

Automatic Embedded Code Generation from Simulation Models

Model-based design tools enable rapid model construction, execution and analysis of simulation results. Models developed with these tools have clean, well-defined interfaces that enable their use in multiple aspects of the system design process.

by Jim Ledin, Ledin Engineering and
Mike Dickens, The MathWorks

As a result of advances in automated code generation technology, another application of these models has become viable: they can serve as input to an automatic embedded code generation process. The code generated from these models is highly efficient, readable, testable and suitable for use in safety-critical applications.

Model-based design is a proven approach for efficiently developing solutions to complex engineering problems. It is a method for developing complex systems using mathematical models of system components and their interactions with the surrounding environment. These models have many applications in the design process, including system simulation, stability analysis, control algorithm design and as a specification for embedded controller software.

Several off-the-shelf software tools for model-based design support the specification, design and validation of high-fidelity plant models for a wide range of system design applications. The resulting models can be as detailed as necessary—assuming that sufficient information is available to support the construction and validation of the model. One important application for high-fidelity plant models is system simulation.

Given a high-fidelity plant model, the next step is to develop a model of the control system and integrate it into the system simulation. By working with a system-level simulation that com-

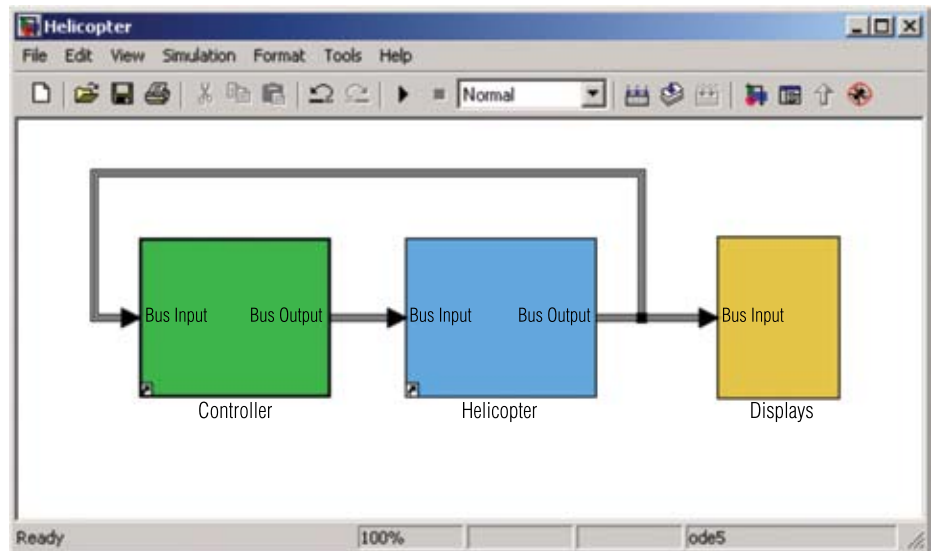


Figure 1 Top-level diagram of helicopter model.

bins the plant and controller, it is possible to thoroughly test the control system design and rapidly make changes and improvements as needed.

Automatic Code Generation

After the controller design has been optimized and validated in the simulation environment, the next challenge for embedded system designers is to develop the final software for production deployment. This task has traditionally required the recoding of

the controller from its simulation implementation to meet the very different requirements of production embedded software. This recoding step often results in new coding errors that must be identified and debugged.

Model-based design, together with advances in automatic code generation technology, makes it straightforward to use the embedded controller simulation model as the basis for final production code. This is possible because the model of the embedded software's behavior contains all the information needed to fully describe the embedded code.

However, the simulation model of the control system may require further adaptation to the embedded execution environment before it is fully usable as a source for the embedded code. Among the changes that may be necessary are conversion from continuous to discrete time, from floating to fixed-point mathematics and the addition of I/O devices

Continuous time is used for dynamic systems because values change continuously over time, such as the temperature of a room that is being heated or cooled. When we simulate a system like this, we compute values at specified time steps, which may be fixed or variable, depending on the desire of the engineer. But the simulation is usually very smooth and is done with continuous-time blocks.

When a controller is implemented, it must work in discrete time if it is a digital controller (an analog controller does not need to worry about time). A digital controller usually runs at a specified sample time, when it picks up values at fixed time steps, does a calculation and controls the system based on the result. This is done on fixed time steps, using a digital clock, and this is referred to as discrete time.

In the Simulink world, for example, a continuous-time sine wave block outputs a nice smooth curve, while the equivalent discrete-time sine wave block outputs a stepped curve, with little steps occurring at each time interval. These digital considerations also have implications for the way that data is represented in the system.

For example, in embedded software, signals are often represented as fixed-point integer values to conserve memory and increase execution speed. Simulation models, on the other hand, typically use double-precision floating-point representation. In converting a floating-point signal to fixed-point, it is necessary to select an integer word size and a scale factor to enable representation of

the signal value as a fixed-point integer.

In addition, the model of the control system must be modified to include I/O interfaces at each point where a signal passes

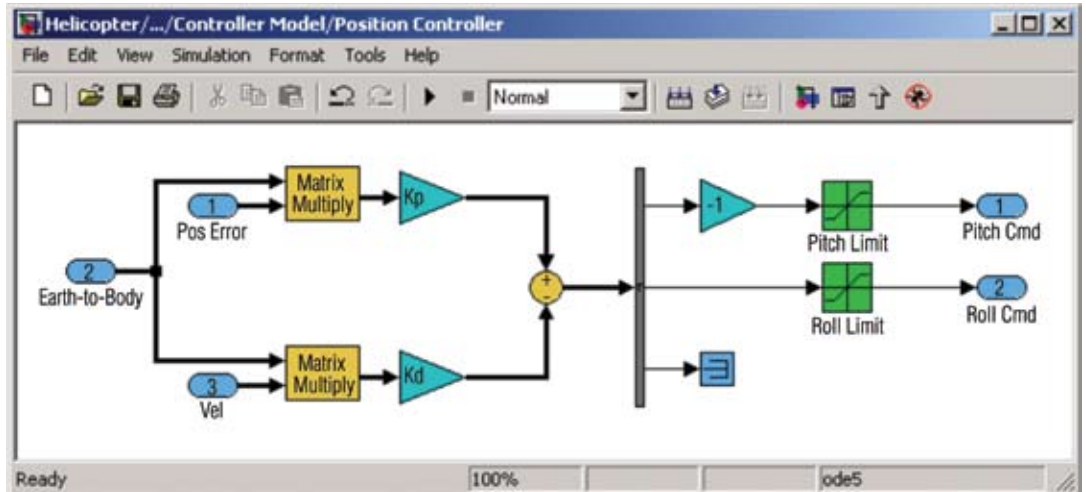


Figure 2 Helicopter position control subsystem.

between the controller software and an external device. These blocks represent the interaction of the controller algorithms with a device driver.

After completing some or all of the design modifications described above, the control system model is ready for use in the code generation process. You can automatically generate source code from the block diagrams for use on target processor hardware.

The correct way to change the generated code is to modify the model or change the code generation options rather than to drop below the model level and edit the source code. The goal of using Model-based design is to gain the benefits of the block diagram modeling environment in the development of embedded software. In many cases, the automatically generated C code can be treated as an intermediate language used in the compilation process and essentially ignored. For some safety- and mission-critical embedded applications, however, the developers must closely examine and test the generated C source.

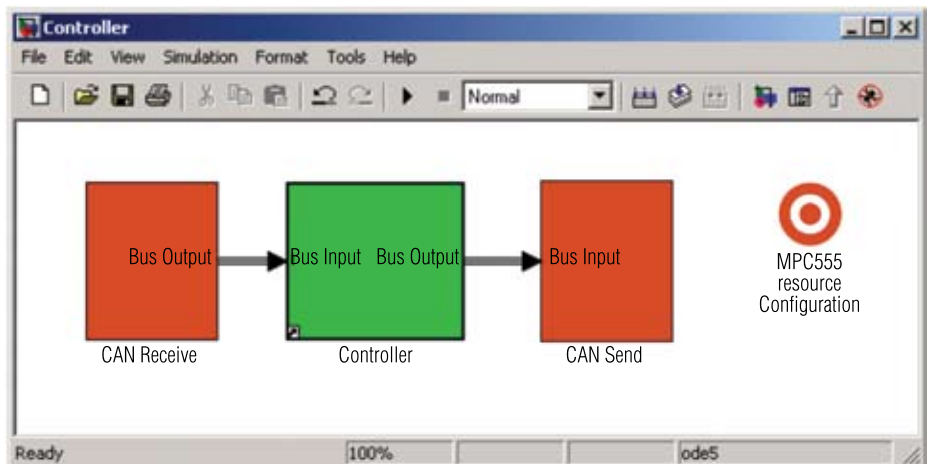


Figure 3 Model for embedded code generation.

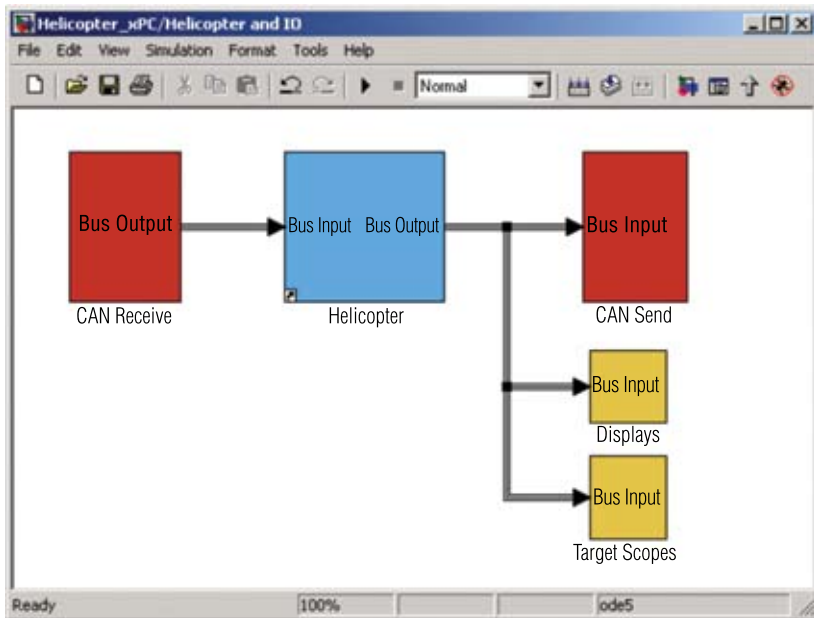


Figure 4 Plant model for HIL simulation.

Sample Application

Let's take a look at an application of model-based design. Figure 1 is a top-level diagram showing a model of a helicopter (the plant) and a control system that operates the helicopter. The plant model is a six-degree-of-freedom (6DOF) representation of the helicopter's dynamic behavior. The controller maintains the stability of the helicopter and flies it along a trajectory that takes it to a series of positions in space.

The three blocks in the diagram of Figure 1 are the Controller, the Helicopter plant model and the Displays. The latter presents information about the plant's behavior during simulation execution. The Controller and Helicopter blocks are library blocks, which means they are stored in a model library. The use of library blocks in model development allows system components to be shared among simulations and enables updates to the models to propagate immediately to all simulations using them.

Figure 2 is an example of one of the lower-level blocks within the Controller model. This is the position control subsystem, which guides the helicopter by developing tilt commands for the main rotor. It employs a proportional-derivative (PD) control algorithm, which uses relative position and the helicopter's velocity as inputs and produces pitch-and-roll rotor tilt commands as outputs.

The helicopter controller design uses a Controller Area Network (CAN) serial data bus for all communications between the embedded microprocessor and the sensors and actuators in the helicopter control system. The processor selected for this application is the Motorola MPC555, which is based on the PowerPC architecture.

The Controller block in Figure 1 represents the embedded software for this system. This controller model has been modified for discrete-time operation and will make use of the MPC555's floating-point processing hardware. The first step in the code generation process is to create a new model containing the Controller block and appropriate I/O blocks to interface with

each of its input and output signals.

Figure 3 shows the model of the controller, together with subsystems containing the CAN I/O interfaces. The remaining subsystem in this diagram, labeled MPC555 Resource Configuration, is required by the MPC555 embedded target for configuring system and I/O attributes such as IRQ levels and CAN interface bit rates.

Note that the Controller block in Figure 3 is identical to the Controller block in Figure 1. If a design problem is detected during testing, the controller can be changed in the diagram of Figure 3. The change will propagate to the simulation of Figure 1 for further analysis and testing.

The control system of Figure 3 is configured to operate in discrete time with a step time of 0.01 seconds. At each time step, the model reads input signals through the CAN Receive block, performs the computations contained in the Controller block, and transmits output signals via the CAN Send block.

The complete set of generated code for this embedded application consists of two .c files and four .h files containing a total of 1850 lines of code and comments. The code generation process also produces an HTML report summarizing the attributes (such as RAM and ROM usage) of the generated code.

Hardware-in-the-Loop Simulation

To test the automatically generated embedded code, it is necessary to either place the target processor into an actual embedded system or connect it to a simulated plant. Early in the system development process, a physical representation of the plant may be unavailable, or it may not be a good idea to test early versions of embedded software on an expensive one-of-a-kind system prototype.

An alternative to executing on the actual plant is to test the embedded controller on a simulated plant. The simulated plant executes in real time and must interface to the embedded processor using the interfaces over which the processor expects to com-

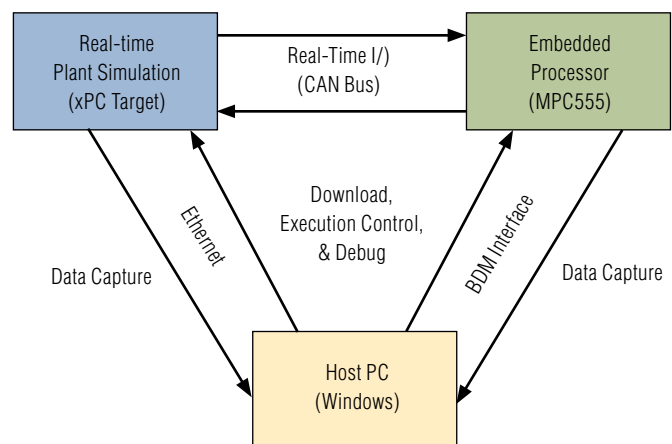


Figure 5 HIL simulation configuration.

municate. This is called hardware-in-the-loop (HIL) simulation, where the embedded processor hardware and the embedded software execute in real time in the context of a larger system simulation.

To create an HIL simulation for this application, it is necessary to copy the Helicopter block from Figure 1 and build a model around it. This model is similar to the one in Figure 3, except that it runs on a target processor that is different from the embedded processor. A target suitable for this application is xPC Target, which uses an ordinary PC as a real-time execution platform. xPC Target loads a small real-time kernel, which supports the execution of compiled Simulink models.

The Helicopter plant model for HIL simulation is shown in Figure 4. This model contains CAN interfaces that communicate with the embedded controller inputs and outputs shown in Figure 3. The block named Displays passes data to the host PC for display during execution. In addition, selected signals are displayed on the monitor of the target PC by the Target Scopes subsystem during execution.

The complete configuration for the HIL simulation with the embedded processor and the plant simulation is shown in Figure 5. The connection between the host PC and target PC is Ethernet. The link from the host PC to the embedded processor is a BDM debug interface.

The configuration and build process for the plant simulation is similar to the process used to build the embedded software. After the plant simulation has been built and downloaded to the target PC and that PC has been connected to the embedded controller via CAN cabling, real-time simulation can begin. Sufficient data must be captured on the host PC during the run to enable the developer to determine whether the controller is operating in accordance with requirements.

This sequence of events described above typically begins early in the product development cycle, dramatically reducing the risk that serious design problems will remain undetected until late in the cycle. By implementing and testing controller designs at an early stage in the project, overall project risk and cost can be significantly reduced. ■

Ledin Engineering
Camarillo, CA.
(805) 482-3109.
[www.ledin.com].

The MathWorks
Natick, MA.
(508) 647-7000.
[www.mathworks.com].