DSP: From Ideas to Implementation

DSP: From Ideas to Implementation

A variety of tools simplifies the trip from abstract DSP models to signal-processing hardware.

by Jon Titus, Senior Technical Editor

More and more designs include circuits and software that process sampled analog signals in the digital domain. The techniques have vastly improved since Texas Instruments introduced its TMS32010 digital-signal processor IC in the mid 1980's. Engineers no longer hand code algorithms and debug via rudimentary hardware. To provide an overview of how engineers now can approach a DSP project, I talked with people at software providers The MathWorks, Mentor Graphics, and Synplicity. On the hardware-and-software side, I spoke with people at Texas Instruments and Analog Devices.

The MathWorks

"To begin with, we need to understand what type of product a designer has in mind," said Ken Karnofsky, marketing director of signal processing and communications at the MathWorks. "Will designers use analog as well as digital components? Will they implement the design on ICs or on board-level products? Will they implement algorithms in hardware, software, or both?" The MathWorks' MATLAB and Simulink software facilitate model-based design so engineers can create an algorithm or a system at an abstract level before they constrain their algorithm with implementation details.

"Say you need a digital filter," said Karnofsky. "Traditionally, someone would code the algorithm in C, VHDL or Verilog. However, implementing the design is an iterative process, and every change requires extra simulations to re-verify filter performance." If you do this with an implementation-independent model in MATLAB or Simulink, you can quickly adjust the algorithm and run new simulations and what-if analyses. "In addition, you cannot test an algorithm in isolation," added Karnofsky. "The algorithm interacts with upstream and downstream processes, so you must know how algorithm behavior affects overall system metrics. You don't have access to tools for that type of analysis when you code algorithms in low-level languages."

Designers should aim to refine a model within a single development environment. "You use an abstract model not only to weigh tradeoffs and 'what if' possibilities, but to make your end application more 'portable,'" stressed Karnofsky. "You might implement something on a DSP chip today and move it to an ARM processor in a few months. In such a case, you can easily go back to the model rather than try to wade through thousands of lines of implementation-specific C or HDL code." When iterations are required, engineers can modify the original model without a significant recoding effort.

To simplify modeling tasks, MATLAB and Simulink give developers pre-built and tested Blocksets that offer video, imaging, audio, down-converter, and other functions. Developers can use these blocks both to explore tradeoffs such as fixed- and floating-point algorithm implementations and to rapidly construct a test environment to verify component behavior in a system context.

After engineers refine MATLAB algorithms or Simulink models, they can automatically generate code to test how an application would "fit" into particular chip architecture and verify that the implementation produces the same result and behavior. "Because you know the model is functionally correct, the code you get out is, in a sense, correct by construction," noted Karnofsky. And, developers can specify coding styles and interfaces for the code MATLAB generates so that code works with external programs that provide a graphical user interface or control a piece of equipment.

The MathWorks also offers engineers a line of Embedded IDE Link tools that connect MATLAB and Simulink models with integrated development environments from Analog Devices, Texas Instruments, Green Hills and other vendors, as well as EDA Simulator Link tools that connect to hardware simulators from Cadence, Mentor Graphics, and Synopsys. These links allow engineers to reuse the system-level model to verify the software or hardware implementation.

Mentor Graphics

"Many of the engineers we work with develop new algorithms or implement algorithms in new ways," said Stuart Clubb, Catapult C Synthesis technical marketing engineer at Mentor Graphics. "They create models in C or C++ because they need to refine them and reuse the code. Often, the engineers realize their algorithms will not run on a DSP chip, so they have started to move to an FPGA or an ASIC. So they need to convert C/C++ code into hardware, which is what Catapult C does."

Unfortunately, hardware and software engineers don't always work the same way. "Software engineers who creates algorithms think in a sequential manner because they expect their code to run on a processor," explained Clubb. "A hardware engineer will think more about parallel operations and the physical resources an algorithm will use." Clubb uses an imaging edge-detection algorithm as an example. The algorithm 'moves' a 3x3 matrix of coefficients across the image pixels, multiplies each image pixel by its corresponding coefficient and sums the results to produce one value. Essentially nine pixels in, one value out. Then you move on and read the next series of pixels. From a hardware perspective, the algorithm creates a memory bandwidth problem because the algorithm reads the same pixel as many as nine times. So, a hardware designer would implement line buffers with a window that would extract nine pixel values simultaneously and enable the multiplications and addition in one clock cycle. "But the designer must described that arrangement in the system's architectural behavior," said Clubb.

Mentor Graphics' Catapult C tool takes the engineers' C++ code and--among other things--unrolls and pipelines loops to create parallel hardware and a timed schedule for an algorithm based on available hardware and memory bandwidth. "For simplicity, think of a 4-tap FIR filter," said Clubb. "You can design it with one multiplier-accumulator [MAC] and perform operations sequentially in four clock cycles--a possible bottleneck. Or you could use four multipliers in parallel to perform all operations in one clock cycle."

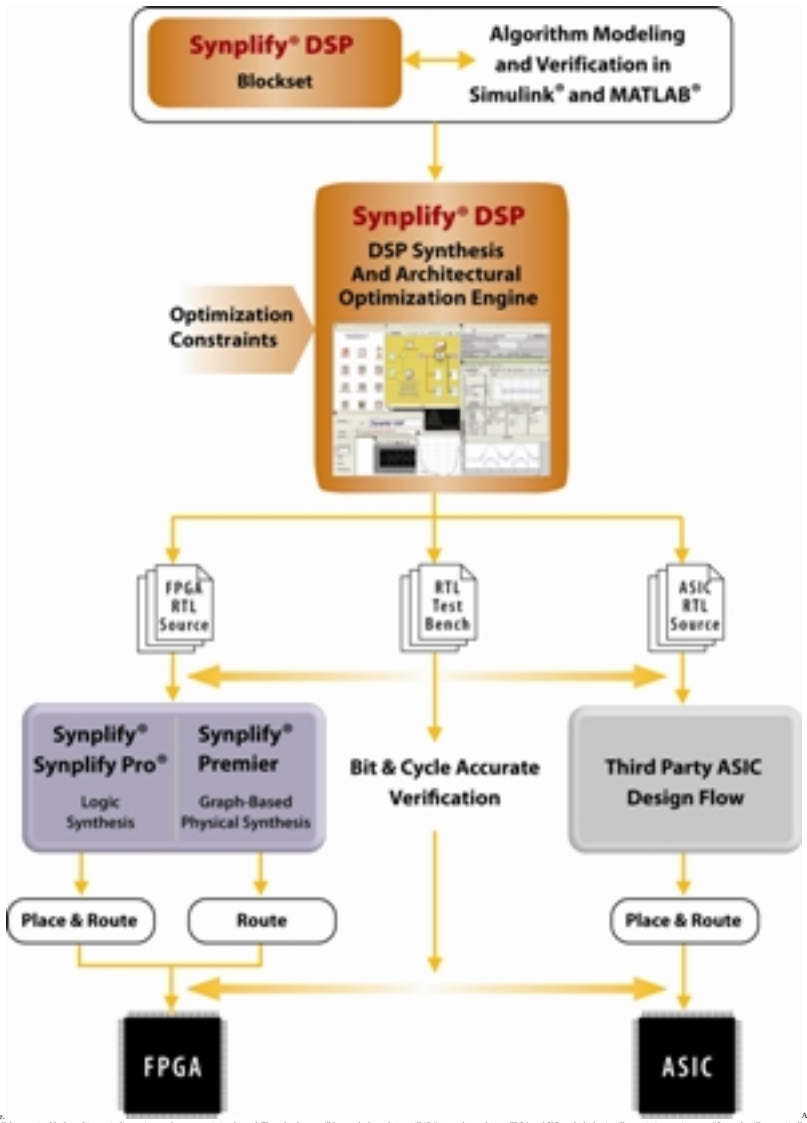
The Catapult C tools show you where bottlenecks occur. Suppose the eight FIR-filter coefficients come from a synchronous single-port memory so you can read only one coefficient at a time. That creates a bottleneck because the hardware can do only one MAC operation per clock cycle. "Often Catapult C finds memory-bandwidth problems in a design rather than bottlenecks at adders, multipliers, and other elements," said Clubb.

To reduce this type of memory bottleneck, you could change a 16-bit x 64-value memory to a 32-bit x 32-value memory and store two 16-bit coefficients at each address. "When you optimize bandwidth at one place, it can catch up with you somewhere else," cautioned Clubb. "If you need to write only one 16-bit coefficient now you must use a 32-bit read-modify-write operation that takes extra time."

Synplicity

When developers have an idea for a device that will employ DSP, they can create an abstract algorithm within Synplicity's Synplify DSP software. The software provides a library of Blocksets for the MathWorks' Simulink and lets developers code portions of an algorithm in MATLAB. "MATLAB generally provides the highest level of abstraction for algorithm prototyping and exploration," said Chris Eddington, director of DSP marketing at Synplicity. "Developers can synthesize everything in our library into architecturally optimized RTL code. Our Blocksets support high-precision fixed-point math and a floating-point override simulation mode so developers don't get bogged down with quantization, underflow and overflow as they start to create algorithms. Developers can then explore and tune fixed-point settings to achieve the algorithm behavior they need."

Making these quantization decisions early in the design process increases a design team's productivity. After developers create an arithmetic data path, often they must modify and explore their "quantization" choices to ensure an algorithm performs properly. (Most hardware-only designs on FPGAs and ASICs rely mainly on fixed-point math.) For some processing tasks, 10- or 12-bit fixed-point operations may suffice, but others may need a wider dynamic range. Tradeoffs also exist between sample rate and quantization so a modeling library must support sample-rate specification. "When you start to quantize standard algorithms they can become unstable," noted Eddington. "So designers must understand the tradeoffs involved with moving from floating-point to fixed-point math. When developers work with an abstract model, they don't have to create one for each type of math, which simplifies testing."



Next, developers can use the Synplify DSP synthesis engine to generate optimized RTL code. According to Eddington, users can choose if they want a serialized architecture, if they want a use a high-frequency clock and share many resources, or if they want to use parallel processing blocks and insert pipeline registers to better meet timing demand. Then, developers will know whether a design will "fit" into a chosen device--FPGA or ASIC--and whether it will meet timing requirements. "Our tools will automatically add registers as needed at the architecture level so the RTL will meet timing on the target device," said Eddington. "Then you can take the RTL code through standard logic synthesis and place-and-route steps to implement your design in hardware. The key is the automatic architecture optimization for the target device."

"Our software handles many details automatically and report results to engineers," said Eddington. "The report could indicate the tools chose an XYZ configuration for a FIR-filter based on design constraint and automatically figured out how many pipeline registers to use based on a sample rates entered in the model."

Eddington explained designers may have to try different design parameters to implement circuits in a given FPGA. "If a circuit goes beyond the capabilities of a device, you can try different 'folding' values that tradeoff serial vs. parallel implementations of operations. Then you can determine which folding value best meets timing requirements and also minimizes device area."

Texas Instruments

At the start of a project, engineers should have a block diagram of the data flow and the functions they need. Then engineers can look for DSP chips that meet their requirements. "We help them consider system-level costs versus device costs," said Leon Adams, DSP strategic marketing manager at Texas Instruments. "Many times functional blocks on a DSP chip reduce the need for separate external components that add cost. Engineers should look at overall costs, not just the cost of a DSP chip."

"Then engineers should think about how much flexibility they need," said Adams. If a design will process video information, for example, choose a DSP chip that includes video-processing hardware accelerators. If you don't plan to process video information, that chip may limit design flexibility. "If you plan to change functions, codecs or algorithms, fully programmable DSP chips make more sense," said Adams.

According to Adams, TI has crafted its DSP development tools to ensure engineers have products available for every stage of their product's design. "One engineer may need reference software, while another might need specific algorithm libraries," noted Adams. "Engineers can get codecs from a third party, license other IP from another source and combine them with our tools and libraries. You don't have to build everything on your own to create a good product." TI promotes its eXpressDSP Algorithm Interoperability Standard (xDAIS) as a way for third-party suppliers to ensure their software will operate with other software in a modular fashion.

A DSP-chip vendor's software-development tools come to the fore when engineers move to a project's code-and-build stage. The TI Code Composer Studio handles multiple projects and it provides an efficient C compiler and optimization capabilities. Users can trade off code size for code performance, for example. "Our layered software structure lets you use only the pieces of DSP code you need, perhaps a DSP/BIOS," said Adams. "Users also can take our drivers and optimize them for their application or they can remove unneeded functions." Then TI adds a higher layer of software that connects with operating systems such as Linux, Windows CE or others.

"Developers work their way up from the BIOS level to what we call Codec Engine, a framework that supports concurrent execution of multiple channels and codecs," explained Adams. "It provides video-imaging, speech and audio plug-ins."

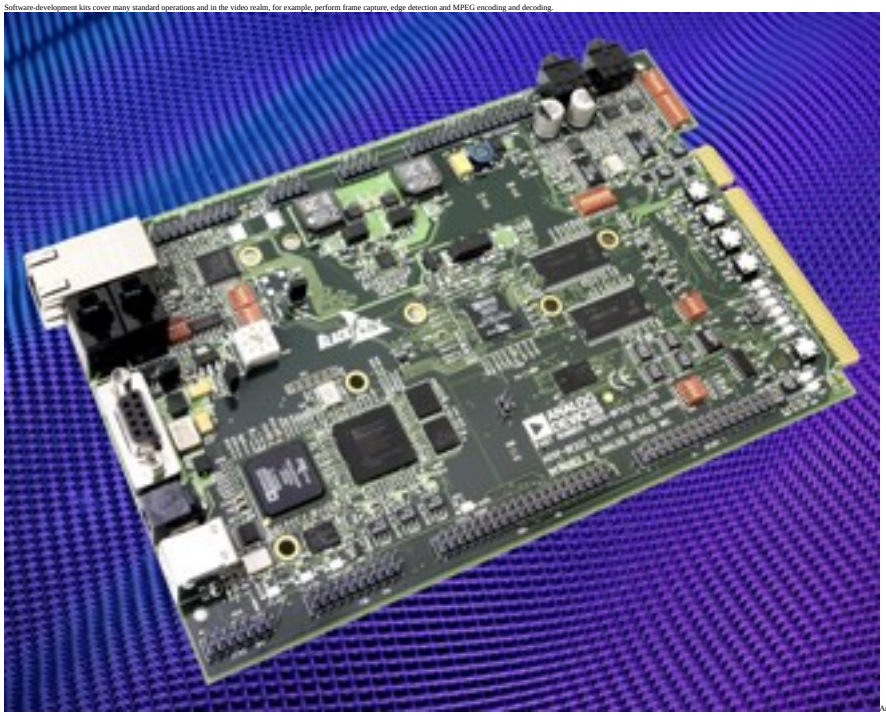
To enhance testing and debugging, TI offers a real-time trace capability in its tools. The trace triggers on specific events and will log information for later analysis. "It lets developers set up their program so if it 'breaks,' it will stop so they can examine data in log files to determine what happened," explained Adams. "Perhaps an interrupt occurred randomly, or noise on a signal line caused an unexpected action, or a bus overload.

In some cases, a DSP or DSP system-on-a-chip IC won't do everything you want. "Then you can turn to an FPGA to complement a DSP chip," said Adams. "An FPGA can implement a special video multiplexer or I/O devices not yet available in a DSP chip. Or you might need some extra image processing in a signal chain."

Analog Devices

"A lot of customers want more than DSP chips," said David Katz, Blackfin applications manager at Analog Devices. "They want vendors to take them most of the way to their final product. So we offer hardware-evaluation boards, software tools, debug and development software, device drivers and several operating systems for our parts. Our designer community includes engineers who want to continue using operating-system XYZ, who use open-source tools and drivers, or who want to concentrate efforts on product features."

Software-development kits cover many standard operations and in the video realm, for example, perform frame capture, edge detection and MPEG encoding and decoding.

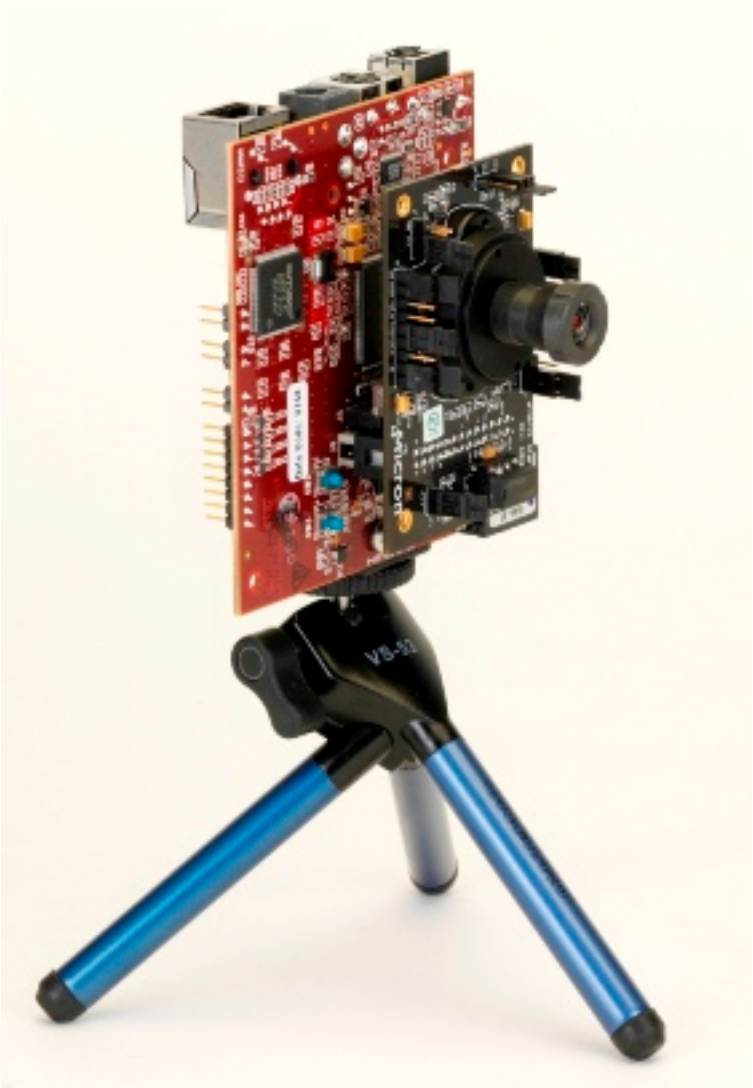


Audio routines include MP3, Dolby Digital, Ogg Vorbis and Ogg Speex codecs. "We offer those standard algorithms as an immediate jumping-off point for customers who know exactly what they want," said Katz.

Analog Devices ships system-service and device-driver software with its VisualDSP++ tool suite. The software gives developers a common application programming interface (API) through which they can control clocking, power, interrupts, DMA operations and I/O ports. If you use a Blackfin ADSP-BF537, for instance, and upgrade to an ADSP-BF561 dual-core processor, you can continue to use the same API and drivers. That approach helps customers avoid implementation details and lets them focus on their algorithms.

In addition to VisualDSP++ tools that handle native C/C++ code, engineers have other development options as well. They can use National Instruments' LabVIEW Embedded Module for Blackfin Processors, for example, that provides a graphical approach to embedded design. As another example, Link for VisualDSP++ from The MathWorks integrates MATLAB and Simulink with VisualDSP++. Both products create entire applications or individual algorithms that handle control-intensive and compute-intensive processes.

On the hardware side, a Multimedia Starter Kit provides two versions of a development board set that has a camera interface, LCD interface and video encode/decode devices as well as a library of algorithms and applications. "A few customers will buy many of our development boards and put them directly in their end product," noted Katz.



Even with all the tools and help available, engineers may not dwell long enough on high-level questions such as "How do we increase future performance and maintain hardware compatibility?" "They choose a DSP chip that runs out of MIPS after they have loaded all their code into it," said Katz. "They should look for a DSP-chip family that includes single-core low-power devices all the way up to multi-core high-performance ICs. That gives them a get-out-of-jail-free card." Engineers also could design with pin-compatible DSP chips so they can start with a high-performance device to get prototypes running. Then they can optimize code and possibly use a less-expensive compatible chip without having to do a board spin before they start to manufacture their product.

For further reading

"Designing High Performance DSP Hardware using Catapult C Synthesis and the Altera Accelerated Libraries," Mentor Graphics. 2007.

www.mentor.com/techpapers/fulfillment/upload/mentorpaper_36558.pdf [1]

Blackfin Online Learning and Development (BOLD) Video Tutorials:

http://my.analog.com/onlinetraining/Static/BOLDList.html [2]

Katz, David J. and Rick Gentile, "Embedded Media Processing," Newnes Press, Burlington, MA. 2006. ISBN: 978-0-7506-7912-1.

McCloud, Shawn, "Using a Catapult C-Based flow to Speed Implementation and Increase Flexibility," Mentor Graphics. 2003. www.mentor.com/techpapers/fulfillment/upload/mentorpaper_22618.pdf [3]

The open-source community supports a Blackfin-uClinux port at: http://blackfin.uclinux.org [4].

Source URL (retrieved on *05/18/2013 - 6:59pm*):
http://www.ecnmag.com/articles/2008/05/dsp-ideas-implementation?qt-video_of_the_day=0

Links:
[1] http://www.mentor.com/techpapers/fulfillment/upload/mentorpaper_36558.pdf
[2] http://my.analog.com/onlinetraining/Static/BOLDList.html
[3] http://www.mentor.com/techpapers/fulfillment/upload/mentorpaper_22618.pdf
[4] http://blackfin.uclinux.org