



Hideo Okawara's Mixed Signal Lecture Series

DSP-Based Testing – Fundamentals 15 Filtering in Frequency Domain

*Verigy Japan
July 2009*

Preface to the Series

ADC and DAC are the most typical mixed signal devices. In mixed signal testing, analog stimulus signal is generated by an arbitrary waveform generator (AWG) which employs a D/A converter inside, and analog signal is measured by a digitizer or a sampler which employs an A/D converter inside. The stimulus signal is created with mathematical method, and the measured signal is processed with mathematical method, extracting various parameters. It is based on digital signal processing (DSP) so that our test methodologies are often called DSP-based testing.

Test/application engineers in the mixed signal field should have thorough knowledge about DSP-based testing. FFT (Fast Fourier Transform) is the most powerful tool here. This corner will deliver a series of fundamental knowledge of DSP-based testing, especially FFT and its related topics. It will help test/application engineers comprehend what the DSP-based testing is and assorted techniques.

Editor's Note

For other articles in this series, please visit the Verigy web site at www.verigy.com/go/gosemi.

Filtering in Frequency Domain

In the last newsletter article the keys to play successful IFFT are discussed. So let's look at one of the typical applications of IFFT. It is a kind of filter processing in the frequency domain. A waveform refining procedure is demonstrated in this issue.

The Keys

Let's review the points for successful IFFT. See Figure 1. In order to generate a real number waveform automatically, the source spectrum must be a full page spectrum which is complex conjugate between the front and the back pages.

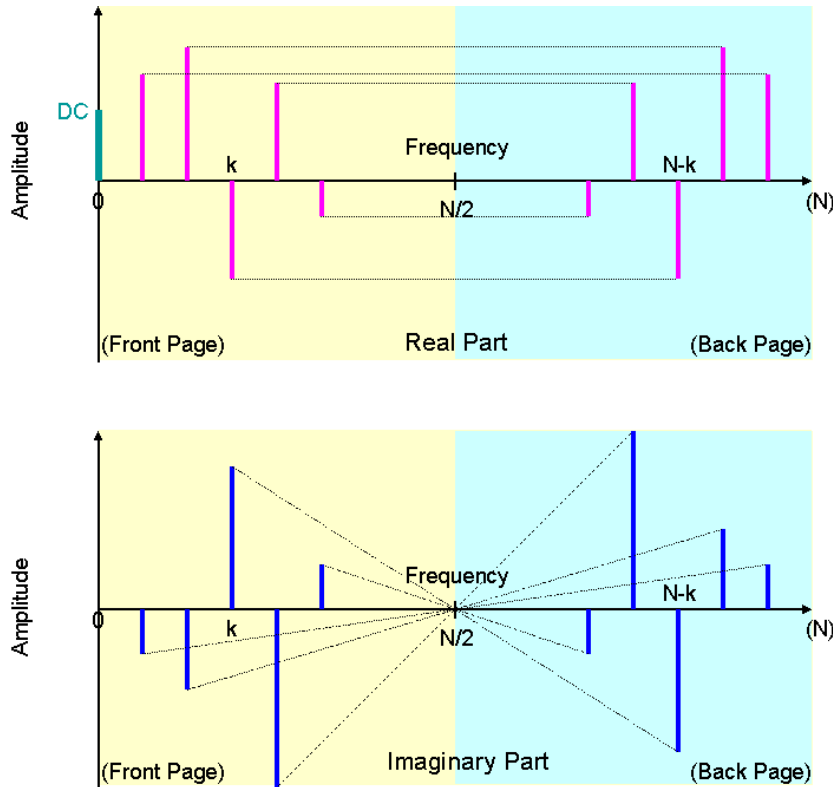


Figure 1: Complex Conjugate

DC 1V is $1.0 + j 0.0$ at Frequency 0, and each one of the AC amplitudes is halved and distributed in the front and back pages. The IFFT result is perfunctorily complex waveform whose imaginary part is zero so that the real part should be picked up.

Refining of Clock Waveform

Let's see the clock refining procedure. Figure 2 shows the experimental set-up for clock measurement. The clock source generates 1250MHz clock signal, which is measured by the waveform sampler.

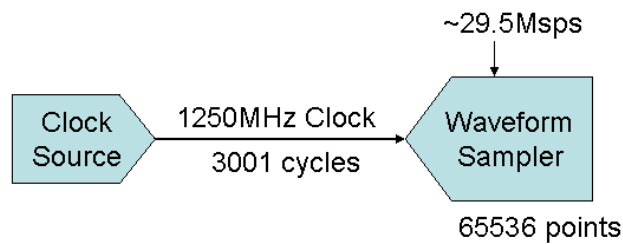


Figure 2: Test Circuit

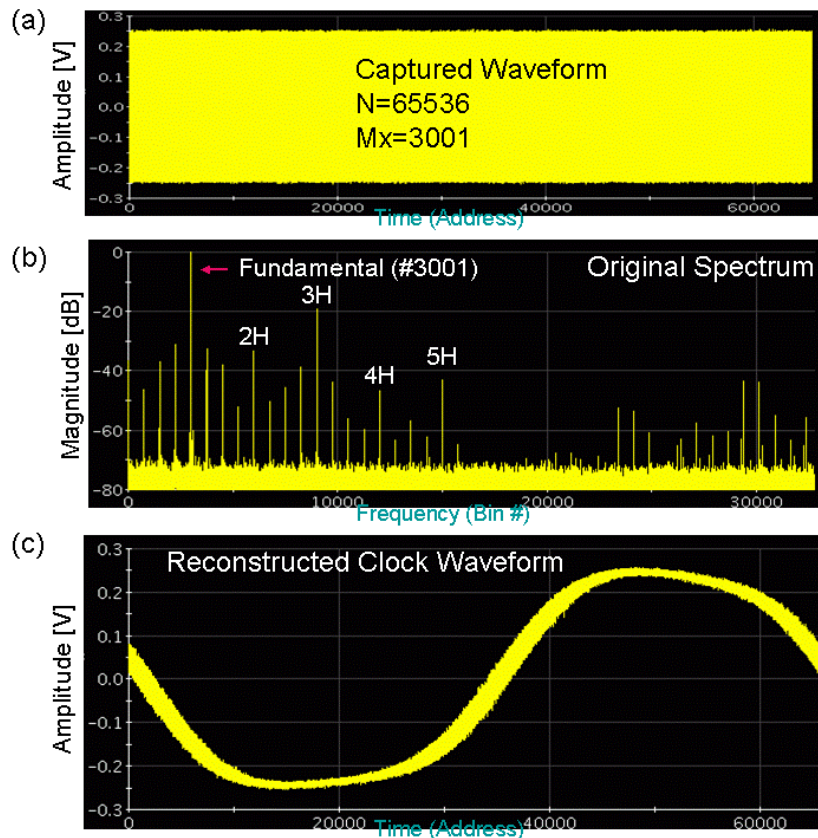


Figure 3: Measured Waveform & Spectrum

The sampler captures 3001 cycles of the clock waveform as Figure 3(a). Applying FFT to the waveform, the frequency domain spectrum is derived as (b). You can see the fundamental spectrum at the bin 3001, and its harmonics at the bin locations of the multiples of 3001. Playing `DSP_SHUFFLE()` with the key number 3001, a single cycle of clock waveform is reconstructed as (c), which contains lots of noise and jitter so that the trace appears very thick.

Applying `DSP_SPECTRUM()` to the reconstructed clock waveform, the frequency spectrum is shown in Figure 4 that shows the vicinity of the DC.

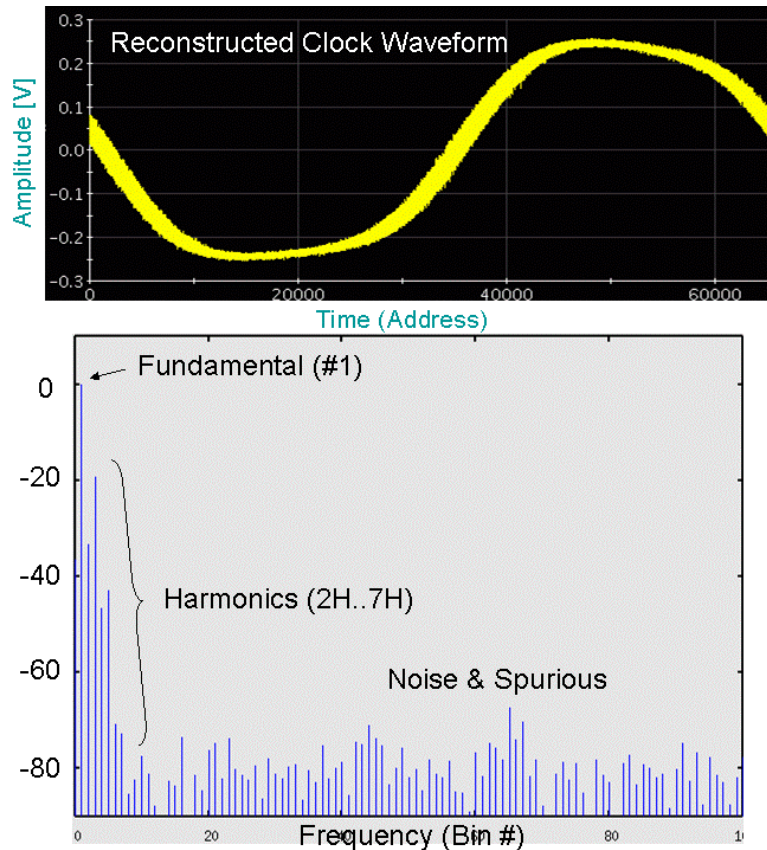


Figure 4: Spectrum of Reconstructed Clock

Carefully examining the spectrum in Figure 4, you can find the fundamental tone at the bin #1 and its harmonics at from #2 to #7. The rest of them seems the noise and spurious in the signal. So collecting the spectra from the bin #0 to #7, IFFT would reconstruct the refined clock waveform. This is the story of this experiment.

Let's move on to the coding. The better and simpler practice in FFT-IFFT filtering procedure is to use complex array for real waveform data.

```

01: INT          i,Nsamp;
02: ARRAY_D      dVsmp,dWave,dWave1;
03: ARRAY_COMPLEX CWave,CSp,CWave1;
04:
05: ...
06:
07: dVsmp=DGT("DOUT").getWaveform(); // Upload waveform
08: Nsamp=dVsmp.size();
09: DSP_SHUFFLE(dVsmp,dWave,3001); // 1 cycle waveform
10: // Reconstructed
11: CWave.resize(Nsamp);
12: for (i=0;i<Nsamp;i++) { // DOUBLE to COMPLEX
13:     CWave[i].real()=dWave[i];
14:     CWave[i].imag()=0.0;
15: }
16:
17: DSP_FFT(CWave,CSp,RECT); // Full-page spectrum
18: Nx=8; // Keep #0 to #7
19: for (i=Nx;i<=(Nsamp-Nx);i++) { // #8 to #(Nsamp-8)
20:     CSp[i].real()=0.0; // Zero
21:     CSp[i].imag()=0.0;
22: }
23: DSP_IFFT(CSp,CWave1); // IFFT
24: dWave1.resize(Nsamp);
25: dWave1=CWave1.getReal(); // Take the Real Part
26:

```

List 1: Waveform Refining Coding (1)

This is the way that the clock waveform is refined as the thin line in the thick trace as Figure 5 shows.

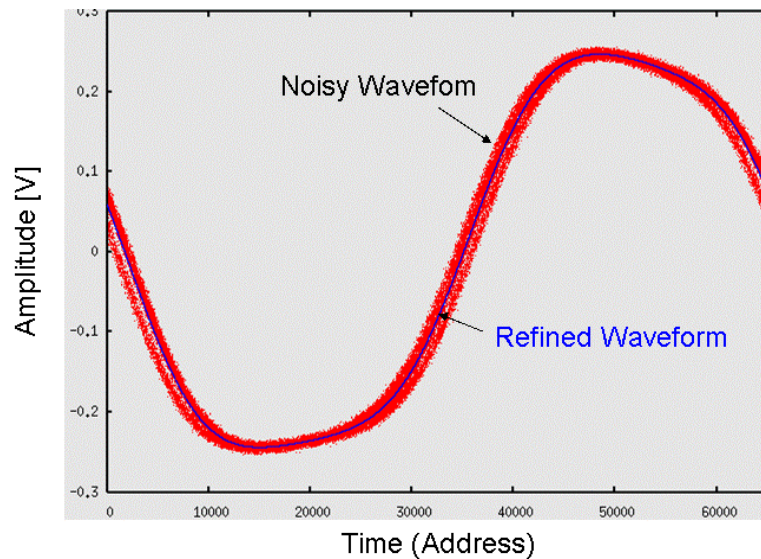


Figure 5: Refined Waveform

The clock frequency in this experiment is very high so that the waveform is rolled off. As Figure 4 shows, several lines of harmonics are good enough to reconstruct the original waveform. If the waveform would look square, much more lines of harmonics would be required to reconstruct the reasonable shape. How many harmonics you need to take account should be decided by examining the spectrum in online debugging.

Just in case you would perform FFT generating the half-page spectrum, lines 17 to 26 in List 1 should be modified as follows.

```

27: INT    Nsp;
28: ARRAY_COMPLEX CSp1;
29:          // FFT input is DOUBLE type.
30: DSP_FFT(dWave,CSp,RECT); // Half-page spectrum
31: Nsp=CSp.size(); // Nsp=Nsmpl/2
32: CSp1.resize(2*Nsp); // For full-page spectrum
33: Nx=8; // Keep #0 to #7
34: CSp1[0].real()=CSp[0].real(); // DC
35: CSp1[0].imag()=0.0;
36: for (i=1;i<Nx;i++) {
37: CSp1[i].real()=0.5*CSp[i].real(); // Slash half
38: CSp1[i].imag()=0.5*CSp[i].imag(); // Slash half
39: CSp1[N-i].real()= CSp1[i].real(); // Complex-
40: CSp1[N-i].imag()= -CSp1[i].imag(); // Conjugate
41: }
42: for (i=Nx;i<(Nsmpl-Nx);i++) {
43: CSp1[i].real()=0.0; // #8 to #(Nsmpl-8)
44: CSp1[i].imag()=0.0; // Zero
45: }
46:
47: DSP_IFFT(CSp1,CWave1);
48: dWave1.resize(Nsmpl);
49: dWave1=CWave1.getReal(); // Take the real part
50:

```

List 2: Waveform Refining Coding (2)