Helping Match DSPs and Data Converters

JTAG Emulator Speeds DSP Board Debugging

Advances Supercharge DSP Development Tools

October 2000
Inside This Issue

Insighter
Introducing Embedded Edge: A new tool to help hone your competitive edge.

Breakpoints
News from the providers of embedded systems development products and services.

Cover: Advances Supercharge DSP Tools
With DSPs jumping in performance and spreading throughout the embedded world, software tools are answering the call to speed development.

Helping Match DSPs and Data Converters
An add-on tool lets you evaluate, configure, and apply ADCs and DACs.

JTAG Emulator Speeds Debugging
Emulation builds confidence in hardware bit by bit and lets the software design begin earlier.

Modeling Tackles Echo Cancellation
Modeling and simulation have become essential to developing advanced DSP systems, as an echo canceler for a wireless phone shows.

Wizards’ Corner
Answers to developers’ questions from experts in embedded systems development.

Launchings
New products and services for embedded systems developers.

On the Edge
What’s so important about an algorithm standard?
Welcome to Embedded Edge, a software magazine designed for embedded developers.

If you live, eat, and drink embedded systems, think of Embedded Edge as your “food” consultant, willing and eager to help you solve your everyday engineering challenges. To that end, Embedded Edge is devoted to bringing you practical information to help you successfully develop high-performance embedded systems—on time and within budget.

Embedded Edge is brought to you by Texas Instruments and the hundreds of members of its DSP Third Party Network. Our mission is practical and educational: to help you use advanced techniques and tools, solve problems, learn about new capabilities, see the approaches your colleagues have taken in their development experiences.

In the magazine, you’ll find comprehensive solutions, tips, tricks, and advice from people who have met and conquered some of the development problems you face every day. You’ll also find information on news, products, and services that will help make your life as an embedded systems developer easier.

We invite you to tell us about the problems you need to solve when developing embedded systems. We’ll put your questions to the collective knowledge of Texas Instruments and the Third Party Network Members, and we’ll do our best to answer them in our pages. Chances are very good that the experts on the TI team not only have seen your problem, but have already conquered it.

Why a new magazine, when hundreds are already out there? Because in an attempt to keep up with the amazing leaps taken by the DSP chips and other ICs, embedded software tools are making incredible advances—but no one is telling the story of the new tools and how to make use of them to build advanced products in ever narrowing windows.

If they’re to get to market with the most sophisticated products, DSP system developers realize they must leave assembly language programming behind. Clearly, the future belongs to high-level languages, optimizing C compilers, and visual development environments and emulators. Our lead article lays out how the world-class capabilities enjoyed for years by native-processor programmers and other engineers are becoming available to their embedded DSP colleagues, supercharging software tools for embedded development.

These advances, alone, aren’t the answer. A development environment for today’s and tomorrow’s advanced products must integrate the array of tools into a cohesive continuum. To meet that need, it must be able to readily accept compliant plug-in tools from diverse sources that add specialized functionality.

If you use TI’s Code Composer Studio to develop code for TI chips, plug-ins to CCS, now appearing from third-party vendors, promise to make your development life a thousand times easier. Inside, you’ll find out how to deploy a plug-in to design in the exact parameters you need in a data converter.

Emulation is a standby that engineers have used for years to design and verify their products. If you’re putting JTAG, or boundary scan, testability on your board or system, check out the article from Spectrum Digital, which shows how to apply emulation to that task.

Modeling and simulation are two other invaluable design and verification capabilities. How they can apply in the embedded world is indicated in the MathWorks article inside, which describes the design of an echo cancellation circuit.

We welcome your comments on this, our first issue, and suggestions on what you’d like to see in upcoming issues.

—Stan Runyon

testman2@earthlink.net
Mentor and Enea Attack High-End Development

Mentor Graphics Corporation (Wilsonville, Ore.; www.mentor.com) and Enea OSE Systems, Inc. (Täby, Sweden; www.enea.com) have agreed to work together to provide best-in-class software development and debugging tools for high-availability, high-reliability embedded applications. As a result of the collaboration, Enea will market and distribute an integrated tool suite consisting of Mentor Graphics’ XRAY debugger; Enea’s OSE RTOS; and support for Microtec, GNU, and Diab compilers.

XRAY includes such features as RTOS object browsing, thread and process awareness with window-per-thread visibility, and support for debugging multiprocessor systems. The OSE RTOS accommodates fault-tolerant systems.

Voice Signal Technologies Finds Funds

Voice Signal Technologies, Incorporated (Cambridge, Mass.; www.voicesignal.com) has secured $8.4 million in Series C equity funding from an investment group headed by Stata Venture Partners, Inc. The money will be used to develop and market high-performance, low-cost speech interface products for mobile devices, smart home controls, interactive toys, and automotive applications.

Tundra to Acquire Quadic Systems

Tundra Semiconductor Corporation (Kanata, Ont.; www.tundra.com) plans to acquire one of its suppliers, Quadic Systems, Inc. (South Portland, Maine; www.quadic.com). The acquisition is intended to boost Tundra’s capability in system interconnect technology, including designing devices based on the high-speed Rapid IO interconnect architecture. In particular, Quadic’s back-end layout and analog design expertise will provide Tundra with in-house capability in chip layout and the design of highly specialized analog circuitry.
ADVANCES SUPERCHARGE DSP SOFTWARE DEVELOPMENT TOOLS

This is the first part of a two-part report on trends in DSP software development tools. Part 2 will appear in the February 2001 issue.

Software for embedded development is becoming supercharged. The driving forces are the recent startling advances in DSP hardware and the mounting pressure to bring embedded products to market ever faster. But further chip advances without commensurate gains in software will diminish productivity, so the embedded development community has begun to demand high-level capabilities in development tools. Not only do designers want assurance that they can tap all the available on-board features, but they insist on doing it without delays in getting to market.

Only high-level and comprehensive development software can respond to all those needs at once. And since that increasingly translates into off-the-shelf commercial software, designers are deserting the home-brewed variety in droves.

If they are to satisfy the overwhelming desire to get to market
fast with the most sophisticated products, DSP developers also have little choice but to leave behind what is probably the last bastion of assembly language programming: DSP software development. They’ve come to realize that the future belongs to high-level languages and optimizing C compilers as well as visual development environments and emulators—even if a few MIPS must be traded off in the bargain.

All of that means that the world-class capabilities enjoyed for years by native-processor programmers are becoming available to their embedded DSP colleagues.

TOOLS FOR NON-DSP EXPERTS
People want higher levels of abstraction for development, says Will Strauss, president of Forward Concepts, the Phoenix, Ariz., market forecaster, “because few understand the math behind DSP algorithms. They want tools that allow non-DSP experts to program DSP chips, like MathWorks’ MATLAB and Texas Instruments’ Code Composer Studio.”

“After MATLAB, CCS seems to be the killer application for TI DSP development—it’s the one thing their competition wishes it had,” Strauss adds.

“Ten years ago,” says Rodger Hosking, vice president of high-performance DSP board vendor Pentek, Inc. of Upper Saddle River, N.J., “developers were reasonably satisfied with acquiring a basic compiler and primitive debugger and writing code for a particular board. They would program a register based on the hardware manual. Today, they won’t put up with that. They are rightfully asking to program system components at a much higher level.”

“Most of our customers work with C,” Hosking says, “rather than assembly code and prefer to write higher-level language constructs, like function calls, or work with libraries.”

But getting to market these days involves more than just high-level development software or integrated development environments. The environment also must accommodate third-party software components. What is a software component? Usually it’s software, like algorithms or protocol stacks, but sometimes it’s software embedded in hardware. At Forward Concepts, says Strauss, “we call them FASICs, for ‘function- and algorithm-specific ICs,’ things like ADSL modem chips from Conexant and MPEG decoders from C-Cube Microsystems. Those are truly embedded DSP, although no one refers to them as that. They’re usually sold under the nomenclature of what they do.”

Strauss continues: “One of the hot functions available for licensing is voice over IP, sought by those who market the G.7 series of speech coders or DTMF detection and encoding and the like. Look at Surf Communications. I view them as an algorithm development house, a licensor of algorithms. It’s a growing trend.”

When developers do step into the commercial marketplace, they’ll find plenty of third-party IDEs for sale, but who knows the targets better than the DSP...
chip makers themselves? Who is in a better position to create development software?

TEXAS-STYLE SPENDING ON SOFTWARE

The answer is provided by at least one chip maker, Dallas, Texas-based Texas Instruments, Inc., which has spent upwards of $1 billion on the software side. Included in that figure is the acquisition of several software companies that have enabled the company to gain strategic capabilities with each purchase. Other chip suppliers, beginning to see the light, are groping for the right handle.

“...The software and the tools are key factors in the DSP selection process, and TI has recognized that,” says Ken Karnofsky, senior marketing manager at The MathWorks, Inc. in Natick, Mass. “They’re several steps ahead of the other DSP vendors.”

Hosking concurs. He notes that software engineers play a stronger role in hardware decision making or board selection “because project managers realize the need to preserve code investments. As a result, software is driving the choice of hardware products.”

Attention to software has created something DSP developers have long wished for and what the general-purpose microprocessor developers have had for years: a complete, open DSP software environment that embraces algorithmic standards and encourages interoperability among multiple vendors. The important message here is that the gap between system design and embedded software implementation, as identified by analysts (Figure 1), is finally being bridged. If nothing else, the gap represents a huge opportunity for new tools and capabilities. Some new standards wouldn’t hurt, either.

“Embedded software is a growth industry that will be more standards-based than in the past,” said Daya Nadamuni in an address at June’s Design Automation Conference in Los Angeles. Nadamuni, a senior analyst for EDA at market watcher Dataquest, Inc. of San Jose, Calif., believes that, “looking to the future, a lot will depend on the rapid evolution and adoption of standards.”

NO TIME TO WAIT

With technological and market events accelerating at a sizzling pace, developers can’t wait for the interminable adoption process so typical with standards writers. The quest for new open standards of interoperability has already led a cadre of third-party vendors to incorporate one DSP algorithm standard, TI’s TMS320 Algorithm Standard, part of its eXpressDSP Real-Time Software Technology, into their standard products (perhaps a standard vocoder or imaging algorithm).

“We have a corporate commitment to make all of our code eXpressDSP-compliant,” says Uwe Lymen, vice president of business development at GAO Research, Inc. of Toronto, which offers DSP telephony and speech software solutions, such as modem, fax, voice over IP (VoIP), voice over DSL, and other modules. “That will make integration much easier for our customers,” adds Frank Gao, the president.

In turn, OEM software developers, who are finding appeal in such off-the-shelf software components, or intellectual property, are also demanding a way to mix and match black boxes from various sources or ensure integration with their own applications with the least possible pain.

The solution may rest in a standard set of rules and guidelines that any algorithm producer can use to ensure the final product’s compliance with respect to any framework or intellectual property. Such a stan-
A standard can be broken down into three groups of rules and guidelines. The first group is mostly common-sense rules, such as making algorithm code reentrant and relocatable. It also would include such rules as requiring algorithms not to access hardware peripherals directly. The second group is simply more arbitrary rules, in the sense that a choice has to made; examples are whether algorithms should be provided in little-endian or big-endian format. The third group is a set of APIs that would be specified to deal with uniform memory management for all algorithms. Essentially, algorithms are relieved of the task of managing resources, making them much easier to integrate into a final system. Finally, a testing mechanism or program can be put in place to guarantee that an algorithm complies with the standard.

**COMMUNICATORS SOUND OFF**

If nothing else, a standard should smooth the way toward greater efficiency in programming, and nowhere is the call for that more vocal than in the communications industry.

“It is imperative that the communications industry become more efficient in product development,” says Mike Coffee, president of Commetrex Corporation of Norcross, Ga., which provides enabling technologies to equipment developers in the communications and digital media industries, many of which deploy DSP-based products.

‘Digital signal processors are becoming so powerful that the life span of fixed-function communications resources is rapidly coming to a close,’ says one industry executive.

“The logical steering currents are changing rather rapidly,” Coffee says. “DSPs are becoming so powerful that the life span of fixed-function communications resources is rapidly coming to a close. Today, we’re doing 48 fax channels on a single 200-MHz DSP chip. At today’s performance levels, it’s more economical to integrate multiple media processing technologies onto one chip.”

He continues: “Look at the market for multiple voice ports; it’s limited. Instead, there’s a greater demand for a board that will support any mix of voice, fax, data, video, text to speech, speech recognition, system resource hardware from the media-processing software, fostering independent competition and development in these two computer telephony value-adding layers. M100, which recently added support for the TMS320 Algorithm Standard, allows independently developed media-processing technologies to be integrated into a common media-processing hardware resource. The idea is to provide the computer telephony industry with a portable open definition of a media-processing environment that is media-neutral and hardware-independent. Any board vendor can develop and market M100-compliant hardware and so on. The problem is, few companies have mastered all of those technologies. The solution is integrating standards, such as the PC industry has had for years, so that multiple vendors can anonymously contribute to the development of industry infrastructure.”

Commetrex, which services both sides of the local loop (for example, those building enterprise communication servers and multi-access networking equipment, especially for the so-called computer telephony developers), is a prime mover in the drive to standardize.

“In the early nineties it took 20 to 30 months for a major product development—perhaps more for, say, a Class 5 switch,” Coffee says. “In today’s industry and technology environment, we don’t have the luxury of taking that long. We need new open value-adding interfaces so equipment vendors can respond rapidly to the shifts in demand.”

He sees the need to develop software components “that make it easy for someone to develop a media-processing system.”

“The problem,” he says, “is that it’s time-consuming to integrate DSP technologies into a proprietary embedded environment. An algorithm interface like T1’s algorithm standard clears the problem. The upshot is we can drop our eXpressDSP-compliant algorithms into a product like our Open Media, our implementation of the MSP Consortium’s Media Stream Processor M100 specification.”

The MSP specification is intended to separate the
products, and any media-processing vendor can develop and market compliant software products.

“The future belongs to more standards and the use of software components,” Coffee says. “The technologies are too complex, the scope too great, the demands for rapid market deployment too exigent. There’s no other way.”

Agreeing is R. Douglas Shute, president and chief executive of Delphi Communications Systems, Inc. of Maynard, Mass. “Increasing feature demand is generating the need for more efficient algorithms and newer DSPs,” he says.

**HERE COME THE OPTIMIZED ALGORITHMS**

Not surprisingly, Delphi—a leader in DSP-based software solutions for voice and data communications—is addressing those requirements with a new generation of optimized algorithms. Its core products include vocoders and software frameworks used in VoIP applications and in base station and terminal software used in cellular and PCS communications. The products tap the company’s software architecture and are fundamental building blocks in both wireless and wired (so-called “wireline”) network applications.

Whenever possible, communications designers are eschewing custom logic or ASICs in favor of software components. On top of that, they’re facing a continually escalating complexity—translated as lines of code—as both applications and hardware continue to become more complex. As a result, the traditional ways of developing and testing code are running out of steam. For those predominately using TI’s DSP chips, especially its recently released powerhouse families, CCS has become the processor-specific toolbox of choice.

Designers at Pentek, for one, use the CCS facilities to talk to the chip, extracting as much power as needed. Then they use complementary Pentek-specific software tools aimed at interface and board concerns. For instance, Pentek’s SwiftNet forms the connection software between a host workstation and the DSP hardware, no matter where it is.

The designers are not only using high-level tools, both in development and at run time, but they are exploring ways of incorporating or providing such tools to Pentek customers. “Our engineering makeup now tilts toward software designers, two to one,” Hosking says.

Designers also must pay more attention to how their software interacts with the other components of their systems. Increasingly, that means they must gravitate to a system design approach and somehow come up with realistic models for a system simulation.

MathWorks is responding to those concerns with MATLAB/Simulink and related offerings. Marketing manager Karnofsky explains, “The perennial question among developers is, How do I get my algorithm out of MATLAB [a MathWorks development tool] and into my embedded processor or DSP? Our answer is Simulink, which adds block diagrams, dynamic simulation, and code generation capability to the developer’s toolbox. It’s geared toward system-level design of real-time systems.”

Thus users can simulate a real-time algorithm and any other external environmental effects. That includes a communications channel or a device to be controlled or the analog and mixed-signal components—say, a data converter—reacting with an embedded processor.

Simulating entire systems leads to prototype code. But the ultimate goal is to produce code that an expert can write by hand, something that may be realized in perhaps a few years.

**The traditional ways of developing and testing code are running out of steam.**

As MathWorks and others work toward that future, they’re guaranteeing the present by ensuring that their products work with widely used available IDEs and real-time operating systems, such as TI’s Code Composer Studio and RTDX real-time host link. “Integrating with standard algorithm components also is part of our vision,” Karnofsky says.
Helping Match DSPs and Data Converters

By Frank Walzer

A common task in system design is to develop the hardware and software interface connecting a processor, such as a DSP, with additional peripheral devices, like a data converter. Initially, it sounds like a relatively trivial task that should be accomplished quickly. However, with the increasing complexity of DSPs and peripherals, the problem becomes more difficult. In addition, due to the large number of available data converters and different processors on the market, the number of possible combinations is huge. Thus, finding the right data converter is far from trivial, and getting the best possible interface in terms of hardware and software is even more difficult, especially in the limited time typically allotted for this subtask in the development process.

In an ideal world, the semiconductor manufacturers would make available at no charge all the necessary software, fully tested and optimized for all combinations of DSPs and analog components. Unfortunately, that's not usually the case.

Texas Instruments offers a more realistic solution in the form of a tool called Data Converter Plug-In (DCP) that can ease interface development. With this tool, you can evaluate, configure, and apply TI's advanced data converter products.

DCP is a plug-in for Code Composer Studio (CCS), TI's DSP software integrated development environment. Code Composer Studio versions are available for all major TI DSP platforms. CCS offers an application programming interface that allows the extension of functionality by installing new software on top of the standard package. Programs developed using this API are called plug-ins. DCP is an example of such a plug-in tool. It's available as a standard component of Code Composer Studio Version 1.20, or it can be downloaded free of charge at www.ti.com/sc/dcplug-in.

EVALUATION

Using the DCP, you can easily evaluate all the configurations offered by a data converter, including the configuration options of the hardware interface as well as the options for the device's functionality. Some examples are signal levels and functionality, trigger options, clock source options, analog signal references, on-chip FIFO configuration, signal processing (filters, compression, and the like), and analog channel.

The purpose of the tool is not to evaluate the analog performance of a data converter. For that task, specific hardware and software are usually required. It could, however, be useful to generate parts of the evaluation software using DCP. TI provides DSP Starter Kits (DSKs) and hardware for data converters (Evaluation Modules, or EVMs) that work together and can be used to measure the data converter's analog parameters and performance without the need to build custom hardware.

CONFIGURATION

There are two options for configuration. The first is a static configuration, which would be used, for example, after a system reset. This
configuration should initialize all the options available for a device. The second configuration option is for dynamic changes, such as sampling different channels of an ADC at different times.

DCP is able to supply a complete static configuration. You simply choose all the options needed in the configuration panel. You can verify the configuration register data on the screen without having to remember individual bit locations and activity levels. In addition, online help is accessible at any time with information on a specific configuration option. The help file has all the necessary information normally contained in the data sheet; it may be accessed through Help buttons, F1, or the Code Composer Studio context-sensitive help option.

Finally, during code generation DCP generates C #define statements that contain the static configuration data, already formatted for direct writing to a configuration register. All configuration #define statements for a project are generated within a single header file (DC_CONF.H). These statements are also used in a generated C source file. For each data converter added to the system, DCP creates a software object (a variable based on a structure). The object is initialized with the static configuration data. The function that finally writes the configuration data into the data converter uses the software object as a source. You can change the data converter configuration dynamically anytime during the program run time. All you need do is change the data in the software object and recall the configuration function.

**CODE GENERATION**

DCP is able to generate a set of C functions that are already adapted to the configuration of CCS and the selected DSP platform.

Of course, it’s very difficult to estimate exactly what software is required for a data converter application. Catalog data converters are used in hundreds of different applications with many different requirements. It’s clear, however, that every project requires a set of base functions for accessing the data converter. By simply breaking the software into several layers, you can identify those functions—they constitute the lowest level of software placed directly on top of the hardware. All of the functions are dependent on the hardware interfaces. That means that they work only with the specific types of DSP and data converter, along with their hardware connections. Thus most generated functions will work only for a given example, and the functions are therefore provided along with the recommended hardware interface. All functions are described in pure C code, making them easily understood. They can be changed if the hardware interface in the design is changed. If you need to, you can optimize and translate selected functions into assembly code.

DCP generates two more files for each data converter in addition to the file already mentioned, DC_CONF.H. One is a C header file (txxxxx_fn.h) containing all the necessary structure definitions and object and function declarations. The second is the matching C source file (txxxxx_ob.c). The “x’s” in the file name are a string defining the device family. For example, t1206_ob.c and t1206_fn.c are for the THS1206 family. The .c file con-
tains the data converter object—a variable of type TTHS1206 for the THS1206 is generated, for example. The file also contains the implementation of the layer 0 functions for this device.

**DATA CONVERTER API**

To standardize the software interface to the application software layer, we have defined software layer 1. This layer is also called Data Converter API 2.0 or DC API 2.0. It defines six functions that are available for all data converters supported by DCP:

- dc_configure() writes configuration data from software object into hardware
- dc_power() powers the data converter up or down
- dc_read() reads a single value from the data converter (either sampled analog data or status information)
- dc_write() writes values to the data converter (output data for D/A conversion or configuration data)
- dc_rblock() reads a block of sample data (usually from ADCs or codecs)
- dc_wblock() writes a block of output data (usually to DACs or codecs)

All these functions have a parameter void * pDc, which is a generic pointer to a data converter software object. Thus you can call the functions with any pointer to a TI data converter object generated by DCP. The implementation of the API functions always calls the actual function to perform the required operation through a function pointer found in the data converter object. That provides a generic interface with a device-specific implementation. For example, calling dc_configure() with a THS106 object ultimately runs the function ths1206_configure(). This function resets the device and sets up the two configuration registers as required for the THS1206.

These functions are currently used in test code, and examples are available for various data converters. Exchanging the data converter object allows you to adapt the code quickly to support new devices as they become available. If necessary, though, the layer 0 functions can be called directly when required for performance reasons.

Often there are additional functions available in layer 0 to support specific operations on a data converter type. For example, there is a ths1206_freset() function to reset the FIFO in the THS1206. As most devices don’t have a FIFO, currently there’s no need to add this function to layer 1.

The Data Converter API 2.0 is generated using two files, TIDC_API.C and TIDC_API.H. These files are automatically added to a current project along with all other created files. The Data Converter API 2.0 will be available with release 2.0 of DCP.

**EXAMPLE PROJECT**

The following illustration is a pro-
Embedded Edge

Data Converter Plug-In

The system consists of a TMS320C6211 DSK and a THS1206 EVM. The EVM can plug directly into the DSK's two expansion connectors, and the DSK can supply the power for the EVM, although for better analog performance an additional power supply is required.

The C6211 DSK is supplied with Code Composer Studio, allowing you full control over the DSP's operation. The host PC and the DSP are connected via the parallel printer port of the PC. Optionally, you can use an XDS510 emulator board and JTAG pod to connect the PC with the DSK's emulation connector. In either case, software can be downloaded to the DSK's memory and then CCS can start, stop, and analyze the DSP program.

Graphics windows set up on the PC can show memory areas on the DSK to display the sample data received from the THS1206 ADC.

Complete examples of software projects are included in the DCP download file, which may be compiled and linked without changes. The resulting .out file can be downloaded and run. In this case, software for the THS1206 device is used in combination with the C6211 DSK.

**USING THE DATA CONVERTER PLUG-IN**

DCP is activated from the CCS menu using the Tools>Data Converter Support item. To change the configuration, you add a new THS1206 device from the DCP system panel. You configure the data files, THS1206_OB.C and THS1206_FN.H, to be generated. These files contain the source code with the functions and the object for the THS1206. In addition, two files, T1206_API.H and T1206_API.C, are created that contain the layer 1 functions. All created source files are automatically added to the file list of the project.

As explained earlier, DCP can configure the THS1206 and generate the files DC_CONF.H, T1206_OB.C and T1206_FN.H. DC_CONF.H contains the static configuration:

```c
/* ADC 1 parameter data */
#define ADC1_TYPE                      THS1206
#define ADC1_CR0_VALUE           (0x00)
#define  ADC1_CR1_VALUE          (0xB8)
#define ADC1_TRIGGER_LEVEL (8)
#define  ADC1_NR_CHANNEL (1)
#define ADC1_SAMPLE_FREQ     (1000)        /* in kHz */
#define  ADC1_SHIFT (0)
```

ADC1_CR0_VALUE and ADC1_CR1_VALUE are the encoded values to be written directly into the THS1206's configuration registers. They also are used to initialize a THS1206 software object in layer 0 to use the ADC are shown in Listing 2.

**USING THE GENERATED SOFTWARE**

As explained earlier, you can use either layer 0 software functions directly or the layer 1 equivalent. Layer 1 software is more generic, so the data converter may be exchanged easily, as shown in Listing 3.

Switching to the Files tab and pressing the Write Files button generates the necessary files. Static configuration data is written to the file DC_CONF.H, causing two new

---

**Listing 3: Example program**

```c
extern THS1206 Ths1206_1;      /* the data converter object */

/* Init function example: */
init_disk(),     /* initialize the DSK */
in_time(DSP_FREQ, ADC1_SAMPLE_FREQ);

/* Init function example: */
dc_configure(&Ths1206_1);     /* reset the THS1206 internal FIFO */
IER = 0x00000012;             /* enable INT4 and NMI */
CSR = CSR | 0x01;              /* int global enable, ready for interrupts */
start_timer();                 /* start timer 0 - generates sample clock */
"{
unsigned volatile int "T1206_CTRL = 0x00;
"while(1) {"asm("idle");
}``

you can use an XDS510 emulator board and JTAG pod to connect the PC with the DSK's emulation connector. In either case, software can be downloaded to the DSK's memory and then CCS can start, stop, and analyze the DSP program. Graphics windows set up on the PC can show memory areas on the DSK to display the sample data received from the THS1206 ADC.
below. This routine needs to read
the sample value from the ADC’s
internal FIFO according to the value
of the selected FIFO trigger level
(ADC1_TRIGGER_LEVEL). If, for
example, the trigger level is set to 8,
the ISR must read eight values.

```
interrupt void data_av_int(void)
{
    int i = 0;
    static int cnt=0;
    /* Read as many values as defined via
       the trigger level */
    /* Make sure the C optimizer does not
       create additional */
    /* loads during software pipelining */
    for (i=0;i<ADC1_TRIGGER_LEVEL;i++)
        ad_buffer[cnt++] =
                dc_read(&Ths1206_1);
        if (cnt==SAMPLE_BLOCK_SZ) cnt = 0;
}
```

This routine is excellent for an
on-chip DMA controller. The DMA
should be set up to read a block of
data triggered by the ADC’s
DATA_AV signal. A software exam-
ple demonstrating the setup is avail-
able as well.

This ISR implementation isn’t
optimized for performance, as the
goal of DCP isn’t to provide optimized
code. Usually you want to optimize
performance or other parameters
later, if necessary. Also, changes are
often required to adapt to different
hardware interfaces. That’s why all
the code is supplied as C source: it’s
easy to understand, change, and port
to new DSP platforms.

Frank Walzer (f-walzer@ti.com) is a field
application engineer for data converter
products within the Advanced Analog
Products group of Texas Instruments,
Inc.’s subsidiary in Freising, Germany,
responsible for the development of TI’s
data converter plug-in software. Pre-
viously he was responsible for European
customer support for boundary-scan
products and DSPs.
The good news: You just received your target DSP board from the assembly shop. The bad news: Now you have to debug it.

Where do you begin? Start with a little common sense and let a JTAG emulator take you the rest of the way. That duo will allow you to bring up the target platform and give you the confidence you need to allow software development to begin on the target.

JTAG is a standard method of boundary-scan testing established by the European-based Joint Test Action Group and the IEEE, which named it IEEE Standard 1149.1. The standard defines a serial interface for the I/O of a chip to be used for the boundary-scan testing. Match appropriate hardware (emulator) and DSP development software to the serial interface and—voilà!—you’ve got yourself a debugging tool.

Used systematically, the JTAG emulator is a valuable tool to help bring up and test new DSP target platforms. Think of it as your window into the hardware system, a way to save a lot of time when the hardware arrives. By methodically starting from a known good position and then migrating into the unknown, you build confidence in yourself and the test system. This time-proven method, backed with good tools, will lead to a successful DSP-based product.

Before you hook up interfaces and applying power, a few simple steps can save you a lot of grief. Inspect the board for missing or damaged components. Use an ohmmeter to check for shorts between ground and all the voltages present on the board. Remember: “Still smoking” is not a positive state.

A typical test system configuration consists of a host PC used for software development, the JTAG emulator, and the DSP target (Figure 1). The hardware designer must bring out the JTAG signals from the DSP to a standardized connector.

Figure 1. A JTAG emulator opens a window into the workings of a DSP target. The emulator pod connects between the target and a host computer.
located close to the processor. The connector interfaces to the actual JTAG emulator (pod). In the case of Texas Instruments’ DSPs, the JTAG header is a 2-by-7 double-row header with one pin clipped for use as a key.

There are many advantages to this approach, as well as a few negatives. On the plus side, the approach is relatively low in cost because a special bond-out chip isn’t required—you’re always working with the actual DSP—and you’re able to use the same emulator across multiple TI DSP platforms and also with multiple processors on board, both homogenous (same type) and heterogeneous (different families). Also, you can use a JTAG emulator with flash utilities to program flash memory. On top of those assets, count the fact that no target monitor is needed (that is, the approach is nonintrusive), the target can be in a chassis or enclosure, and everything is independent of processor speed and packaging.

Among the disadvantages, count a limited number of hardware breakpoints (code and data), no trace capability (though it is coming on future devices), and the fact that the target hardware must be somewhat functional before you can use it.

Getting started with a JTAG emulator requires a little up-front design preparation. First, make sure the target DSP board accommodates the 2-by-7 JTAG header. The mounting can be either vertical or at right angles, depending on the debugging environment, either stand-alone or in a chassis. To design this interface, refer to TI’s JTAG documentation or replicate the JTAG logic used on an evaluation module or DSP starter kit board. The JTAG header should be as close to the DSP as possible. The single JTAG header can work with one DSP or multiple DSPs when the JTAG scan path is daisy-chained among the devices.

Next, install the debugger software (Code Composer Studio) and appropriate emulator drivers on the host development system. Then, before bringing up the new board, verify the Code Composer Studio and emulator installation by bringing up the debugging tool chain with a known working DSP target. This procedure establishes that two of the three items in the chain are functional. If so, the target is ready for its “send-up.”

The first thing to do when debugging with a JTAG emulator is to verify the host computer-emulator-target board connection (Figure 2)—that's your first confidence level. With power removed from the target DSP board and emulator, connect the tail of the JTAG emulator to the JTAG header on the DSP board. Make sure the emulator is connected to its host computer’s interface (parallel port, USB, ISA, or PCI board). Now apply power to the emulator and target. Next, launch the Code Composer Studio debugger. If it doesn’t come up, make sure that the JTAG header is wired correctly and that the processor has a clock or is not being held in reset, and see whether the debugger and drivers are installed correctly.

If the debugger comes up, it’s time to develop a hardware debugging game plan. One proven strategy is to work from a known good position when investigating and proving other sections of the system. Work on-chip first, then move off-chip—that’s where the JTAG emulator excels.

By using the Code Composer Studio debugger, you can verify that the chip is functional—your second confidence level. Remember, the chip was tested at the factory before you received it. Starting with the on-chip hardware, let Code Composer Studio reset the DSP and verify the default register values, read and write to all internal registers, verify memory ranges and on-chip RAM (by using pattern memory fill and read commands), and finally, verify that the I/O operates correctly.

Besides verifying the registers and memory, you can also use the debugger to verify I/O pin operation. Because many output lines are mapped into registers or memory, you can stimulate individual lines with the debugger—writing to these locations with patterns is much like memory pattern writing—and verify the levels with a scope. This is the first action to look at outside the DSP.

To test the input lines, stimulate
them with a zero voltage level or VCC and verify the values in the memory location or register by reading the values with the debugger. Once you're satisfied that the DSP is functional, it's time to look at the off-chip portion of the system—the third confidence level. Obviously, all systems possess off-chip logic, even if the software runs on-chip, and you've got to get signals on and off the DSP.

Once you're satisfied that the DSP is functional, it's time to look at the off-chip portion of the system— the third confidence level. Obviously, all systems possess off-chip logic, even if the software runs on-chip, and you've got to get signals on and off the DSP.

Now you're moving into new territory, where you can redeploy the debugger to verify off-chip RAM by using pattern memory fill and read commands, verify memory ranges, and verify the correct operation of the I/O.

Testing off-chip memory is a bit more challenging. You must come up with a plan to check for problems with both the address and data lines. The lines can be shorted to zero, shorted to VCC, shorted together, or even open or not connected. Right off, you can take several cursory steps to verify the data lines: Fill memory first with all 0s and verify, then all 1s and verify, finally all 0s and verify. Alternatively, you can opt to fill memory with 0xaaa and verify, then 0x5555 and verify.

Checking address lines also is a building-block approach. Because each ascending address line doubles the range of memory, it's best to start checking with a very small block (say, 16 words). First, fill all the available memory with 0s to clean the slate. Next, fill the memory block with a data pattern (0xffff) and look through the rest of the memory to see whether or not the same data shows up somewhere else. If it does, you've got a short. To test all of the available memory, double the block size of the data pattern and look through the rest of the memory until the usable memory size is reached.

Off chip I/O typically consists of memory-mapped peripherals or peripherals in the I/O space. The technique used here is exactly the same as that in testing the on-chip I/O: Stimulate the output lines by writing to them with the debugger and verifying the levels with a scope. To verify input lines, place a level on the lines and read out with the debugger.

OK, you see that this procedure works. Do you now have to go through it all again for your next two prototypes? Yes, but there is a better way to do it. General Extension Language, or GEL, an interpretive scripting language similar to C, lets you extend Code Composer Studio's usefulness. Basically, the idea is to put your desired interpretive debugger commands into GEL files. Listing 1 shows a sample GEL file that you can use to fill a block of memory with all 1s. Just load the file with Code Composer Studio, run the file, and quickly inspect the memory (with the debugger) for errors. A file like this can be replicated, renamed, and edited to fill any memory with

Now you're moving into new territory, where you can redeploy the debugger to verify off-chip RAM by using pattern memory fill and read commands, verify memory ranges, and verify the correct operation of the I/O.

Testing off-chip memory is a bit more challenging. You must come up with a plan to check for problems with both the address and data lines. The lines can be shorted to zero, shorted to VCC, shorted together, or even open or not connected. Right off, you can take several cursory steps to verify the data lines: Fill memory first with all 0s and verify, then all 1s and verify, finally all 0s and verify. Alternatively, you can opt to fill memory with 0xaaa and verify, then 0x5555 and verify.

Checking address lines also is a building-block approach. Because each ascending address line doubles the range of memory, it's best to start checking with a very small block (say, 16 words). First, fill all the available memory with 0s to clean the slate. Next, fill the memory block with a data pattern (0xffff) and look through the rest of the memory to see whether or not the same data shows up somewhere else. If it does, you've got a short. To test all of the available memory, double the block size of the data pattern and look through the rest of the memory until the usable memory size is reached.

Off chip I/O typically consists of memory-mapped peripherals or peripherals in the I/O space. The technique used here is exactly the same as that in testing the on-chip I/O: Stimulate the output lines by writing to them with the debugger and verifying the levels with a scope. To verify input lines, place a level on the lines and read out with the debugger.

OK, you see that this procedure works. Do you now have to go through it all again for your next two prototypes? Yes, but there is a better way to do it. General Extension Language, or GEL, an interpretive scripting language similar to C, lets you extend Code Composer Studio's usefulness. Basically, the idea is to put your desired interpretive debugger commands into GEL files. Listing 1 shows a sample GEL file that you can use to fill a block of memory with all 1s. Just load the file with Code Composer Studio, run the file, and quickly inspect the memory (with the debugger) for errors. A file like this can be replicated, renamed, and edited to fill any memory with

Now you’re moving into new territory, where you can redeploy the debugger to verify off-chip RAM by using pattern memory fill and read commands, verify memory ranges, and verify the correct operation of the I/O.

Testing off-chip memory is a bit more challenging. You must come up with a plan to check for problems with both the address and data lines. The lines can be shorted to zero, shorted to VCC, shorted together, or even open or not connected. Right off, you can take several cursory steps to verify the data lines: Fill memory first with all 0s and verify, then all 1s and verify, finally all 0s and verify. Alternatively, you can opt to fill memory with 0xaaa and verify, then 0x5555 and verify.

Checking address lines also is a building-block approach. Because each ascending address line doubles the range of memory, it’s best to start checking with a very small block (say, 16 words). First, fill all the available memory with 0s to clean the slate. Next, fill the memory block with a data pattern (0xffff) and look through the rest of the memory to see whether or not the same data shows up somewhere else. If it does, you’ve got a short. To test all of the available memory, double the block size of the data pattern and look through the rest of the memory until the usable memory size is reached.
can test and verify your system with enough confidence for the team to focus on its coding efforts. But wait. How are you going to test and debug those 20 preproduction boards coming in? (The fun doesn’t stop, does it?) Building on your confidence about the target system, all you need do is embellish your game plan with provisions for automating the testing process. Not only does doing that get you out of the testing business, but it also compresses the time needed to test and document the procedure and allows reuse when automated test fixtures are employed for large product volumes.

Because you know the system is stable enough to write software, you can develop a software test system that enlists the Code Composer Studio development suite, GEL files, and minimal additional hardware. And you’re not merely testing hardware—you’re testing hardware at full speed. Any lurking hardware timing issues will be exposed.

As you go, write as much software as possible in C. Doing that helps with documentation, and the code is a lot easier to maintain. Because this won’t be your last DSP project, you’ll want to reuse as much code as possible, and that means writing the test system in modules so that various tests can be linked in quickly.

To get going, look at what you did manually, then automate it: Note that the JTAG connection will be verified just by coming up and loading the test program. Start by writing memory tests (patterns) for both code and data spaces. Then, use loop-back connectors to allow software to generate stimulus to verify both input and output lines.

Finally, use the GEL files to reset the system, load the test program, set breakpoints for error or pass conditions, and run the code. A sample GEL file to accomplish those tasks is shown in Listing 2.

The GEL file shown should start running and stop executing by reaching one of the error breakpoints or the pass breakpoint. The operator watching the debugger screen can see which label the program stopped at. Many of the tests can be verified by not installing the loop-back connectors.

Of course, you’ll have to debug the test program by running it in the nonautomated mode—that is, adding in test modules one at a time to build confidence in the test software as you develop it.

Again, your strategy in writing test software is to start from a known good position and work your way into the suspect areas. For example, load your test code into on-chip memory, test whatever you can, and then go on to test the off-chip memory and I/O.

Where can you put this test code to work? You can use it, with a JTAG emulator and manually mounted loop-back connectors, on the test bench, and in limited production. In high-volume test situations, those using a spring pin fixture, the loop-back connectors can be hardwired on the fixture, along with the JTAG emulator.

The procedure just outlined is great for RAM-based systems. What about flash memory-based systems? The technique’s very similar. “Flash” the test code into the memory (up-front step), use the GEL file to load only the symbols, put an error/pass code into a known memory location, and let the program run for a given amount of time. Examine the specified memory location for a pass/fail condition, and you’re through.

Mike Strain is a cofounder of Spectrum Digital, Inc. (www.spectrumdigital.com), a Stafford, Texas-based producer of JTAG emulators and evaluation modules for TI DSPs. He’s worked with embedded processors for more than 25 years.
Modeling and simulation have become essential to advanced DSP system development. One of the goals is to facilitate the development process by hiding sophistication and complexity and presenting a simple interface to the user.

There are several core requirements: tools for algorithm development, a diverse set of computing "blocks," a display system that helps the developer see how the computations are behaving, and a simulation system that can handle a broad variety of devices that are used in modern electronic products.

An echo cancellation unit for a wireless telephone serves as an instructive illustration of the use of such a tool set.

Echoes are a byproduct of the march of technology. As phones shrink, speakers and microphones nudge closer together. Space for bulkheads and expensive transducers goes for a premium. Invariably, smaller phones generate bigger echoes. In fact, the tendency to produce echoes is roughly inversely proportional to some power of the distance between the phone's speaker and microphone.

The problem is worse in mobile phones because the ambient noise sources are more complex. Take automobiles, in which the engine, radio, air conditioner, and heating unit, among other things, all contribute to the ambient noise. Echo cancellation techniques are clearly required.

In practice, cancellation techniques estimate the echo from the transmitting (far-end) phone and subtract it from the signal at the earpiece of the receiving (near-end).
phone. Attenuating the echo in tele-
phone handsets is not an easy task.
Like other acoustic processing tech-
niques, designers must model not
only the communications device but
also the environment. The exact
model depends on the placement of
the speaker and the microphone, as
well as on the acoustics of the
environment.

HOW ECHOES ARE FORMED
An echo results when the speech
signal enters not only the micro-
phone in a handset, but the earpiece
as well. Unfortunately, the electron-
ics in the handset also amplifies the
speech signal leaking into the ear-
piece and adds delay. Thus a feed-
back loop is created between the
microphone and the earpiece within
the handset.

A typical acoustic echo cancella-
tion system can be modeled as
shown in Figure 1. The figure por-
trays two telephones, but the one on
the right (the “terminal model,” or
near-end phone) is shown in more
detail. The far-end phone may be
assumed to be identical to the near-
end one, although this assumption
isn’t absolutely necessary.

The echo is actually established,
as it were, in the near-end handset.
At the near end, speech and ambient
noise enter the microphone and the
earpiece, creating an echo. To cancel
the echo, you have to identify it, esti-
mate it quantitatively, and then sub-
tract it from the signal that’s sent to
the far-end phone. Otherwise, the
signal heard by the near-end phone
will contain speech corrupted by the
echo.

Echo characteristics—loudness,
duration, frequency, degree of ran-
domness—vary dynamically with
each phone call. The characteristics
of phone lines, the path connecting
different users, and the telephone
equipment itself all are variables.

Therefore no a priori information is
available to mitigate the echoes.
From the perspective of the near-
end handset, the only information
available is the far-end speech as it’s
heard at the earpiece and as it’s
heard at the microphone. The differ-
ence between these two signals is
caused by the environment of the
near-end enclosure.

Obviously, echo cancellation
techniques must be adaptive to
model this dynamic situation.
Practice has shown that a single dig-
ital signal processor is adequate to
handle the echo cancellation job, as
well as the other tasks required of it
in a modern phone.

Accurately modeling an echo can-
celation implies modeling the echo
cancellation algorithm and the sys-

The first step is to model the echo.
The signal that enters the earpiece
in the near-end enclosure of Figure 1
is the far-end speech, which we
assume to have echoes. The earpiece
can be represented by an infinite
impulse response (IIR) filter because
actual earpieces exhibit nonlinear
phase characteristics.
The next block in the signal path
is an echo response block, where an
echo transfer function filter is
applied. This block is an IIR filter
that models the reverberation
caued by the placement of the ear-
piece relative to the microphone.
Finally, the microphone is modeled
by yet another IIR filter; again, this
type of filter is chosen because it’s
able to model the nonlinear phase
characteristics of the phone equip-
ment. From the microphone, the
signal travels into the near-end echo
cancellation unit.

The echo signal is now complete-
ly modeled. It’s ready for processing
in the near-end echo canceler.

THE LMS ADAPTIVE FILTER
The main processing agent of the
near-end echo cancellation unit is
the LMS adaptive filter block, a
built-in component of the DSP
Blockset, which is a library of DSP
algorithms for use with Simulink
(see “A Tool Set for Modeling and
Simulating Electronic Equipment”).

Adaptive filters are a class of filters
that can change (adapt) their coeffi-
cients as the filter is operating.
There are two inputs: the far-end
talk, x(k), and an error signal, e(k),
which is the difference between the
adaptive filter output and the far-end
talk signal. The purpose of the filter
is to make the error in the noise sig-
nal, over time, as close to zero energy as possible. That is, the algorithm tries to minimize the mean square value of the error signal (the difference between the echo signal and the previous filter output).

In this system, the adaptive filter can remove the component of this difference that is correlated with the echo signal, and the result is the original (clean) signal. Inside the adaptive finite impulse response (FIR) subsystem, the key components of the adaptive FIR filter are a buffer block and an inner product block. The buffer block generates a vector of the last $n$ points of the signal $x(k)$ at each time step (where $n$ is the buffer size and $n-1$ is the buffer overlap). The inner product is formed between the output of the buffer and the filter coefficients. The filter coefficients, or tap weights, are updated at each sample according to:

$$W(k+1) = W(k) + \mu e(k) x(k) / \|x(k)\|^2/n$$

where $x(k)$ is a length $n$ vector of the last $n$ input samples; $W(k)$ is a vector of tap weights at time $k$, the current sample instant; $e(k)$ is the error at time $k$ and $\mu$ is the adaptation parameter. The factor in the denominator of the update term makes this “normalized” (with respect to 1) LMS algorithm. Figure 2 shows the subsystem implementing this function.

The output of the LMS adaptive filter is subtracted from the echo signal by the sum block, which is on the lower left side of the echo canceler in Figure 1. The difference between the echo signal and the output of the LMS adaptive filter is the far-end listen signal. The purpose of the echo cancellation unit is to eliminate echoes in the far-end listen signal.

You can gauge the effectiveness of
A TOOL SET FOR MODELING AND SIMULATING ELECTRONIC EQUIPMENT

Integrated development environments for embedded systems have been available for years. For a broad class of applications that don’t involve DSPs, IDEs generally include an RTOS, compilers, debuggers, low-level simulators, and process monitors. However, for DSP-based applications, a decidedly different class of IDE is needed, in which the mathematical nature of the process is modeled and simulated.

What’s needed for modeling and algorithm development is an integrated technical computing environment that combines numerical computation, advanced graphics and visualization, and a high-level programming language. What you want is a procedural, command-line-oriented system that contains functions for data analysis, such as data manipulation and reduction, interpolation, scaling, extraction of sections of data, and smoothing and filtering. What’s needed for simulation is a hierarchical block diagram design and simulation tool that can be used with purely digital systems, with analog and mixed-signal systems, and with event-driven systems. Furthermore, the modeling and simulation tools, along with the target hardware, should be integrated so they can be routinely used in tandem.

MathWorks’ MATLAB and Simulink, along with the MathWorks Developer’s Kit for Texas Instruments DSP, represent such tools; together, they provide an end-to-end plug-in development environment for DSP-based devices.

MATLAB is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language (called M). It is a procedural, command-line-oriented system that contains functions for data analysis, such as data manipulation and reduction, interpolation, scaling, extracting sections of data, and smoothing and filtering. The Signal Processing Toolbox is a collection of MATLAB routines that are used for applications involving signal processing. Some of its core functions involve the creation, editing, and analysis of low-pass, high-pass, bandpass, and bandstop FIR and IIR digital filters.

Simulink is a hierarchical block diagram design and simulation tool. Designers can use it with purely digital systems, with analog and mixed-signal systems, and with event-driven systems to visualize signals and codevelop software with existing C code.

Simulink and MATLAB are integrated; they’re routinely used in tandem. From Simulink, users can access MATLAB built-in functions (such as for visualization and analysis) and their own programs, created in the M language (called M files).

The MathWorks DSP Blockset is a library of blocks for use with Simulink; it contains an extensive and far-reaching set of algorithms for filter design (including adaptive and multirate filters), transforms, and spectral estimation.

Simulink works with another core MathWorks product called Real-Time Workshop, which produces ANSI-standard C code that can be compiled by TI’s Code Composer Studio, a complete system for developing DSP-based applications. CCS even includes facilities for managing the administrative aspects of development.

Simulink plugs into many of the system aspects of CCS. For example, one component of the MathWorks Developer’s Kit for Texas Instruments DSP can integrate blocks with custom code (through Simulink), automatically build a CCS project, and target a C6701 EVM board.

TI and third parties offer a broad range of boards, referred to variously as evaluation, prototyping or development boards, to test and validate their designs on real hardware. Once the algorithm has been tested in real time on a development board, designers can measure the results and perform a final evaluation of the algorithm.
You can gauge your design by examining two parameters: the adaptive filtering convergence time and the echo return loss enhancement (ERLE), which refers to the ratio of echo power before and after echo cancellation. ERLE provides a figure of merit for determining the effectiveness of the echo cancellation process; it assumes that there's always a certain amount of loss incurred by echo cancellation (echo power) and then shows the rate of improvement after echo cancellation. In the graph of the echo cancellation system's performance (Figure 3), the blue line is the power of the far-end talk; the green line represents the power of the far-end listen signal, which needs to be as low as possible to reduce the echo; and the red line shows the ERLE, which indicates how much of the echo we removed. Note that the magnitude of ERLE increases over time and then stabilizes, indicating that any echoes have been canceled. There is a delay in the echo cancellation system because the system must be operational for some (small) amount of time before it becomes effective. The time interval is the adaptive filtering convergence time.

To determine the rate of convergence, we can plot the ERLE over time. This plot will show us the amount of time required for the filter taps to converge. The taps converge at about 2.25 seconds, with an echo return enhancement loss of almost 30 dB (Figure 4).

Anne Mascarin (annem@mathworks.com) is the DSP Market Segment Manager at The MathWorks, Inc. in Natick, Mass. Previously, she worked at DSP Software Engineering and Analog Devices. Her professional interests include speech and image processing.
Can I allocate from different heap sections using DSP/BIOS?

Yes, DSP/BIOS II provides an API, MEM_alloc(), that allows programmers to allocate memory from any MEM segment defined to contain a heap in the configuration tool. The standard C run-time library function malloc() will allocate memory from the memory segment specified through the configuration tool in the field Segment for malloc() / free() if the No Dynamic Memory Heaps check box isn't checked.

What's the relationship between CIO's printf and SYS_printf?

SYS_printf is a limited version of sprintf. It prints messages to the DSP/BIOS trace buffer by default. It's used by SYS_abort for writing short error messages before halting execution in an error condition. LOG_printf is a better choice for user messages when an emulator's available, since it's more efficient. More information can be found in the DSP/BIOS User's Guide, in Chapter 9, LOG and SYS sections; section 3.4, “Instrumentation APIs”; and section 5.2, “System Services.”

Using Code Composer, can I debug a target board containing two DSPs of different platforms in a single JTAG scan path?

In this case, you'll need to launch two separate instances of Code Composer to support each of the DSP platforms. Two separate directories should be created for Code Composer files; the set-up utility will need to be run in each of these directories, and the DSP not being targeted in one instance of Code Composer should be bypassed. Do the same for the remaining DSP. Bypassing DSPs and scan chain devices is discussed in Chapter 1, “Setting Up Code Composer,” of Code Composer User's Guide.

How can I tell the silicon revision number of a DSP chip?

In many cases, this is as simple as looking at the info stamped on the part. If the DSP is packaged at a certain site, then the code “Cxx” will be on one of the lines. The “xx” is the silicon revision of the device. The register that Code Composer or Code Composer Studio reads out of the device contains a number that's similar to, but not the same as, the silicon revision. Instead, it's the silicon revision of the CPU itself. The device can go through a change and get a new silicon revision, but unless the CPU core itself changes, the number in the CSR register stays the same.

What's the maximum number of DSPs the XDS510 can debug or drive at the same time?

There are two issues: XDS510 board driving capability and Parallel Debugger Manager (PDM) limitation. Theoretically you can debug any number of DSPs with the XDS510 for TI’s JTAG-based devices (C5x, C4x, etc.). However, the XDS510 board can drive 16 to 32 processors without buffering.

PDM is able to open a maximum of approximately 82 processors under OS/2 or Solaris. This limitation comes from the operating system (an OS typically has a maximum number of windows that it can open), since PDM always opens a window for each processor.

How do I set breakpoints in ROM?

The regular breakpoints you use with the emulator (BA command or clicking the mouse left button in the line) are software breakpoints. They're implemented by replacing the instruction with a SWI instruction (emulation interrupt). Because the emulator needs to “replace” (write) the SWI instruction in memory, software breakpoints are possible only in RAM.

The C4x and C5x DSPs have an on-chip analysis module to solve this problem. The module allows you to break in a set of events; one of them may be a program access into a specific location. That gives you the same functionality as a regular breakpoint. The TI debugger offers a pull-down menu to program the DSP analysis modulo.
Internet Stack for Real-Time Kernel

An embedded Internet stack, Precise/RTCS, is available now for DSP/BIOS II, a scalable, extensible real-time kernel from Texas Instruments. By enabling network connectivity over Ethernet and other network interfaces, Precise/RTCS extends the scope of DSPs to such applications as IP phones, wireless networking, xDSL modems, and voice and fax over IP. A single-project, royalty-free license starts at $17,500; full source code is provided. Precise Software Technologies, Inc., Ottawa; (613) 596-2251, www.psti.com

Tool Speeds DSP Code Writing

The eXpressDSP Component Wizard tool helps to write algorithms according to Texas Instruments’ TMS320 DSP Algorithm Standard. Step-by-step screens help automatically produce a source code sequence and associated project file that can be compiled into a custom, vendor-specific, eXpressDSP-compliant algorithm. The eXpressDSP Component Wizard, an eXressDSP-compliant plug-in, is available for immediate delivery and sells for $295. Hyperception, Inc., Dallas, Texas; (214) 343-8525, www.hyperception.com

Fast A/D Converter Module

The HEGD12, an 8-channel, 16-bit A/D converter module, features a 20-MHz input sampling rate and 8x down conversion. The unit is targeted at wireless products using the Heron multiprocessor DSP systems based on the TMS320C6000 platform—in particular electronic intelligence, sonar, and high-speed instrumentation—and up to four modules can be combined on a PCI or VME carrier. It is fully supported by Hunt Engineering’s eXpressDSP-compliant framework and its recently announced Linux package. The starting price is $1,475; available now. Hunt Engineering, Somerset, U.K.; +44 (0) 1283 819991, www.hunteng.co.uk

DSP Library Offers Telecom Functions

Version 4.0 of the SigLib ANSI-C source code DSP library accommodates many fundamental telecommunications operations, among them QAM, FSK, QPSK, and associated functions, like scrambling and descrambling. The highly portable library, which runs on the TMS320C3x, TMS320C4x, and TMS320C6000, also allows code reuse without modification on new DSP architectures and integrates with other libraries and RTOSs. The price is $350, and availability is immediate. Numerix Ltd., Leicestershire, U.K.; +44 (0) 7050 803996, www.numerix-dsp.com

Tool Aids Code Development

An enhancement tool, Development Assistant for C can be used independently or in parallel with Code Composer or Code Composer Studio, as well as with individual TMS320 code generation tools. Among the functions it adds are an editor with both structured and nonstructured flowcharts, symbol browser, call and type hierarchy graphs, a make file generator, software metrics, an interface to version control systems and debuggers, project and message managers, and a static code analyzer. Available now, Development Assistant for C sells for $660. RistancASE GmbH, Wallisellen, Switzerland; +41-1-883-35-70, www.ristancase.ch

RTOS Runs on C6000

The C Executive RTOS is available for Texas Instruments’ TMS320C6000 DSP platform. The initial release has been qualified on the TMS320C6211 on TI’s DSK C6211 evaluation board and supports C62x and C67x processors. The C Executive development package sells for $2,500; it’s available now. JMI Software Systems, Inc., San Jose, Calif.; (603) 750-0170, www.jmi.com
Application Builder Churns on TI DSPs

Visual Application Builder (VAB), component-based DSP design software, is available for Texas Instruments' DSPs, including members of the TMS320C2000, C3x, C5000, and C6000 platforms. Combined with a TI-based DSP board, VAB, an eXpressDSP-compliant plug-in, makes a suitable platform for building systems for instrumentation, modeling and simulation, image processing, and other numeric-intensive tasks. VAB for TI DSPs sells for $1,495 and is available now. Hyperception, Inc., Dallas, Texas; (214) 343-8525, www.hyperception.com

Fast D/A Module Ties to DSP Systems

The HEGD14 DAC module is a 33-MHz, 14-bit, multichannel D/A unit that extends the interface options available for Heron multiprocessor TMS320C6000-based DSP systems. The module accepts high-bandwidth DMA data streams from the DSP hardware. It measures 4x2.5 in. and fits directly into the module slots for Heron carrier boards. Heron DSP systems, manufactured by Hunt Engineering, use a modular architecture based on standard PCI carrier boards, allowing different resources to be combined to form fully integrated real-time signal-processing systems. Available now, the module sells for $2,150. Traquair Data Systems, Inc., Ithaca, N.Y.; (607) 266-6000, www.traquair.com
What’s So Important About an Algorithm Standard?

By Bob Frankel

No matter the DSP application, everybody wants to write more code in less time. Like it or not, today’s intense competitive pressure sets the pace.

Some believe the answer is to work harder; others, smarter. From where I sit, the best solution is to avoid work altogether. How? Take advantage of the work of others.

To put it more formally, if the embedded industry is to continue its explosive growth, it must reuse previously written DSP software. That’s an accepted fact with certain software, such as a kernel, which could be written in-house if the time to do so weren’t better spent elsewhere.

MAKE OR BUY?

When it comes to the core DSP routines, those that manipulate the signal data, few people understand the algorithms or how to develop them. Clearly, there’s no make-or-buy decision here—most of us would look for an off-the-shelf V90 modem or echo cancellation or speech recognition software.

Here’s the problem: Those algorithms can’t simply drop into a customer’s program. The third-party algorithm suppliers, often small independents, end up doing an incredible amount of custom engineering for the end-equipment manufacturers that buy their stuff. You may as well stamp the box “Some Assembly Required.”

Really, the problem arises because of the trend toward multifunction systems. No one supplier can provide all of the required algorithms, so the developer has to worry about bolting diverse pieces together and making them work together. It can take months to integrate third-party algorithms with other pieces of software.

Those are months that OEMs could better spend getting their products to market—and algorithm writers could use to produce better and more exciting code, instead of worrying about their customers’ code.

What’s the solution? An algorithm standard.

ALGORITHM STANDARD

What’s an algorithm standard? It’s a system accepted by major algorithm vendors to ensure that everything works together without “blood, sweat, and tears.” It’s a document that lays out the rules of the road—that is, it specifies guidelines and common-sense software engineering practices. Such a document brings order to an otherwise random or chaotic situation.

Just including naming conventions, for example, avoids clashes between software from two different vendors. Going farther are conventions that say who can touch an interrupt and who can’t; how algorithms ask for memory, the most precious DSP resource; and so on.

Such an algorithm standard doesn’t have to be radical or untested or revolutionary—it can reflect the “state of the practice,” making it easy for everyone to follow. Third-party developers who face alternative solutions no longer have to make an arbitrary choice; they can follow the standard. It all boils down to: Write once, deploy widely.

With an algorithm standard in place, OEMs can go shopping without worrying about conflicts, especially when the product carries a compliance sticker. The algorithms in the “display case” all have been coded, developed, and packaged according to a set of baseline specifications. Software components will have come of age.

Believe it: An algorithm standard is the catalyst for the continued growth of the embedded software industry.

Bob Frankel is a TI Fellow and the Chief Software Strategist in Texas Instruments, Inc.’s Software Development Systems group in Santa Barbara, Calif.
IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:
Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Copyright © 2002, Texas Instruments Incorporated