

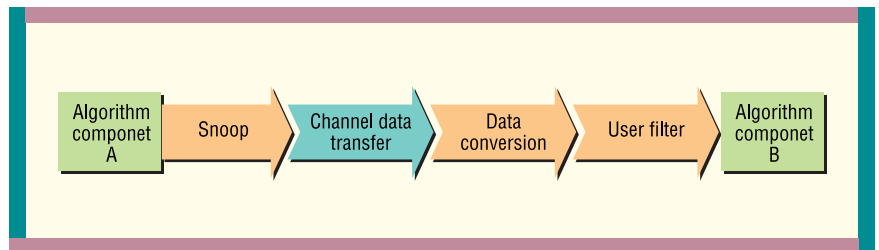
With TI's algorithm standard and the proper framework, you can use DSP algorithms without modifying the original source.

# Component-based Programming Comes to DSP Algorithms

By Arda Erol

The DSP industry is similar to the general-purpose processor industry. Every year new DSPs significantly outdo older models in speed, power consumption, and price. As a result, these processors are used in areas once considered out of reach. Although faster clock speeds relieve the signal-processing software developer from hand-optimizing code, the use of more complex algorithms and the higher ratio of software versus hardware in many applications result in more complex software. However, complex software is nothing new for the mainstream software industry, which has embraced component-based programming techniques such as COM (Component Object Model), DCOM (Distributed Component Object Model), and CORBA (Common Object Request Broker Architecture).

Although these models don't target real-time applications, component-based programming techniques can be used at many levels of complexity, allowing you to easily bal-



*Figure 1. Component-based communication channels enable additional operations to be inserted in the data path, allowing advanced features such as real-time monitoring of data and data format conversions in heterogeneous systems.*

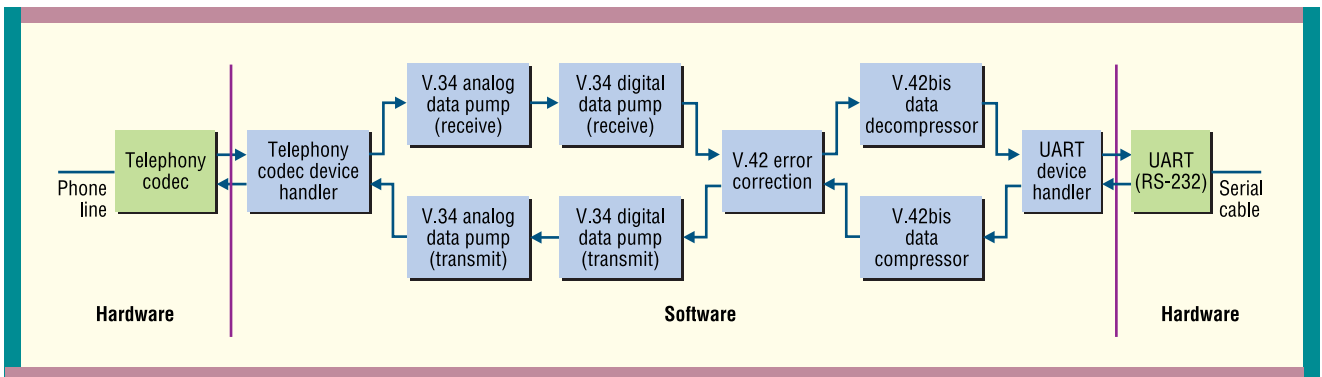
ance modularity (which comes with a whole set of advantages) and performance. Despite widespread belief in the DSP industry to the contrary, component-based methods can be easily applied in C and assembly language without an object-oriented language, such as C++.

### A RICH FRAMEWORK

With the advent of Texas Instruments' TMS320 DSP Algorithm Standard, component-based programming is feasible at the algorithm level. The standard lets you easily

use DSP algorithms from other companies in your own applications. You also need a system-level framework to encapsulate those components in applications. A powerful framework becomes more important when an application requires multiple DSPs or a mix of DSPs and general-purpose processors.

A rich framework enables you to move entire algorithms or even parts of an algorithm from one task or processor to another and reuse algorithmic building blocks in different algorithms (see "Bringing It All Together," page 24). More impor-



**Figure 2.** Like most DSP applications, a V.34 modem can be implemented as a collection of independent software components, even hardware device handlers. You can do that even with hardware device handlers, which usually represent I/O points of an application, like the telephony codec device handler and the UART device handler shown here.

tantly, you can do that without modifying—or even possessing—the source code of the original algorithms or algorithmic building blocks.

If you use traditional programming techniques, modifying the source code is unavoidable: Functions must be organized into separate builds to run on different processors, and the necessary communications must be added between the separated modules. Even then, every time you move part of

decreases, the interchangeability of components decreases and the system ceases to be scalable.

## HEAVYWEIGHT AND LIGHTWEIGHT

Algorithms can be characterized as either heavyweight or lightweight. A heavyweight algorithm is a complete DSP algorithm such as a G.723 vocoder, an MPEG-2 video encoder, or a V.34 data pump. Lightweight algorithms, or algorithmic

source code, and easier system-level debugging.

With this approach, the DSP industry can define standard interfaces for different categories of algorithms. TI's TMS320 DSP Algorithm Standard, part of eXpressDSP, is an excellent example. Establishing such standards is key to enabling developers to interchange different implementations of an algorithm from different vendors without substantial effort.

At a lower level, algorithms can be

## Component granularity plays an important role in finding the correct balance between meeting run-time requirements and maintaining modular programming.

an algorithm from one processor to another, you have to repeat the same tedious process.

Component granularity plays an important role in finding the correct balance between meeting run-time requirements and maintaining modular programming. As the granularity of components increases, so does the chance to incur unnecessary overhead in an application. On the other hand, as the granularity

mic building blocks, are smaller DSP functions, such as a finite impulse response filter or a discrete cosine transform.

At the highest level of component granularity, each heavyweight algorithm is bundled into a distinct component. The advantages include distribution of algorithms in binary form, deployment of complete algorithms to different tasks and processors without modifying algorithm

split into smaller building blocks, such as filters, echo cancellers, and fast Fourier transforms. The finer granularity has additional benefits: deployment of a single heavyweight algorithm over multiple processors without modifying (or, again, even possessing) the algorithm source code, block-diagram-based design of DSP algorithms, reuse of algorithmic building blocks, and easier algorithm-level debugging.

## BRINGING IT ALL TOGETHER

Spectrum Signal Processing is currently developing a component-based application framework called Accelera. Accelera allows you to distribute components from various sources to multiple processors. On each processor, you can group components into one or more threads or tasks. At run time, you can create snoop connections to monitor intercomponent communications without stopping the application code. Overall, Accelera supports binary algorithm code distribution; code reusability;

with two target processors and one development (host) processor. The solid circles show algorithm components (lightweight or heavyweight), and the dashed circles show threads and tasks. Components can be implemented as collections of other components, as in the case of c5.

Once you've implemented the lowest-level components (in C or assembly) or imported them (for example, eXpressDSP-compliant algorithms or Simulink-generated code), you can bring them into the

Accelera block-diagram design tool and combine them to form more complex components or a complete application to be deployed on one or more target processors. The set of processors can be a combination of DSPs, microcontrollers, and general-purpose processors. When an application is executed on the target processors, the block-diagram design tool turns into a real-time debugging tool that lets you to snoop on communications between components.

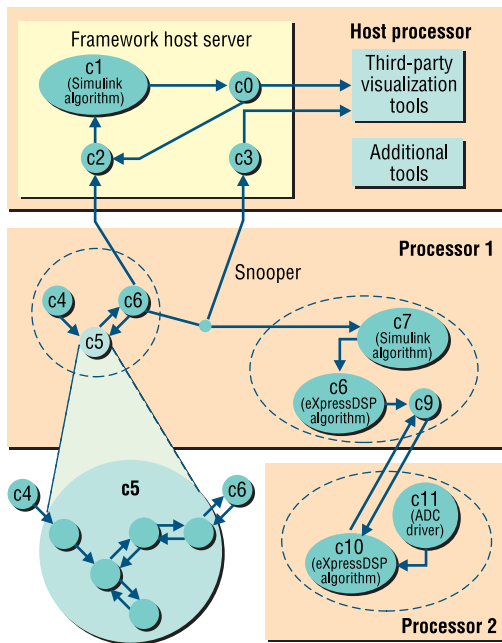
If you notice load-balancing problems, you can move components from processor to processor without touching the components' source code. For embedded solutions, you can detach the host system from the target processors at the end of the development cycle by encoding the default configuration of all components into the DSP software.

Several approaches to component-based programming at the algorithmic building-block level have been tried. One approach—defining each component as a thread or task—is taken by a number of commercially available development environments. The one shortcoming is insufficient component granularity. Those systems don't prevent you from implementing even small algorithmic building blocks as separate components; because of the context switch overhead between components, however, you must minimize the number of algorithm components to maximize performance.

## The system integrator can introduce a custom implementation.

Hence you must consider the capabilities and number of processors to efficiently split the algorithm into components. This approach greatly increases the dependency on a specific target platform.

A more advanced approach to component-based software development involves defining algorithm components to be independent of operating system entities, such as threads and tasks. With this approach, you group multiple components inside a single thread or task and schedule the components in a static order or a dynamic order programmed by the user. That eliminates context switching when it's not a requirement of the application. In addition, since components within a thread are inherently synchronized, the need to synchronize communications between components is also eliminated. Con-



ty; integration with DSP algorithm design tools, such as Simulink; support for algorithm standards, such as eXpressDSP; partitioning of algorithms to heterogeneous multiprocessor systems, including hosts; and advanced real-time debugging support with the ability to monitor and modify intercomponent communications.

The figure shows Accelera in a system



## Hand-In-Hand Power of DSP + Flexibility of FPGA

# SMT335E

TI's C6xxx DSP

SDB Connector



Xilinx Virtex 2000E FPGA

**Features:** Compact module, four SDB interfaces for fast IO or for inter processor communication(200MB/S/SDB), six built-in comports, FLASH for embedded applications.  
Available support for PCI, CPCI, VME and VXI. Can cascade multiple modules.

### SUNDANCE DIGITAL SIGNAL PROCESSING INC.

4790 Caughlin Parkway 233, Reno, Nevada 89509-0907, U.S.A.

Tel: (775) 827-3103 · Fax: (775) 827-3664

### SUNDANCE MULTIPROCESSOR TECHNOLOGY LTD.

Chiltern House, Waterside, Chesham, Bucks, HP5 1PS, United Kingdom.

Tel: +44 (0)1494 793167 · Fax: +44 (0)1494 793168

Email: [sales@sundance.com](mailto:sales@sundance.com) <http://www.sundance.com>

SUNDANCE

sequently, you have very efficient data-passing techniques, such as passing pointers to data, rather than transferring actual data.

Component-based programming can be extended to the implementation of communication libraries that provide the means of exchanging data between algorithm components. With such libraries, the system integrator can choose the optimal communications implementation based on the communication needs of the algorithms used and their physical locations. Moreover, when faced with unsupported communications hardware, the system integrator can introduce a custom implementation, which can then be

Furthermore, interaction between components can even be tampered with, which can be turned into a very powerful debugging tool (Figure 1).

One important advantage of component-based design at the algorithmic building-block level is its ability to provide a natural integration path for DSP development tools, like MathWorks' MATLAB and Simulink. Following the thread-independent component approach, you can convert even individual Simulink blocks into components without incurring much overhead. That way, once the application is deployed on a target platform, only frequently called

ing the processor is fast enough to handle a multichannel modem implementation.

Moreover, the connections between the components are easy to modify for different tasks. For example, you can add an additional component before the data compressor and decompressor blocks to multiplex data to multiple instances of the error correction and data pump components to send data over multiple phone lines. Best of all, that can be done without access to the existing components' source code.

## EASE OF DEVELOPMENT

If the component model in use is also supported on a host processor, it becomes easier to develop each component independently. For example, the programmer of the data compressor can connect the input and output of the component to a host processor that reads from a file and another one that writes to a file. The programmer can then test the data compressor on actual hardware before the other components are completed by other programmers.

Similarly, the programmer of the error correction component can create two instances of it and connect them to each other for debugging and then stream test data from the host PC via host-to-DSP channels. During system integration, the data pump blocks and the compressor and decompressor blocks can be grouped into two threads, thereby avoiding task switching and synchronization overhead among components within each thread. ♦

*Arda Erol* (arda\_erol@spectrumsignal.com) leads a software group at Spectrum Signal Processing, Inc. in Burnaby, B.C., that develops future software products. His expertise includes DSP software development, real-time operating systems, and video/image processing.

## The very same component-based approach can be extended to the entire set of operating system entities.

used by all existing algorithm components.

Similarly, there's no reason to stop at the communications layer. The very same component-based approach can be extended to the entire set of operating system entities, including heaps and threads, to provide the same level of flexibility throughout the system. For example, by letting the user create multiple heaps in different memory regions, where each memory region may have distinct physical properties, a component-based heap implementation solves the thorny problem of having more than one physical type of directly accessible memory.

A well-designed component model can also provide a means of monitoring interactions between components. The user gains insight into any part of the system at run time without placing debug statements inside the application code.

algorithm components need to be ported to assembly language (or reimplemented by hand in C), and you don't have to worry about other components of the application. Even if all components are eventually reimplemented by hand, using imported blocks from DSP algorithm design tools makes development a lot easier, as it provides a functional system much earlier in the development cycle.

## A REAL-LIFE EXAMPLE

V.34 modem software, for example, can be implemented as a collection of individual components (Figure 2), and each component can consist of yet another set of subcomponents. The components can be easily distributed to multiple processors, or multiple instances of the same collection of components can be executed on a single processor, assum-