Design How-To

Fixed vs. floating point: a surprisingly hard choice

Boris Lerner, Analog Devices

2/6/2007 08:00 AM EST

Post a comment

Professional audio effects processor

Now we come to an application where the choice is not quite so clear. There is no hard limit on power consumption as in case of mobile applications—audio effects processors usually plug into a wall outlet. Unless the processor is so hot that it will require forced cooling (a very expensive proposition), the discussion of power comes down to the cost of the power supply.

Let's say, for example, that the effects processor we are trying to build has to do a La Scala reverb for a two-channel system. One would go to the La Scala opera house and measure its impulse response long enough for the final echo to become inaudible. Or one can cheat and download the already measured impulse response from a web site. I did and the impulse response was about 2 seconds long. Presuming a 96 KHz sample rate, this

translates into 192000 samples in the FIR delay line. (The real-world sample rate can be 48 KHz or 192 KHz, so I chose the middle ground.) Thus, to have a precise implementation of the reverb, we would need to implement a 192000 tap FIR.

Of course, doing a 192000 tap FIR directly is pretty much nuts—this would require 192000 multiplies per output sample. At 96000 samples per second, this would mean doing over 18 billion multiplies per second. Multiply this by two to account for the two output channels, add in any additional processing that needs to be done, allow headroom for future expansion, and we are easily talking about 100 billion MACs per second! One could do this by using 100 processors in parallel, but the resulting selling price of the box (and its size) would have a negative effect on its sales, to say the least. Thus, some optimization is necessary and the optimization we choose depends on the choice of the processor.

Let's consider the floating-point implementation first. As mentioned before, floating-point processing is good for doing large FFTs, so we can implement the FIR in frequency domain. One algorithm commonly used for this is the overlap-add FFT. (This is basically a way to compute large FFTs from a set of smaller ones.) In this case we'll choose a 1024-point window for

smaller ones.) In this case we'll choose a 1024-point window for computing an overlap-add FFT.

A glance at ADI's web site shows ADSP-21367 SHARC can perform a 1024-point complex SIMD FFT in about 9200 cycles. Thus, to compute 1024 samples of outputs, one would have to do a 1024-point FFT, followed by 1024 complex multiplies, followed by a 1024-point IFFT. The complex multiplies take 2048 cycles on a SHARC processor. Thus, to compute 1024 outputs, one would need 9200+2048+9200 = 20448 cycles, i.e., about 20 cycles per output. Add in the zero padding required to perform an overlap-add FFT, accesses to external memory to bring in the pre-computed FFT of the impulse response, multiply everything by two to account for the two channels, and we are at about 100 cycles per output. At 96000 samples per second, this gives 9,600,000 cycles per second. In other words, this will consume only 10 MIPs of the SHARC's processing budget.

Now, let's examine the same task, but implemented in a fixed-point processor. Unfortunately, the FFT implementation above runs into major issues. A 1024-point FFT/IFFT combination has a gain of 1024, i.e., 10 bits. To avoid overflow, during the computation of the FFT/IFFT the signal will have to be shifted down by 10 bits. If we want our output to have 24-bit resolution, we would have to perform 34-bit operations. Since the A/D and D/A converters usually offer less than 24 bits of SNR, let's just use 32-bit operations to get 22-bit performance. A fixed point 16-bit processor has to do four multiplies—as well as a few shifts and adds—to compute a 32-bit multiply.

In case of the Blackfin processor, we need six cycles for every 32-bit multiply. As a result, the processing that required 10 MHz on a SHARC requires 60 MHz on a Blackfin. Surprisingly, Blackfin probably has the advantage at this point. Either process can easily handle the workload, so the decision comes down to other factors, such as power. At frequencies this low, the leakage power is a significant chunk of the overall power consumption, making the frequency difference less relevant. Blackfin has significantly lower leakage power, so it probably comes out ahead in this case. However, as the processing load goes up, the dynamic power component becomes more and more dominant. Before long, the factor of six differences in the frequency will give SHARC the advantage. In the extreme case, the Blackfin implementation runs out of MIPs altogether and SHARC becomes the only choice.

In a different approach, one could implement an approximation of the required reverb. This could be done by keeping only the dominant terms and zeroing out all the rest. Suppose that we want to keep 1% of the terms non-zero, making this a 1920 tap FIR. We can no longer use an FFT to do this filter, because FFT can not compute terms selectively— it must compute them all. Thus, we would implement this FIR directly. This means that 1920 multiplies per output sample are required to implement it. To keep 24-bit precision, we need to multiply this by 6. Blackfin can do two 16-bit multiplies per cycle, so divide by 2 to get the cycle count. All in all, this comes down to 5760 cycles per output. To get the total workload, multiply this by the number of channels (2), multiply again by the 96000 MHz sample rate, and allow some headroom for overhead and new features. Doing so, we arrive at 2 GHz operation. So, we zero 99.9% of the impulse response terms and do this at 200 MHz. The frequency is still much higher than the FFT-based solution we had with SHARC. Perhaps more importantly, the filter's performance begins to suffer when we zero out 99.9% of the terms.

In real life, a pro audio effects processor would have much more functionality than just a reverb. To fully understand the fixed/float tradeoffs, you would have to do this analysis for all pieces of the processing. This is not a simple task, but the reverb example makes it clear that a floating-point solution can burn less power than a fixed-point solution—in spite of the fact that datasheet power numbers suggest the opposite.

Automatic echo canceller in hands-free portable device
An automatic echo canceller often employs an FFT to compute

Top Comments of the Week

A Group of Programmers ...: A Group of Programmers is obviously a Kludge of Programmers. "During the office reorganization he was sorted and merged with a kludge...
Stargzer on A Charm of CPUs & a ...

Re: Not surprising really: Why work so hard in engineering school if the job will just get shipped overseas anyway? I hear that a lot.
MeasurementBlues on Engineers in America: ...

Continuing on with the same subject . . .: If I

correlation to the reference signal. You might think that the use of FFTs would make floating point the obviously superior choice. However, there is an important difference between the FFT of the previous example and the FFT here. In the previous example, the listener hears the result of the FFT/IFFT. Thus, we had to keep a very high quality SNR. In the echo canceller, the FFT is used only to compute the time delay of the echo—it does not actually modify the audio stream. Thus, you can get away with doing an FFT in 16-bit precision. This changes the cycle count analysis significantly.

It is also important to note that echo cancellers are often used in hands-free portable devices. This puts a hard limit on the power budget. A floating-point processor will simply have too much leakage power to be a viable candidate.

As we can see, in corner cases the choice between fixed- and floating-point is clear. However, in grayer areas the analysis can get pretty complicated. Things get even more complicated when you throw in other considerations such as hardware acceleration to offload processing, ease of programming (which affects time to market), maximum dI/dt (which affects the cost of the power supply decoupling), and board layout. However, it's worth doing this analysis and not just sticking with your preconceived notions about fixed- and floating-point processors. You may very well be surprised to discover that you've been using the wrong type of processor all along.

About the author

Boris Lerner is a senior DSP-applications engineer at Analog Devices. His areas of expertise are DSP and communications hardware, software, algorithms, signal integrity, FPGA design, board-level design, and debugging. He received a bachelor's degree and a doctorate in mathematics from the University of Colorado–Boulder and has a master's degree in electrical engineering from the University of Florida (Gainesville).

EMAIL THIS PRINT COMMENT

< PREVIOUS PAGE 2 / 2

More Related Links

Ooh! Vacuum Tubes Lit With Tri-Colored LEDs

Business for Engineers: Risky Engineering Behavior

Krzanich Confirms Intel Overestimated PC Chip Demand

Which License Should You Use for Your Software?

I Love Adafruit's RGB LCD Shield Kit

Comments

VIEW COMMENTS: **NEWEST FIRST** | OLDEST FIRST | THREADED VIEW

Be the first to post a comment regarding this story.