Wind River’s BetterState™ provides powerful graphical programming capabilities based on graphical constructs such as statecharts, state transition diagrams, and flowcharts. BetterState facilitates software development with diagrams that are easy to comprehend and maintain, and accelerates programming with rapid prototyping, automatic code generation, and faster design iterations.

BetterState is optimized for each phase of the embedded software development process. Using BetterState requires just three steps to convert software to production-ready software modules:

1. Specification of software modules using advanced graphical constructs
2. Implementation of software modules via automatic code generation in C, C++, or Java for stand-alone or RTOS-based applications, and integration within a given application framework
3. Graphical debugging of software modules that execute on a host or on a target

Software module specification
BetterState provides a broad range of features that enable users to efficiently specify the desired behavior of a software module.

Features
- Graphical user interface for mixed statechart and flowchart specification
- Diagrammatic constructs and user-selectable options for tailored code generation
- C/ C++ and Java optimized code generation
- RTOS support
- Simultaneous diagram and source level debug
- Round-trip engineering

Benefits
- Intuitive graphical programming tool
- Automatic modular and portable code generation
- One diagram can generate multiple source files
- Easier and faster than hand-writing code
- Integrated with other commercial tools

Supported hosts
- Windows 95/98/NT

For a free download of BetterState Lite, go to http://www.isi.com/products/betterstate/bsform.htm
Statechart constructs

State is the basic element of a statechart. Each state has the following characteristics:
- Default state – a means to specify the default activated state within a chart or thread.
- History state – a means to return to the last active state the next time a chart/thread becomes active.
- Terminal state – a means to suspend a chart’s execution until the chart is reactivated.
- Actions executed “On-Entry,” “During,” and/or “On-Exit.”
- Assertions executed if the corresponding pre- or postcondition fails.
- Layer specification defining the state layer membership.
- Transition – the conditional connection between two states.
  - Transitions are condition-based, event-based or both.
  - An action associated with each transition is executed when the transition occurs.

Hierarchical state – a graphical representation of a state within a state.

Concurrent/independent threads – a graphical way to represent two or more “independent” threads of execution within a given state.

Non-resting state – a state where the outgoing transition is immediately evaluated upon completion of the previous transition and its related actions. If the condition transition is true, the transition related actions are immediately executed.

Flowchart constructs

Designers can incorporate traditional flowchart constructs (cascaded decision polygons) on the same diagram as statechart constructs.

Advanced software design constructs

In addition to statechart and flowchart constructs, BetterState supports graphical specification of advanced software concepts including:
- Visual synchronization – a graphical means to “synchronize” threads of execution based on active states in one or multiple threads of execution.
- Visual priority – a means to resolve a “race” condition that occurs when two or more transition conditions are active at the same instant.
- Critical region – a graphical representation of an interthread exclusive region, wherein only one state can be active at a time.
- Sub-statechart – a means to reuse a chart within other charts.
- Local variables can be defined using the variable box.
- Documentation can be added directly on a diagram using the documentation box or within each of the software constructs.

Testing constructs

The graphical specification can be used for testing with:
- Assertions – actions that are executed if the respective pre- or postcondition fails.
- Layers – provide a means of dividing graphical design constructs into several groups and then generating code for any group or set of groups.

BetterState provides utilities to make the graphical development of software modules more user friendly. These utilities include an interactive point-and-click graphical editor, diagram browsers, support for multiple diagrams within a single project, find-and-replace, HTML report generator, and internationalization/localization in English, Kanji, and German.
**Software module automates code generation and integration**

BetterState’s sophisticated code generation algorithms automatically generate production-ready, optimized code. Users retain control over the code generation process through several user-selectable options. Code can be generated for an entire project, a single chart, or for a subset of a chart using the ability to generate code for selected layers.

BetterState supports procedural or event-driven code generation in C, C++, and Java. The generated code can also be targeted for real-time operating systems.

Generating code specific to an application can be accomplished via user-selectable options, although many options depend on the application language. Option examples include the selection of a code if-then-else statement or switch-case statement, priority selection, action ordering selection, header files to be included in the generated code, inheritance definitions in C++ and Java, and formal parameter interface definition. As a result of the user-selectable option, BetterState-generated code can be easily integrated with a higher-level application or a test harness.

**Software module debugging**

Once a software module has been implemented and integrated, BetterState can be used as a graphical debugging tool. Debugging features within BetterState include:

- Interactive state animation. While executing BetterState-generated code on a host or target platform, the diagram is automatically animated to reflect program execution, the active state, the previous state and the path from the previous state to the active state.
- Break point and a break point callback that can be set for each state directly through the BetterState diagram.

If a source code debugging approach is preferred, BetterState supports round-trip engineering, which permits designers to manually modify the generated code and automatically import the modification into the BetterState diagram.

**Open architecture**

BetterState’s open architecture facilitates its integration within a development process or integration with other development tools. Features of BetterState’s open architecture include a command-line API to open a diagram, generate code, or start a new project; an open file format using the extensible markup language (XML); pre- and postcode generation system calls; and import/export of independent charts between projects.