Implementing the Filter Chain of a Digital Down-Converter in HDL

This demo uses the Filter Design Toolbox and Fixed-Point Toolbox to design a three-stage, multirate, fixed-point filter that implements the filter chain of a Digital Down-Converter (DDC) designed to meet the Global System for Mobile (GSM) specification.

Using the Filter Design HDL Coder we will generate synthesizable HDL code for the same three-stage, multirate, fixed-point filter. Finally, using Simulink and Link for ModelSim, we will co-simulate the fixed-point filters to verify that the generated HDL code produces the same results as the equivalent Simulink behavioral model. Note that you need all the products mentioned above to execute the complete demo.

Contents
- Digital Down-Converter
- GSM Specifications
- Cascaded Integrator-Comb (CIC) Filter
- Compensation FIR Decimator
- Third Stage FIR Decimator
- Multistage Multirate DDC Filter Chain
- Generate VHDL Code
- HDL Co-simulation with ModelSim in Simulink
- Verifying Results
- Summary

Digital Down-Converter

Digital Down-Converters (DDC) are a key component of digital radios. The DDC performs the frequency translation necessary to convert the high input sample rates found in a digital radio, down to lower sample rates for further and easier processing. In this example, the DDC operates at approximately 70 MHz and must reduce the rate down to 270 KHz.

The DDC consists of a Numeric Controlled Oscillator (NCO) and a mixer to quadrature down convert the input signal to baseband. The baseband signal is then low pass filtered by a Cascaded Integrator-Comb (CIC) filter followed by two FIR decimating filters to achieve a low sample-rate of 270 KHz ready for further processing. The final stage often includes a resampler which interpolates or decimates the signal to achieve the desired sample rate depending on the application. Further filtering can also be achieved with the resampler. A block diagram of a typical DDC is shown below.
This demo focuses on the three-stage, multirate, decimation filter, which consists of a CIC and two decimating FIR filters. The CIC filter is suitable for this high speed application (69.333 MHz) because of its ability to achieve high decimation factors and the fact that it’s implemented without using multipliers. The CIC in this example will perform decimation by 64. The second filter is a CIC-compensation FIR filter (CFIR) which has an inverse-sinc passband response, and decimates by 2. The third stage filter is a programmable FIR filter (PFIR) which ensures that the overall filter response meets the GSM spectral mask. It also decimates by 2 to achieve an overall decimation factor of 256.

**GSM Specifications**

The GSM bandwidth of interest is 160 KHz. The GSM requirements for the overall response of the three-stage, multirate filter of the DDC includes decimating the input signal by 256, achieving less than 0.1 dB of peak-to-peak passband ripple, and it must also achieve 18 dB of attenuation at 100 KHz. The GSM out of band rejection mask is also shown below.
Cascaded Integrator-Comb (CIC) Filter

The design and cascade of the three filters can be performed via the graphical user interface FDATool,

but we’ll use the command line functionality.
To avoid quantizing the fixed-point data coming from the mixer, which has a word length of 20 bits and a fractional length of 18 bits, \((S_{20,18})\), we'll set the input word length and fractional length of the CIC to the same values, \(S_{20,18}\). We must also define the word lengths per section of the CIC. These values are chosen to avoid overflow between sections. We define the CIC as follows:

\[
R = 64; \quad \% \text{Decimation factor}
\]
\[
M = 1; \quad \% \text{Differential delay}
\]
\[
N = 5; \quad \% \text{Number of sections}
\]
\[
IWL = 20; \quad \% \text{Input word length}
\]
\[
OWL = 20; \quad \% \text{Input word length}
\]
\[
h\text{cic} = \text{mfilt.cicdecim}(R,M,N,IWL,OWL,[50 29 24 24 24 24 24 24 24 24]);
\]
\[
h\text{cic}.\text{InputFracLength} = 18;
\]

Let’s plot and analyze the theoretical magnitude response of the CIC filter which will operate at the input rate of 69.333 MHz.

\[
\text{Fs}\_\text{in} = 69.333e6;
\]
\[
h = \text{fvtool}(h\text{cic},'Fs',\text{Fs}\_\text{in});
\]
\[
\text{set(gcf, 'Color', 'White');}
\]

The first thing to note is that the CIC filter has a huge passband gain, which is due to the additions and feedback within the structure. We can normalize the CIC’s magnitude response by cascading the CIC with a gain that is the inverse of the gain of the CIC. Normalizing the CIC filter response to have 0 dB gain at DC, will make it easier to analyze the overlayed filter responses of the CIC and the next FIR filter stage.
The other thing to note is that zooming in the passband region we see that the CIC has about -0.4 dB of attenuation (droop) at 80 KHz, which is within the bandwidth of interest. A CIC filter is essentially a cascade of boxcar filters and therefore has a sinc-like response which causes the droop. This droop needs to be compensated by the FIR filter in the next stage.
Compensation FIR Decimator

The second stage of our DDC filter chain needs to compensate for the passband droop caused by the CIC and decimate by 2. Since the CIC has a sinc-like response, we can compensate for the droop with a lowpass filter that has an inverse-sinc response in the passband. This filter will operate at 1/64th the input sample rate which is 69.333 MHz, therefore its rate is 1.0833 MHz. The filter that meets these constraints is the following:

```matlab
% Filter specifications
Fs = 1.0833e6;   % Sampling frequency 69.333MHz/64
N = 20;          % 21 taps
Npow = 5;        % Sinc power
w = 0.5;         % Sinc frequency factor
Apass = 5.7565e-4; % 0.01 dB
Astop = 0.01;    % 40 dB
Aslope = 60;     % 60 dB slope
Fpass = 80e3/(Fs/2); % 80 kHz passband-edge frequency

% Design filter.
cfir = firceqrip(N,Fpass,[Apass,Astop],’passedge’,’slope’,Aslope,...
       ’invsinc’,[w,Npow]);
```

% Define the fixed-point multirate filter. By default, the fixed-point
% attributes of the accumulator and multipliers are set to ensure that full
% precision arithmetic is used.
hcfir = mfilt.firdecim(2,cfir);
set(hcfir,...
Using the info method we can review the filter details in a nice format.

```
info(hcfir)
```

<table>
<thead>
<tr>
<th>Filter Structure</th>
<th>Direct-Form II Polyphase Decimator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimation Factor</td>
<td>2</td>
</tr>
<tr>
<td>Filter Order</td>
<td>20</td>
</tr>
<tr>
<td>Stable</td>
<td>Yes</td>
</tr>
<tr>
<td>Linear Phase</td>
<td>Yes (Type 1)</td>
</tr>
<tr>
<td>Numerator</td>
<td>16016</td>
</tr>
<tr>
<td>Input</td>
<td>0 dB</td>
</tr>
<tr>
<td>Output</td>
<td>0 dB</td>
</tr>
<tr>
<td>Product</td>
<td>0 dB</td>
</tr>
<tr>
<td>Accumulator</td>
<td>0 dB</td>
</tr>
<tr>
<td>Round Mode</td>
<td>convergent</td>
</tr>
<tr>
<td>Overflow Mode</td>
<td>wrap</td>
</tr>
</tbody>
</table>

Cascading the CIC with the inverse sinc filter we can see if we eliminated the passband droop caused by the CIC.

```matlab
hcas1 = cascade(hcicnorm,hcfir);
h = fvtool(hcicnorm,hcfir,hcas1,'Fs',[Fs_in,Fs_in/64,Fs_in]);
axis([0 .1 -0.8 0.8]);
legend(h,'hcic','hcfir','cascade');
set(gcf, 'Color', 'White');
```
As we can see in the filter response of the cascade of the two filters, which is between the CIC response and the compensating FIR response, the passband droop has been eliminated.

### Third Stage FIR Decimator

As indicated earlier the GSM spectral mask requires an attenuation of 18 dB at 100 KHz. So, for our third and final stage we can try a simple equiripple lowpass filter. Once again we need to quantize the coefficients to 16 bits. This filter also needs to decimate by 2.

\[
N = 62; \quad \text{% 63 taps}
\]
\[
F_s = 541666; \quad \text{% 541.666 kHz}
\]
\[
F = [0 \ 80e3 \ 100e3 \ F_s/2]/(F_s/2);
\]
\[
A = [1 \ 1 \ 0 \ 0];
\]
\[
W = [2 \ 1]; \quad \text{% Weight the passband more than the stopband}
\]
\[
pfir = firgr(N,F,A,W);
\]
\[
hpfir = mfilt.firdecim(2,pfir);
\]
\[
set(hpfir,...
\]
\[
'Arithmetic', \ 'fixed',...  
'CoeffWordLength', \ 16,...  
'InputWordLength', \ 20,...  
'InputFracLength', \ -12);
\]

When defining a multirate filter by default the accumulator word size is determined automatically to maintain full precision. However, because we only have 20 bits for the output, let’s set the output format to a word length of 20 bits and a fractional length of -12.

\[\]
Again we can use the info method to view the filter details.

```
info(hpfir)
```

### Multistage Multirate DDC Filter Chain

Now that we have designed and quantized the three filters, we can get the overall filter response by cascading the normalized CIC and the two FIR filters. Again, we're using the normalized CIC filter to ensure that the filter responses use the same scale.

```
hcasnorm = cascade(hcicnorm,hcfir,hpfir);
h = fvtool(hcasnorm,'Fs',Fs_in,'NumberofPoints',8192*3);
axis([0 1 -200 10]); % Zoom-in
set(gcf, 'Color', 'White');
```
To see if the overall filter response meets the GSM specifications, we can overlay the GSM spectral mask on the filter response.

```matlab
drawgsmmask; drawnow
pause(3)
```

We can see that our overall filter response is within the constraints of the GSM spectral mask. We also need to ensure that the passband ripple meets the requirement that it is less than 0.1 dB peak-to-peak. We can verify this by zooming in using the axis command.

```matlab
axis([0 .09 -0.08 0.08]);
```
Indeed the passband ripple is well below the 0.1 dB peak-to-peak GSM requirement.

**Generate VHDL Code**

FDATool also supports the generation of HDL code from the dialog shown below.
From FDATool as well as the command line you can generate VHDL or Verilog code as well as test benches in VHDL, Verilog, or as ModelSim .do files. Also, you have the ability to customize your generated HDL code by specifying many options, so that the generated code meets your coding standards and guidelines.

However, here we will use the command line functionality to generate the HDL code.

Now that we have our fixed-point three-stage multirate filter meeting the specs we are ready to generate HDL code. Before we do that let’s replace our Direct-form FIR Decimators with Direct-form Transposed FIRs because the Filter Design HDL coder currently only supports the Transposed structures while Simulink only supports the Direct-form structures.

```matlab
hcfirt = mfilt.firtdecim(2,cfir); % FIRT Decim required for HDL Code Gen
set(hcfirt,...
    'Arithmetic', 'fixed',...
    'CoeffWordLength', 16,...
    'InputWordLength', 20,...
    'InputFracLength', -12);

hpfirt = mfilt.firtdecim(2,pfir);
set(hpfirt,...
    'Arithmetic', 'fixed',...
    'CoeffWordLength', 16,...
    'InputWordLength', 20,...
    'InputFracLength', -12);

% Again, we need to set the output format.
set(hpfirt,...
    'FilterInternals', 'specifyPrecision',...
    'outputWordLength', 20,...
    'outputFracLength', -12,...
    'RoundMode', 'round',... % = nearest in SL
    'OverflowMode', 'Saturate');
```

Cascade of CIC and two FIR filters and generate VHDL

```matlab
hcast = cascade(hcic,hcfirt,hpfirt);
generatehdl(hcast,'TargetLanguage','VHDL','Name','filter');
```

### Starting VHDL code generation process for filter: filter
### Cascade stage # 1
### Starting VHDL code generation process for filter: filter_stage1

12
### Generating: D:\work\ddc\hdlsrc\filter_stage1.vhd
### Starting generation of filter_stage1 VHDL entity
### Starting generation of filter_stage1 VHDL architecture
### Integrator section, # 1
### Integrator section, # 2
### Integrator section, # 3
### Integrator section, # 4
### Integrator section, # 5
### Comb section, # 1
### Comb section, # 2
### Comb section, # 3
### Comb section, # 4
### Comb section, # 5
### Successful completion of VHDL code generation process for filter: filter_stage1

### Cascade stage # 2
### Starting VHDL code generation process for filter: filter_stage2
### Generating: D:\work\ddc\hdlsrc\filter_stage2.vhd
### Starting generation of filter_stage2 VHDL entity
### Starting generation of filter_stage2 VHDL architecture
### HDL latency is 0 samples
### Successful completion of VHDL code generation process for filter: filter_stage2

### Cascade stage # 3
### Starting VHDL code generation process for filter: filter_stage3
### Generating: D:\work\ddc\hdlsrc\filter_stage3.vhd
### Starting generation of filter_stage3 VHDL entity
### Starting generation of filter_stage3 VHDL architecture
### HDL latency is 1 samples
### Successful completion of VHDL code generation process for filter: filter_stage3

### Generating: D:\work\ddc\hdlsrc\filter.vhd
### Starting generation of filter VHDL entity
### Starting generation of filter VHDL architecture
### HDL latency is 2 samples
### Successful completion of VHDL code generation process for filter: filter

---

**HDL Co-simulation with ModelSim in Simulink**

To verify that the generated HDL code is producing the same results as our Simulink model, we’ll use Link for ModelSim to co-simulate our HDL code in Simulink. We have a pre-built Simulink model that includes two signal paths. One signal path produces Simulink’s behavioral model results of the three-stage, multirate filter. The other path produces the results of simulating, with Mod-
elSim, the VHDL code we generated.

```matlab
open('Filt_HDL_Cosim.mdl');
```

For the behavioral model simulation we will generate a Simulink block of the three-stage, multirate filter we designed and place that block in the Simulink model where we'll co-simulate with ModelSim.

```matlab
% Generate Simulink nlock of cascaded filters.
hcas = cascade(hcic,hcfir,hpfir);
block(hcas,'OverwriteBlock','on');
open_system('Filt_HDL_Cosim.mdl');
```

% Make sure filter block is connected.
```matlab
pos=get(gcbh,'position');set(gcbh,'position',pos+[-2 0 2 0]);
```

% Start ModelSim.
```matlab
vsim('tclstart',ddclinkdemocmds,'socketsimulink',4449);
pause(5) % Wait until ModelSim starts.
```

Run Simulink simulation and open the Scope to view results.

```matlab
sim('Filt_HDL_Cosim');
open_system('Filt_HDL_Cosim/Verification_Results/Time Scope');
```
Verifying Results

The trace on the top is the excitation chirp signal. The next signal labeled "ref" is the reference signal produced by the Simulink behavioral model of the three-stage multirate filter. The bottom trace labeled "cosim" on the scope is of the ModelSim simulation results of the generated HDL code of the three-stage multirate filter. The last trace shows the error between Simulink's behavioral model results and ModelSim's simulation of the HDL code.
Summary

We used several MathWorks products to design and analyze a three-stage, multirate, fixed-point filter chain of a DDC for a GSM application. Then we generated HDL code to implement the filter and verified the generated code by comparing Simulink’s behavioral model with HDL code simulated in ModelSim via Link for ModelSim.