Analog Dialogue

A forum for the exchange of circuits, systems, and software for real-world signal processing

Volume 39, Number 1, 2005

In This Issue

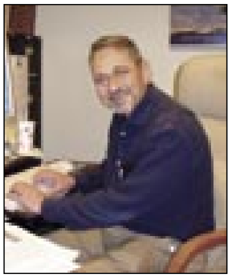
INNOVATION • PERFORMANCE • EXCELLENCE
40 years ADI 1965 2005

ANALOG DEVICES

www.analog.com/analogdialogue

Editors' Notes

40 YEARS OF REAL WORLD SIGNAL PROCESSING



Forty years ago, Ray Stata and Matt Lorber opened the doors of Analog Devices for business, offering a line of high-performance operational amplifiers. We've survived and prospered beyond their fondest expectations, and are still rarin' to go. In celebration of that anniversary, *Analog Dialogue's* four print installments will each be devoted to one of our major technologies. We start with digital signal processing (DSP).

ANALOG SIGNAL PROCESSING GOES DIGITAL

In 1986, a new—and apparently unlikely—contender entered the young field of digital-signal-processor manufacturing—then dominated by TI, the colossus of "Speak & Spell,"—with a single-chip DSP, our ADSP-2100. As we celebrate our 40th year in the business of components for signal processing, it seems worthwhile to reproduce here our editorial comments that accompanied the introduction of the first Analog Devices DSP in these pages (*Analog Dialogue* 20-2, 1986):

> "Microprocessor?" we hear you ask. "Isn't it a bit unseemly for a nice 'Analog' IC company to be designing a microprocessor? (What could be more *digital*?)"
>
> Good question.
>
> Our objective has always been to design and manufacture cost-effective components that are key elements of the signal path for processing real-world (i.e., *analog*) data and for which performance is maximized and errors minimized.
>
> The signal path? *Real-world data* almost always starts out as *analog* (i.e., parallel, non-numeric) variables, which are measured by sensors that provide *analog* electrical signals—voltage and current. The signals must be accurately and speedily amplified, conditioned (almost always in parallel) and converted to digital for processing. Once in digital form, they must be processed rapidly. Often they again wind up as analog signals.
>
> Key elements of the signal path may include preamplifiers, analog signal processors, data converters—to and from digital—and, when the signal is in digital form, a digital processor. Inadequacy in any one of the key elements—amplifier, analog processor, data converter, or microprocessor—can cause poor performance of the overall system.
>
> Obstacles in the signal path include noise, drift, nonlinearity, and measurement lag at the analog stages, similar obstacles in conversion—and throughput delays in digital processing, often because of the lack of parallelism in von Neumann architectures.
>
> Throughout our history, ADI's role in the signal path has been to initiate new products (or product lines) when dissatisfied with the performance and cost-effectiveness of what's available (which is often limited to user-assembled kludges, when nothing else is available). At this point in time, we (and our worthy competitors) have virtually eliminated the user-assembled amplifier, signal conditioner, and data converter, by designing and marketing families of high-performance, cost-effective products.
>
> We have always been dissatisfied with the cost, power dissipation, and slow throughput in the digital domain; this concern led to our pioneering development of CMOS multipliers and other digital signal-processing ICs (note that because we were already familiar with analog multipliers, digital multipliers became just another analog signal-processing tool). Note also our commitment to *signal processing*—not payroll, desktop publishing, or order-handling products).
>
> And our dissatisfaction with insufficient throughput in DSP processors led to the design of the ADSP-2100, which stresses the use of that *analog* characteristic, parallelism, to minimize instruction cycles, whether in processing, data transfer, or interrupt handling. It's neat! We invite you to read about it.

Since that time, such names as SHARC®, TigerSHARC®, Blackfin®, EZ Kit, and VisualDSP++® have become household words, as they remove barriers whenever DSPs are considered.

Dan Sheingold [dan.sheingold@analog.com]

FROM NUMBER CRUNCHING TO MULTIMEDIA

In the early days of digital signal processing, the ADSP-2100 single-chip microprocessor was typically used for applications that required high-speed numeric processing. Integrating a 16-bit arithmetic-logic unit (ALU), 16-bit multiplier-accumulator (MAC), 16-bit shifter, two data-address generators, and a program sequencer, it used external memory for program and data storage. Operating at 8 MHz, it dissipated 600 mW. In a single clock cycle it could: generate the next program address; fetch the next instruction; perform one or two data moves; update one or two data address pointers; and perform a computational operation.

Over the intervening twenty years, digital signal processors have gotten smaller, faster, less expensive, more powerful, and more efficient—and they integrate up to 24 Mbits of on-chip memory. Even more important, perhaps, are the host of peripherals that can be found on modern embedded processors. The ADSP-BF537 Blackfin processor, for example, includes an IEEE 802.3-compliant 10/100 Ethernet medium access controller, Controller Area Network (CAN) 2.0B interface, parallel peripheral interface (PPI) supporting ITU-R 656 video data formats, and dual-channel, full-duplex synchronous ports (SPORT) supporting eight stereo I^2S channels. The ADSP-21367 SHARC processor's digital audio interface (DAI) includes an S/PDIF digital audio receiver/transmitter, 8-channel sample-rate converter, sixteen pulse-width modulators, four PLL clock generators, eight serial ports, and ROM-based audio decoder and post-processor algorithms. The ADSP-TS201 TigerSHARC processor includes an 8-Gbps 64-bit external port, 14-channel direct memory-access (DMA) controller, and four 8-Gbps bidirectional link ports. Together they provide unparalleled interface capabilities without the use of any additional external glue logic.

Processing power and peripherals have created opportunities for digital signal processors in diverse applications—including professional audio mixing consoles, *always-on* cell-phone coverage, home-theater surround sound, fingerprint recognition, network music players, wireless video, satellite radio, and 3D motion tracking. Some of these are described below. Details about these applications and many more can be found at http://www.analog.com/processors/news/customerstories.

The TigerSHARC processor is the only processor capable of implementing a software-defined digital baseband for 3G base stations, allowing the same platform to be easily adapted for use in multiple regions—and to be easily upgraded to support new capabilities. The TigerSHARC processor is also the first to implement an all-software physical layer for IEEE 802.16 WiMAX broadband wireless modems. Its best-in-class I/O bandwidth and scalable architecture allow OEMs to differentiate their products through advanced techniques, such as smart antennas using space-time coding and adaptive beam-forming.

The 32-bit floating-point SHARC processor has the necessary speed and efficiency to handle the complex post-processing algorithms required to deliver 6.1 discrete channels of surround sound from any audio material, allowing listeners to take full advantage of their home-theater speaker systems, even when listening to VHS tapes, FM radio broadcasts, or stereo music CDs. The SHARC processor's digital audio interface, large memory array, and VisualDSP++ graphical system design and development environment combine to allow manufacturers to base multiple products with various I/O requirements on a single hardware design, fully leveraging their design time and development costs.

Blackfin processors provide both control functions and multimedia processing capabilities, enabling diversity receivers to operate in harsh weather and low light conditions. Providing fast information transfer, these receivers allow soldiers, police officers, and firefighters in the field to exchange audio, video, and data from sources such as cameras, microphones, and global-positioning systems (GPS)—increasing personnel safety in environments that are subject to high levels of interference. The low power consumption and dynamic power management inherent in Blackfin processors is crucial for their successful use in compact, portable, battery-powered equipment.

Scott Wayne [scott.wayne@analog.com]

Analog Dialogue

www.analog.com/analogdialogue **dialogue.editor@analog.com**

Analog Dialogue is the free technical magazine of Analog Devices, Inc., published continuously for 39 years—starting in 1967. It discusses products, applications, technology, and techniques for analog, digital, and mixed-signal processing. It is currently published in two editions—*online*, monthly at the above URL, and quarterly *in print*, as periodic retrospective collections of articles that have appeared online. In addition to technical articles, the online edition has timely announcements, linking to data sheets of newly released and pre-release products, and "Potpourri"—a universe of links to important and rapidly proliferating sources of relevant information and activity on the Analog Devices website and elsewhere. The *Analog Dialogue* site is, in effect, a "high-pass-filtered" point of entry to the **www.analog.com** site—the virtual world of *Analog Devices*. In addition to all its current information, the *Analog Dialogue* site has archives with all recent editions, starting from Volume 29, Number 2 (1995), plus three special anniversary issues, containing useful articles extracted from earlier editions, going all the way back to Volume 1, Number 1.

If you wish to subscribe to—or receive copies of—the print edition, please go to **www.analog.com/analogdialogue** and click on **<subscribe>**. Your comments are always welcome; please send messages to **dialogue.editor@analog.com** or to these individuals: *Dan Sheingold*, Editor **[dan.sheingold@analog.com]** or *Scott Wayne*, Managing Editor and Publisher **[scott.wayne@analog.com].**

ISSN 0161-3626 ©Analog Devices, Inc. 2005

A Smart Modem for Robust Wireless Data Transmission Over ISM Bands (433 MHz, 868 MHz, and 902 MHz)

By Patrick Butler [patrick.butler@analog.com]
Austin Harney [austin.harney@analog.com]

In the last few years, radio-frequency technology has advanced by leaps and bounds, resulting in a phenomenal number of new wireless applications. Most of these applications—Bluetooth®,[1] WLAN 802.11b,[2] and cordless telephones, for example—are appearing alongside the microwave oven in the license-free UHF band at 2.4 GHz. Because of the heavy traffic in the 2.4-GHz band, and its associated co-existence issues, interest has increased in the ISM (industrial, scientific, medical) UHF bands—available at the lower frequencies of 868 MHz and 433 MHz in Europe, and 902 MHz to 928 MHz in the United States.

Unlike at 2.4 GHz, however, there is no common global standard for the lower-UHF bands; this means that a manufacturer's system would have to be adaptable to each region's regulations. However, this burden has been eased considerably by the introduction of flexible ISM-band transceivers, such as the ADF7020,[3] which allow operation from 433 MHz to 960 MHz.

Unfortunately, one cannot entirely eliminate the problem of interference and co-existence by simply switching to these lower-UHF bands. As might be expected, there are plenty of legacy systems already operating in these bands. In wireless systems, data will be corrupted if an interferer collides with the wanted signal—resulting in an insufficient signal-to-noise ratio (SNR) at the receiver. A traditional way of dealing with this problem is to use some sort of error-detection technique, e.g., *cyclic redundancy checking* (CRC). CRC can detect this corruption to a certain extent and trigger the retransmission of erroneous packets (this is usually called *automatic repeat request*, ARQ), but at the cost of considerable delay and loss of performance in real-time applications.

This need to retransmit corrupted packets is not particularly onerous for a low-throughput system—one that sends a burst of data from a remote sensor once every few minutes, for example. But it does become a problem for applications such as wireless audio or video transmission, with their higher data rates, since the latency introduced by ARQ might be unacceptable. It also introduces problems in industrial process-control and telemetry systems, which must maintain throughput in a noisy environment without the need for many retransmissions. Such longer associated transmission times also increase the overall system power consumption.

A powerful solution to this dilemma lies in the use of *forward error-correction* (FEC) techniques, able to detect and correct errors over a large enough number of bits to compensate for partial packet loss and ensure service quality. A low-cost, yet powerful, processor such as the Blackfin®[4] ADSP-BF531[5] can be used to implement intensive error-correction techniques requiring *millions of instructions per second* (MIPS)—convolutional coding with bit-scrambling and interleaving, for example—to deliver a data rate of over 100 kbps with a transmission error rate of less than 10^{-6}.

When used in conjunction with the ADF7020 ISM-band transceiver IC, with its typical range of several hundred meters (line of sight), this approach provides a robust solution for designers wanting to replace their current wire-line solutions without compromising quality of service. Thanks to its 400-MIPS (*million instruction-per-second*) and 800-MMACS (*million multiply-accumulate-per-second*) capabilities, the ADSP-BF531 can also accommodate protocols to support various wireless configurations and topologies, including point-to-point, multi-point, and broadcast, as well as sophisticated encryption and source coding and decoding algorithms such as Motion JPEG (MJPEG).

Figure 1 is a detailed circuit diagram of a wireless digital modem built around the ADF7020 ISM-band transceiver and its companion controller, the ADSP-BF531. The two main chips share the same power supply voltage (2.3 V<V_{CC}<3.6 V), and they are

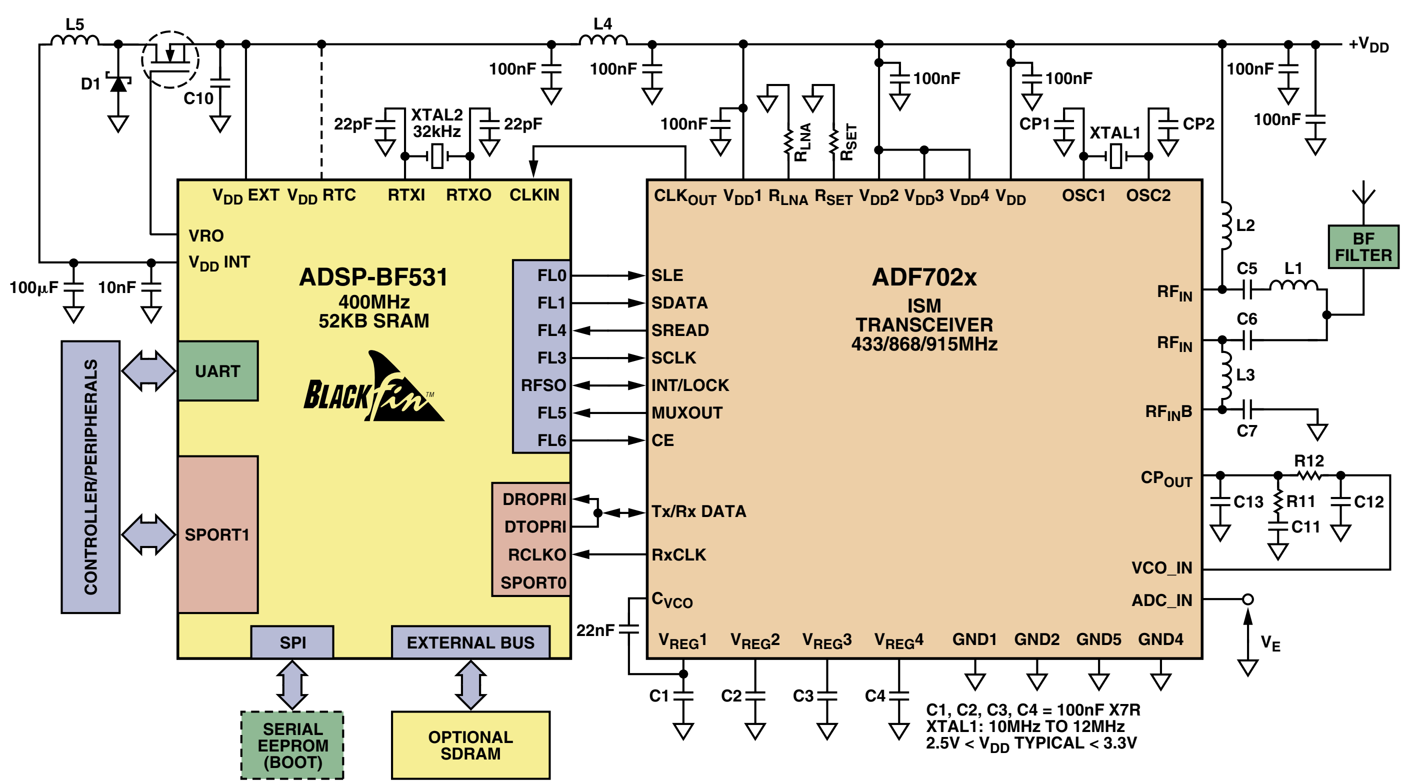


Figure 1. Circuit diagram of the modem.

directly connected for *control* operations, using the ADSP-BF531 flags (digital I/Os) and *transmit/receive* operations, using one of the serial synchronous ports (SPORT0).

Data will be transmitted to—or received from—the modem, either asynchronously over the UART or synchronously with the remaining SPORT.

A Versatile Transceiver

The ADF7020 is a complete monolithic radio transceiver built using 0.25-µm CMOS technology. It is capable of operating in the 433-MHz and 868-MHz European ISM bands (ETSI EN300 220-1 standard),[6] and the North American 902-to-928-MHz band—covered by FCC Part 15 regulations.[7] Requiring few external components and offering a high degree of flexibility, it allows the user to configure the part for specific applications. For example, there is a choice among different modulation schemes, such as FSK, GFSK, ASK, and OOK. The user can also trade off between sensitivity and selectivity—a useful approach for systems that have tough linearity requirements. The maximum data rate for the ADF7020 is 200 kbps; its sister part, the ADF7025,[8] has an even greater data rate: 384 kbps.

Like most recent ISM-band transceivers, the ADF7020 utilizes a fractional-N phase-locked-loop (PLL) synthesizer, which allows the selection of the channels at 433 MHz, plus any channel between 868 MHz and 928 MHz, with a resolution better than 1 kHz. This frequency agility allows the ADF7020 to be used in frequency-hopping systems—as specified in the US FCC Part 15 regulations—but it is also possible to operate on a single channel in the US band if the output power is below –1.5 dBm.

The high-resolution fractional-N synthesizer also forms part of a novel *automatic frequency-control* (AFC) loop, which compensates for incoming frequency errors and allows lower-tolerance, less-expensive, crystals to be used. The block diagram of the ADF7020 is shown in Figure 2. The PLL loop filter components can be determined with the help of the ADIsimPLL[9] simulation software, available on the Analog Devices website.

Forward Error-Correction with the Blackfin Processor

While the use of a really high-performance processor in conjunction with a radio is common in digital cellular systems, it might at first glance seem inappropriate for meeting the goal of a low-cost digital modem. Implementing FEC operations at several hundred kilobits per second, however, requires computationally intensive digital signal-processing power comparable to that provided by the Blackfin ADSP-BF531. While a standard 8051 or ARM-based microcontroller, for example, can adequately handle the user interface, protocol stack, RF transceiver supervision, and power sequencing, it would not have the computation "horsepower" required for the FEC scheme. In addition to implementing the control functions, the computing power and real-time capabilities of the ADSP-BF531 allow it to: increase the effective channel data rate, reduce communication latency, compensate for channel propagation variations to maintain link quality, and ensure communication security.

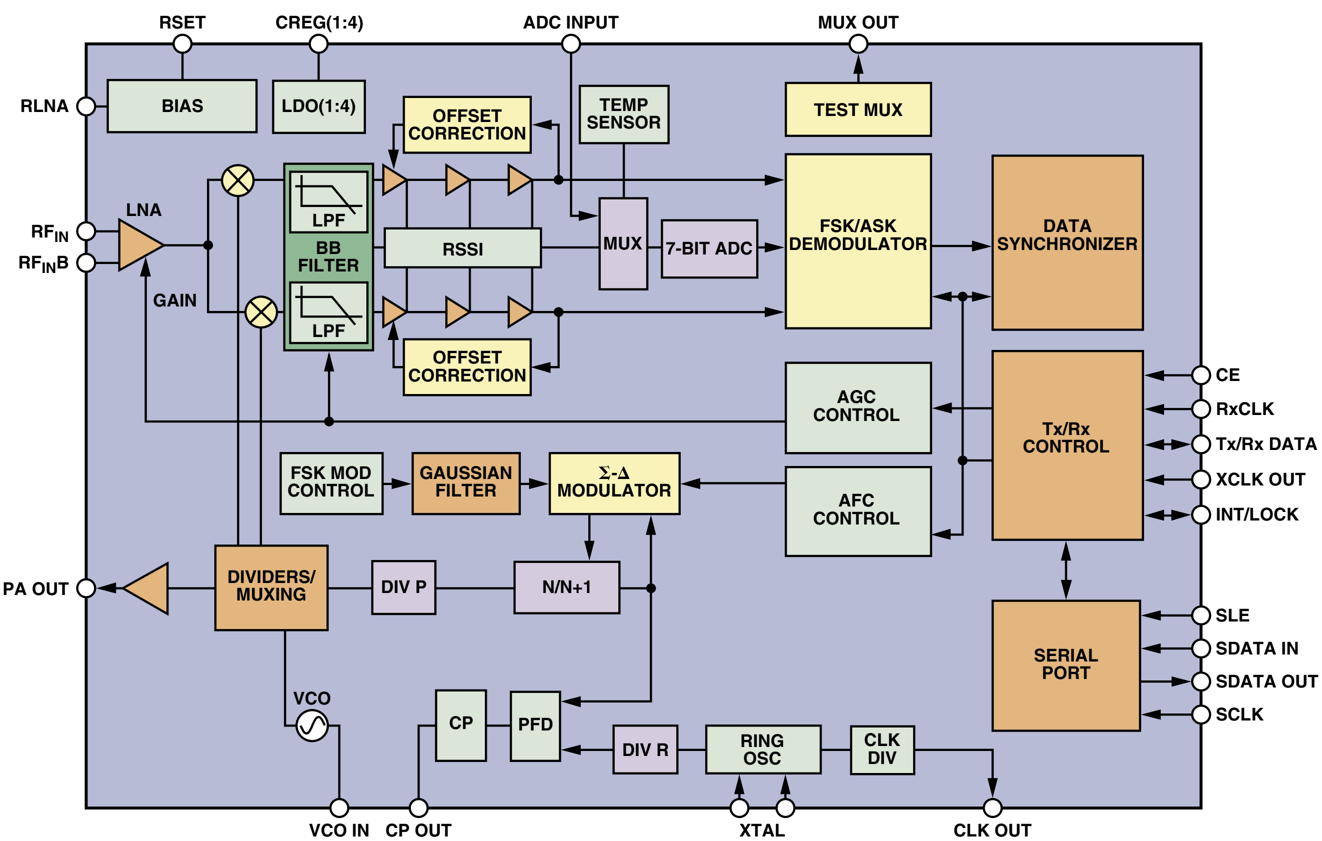


Figure 2. Functional block diagram of the ADF7020.

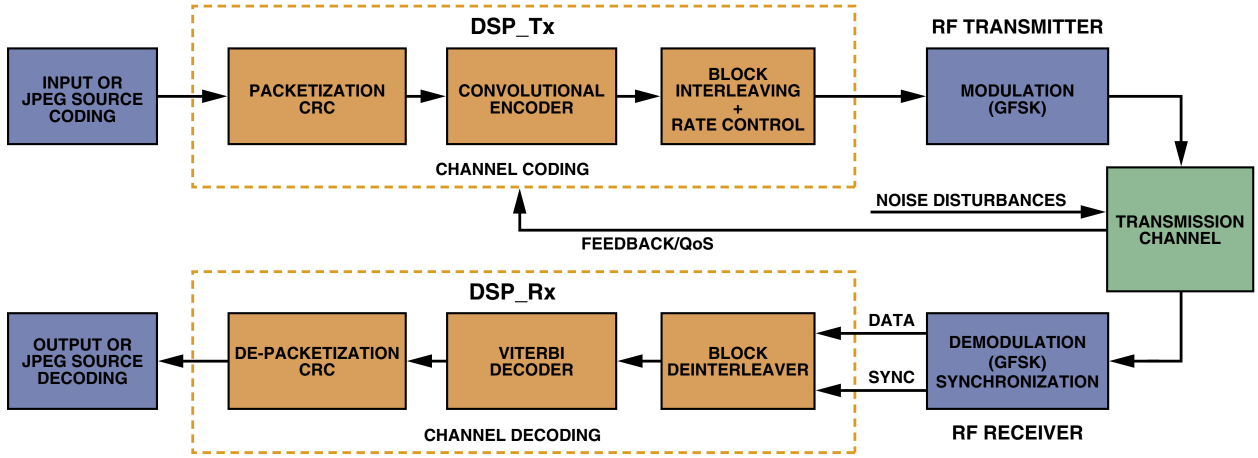


Figure 3. Signal-processing functions.

Figure 3 illustrates the various functions to be carried out across the transmission channel, including processing functions handled for both *transmit* (Tx) and *receive* (Rx) operations. The Blackfin processor, when sitting on the transmitter side handles both data-rate control and data partitioning, so data is transmitted in packets at a quasi-constant rate. The data packets are processed for forward error-correction (FEC) before they modulate the carrier's frequency. This is achieved by adding redundant bits that the receiver will use to detect and correct errors. The bits added to the incoming packets will, of course, increase the required bandwidth for a given information bit rate.

Among the different applicable methods of FEC, *convolutional coding*, while quite simple to implement, gives good protection against channel Gaussian noise disturbances and helps meet minimum Hamming-distance criteria. A convolutional encoder is a finite state-machine comprising an L-stage shift register, N modulo-2 adders, and a multiplexer to convert the output into a serial bit stream. The connections between the shifter outputs and the adder inputs determine the polynomial code. Using two specifically applicable instructions, the Blackfin core performs all these operations very efficiently.

At the other end of the transmission channel, the decoder section implements the Viterbi algorithm (hard-input/hard-output). For maximum likelihood decoding, the Viterbi decoder compares all the possible code sequences to the received code vector. The code sequence whose Hamming distance from the received sequence is the shortest is the good one. For a code like (1/2, 7, 371, 247) with a constraint length, $K = L + 1$ of 7, the decoder can correct up to six consecutive erroneous bits. Depending upon the system requirements, constraint lengths (K) from 5 to 9 must be supported by the ADSP-BF531 in such wireless applications.

However, even a convolutional code with a constraint length of 9 does not protect against burst noise that might hit the transmitted packets over a longer length of time. The use of a complementary protection technique based on temporal diversity is mandatory. *Temporal diversity*, i.e., spreading the bits or symbols out over time, improves the performance of a coded communication system in the presence of multiple paths, fading, and burst noise. It thus reduces the probability of a consecutive number of bits being corrupted. Scrambling and simple block interleaving functions achieve this objective without employing more complex corrective codes (like Reed-Solomon). Here again, the ADSP-BF531 is helpful with two specific vector instructions—one that computes the Viterbi trellis butterflies and one that reconstructs data for the path-search (trace-back) operation.

This encoded data is then passed on to the ADF7020 transmitter section, which does some additional filtering and Gaussian frequency-shift-keying (GFSK) modulation. GFSK modulation has the advantage of reducing the occupied spectral bandwidth—a helpful operation when seeking to meet adjacent-channel requirements for the European 868-MHz bands.

On the receiver side, the ADF7020's internal preamble-matching circuitry helps to fulfill the critical packet-synchronization task. This hardware function permits the recognition or identification of a 12-, 16-, 20-, or 24-bit-long programmable synchronization word, or a packet preamble, without the intervention of the ADSP-BF531 core. Upon a valid preamble match, the circuitry asserts the ADF7020 INT/LOCK pin, which signals the beginning of a new packet to the serial port (RFS0) and triggers the Viterbi decoder. This unique circuitry is somewhat error-tolerant—in a sense, it even allows a valid match for up to three incorrect bits. This reduces the number of packets lost due to preamble misses, as the preamble is not encoded and is therefore not protected. To further reduce preamble misses, the receiver uses one of the ADSP-BF531 32-bit timers as a watchdog that generates the expected pulse on RFS0 if the INT/LOCK signal does not show up after a few symbols. This use of a hardware mechanism to retrieve packet synchronization markers was chosen in order to save a lot of processor MIPS—compared to a full implementation with software analysis and tracking.

Real-World Application—Wireless Video Over ISM

As noted earlier, efficient wireless digital-video transmission calls for robustness against channel failures. Video codecs are excellent candidates for applications with smart, reliable Blackfin processor-based wireless modems. Given the limitation of the ISM wireless channel bandwidth, a relatively high image/video compression ratio is required in order to deliver the expected frame rate and quality for a given image size without too much latency. Unfortunately, Motion JPEG and other video codecs require a very low transmission-error rate, typically 10^{-6}, because the source-coding process removes most of the redundant information. This is particularly true with some efficient entropy coders, such as Huffman, where a single erroneous bit makes the original data impossible to decode. A required bit-error rate (BER) less than 10^{-6} places very stringent requirements on the radio, but it can be achieved by using a channel coding scheme like the one described above.

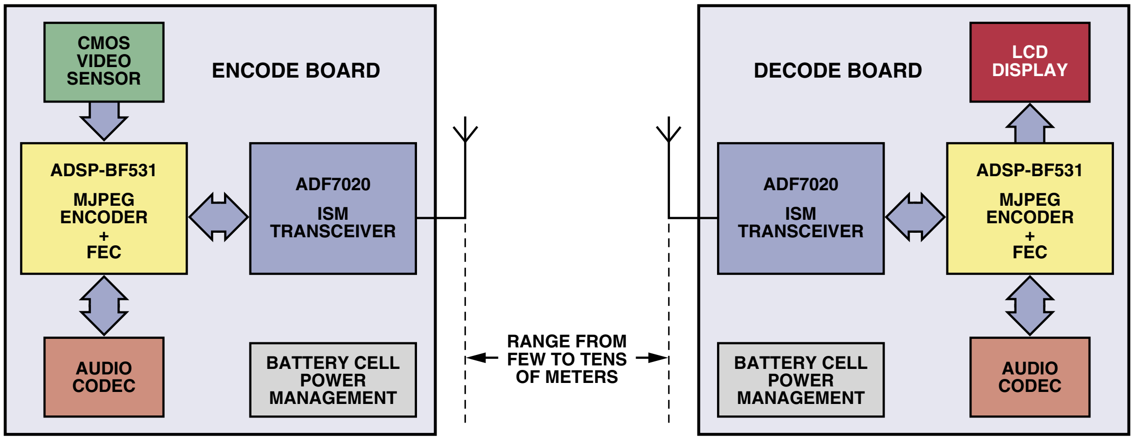


Figure 4. Video transmission system application.

A very low BER does not ensure that all the data packets will be entropy-decoded correctly. To improve the image quality, it is necessary to provide some mechanism to conceal part of an image if too many important bits in a packet are corrupted. For this purpose, every packet is segmented and entropy-coded separately. After the detection of an erroneous segment or block, its content is discarded. Depending upon the information lost, the dc and first two ac coefficients of the discrete cosine transform (DCT) of the corresponding image block are estimated from the coefficients of the neighboring blocks. The final low-pass 2D 3 × 3 de-blocking filter stage, designed to remove DCT blocking artifacts, helps to smooth resulting distortion.

The ADSP-BF531 has more than enough power to handle both the MJPEG encoding or decoding and the channel FEC processing. No external memory is required for frame sizes up to QCIF (176 pixels by 144 pixels) with a 4:2:2 video format. Larger frames are possible at the cost of an external SDRAM, which can also be used to store compressed video. This very low-cost processor can interface directly to CCIR-656-compatible low-cost CMOS image sensors or TFT displays via its *parallel peripheral interface* (see "Blackfin Processor's Parallel Peripheral Interface Simplifies LCD Connection in Portable Multimedia").[10] Standard low-cost, low-power PCM audio codecs can be connected to the available serial port, SPORT1, to support digital transmission of speech or audio. Or, the processor can provide speech coding and decoding with moderate delay by executing a software codec similar to the FR-GSM (13 kbps).

With a raw data rate of 200 kbps it is possible to achieve "baseline" MJPEG transmission over ISM at a rate of about four QCIF 4:2:2 color-frames per second (fps), while leaving 20 kbps for speech. This is acceptable for simple low-cost consumer appliances, such as video baby monitors, video door phones, or wireless home-security cameras. The functional block diagram of such a point-to-point video transmission system (baby monitor) is shown in Figure 4. The overall bill of materials (BOM) for this application is in the $75 range; and the 2.5" LCD TFT display is the most expensive part.

The application code corresponding to the system block diagram shown at Figure 4 is available from Arbos Ingénierie (www.arbos-dsp.com),[11] a French DSP third-party partner of Analog Devices.

CONCLUSION

The unique combination of the ADF7020 ISM-band transceiver and the ADSP-BF531 Blackfin processor exhibits excellent radio-link performance at a very attractive cost, with demonstrable versatility in various ISM digital wireless transmission systems. Further improvements to this communications model can be anticipated with future members of the ADF702x RF transceiver family and new TCP/IP friendly Blackfin DSP processors, together with additional channel- and source-coding software modules.

REFERENCES—VALID AS OF MARCH 2005

1 http://www.bluetooth.com
2 http://grouper.ieee.org/groups/802/11/main.html
3 http://www.analog.com/en/prod/0,2877,ADF7020,00.html
4 http://www.analog.com/processors/processors/blackfin/index.html
5 http://www.analog.com/en/prod/0,2877,ADSP-BF531,00.html
6 http://www.linxtechnologies.com/documents/EN300220-1_2000.pdf
7 http://www.fcc.gov/oet/info/rules
8 http://www.analog.com/en/prod/0,2877,ADF7025,00.html
9 http://www.analog.com/en/content/0,2886,770_850_16127,00.html
10 http://www.analog.com/library/analogdialogue/archives/39-01/lcd_drive.html
11 http://www.arbos-dsp.com

This article can be found at http://www.analog.com/library/analogdialogue/archives/39-03/smart_modem.html, with a link to a PDF.

Blackfin® Processor's Parallel Peripheral Interface Simplifies LCD Connection in Portable Multimedia

By David Katz [david.katz@analog.com]
Ching Lam [ching.lam@analog.com]
Rick Gentile [richard.gentile@analog.com]

As low-power, fixed-point processors such as ADI's Blackfin[1] family increase in performance and popularity, they can serve more and more multimedia applications. Many of these applications require small, low-power *liquid-crystal-display* (LCD) panels that have, in general, lower video resolutions than the full NTSC/PAL video used for broadcast TV. These panels are usually controlled either by a microcontroller or a dedicated LCD controller chip. But today, Blackfin processors have sufficient performance to handle both signal processing and control functions, and also to interface directly to the LCD displays—considerably reducing system cost and complexity. This article will discuss how the ADSP-BF561[2] Blackfin processor's *parallel peripheral interface*[3] (PPI) integrates LCD display capability into the world of high-performance media processing, allowing a single processor to be used for both system processing and display driving.

Passive vs. Active

There are two major categories of LCD array technology—*passive*-matrix and *active*-matrix.

In the former, a glass substrate imprinted with rows forms a liquid-crystal sandwich with a substrate imprinted with columns. Pixels are defined at row-column intersections. To activate a given pixel, a timing circuit energizes the pixel's column while grounding its row. The resulting voltage differential renders the liquid crystal opaque in the vicinity of that pixel location, blocking light from coming through.

Although it is straightforward, passive matrix technology does have some shortcomings. For one, screen refresh times are relatively slow (which can result in *ghosting* for fast-moving images). Also, there is a tendency for the voltage field at a row-column intersection to *bleed* over into neighboring pixels, partly untwisting the liquid crystals and blocking some light from passing through the surrounding pixel area. The effect is to blur edges in the image and reduce contrast.

Active-matrix LCD technology, using an IC-like manufacturing process, is a considerable improvement. Each pixel has a capacitor, to retain charge between refresh cycles, and a transistor switch (giving rise to the popular term, *thin-film-transistor*—TFT—display). To address a particular pixel, its row is enabled, and a voltage is applied to its column. This has the effect of isolating only the pixel of interest, so others in the vicinity are not influenced. The current drawn in controlling a given pixel is reduced, so pixels can be switched at a faster rate, leading to faster refresh rates for TFTs compared to passive displays. What's more, modulating the voltage level applied to the pixel allows many discrete levels of brightness. Today, it is common to have 256 levels, corresponding to 8 bits of intensity.

For color displays, each pixel actually has three subpixels—with red, green, and blue (R-G-B) filters—that the human eye sees as a single-color spot. For example, a 320×240 pixel display actually has 960×240 subpixels, accounting for the R, G, and B components. Each subpixel has 8 bits of intensity, thus forming the basis of the common 24-bit color LCD display.

Since LCD technology relies on regulating the passage of light at the pixel level, one might wonder where the light would be generated. Many small, low-cost monochrome LCDs are *reflective*, meaning that external light reflects off the substrates but is blocked in areas where a liquid crystal segment is charged.

Since TFT color displays have millions of transistors that filter the incoming light, reflective displays would not be effective in active-matrix technology. Instead, the displays are backlit (or *transmissive*); typically a fluorescent light—or a white *light-emitting-diode* (LED) array, integrated into the display—generates light that is modulated as it is transmitted through the various layers of the LCD "sandwich." Unfortunately, the large surface area consumed by the transistors necessitates a greater light output from the backlight. In addition, each transistor of a TFT display dissipates power, so active-matrix displays are somewhat power-hungry compared with their passive cousins.

Components of a TFT-LCD System

Connecting to a TFT-LCD panel can seem complicated, considering all of the different components involved. First, there's the *panel* itself, which houses an array of pixels arranged for strobing by row and column at high speed, referenced to the pixel-clock frequency.

The backlight is often a *cold-cathode fluorescent lamp* (CCFL). In a CCFL, excited gas molecules emit bright light while generating very little heat. This low dissipation, plus their durability, long life, and straightforward drive requirements, make them ideal for LCD panel applications. As mentioned above, LEDs are also a popular backlight method, mainly for small- to mid-sized panels. They have the advantages of low cost, low operating voltage, long life, and good intensity control. However, in larger panels, LED backlights can draw a lot of power compared to CCFL solutions.

An LCD *controller* contains most of the circuitry needed to convert an input video signal into the proper format for display on the LCD panel. It usually includes a *timing generator*, which controls the synchronization and pixel-clock timing of the individual pixels on the panel. Additionally, it can offer a wide variety of extra features—such as on-screen display, graphics overlay blending, color lookup tables, dithering, and image-rotation. The more elaborate chips can be very expensive, often surpassing the cost of the processor to which they're connected. Some media processors, like ADI's Blackfin family, have ports that act electrically as an LCD interface—without requiring an external chip.

An *LCD driver* chip is necessary to generate the proper voltage levels to the LCD panel. It serves as the *translator* between the output of the LCD controller and the LCD panel. The rows and columns are usually driven separately, with timing controlled by the timing generator. Since dc currents will stress the crystal structure and ultimately cause deterioration, liquid crystals must be driven with periodic polarity inversions. Therefore, depending on the implementation, the voltage polarity applied to each pixel varies on either a per-frame, per-line, or per-pixel basis.

Connecting to TFT-LCD Modules

With the trend toward smaller, cheaper multimedia devices, there has been a push to combine the driver, controller, and LCD panel. Today, integrated TFT-LCD modules include timing generation and drive circuitry—thus requiring only a data-bus connection, clocking/synchronization lines, and power supplies. However, in order to meet panel-thickness and cost requirements in smaller PDA-type LCD panels, the timing generator often cannot be integrated into the LCD module. In this case, a separate external timing ASIC is required to produce timing signals to drive the individual rows and columns of the LCD panel.

Nevertheless, the ADSP-BF561 Blackfin Processor can *directly* connect to many TFT-LCD modules through its *parallel peripheral interface* (PPI). The PPI is a multifunction parallel interface that can be configured between 8- and 16 bits in width. Supporting bidirectional data flow, it includes three synchronization lines and a clock pin for connection to an externally supplied clock. In addition to connecting to LCD panels, the PPI can gluelessly decode ITU-R BT.656 data and can also interface to ITU-R BT.601 video streams.

Because the ADSP-BF561 provides many general-purpose timers with *pulse-width-modulation* (PWM) capability, it can be configured to provide the proper LCD timing to a module, thus eliminating the need for an external timing ASIC. Figure 1 shows a block diagram of the basic connection between the Blackfin Processor and a TFT-LCD module. Also shown is the ADSP-BF561 EZ-KIT Lite evaluation board;[4] its many conveniences provide an easy way to get started with a wide variety of Blackfin applications, including the one discussed here.

Power Requirements

TFT-LCD panels typically need two separate power supplies. First, the panel itself has a power supply line. Although the voltage supply requirement varies among LCD panels, the usual values are either 3.3 V or 5 V. Second, CCFL backlights need a high-voltage supply to excite the gas molecules to fluorescence. This voltage is usually generated with a dc-ac inverter on a separate circuit board within the TFT-LCD module. On the other hand, LED backlights, not requiring a high-voltage ac supply, can usually be powered directly from a 5-V or 12-V dc source.

Clocking and Synchronization

The pixel clock period defines the pixel sampling rate, so speeds vary depending on panel resolution and refresh interval. For instance, a VGA panel (640 × 480 active pixels) with a 60-Hz refresh rate would require a 250-MHz clock, whereas a QVGA panel (320 × 240 active pixels) could run at 5 MHz.

The synchronization lines control the time during which each line and video frame is scanned and displayed on the LCD. There are two scanning methods, *interlacing* and *progressive* scan. In interlacing, the odd lines of the video frame are first drawn onto the screen, and then the even lines are filled in. In progressive scan, the video lines are displayed continuously in sequence.

Many newer progressive scan TFT-LCD panels use the synchronization lines to control where each line and frame begins and ends. The horizontal sync (HSYNC) indicates the beginning of each new line, while the vertical sync (VSYNC) denotes the beginning of each new frame. They ensure the generation of an aligned and viewable image. The polarity of the HSYNC and VSYNC pulses and the durations of the pulse widths vary among panels.

The ADSP-BF561 generates the HSYNC and VSYNC signals with configurable PWM outputs in order to achieve the greatest flexibility. This allows adjustments for the polarity, pulse-width, and period specified by a particular TFT panel.

Often, LCD timing requirements specify an *invalid* data period between the assertion of the horizontal sync signal and the actual displayed image data. The ADSP-BF561's PPI can handle this timing by allowing outgoing data to be delayed by a specified number of clock cycles after the HSYNC signal is received.

Data Lines

Although the module's data interface is straightforward, there are many things to consider in choosing the appropriate RGB data format. The three most common configurations use either 8 bits per channel for RGB (8-8-8 format), 6 bits per channel (6-6-6 format), or 5 bits per channel for R and B—and 6 bits for G (5-6-5 format).

The 8-8-8 RGB data format provides the greatest color clarity. With a total of 24 bits of resolution, more than 16 million shades of color are available. This format offers the precision and resolution needed for high-performance LCD TVs.

The 6-6-6 format is popular in portable electronics. The 18 bits of resolution provide over 262,000 shades of color. However, because the 18-pin (6+6+6) data bus doesn't conform nicely to 16-bit processor data paths, a popular industry compromise is to use 5 bits each of R and B, and 6 bits of G (5+6+5 = 16) to match the 16-bit data bus. This scenario works well because, of the three, green is the most visually important color. The least-significant bits of both red and blue are tied to their respective most-significant bits at the panel. This ensures a full dynamic range for each color channel (from full saturation to total black).

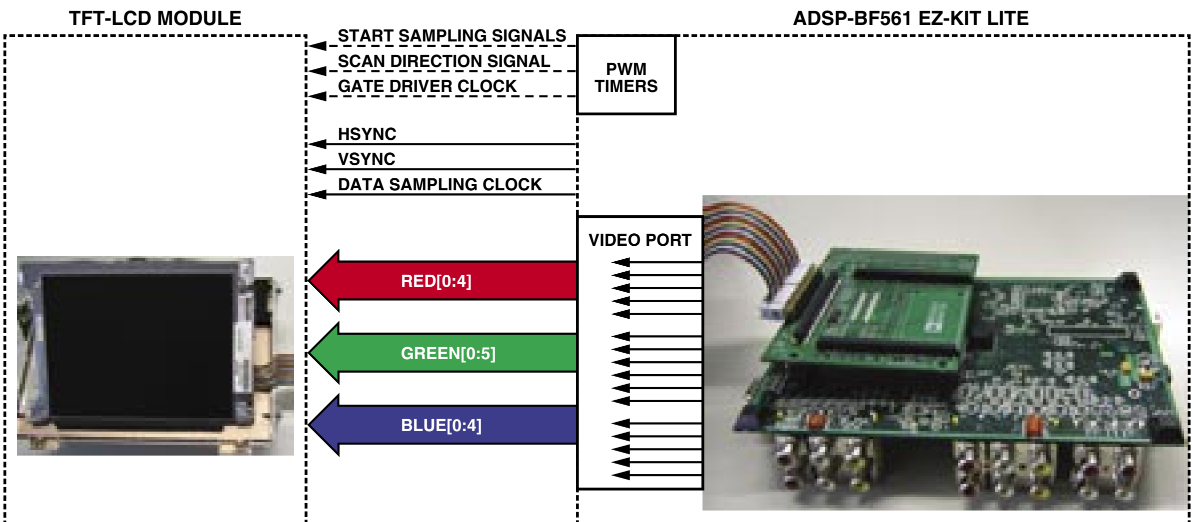


Figure 1. 5-6-5 LCD connection: The ADSP-BF561 eliminates the need for a timing ASIC by supplying the dotted-line connections.

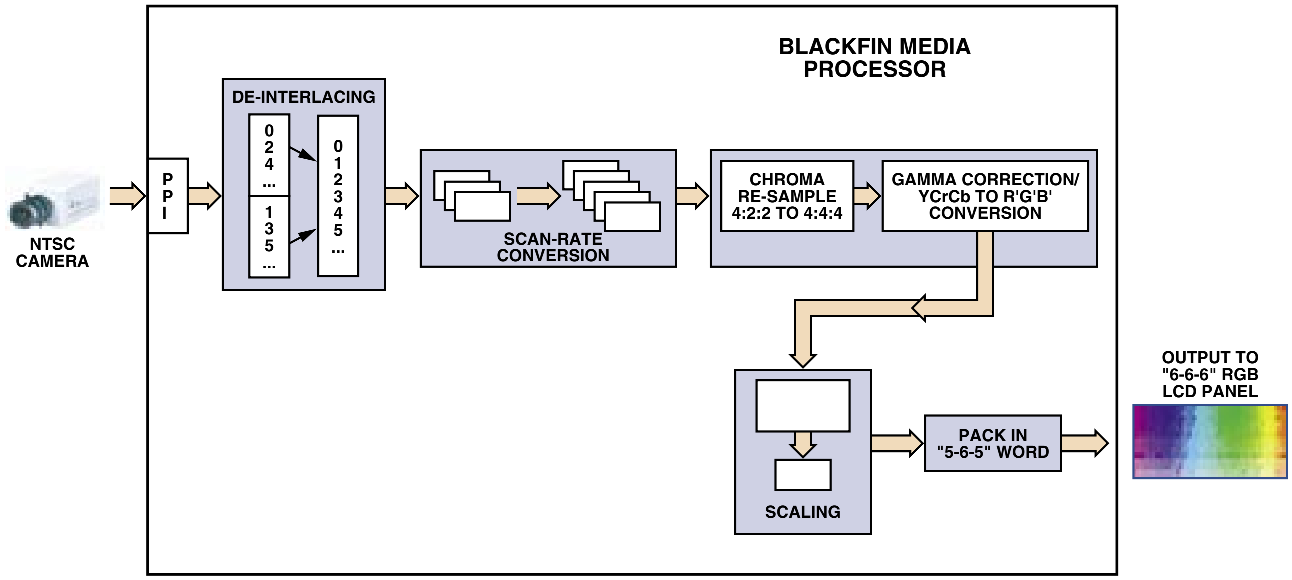


Figure 2. Example of system flow: converting a signal from a camera source to an LCD display output.

System Algorithm Flow

To understand what's involved in emulating an LCD controller on a media processor (in order to replace an external device), let's take a look at the system flow involved in displaying an incoming raw video stream on an integrated TFT-LCD module. Consider the example of Figure 2, where the digitized output of an NTSC camera provides the image stream applied to the video port of the ADSP-BF561 processor. We will discuss each of the steps shown in the figure.

De-interlacing

In interlaced video, used by the NTSC camera in the example, odd and even fields are separated, so that all odd lines in a given frame are transferred before any even lines. For this example, the video stream from the camera must be de-interlaced after it enters the video port. This is done in one of several ways, depending on the desired output quality. The simplest method is *line doubling*, which copies each odd line onto the subsequent even line, effectively eliminating the even field in favor of a shifted version of the odd field. Because this creates noticeable artifacts, more processing-intensive methods are often used. These include *linear interpolation*, *motion compensation*, and *median filtering*. This latter method replaces each pixel's intensity value with the median gray-scale value of its immediate neighbors to help eliminate high-frequency noise in the image.

Scan-Rate Conversion

Once the video has been de-interlaced, a scan-rate conversion process may be necessary in order to insure that the input frame rate matches the output display-refresh rate. In order to equalize the two, fields may need to be dropped or duplicated. As with de-interlacing, some sort of filtering is desirable to smooth out high-frequency artifacts caused by creating abrupt frame transitions.

Chroma Resampling and Color Conversion (YCrCb → RGB)

Some cameras supply pixel information in raw form, exactly as the image sensor supplies it. This could mean one red, blue, and green value for each pixel in the sensor, or one Y, Cr, and Cb value for each pixel. Y, Cr, and Cb are mathematically related to the RGB values, but being less inter-correlated than RGB data, they allow better compression ratios. More commonly, though, the camera outputs a condensed stream that takes advantage of the physiology of the eye, providing greater weighting for green (in the RGB case) or for intensity (Y) in the YCrCb space. In the example of Figure 2, the video stream enters the PPI in 4:2:2 YCrCb format. "4:2:2" implies that there are four luma (Y) intensity values for every two chroma (Cr and Cb) values on a given video line. Each (Y,Cb) or (Y,Cr) 16-bit pair represents one pixel value.

For display on an LCD panel, the data stream ultimately needs to be converted to RGB space. More correctly, it needs to be transformed to R'G'B' space, which is a *gamma*-corrected version of RGB space. Gamma correction adjusts for the nonlinear properties of the LCD panel, since the brightness of a given pixel is not a linear function of the voltage applied at that pixel site. Varying gamma changes the ratios of red to green to blue in an image, as well as image brightness. Figure 3 shows a sample equation set for converting between YCrCb space and R'G'B' coordinates.

$$Y = (0.301)R' + (0.586)G' + (0.113)B'$$
$$Cb = -(0.172)R' - (87/256)G' + (0.512)B' + 128$$
$$Cr = (0.512)R' - (0.430)G' - (0.082)B' + 128$$

$$R' = Y + 1.371(Cr - 128)$$
$$G' = Y - 0.698(Cr - 128) - 0.336(Cb - 128)$$
$$B' = Y + 1.732(Cb - 128)$$

Figure 3. Sample of conversion equations between gamma-corrected RGB and YCrCb color spaces (assuming 8-bit pixel components).

Prior to R'G'B' conversion, the Cb and Cr channels must be resampled to achieve a 4:4:4 format, where one byte each of Y, Cb, and Cr represents one pixel value, as shown in Figure 4. A clear-cut way to resample is to interpolate the missing chroma values from their nearest neighbors by simple averaging. Higher-order filtering might be necessary for some applications, but this simplified approach is often sufficient. In reality, the steps of chroma resampling and color space conversion can both be performed as a single operation, since each discrete step involves linear pixel operations.

Scaling

Video scaling, the next step, is very important because it allows the generation of an output stream whose resolution is different from that of the input format. Ideally, the fixed scaling requirements (input data resolution, output panel resolution) are known ahead of time, in order to avoid the computational load of arbitrary scaling between input and output streams. As a much simpler, cheaper option, the processed image can be cropped to fit within the confines of a smaller LCD panel.

Depending on the application, scaling can be done either upwards or downwards. It is important to understand the nature of the image content to be scaled (e.g., the presence of text and thin lines). Improper scaling can make text unreadable or cause some horizontal lines to disappear in the scaled image.

The most straightforward methods of scaling involve either dropping pixels or duplicating existing pixels. That is, when scaling down to a lower resolution, a number of pixels on each line (and/or some number of lines per frame) can be discarded. While this represents a low processing load, the results will yield aliasing and visual artifacts.

A small step upward in complexity uses linear interpolation to improve the image quality. For example, when scaling down an image, interpolation in either the horizontal or vertical directions provides a new output pixel to replace the pixels used in the interpolation process. As with the previous technique, information is still thrown away, so artifacts and aliasing will again be present.

If the image quality is paramount, there are other ways to perform scaling—without reducing quality. These methods strive to maintain the high frequency content of the image consistent with the horizontal and vertical scaling, while reducing the effects of aliasing. For example, assume an image is to be scaled by a factor of $Y \times X$. To accomplish this scaling, the image could be up-sampled (interpolated) by a factor, Y, filtered to eliminate aliasing, and then down-sampled (decimated) by a factor X. In practice, these two sampling processes can be combined within a single multirate filter.

Bit Extraction/Byte Packing

As described earlier, it is preferable to transfer 16 bits on each outgoing LCD clock cycle. This 5-6-5 bit packing can be accomplished with the source data. The Blackfin architecture offers a choice between two efficient methods to create the desired byte stream. The first is simply to shift the appropriate bits from each color (red, blue, and green), into a target register. The second is to make use of an EXTRACT/DEPOSIT instruction pair to pull out some number of bits, beginning at a specific bit location, and deposit the result in a target register.

Application Note EE-256[5] provides a detailed description of a system where the processor, mounted on an ADSP-BF561 EZ-KIT Lite evaluation board,[6] receives a streaming video input from a DVD player and connects to a TFT-LCD module. The Blackfin generates all necessary timing and performs decimation, color conversion, resampling, and output formatting. System data flows and buffer management are described in detail, and sample code for a working application with a specific LCD module is provided for download.

CONCLUSION

Due to its performance and popularity, members of the Blackfin processor family are serving in increasing numbers of multimedia applications. They are especially useful in system designs calling for displays that require small, low-power, moderate-resolution *liquid-crystal-display* (LCD) panels. For many of these applications, Blackfin processors have sufficient performance to handle both signal processing and control functions, and also to interface directly to the LCD displays—considerably reducing system cost and complexity. This article has suggested how such a system can be accomplished, by employing a portion of the ADSP-BF561 Blackfin processor's spare computing power and its *parallel peripheral interface* for display driving.

REFERENCES—VALID AS OF JANUARY 2005

[1] http://www.analog.com/processors/processors/blackfin/

[2] http://www.analog.com/en/epProd/0%2%2CCADSP-BF561%2C00.html

[3] http://www.analog.com/UploadedFiles/Associated_Docs/84676332PPI_TL.pdf

[4] http://www.analog.com/en/epHSProd/0%2C2542%2CBF561%2DHARDWARE%2C00.html

[5] http://www.analog.com/UploadedFiles/Application_Notes/44507396349702717122866EE256v01.pdf

[6] http://www.analog.com/en/epHSProd/0%2C2542%2CBF561%2DHARDWARE%2C00.html

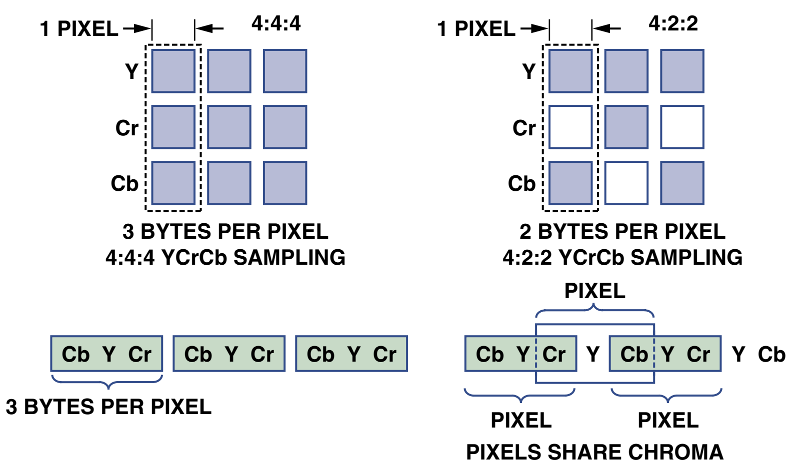


Figure 4. Illustration of 4:4:4 and 4:2:2 YCrCb sampling.

This article can be found at http://www.analog.com/library/analogdialogue/archives/39-01/lcd_drive.html, with a link to a PDF.

Enhance Processor Performance in Open-Source Applications

By David Katz [david.katz@analog.com]
Tomasz Lukasiak [tomasz.lukasiak@analog.com]
Rick Gentile [richard.gentile@analog.com]

As "open source" C/C++ algorithms become an increasingly popular alternative to royalty-based code in embedded processing applications, they bring new technical challenges. Foremost among these is how to optimize the acquired code to work well on the chosen processor. This issue is paramount because a compiler written for a given processor family will exploit that processor's strengths at the possible expense of inefficiencies in other areas. Performance can be degraded when the same algorithm is run directly out-of-the-box on a different platform. This article will explore the porting of such open-source algorithms to Analog Devices Blackfin® processors,[1] outlining in the process a "plan of attack" leading to code optimization.

What is Open Source?

The generally understood definition of "open source" refers to any project with source code that is made available to other programmers. Open-source software typically is developed collaboratively within a community of software programmers and distributed freely. The Linux[2] operating system, for example, was developed this way. If all goes well, the resulting effort provides a continuously evolving, robust application that is well-tested because so many different applications take advantage of the code. Programmers are encouraged to use the code because they do not have to pay for it or develop it themselves, thus accelerating their project schedule. Their successful use of the code provides further test information.

The certification stamp of "Open Source" is owned by the Open Source Initiative (OSI). Code that is developed to be freely shared and evolved can use the Open Source trademark if the distribution terms conform to the OSI's Open-Source Definition.[3] This requires that the software be redistributed to others under certain guidelines. For example, under the General Public License (GPL), source code must be made available so that other developers will be able to improve or evolve it.

What is Ogg?

There is an entire community of developers who devote their time to the cause of creating open standards and applications for digital media. One such group is the Xiph.Org Foundation,[4] a nonprofit corporation whose purpose is to support and develop free, open protocols and software to serve the public-, developer-, and business markets. This umbrella organization (see Figure 1) oversees the administration of such technologies as *video-* (Theora), *music-* (the lossy Vorbis and lossless FLAC), and *speech* (Speex) *codecs.*

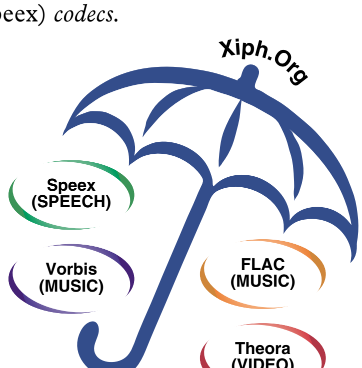


Figure 1. Xiph.Org open-source 'umbrella'

The term *Ogg* denotes the container format that holds multimedia data. It generally serves as a prefix to the specific codec that generates the data. Vorbis, an audio codec we'll discuss here, uses Ogg to store its bitstreams as files, so it is usually called "Ogg Vorbis."[5] In fact, some portable media players are advertised as supporting Ogg files, where the "Vorbis" part is implicit. Speex, a speech codec discussed below, also uses the Ogg format to store its bitstreams as files on a computer. However, *Voice over Internet Protocol* (VoIP) and other real-time communications systems do not require file storage capability, and a network layer like the Real-Time Transfer Protocol[6] (RTP) is used to encapsulate these streams. As a result, even Vorbis can lose its Ogg shell when it is transported across a network via a multicast distribution server.

What is Vorbis?

Vorbis is a fully open, patent-free, royalty-free audio compression format. In many respects, it is very similar in function to the ubiquitous MPEG-1/2[7] layer 3 (MP3) format and the newer MPEG-4[8] (AAC) formats. This codec was designed for mid- to high-quality (8-kHz to 48-kHz bandwidth, >16-bit, polyphonic) audio at variable bit rates from 16 to 128 kbps/channel, so it is an ideal format for music.

The original Vorbis implementation was developed using floating-point arithmetic, mainly because of programming ease that led to faster release. Since most battery-powered embedded systems (like portable MP3 players) utilize less expensive, more battery-efficient fixed-point processors, the open-source community of developers created a fixed-point implementation of the Vorbis decoder. Dubbed *Tremor,* the source code to this fixed-point Vorbis decoder was released under a license that allows it to be incorporated into open-source and commercial systems.

Before choosing a specific fixed-point architecture for porting the Vorbis decoder, it is important to analyze the types of processing involved in recovering audio from a compressed bitstream. A generalized processor flow for the Vorbis decode process (and other similar algorithms) is shown in Figure 2. Like many other decode algorithms, there are two main stages: front-end and back-end.

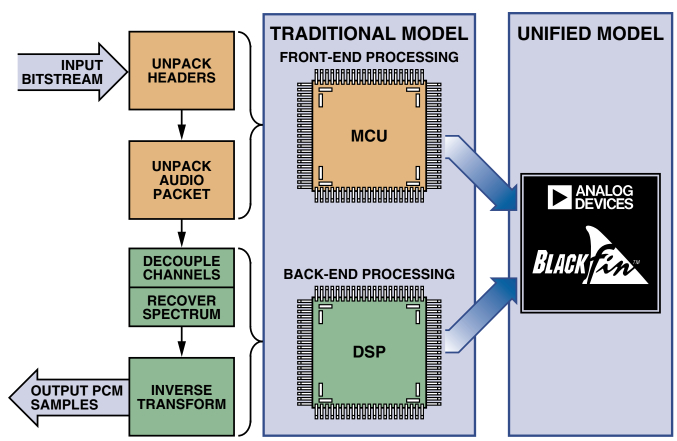


Figure 2. Generalized processor flow for the Vorbis decode process.

During the *front-end* stage, the main activities are header and packet unpacking, table lookups, and Huffman decoding. Operations of this kind involve a lot of conditional code and a relatively large amount of program space, so embedded developers commonly use microcontrollers for the front end.

Back-end processing is defined by filtering functions, inverse transforms, and general vector operations. In contrast to the front-end phase, the back-end stage involves more loop constructs and memory accesses, often using smaller amounts of code. For these reasons, back-end processing in embedded systems has historically been dominated by full-fledged DSPs.

The Blackfin processor architecture unifies microcontroller (MCU) and DSP functionality, so there is no longer a need for two separate devices. It can be used efficiently to implement both front-end and back-end processing on a single chip.

What is Speex?

Speex is an open-source, patent-free audio compression format designed for speech. While Vorbis is used to compress all types of music and audio, Speex targets speech only. For that reason, Speex can achieve much better results than Vorbis on speech at the same quality level.

Just as Vorbis competes with royalty-based algorithms like MP3 and AAC, Speex shares space in the speech codec market with GSM-EFR and the G.72x algorithms, such as G.729 and G.722. Speex also has many features that are not present in most other codecs. These include variable bit rate (VBR), integration of multiple sampling rates in the same bitstream (8 kHz, 16 kHz, and 32 kHz), and stereo encoding support. Also, the original design goal for Speex was to facilitate incorporation into Internet applications, so it is a very capable component of VoIP phone systems.

Besides its unique technical features, Speex has the major advantages that it costs "nothing"—and can be distributed and modified to conform to a specific application. The source code is distributed under a license similar to that of Vorbis. Because the maintainers of the project realized the importance of embedding Speex into small fixed-point processors, a fixed-point implementation has been incorporated into the main code branch.

Optimizing Vorbis and Speex on Blackfin Processors

Immediate "out-of-the-box" code performance is a paramount consideration when an existing application, such as Vorbis or Speex, is ported to a new processor. However, software engineers can reap a big payback by familiarizing themselves with the many techniques available for optimizing overall performance. Some require only minimal extra effort.

The first step in porting any piece of software to an embedded processor like Blackfin is to customize the low-level I/O routines to fit the system needs. For example, the reference code for both Vorbis and Speex assumes that data originates from a file *and* processed output is stored into a file (mainly because both implementations were first developed to run on Unix/Linux systems where file I/O routines were available). In an embedded media system, however, the input and/or output are often connected to A/D and D/A *data converters* that translate between the digital and real-world analog domains. Figure 3 shows a conceptual overview of a possible Vorbis-based media player implementation. The input bitstream is transferred from a flash memory and the decoder output drives an audio DAC. Also, while some media applications (for example, portable music players) still use files to store data, many systems replace storage with a network connection.

When optimizing a system like the Vorbis decoder to run efficiently, it is a good idea to have an organized plan of attack. One possibility is to first optimize the algorithm from within C, then to streamline system data flows, and finally to tweak individual pieces of the code at an assembly level. Figure 4 illustrates a representative reduction of processor load through successive optimization steps and shows how efficient this method can be.

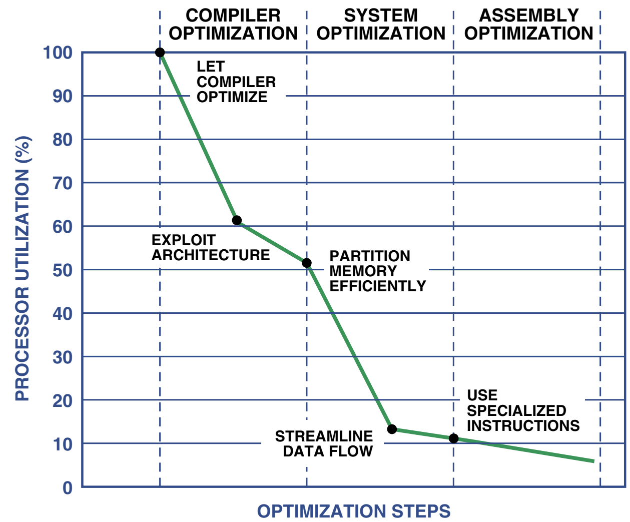


Figure 4. Steps in optimizing Vorbis source code on Blackfin, leading to significantly reduced processor utilization.

Compiler Optimization

Probably the most useful tool for code optimization is a good profiler. Using the *statistical* profiler in VisualDSP++ for Blackfin[9] allows a programmer to quickly focus on hotspots that become apparent as the processor is executing code. In many implementations, 20% of the code takes 80% of the processing time. Focusing on these critical sections yields the highest marginal returns. It turns out that *loops* are prime candidates for optimization in media algorithms like Vorbis because intensive number-crunching usually occurs inside of them.

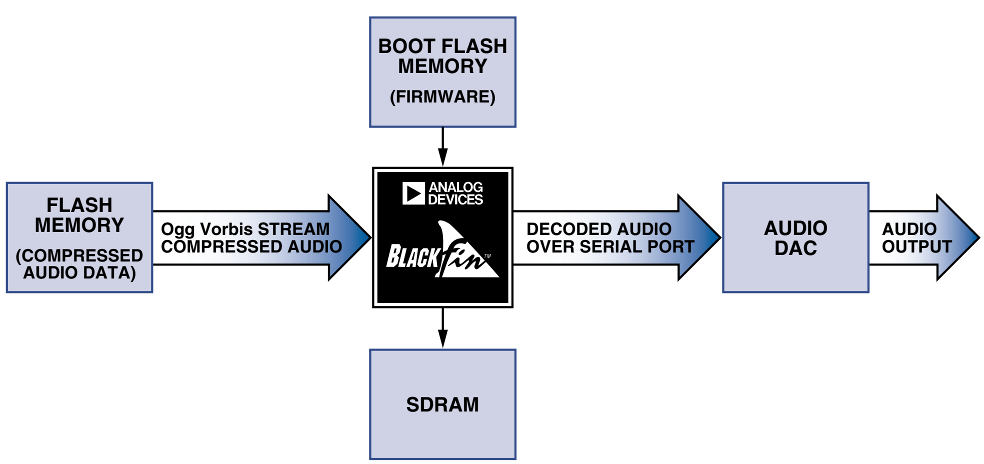


Figure 3. Example: Vorbis media player implementation.

There are also global approaches to code optimization. First, a compiler can optimize for either memory conservation or speed. Also, functions can be considered for automatic *inlining* of assembly instructions into the C code. (The compiler's `inline` keyword is used to indicate that functions should have code generated inline at the point of call. Doing this avoids various costs such as program flow latencies, function entry and exit instructions, and parameter passing overhead.) This, too, creates a trade-off between space and speed. Lastly, compilers like the one available for Blackfin can use a two-phase process to derive relationships between various source files within a single project to further speed up code execution (*inter-procedural analysis*).

As mentioned above, most reference software for media algorithms uses floating-point arithmetic. But software written with fractional fixed-point machines in mind still misses a critical piece. The language of choice for the majority of codec algorithms is C, but the C language doesn't "natively" support the use of fractional fixed-point data. For this reason, many fractional fixed-point algorithms are *emulated* with integer math. This may make the code highly portable, but it doesn't approach the performance attainable by rewriting some math functions with machine-specific compiler constructs for highest computational efficiency.

A specific example illustrating this point is shown in Figure 5. The left column shows C code and Blackfin compiler output for emulated fractional arithmetic that works on all integer machines. One call to perform a 32-bit fractional multiplication takes 80 cycles. The right column shows the improvement in performance obtainable by utilizing (`mult_fr1x32x32`), an intrinsic function of the Blackfin compiler that takes advantage of the underlying fractional hardware. With this fairly easy modification, an 86% speedup is achieved.

System Optimization

System optimization starts with *proper memory layout*. In the best case, all code and data would fit inside the processor's L1 memory. Unfortunately, this is not always possible, especially when large C-based applications are implemented within a networked application.

The real dilemma is that processors are optimized to move data independently of the core via *direct memory access* (DMA), but MCU programmers typically run using a cache model instead. While core fetches are an inescapable reality, using DMA or cache for large transfers is mandatory to preserve performance.

To introduce the discussion, let's consider several attributes inherently supported by the Blackfin bus architecture. The first is the *ability to arbitrate requests without core intervention*. Because internal memory is typically constructed in sub-banks, simultaneous access by the DMA controller and the core can be accomplished in a single cycle by placing data in separate banks. For example, the core can be operating on data in one sub-bank while the DMA is filling a new buffer in a second sub-bank. Under certain conditions, simultaneous access to the same sub-bank is also possible.

There is usually only one physical bus available for access to external memory. As a result, the arbitration function becomes more critical. Here's an example that clarifies the challenge: on any given cycle, an external memory location may be accessed to fill the instruction cache at the same time that it serves as the source and destination for incoming and outgoing data.

Instruction Execution

Blackfin processors use hierarchical memory architectures that strive to balance several levels of memory having differing sizes and performance levels. On-chip *L1 memory*, which is closest to the core processor, operates at the full clock rate. This memory can be configured as SRAM and/or cache. Applications that require the most determinism can access on-chip SRAM in a single core clock cycle. For systems that require larger code sizes, additional on-chip and off-chip memory is available—with increased latency.

SDRAM is slower than L1 SRAM, but it's necessary for storing large programs and for data buffers. However, there are several ways for programmers to take advantage of the fast L1 memory. If the target application fits directly into L1 memory, no special action is required other than for the programmer to map the application code directly to this memory space—as in the Vorbis example described above.

If the application code is too large for internal memory, as is the case when adding, say, a networking component to a Vorbis codec, a caching mechanism can be used to allow programmers to access larger, less expensive external memories. The *cache* serves as a way to automatically bring code into L1 memory as it is needed. Once in L1, the code can be executed in a single core cycle, just as if it were stored on-chip in the first place. The key advantage of this process is that the programmer does not have to manage the movement of code into and out of the cache.

The use of cache is best when the code being executed is somewhat linear in nature. The instruction cache really performs two roles. First, it helps pre-fetch instructions from external memory in a more efficient manner. Also, since caches usually operate with some type of "least recently used" algorithm, instructions that run the most often are typically retained in cache. Therefore, if the code has been fetched once and hasn't yet been replaced, it will be ready for execution the next time through the loop.

ORIGINAL

```
int32 MULT31(int32 a, int32 b) {
 int32 c;
 c = ((long long)a * b) << 1;
 return c;
}

R2 = R0 ; // lo(a)
R0 = R1 ; // lo(b)
R1 >>>= 0x1f ; // hi(a)
R3 = R2 >>> 31 ; // hi(b)
[ SP + 0xc ] = R3 ;
CALL ___mulli3 ; // 64x64 mult (43 cycles)
R2 = 1 ;
R3 = R2 >>> 31 ;
[ SP + 0xc ] = R3 ;
CALL ___lshftli ; // 64 shift (28 cycles)
P0 = [ FP + 0x4 ] ;

80 cycles
```

IMPROVED

```
fract32 MULT31 (fract32 a, fract32 b) {
 fract32 c;
 c = mult_fr1x32x32(a, b);
 return c;
}

A1 = R0.L * R1.L ( FU ) || P0 = [ FP + 0x4 ] ;
A1 = A1 >> 16 ; // use of accumulator
R2 = PACK ( R0.L , R1.L ) ;
CC = R2 ;
A1 += R0.H * R1.L ( M ) , A0 = R0.H * R1.H ;
CC &= AV0 ;
R2 = CC ;
A1 += R1.H * R0.L ( M ) ;
A1 = A1 >>> 15 ;
R0 = ( A0 += A1 ) ;
R0 = R0 + R2 ;

11 cycles (14%)
```

Figure 5. Compiler intrinsic functions are an important optimization tool.

Wary real-time programmers have not trusted cache to obtain the best system performance because system performance will be degraded if a block of instructions is not in cache when needed for execution. This issue can be avoided by taking advantage of *cache-locking* mechanisms. When critical instructions are loaded into cache, the cache lines can be locked to keep the instructions from being replaced. This lets programmers keep what they need in cache, while allowing less-critical instructions to be managed by the caching mechanism itself. This capability sets the Blackfin processor apart from other signal processors.

Data Management

Having discussed how code is best managed to improve performance on this application, let's now consider the options for data movement. As an alternative to cache, data can be moved in and out of L1 memory using a DMA controller that is independent of the core. While the core is operating on one section of memory, the DMA is bringing in the next data buffer to be processed.

The Blackfin data-memory architecture is just as important to the overall system performance as the instruction-clock speed. Because there are often multiple data transfers taking place at any one time in a multimedia application, the bus structure must support both core and DMA accesses to all areas of internal and external memory. It is critical that arbitration of the DMA controller and the core be handled automatically, or performance will be greatly reduced. Core-to-DMA interaction should only be required to set up the DMA controller, and later to respond to interrupts when data is ready to be processed. In addition, a data cache can also improve overall performance.

In the default mode, a Blackfin performs data fetches as a basic core function. While this is typically the least efficient mechanism for transferring data, it leads to the simplest programming model. A fast scratchpad memory is usually available as part of L1 memory; but for larger, off-chip buffers, the access time will suffer if the core must fetch everything. Not only will it take multiple cycles to fetch the data, but the core will also be busy doing the fetches.

So, wherever possible, DMA should always be employed for moving data. Blackfin processors have DMA capabilities to transfer data between peripherals and memory, as well as between different memory segments. For example, our Vorbis implementation uses DMA to transfer audio buffers to the audio D/A converter.

For this audio application, a "revolving-door" double-buffer scheme is used to accommodate the DMA engine. As one half of the circular double buffer is emptied by the serial port DMA, the other half is filled with decoded audio data. To throttle the rate at which the compressed data is decoded, the DMA interrupt service routine (ISR) modifies a semaphore that the decoder can read—in order to make sure that it is safe to write to a specific half of the double buffer. In a design that lacks an operating system (OS), polling a semaphore means wasted CPU cycles; however, under an OS, the scheduler can switch to another task (like a user interface) to keep the processor busy with real work.

The use of DMA can lead to incorrect results if data coherency is not considered. For this reason, the audio buffer associated with the audio DAC is placed in a noncacheable memory space, since the cache might otherwise hold a newer version of the data than the buffer to be transferred by the DMA.

Assembly Optimization

The final phase of optimization has to do with rewriting isolated segments of the open-source C code in assembly language. The best candidates for performance improvement by an assembly rewrite are usually interrupt service routines (ISRs) and reusable signal-processing modules.

The impetus for writing interrupt handlers in *assembly* is that an inefficient ISR will slow the responses of other interrupt handlers. For example, some audio designs must use the audio ISR to format AC97 data bound for the audio DAC. Because this happens periodically, a long audio ISR can slow down responses of other events. The best way to reduce the interrupt handler's cycle count is to rewrite it in assembly.

A good example of a *reusable signal-processing module* is the *modified discrete cosine transform* (MDCT) used in back-end Vorbis processing to transform a time-domain signal into a frequency domain representation. The compiler can never produce code as "tight" as a skilled assembly programmer can, so the C version of the MDCT is inefficient. An assembly version of the same function can exploit the hardware features of the Blackfin architecture, such as single-cycle *butterfly add/subtract and hardware bit-reversal.*

Today, Blackfin ports of both Vorbis and Speex exist and are available on request. These ports run on the ADSP-BF533 EZ-KIT Lite.[10] An open-source port of μClinux is also available at blackfin.uclinux.org.[11] Together, these enable a wide variety of applications that seek to integrate royalty-free speech or music capability while retaining plenty of processing room for additional features and functionality. For example, the new ADSP-BF536/ADSP-BF537,[12] with its integrated Ethernet MAC, opens the door to low-cost networked audio and voice applications. Clearly, open-source code heralds a revolution in the embedded processing world, and Blackfin processors are poised to take full advantage of this situation.

FOR FURTHER READING

Parris, Cliff and Phil Wright. *Implementation Issues and Tradeoffs for Digital Audio Decoders.* Monmouthshire, UK: ESPICO, Ltd.

REFERENCES—VALID AS OF FEBRUARY 2005

[1] http://www.analog.com/processors/processors/blackfin/
[2] http://www.linux.org/
[3] http://www.opensource.org/docs/definition.php
[4] http://www.Xiph.Org/
[5] http://www.vorbis.com/
[6] http://www.ietf.org/rfc/rfc1889.txt
[7] http://www.chiariglione.org/mpeg/
[8] http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm
[9] http://www.analog.com/en/prod/0,2877,VISUALDSPBF,00.html
[10] http://www.analog.com/en/prod/0,2877,BF533-HARDWARE,00.html
[11] http://blackfin.uclinux.org/
[12] http://www.analog.com/Analog_Root/static/promotions/blackfin750/index.html

This article can be found at http://www.analog.com/library/analogdialogue/archives/39-02/open_source.html, with a link to a PDF.

PRODUCT INTRODUCTIONS: VOLUME 39, NUMBER 1

Data sheets for all ADI products can be found by entering the model number in the Search Box at www.analog.com

January

ADC, Pipelined, dual 10-bit, 105-MSPS **AD9216**
ADC, Pipelined, quad 8-bit, 65-MSPS, LVDS outputs **AD9289**
ADC, Pipelined, 14-bit, 80-MSPS **AD9444**
ADC, Pipelined, 8-bit, 250-MSPS, CMOS outputs, single 3.3-V supply **AD9481**
Amplifier, Operational, low-power, precision, JFET input, rail-to-rail output **AD8641**
Amplifier, Operational, dual, high-performance CMOS, rail-to-rail outputs **AD8692**
Charge-Pump Driver, 4-channel, LED backlight for color LCD display **ADM8843**
Charge-Pump Driver, 6-channel, LED backlight for color LCD display **ADM8845**
Comparators, General-Purpose, include voltage reference **ADCMP35x**
Comparators, General-Purpose, open-drain/push-pull **ADCMP370/ADCMP371**
Comparators, Dual, high-speed, PECL/LVPECL, single supply **ADCMP551/ADCMP552**
Comparator, Single-Supply, high-speed, PECL/LVPECL **ADCMP553**
Controller, Power Supply, high-efficiency, dual-output **ADP3026**
Controller, Synchronous-Buck, 1-/2-/3-phase adjustable-output **ADP3182**
DC-to-DC Converter, step-up, operates at 1.2 MHz .. **ADP1610**
Temperature-Sensor, hub and fan controller, up to 10 remote sensors **ADT7470**
Temperature-to-Digital Converter, 10-bit, includes 4-channel ADC, quad DAC **ADT7519**
Temperature Sensors, ±0.5°C accuracy, pulse-width modulated outputs **TMP05/TMP06**

February

Accelerometer, dual-axis, ±18-*g* **ADXL321**
Embedded Processors, TigerSHARC, 500 MHz/600 MHz, 24-/12-/4-Mbits on-chip **ADSP-TS201/ADSP-TS202/ADSP-TS203**
Supervisor, Power Supply, super sequencer and monitor **ADM1068**

March

Amplifier, Operational, 1.5-GHz, ultrahigh-speed, current-feedback **AD8000**
Amplifier, Operational, dual, rail-to-rail outputs, 310 mA output drive **AD8397**
Amplifier, Operational, 20-MHz, precision, CMOS, rail-to-rail I/O **AD8615**
Amplifier, Operational, quad, single-supply, zero-drift, rail-to-rail I/O **AD8630**
Amplifier, Operational, quad, low-cost, high-speed, rail-to-rail outputs **ADA4851-4**
Filter, Video, selectable cutoff frequencies **ADA4410-6**
ADCs, Sigma-Delta, 3-channel, 16-bit/24-bit, low noise, low power **AD7798/AD7799**
Audio Codec, AC'97 SoundMAX® **AD1986**
CCD Signal Processor, 2-channel, 14-bit, precision timing generator **AD9942**
DAC, Current-Output, 10-bit, 120 mA current sink **AD5398**
DAC, Current-Output, multiplying, 12-bit, wideband, serial interface **AD5444**
DACs, Current-Output, multiplying, 8-/10-/12-bit, wideband, serial interface **AD5450/AD5451/AD5452**
DACs, Voltage-Output, 8-/10-/12-/14-bit, *nano*DAC, SC70 package **AD5601/AD5611/AD5621/AD5641**
DAC, Voltage-Output, 16-bit, *nano*DAC, SOT-23 package **AD5662**
Level Translators, 4-/8-channel, bidirectional logic **ADG3304/ADG3308**

AUTHORS

Patrick Butler (page 3) is a DSP field applications engineer at Analog Devices France, within the Western Europe sales organization. For nearly 18 years, he has provided support for general-purpose DSP and ISM-band RF parts to French customers. Recently, he has focused on home video surveillance and industrial RF modem markets.

A 1979 engineering graduate, he worked as a design engineer for Schlumberger ATE (now Credence), AMD, and Harris Semiconductor. His hobbies include collecting jazz records and old vintage wines, as well as playing with his two young sons.

Rick Gentile (page 7, 11), who joined ADI in 2000 as a senior DSP applications engineer, currently leads the Blackfin® DSP applications group. Before joining ADI, Rick was a member of the technical staff at MIT Lincoln Laboratory, where he designed several signal processors used in a wide range of radar sensors. He received a BS from the University of Massachusetts at Amherst and Master of Science degrees from Northeastern University—in both Electrical and Computer Engineering. He enjoys sailing, windsurfing, and spending time with his daughters.

Austin Harney (page 3) graduated from University College, Dublin, Ireland, in 1999 with a BEng and joined Analog Devices following graduation. He is currently an applications engineer for the ISM-band wireless product line, based in Limerick. In his spare time, Austin enjoys football, music, and spending time with his daughter.

David Katz (page 7, 11) is a senior DSP applications engineer, involved in specifying and supporting Blackfin media processors. He has published dozens of articles on embedded processors—in both U.S. and international publications. Previously, he worked at Motorola as a senior design engineer in cable-modem and automation groups. David holds BS and MEng degrees in Electrical Engineering from Cornell University.

Ching Lam (page 7) is a DSP applications engineer at the Norwood, Massachusetts facility, supporting ADI's Blackfin processors. Joining Analog Devices in 2001, she is a graduate of Boston University with a BSEE degree.

Tom Lukasiak (page 11) has been a DSP applications engineer at Analog Devices since 2000. He is currently focusing on Blackfin processor products. Tom earned his ScB (2000) and ScM (2002) degrees in Electrical Engineering from Brown University.

Analog Dialogue

Analog Devices, Inc.
Worldwide Headquarters
Analog Devices, Inc.
One Technology Way
P.O. Box 9106
Norwood, MA
02062-9106 U.S.A.
Tel: 781.329.4700
(800.262.5643,
U.S.A. only)
Fax: 781.461.3113

Analog Devices, Inc.
Europe Headquarters
Analog Devices SA
17-19 rue Georges Besse
Antony, 92160
France
Tel: 33.1.46.74.45.00
Fax: 33.1.46.74.45.01

Analog Devices, Inc.
Japan Headquarters
Analog Devices, KK
New Pier Takeshiba
South Tower Building
1-16-1 Kaigan, Minato-ku,
Tokyo, 105-6891
Japan
Tel: 813.5402.8210
Fax: 813.5402.1064

Analog Devices, Inc.
Southeast Asia Headquarters
Analog Devices
22/F One Corporate Avenue
222 Hu Bin Road
Shanghai, 200021
China
Tel: 86.21.5150.3000
Fax: 86.21.5150.3222

Printed in the U.S.A. M02000391-XX-4/05

ANALOG DEVICES

www.analog.com/analogdialogue