Lab 7. Filter Implementation in Matlab

Introduction

This lab examines the process of moving from filter design to hardware implementation. The lab investigates various architectures and various quantization schemes, and discusses hardware description language (HDL) code generation.

Filter Architectures

After the filter design process has generated filter coefficients $b$ and $a$, or a filter object $h$ (such as adaptfilt or mfilt object), the filter is implemented in a filter architecture using low-level operations of sum, gain, and delay (essentially, a block diagram of the difference equation).

Approximately two dozen filter architectures are supported by Matlab’s Signal Processing and Filter Design Toolboxes:

- Direct-form I
- Direct-form I, second-order sections
- Direct-form I transposed
- Direct-form I transposed, second-order sections
- Direct-form II
- Direct-form II, second-order sections
- Direct-form II transposed
- Direct-form II transposed, second-order sections
- Direct-form FIR
- Direct-form symmetric FIR
- Direct-form antisymmetric FIR
- Overlap-add FIR
- Lattice allpass
- Lattice autoregressive (AR)
- Lattice autoregressive moving average (ARMA)
- Lattice moving-average (MA) for maximum phase
- Lattice moving-average (MA) for minimum phase
- Coupled, allpass lattice
- Coupled, allpass lattice with power complementary output
- State-space
- Scalar gain object
- Filters arranged in series
- Filters arranged in parallel
FilterStructure

The filter command, when given \( b \) and \( a \) coefficient inputs, implements a direct-form II transposed architecture. When filter is given object input, it implements an architecture determined by the constructor used to create the object. The filter architecture of a filter object is stored in its FilterStructure field.

To create a filter object with a given architecture from filter coefficients \( b \) and \( a \), use the dfilt command. For example,

\[
h = \text{dfilt.df2t}(b,a)
\]

creates a direct-form II transposed discrete-time filter (dfilt) object that implements the filter with coefficients \( b \) and \( a \).

To display the FilterStructure field of a filter object \( h \), type:

\[
\text{get}(h, 'FilterStructure')
\]

FilterStructure is a read-only property. To use a different structure, construct a new filter with the appropriate method (or, for quantized filters only, use convert to switch to a new structure).

For a complete list of dfilt constructors, and a list of associated methods for working with filter objects, type:

\[
\text{help dfilt}
\]

Try:

\[
\text{help dfilt}
\]

\[
hm = \text{mfilt.firdecim}(3)
\]

\[
\text{get}(hm, 'FilterStructure')
\]

\[
b = \text{fir2}(12,[0 0.4 0.5 1],[1 1 0 0]);
\]

\[
hd1 = \text{dfilt.dffir}(b)
\]

\[
hd2 = \text{dfilt.fft FIR}(b)
\]

\[
isstable(hd1)
\]

\[
isstable(hd2)
\]
Filter Architecture Reference

As a reference, the graphical user interface archref allows you to browse different architectures, view a representative block diagram, and see the corresponding syntax for dfilt when creating a discrete-time filter object with the specified architecture in its FilterStructure field.

We'll look at archref a little later in this lab.

archref

(You need SGO1 and SGO2 software for this. It is Archive.zip under Software in EE-289 Labs).

Filter Realization

The Filter Design and Analysis tool allows you design a filter (or import one from the Workspace) and then realize it in a Simulink block diagram with a specified architecture.

Choose File → Import Filter from Workspace in FDATool or click the Import Filter sidebar button to open the Import Filter panel. You can then enter the names of Workspace variables for filter coefficients and choose a filter architecture from the pop-up menu.

You can convert from one architecture to another within the FDATool by choosing Edit → Convert Structure.

To realize a filter, click the Realize Model sidebar button.

The generated Simulink subsystem block uses delay, gain, and sum blocks in fixed-point mode. (Requires Simulink Fixed Point to run.)

Try:

[b a] = butter(8,0.25);
fdatool

Import the filter into the FDATool with various architectures and then realize your design in Simulink.

Test your realizations in the following model:

realmode

Can you determine the frequencies of interest?
Is this model LP, BP, or HP?

Change the design of the filter by changing the order and the cutoff frequencies and repeat as needed.

Second-Order Sections

Second-order sections (SOS) are second-order factors of a transfer function $H(z)$ with gain $g$:

$$H(z) = g \prod_{k=1}^{N} H_k(z) = g \prod_{k=1}^{N} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{a_{0k} + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

When quantized, higher order filters suffer from an accumulation of coefficient errors. Filters whose transfer functions have been factored into second-order sections are much more robust. Sections can be combined in parallel or in cascade to create higher order filters.

Matlab represents the second-order sections form of a filter as an $N \times 6$ array, where $N$ is the number of sections. Each row contains a single section, where the row elements are the three numerator and three denominator coefficients that describe the second-order section:

$$S = \begin{bmatrix}
    b_{01} & b_{11} & b_{21} & a_{01} & a_{11} & a_{21} \\
    b_{02} & b_{12} & b_{22} & a_{02} & a_{12} & a_{22} \\
    \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
    b_{0N} & b_{1N} & b_{2N} & a_{0N} & a_{1N} & a_{2N}
\end{bmatrix}$$

$[S \ g] = \text{tf2sos}(b,a)$

returns a matrix $S$ in second-order section form with gain $g$ equivalent to the filter represented by coefficient vectors $a$ and $b$.

$\text{hds} = \text{sos}(hd)$

returns a filter object $hds$ in second-order sections with a direct-form II transposed structure.

On the View menu in both FDATool and FVTool there is an option for SOS View. When you have an SOS filter in either tool, SOS View lets you see the performance of the sections of the filter - individually, cumulatively, or overall.

Try:

$[b \ a] = \text{butter}(8,0.25)$;
$hd = \text{dfilt.df1}(b,a)$
$hd.\text{Numerator}$
$hd.\text{Denominator}$
$\text{hds} = \text{sos}(hd)$
$\text{hds.sosMatrix}$
Import the filter into the FDATool in SOS form and view the individual, cumulative, and overall performance of the sections.

How does the SOS version compare to the original version in terms of performance?

Digital Signals

The value of a signal, at any instant, is its instantaneous amplitude. Time can assume continuous values $t$ or discrete values $n/f_s$, where $f_s$ is the sampling frequency. The amplitude can also assume a continuum of values or be quantized to a finite number of discrete levels. This results in four possible kinds of signals:

- Analog: continuous time, continuous level
- Quantized: continuous time, discrete level
- Sampled: discrete time, continuous level
- Digital: discrete time, discrete level

Sampling and quantization form the critical link between analog and digital signals.

Quantization

Hardware implementations can require filters to use minimum power, generate minimum heat, and avoid computational overload in their processors. Meeting these constraints often requires the use of quantized filters.

Because of finite signal lengths and the finite memory of computer processors, only a finite set of quantized sequences is possible. Sampling and quantization round or truncate signal values within the finite set of possibilities. Quantized samples are represented by a group (word) of zeros and ones (bits) that can be processed digitally. The finer the quantization, the larger the number of bits in the sample word.

Like sampling, improper quantization leads to loss of information. Unlike sampling, however, no matter how fine the quantization, the effects are irreversible, since word lengths must be finite. Finite word lengths appear as nonlinear effects (such as overflow and limit cycles) and can make systematic treatment of quantization extremely difficult. Quantization noise can be described in statistical terms, and is usually considered only in the final stages of design.

The Filter Design Toolbox gives you a laboratory for experimenting with quantization effects before a filter is implemented in hardware.
The toolbox includes tools to simulate the effects of quantization, and allows you to investigate the transition from floating-point to fixed-point arithmetic and observe the effects on performance. You can use the toolbox quantization functions for constructing, applying, and analyzing quantizers, quantized filters, and quantized FFTs.

**Fixed-Point Arithmetic**

You specify how numbers are quantized using fixed-point arithmetic. The two most important parameters are:

- Word length \( w \) in bits
- Fraction length \( f \) in bits

The fraction length is the number of bits between the binary point and the least-significant bit.

Where you place the binary point determines how fixed-point numbers are interpreted. For example, for a signed (two’s complement) fixed-point number, \( 10.110 \) represents \(-2 + 2^{-1} + 2^{-2} = -1.25\).

A fixed-point quantization scheme determines the dynamic range of the numbers that are used. Numbers outside this range are always mapped to fixed-point numbers within the range when you quantize them. The *precision* is the distance between successive numbers occurring within the dynamic range in a fixed-point representation.

- For a signed fixed-point number with word length \( w \) and fraction length \( f \), the dynamic range is from \(-2^{w-f-1} \) to \(2^{w-f-1} - 2^{-f}\).
- For an unsigned fixed-point number with word length \( w \) and fraction length \( f \), the dynamic range is from 0 to \(2^{w-f} - 2^{-f}\).
- In either case the precision is \(2^{-f}\).

When you quantize a number outside of the dynamic range, *overflows* occur. Overflows are more frequent with fixed-point quantization than with floating-point quantization, because the dynamic range is less for equivalent word lengths. Overflows can occur when you create a fixed-point quantized filter from an arbitrary floating-point design. You can either normalize your coefficients (and introduce a corresponding scaling factor for filtering) to avoid overflows, or else saturate or wrap.

**Quantized Filters**

The Filter Design Toolbox implements single-input-output filters as fixed-point filters or as single-precision or double-precision floating-point filters. Both single-precision floating-point and fixed-point filters are referred to as *quantized filters*. 
To use the fixed-point capabilities in the Filter Design Toolbox, either:

- Use `dfilt` or `fdesign` to create a filter object `hd` and set the `Arithmetic` property of the object to `'single'` or `'fixed'`:
  ```matlab
  set(hd,'Arithmetic','fixed')
  ```
- Copy an existing filter to inherit its properties.
- Use the FDATool.

Fixed-point properties of a filter object `hd` become dynamically available when you set the `Arithmetic` property to `'fixed'`. To see the writable properties of `hd`, simply display the variable in the Workspace. To see the complete list of properties, type
  ```matlab
  get(hd)
  
  ```

Use the `set` command (or index into the object structure array) to change properties such as word length, fraction length, rounding, and overflow handling. The documentation reference *Fixed Point Filter Properties* gives a full description of filter properties and values.

When you apply a quantized filter to data, the filter coefficients are quantized to your specifications, but so are the data that you filter (both the input and the output) and the results of any arithmetic operations that occur during filtering.

Try:

```matlab
[b a] = butter(8,0.25);
hd = dfilt.df1(b,a)
get(hd)
set(hd,'Arithmetic','fixed')
get(hd)
set(hd,'OverflowMode','Saturate')
get(hd)
```

```matlab
d = fdesign.lowpass('n,fs');
set(d,'Fcutoff',0.2)
designmethods(d)
d2 = butter(d)
fvtool(d)
```

```
edit qeffects
qeffects
```
Quantization in the FDATool

The Filter Design and Analysis Tool allows you to quantize filter designs and experiment with fixed-point properties to interactively evaluate performance tradeoffs in hardware.

In the FDATool, click the **Set Quantization Parameters** button and select a filter arithmetic. Choosing fixed-point arithmetic displays a tabbed panel with editable fields for each of the quantization properties of the filter.

Try:

```
b = fir1(101,0.45,'low',kaiser(102,7.8573));
```

Additional Experiments

Do the following:

1. In MATLAB type ‘archref’ and see the various structures for filters. (You need SGO1 and SGO2 software for this. It is Archive.zip under Software in EE-289 Labs).

2. Invoke ”fdatool” in MATLAB


5. Look at Filter Coefficients (under Analysis) for single and second-order sections.


7. Convert to different structures under Edit and observe frequency behavior.

8. “Realize Model” (push the “Realize Model” icon on the left hand side) for any one realization and note that you get a model ready for SIMULINK.

9. When looking at various filter structures, determine the transfer function of Direct Form I and Direct Form II.
HDL Code Generation

The HDL Filter Designer Toolbox provides a development environment for application-specific integrated circuit (ASIC) and field programmable gate array (FPGA) designs. It will automatically generate hardware description language (HDL) code based on filters designed in the toolbox.

System designers and hardware developers use HDLs such as very high speed integrated circuit (VHSIC) hardware definition language (VHDL), and Verilog to develop hardware designs. HDLs provide a proven method for hardware design, but coding filter designs, and hardware designs in general, is labor intensive and the use of these languages for algorithm and system-level design is less than optimal.

Using the Filter Design HDL Coder, system architects and designers can spend more time on fine-tuning algorithms and models through rapid prototyping and experimentation and less time on HDL coding. Architects and designers can efficiently design, analyze, simulate, and transfer system designs to hardware developers.

For example, an architect or designer might use the Filter Design Toolbox, the Filter Design and Analysis Tool, and the Filter Design HDL Coder to design a filter. Then, with the click of a button, the Filter Design HDL Coder generates a VHDL or Verilog implementation of the design and a corresponding test bench.

Generated code adheres to a clean HDL coding style that enables architects and designers quickly to address customizations as needed. The test bench feature increases confidence in the correctness of the generated code and saves time spent on test bench implementation.

With the HDL Filter Designer Toolbox installed, choose Targets → Generate HDL in the FDATool.

(The material in this lab handout was put together by Paul Beliveau and derives principally from the MathWorks training document “Matlab for Signal Processing”, 2006.)

©2009GM