An FPGA Implementation of the Digital IF Processor

Grant Hampson

March 7, 2002

Introduction

This document describes the simulation and FPGA implementation of the IIP Radiometer Digital IF Section described in [1]. The digital IF processor presented here is a slightly different to that of [1]. The digital IF processor implemented here is illustrated in Figure 1 where there are four distinct sections; namely demultiplex, FS/4 down conversion, filtering and FS/4 up conversion.

The structure differs by keeping the FS/4 down conversion and FIR filter coefficients separate. In the original design the FIR filter coefficients incorporated the ± 1 sign changes. Unfortunately the FIR compiler [2] couldn't synthesize this successfully. The resulting increase in size due to the separation is very small compared to the overall size of the design.

This document is broken into four separate sections. The first section illustrates the results of Matlab simulations of the structure shown in Figure 1. Section 2 deals with an implementation of the structure in an Altera APEX FPGA. Section 3 presents simulation results of the FPGA implementation. Finally, Section 4 presents results of a Matlab simulation using recorded data from an AD9054 evaluation board.

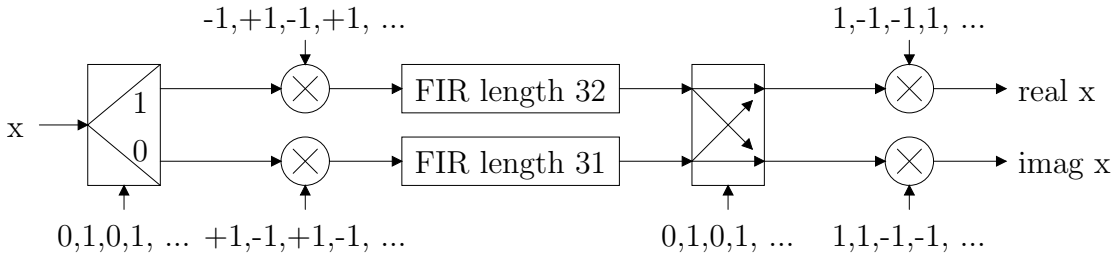
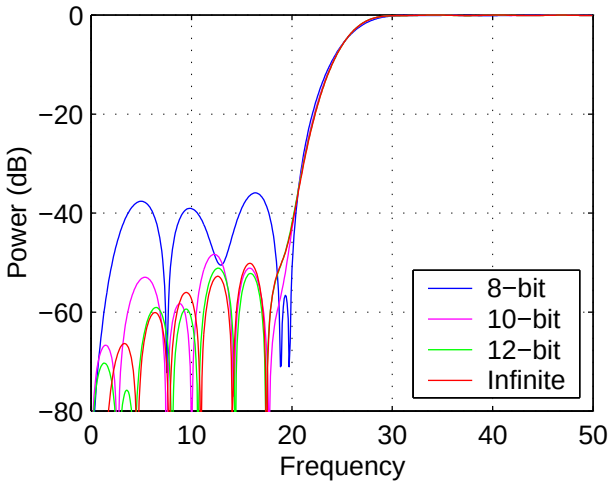


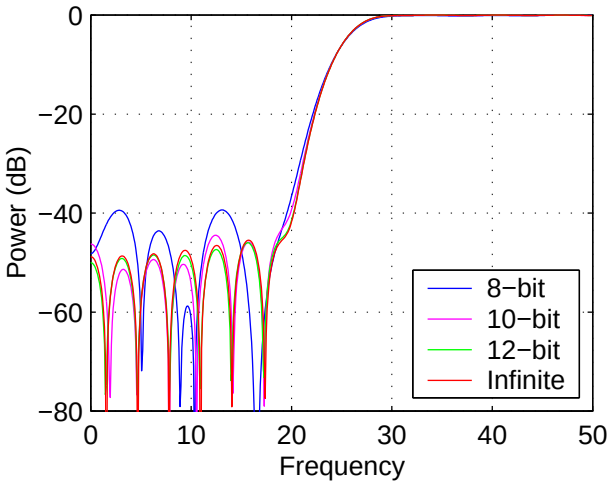
Figure 1: The digital IF processor architecture implemented in this document. The input sample rate of x is $F_s = 200MHz$ providing a BW of 100MHz (although only 50MHz of this is used.) The spectrum is shifted down in frequency by 50MHz (FS/4 down) and filtered to remove the image component. The spectrum is then shifted up in frequency by 25MHz (FS/4 up.) The output of the digital IF processor is a 100MHz complex output with 50MHz of BW.

1 Matlab Simulations of the Digital IF Processor

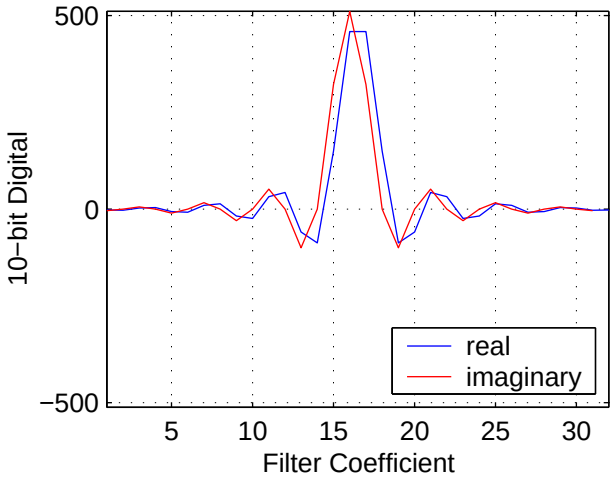
A Matlab simulation of the structure shown in Figure 1 is relatively straight forward. The Matlab code for the simulation is listed in Appendix A for completeness. Note that the filter coefficients of the FIR filters are designed in [1]. Figure 2(a&b) illustrate how quantization effects the filter responses. It was decided that the 10-bit quantization produces an optimal filter implementation size when traded off for filter performance. The filter coefficients implemented are shown in Figure 2(c).



(a) Real FIR Filter Response



(b) Imaginary FIR Filter Response



(c) 10-bit Quantized Filter Coefficients

Figure 2: FIR filter responses for various levels of quantization. 8-bit coefficients result in a poor response when compared to the responses of 10, 12-bits and no quantization. (c) The filter coefficients used in the real and imaginary parts of the Fs/4 down conversion. The real filter contains 32 coefficients, whereas the imaginary filter has only 31 coefficients. The differing number of coefficients changes the filter symmetry and implementation size.

The input spectrum to the digital IF processor is shown in Figure 3. This spectrum is shown for 8-bit input quantization. The output spectrum for various levels of output quantization are shown in Figure 4. For 8-bit output quantization a 30dB out of band suppression is achieved. By using additional output bits it is possible to achieve up to 48dB of suppression (also indicated by Figure 2(a)). The curves indicate that there is little gain for going beyond 12-bit output resolution (46dB suppression.) Note that the two input tones are now located at 16 and 27MHz with the correct power levels, i.e., the lower frequency tone has 6dB more gain.

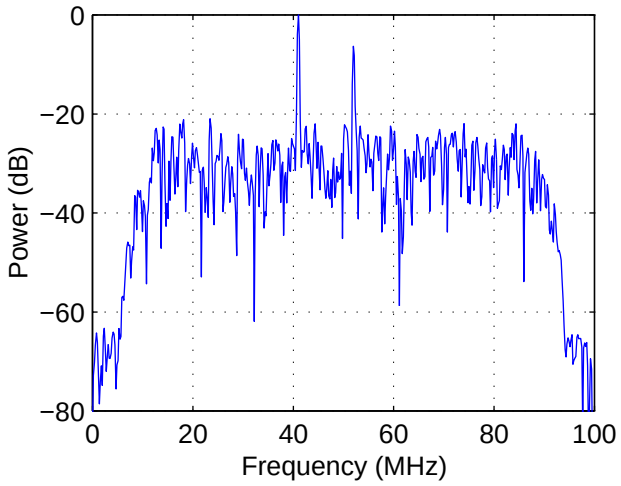


Figure 3: Input spectrum consists of gaussian noise and two sinusoids of differing amplitude. The first sinusoid at 41MHz is 6dB larger than the second sinusoid at 52MHz. The spectrum is band limited from 10 to 90MHz.

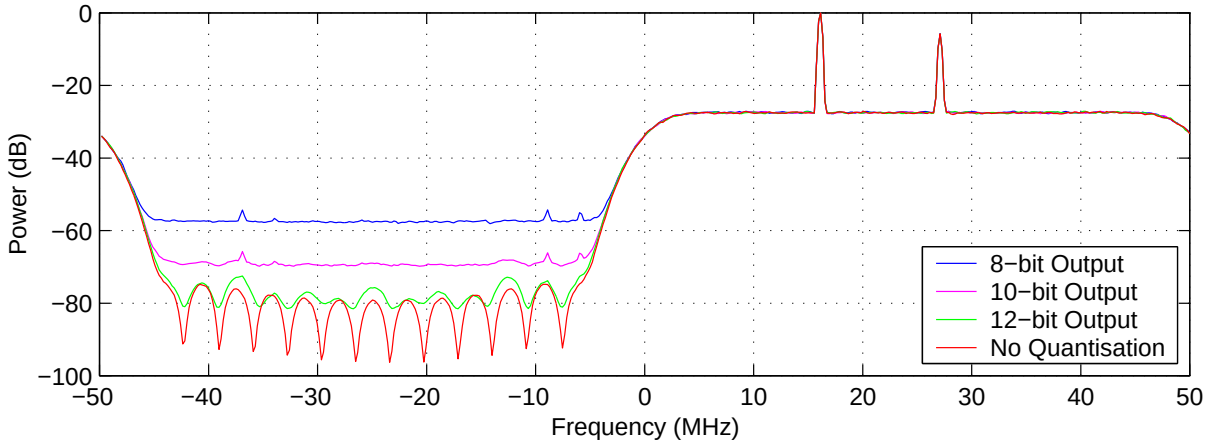


Figure 4: The output spectrum of the digital IF processor for varying levels of output quantization (1000 integrations.) The two tones are shifted to 16 and 27MHz. A greater number of output bits increases the output resolution and thus the suppression. 8-bit quantization produces 30dB of out of band suppression, 10-bit is 41dB, 12-bit is 46dB. With no quantization in the entire system it is possible to achieve 48dB of suppression.

2 An Altera FPGA Implementation

Implementation of the digital IF processor in an Altera APEX is relatively straight forward. Figure 5 illustrates the first section of processing in the digital IF processor. This is a screen capture from the Altera FPGA compilation tool called Quartus II. The input section is targeted for a AD9054 analogue to digital converter [3]. The FS/4 down conversion is implemented using simple negators and multiplexers. A controller implements the required synchronization of control bits in the entire processor. The contents of this controller are listed in Appendix B of this report.

Figure 6 illustrates the implementation of the FIR filters. The Altera FIR compiler [2] is available for free evaluation (but no FPGA programming files can be created). This mega core is straight forward to use. The mega core also produces Matlab files which can be used to simulate the generated filter in Matlab. As state previously the two FIR filters symmetry changes the implementation size of the filter. When compiled separately the size of the real and imaginary FIR filters is 1680 and 1171 logic elements, respectively. The large number of zero coefficients in the imaginary FIR reduces its size significantly. The FIR compiler is available for $500.

The final stage of the digital IF processor is shown in Figure 7. This section will either perform FS/4 up or down depending on how the controller is configured. Once again it is created from multiplexers and negators.

The entire digital IF processor design fits comfortably in a EP20K100ETC144-1X FPGA (an Altera APEX FPGA.) This FPGA is currently worth $148 from Arrow Electronics. It uses 2965 logic elements of 4160 logic elements available (71%). According to simulation results it should be capable of clock speeds up 126MHz.

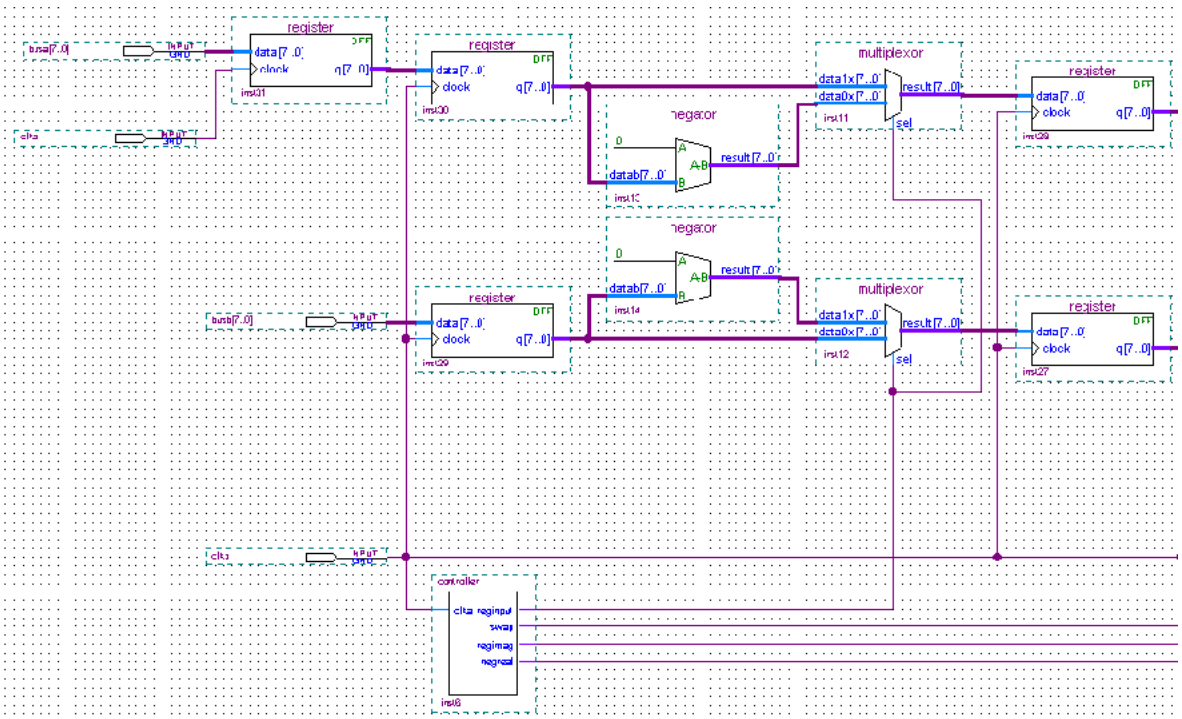


Figure 5: The first stage of digital processing involves an FS/4 down conversion. This is implemented using two negators and two multiplexers.

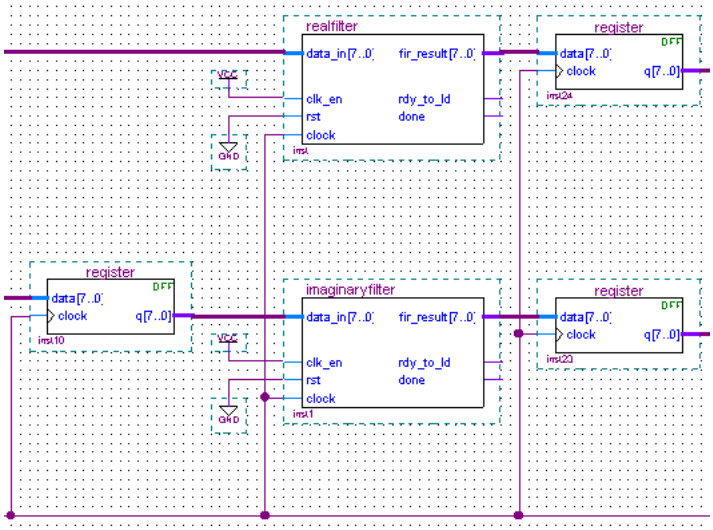


Figure 6: The filtering stage of the digital IF processor requires two FIR filters. Since the real filter contains one tap less, an additional register is required on the input to synchronize the two data streams.

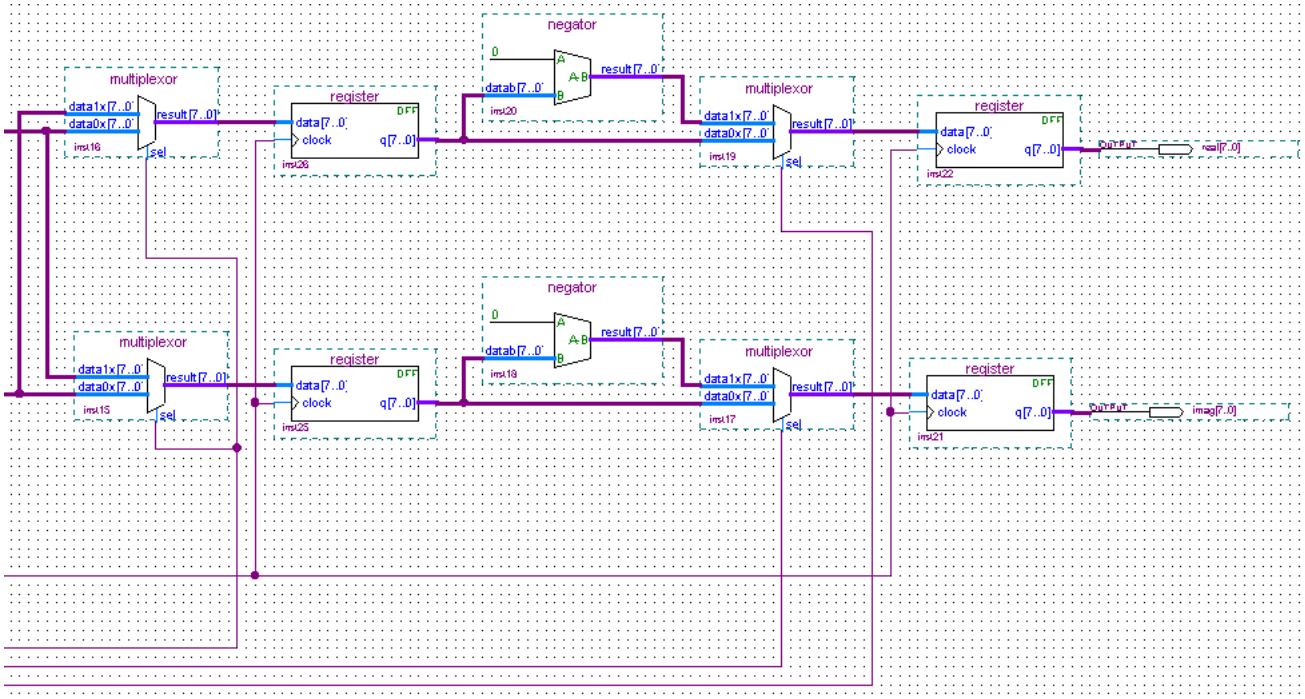
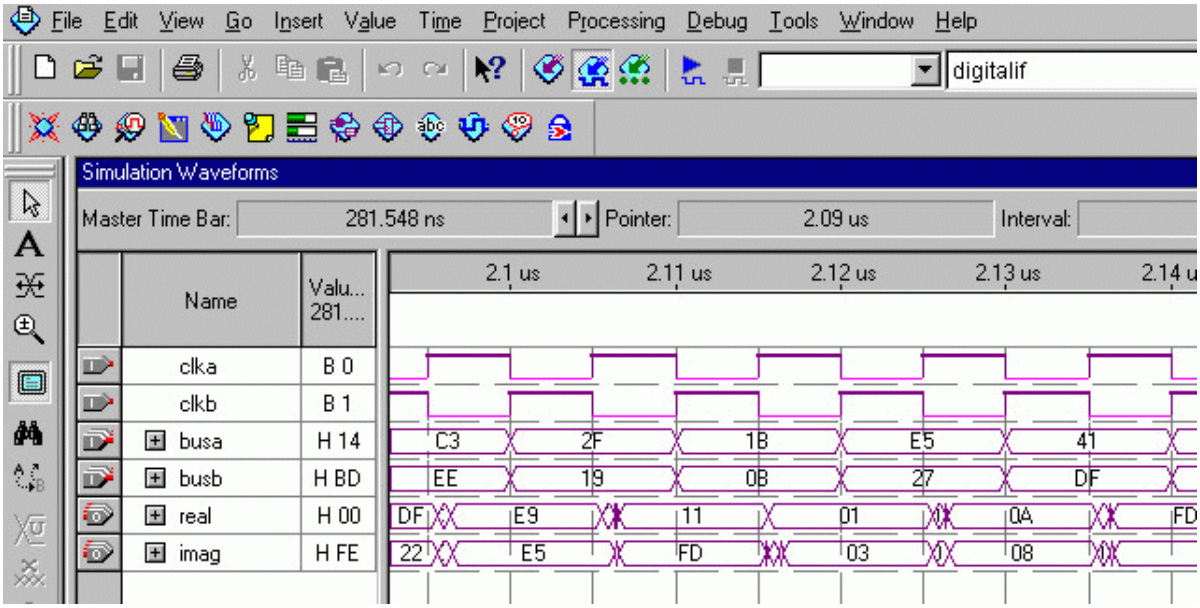


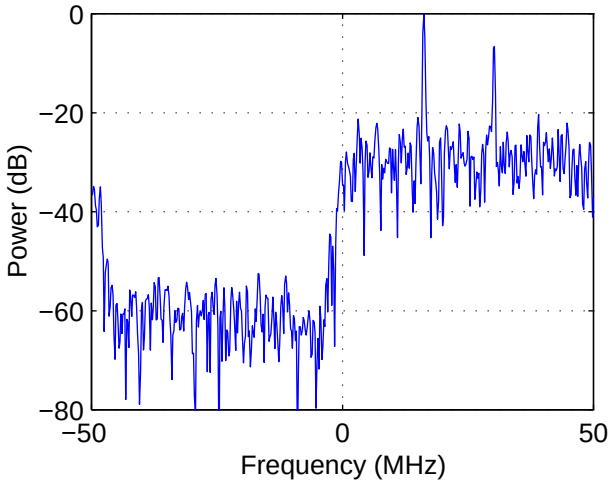
Figure 7: The final stage of the IF processors is an FS/4 up or down conversion depending on which band is being used.

3 FPGA Simulation Results

The Quartus simulation tool can be programmed to read input stimulus from a file. Matlab is used to take the same input stimulus used in previous simulations and create a file containing the same stimulus. The result of the simulation is shown in Figure 8(a). It is possible to save this output data into a text file where it can be analyzed in Matlab. The spectrum of this data is shown in Figure 8(b) for the same stimulus shown previously in Figure 3. The resulting spectrum is similar to that of Figure 4, except here there is no integration used since it takes a few minutes for each FPGA simulation data set.



(a) Quartus Simulation Tool Output

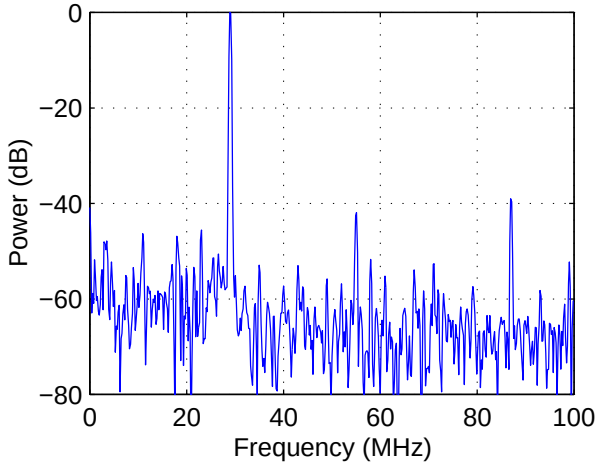


(b) Altera FPGA Output Spectrum

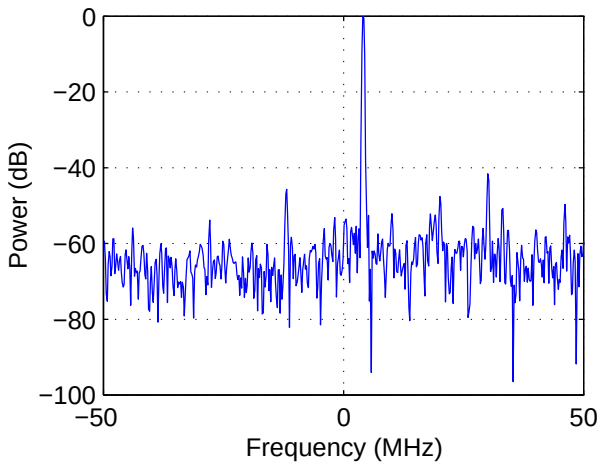
Figure 8: Simulation of the digital IF processor in Quartus. The raw output (a) is timing diagrams which can be saved and analyzed in Matlab, shown by the spectrum (b).

4 Real data from an AD9054 Evaluation Board

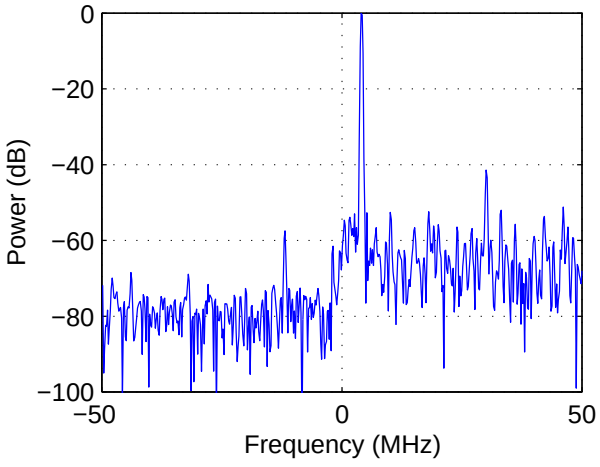
Since we cannot create FPGA programming files using the FIR mega core, it is not possible to implement it in any real hardware. However, it is possible to use previously captured data from an AD9054 evaluation board [4] and run it through the Matlab simulation. Figure 9(a) illustrates the spectrum of the captured data, with a 50MHz low pass anti-aliasing filter (instead of the band pass 10-90MHz filter.) A tone is present at 29MHz which is almost full scale. The output of the Matlab simulation is shown in Figure 9(b). Since there is little noise present in the input data it is difficult to see the band pass filter shape. The 29MHz tone is correctly translated to 4MHz. The simulation is conducted with 8, 10 and 12-bit output data paths, each giving varying levels of out of band suppression.



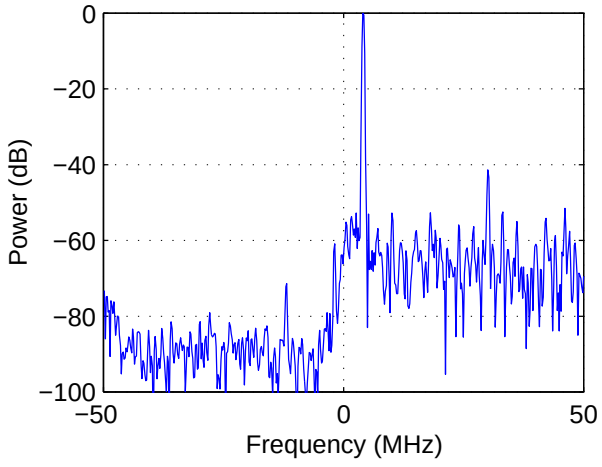
(a) AD9054 Captured 8-bit Data Spectrum



(b) Digital IF Processor 8-bit Output Spectrum



(c) Digital IF Processor 10-bit Output Spectrum



(d) Digital IF Processor 12-bit Output Spectrum

Figure 9: Simulation of the digital IF processor in Matlab using previously captured data from an 8-bit AD9054 evaluation board. The output data path quantization level is 8, 10 and 12-bits.

5 Summary and Conclusions

This document has shown the simulation of the digital IF processor in Matlab which has revealed many interesting aspects of the design. The quantization level of the FIR filter coefficients appears to not as critical as the output quantization level. The filter coefficients provide acceptable performance at 10-bits.

An implementation of the digital IF processor in an Altera APEX FPGA was also discussed in detail. The design utilizes a FIR mega core function from Altera which retails for $500. The complete design fits comfortably in an EP20K100ETC144-1X FPGA, which costs $148.

Simulation results from the Quartus software were also presented for verification. Additionally, real captured data from an AD9054 evaluation board were input into the Matlab simulation and output spectrums recorded for different quantization levels.

References

[1] S. W. Ellingson, "Design of the IIP Radiometer Digital IF Section," February 11 2002. http://esl.eng.ohio-state.edu/rfse/iip/iipdigif.pdf.

[2] *FIR Compiler MegaCore Function*, Altera Corporation, December 2001. http://www.altera.com/literature/ug/fircompiler_ug.pdf.

[3] *8-bit, 200MSPS A/D Converter*, Analog Devices Corporation, July 2001. http://www.analog.com/productSelection/pdf/AD9054A_d.pdf.

[4] G. A. Hampson, "AD9054 Prototype Evaluation," March 5 2002. http://esl.eng.ohio-state.edu/rfse/iip/ad9054proto.pdf.

Appendix A: Matlab Digital IF Processor Simulation Code

```
% Matlab code to test Digital IF Processor
% Grant Hampson,  Feb 12, 2002

clear all
load filtcoff.mat % load FIR filter coefficients

N = 1024;                       % number of time domain samples for filter

tone1 = 1*sin(2*pi*52e6*n/FS); % tones must be between 10-90MHz, fc = 50MHz
tone2 = 2*sin(2*pi*41e6*n/FS);

integ = 1000;                   % number of integrations
inty = 0;                       % zero integrator

realinsign = ones(N/2,1);       % sign change for demuxed inputs
realinsign(2:2:N/2) = -1;
imaginsign = ones(N/2,1);
imaginsign(1:2:N/2) = -1;       % opposite indexes negate

htreal = ht(1:2:63);            % real part      - even filter indexes, 32 tap filter
htimag = ht(2:2:63);            % imaginary part - odd filter indexes,  31 tap filter

maxfiltval = max([max(abs(htreal)) max(abs(htimag))]); % largest filter value
htreal = fix(511*htreal/maxfiltval); % quantise coefficients to 10-bits
htimag = fix(511*htimag/maxfiltval);

figure(1)
plot(1:32,htreal,'b',1:31,htimag,'r'),grid
xlabel('Filter coefficient'),ylabel('10-bit coefficient digital value')
title('FIR Filter Coefficients - blue is real, red is imaginary')
ylim([-512 511])

fidr = fopen('realfilt.dat','w');  % write coefficients to file for FIR compiler
for i = 1:32
   if i < 32
      fprintf(fidr,'%f\n',htreal(i));
   else
      fprintf(fidr,'%f',htreal(i));  % no carriage return!
   end
end
fclose(fidr);
fidi = fopen('imagfilt.dat','w');
for i = 1:31
   if i < 31
      fprintf(fidi,'%f\n',htimag(i));
   else
      fprintf(fidi,'%f',htimag(i));
   end
end
fclose(fidi);
```

```
multout = zeros((N/2)-1,1);        % FS/4 up-conversion exp(+j*pi*k/2)
multout(1:4:(N/2)-1) = 1;          % 1
multout(2:4:(N/2)-1) = sqrt(-1);   % j
multout(3:4:(N/2)-1) = -1;         % -1
multout(4:4:(N/2)-1) = -sqrt(-1);  % j

b = fir1(64,[.1 .9]);              % 10-90MHz BP filter on input

for i=1:integ
   x = randn(N,1) + tone1 + tone2; % new random sequence each integration
   x = filter(b,1,x);              % 10-90MHz BP filter
   x = round(127*x/max(abs(x)));   % quantise input to 8-bits

   yreal = filter(htreal,1,x(1:2:N).*realinsign); % FS/4 down, FIR filter 32
   yimag = filter(htimag,1,x(2:2:N).*imaginsign); % FS/4 down, FIR filter 31

   yreal = fix(yreal/2048); % make it a 8-bit output
   yimag = fix(yimag/2048);

   yreal = yreal(2:(N/2));  % Real filter one tap longer so delay it
   yimag = yimag(1:(N/2)-1);

   y = yreal+ sqrt(-1)*yimag; % make it a complex number
   y = y.*multout;            % FS/4 up-conversion (25MHz)

   Y = fft(y.*blackman((N/2)-1));
   inty = inty + real(Y).^2 + imag(Y).^2;
end
inty = inty/integ; % average integrations

figure(2)
subplot(211),plot(1:N,x),grid
ylim([-128 127])
xlim([1 N])
xlabel('Sample'),ylabel('8-bit Digital Value')
title('Real Data, FS=200MHz')
f = [0:N-1]*FS/N;
subplot(212),plot(f/1e6,20*log10(abs(fft(x.*blackman(length(x))))),grid
xlabel('Frequency (MHz)'),ylabel('Power (dB)')
title('Real Data Frequency Response, FS=200MHz')

figure(3)
N = (N/2)-1;           % only half the samples at output
FS = FS/2;             % output is decimated by 2 also
f = [-N/2+1:N/2]*FS/N; % complex spectrum
subplot(211)
plot(1:N,real(y(1:N)),'b',1:N,imag(y(1:N)),'r'),grid
xlabel('Sample'),ylabel('8-bit Digital Value')
title('Converted Output Data, real (b), imag (r)')
ylim([-128 127])
subplot(212)
plot(f/1e6,20*log10(abs(fftshift(fft(y.*blackman(length(y)))))), ...
     f/1e6,10*log10(fftshift(inty)),'r'),grid
xlabel('Frequency (MHz)'),ylabel('Power (dB)')
title('Converted Data Frequency Response, inst (b), integ (r)')
```

Appendix B: Controller FPGA Code

```
-- Controller for Digital IF processor
-- Grant Hampson, 13 Feb, 2002

SUBDESIGN controller
(
   clka
            : INPUT;

   neginput,
   swap,
   negimag,
   negreal
            : OUTPUT;
)

VARIABLE
   dif_sm[1..0] : DFF;

BEGIN
   --
   -- State Machine for controlling Digital IF
   --
   -- Current   Current      Next       Next
   -- State     Inputs       State      Outputs
   TABLE
   dif_sm[].q => dif_sm[].d, neginput, swap, negimag, negreal;
        B"00" =>      B"01",     B"0", B"1",    B"0",    B"0";
        B"01" =>      B"10",     B"1", B"0",    B"0",    B"1";
        B"10" =>      B"11",     B"0", B"1",    B"1",    B"1";
        B"11" =>      B"00",     B"1", B"0",    B"1",    B"0";
   END TABLE;
   dif_sm[].clk = clka;
END;
```