Our daily efforts to expand the confines of technology have made embedded-system programmers an overworked bunch. They must keep up with the enormous increases in complexity due to evolving technology and new customer demands while facing the shorter schedules necessary to compete in today’s product market.

The recent explosion in new-product development has also created a severe shortage of skilled software designers to spread across projects. In an effort to aid the beleaguered programmers and capture a larger market share, embedded-software vendors now offer off-the-shelf code-generation tools to automate portions of the development process. These tools promise to transform embedded-software development from a line-by-line programming task to a fully automatic process that generates optimized, ready-to-run source code from a graphical model of the system.

However, don’t expect to rapidly fill openings in your software staff with a bank of PCs constantly churning out embedded source code. The development tools available today require a significant learning effort and possibly a complete change of programming philosophy before designers are ready to craft a system model compatible with automatic code generation. And
a change in attitude among your current software staff may be necessary to overcome prejudice and accept a completely different embedded-development technique. Based on experience with early products, many developers feel that code generators require significant overhead and produce inefficient code. However, modern tool vendors claim that their code generators produce error-free code that exactly matches the specification and is more compact than handwritten code. Successful code generation begins at the specification stage. Instead of a written specification in textual format, you must develop a formal graphical representation, or model, that defines the intended functions and behavior of the proposed system. The model must include enough information to unambiguously describe the embedded process. In general, most system-modeling programs allow users to construct system representations by selecting prebuilt blocks from a library and connecting them together to form the graphical models. Becoming adept at system modeling requires the same devotion and training period as learning a new programming language.

Modeling programs are almost exclusively based on the object-oriented methodology, meaning that systems are built from reusable components called objects (Reference 1). Unlike traditional programming techniques that separate data and function, object-oriented software consists of a collection of objects that have both data structure and behavior. In other words, objects do things (behavior) and know things (data). Although this idea sounds simple, the process required to model a system is complicated and is best learned following the rules of the modeling software you select. Although any target language is possible, C, C++, and Java are the most popular languages for code generators, because they are or can be structured in an object-oriented format.

Although code generation may be your main objective, graphical modeling of a system can produce additional benefits that also reduce product-development costs. For example, most modeling systems have built-in error-checking features to automatically check for inconsistencies and ambiguities. This type of error checking is not available with a standard written specification and potentially eliminates mistakes early in the development cycle.

DESKTOP SIMULATION

Another benefit of a graphical model is the ability to simulate the operation of the embedded system before committing the design to hardware. You have the opportunity to test the performance and optimize the design on a standard PC to verify that the product meets system requirements. Of course, a simulation will not duplicate the deterministic performance of a real-time system, but it lets you easily test many functions, such as the user interface. Simulation of a graphical model results in an executable specification that customers and end users can evaluate to ensure that everyone has the same product expectations (Reference 2). A graphical model is also a good vehicle for communication among team members, other departments, and outside contractors to support concurrent engineering tasks.

When you generate embedded-system code from a graphical model, you have ready-to-run software as soon as the prototype hardware is available. You can also use the model to create test scenarios to assist in hardware verification or system validation. Some code-generation vendors include features to also automatically generate test sequences to verify production hardware. This technique is especially effective in troubleshooting and debugging the hardware, because it exercises all functional paths. Often, customers need a full test suite to verify compliance with the original specification—a requirement that a modeled system easily meets.

One of the most powerful benefits of automatic code generation is back-annotation or reverse-engineering. If you change object code in the lab, you can automatically synchronize and update the model to incorporate the change. This feature connects the design and coding activities and guarantees
that the model always reflects the latest product status. Reverse-engineering also implies that you can create a model directly from embedded source code; however, you can expect varying degrees of success, depending on how the code was originally created.

The bane of most programmers—documentation—is a snap with a graphical-model-based development project. Most modeling programs allow users to extract all system data, in both graphical and textual formats, to create standard or custom documentation. Like reverse-engineering, automatic documentation ensures that the paperwork tracks the hardware. Changes to the documentation immediately follow product changes and software updates when the process is automated.

Models for many automatic-code-generation products are prepared using UML (Unified Modeling Language). Originally adopted in 1997, UML is currently the most popular language for object-oriented modeling. Historically, UML is a consolidation of as many as 50 modeling approaches advanced in the early 1990s. The OMG (Object Management Group) maintains the language, and you can download the latest version of the specification without charge from its Web site, www.omg.org. Version 1.4 is the latest UML working specification; however, the Version 2.0 proposal is available for public review and comment at www.u2-partners.org/artifacts.htm.

**WHAT IS UML?**

UML is a standard, nonproprietary graphical technique for capturing the analysis and design of object-oriented software. The language includes a set of well-defined graphical symbols so that other designers can exchange and understand models. Like any object-oriented method, UML is based on classes and objects. A class is a set of objects that have the same data and behavior. Class attributes define the encapsulated data, and class services describe the functions that act on attributes. UML defines multiple diagram types to document the software system from various points of view. Some diagrams are high-level; others delve into the details about a small portion of the software. UML diagrams include:

- class diagrams, which describe the

**AT A GLANCE**

- Automatic code-generation software produces optimized, error-free source code from a graphical model of the system.
- Wide acceptance of UML promises to standardize software architectural descriptions and boost code-generation usage.
- Round-trip and reverse-engineering features ensure that the system model automatically reflects code changes made in the lab.
- In addition to producing ready-to-run source code, modeling packages also produce up-to-date and near-instantaneous system documentation.
- Built-in simulators test system performance from models before committing the design to hardware or generating target code.

**GRAPHICS-CODE GENERATOR DELIVERS VIRTUAL PROTOTYPES**

Some of the most challenging programming problems in today’s embedded designs arise when you’re dealing with complex graphical displays with animation and moving images. Altia Inc has tackled this difficult area with DeepScreen 2.0, a graphics-code generator for embedded, real-time displays. Altia claims that you can substitute weeks of hand coding with a few hours’ work with DeepScreen to create graphical prototypes, allowing you to focus on the application code (Figure A).

With DeepScreen, embedded-system developers can generate code for a variety of target RTOSs and processors. Because it generates C code, it is easy to modify the code to work on any target system, including “homegrown,” proprietary RTOSs. To further speed development, Altia provides specific code generators for popular processors that require no modifications. DeepScreen directly supports Linux, Windows CE, and QNX for x86, StrongARM, Hitachi, and MIPS. DeepScreen generates code for embedded-Linux applications using the Microwindows, X, or nanoX graphics libraries.

Developers use DeepScreen in conjunction with Altia Design 4.5, Altia’s graphics-design environment. With it, they can create interactive, animated graphics without programming. It allows them to develop prototype interface components, such as knobs and buttons, as well as the screen graphics, which they can later deploy into the final product. After creating their display graphics in Altia Design, embedded-system programmers simply run the graphics through DeepScreen to automatically generate complete code for their target systems.

Altia DeepScreen starts at $25,000, and Altia Design starts at $10,000 for single node-locked licenses.
static structure of a model showing internal data, services, and their associations to other objects;

- object diagrams, which show objects and their relationship at an instant of time during the execution of a system;
- use-case diagrams, which document high-level analysis with usage models to identify system behavioral requirements;
- sequence diagrams, which describe real-time specifications or complex scenarios by showing the time sequence of events among objects during execution;
- collaboration diagrams, which are similar to sequence diagrams and describe the behavior of a set of objects that exchange messages to accomplish a purpose;
- state-chart diagrams, which show the sequence of states that an object goes through during its life cycle in response to external stimuli;
- activity diagrams, which are a special case of state-chart diagrams used to document work flow and model business processes; and
- deployment diagrams, which model the topology of the hardware executing the software system.

Rational Rose from Rational Software Corp is one of the earliest and most widely used commercial UML-modeling tools. Versions are available for development of standard business and real-time embedded-software systems. Rational Rose, which Solaris, Windows, or Unix platforms can host, is available as a stand-alone product or as part of various Rational development suites (Figure 1).

Code-generator targets include ANSI C and C++, Visual C++, Java, Visual Basic, and Ada. Source code is also optimized for embedded real-time operating systems, including VxWorks, ChorusOS, OSE, QNX, Linux, and others. Prices for Rational’s latest development suites with upgrades for remote collaboration and project tracking range from $3995 for AnalystStudio to $7695 for the Enterprise version.

Specializing in real-time embedded software, the Rhapsody Visual Programming Environment from I-Logix integrates development into one associative process, in which design, code, and documentation are continuously synchronized. You can generate C, C++, or Java directly from a UML model, merge your design with legacy code, or add your own custom code. You can extract software components from a model and use them in new designs without support from the original developer. Figure 2 shows a representative user interface for the Rhapsody in C programming environment. The recently introduced Version 4.0 of Rhapsody features a new user interface, increased scalability, and support for all UML diagrams. I-Logix offers unique software versions depending on the type of code produced and the target operating system. For example, the single-user version of Rhapsody for C++ sells for $995, and the operating-system add-on for VxWorks costs $295 more.

**FREE MODELS**

Several free alternatives exist for those interested in experimenting with UML without the expense of an off-the-shelf commercial program. Both Rational and I-Logix offer free, time-limited trial versions of their modeling software for buy-evaluation. Another possibility is ArgoUML (http://argouml.tigris.org). The ArgoUML UML modeling tool is not only free, but also an open-source development project to which you are invited to contribute (Figure 3). ArgoUML is coded entirely in Java and uses the Java
foundation classes, allowing it to run on virtually any platform. It supports class, state-machine, activity, use-case, collaboration, and combined-object/component/deployment diagrams. You can easily launch ArgoUML with Sun’s Java Web Start (http://java.sun.com/products/javawebstart/), simply by clicking on a Web-page link in your browser. Although ArgoUML includes extensive UML capability, the code-generation features are currently only partially implemented. You can use ArgoUML for commercial-development projects, and, because it is open-source, you incur no license fees.

Automatic code-generation tools are also available for 8- and 16-bit systems, which you traditionally program in C. VisualState from IAR Systems includes a designer segment to translate your ideas into UML state-chart diagrams, an analysis tool to verify your machine model, a debugger to exercise your design through simulation and with the actual hardware, a table-based C code generator for your embedded target, and an automatic documentation generator. IAR Systems claims that the generated code is completely consistent with the design, executes deterministically, and is more compact than handwritten code.

If you are new to object-oriented design and programming, you can find valuable information and online forums about modeling and tools at the Objects by Design Web site, www.objectsbydesign.com. The site includes an online list of available UML-based development tools, currently listing more than 85 tools with pricing, a short description, the target language, and a pointer to the vendor’s Web site. The Objects by Design Web site also gives valuable tips on choosing a modeling tool, by comparing features such as repository support, round-trip engineering, documentation, specification compatibility, model exchange, version control, diagram customization, printing, robustness, and support.

Although many automatic-code-generation systems are based exclusively on UML, you can find popular and successful development tools built on proprietary graphical modeling systems (see sidebar “Graphics-code generator delivers virtual prototypes”). Prime examples are the widely used analysis, visualization, design, and development tools based on Matlab from The MathWorks. For example, Simulink is an interactive tool for modeling, simulating, and analyzing dynamic systems. It enables you to build graphical block diagrams, evaluate system performance, and refine your designs. Simulink integrates with The MathWorks’ Stateflow for modeling event-driven behavior and Real-Time Workshop Embedded Coder to automatically generate ANSI C code.

Figure 4 shows an example application combining several MathWorks tools. All tools require Matlab, which sells for $1990, and extension tools are priced separately at $2800 for Simulink or Stateflow and $5000 for the Real-Time Workshop Embedded Coder.

One of the latest catch phrases from tool vendors is “rapid prototyping,” implying that you can use a product to validate an approach in a matter of days as opposed to the several weeks required to design, assemble, and test a breadboard with the conventional hardware proof-of-concept technique. One bundle of products that actually deliver on this promise is the VisSim/TI C2000 Rapid Prototyper from Visual Solutions and the 2407 eZdsp evaluation board from Texas Instruments. VisSim is a Windows-based program for the modeling and simulation of dynamic systems. VisSim Personal Edition combines a drag-and-drop block-diagram interface with a simulation engine
Automatic code generation for linear, nonlinear, continuous-time, discrete-time, time-varying, and hybrid systems.

The Rapid Prototyper includes a digital-motor-control peripheral-block set from Texas Instruments that automatically generates code for embedded controllers targeting the C2000 family of DSPs. You can buy the software, evaluation board, power supply, and cables for $3295 at the Visual Solutions Web site.

Opportunities abound for labor savings with automatic code generation in embedded-system development. For example, many applications require custom object code for multiple target platforms. Or perhaps you are duplicating or recoding software from a previous-generation product. Maybe a change in operating systems will force major modifications to existing code. And don’t forget the headaches when hardware changes due to obsolete microprocessors or peripheral parts. Sometimes, a subtle bug or a simple customer change request forces major code restructuring. You can avoid many of these problems with a model-based software system featuring automatic code generation. Think ahead.

References

The combination of Stateflow, Simulink, and Matlab from The MathWorks creates an integrated simulation environment in which you can model, simulate, and analyze complex dynamic systems.