

## Device Driver Development Environments for Embedded Communication Applications

Very recently, a new kind of software development tool has appeared that is now becoming part of an embedded communication application developer's standard repertoire of tools. This new kind of tool is called a Device Driver Development Environment (DDDE). It accelerates and provides automated support for development of Board Support Packages and Device Drivers, using an approach based on reusable code.

### OVERVIEW

In the world of software development for embedded communication applications, as in other technical disciplines, there is great benefit in the use of design automation tools. Some of the special challenges of software design in the embedded communication field are caused by highly constrained software execution environments. In addition, extremely precise programming of software-hardware interfaces must be attained to get the embedded software to interact properly with a spectrum of specialized communication interfaces. An example is the Motorola PowerQUICC-II communications processor, whose on-chip CPM (Communications Processor Module) contains more than 15 sophisticated communication interfaces.

One of the traditionally most neglected areas of embedded communications software development has been the area of design tools for device drivers and board support packages. These are low-level software components that forge the actual connection between the microprocessor "engine" of a communications processor, and higher-level software such as

application tasks, communication protocol software and real-time operating systems. Many of these connections are highly unusual and highly critical to the system. The low-level software components that implement them are tedious to build, because of the complexity and intricacy of the software-hardware interfaces specified by semiconductor manufacturers.

Very recently, a new kind of software development tool has appeared that is now becoming part of an embedded communication application developer's standard repertoire of tools. This new kind of tool is called a Device Driver Development Environment (DDDE). It accelerates and provides automated support for development of Board Support Packages and Device Drivers, using an approach based on reusable code.

### APPLICATION DEVELOPMENT AND EXECUTION ENVIRONMENTS

Embedded communications application development environments consist of a number of interconnecting tools. Some common commercially available tools are shown in Figure 1.

Many commercial off-the-shelf (COTS) embedded software development environments contain the components labeled "Integrated Development Environment" in the diagram above. But often, actual software projects include additional tools, such as computer-aided software engineering packages, test management and version control tools.

Embedded communication systems development environments run on "host" computers that are typically high-end PCs or UNIX platforms. However, the code that's developed is to be run on a different "target" computer that will actually be at the heart of the actual embedded communication product under development. The embedded "target" computer has a layered structure as shown in Figure 2.

At the top level is the application software that expresses the communications application expertise of the development organization. Typically, the application software is structured as multiple concurrent tasks, so a real-time operating system (RTOS kernel) appears in the level below to support multitasking. The level below

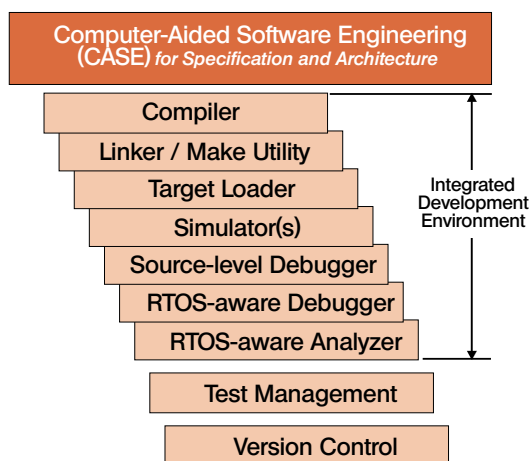


Figure 1. Common Commercially-Available Embedded Communications Application Software Development Environment Components.

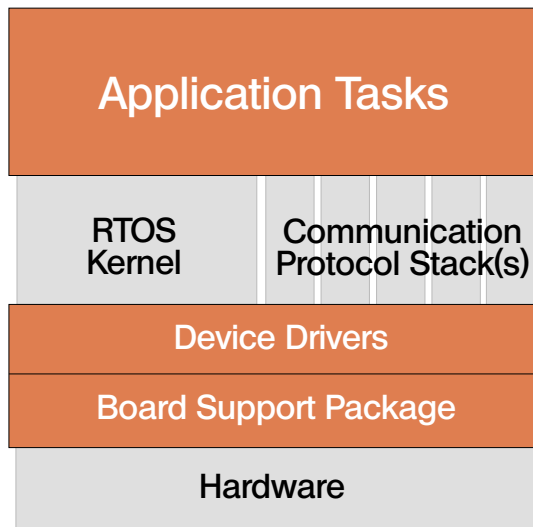


Figure 2. Executable Code on Embedded Communications Target Computers.

the application also contains communication protocol stacks. The next level down contains device drivers and a Board Support Package (BSP). The BSP is code that ties higher-level software to the basic underlying hardware of the embedded communicator to create an environment in which the higher-level software can execute. The device drivers contain hardware-specific code that allows higher-level software to operate the communication devices of the embedded system.

At the lowest level is the hardware of the embedded communications processor and its on-chip peripheral communication devices.

## A NEW ELEMENT: THE DDDE

In addition to those embedded communication software development elements shown in Figures 1 and 2, a new element is now becoming part of the embedded communication application developer's standard repertoire of tools and reusable code. This new element is called a Device Driver Development Environment (DDDE). This kind of tool provides automated support and acceleration for the long-neglected area of Board Support Package and Device Driver development.

A single development project often requires the development of a number of differing BSPs and sets of device drivers. These may be categorized as follows:

- Evaluation Board Basic BSP.
  - For projects that begin development using a commercially-available target computer board (COTS hardware).
- Evaluation Board BSP with RTOS Support.
  - Modified BSP to allow application software development to begin using an RTOS.
- Evaluation Board BSP covering all Application Requirements.
  - Modified BSP to handle additional requirements of application software.
- Target Hardware Board Development BSP's.
  - BSPs to help hardware engineers to develop the customized hardware board that will serve as the true platform for the embedded communication software. This BSP will evolve incrementally as the hardware is developed incrementally.
- Target Hardware Board BSP supporting RTOS and Application.
  - For Software-Hardware Integration.
- Final Target Hardware BSP.
  - Modified BSP to accommodate last changes to hardware and application.
- Production Test BSP.
  - Overhauled BSP and exercise programs for hardware manufacture tests.
- Target Hardware Diagnostics Package.
  - Overhauled BSP and test programs for hardware field diagnostics.

Because of this long list of BSP variants that are often needed in a single project, there is great benefit to automated support for the BSP construction process. This is driving the DDDE tools industry and driving demand for its products.

A DDDE tool is organized with a modern user interface that serves to encapsulate a "Knowledge Base" shown in Figure 3.

Because of the complexity and uniqueness of many on-chip and off-chip communication devices, the source code blocks stored in the knowledge base must be hand-coded by (human) experts in BSP and

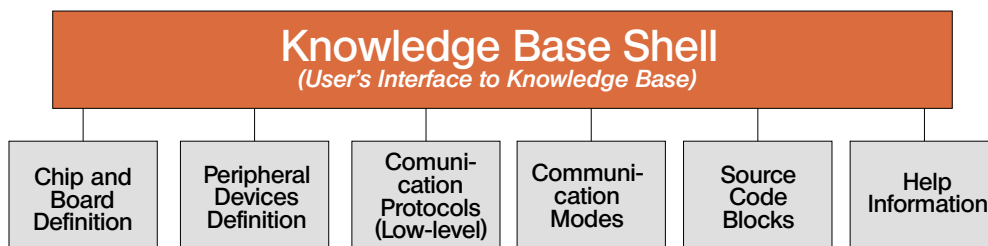


Figure 3. DDDE Tool Shell and Knowledge Base.

# DEVICE DRIVERS

device driver development. The code is structured in parameterized fashion, and must be thoroughly tested before deposit into the knowledge base.

## STEPS TO AUTOMATED BSP AND DRIVER SUITE CONSTRUCTION

Users must go through a series of standard steps to have their DDDE tool construct a BSP and accompanying drivers suite:

- Project Environment Specification
- Hardware Configuration Setup
- Device Driver Selection and Specification
- Code Construction

Project Environment Specification includes compiler selection and RTOS selection. Choice of compiler influences the constructed code through factors such as data structure packing and interrupt handling. RTOS requirements can influence the entire structure of a BSP by, for example, modifying the BSP's execution sequence and handling of interrupts.

Hardware Configuration Setup includes specification of such issues as communication processor memory map, clocks, external interrupts priorities and nesting, internal interrupts handling, and reset behaviors. For many software/firmware developers, some of these issues may be unclear; hence a good DDDE tool will always offer default values for all choices and parameters that will lead to construction of a reasonable "first-cut" BSP and drivers suite. In addition, extensive user help facilities must be provided by the DDDE tool to clarify detailed technical issues.

Device Driver Selection and Specification is the heart of a DDDE tool. For example, this is done at a "Chip Explorer" screen in DriveWay, a popular DDDE tool shown in Figure 4.

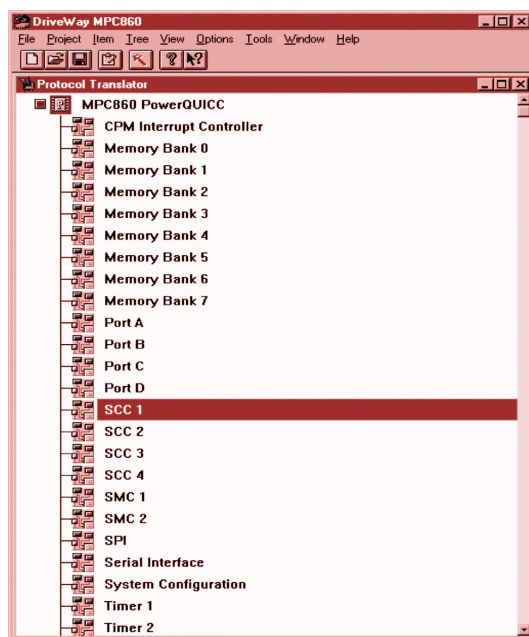


Figure 4. DriveWay DDDE Tool "Chip Explorer" Screen.

The Chip Explorer screen shown above is simply an indented list showing all of the peripheral devices (communication devices and otherwise) for which drivers are available, for the specific communications chip being displayed. Devices are listed in alphabetical order, and extensive technical "help" is available for each. A user begins the process of detailed specification of a communication device and its driver, by simply clicking on the device in this screen. A series of more and more detailed specification screens are then displayed for this device, until its driver requirements are completely described. At all stages, reasonable default values appear in all fields of all screens - so that "first cut" reasonable drivers can be obtained quickly without the need for extensive research into a plethora of options and registers.

Once a driver's requirements have been specified, the Chip Explorer screen is redisplayed. A user can then select another communication device and go through its driver specification cycle. After all communication devices of interest have been specified in this way, the user can request Code Construction by a single mouse-click.

Code Construction is an automated process of integrating information and source code blocks from the Knowledge Base, with user specifications input in the previous steps. Code Construction typically takes several minutes, and results in several tens of thousands of lines of BSP and driver source code, neatly organized into groups of files. These include:

- Boot code files (in Assembler)
- Terminal Device Driver Files (for target-resident debugger interface)
- Network Device Driver Files (for source-level debugger interface)
- Communication Device Driver Files (in C-language)
- Communication Device Driver Test Function Code Files
- Other Device Drivers and Test Function Code
- Real-Time Operating System Interface Code Files
- Configuration Files
- Makefiles

Code Construction is flexible, so that if a user inserts additional code into these files and then requests code construction again - the user's added code is protected and preserved in the newly constructed code. Very often, several Code Construction cycles are necessary as users gain a better understanding of the communication devices and the application needs of their communication system.

## NEW PARADIGM ACCELERATES BSP AND DRIVER SUITE CONSTRUCTION

The advent of DDDE tools and their addition to the embedded communications application developer's standard repertoire of tools and reusable code, has introduced a new paradigm for BSP and driver suite construction. It is no longer necessary to spend long man-months and calendar months dealing with the bit-and-byte level "nuts-and-bolts" hardware/software

## DEVICE DRIVERS

interface issues. DDDE tools have raised the level of abstraction of communication hardware/software interface issues to a high-level mouse-and-menus language. And this high-level approach has provided benefits of increased quality and increased efficiency of software development.

It is believed that these tools and the low-level code that they help to construct, can contribute significantly to shortening the time-to-market for new embedded communication products ■

---

*David Kalinsky is Director of Customer Education at Aisys, a creator of device driver development environments. He is also a lecturer and seminar leader on technologies for embedded software. Kalinsky has built high-tech training programs on aspects of*

*software engineering for the development of real-time and embedded systems for a number of Silicon Valley companies. He has been involved in the design of many embedded medical and aerospace systems. Kalinsky holds a Ph.D. in nuclear physics from Yale.*



**Dedicated Systems**  
**Encyclopaedia**

---

<http://www.dedicated-systems.com>

# AD POLYHEDRA