



Automatic Hardware Implementation of Digital Filters for an Audio Codec

by MediaTek Inc.

Engineering an audio codec filter chain requires a careful balance of performance, power, and size. Our group must design solutions that not only meet rigorous standards for signal-to-noise ratio (SNR) and total harmonic distortion (THD), but also minimize power consumption and the total area of silicon required on the chip.

Products Used

- MATLAB®
- Filter Design Toolbox™
- Filter Design HDL Coder™
- Signal Processing Toolbox™

We used to implement our designs by hand-writing Register Transfer Level (RTL) code. While this approach produces a relatively small chip area, it leads to long development times, and any subsequent changes to requirements can result in significant rework of the implementation. This approach also carries some risk from a business perspective: We often do not know how difficult it will be to place and route the design until the very end of the design process, when it is all but impossible to make changes and still meet our release date.

We have adopted a new approach, one in which we design in MATLAB® and use Filter Design HDL Coder™ to generate synthesizable RTL code. By connecting system design to silicon, this approach enables us to rapidly evaluate filter architectures and optimize for silicon area. It has reduced our RTL code development cycle from three months to less than two weeks. System modifications that used to take almost a month to complete can now be made in as little as three days.

Architecting an Audio Codec

Audio codec consists of two separate processing chains (Figure 1). The audio encoder provides the interface from the microphone to the digital signal processor (DSP). The audio decoder works in the opposite direction, converting signals from the DSP to data that is sent to the speaker.

A key technical challenge in the systems that we design is that both the analog-to-digital converter (ADC) that follows the microphone preamplifier and the digital-to-analog converter (DAC) that precedes the loudspeaker power amplifier must operate at the relatively high frequency of 6.5 MHz.

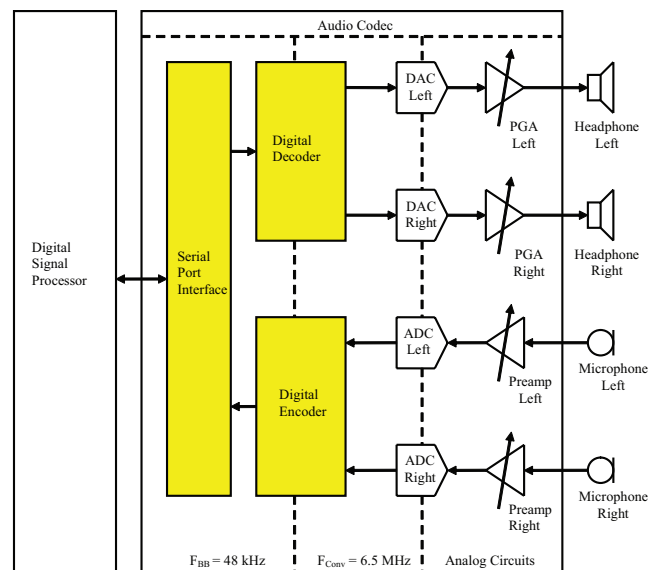


Figure 1. High-level block diagram of the audio codec.

On the other hand, the DSP processes data at a common rate of 48 kHz. The digital section of the audio codec converts between these two frequencies by applying a series of digital filters. The stereo encoder channel, for example, uses eight filters to decimate its input signal, while the stereo decoder uses nine filters to interpolate its input signal.

In the architecture phase of design, we use MATLAB to model the encoder and decoder channels. Using parameters provided by our analog designers, we model digital-to-analog and analog-to-digital converters and construct multirate digital filtering chains.

Next, we use MATLAB, Signal Processing Toolbox™, and Filter Design Toolbox™ to post-process the results produced by the model by calculating fast Fourier transforms (FFTs) and estimating SNR and THD. In this way, we can work out an optimized architecture before moving to implementation.

Implementation Using a Finite State Machine Sequencer

In our previous design methodology, we used a single multiplier-accumulator (MAC) and a finite state machine sequencer to implement the signal processing chain (Figure 2). The sequencer is responsible for addressing the RAM and ROM and controlling their operations. It also sets the input arguments for the MAC. The ROM stores filter coefficients, the RAM stores intermediate multiply-accumulate results, and the MAC performs the calculations based on the coefficients and data samples.

Traditionally, the sequencer was designed using a pencil-and-paper process that was complex and time-consuming. The handwritten RTL code was so inflexible that even small modifications proved difficult. On occasion, we also found that the RTL code for the sequencer finite state machine

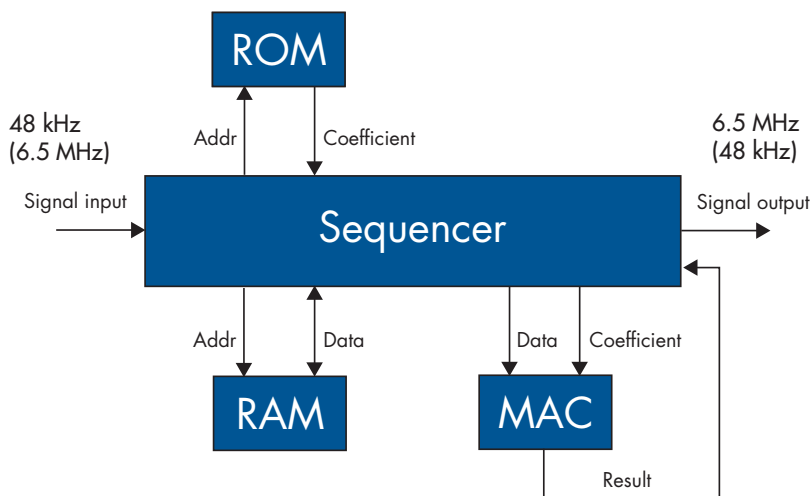


Figure 2. Block diagram of a traditional audio codec design.

was almost impossible to place and route. There were about 2,000 states in the finite state machine with more than 40 variables assigned in these states. This resulted in a very low area utilization ratio of 10% during placement and routing of the gate-level netlist. In other words, the proposed silicon area was 10 times greater than anticipated! It took a tremendous effort to synthesize the sequencer logic into a custom ROM memory. Yet still there was about 2x area penalty for the sequencer over our initial projection for the combinatorial logic (custom ROM usually needs more silicon area).

Automating Implementation

In our new design methodology, we have eliminated the complex sequencer. Instead, we use a series of digital filters designed with MATLAB and Filter Design Toolbox and implemented with Filter Design HDL Coder (Figure 3). Each digital filter block operates independently so that it can be easily modified, removed, or added to the overall chain.

The decoder chain includes four half-band FIR filters and a sample rate converter. Each filter interpolates by two, taking the initial 48 kHz signal to 96 kHz, 192 kHz, 384 kHz, and 768 kHz, respectively. A sample rate converter then converts directly from 768 kHz to the target frequency, 6.5 MHz.



Figure 3. The new design architecture, in which each filter is implemented independently using Filter Design HDL Coder. R1, R2, and Rn represent the rate change (interpolation/decimation) of the first, second, and nth filter in the chain.

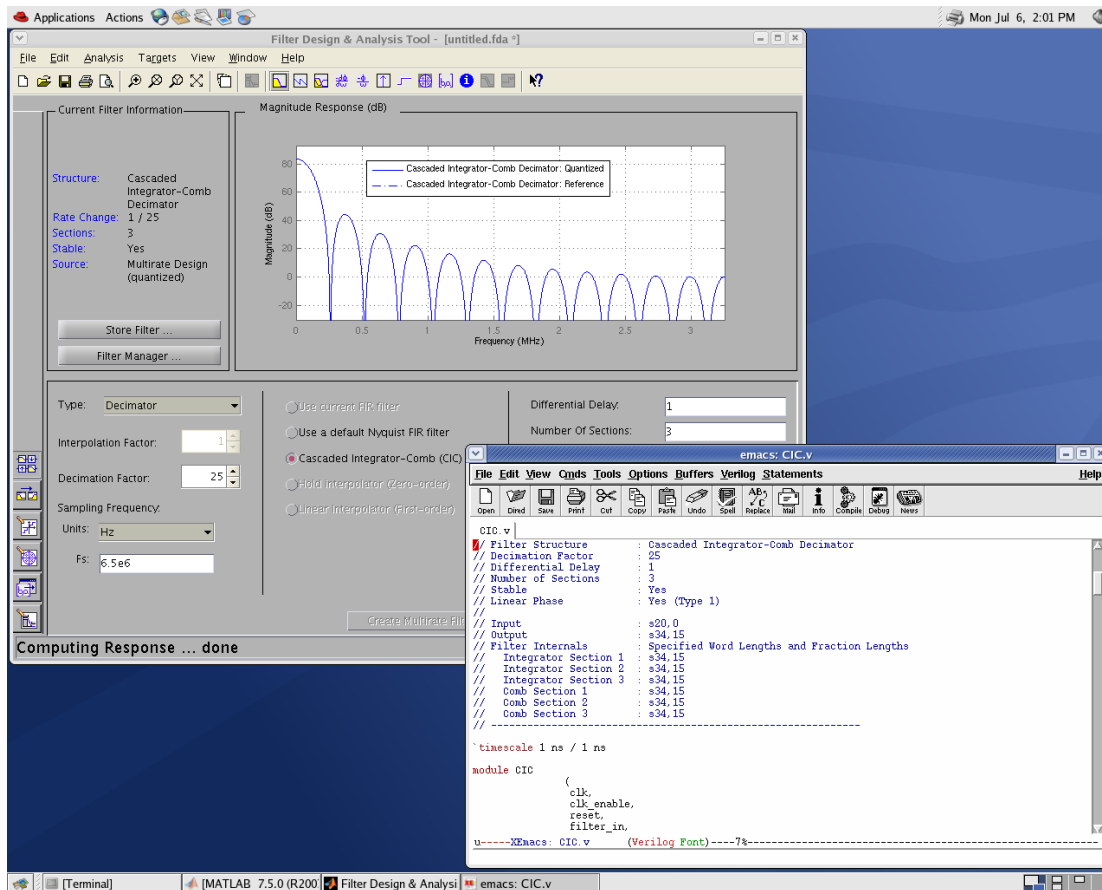


Figure 4. Filter Design HDL Coder interface showing part of the filter architecture and a segment of generated code.

The encoder chain includes two cascaded integrator-comb (CIC) decimators, as well as two halfband FIR filters. The first CIC decimator decimates by 25 from a channel input rate of 6.5 MHz down to 260 kHz. It is followed by a simple, hand-written interpolate-by-48 zero stuffer. Interpolated data at a rate of 12.48 MHz is then subjected to decimation by 65 within the second CIC decimator. It is later processed at a rate of 192 kHz within first halfband FIR decimation-by-2 filter. Finally, the second halfband decimation filter downsamples data from 96 kHz to the channel output rate of 48 kHz. In the case of the encoder channel, Filter Design Toolbox

simplified the CIC design by enabling us to rapidly evaluate numerous design options, such as the number of bits to use.

Generating RTL Code and Optimizing for Area

After designing the individual filters in the encoder and decoder chains, we generated Verilog code for each filter using Filter Design HDL Coder (Figure 4). At this stage, we could begin optimizing the implementation to minimize the silicon area of the design.

We tried several Filter Design HDL Coder optimization and architecture options for RTL generation. For example, to optimize the

halfband FIR filters, we first tried an option that produces a fully parallel architecture in which the filter clock rate is the same as the data rate. We then used Synopsys® Design Compiler to synthesize the code and report on the area allotted. The distributed arithmetic option in Filter Design HDL Coder, which required a clock rate 16 to 20 times higher than its data rate, produced a design that used about 25% of the area of the fully parallel design. Since we have a sufficiently high frequency clock already available on our chip, this was the best choice.

Verifying the Implementation

We used MATLAB scripts, RTL test benches, and Verilog simulations to verify the RTL implementation. In fact, we reused many of the MATLAB scripts developed in the architecture phase to generate stimulus signals and post-process the results by plotting FFTs and calculating SNR and THD.

After this first round of verification, we passed the RTL code to our colleagues for synthesis. The place-and-route step of the synthesis produces timing data in an SDF (standard delay format) file. We included this timing data in another round of gate-level simulations to ensure that there were no race conditions in the logic. The design was then sent off for fabrication.

Tests of the initial chip run uncovered no problems whatsoever with the digital portion of the audio codec. This meant that our team was free to lend a hand with the analog parts of the design and with the rest of the testing effort.

The Effect of Newer Fabrication Technology

Changes in fabrication technology are shifting the balance of analog and digital components on the chips that we design. While analog transistors must maintain a certain size to drive their required loads, digital transistors shrink when a new fabrication process is introduced. When migrating from a 0.25 μ m to a 45nm fabrication process, for example, the overall area of a mixed-signal audio codec went from 50% digital and 50% analog to 25% digital and 75% analog.

While our new design methodology requires more silicon area than our traditional methodology, reductions in the size of digital components in the new fabrication processes offsets this increase while allowing us to retain substantial development time savings.

In the future, it might be possible to further optimize our design process by using other low-area architectures offered by Filter Design HDL Coder, such as partly serial architectures, or by implementing the single sequencer architecture with automatically generated code. ■

Resources

VISIT

www.mathworks.com

TECHNICAL SUPPORT

www.mathworks.com/support

ONLINE USER COMMUNITY

www.mathworks.com/matlabcentral

DEMOS

www.mathworks.com/demos

TRAINING SERVICES

www.mathworks.com/training

THIRD-PARTY PRODUCTS AND SERVICES

www.mathworks.com/connections

Worldwide CONTACTS

www.mathworks.com/contact

E-MAIL

info@mathworks.com

© 2009 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

91761v00 09/09

For More Information

- **MediaTek**
www.mediatek.com
- **MathWorks Products for Digital Signal Processing**
www.mathworks.com/dsp