

DSP/BIOS

Standard Software Infrastructure for the DSP Mainstream

This article presents DSP/BIOS, a Real-Time kernel for DSP Development. After analysing the development methods used in the DSP world, the author explains the benefits of using such a tool. DSP/BIOS also includes some instrumentation tools that help the developer in tuning applications.

THE DSP SOFTWARE CHALLENGE

Programmable DSPs have evolved over the past decade from dedicated number crunchers into a more universal embedded processor, one not only optimized for executing numerically-intensive signal processing algorithms but also capable of performing many system control and communication tasks normally relegated to general-purpose microcontrollers. While this trend first became apparent in high-end 32-bit DSPs, a similar architectural evolution has begun to take hold in lower-cost 16-bit devices such as the TMS320C5x, C54x, and C2xx - devices representative of an industry segment we term the mainstream of DSP.

As developers and integrators begin to embed mainstream 16-bit DSPs in more sophisticated applications addressing market segments ranging from telecommunications to motor control, managing the corresponding rise in software complexity emerges as a primary challenge to keeping overall product life-cycle costs in check. In fact, it is the development and integration of software (not hardware) which has become the critical factor in ensuring timely delivery of robust DSP-based products to the marketplace, tracking a similar trend well understood within the broader embedded systems community.

The use of high-level programming languages like C has already proven itself as the single most effective tool for streamlining software development in general; and not surprisingly C plays an evergrowing role in the development of mainstream DSP applications. High-level language support for 16-bit TMS320 DSPs has significantly improved over recent years in terms of instruction-set assistance as well as compiler maturity, enabling a wider circle of developers to rapidly tackle complex system functions in mainstream DSP applications without recourse to assembly-language programming. At the same time, severe cost constraints within these applications demand efficient utilization of MIPS and memory resources — incompatible with the overhead typically incurred by C — and consequently dictate heavy use of hand-tuned assembly-language to meet system performance objectives.

Beyond high-level languages, another effective methodology for shortening embedded application development cycles has been (re)use of standard software components, the most basic of which are commercially available as real-time kernels augmented by I/O libraries. While such products do exist within the DSP arena (and have even become commonplace in high-end segments), pressure to conserve precious system resources — especially memory — have precluded use of this technology on a broad basis within mainstream 16-bit applications. Software developers will consequently (re)invent their own basic task scheduling and I/O routines from one project to the next; and while ostensibly justified by system performance constraints, these sorts of in-house kernels invariably introduce hidden support costs further downstream in the product lifecycle.

The lack of standard kernel components for 16-bit DSPs causes further repercussions, especially when integrating third-party signal processing algorithms into a particular application. Without widely deployed conventions for scheduling tasks and managing I/O

streams, independent software vendors today will often carry out protracted amounts of custom engineering to fold their own software components into a developer's target environment, adding both time and cost to system integration and product test. The vision of "plug-and-play" algorithms for mainstream DSP applications remains an elusive one.

Recognizing, then, a software landscape where high-level language must be conservatively deployed and where algorithms are costly to integrate with inhouse kernels, the following observation should come as no surprise: given the size and growth of the mainstream DSP market as measured in unit or dollar volume, the number of developers in this space is alarmingly low compared with the robust embedded microcontroller community. A similar observation can be made about the diminutive stature of the third-party software vendors who supply the mainstream DSP market with critical intellectual content in the form of signal processing algorithms, but who find themselves caught in unending cycles of timeconsuming custom engineering.

To grow the number of designs deploying main-

The vision of "plug-and-play" algorithms for mainstream DSP applications remains an elusive one.

RTS Ad

stream DSPs for new applications, and to reduce the time required to bring these products to market, DSP developers must be given the same advantage as other software practitioners — the advantage of standard infrastructure.

To meet this challenge, Spectron and Texas Instruments are together taking first steps towards establishing a pervasive software infrastructure for the DSP industry. At the heart of this initiative lies DSP/BIOS, a small firmware kernel of basic tasking and I/O services scaled to meet the requirements of mainstream applications employing 16-bit TMS320 DSPs. Coupled with hosted visual utilities for real-time program analysis, DSP/BIOS furnishes software developers with a common set of capabilities on par with other segments of the embedded system community and yet efficient enough to meet the performance constraints of mainstream DSP applications. By building upon its open APIs, DSP/BIOS enables third-party ISVs to offer suites of interoperable tools and components that further empower developers and integrators to reduce time-to-market through prudent management of rising software complexity.

A REAL-TIME DSP ENVIRONMENT

The basic development environment for today's mainstream TMS320 DSPs comprises two categories of software tools:

- a program generation suite featuring a C compiler, assembler, and linker for the target processor family
- a program debugger for executing application software on an adjoining hardware platform.

While comparable in capability to their microcontroller counterparts and unquestionably essential to any software development environment, these tools nevertheless fail to directly address a critical dimension of embedded systems in general and signal processing applications in particular — the dimension of real-time.

Working in concert with the basic TMS320 software tools, DSP/BIOS effectively extends the mainstream DSP development environment with an equally essential set of real-time software capabilities. The following figure depicts the additional elements introduced into the environment by DSP/BIOS as well as their relationship with other software tools and com-

ponents supplied by Texas Instruments and its third-parties.

DSP/BIOS Kernel

This small, firmware kernel forms the hub of the environment, furnishing basic run-time services to embedded application programs executing on target DSP hardware. Requiring less than 1K words of memory, the kernel supports real-time threads through its preemptive task scheduler, real-time I/O streams through core modules for managing data flow, and real-time capture through a series of functions that record information about the target program during its course of execution. The latter capabilities drive a companion set of hosted DSP/BIOS analysis utilities that are outlined below.

The target application invokes DSP/BIOS kernel services by embedding corresponding API calls within its program source code. Respecting the more conservative use of high-level language within the DSP community, especially for 16-bit applications, DSP/BIOS supports both C as well as assembly-language renditions of its kernel

APIs. Once compiled or assembled, applications invoking DSP/BIOS services are bound into executable programs using the standard TMS320 linker along with a library of kernel object modules. To further reduce target memory costs, Texas Instruments plans to embed DSP/BIOS kernel firmware directly into the on-chip ROM of future "mass-market" TMS320 DSPs.

The DSP/BIOS kernel implements basic tasking and I/O services that represent a necessary, but not always sufficient, set of run-time support functions required by many target applications. To complete the picture, an embedded DSP application generally needs additional software components of a less generic nature than the kernel itself, notably: drivers, which encapsulate low-level access routines for controlling system-specific hardware I/O devices; and algorithms, which process incoming data streams in an application-dependent fashion.

Though developers can implement hardware drivers and signal processing algorithms directly within the context of DSP/BIOS tasking and I/O services, we foresee the emergence of third-party program frameworks to help bridge the gap between the rudimentary capabilities of the underlying kernel and the specific requirements of vertical applications. Effectively a

DSP/BIOS, a small firmware kernel of basic tasking and I/O services scaled to meet the requirements of mainstream applications employing 16-bit TMS320 DSPs.

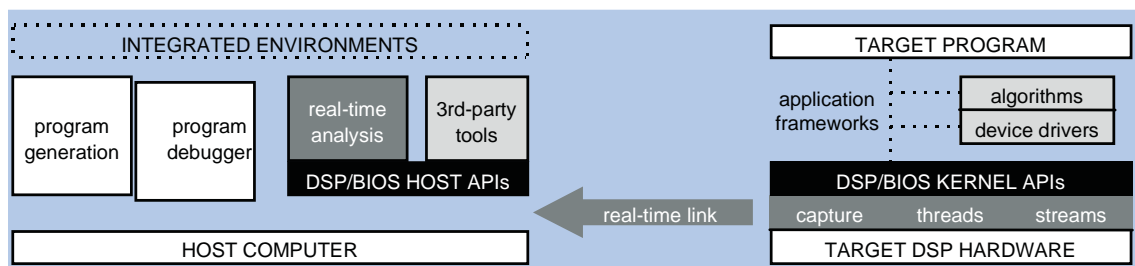


Figure 1.

software skeleton, the framework would dictate programming interfaces and conventions that each driver or algorithm must respect, ensuring these elements can be slotted in and out of the target application as circumstances require; the framework itself would then implement the necessary software "glue" atop the DSP/BIOS kernel to bind these plug-in modules into a more cohesive whole.

DSP/BIOS Analysis Utilities. Flanking the familiar program debugger, these utilities work hand-in-hand with the target-resident DSP/BIOS kernel to enable real-time program analysis, a set of visual capabilities that allow developers and integrators to trace, monitor, and probe a DSP application during its course of execution. Hosted program analysis presumes the presence of the DSP/BIOS kernel within the target system, not only to support real-time communication with the host over a suitable data link but also to provide rudimentary run-time services to the application itself — unlike a traditional debug monitor which is largely transparent to the executing program. By simply structuring their application around basic DSP/BIOS tasking and I/O services, developers automatically instrument the target for capturing and uploading real-time information that drives the visual analysis utilities on the host; supplementary APIs allow explicit information capture under target program control as well.

From the perspective of its hosted visual utilities, DSP/BIOS affords several broad capabilities for real-time program analysis:

- **event logging**

capabilities for displaying time-ordered sequences of events written to kernel log objects by independent real-time threads, tracing the program's overall flow of control; events can be logged explicitly by the target program through DSP/BIOS API calls or implicitly by the underlying kernel when threads are readied, dispatched, and terminated.

- **statistics accumulation**

capabilities for displaying summary statistics amassed in kernel accumulator objects, reflecting dynamic program elements ranging from time-varying data values to elapsed duty cycles of independent threads; statistics can be accumulated explicitly by the target program or implicitly by the kernel when scheduling threads for execution.

- **file streaming**

capabilities for binding kernel I/O objects to host files, providing the target program with standard data streams for deterministic testing of algorithms; other (real-time) target data streams managed with kernel I/O objects can be tapped and captured on-the-fly to host files for subsequent analysis.

When used in tandem with a standard debugger during software development, the DSP/BIOS real-time analysis utilities provide critical visibility into target program behaviour at exactly those junctures where the debugger offers little or no insight — during program execution. Even after the debugger halts the

program and assumes control of the target, information already captured through DSP/BIOS can provide invaluable insights into the sequence of events that led up to the current point of execution.

As windowed program generation and debug tools become prevalent inside the DSP community, the next step in their evolution would be incorporation of DSP/BIOS real-time analysis capabilities into an integrated programming environment, enabling developers to seamlessly transition from one tool to the next within a unified visual setting. To facilitate third-party offerings in this arena, all of the hosted DSP/BIOS utilities rely upon an open programmatic interface for interacting with an operative target system incorporating the DSP/BIOS kernel. Due to their modular design, individual DSP/BIOS utilities can be readily subsumed within an integrated environment and, where appropriate, extended with additional features; an open DSP/BIOS host interface further encourages third-party vendors to field their own suites of tools and utilities for program analysis, especially in concert with a vertical application framework for the target system.

Later in the software development cycle, when traditional debuggers become ineffective for attacking more subtle problems arising from time-dependent interaction of program components, the DSP/BIOS analysis utilities assume an expanded role as the software counterpart of the ubiquitous hardware logic analyzer. This dimension of DSP/BIOS becomes even more pronounced after software development concludes, when the embedded DSP/BIOS kernel coupled with its hosted analysis utilities lay the necessary foundation for a new generation of manufacturing test and field diagnostic tools capable of interacting with target application programs in operative production systems. Here too, the open host interface underlying the DSP/BIOS analysis utilities become a catalyst for next-generation environments less geared towards program developers but rather addressing the unique requirements of field and factory personnel.

STANDARD DSP INFRASTRUCTURE

A key design objective of DSP/BIOS has been to furnish mainstream DSP developers with a basic set of software building-blocks which, at once, must support a wide range of embedded applications and yet incur only minimal time/space overhead within the system as a whole. While a notable technical achievement in its own right, this design objective is prefatory to a more overarching goal of Spectron and Texas Instruments — to establish DSP/BIOS as pervasive software infrastructure for the industry as a whole.

One immediate benefactor of DSP/BIOS as common foundational software would, of course, be the developers themselves. No longer shackled by the need to perpetually re-invent the software equivalent of wiring and plumbing with each new edifice they build, developers can concentrate their efforts upon the differentiating elements of the target DSP application which

ultimately contribute competitive product advantage in the marketplace. Besides shortening the application development cycle through reuse of common software components, DSP/BIOS leads to greater product robustness and reliability for an interrelated pair of reasons:

- 1) In-house kernels, though small and seemingly simple, lead to a disproportionately high incidence of target software defects due to insufficient validation of the kernel itself; the DSP/BIOS kernel by contrast has already been subjected to levels of functional and performance testing far beyond the scope of most project budgets, and will achieve even higher degrees of stability through widespread deployment in a broad class of applications.
- 2) By building upon the run-time services implemented within the DSP/BIOS kernel, target applications automatically accrue the benefits of real-time program analysis through a rich set of host utilities; while an essential companion to familiar program debuggers during software development, the DSP/BIOS analysis utilities are without peer for diagnosing more subtle application defects that arise later in the product life-cycle.

Put another way, developers enjoy a certain economy of scale by embracing DSP/BIOS as common software infrastructure, in that the cost of thoroughly validating the target kernel as well as creating feature-rich host analysis utilities are born by Spectron and amortised across hundreds of projects.

Aimed at current as well as future generations of mainstream TMS320 devices, DSP/BIOS supports a uniform set of capabilities across multiple processor families in much the same way as the familiar TMS320 program generation and debug tools maintain their innate "look-and-feel" despite differences in the target processor architecture. While this methodology could well enable new standards for migrating application components among mainstream DSPs (though software portability in reality assumes less importance in many performance- and cost-driven systems that eschew C in favour of hand-crafted assembly language), the real advantage of uniform DSP/BIOS support across a range of TMS320 devices becomes the ability to migrate the developers themselves — to re-use their prior knowledge and experience in working with DSP/BIOS as they transition from one project to the next.

Another major benefactor of broad DSP/BIOS support would be third-party vendors who supply mainstream developers with tools and components ranging from hardware platforms and processor emulators to program debuggers and signal processing algorithms. Lacking industry direction, these vendors (like developers) will continually re-invent foundational elements already present in DSP/BIOS and expend engineering effort better channelled towards adding value to their own products.

Pervasive software infrastructure also creates a measure of stability for third-party vendors, removing the uncertainty of which (if any) kernel-level tools and

components their own products should support. Already encumbered with support for different processor families, eliminating the added dimension of multiple software kernels mitigates an otherwise combinatorial number of products a third-party vendor must offer. The ultimate benefactor once again becomes the DSP application developer, who can leverage the ubiquity of DSP/BIOS through a growing set of inter-operable third-party products.

The key point here is that robust software inter-operability, whether plug-and-play algorithms or mix-and-match tools, will only flourish when the DSP industry consolidates around a single kernel standard. Though ostensibly monopolistic in nature, we learn from other industry segments that introducing choice at the infrastructure level — be it PC operating systems, video recording formats, or shapes of electrical outlets — can actually inhibit competitive growth by fragmenting the marketplace. By embracing DSP/BIOS, our industry can anticipate a new generation of third-party tools and components whose maturity, richness, and inter-operability gives mainstream DSP developers unprecedented advantage in tackling more sophisticated applications while keeping product lifecycle costs in check. ■

As Vice President of Technology and co-founder of Spectron Microsystems, Bob Frankel is responsible for the strategic direction and software architecture for Spectron's systems software products. Generally regarded as a visionary in the DSP industry, Bob was one of the first to conceptualize how software could harness the capabilities of DSPs beyond simple number crunching devices. Bob holds an MS degree in Computer Science

Eonic Ad