

Model-Based Design of Fixed-Point Filters for Embedded Systems

Mark Corless and Arvind Ananthan
The MathWorks, Inc.

Copyright © 2009 The MathWorks, Inc. Published by SAE International with permission.

ABSTRACT

Digital filters are used in many automotive applications ranging from identification and conditioning of signals in an engine controller to digital radio receivers. Often these filters are implemented in fixed point for reasons such as throughput or cost. Selecting a fixed-point implementation requires trading off behavioral performance for available resources on the embedded processor. In addition, many of these filters must support calibration at processor startup or run time. Hence the selected implementation must meet the behavioral requirements for a set of digital filters. A common example application is audio processing to optimize cabin acoustics. In this case, the embedded audio processor queries the vehicle identification over the vehicle network then selects a bank of digital filters which are calibrated to provide optimal acoustics to the passengers.

This paper describes a workflow that applies Model-Based Design to develop an algorithm with selectable banks of fixed-point digital filters. In this workflow, the algorithm specification begins in floating point. Simulation test benches are then created to explore and verify the behavior. The algorithm is then converted to fixed-point in stages. The test benches are reused to verify correct behavior is maintained throughout the elaboration of the algorithm specification to fixed point. Automatic code generation is then applied to implement the algorithm in C code which takes advantage of processor specific intrinsic functions for fixed-point mathematics on an Analog Devices' Blackfin processor. The example workflow is described in the context of an acoustic tone controls application, but the approach can be applied to many applications requiring digital filters

which will be deployed to a fixed-point embedded processor.

INTRODUCTION

When developing applications containing fixed-point digital filters whose coefficients can vary over time, designers must carefully select filter structures and quantization scaling that ensures stable and correct system behavior. The negative impact of poorly designed filters ranges from unexpected sounds generated by speakers which can create dangerous driving scenarios by distracting or startling drivers, to incorrect detection of spark knock which can lead to destruction of engine components.

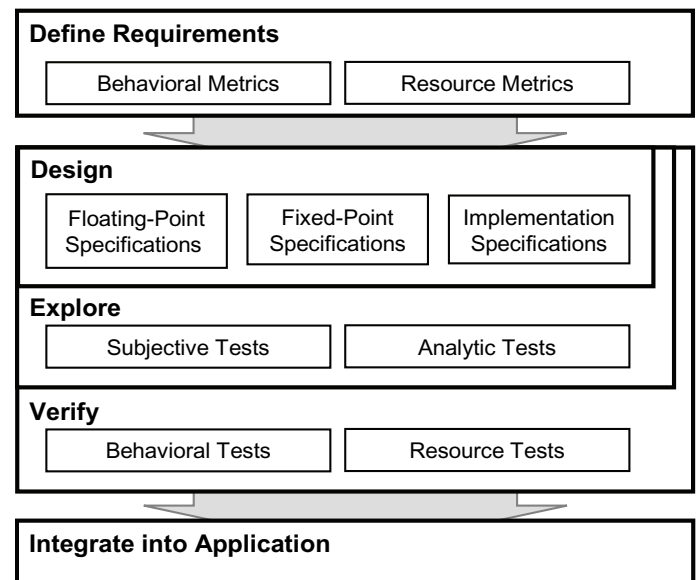


Figure 1. Typical algorithm development tasks.

This introduction will provide an overview of some typical development tasks and challenges required to prevent these types of failures in the field, followed by an overview of the tone controls algorithm which will be used to demonstrate development of a fixed-point audio algorithm using Model-Based Design.

AUDIO DSP DEVELOPMENT TASKS — The traditional DSP development process includes floating-point algorithm design with simulation, elaboration to fixed point, and implementation and evaluation in real time [1]. Additional development tasks are illustrated in **Figure 1**. Developing a DSP application often requires a combination of scripting, modeling, and code generation tools [1]. Designers are then challenged to apply these tools in a manner that minimizes effort and maximizes reuse throughout the development process.

Many automotive, aerospace, and communication companies apply Model-Based Design to identify issues early in the development process and reduce the time-to-market for embedded systems development. 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. Some example applications of Model-Based Design include development of engine knock reduction systems [2], automotive audio playback mechanisms [3], and cochlear implants to improve hearing [4]. Applying Model-Based Design to develop audio applications enables designers to address common development challenges such as analysis of fixed-point tradeoffs, exploration of system behavioral performance, and rapid prototyping innovations.

Analyze Fixed-Point Tradeoffs — Designing digital filters requires balancing dynamic range and signal-to-noise ratio (SNR). In an audio tone controls application, this often requires that designers sacrifice SNR to provide headroom, particularly in the case where some spectral portion of the audio band is being boosted [5]. In applications where digital filter coefficients vary over time, digital filter morphing techniques must be applied to reduce audible transient distortions, which can produce disturbing acoustic effects. [6,7,8]. Designers select filter structures, precisions, and scale ordering to improve behavioral performance and reduce these effects [9]. DSP designers often begin their algorithm development in MATLAB®, a high-level language and interactive environment, and extend upon this environment to verify their designs [10].

Explore System Behavioral Performance — The final fixed-point design will be dependent on the overall system performance. For example, the behavioral performance of an automotive hands-free phone application will be highly dependant upon its ability to quickly adapt to changes in the acoustic environment.

Algorithm and environment modeling are often specified in Simulink®, a platform for multidomain simulation of dynamic systems. Simulink provides an interactive graphical environment, a customizable set of block libraries, and can be extended for specialized applications such as signal processing. Designers of hands-free applications often model and simulate the performance of echo and noise cancellation in Simulink to explore a wide range of design choices and reduce the time and expense of the design process [11].

Prototype Innovations — Typically acoustic engineers and software engineers come together to create market-differentiating features. Commonly, acoustic engineers prefer a graphical interface to explore innovations. These ideas must then be transferred to the software development team to prototype an implementation. The MATLAB and Simulink environment can be extended to rapidly design and prototype audio applications [12]. To reduce the time to implementation, acoustic engineers can use Simulink to specify the flow of the algorithm, and then apply Real-Time Workshop Embedded Coder™ C code generation technology to quickly implement the algorithm in real time. For example, acoustic engineers can graphically specify the flow of a surround-sound audio application and automate the generation and integration of C code into a real-time prototyping environment. After the best algorithm solution is identified, is it implemented on a DSP and committed to production hardware [13].

APPLY MODEL-BASED DESIGN TO AN AUDIO TONE CONTROLS ALGORITHM — This paper describes how scripting, modeling, and code generation are combined to develop a tone controls algorithm. A focus is placed on effectively elaborating design specifications from floating point to fixed point, creating reusable exploration and verification test benches, and automating analysis and collection of resource metrics.

The tone controls algorithm is similar to what can be found in a typical automotive radio as a three-band equalizer. The user can adjust the amount of lower-frequency content (bass), mid-range frequency content (mid), and higher-frequency content (treble). The user can select from a set of filter responses to increase the frequency content (boost), decrease the frequency content (cut), or maintain unity gain across the band (detent).

This algorithm was selected because it is functionally straightforward to understand, yet requires the designer to make typical fixed-point filtering tradeoffs. The requirement that the response for each filter bank be user-selectable at run time introduces additional fixed-point dynamics whose affects can be assessed through simulation. Although first- and second-order curves are commonly used in tone control applications, we chose fourth-order curves for this example to highlight

quantization and scaling techniques that can be applied to higher order filters.

Model-Based Design was applied to develop this algorithm. A floating-point behaviorally correct model was specified, explored, and verified based on a set of requirements. The design was elaborated to include fixed-point specifications, and then implementation specifications before being integrated into a standalone application. The development tasks addressed in this example are shown in **Figure 1**. A final set of design specifications would typically be achieved through iterations of these development tasks. For readability, these development tasks are discussed sequentially.

DEFINE REQUIREMENTS

Algorithm development often begins with both behavioral and resource utilization requirements. These requirements guide the development process and provide measures to evaluate performance.

DEFINE BEHAVIORAL METRICS — Magnitude response, phase response, Signal-to-Noise Ratio (SNR), and Spurious Free Dynamic Range (SFDR) are common metrics to evaluate the behavioral performance of applications with digital filtering components. For this tone control example, the design engineer is given a set of 15 bass, mid, and treble curves that users can select. The desired curves for this example are shown in **Figure 2**.

Along with these reference magnitude response curves, there is typically a requirement metric that the measured response be within some delta from the reference. For this example, the final fixed-point magnitude response must not deviate from the reference response by more than 0.2 dB. There is also a subjective behavioral requirement that the levels of boost and cut can be varied at run time without audible clicks or pops.

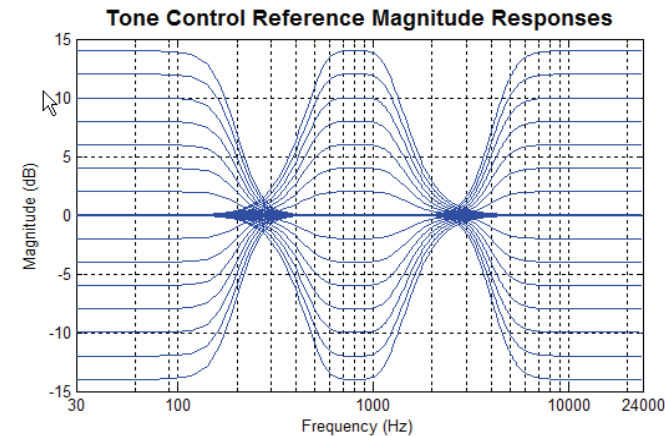


Figure 2. Tone control magnitude responses.

DEFINE RESOURCE METRICS — Resource requirements ensure that the algorithm implemented as a software component can be integrated into the larger application. Execution cycles, RAM/ROM usage, and stack usage are common metrics to evaluate resource usage on an embedded processor. In this example, the tone control algorithm is to be integrated into an application deployed on an Analog Devices’ Blackfin® processor.

The level of detail for behavioral and resource requirements will vary based on existing application knowledge. In many cases, requirements are incomplete, conflicting, or unfeasible. Some of the following sections will show how Model-Based Design can identify holes in requirements early in the development process.

DESIGN BEHAVIORAL FLOATING-POINT ALGORITHM

New design tasks often begin in a floating-point environment because it allows the designer to focus on the behavior of the algorithm separate from fixed-point quantization effects. Model-Based Design enables a designer to create an executable specification for the algorithm behavior in floating point. This executable specification will be used to assess the behavioral requirements as well as to provide a basis for further elaboration to a fixed-point design. The specification for the tone control algorithm consists of model and parameterized data elements as shown in **Figure 3**.

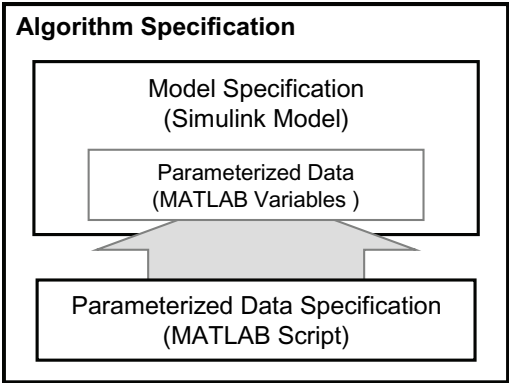


Figure 3. Algorithm specification.

SPECIFY ALGORITHM MODEL — To facilitate communication among different design teams, structural specifications are often expressed graphically while finer specifications may be expressed graphically or textually. Simulink® provides a graphical hierarchical environment to specify and simulate the design using models. Within Simulink, the designer can also specify models using state machines (Stateflow® software [14]) or textually (Embedded MATLAB™ subset [15]). In this example, the algorithm is graphically specified in Simulink.

A screen shot of the tone controls high-level model specification is shown in **Figure 4**. A frame-based stereo signal is passed through bass, mid, and treble tone controls. Filter selection signals are grouped as a multiplexed input to the model.

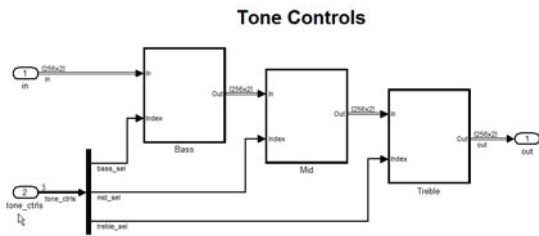


Figure 4. Tone controls, floating-point model.

The subsystems for each tone control contain further specifications for the flow of dynamically selected coefficients. The model specification of the Bass Tone Control is shown in **Figure 5**. The banks of filter coefficients, designed in the technical computing language MATLAB®, are stored in a matrix. The input selection signal is used to select the appropriate filter coefficients from the matrix. To address the requirement that there be no clicks or pops when switching between filters, the selected coefficient array is passed through a *Discrete Filter* block configured as a leaky integrator to smooth the transition between filter selections. The coefficients are then split into numerator and denominator coefficients and fed into a *Biquad Filter* block. The Mid and Treble subsystems are specified in a similar manner.

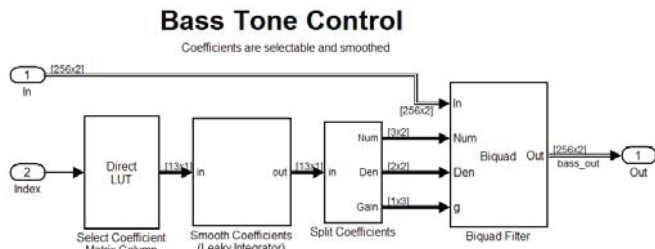


Figure 5. Bass Tone Control model.

SPECIFY ALGORITHM PARAMETERS — Algorithm parameters, such as filter coefficients are often designed using a technical computing language. A technical computing language enables designers to document the design flow of parameters in the algorithm. These parameters are then accessed by the graphical modeling and simulation environment.

In the tone controls example, the parameter design flow is specified in a MATLAB script which is automatically run at the start of the corresponding Simulink simulation. The filter banks are designed using the *parameq* function from Filter Design Toolbox™ [16]. In this example, the magnitude response curves are achieved using a family of fourth-order low pass shelving filters for

bass, fourth-order peaking filters for mid, and fourth-order high-pass shelving filters for treble. Each family of curves is packed into a matrix that is the parameter referenced in the Simulink structural specification.

EXPLORE FLOATING-POINT BEHAVIOR IN SIMULATION

Exploration test benches provide insight into the behavior of the design specifications. In this tone controls example, different test benches are used to explore the behavior analytically and subjectively.

ANALYTICALLY TEST BEHAVIOR — Analytical exploration test benches calculate metrics and provide graphical insight into the behavior of the algorithm design specification. In this tone controls example, the magnitude response is estimated during simulation. The structure of the magnitude response exploration test is shown in **Figure 6**.

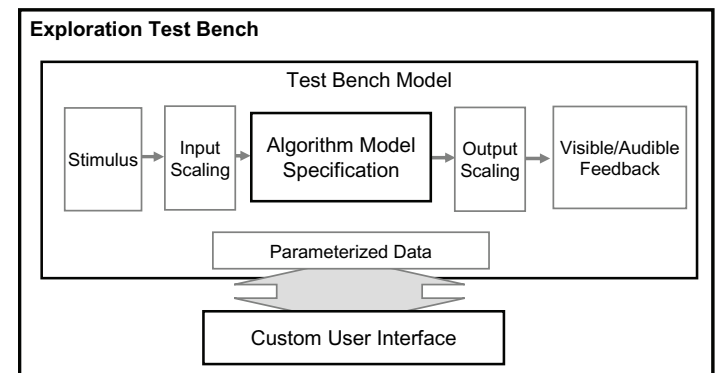


Figure 6. Exploration test bench.

As the algorithm design specification evolves, the test bench can be reused to test changes in the specification. The algorithm is specified in a separate model that is referenced by the test bench model – this provides for the separation of algorithm and test bench, which streamlines the design elaboration by making the test bench reusable throughout the design process. Interface subsystems connect the algorithm specification model to the stimulus and visualizations. These interface subsystems contain *Data Type Conversion* blocks that automate converting the test signals to the data type specified in the algorithm. This enables the same test bench to be used for both floating- and fixed-point algorithm specifications.

A custom user interface is programmed in MATLAB to interactively explore the behavior of the model during simulation [17]. The user interface provides pull down menus for each of the possible levels of boost or cut. A screen shot of the final magnitude response test bench is shown in **Figure 7**.