

PC-BASED PROTOTYPING OF AUDIO APPLICATIONS USING MODEL-BASED DESIGN

MARK CORLESS¹ AND ARVIND ANANTHAN²

¹ *Principal Application Engineer, The MathWorks, Michigan, USA*

Mark.Corless@mathworks.com

² *Technical Marketing, The MathWorks, Massachusetts, USA*

Arvind.Ananthan@mathworks.com

Personal computers (PCs) are increasingly being used as the primary development environment for creating, designing, and simulating audio algorithms and complete audio systems with live inputs from multichannel audio devices. In this paper, we show how audio algorithms can be designed and simulated in a textual and graphical modeling environment for a PC. These models interface with typical multichannel audio devices through PortAudio which enables communication with standard audio interfaces, such as Direct Sound, WDM-KS, and ASIO. The paper further delves into some typical challenges that engineers face when working with live multichannel audio algorithm simulation, namely, channel synchronization, dropped frames, and latency issues. We also demonstrate a technique for minimizing and measuring roundtrip latency. We used three sets of audio device hardware for our experiments—Behringer UCA202, M-Audio Delta 66, and M-Audio Firewire 410—to stream live audio through the simulation model. The example used throughout this paper is an automatic gain control algorithm which is modeled using a combination of Simulink, Signal Processing Blockset, Stateflow, and Embedded MATLAB code. Prototyping in this environment can enable designers to explore ideas and verify the correctness of their design early in the design cycle, thus reducing design iterations during the final stages where problems are typically more expensive to resolve.

INTRODUCTION

Personal computers (PCs) are commonly used to develop and prototype audio applications. Designers leverage application program interfaces (APIs), such as PortAudio, that hide the complexities of interfacing with audio devices and simplify development of audio applications [1]. Designers commonly enable PortAudio to communicate with interfaces like Windows Driver Model-Kernel Streaming (WDM-KS) or Audio Streaming Input Output (ASIO) to efficiently stream synchronized, low-latency, multichannel audio using their PCs [2].

Many companies that develop products related to signal processing apply Model-Based Design to identify issues early in the development process and reduce the time-to-market for embedded systems development [3]. Some aspects of Model-Based Design include creating simulation models to specify and explore functional behavior, implementing these specifications through C-code generation, and continuously testing and verifying the design against requirements. When applying Model-Based Design, designers often use tools such as MATLAB, Simulink, and Stateflow. Example applications of these tools include:

- home-theater surround-sound systems [4]
- echo and noise cancellers for hands-free systems [5]
- audio playback mechanisms [6] for automotive radios
- digital audio broadcasting (DAB) receivers [7]

Tools for Model-Based Design can be used to prototype audio applications on a PC by leveraging PortAudio with interfaces such as ASIO. In this paper, we demonstrate a PC platform for prototyping audio applications. We provide an example of modeling an audio algorithm that involves streaming live audio through the model to enable real-time tuning, and then we discuss typical concerns such as latency and dropped frames. Latencies are compared for a stereo Behringer UCA202 audio card and a multichannel M-Audio Firewire 410 card.

1 MODELING AN AUDIO ALGORITHM

Audio algorithms can be expressed using a variety of techniques that depend on the type of algorithm, level of complexity, and intended audience for the specification. Signal flow and state flow of algorithms are often best expressed graphically, which facilitates their review by a wide audience. Textual specifications are often

preferred for specifying low levels of detail. The mix of graphical and textual specifications depends highly on how the designer prefers to visualize and communicate ideas.

An effective prototyping platform should be flexible enough to graphically specify signal flow and state logic, as well as enable the designer to textually specify the algorithm. We will demonstrate this flexibility in algorithm specification using an automatic gain control (AGC) example that features a combination of Simulink, Stateflow, and Embedded MATLAB code.

AGC is commonly used in audio applications. For example, in digital still cameras sound is recorded along with captured video frames. AGC on the audio is required to amplify speech segments to an intelligible sound level and avoid amplifying noise only segments [1]. Similarly, in Voice over Internet Protocol (VoIP) applications, where audio levels can vary significantly depending upon the environment and user setting, AGC is used to adjust the level of an audio signal to achieve consistent and comfortable audio levels [9]. Although AGC is a common algorithmic component, designers continue to build upon fundamental techniques and experiment with new techniques to improve the behavioral performance while maintaining an efficient implementation.

1.1 Specifying Behavior Using Signal Flow

Simulink and Signal Processing Blockset [10] were selected to graphically specify the high-level signal flow of the algorithm. Graphically specifying the signal flow enables designers to create a high-level executable specification similar to diagrams commonly found in text books and papers. Graphical signal flow specification is often a preferred starting point for high-level design reviews with management or multiple teams. Figure 1 shows the Simulink high-level signal-flow specification for our AGC algorithm example; the input signal flows through a series of blocks: Estimate Level, Detect Signal, Follow Level, and Map Level to Gain to dynamically calculate and apply a gain.

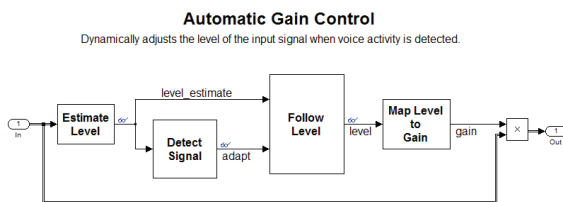


Figure 1: Signal flow for Automatic Gain Control model.

The Estimate Level subsystem, shown in Figure 2, is also specified using blocks to represent signal flow. A

simple technique of selecting the maximum absolute value within the input frame is used to estimate the level.

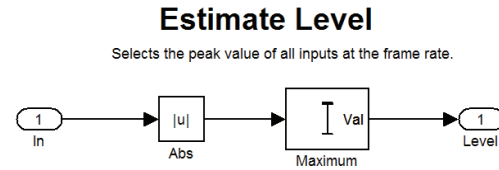


Figure 2: Estimate Level signal flow.

The Map Level to Gain subsystem, shown in Figure 3, is also specified using signal flow. A lookup table is used to specify gain shaping and then a gain is applied that can be used to adjust the overall reference level.

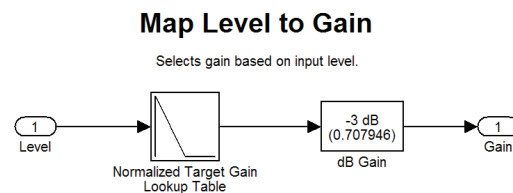


Figure 3: Map Level to Gain specification.

1.2 Specifying Behavior Using States

Some algorithms are best represented using constructs such as states and transitions. Stateflow enables graphical representation of hierarchical and parallel states and the transitions between them [10]. In this AGC example, the Detect Signal subsystem determines when a voice signal is present and outputs an adapt signal indicating whether to update the gain calculation. Since a simple detection mechanism can be thought of as two states (signal detected and no signal detected), we specified the behavior in Stateflow, as shown in Figure 4.

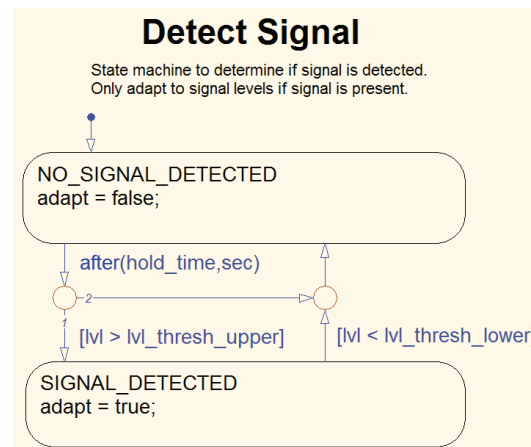


Figure 4: Detect Signal state specification.

The detector moves between states depending on the level of the input signal with respect to the upper and lower thresholds. Additional hysteresis is achieved by using the `after` syntax, which specifies how many seconds to stay in the no-signal detected state before moving into the signal-detected state.

1.3 Specifying Behavior Using a Textual Programming Language

Often audio algorithm designers are used to programming in textual languages like C and MATLAB. Specification of control flow such as nested if-then-else constructs can be specified graphically, but the designer may prefer to specify them textually. The Level Follower is an example of a simple nested if-then-else flow used to track the peaks of the estimated level signal based on an adapt indicator. We chose Embedded MATLAB code [10] to specify the Level Follower, as shown in Figure 5. Embedded MATLAB, a subset of the MATLAB language, supports accelerated simulations as well as C-code generation.

```
function level = follow_level(level_estimate,...
                             adapt, atk, dcy)
% atk = Attack rate, typically close to 1
% dcy = Decay rate, typically much smaller than atk
persistent level_n1
if isempty(level_n1)
    level_n1 = 0;
end

if adapt
    err = level_estimate - level_n1;
    if err > 0
        atk_dcy = atk; % Attack (quickly)
    else
        atk_dcy = dcy; % Decay (slowly)
    end
    level = level_n1 + err * atk_dcy;
else
    level = level_n1;
end

level_n1 = level;
```

Figure 5: Level Follower textual specification.

Simulink, Stateflow, and Embedded MATLAB code all support integration of custom and legacy C code into the simulation environment; this support is not demonstrated in the example. Integrating C-callable routines enables designers to reuse existing intellectual property during the development process as well as integrate C-callable libraries of hardware optimized routines.

1.4 Assessing Behavior

To assess the behavior of the AGC algorithm, we created a test bench model shown in Figure 6. The test bench model uses a Wave file as an input, feeds it through the AGC, and records the results.

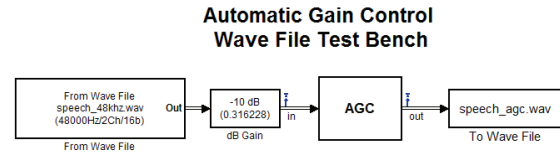


Figure 6: AGC Wave File Test Bench.

The `level_estimate`, `level`, and `adapt` signals within the AGC algorithm are attached to a Simulink scope, indicated by the small eyeglass icon that appears next to the corresponding signal line in Figure 1. Figure 7 shows a screen shot of the Simulink scope plot. We used this visualization to help verify that algorithm internals were operating as expected.

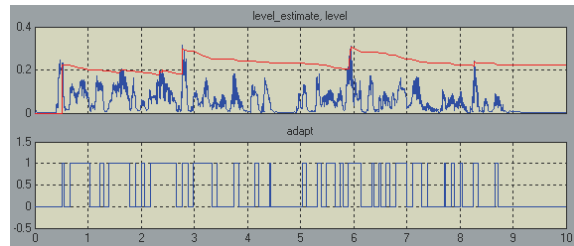


Figure 7: Scope Plot of AGC intermediate signals.

We also used MATLAB to post-process the simulation results. For example, the `in` and `out` signals of the test bench were logged to the MATLAB workspace, which is indicated by the antenna icons associated with these signals in Figure 6. We then wrote a MATLAB script to plot and format the results shown in Figure 8.

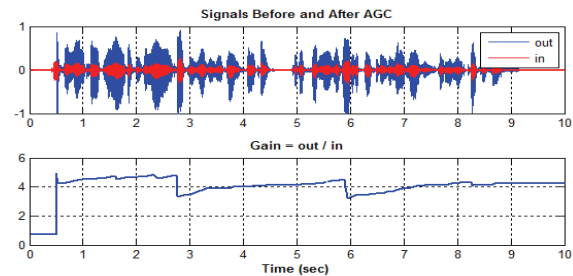


Figure 8: MATLAB plot of logged signals.

With this platform, the AGC example demonstrates how designers can specify their ideas in a variety of techniques, including signal flow, state-based, and textual approaches. Simulation provides insight into the algorithm behavior by enabling the designer to monitor internal signals as well as log signals for post-analysis. These results can then be used as a baseline for exploring other enhancements to the algorithm, such as more intelligent ways to detect voice signals or explore the effect of different gain curves.

2 STREAMING AUDIO THROUGH MODELS

Once the basic structure of the algorithm is understood through simulation, it is often desirable to dynamically tune and assess the impact of parameter changes by listening to the simulation in real time. This platform uses the From Audio Device and To Audio Device blocks, which are provided with the Signal Processing Blockset, to support streaming through an audio device. Together, these Audio Device blocks enable designers to stream audio through the Simulink environment to prototype audio applications.

In the following sections, we provide an overview of connecting to an audio device and provide examples of streaming audio through a model, ensuring multichannel synchronization and tuning model parameters during real-time simulation.

2.1 Interfacing with an Audio Device

Designers can choose from many sizes, types, and formats of audio devices to develop PC audio applications. Designers select an audio device based on sample rates (i.e., 44.1 kHz, 48 kHz, and 96 kHz), and signal types (i.e., analog, S/PDIF, and ADAT), as well as choose from a variety of PC interfaces (i.e., PCI/PCMCIA, USB, and Firewire) [11].

Most audio devices come with a DirectSound interface; higher performance audio devices come with ASIO and/or WDM-KS interfaces [12]. A key differentiator between these interfaces is achievable latency. ASIO typically has the least latency, WDM-KS often has comparable latency to ASIO, and DirectSound has the most latency.

The PortAudio API enables designers to access this wide variety of audio devices using a standard interface. Because the Audio Device blocks use PortAudio to communicate with audio devices, they can support a wide variety of choices. By default, the Audio Device blocks use PortAudio compiled to use the DirectSound interface, as almost all PC audio devices provide support for this. To take advantage of latency improvements gained by using WDM-KS or ASIO, designers can recompile PortAudio to access these and other interfaces [16].

We used three audio devices for our experiments to stream live audio through the simulation model: Behringer UCA202 [13], M-Audio Delta 66 [14], and Firewire 410 [15]. We investigated the default PortAudio configuration for DirectSound, as well as recompiled PortAudio to support ASIO and WDM-KS.

2.2 Streaming Audio out of a Model

We created a test-bench model to read from a Wave file source and stream the output using the To Audio Device block, as shown in Figure 9. We also added a manual switch block that lets us select whether to listen to the original source or the output of the AGC. This configuration enabled us to listen to the output as well as continue watching the internal signal activity on the Simulink scope.

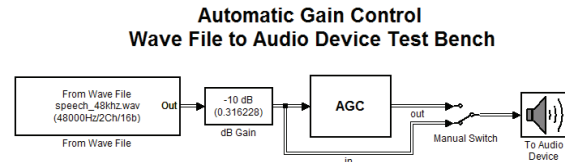


Figure 9: AGC test bench for Wave file to audio device.

When we were satisfied with the performance testing against Wave file inputs, we replaced the Wave file source with a From Audio Device block as shown in Figure 10. This configuration allowed us to verify performance against an external audio source.

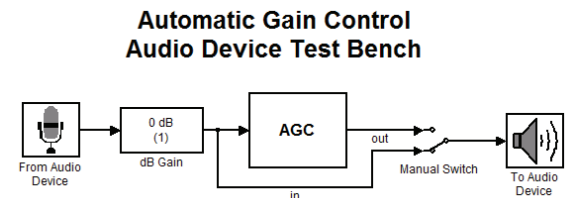


Figure 10: AGC Audio Device Test Bench.

2.3 Ensuring Multichannel Synchronization

In the previous tests, the inputs and outputs were both stereo signals. When dealing with additional output channels, PortAudio may provide the Audio Device block with multiple choices. Typically these choices include stereo pairs as well as passing all signals through a single port. For example, the M-Audio Firewire 410 exposes options for selecting individual pairs (i.e., Line 1/2, Line 3/4, Line 5/6, and Line 7/8) as well as a *Multi* option for all outputs.

A designer could use multiple To Audio Device blocks in a model to route different stereo channels to different audio device channels (Lines). This design is fine if the channels of stereo data are unrelated; if they are related, as in a Dolby 5.1 system, signals should be routed to the output using the Multi option to accommodate how Windows opens device drivers. If channels are specified using multiple To Audio Device blocks, additional delay will be introduced between the channels at the driver level [17]. Such an additional delay between channels can result in an undesirable shift of the sound

image [20]. An example modeling pattern for synchronizing output channels is shown in Figure 11. In this example, the input is 2 channels with a frame size of 128 samples. The signal is passed through a Matrix Concatenation block that packs the signals into 8 channels of data which will be sent to the audio device. This pattern ensures that all 8 signals will be synchronized at the output.

Synchronized Output Channels 2 Input, 8 Output

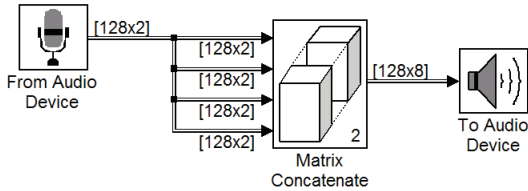


Figure 11: Pattern for synchronizing output channels.

3 TUNING PARAMETERS IN REAL TIME

Often the primary motivation for streaming audio through the model is to enable interactive real-time tuning of algorithm parameters. In this section, we discuss graphical and programmatic mechanisms for interfacing with the simulation while streaming audio.

3.1 Changing Parameters Using Block Dialogs

Many of the Simulink block parameters are tunable. A tunable parameter is a parameter whose value can be changed without recompiling the model [18]. For example, the Gain parameter of the dB Gain block, shown in Figure 12, is tunable. The Gain value can be altered while a simulation is running. If a parameter is not tunable and the simulation is running, the dialog box control that sets the parameter is disabled.

When testing the AGC algorithm, we adjusted the dB Gain of the Map Level to Gain subsystem and listened to the resulting change in amplitude.

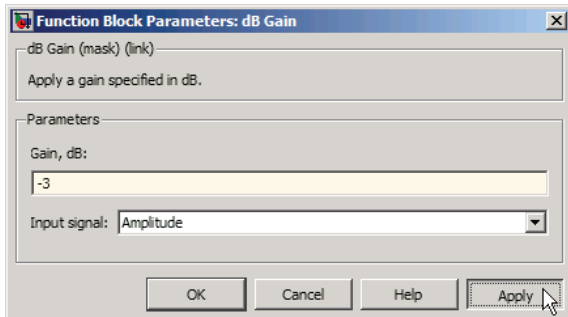


Figure 12: dB Gain block dialog

3.2 Changing Parameters Using MATLAB Variables

Often it is convenient for algorithm parameters to be expressed as MATLAB variables. For example, the Detect Signal subsystem specifies the hold time to go from no signal detected to signal detected as the variable `hold_time`. During simulation, `hold_time` can be programmatically modified in the MATLAB Command Window, as shown in Figure 13. This change can be applied to the Simulink diagram by pressing the Update Diagram button in the Simulink Toolbar.

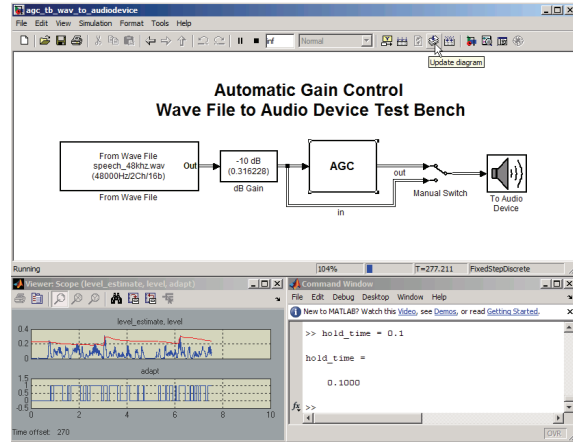


Figure 13: Adjusting the `hold_time` variable.

This programmatic approach can be extended to synchronize changes of several parameters. For example, Figure 14 shows a MATLAB script that we used to adjust parameters from the Detect Signal and Follow Level subsystems of the AGC algorithm. The last call in the script is to the `set_param` function that programmatically updates the Simulink model. This action is equivalent to manually pressing the Update Diagram button.

```

%% Specify Parameters
hold_time = 0.6; % Signal Detector: Hold time (seconds)
atk = 0.91; % Follow Level: Attack coefficient
dcy = 0.03; % Follow Level: Decay coefficient

%% Update Simulink Diagram with Changes
set_param('agc_tb_wav_to_audiodevice',...
    'SimulationCommand','update')

```

Figure 14: Script that programmatically updates a diagram.

In this section we have introduced the basic techniques for changing parameters during simulation. Designers can build upon these techniques to create additional interfaces for tuning parameters. For example, designers can programmatically create a custom user interface in MATLAB that enables users to interact with the model during simulation [19].

4 PREVENTING DROPPED FRAMES

For our simple AGC algorithm we were able to successfully stream audio using the default configurations of our system without dropping frames of data. As the size and complexity of the algorithm increases, it is possible that a PC will not be able to keep up with reading and writing to the audio device buffers, resulting in dropped frames. Dropped frames manifest themselves as audible clicks, pops, or choppy audio. In this section we discuss some basic concepts that designers should keep in mind to improve throughput performance of their system and prevent dropped frames.

4.1 Managing Windows Processes

Simulink is typically just one of many processes running in the Windows Operating System. These processes compete for CPU usage. PC audio application designers often turn off background tasks (i.e., backup programmers or virus checkers) and disconnect network connections [21]. We used the Windows Task Manager to assess the impact of other processes on CPU usage. As the impact of other processes may be sporadic, we commonly left these processes running for personal development, but disabled things like network connections during demonstrations of larger algorithms.

4.2 Improving Simulink Simulation Speed

When modeling algorithms in Simulink, designers should keep in mind some common techniques for improved simulation speed that also improve throughput. Techniques for improving simulation speed include using frames, restricting multirate models to rates that are integer multiples of each other, and minimizing visualizations [21][23][24].

In our tests, we saw a large reduction in processor usage by reducing the update rate of visualizations. By default, visualization blocks, such as the Spectrum Scope and Display blocks, are configured to update at the rate of the signal entering the block. Most visualization blocks provide a setting to reduce the update rate, thus reducing processing time required to render the visualization. We found that a lower update rate was often sufficient to gain insight while streaming live audio. An example of a model that contains Spectrum Scope and Display blocks configured to update at rates less than the signal entering them is shown in Figure 15. The update rate of the Display block is controlled by the Decimation parameter in the block dialog. The update rate of the Spectrum Scope block is dependent upon its Buffer size parameter. Increasing the Buffer size reduces the update rate and increases the resolution of the visualization. Here the Buffer size parameter is set to 4096, so it updates 16 times slower than the default size when connected to the input framed signal of 256 samples.

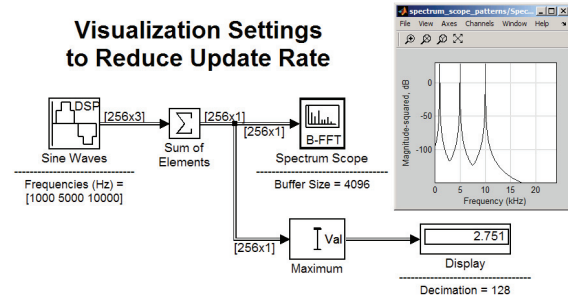


Figure 15: Reducing visualization update rates.

4.3 Configuring Audio Device Buffers

Buffering of audio data is a common technique to maintain a continuous audio stream in PC-based audio applications [12]. Increasing the size of these buffers addresses the majority of problems associated with clicks and pops, but the tradeoff is that additional latency is introduced and any change that the user makes at run time does not take effect until the next buffer is accessed.

The Audio Device blocks in Simulink enable the designer to specify the buffer size used to communicate with the audio device as well as the queue duration, which is interposed between the Audio Device block and the audio device. Increasing the size of these buffers will enable the blocks to absorb peaks in CPU performance. In our experiments, the default buffer settings were sufficient for tuning algorithms in real-time without dropping frames.

5 REDUCING LATENCY

The amount of tolerable latency is highly dependent upon the application for which the algorithm is being developed. For example, tolerable delay between video and audio can be up to 45 msec [25], while the tolerable delays for monitoring audio during audio production range from 3 to 23 msec [12]. Another form of latency is the time it takes to hear a response to tuning parameters. In this case 100 msec is a common tolerable latency [12].

In the following section we discuss some techniques to minimize the roundtrip latency for prototyping audio applications. We also discuss how to configure the audio device in Windows, as well as the Audio Device blocks in the model. We then discuss a method for measuring latency and we provide a table of measured latencies for different PC and Audio Device combinations.

5.1 Configuring the Audio Device in Windows

As the driver for the audio device plays a large role in achieving reduced latency, we investigated the performance of the audio device independently in the Windows environment before attempting to measure latency in the Simulink environment.

Typically a higher-performance audio device will provide a control panel to configure the device. For example, the Behringer UCA202 provided an ASIO-USB control panel that contained a system performance field. This field allowed selection of such options as Highspeed, Rapid, Fast, and Normal. Each of these options would affect the expected latency.

To help identify the most appropriate for our test, we used the CEntrance ASIO Latency Test Utility [26] to measure latency for different configurations. For example, on one laptop we chose the Rapid option as it provided repeatable latency measurements of 12.65 msec for a frame size of 128 samples. Thus, we established that this would be the best possible latency that our laptop setup could hope to achieve.

5.2 Configuring the Audio Device Blocks for Minimal Latency

As described previously, Audio Device blocks are implemented by using a multiple buffer scheme that can help prevent dropped frames during streaming of audio. We found that when using the DirectSound interface, buffers were required to stream audio. However, when we switched to the WDM-KS or ASIO interface, we could significantly reduce the size of these buffers, which reduces the overall latency.

An example configuration of the From Audio Device block for the Behringer UCA202 card is shown in Figure 16. To achieve minimal latency, we configured the Queue duration to be zero, and the Buffer size and Frame size parameters to the lowest power of 2 that was sufficient to prevent dropped frames. For our setup using the Behringer USB 202, we set the Buffer size and Frame size parameters to 128, which also represents the lowest value that we could use with the CEntrance application to measure latency without dropped frames. It is also important to note that we used release R2009a of MATLAB and its associated products to run these tests. With the previous versions of MATLAB, it may be necessary to specify Queue duration to be greater than zero.

To ensure minimal computation time within the Audio Device block, the Device data type should be set to that of the audio device and the Output data type should be the smallest integer that can contain the device data

type. We configured the To Audio Device block in a similar manner.

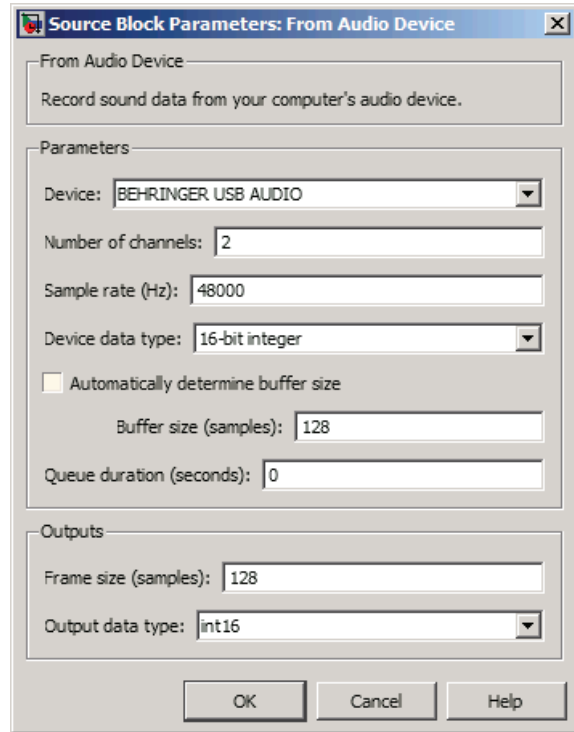


Figure 16: Parameters associated with the From Audio Device block.

5.3 Measuring Latency

To explore achievable latencies, we physically connected the output to the input of the audio device. We then created a Simulink model to stream 48-kHz audio through the audio device as shown in Figure 17. We created a test input variable in the MATLAB workspace consisting of a small 6-kHz sine wave summed with four periods of a large 1.5-kHz sine wave. Applying this type of signal made it easy to verify that no frames were dropped and to calculate the roundtrip delay between the input and the output.



Figure 17: Streaming 48-kHz audio through an audio device.

After running the test, we created a MATLAB script to calculate the roundtrip delay using a cross-correlation of the input and output. An excerpt of this script is shown in Figure 18.

```

in = double(input);
out = double(wavread('output.wav', 'native'));
c = xcorr(out, in);
index_center = (length(c)+1) ./2;
[C, index_max] = max(c);
delay = index_max - index_center;

```

Figure 18: Script to calculate roundtrip delay.

We also used MATLAB to plot the results so that we could visually verify that no frames of data were dropped during the test. An example plot of results for testing the M-Audio Delta 66 is shown in Figure 19.

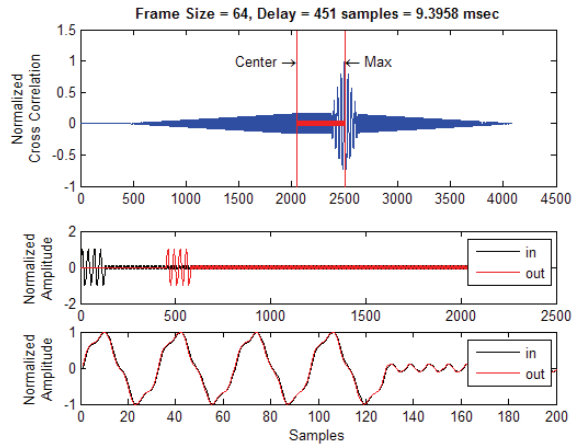


Figure 19: Plot of latency results for M-Audio Delta 66 using WDM-KS.

We repeated this test for several different PC and audio device combinations. The results of these tests are shown in Table 1, “Comparison of Measured Roundtrip Latencies at 48-kHz Sample Rate.” In general, for ASIO, the latency was about two frames more than the latency measured using the CEntrance utility. We attribute this to buffers required for the Audio Device blocks to interface with the audio device. Our tests also showed that using a 32-bit PC running Windows XP enabled us to explore different buffer sizes, but when we tested the M-Audio Firewire 410 on a 64-bit bit PC running Windows Vista, the buffer size was fixed to 256 bits. As 256 bits was also the only buffer size exposed in the CEntrance latency utility, this value appears to be a current limitation of the Windows Vista operating system.

We found that this platform is capable of achieving roundtrip delays as low as 6.73 msec. This performance is suitable for developing many audio applications. This platform can also be extended to support code generation for deployment to common processors. In applications requiring even lower latency, designers can take advantage of code generation technologies that enable prototyping of models on embedded digital signal processors [27][28][29].

Table 1: Comparison of Measured Round Trip Latencies at 48-kHz Sample Rate

Audio Device	Interface	Buffer Size	Latency	
			Samples	msec
Behringer UCA 202 ¹	ASIO	128	786	16.375
M-Audio Delta-66 ²	ASIO	64	323	6.729
	WDM-KS	64	451	9.396
M-Audio Firewire 410	ASIO	256	1208	25.167
	WDM-KS	256	1472	30.667

¹ Test ran on a Lenovo T60 Laptop, 32-bit, Windows XP SP3, Intel Core Duo T2500 @ 2GHz, 2GB RAM
² Test ran on a PC, Windows XP SP3, 32-bit, Intel Core 2 6700 @ 2.67GHz, 3.25GB RAM
³ Test ran on a PC, Windows Vista, 64-bit

6 CONCLUSION

In this paper we demonstrated a platform for real-time prototyping audio algorithms on a PC.

This platform lets designers specify algorithms by using graphical signal flow, state flow and textual techniques. As an example, we specified a model of an automatic gain control algorithm using a combination of Simulink, Signal Processing Blockset, Stateflow, and Embedded MATLAB code. We also created a simulation test bench to analyze and visualize the behavior of the algorithm.

This platform enables designers to stream audio and interactively tune the algorithm in real time. To demonstrate these capabilities, we elaborated the automatic gain control exploration model to use a Wave file as an input and stream the audio out through an audio device. We continued evolving the model to stream inputs and outputs through an audio device. In both cases, we interactively adjusted algorithm parameters in real time and listened to the effects.

To effectively apply this approach to prototype audio applications, the designer must assess the impact of latency. We described a technique for measuring the roundtrip latency and compared latency measurements for several different PCs and audio devices. This platform uses PortAudio as the interface to the audio

device, so it is extensible to industry standard audio devices and interfaces, such as DirectSound, WDM-KS, and ASIO.

Prototyping in this environment enables designers to verify if their design is correct early in the design cycle, thus reducing design iterations during the final stages and thus reduce overall development time. Although not discussed in detail in this paper, this platform also provides a path to code generation for deployment on PCs as well as embedded processors.

REFERENCES

- [1] Bencina, R. & Burk, P., "PortAudio - an Open Source Cross Platform Audio API," Proceedings of the International Computer Music Conference, Havana, International Computer Music Association, pp.263-266., 2001.
- [2] Letz, Stephane, "Porting PortAudio API on ASIO," GRAME - Computer Music Research Lab Technical Note - 01-11-06, November 2001.
- [3] Corless, Mark, and Ananthan, Arvind, "Model-Based Design of Fixed-Point Filters for Embedded Systems," Society of Automotive Engineers World Congress 2009-01-0150, April 2009.
- [4] Philips Consumer Lifestyle (Philips) Develops One-Piece Surround Sound System with MathWorks Tools, The MathWorks, April 2008: www.mathworks.com/company/user_stories/user_story17418.html
- [5] Eslinger, Greg and Dixon, John, "Dynamic Adjustment of System Parameters Improves Echo and Noise Cancellation," Texas Instruments, December 2006: <http://focus.ti.com/lit/wp/spry094/spry094.pdf>
- [6] Schubert, Peter J., Vitkin, Lev, and Braun, David, "Model-Based Development for Event-Driven Applications using MATLAB: Audio Playback Case Study," Systems Engineering, January 2007: <http://delphi.com/pdf/techpapers/2007-01-0783.pdf>
- [7] Harman Becker Designs and Verifies OFDM Radio Receivers Using MathWorks Tools, The MathWorks, March 2009: http://www.mathworks.com/company/user_stories/userstory19636.html
- [8] Archibald, Fitzgerald J., "Software Implementation of Automatic Gain Controller for Speech Signal," Texas Instruments SPRAAL1, July 2008.
- [9] GIPS Automatic Gain Control, Global IP Solutions, http://www.softfront.co.jp/products/lib/SP_AGC.pdf
- [10] <http://www.mathworks.com>
- [11] Walker, Martin, "Choosing A PC Audio Interface: The SOS Guide," Sound On Sound, November 2004: http://www.soundonsound.com/sos/nov04/article_s/pcmusician.htm
- [12] Walker, Martin, "Optimising The Latency Of Your PC Audio Interface," Sound on Sound, January 2005: <http://www.soundonsound.com/sos/jan05/articles/pcmusician.htm>
- [13] <http://www.behringer.com/EN/Products/UCA202.aspx>
- [14] http://www.m-audio.com/products/en_us/Delta66.html
- [15] http://www.m-audio.com/products/en_us/FireWire410.html
- [16] Rogers, Alec, "Customized Audio Driver Support (Using ASIO, ALSA, etc. with Signal Processing Blockset Audio Device Blocks)," February 2008: www.mathworks.com/matlabcentral/fileexchange/18599
- [17] Schillebeeckx, P., Paterson-Stephens, I. and Wiggins, B., "Using MATLAB/Simulink as an implementation tool for Multichannel Surround Sound," Proceedings of the 19th International AES conference on Surround Sound, Germany, pp. 366-372, June 2001.
- [18] Simulink, Tunable Parameters, The MathWorks: <http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/index.html?access/helpdesk/help/toolbox/simulink/ug/f7-20739.html#f7-23615>
- [19] Fillyaw, Chris, Friedman, Jonathan, and Prabhu, Sameer M., "Creating Human Machine Interface (HMI) Based Tests within Model-Based

Design,” Society of Automotive Engineers 2007-01-0780, January 2007.

of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

- [20] Fonseca, Nuno and Monteiro, Edmundo, “Latency Issues in Audio Networks,” Audio Engineering Society 118th Convention, May 2005.
- [21] Burk, Phil, “Audio Latency,” 2002:
<http://www.portaudio.com/docs/latency.html>
- [22] Simulink, Improving Simulation Performance and Accuracy, The MathWorks:
<http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/index.html?/access/helpdesk/help/toolbox/simulink/ug/f11-33464.html>
- [23] Simulink, Performance & Memory Management Guide, The MathWorks:
<http://www.mathworks.com/support/tech-notes/1800/1806.html>
- [24] Recorded Webinar: Tips for Speeding up Simulink Models for Signal Processing & Communications, The MathWorks, February 2005:
<http://www.mathworks.com/company/events/webinars/wbnr30446.html>
- [25] ATSC Implementation Subcommittee Finding: Relative Timing of Sound and Vision for Broadcast Operations, Advanced Television Systems Committee, June 2003.
- [26] <http://www.centrance.com/products/ltu/>
- [27] Wilber, Jack, “The Sound of Innovation at Cochlear Limited,” The MathWorks News & Notes, October 2006:
http://www.mathworks.com/company/newsletters/news_notes/oct06/cochlear.html
- [28] Ananthan, Arvind, “Rapid-prototyping and Implementing Audio Algorithms on DSPs using Model Based Design and Automatic Code Generation,” Audio Engineering Society 123rd Convention, October 2007.
- [29] Ananthan, Arvind, “Integrating Simulink Model with VisualDSP++ Project,” November 2008:
www.mathworks.com/matlabcentral/fileexchange/22243

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list