design and Realization Process

Further down the design process, models can be used to communicate between engineering teams with different specializations, allowing them to work together and to communicate between stages in the overall process. Moreover, initial design models can be incrementally extended to include increasing implementation detail. Thus, model-based design allows experimenting with different design alternatives, even in very early conceptual design stages, while having an executable specification and taking detailed implementation effects into account. This is in contrast to a document-centered approach where each of the design stages generates new models of the same system under design from the specification of the previous design stage.

Even more sophisticated is the use of model transformation to generate different representations of the same system, which further minimizes the effort to move from one design stage to another. In particular, the use of automatic code generation technology and hardware-in-the-loop testing alleviates errors introduced during manual implementation and realization tasks and shortens the path to product delivery by generating code for testing, calibration, and the final production. An important benefit of this model-based design paradigm is the traceability of design decisions all the way down to the implementation. So test results can be directly interpreted as high-level design decisions. Finally, even though electronic models are easier to navigate than paper documents, the formal system design process still requires detailed documentation. Advanced tools allow automatic generation of this documentation from a model while model-based design forces the design process as well as the final product to be documented for maintenance and future developments.

This paper applies the model-based design process to the design of a power window control system (shown in Figure 2) as typically found in modern automobiles, and the verification and validation of the developed models through real-time implementation. This verification and validation process covers both testing during simulation and testing on the real system to tune the model so that it approximates the behavior of the real system well. The MATLAB®-Simulink® environment [6,7] is used throughout the design process since it provides high-level formalisms such as SimMechanics [9] and SimPowerSystems [10] to support detailed modeling of the window system, the plant. Similarly, high-level formalisms such as Stateflow® [8] allow intuitive and elegant modeling of intricate control behavior such as fixed-point effects. This unprecedented level of detail brings the design process much closer to the realization before committing to an implementation, uncovers incompatibilities (e.g., different system of units for quantities) while the system is still in its electronic form and can be modified easily.[1] Further, experimenting with different design alternatives is possible, even in very early conceptual design stages, while

[1] The functionality is *what* a system does, the implementation models *how* the system does it, and the realization is the actual *physical* system.

having an executable specification and taking detailed implementation effects into account. Finally, automation shortens the path to product delivery by generating code for testing and calibration as well as the final production code.

Figure 2: A Typical Automobile Power Window

The paper is organized as follows: Section 2 discusses the behavioral modeling of a power window control system, and presents simulation results that demonstrate concept feasibility. The detailed software design aspects including model-based testing, requirements management, source control management, and documentation for the power window control system are covered in Section 3. Section 4 focuses on production code generation and embedded system integration. The results of hardware validation are presented in Section 5, and the conclusions are outlined in Section 6.

2. Behavioral Modeling of a Power Window Control System

A typical power window system is designed to meet various requirements. For the system under consideration the following requirements drive the design process:
1. The window has to start moving within 200 [ms] after the command is issued.
2. The window has to be fully opened and fully closed within 4 [s].
3. The force to detect when an object is present should be less than 100 [N].
4. If the up or down command is issued for at least 200 [ms] and at most 1 [s], the window has to be fully opened or closed, respectively.
5. When an object is present, the window should be lowered by approximately 10 [cm].
6. The driver command has priority over the passenger command.

Given these initial requirements, the discrete event core control algorithm can be modeled elegantly by exploiting the hierarchical state behavior of Stateflow (e.g., priority of driver commands over any of the passenger commands), as shown in Figure 3. The behavior of the discrete event control algorithm can be verified by submitting it to a variety of inputs that correspond to driver and passenger commands, in order to verify that the requirements are correctly captured in the Stateflow diagram. The animation capability of Stateflow provides visual feedback regarding the functioning of the control algorithm.

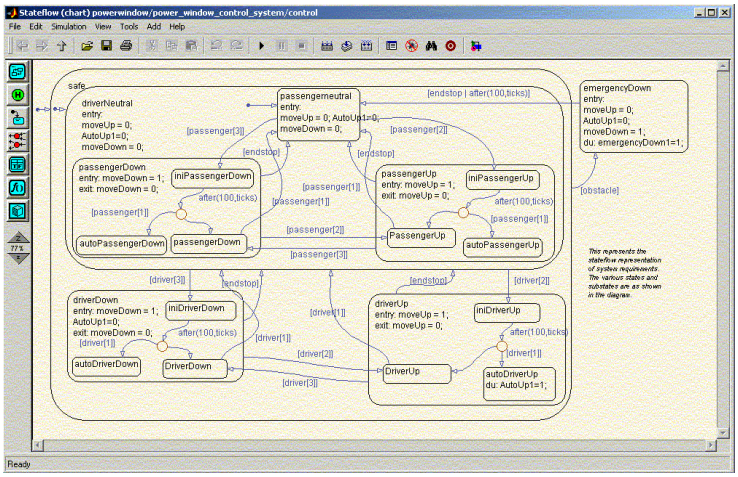

Figure 3: Discrete Event Control Algorithm Model

In order to study the continuous-time behavior (e.g., the 10 [cm] bounce-back in case an obstacle is detected), the Stateflow control algorithm can then be connected to a second order plant model in Simulink, as shown in Figure 4. This second order plant model allows calibration of the parameter that governs the downward movement of the power window to ensure that the 10 [cm] bounce-back requirement is met.

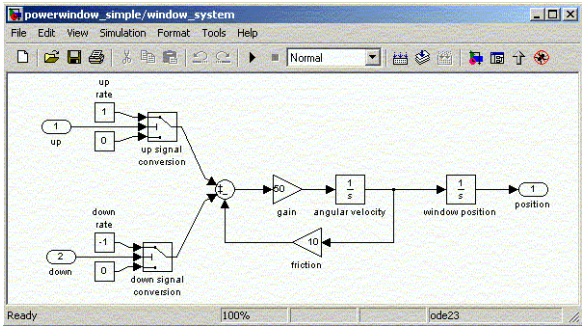

Figure 4: Simple Second Order Power Window Plant Model

Other requirements (e.g., the maximum force of 100 [N] on an obstacle) necessitate a more detailed plant model. This is facilitated by modeling formalisms that capture the physics in terms of energy flow. Tools such as SimPowerSystems for the electrical and SimMechanics for the mechanical part can model the energy exchange between physical components, so the designer does not have to perform the tedious analysis of the underlying signal flow in the physical model to implement it in Simulink.

For the power window, the direct-current (DC) motor and the electrical circuit driving the power window can be modeled using the SimPowerSystems blockset. The motion of the DC motor can be connected to a model of the scissors mechanism (which moves the window glass up and down), built using rigid bodies, joints, and other components from the SimMechanics blockset. Finally, the physical layout and geometry of the power window mechanism can be visualized by the Virtual Reality Toolbox [11] (e.g., to study the rotation direction of the worm gear). This integrates computer aided design (CAD) with the controller development as many CAD tools can export the virtual reality modeling language (VRML) format used by the Virtual Reality Toolbox. Figure 5 shows the various elements of the power window model.

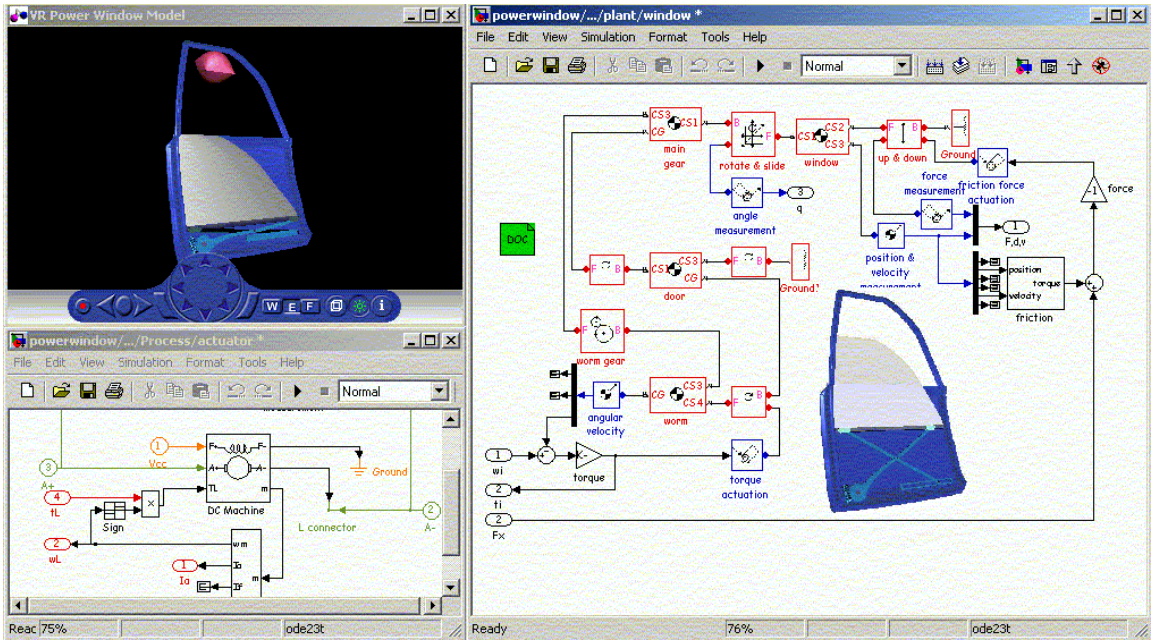

Figure 5: Power Window Electrical and Mechanical System Model and 3-D Visualization

Once the detailed model is constructed it can be used to run simulations of the discrete control algorithm interacting with the electro-mechanical plant model. These simulations can verify that the behavior of the control algorithm approximates the desired behavior as specified in the requirements. This involves subjecting the model to various test cases approximating the driver and passenger commands, and verifying that the system outputs (such as window position and force exerted on the obstacle) are within the limits outlined in the requirements. The control commands can be observed in the same environment to ensure that control response requirements are met as well. Figure 6 shows simulation results for a test case verifying that after 1 [s], the window is commanded to go up automatically and an obstacle is present.

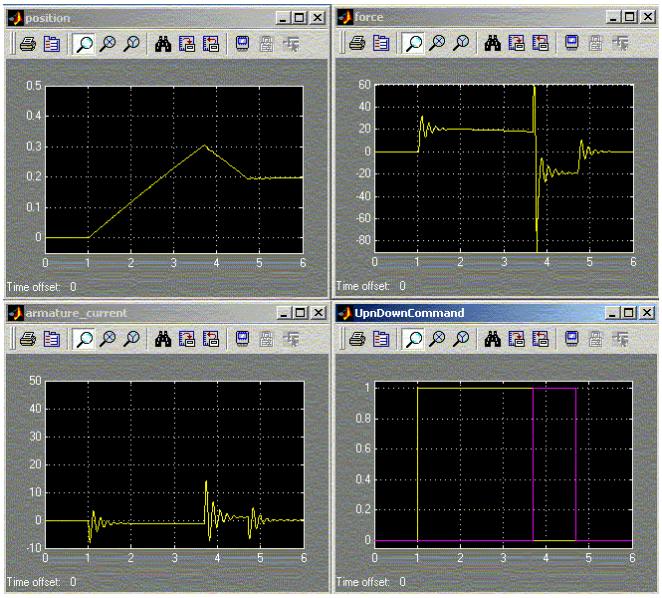

Figure 6: Simulation Results

During all of these design and implementation stages, the controller specification remains in an executable form and allows validation against the specific requirements under investigation. Modeling and simulation play a critical role in ensuring that the requirements are valid and in determining if any requirements conflict. Simulation is thus

a key validation step since it ensures that a system can be realized such that it satisfies the requirements. In addition, simulation can form the basis for multi-objective parameter synthesis methods for robust control design.

Since models of physical systems are approximate in nature, it is important to note that the validation steps carried out so far are limited by the fidelity of the plant model. One way to mitigate this risk is by rapidly prototyping the control system with a real physical system instead of the approximate plant model. This allows determining whether the control algorithm can ensure that the performance of the real system meets the requirements. Thus, it provides an estimate of the accuracy of the models and also validates the requirements.

A typical approach to rapid prototyping involves the use of a powerful, general-purpose computer with flexible input and output hardware as the controller. Real-Time Workshop [12] and xPC Target [14] provide this rapid prototyping capability by automatically generating the control model in the form of C code that runs on a real-time operating system. The real-time general-purpose computer is an xPC TargetBox that is capable of running the real-time application generated by xPC Target. This configuration is used to interactively test and calibrate controller parameters such as armature current thresholds for obstacle detection. Because automatic code generation allows a seamless transition between model and realization, even control structure changes can be quickly experimented with at this point.

3. Detailed Software Design for the Power Window Control System

As mentioned before, recent advances in code generation enable very efficient code to be synthesized directly from models that were used for control system specification, development, verification, and validation. Software engineers accomplish this by adding software-specific detail to the existing model. When automatically generated code is used without modification the models serve as the final implementation.

A variety of tools and techniques are used throughout this key step in the design process. Requirements capture and traceability tools are used to associate the requirements with the implementation, so that when the requirements change their effect on the design can be evaluated. Once software is generated, it is subjected to testing and code coverage tools are used to evaluate the completeness of the testing process. Code coverage analysis involves dynamically analyzing the way the code executes and then reporting on measurements such as statement coverage, decision coverage, condition coverage, and modified condition/decision coverage. Model-based coverage involves analyzing the model execution behavior and then reporting on the decision coverage, condition coverage, and modified condition/decision coverage metrics [2]. The basic goal of model-based coverage is to provide the equivalent information of code coverage in the context of the model under simulation. The entire process of creating test vectors, generating expected outputs based on requirements, and coverage analysis in the context of a model is referred to as model-based testing.

For the case of the power window control system, the various test cases were manually generated and incorporated into the model using the Signal Builder component of Simulink, as shown in Figure 7. This allows all the test cases to be defined and facilitates running the test cases either individually or as a test suite in order to obtain model coverage metrics.

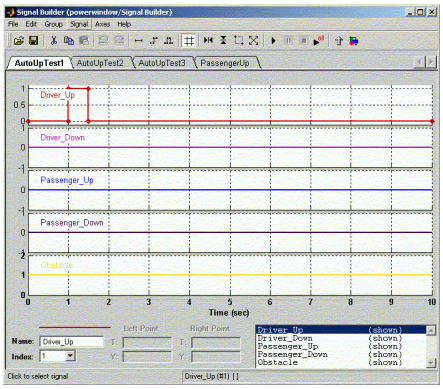

Figure 7: Test Vectors for Power Window Model

To associate the model with the system requirements, the Requirements Management Interface [16] is used and allows each requirement to be tied to the appropriate hierarchical level of the model where that requirement is realized.

Once the requirements have been associated with the model they can be used to derive the expected behavior of the model, which in turn can be used to create self-validating models by using the blocks from the Model Verification library in Simulink, as shown in Figure 8. When the results of the simulation do not meet the requirements, the blocks from the model verification library can be set up to stop the simulation and report on the requirement being violated. Combining the Signal Builder with the Model Verification blocks allows an automated approach to subjecting the model to various test conditions and ensuring that all requirements are being met.

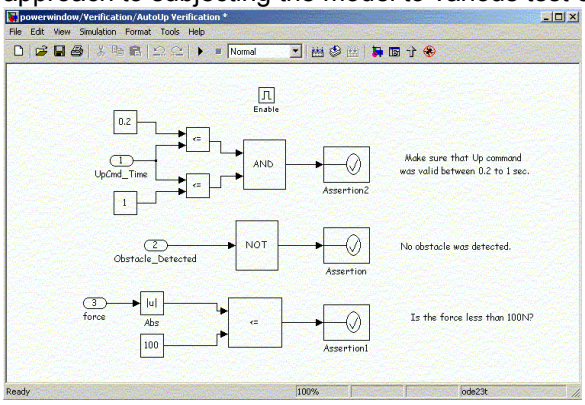

Figure 8: Power Window Verification Model

As mentioned before, it is necessary to assess the completeness of the testing, to ensure the model meets all the requirements in various operating modes. This is accomplished by using the model coverage tools in Simulink, which assess the cumulative results of a test suite to determine which blocks were not executed or which states were not reached. A coverage analysis report is generated after a simulation run as shown in Figure 9.

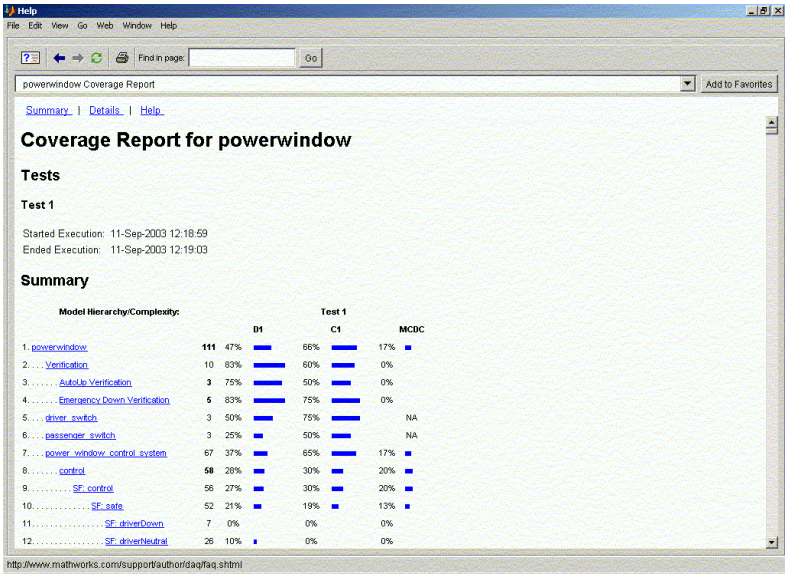

Figure 9: Coverage Analysis Report

Once the model-based testing process is complete, the model is ready for deployment. The model is often documented at this point in order to capture the various design decisions, and report on simulation results that demonstrate the verification and validation work. This can be accomplished in an automated fashion by using the MATLAB and Simulink Report Generator [17]. These tools together with the DocBlock component of Simulink are

used to create a self-contained report of the requirements, models, test cases, and simulation results, as shown in Figure 10.

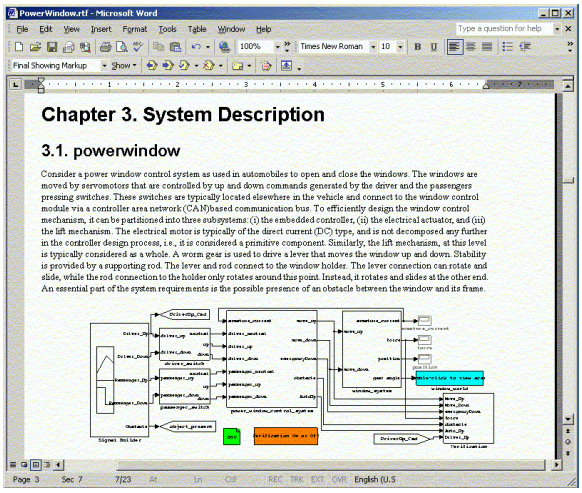

Figure 10: Model Documentation Report

Formal software development processes emphasize the use of Software Configuration Management (SCM) [3] for storing, versioning, and retrieving the various developmental stages of software, so that changes to the software are carefully controlled. Software engineers check out software, make changes, and then check in the software so their changes may be merged with others. The same process can be applied to the model-based design environment via the SCM interface available for Simulink, as shown below in Figure 11.

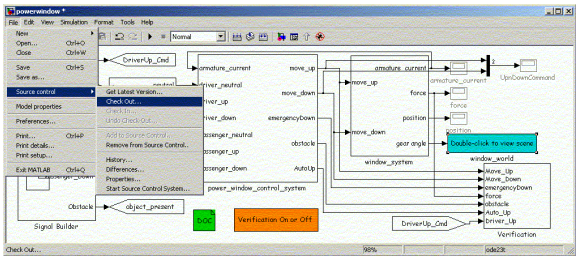

Figure 11: Model Source Configuration Management

4. Production Code Generation and Embedded Systems Integration

Once the model has been verified, validated, and documented, code can be generated from the model for implementation purposes. The production code is automatically generated by the Real-Time Workshop Embedded Coder [13] either in fixed-point or floating-point format. If fixed-point code is desired, the block diagram specification can be enhanced to include the fixed-point parameter settings. Thus, the same model can be refined further for software engineering purposes. Since the Motorola MPC555 processor was selected for implementing the power window control algorithm, the Embedded Target for Motorola MPC555 [18] is used to customize the generated code to run on an MPC555 processor, and target the I/O devices on the processor to achieve real-time control of the actual window. An important consideration in generating code from a model is traceability between code and model. This is accomplished by an HTML report created as part of the code generation process. It hyperlinks code back to the corresponding portion of the model, as shown in Figure 12.

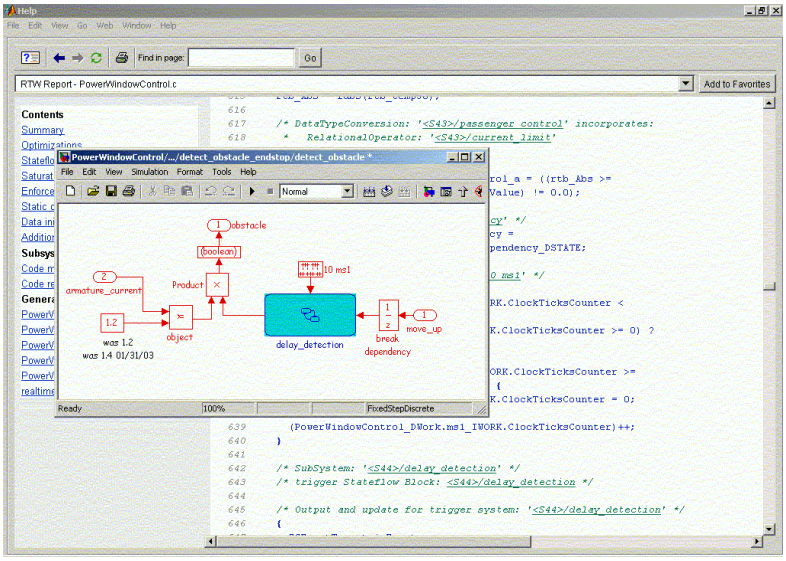

Figure 12: Traceability Between Generated Code and Model

When the generated code is specific to a particular processor and the associated compiler-debugger toolchain, the code generation report can be enhanced with measurements of ROM/RAM usage. These measurements can be used as a guide to further optimize the generated code, using various optimization settings and user configuration options available in Real-Time Workshop Embedded Coder and the Embedded Target for Motorola MPC555. Further, the task of interactively testing and calibrating controller parameters, such as armature current thresholds, using commercial controller area network (CAN) [5] Calibration Protocol (CCP) based tools is facilitated by an ASAP2 file created during the code generation process.

The generated code is implemented on a Phytec MPC555 evaluation board, and the power window system is controlled using the digital output and PWM ports on the MPC555. An H-Bridge is used to convert the digital and PWM outputs from the MPC555 to the high voltage, current, and direction reversal capabilities required for driving the power window DC motor. The switches and the current feedback for obstacle detection are read using the analog to digital converter blocks on the MPC555. The code for these blocks is automatically generated using the corresponding I/O blocks from the Embedded Target for Motorola MPC555 library.

5. Verification and Validation of the Power Window Control System

Once the design has been physically realized on the target embedded system, the Data Acquisition Toolbox [19] is used to measure window position and force, to verify that the physical system behavior meets the initial requirements. The Data Acquisition Toolbox can also be used in earlier stages of the design process to acquire data for calibrating the plant model at different levels of detail, as required by the design of the control algorithm. Figure 13 shows a plot of the position response of the power window in the presence of an obstacle, obtained from a position sensor added to the system for data acquisition purposes. It is clear that the power window bounces back approximately 10 [cm] in the presence of an obstacle, and therefore the design meets the initial bounce-back requirement.

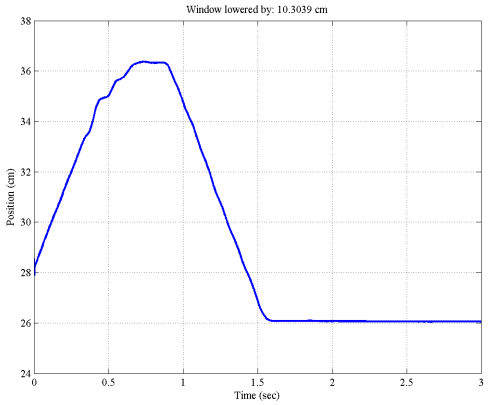

Figure 13: Position Response of Power Window

Figure 14 shows a plot of the force exerted on the obstacle, obtained via a load cell added to the system for data acquisition purposes. As shown, the force generated on the obstacle is well within the 100 [N] limit specified in the requirements. Satisfaction of the other requirements can be verified as well, so the developed model is fully realized in real-time, and meets the requirements for the system.

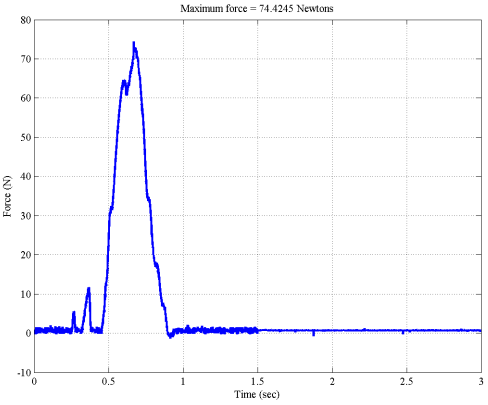

Figure 14: Force Response of Power Window

6. Conclusion

This paper illustrates how the entire embedded control systems development process from conceptualization to implementation can be realized using a model-based design approach in combination with an integrated tools suite. Following the model-based design approach results in designs that are consistent with requirements, and allows verification and validation throughout the development process. Errors can be detected early when the cost to correct them is less than at any later stage of the development process. Further, automatic generation of code avoids the errors associated with manual implementation approaches. The same model can be used and refined for a variety of tasks carried out in a typical development process. The net benefit of model-based design is improved efficiency of the development process, which translates to faster time-to-market, reduced development costs, and higher quality.

References

[1] www.mathworks.com
[2] Aldrich, B., “Using model coverage analysis to improve the controls development process,” AIAA 2002.
[3] Dahlqvist, A, U. Asklund, I. Crnkovic, A. Hedin, M. Larsson, J. Ranby and D. Svensson, “Product Data Management and Software Configuration Management—Similarities and Differences,” The Association of Swedish Engineering Industries, September, 2001

[3] Mosterman, P.J., J. Sztipanovits, and S. Engell, “Computer Automated Multi-Paradigm Modeling in Control Systems Technology,” IEEE Transactions on Control System Technology, 2003.
[4] Müller-Glaser, K.D., G. Frick, E. Sax, and M. Kühl, “Multi-Paradigm Modeling in Embedded Systems Design,” IEEE Transactions on Control System Technology, 2003.
[5] Robert Bosch GmbH, “CAN Specification,” Technical Report, Robert Bosch GmbH, Postfach 30 02 40, D-70442, Stuttgart, Germany, 1991.
[6] The MathWorks Inc., “Using MATLAB,” Version 6.5, The MathWorks Inc., Natick, MA, August, 2002.
[7] The MathWorks Inc., “Using Simulink,” Version 5.0.2, The MathWorks Inc., Natick, MA, April, 2003.
[8] The MathWorks Inc., “Stateflow and Stateflow Coder,” User's Guide, Version 5.0, The MathWorks Inc., Natick, MA, July, 2002.
[9] The MathWorks Inc., “SimMechanics User’s Guide," Version 2.0, The MathWorks Inc., Natick, MA, November, 2002.
[10] The MathWorks Inc., “SimPowerSystems User’s Guide," Version 2.3, The MathWorks Inc., Natick, MA, July, 2002.
[11] The MathWorks Inc., “Virtual Reality Toolbox User’s Guide," Version 3.1, The MathWorks Inc., Natick, MA, October, 2002.
[12] The MathWorks Inc., “Real-Time Workshop User's Guide," Version 5.0, The MathWorks Inc., Natick, MA, July, 2002.
[13] The MathWorks Inc., “Real-Time Workshop Embedded Coder User's Guide," Version 3.0, The MathWorks Inc., Natick, MA, July, 2002.
[14] The MathWorks Inc., “xPC Target User's Guide," Version 2, The MathWorks Inc., Natick, MA, July, 2002.
[16] The MathWorks Inc., “Requirements Management Interface User's Guide," Version 1.04, The MathWorks Inc., Natick, MA, July, 2002.
[17] The MathWorks Inc., “Report Generator User's Guide," Version 1.2, The MathWorks Inc., Natick, MA, July, 2002.
[18] The MathWorks Inc., “Embedded Target for Motorola MPC555 User’s Guide," Version 1.0.1, The MathWorks Inc., Natick, MA, July, 2002.
[19] The MathWorks Inc., “Data Acquisition Toolbox User’s Guide," Version 2.2, The MathWorks Inc., Natick, MA, July, 2002.

Biographies

Sameer M. Prabhu is a principal applications engineer with The MathWorks in their Detroit, Michigan office, where he primarily works with customers in the automotive area to understand and address their key technical needs. Prior to joining The MathWorks, Sameer engaged in the R&D of complex control systems through his work at Visteon, Caterpillar, and TELCO. The results of his work have been documented through publications and United States patents. Sameer graduated from the University of Bombay in 1991 with a B.S. in Mechanical Engineering. He also received his Ph.D. in Mechanical Engineering from Duke University in 1996 in the area of robotic controls and artificial intelligence. Sameer can be reached at:
The MathWorks, Inc.
39555 Orchard Hill Place, Suite 280, Novi, MI 48375, USA
Phone: 248.596.7944 email: sprabhu@mathworks.com

Pieter J. Mosterman is a senior research scientist in real-time and modeling and simulation technologies at The MathWorks, Inc. Previously, he held a research position at the German Aerospace Center (DLR) in Oberpfaffenhofen. He received his Ph.D. in Electrical and Computer Engineering from Vanderbilt University, and his M.Sc. degree in Electrical Engineering from the University of Twente. His primary research interests are in hybrid dynamic systems and Computer Automated Multi-Paradigm Modeling (CAMPaM) with principal applications in training systems and fault detection, isolation, and reconfiguration. He designed several simulation environments such as the *Electronics Laboratory Simulator* (nominated for the Computerworld Smithsonian Award) and HYBRSIM (a paper on which received the IMechE Donald Julius Groen Prize).

Dr. Mosterman co-chaired the *14th International Workshop on Principles of Diagnosis* and is Mechatronics area editor of *Simulation*, associate editor of *IEEE Transactions on Control Systems Technology* and the *International Journal of Applied Intelligence*, and was guest editor of special CAMPaM issues of *ACM Transactions on Modeling and Computer Simulation* and *IEEE Transactions on Control Systems Technology*.