# Hideo Okawara's
# Mixed Signal Lecture Series

# DSP-Based Testing – Fundamentals 21
# Trend Removal (Part 1)

*Verigy Japan*

*January 2010*

## 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.

## Trend Removal (Part 1)

When you retrieve measured waveforms from a DUT ADC or a digitizer, you may have experienced to see ugly DC offset drift and to have hard time to get a flat noise floor in the spectrum. It could often occur when DC blocking capacitors are provided in the test signal path in a DUT board. In the articles of this month and next month, how we could cope with such situations is discussed.

## DC Drifting Waveform

Figure 1 shows a good example that the measured waveform suffers from the severe DC offset drift. This signal is captured on a DUT board with a DC blocking capacitor inserted in the signal path. The target waveform is approximately 1MHz and it is sampled by a waveform digitizer at the rate of 110Msps.

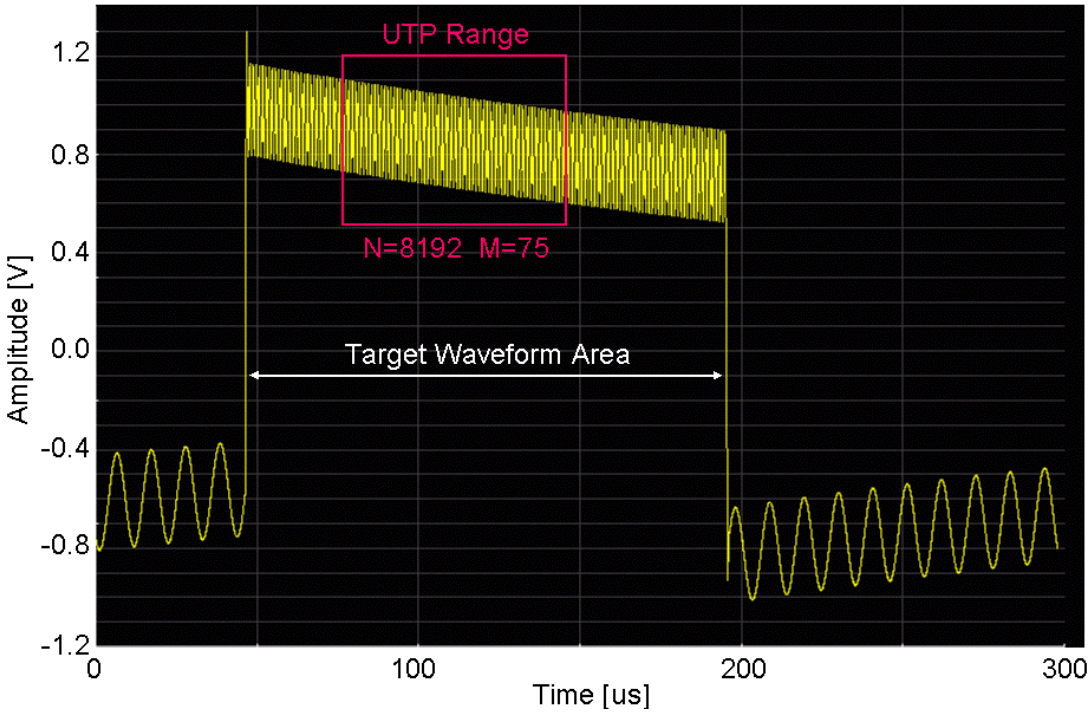


**Figure 1: Waveform with DC Offset Drift**

The unit test period (UTP) range highlighted in Figure 1 contains exact 75 cycles of the sinusoidal waveform and the number of sampling data points is 8192. So the coherency condition is strictly settled. Figure 2 shows the UTP waveform precisely. When applying FFT to the UTP with no windowing, the frequency spectrum appears as Figure 3.

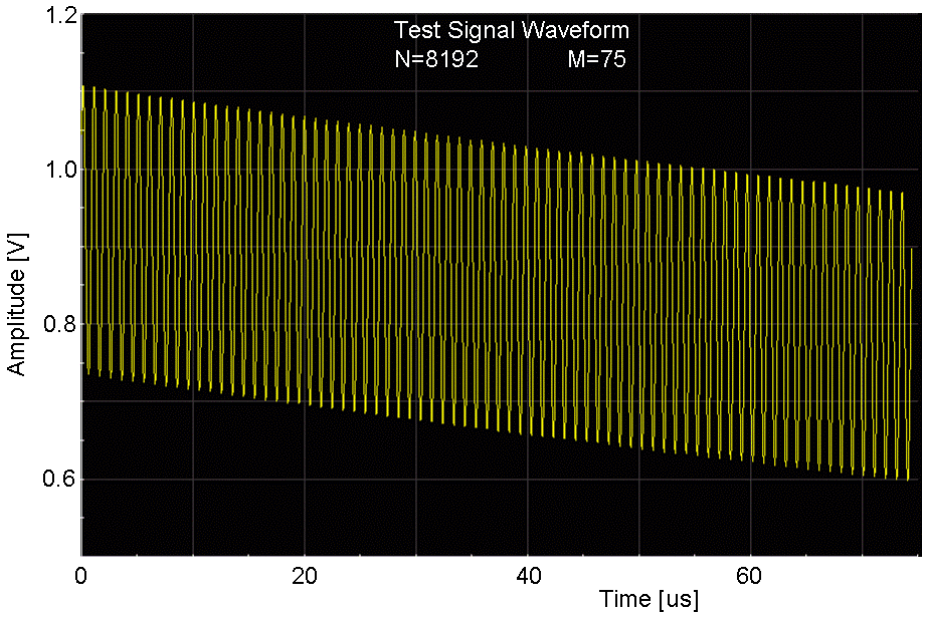


**Figure 2: UTP Waveform**

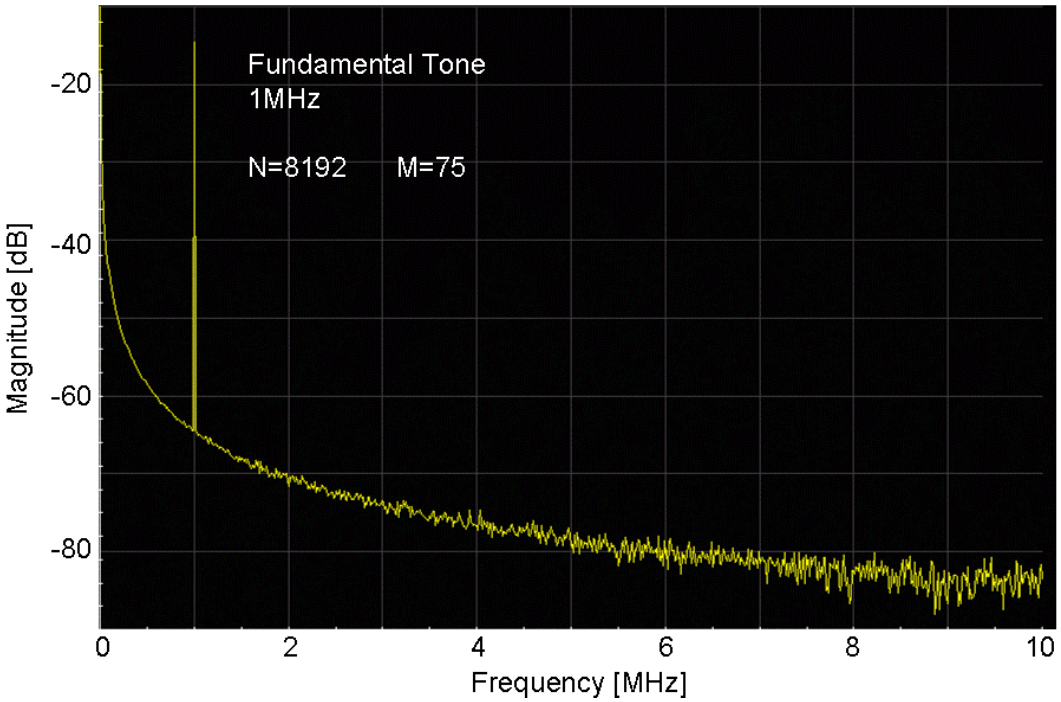


**Figure 3: Frequency Spectrum of the UTP (FFT with No Window)**

The fundamental tone looks sharp because of the exact coherent condition; however you can definitely recognize a weird noise floor slope because of the DC offset drift. You cannot calculate a reasonable SNR under this situation.

By spending a longer wait time, the DC drift caused by the capacitor could be settled negligibly small; however, if it is a production test, you cannot afford to patiently wait until it would be settled. In this test condition, you already know the true cause of the DC drift, it comes from the capacitor on the board and the device performance has nothing to do with the DC drift. In such a situation you may want to remove the DC trend by utilizing any DSP technique. If you could suppress the DC trend and extract the target signal with no DC offset drift, you can recover a reasonable signal and noise spectrum.

Because of the big amplitude fluctuation in Figure 2, it is not a good idea to directly estimate an accurate DC trend just as it is. So firstly you should suppress the main signal for highlighting the drift. The steps to remove the DC offset drift are as follows;

1) Suppress the major signal roughly for highlighting the DC drift.
2) Estimate the DC drift trend by a curve fitting routine.
3) Remove the estimated DC trend from the original signal.
4) Apply FFT to the trend removed signal for spectrum analysis.

In this example, exact 75 cycles (M) of sine waveform is captured in the UTP so that DSP_SIN_FIT() can effectively perform the sinusoidal waveform estimation. See List 1. It puts out the estimated sinusoidal waveform directly as array “dWave0” (Line 9) which is illustrated in Figure 4 as the light-blue line. By subtracting the estimated signal from the original waveform at Line 10, the residual noise signal is extracted as the yellow line in Figure 5. Now that it looks almost a clear curve so that you can estimate the DC drifting trend by utilizing a least square curve fit method at Line 17, whose code is not included in this article. Since there are lots of computer math books available in bookstores, you can easily find out an appropriate routine example from it or you could create one by yourself if you would be familiar with the algorithm.

```
01:   INT           i,N,M,Nx;
02:   DOUBLE        dA,dP,dQ,dO;
03:   ARRAY_D       dVdata,dWave0,dResidual,dXaxis;
04:   ARRAY_D       dCoef,dTrend,dVmodified,dSp;
05:
06:   //  dVdata[] is the original waveform data
07:   //  dVdata[] contains N points and M cycles of sine waveform.
08:
09:   DSP_SIN_FIT(dVdata,dWave0,&dA,&dQ,&dO,M);   // Sine Fit API
10:   DSP_SUB_VEC(dVdata,dWave0,dResidual);       // Remove major signal
11:
12:   dXaxis.resize(N);                          // X-axis step (simple)
13:   for (i=0;i<N;i++) dXaxis[i]=(DOUBLE)i;// Preparation for curve fit
14:
15:   Nx=2;                                      // 2nd order Curve Estimation
16:   dCoef.resize(Nx+1);                        // Coefficients Container
17:   LeastSquareCurveFit(dXaxis,dResidual,dCoef,Nx); // Curve Fit
18:
19:   dTrend.resize(N);                          // Estimated 2nd Order Curve
20:   for (i=0;i<N;i++) dTrend[i]=dCoef[0]+dCoef[1]*i+dCoef[2]*i*i;
21:
22:   DSP_SUB_VEC(dVdata,dTrend,dVmodified);     // Trend Removal
23:   DSP_SPECTRUM(dVmodified,dSp,DB,1.0,RECT,0);// Modified Data Spectrum
24:
```

**List 1: Trend Removal by utilizing Sine Fit Signal Estimation**

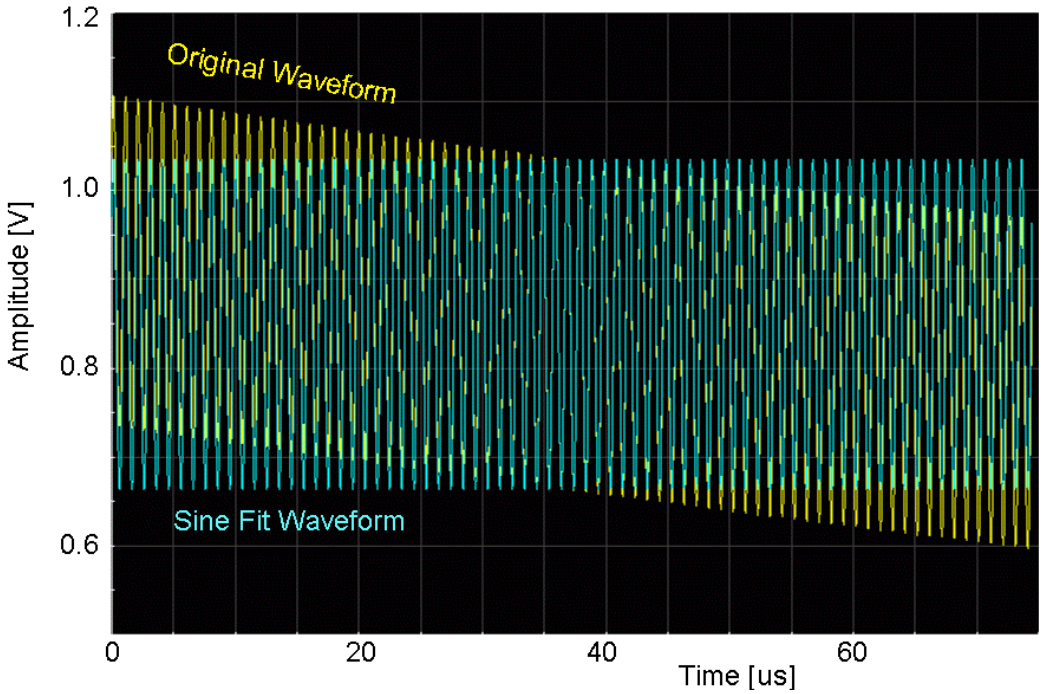


**Figure 4: Waveform Estimated by DSP_SIN_FIT()**

Anyway, the DC drifting trend is approximated as a 2nd order polynomial as the red line in Figure 5. Now that you have clearly estimated the DC drifting trend as a clear monotonic curve, so you can subtract this trend from the original waveform, deriving the target signal as Figure 6.

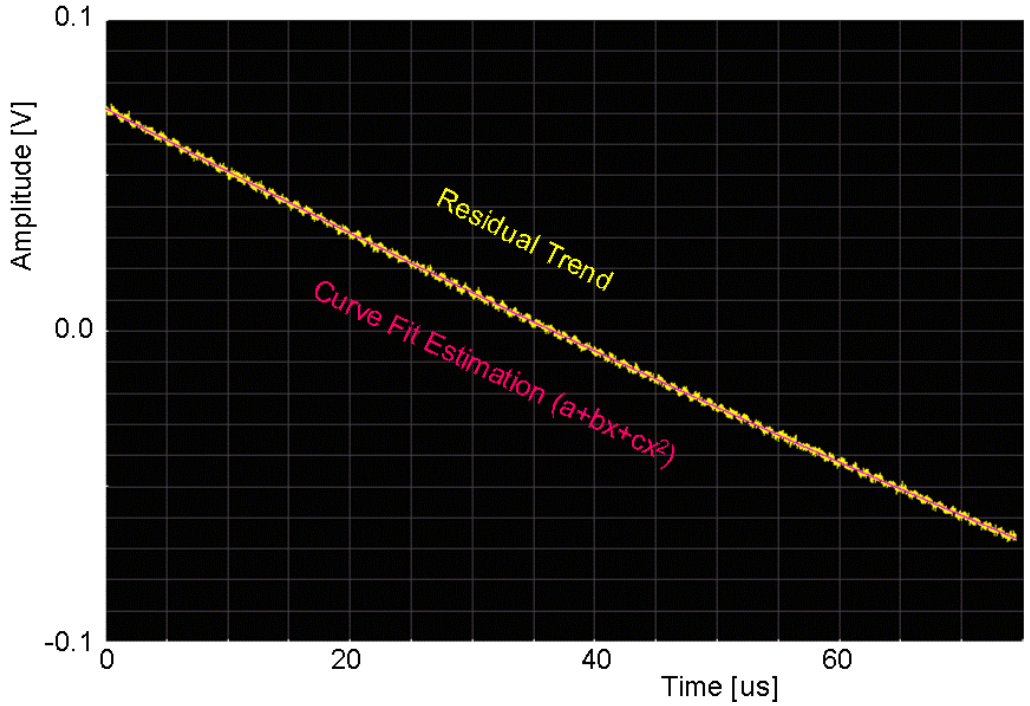


**Figure 5: Residual Noise (Yellow) and Curve Fit Result (Red)**

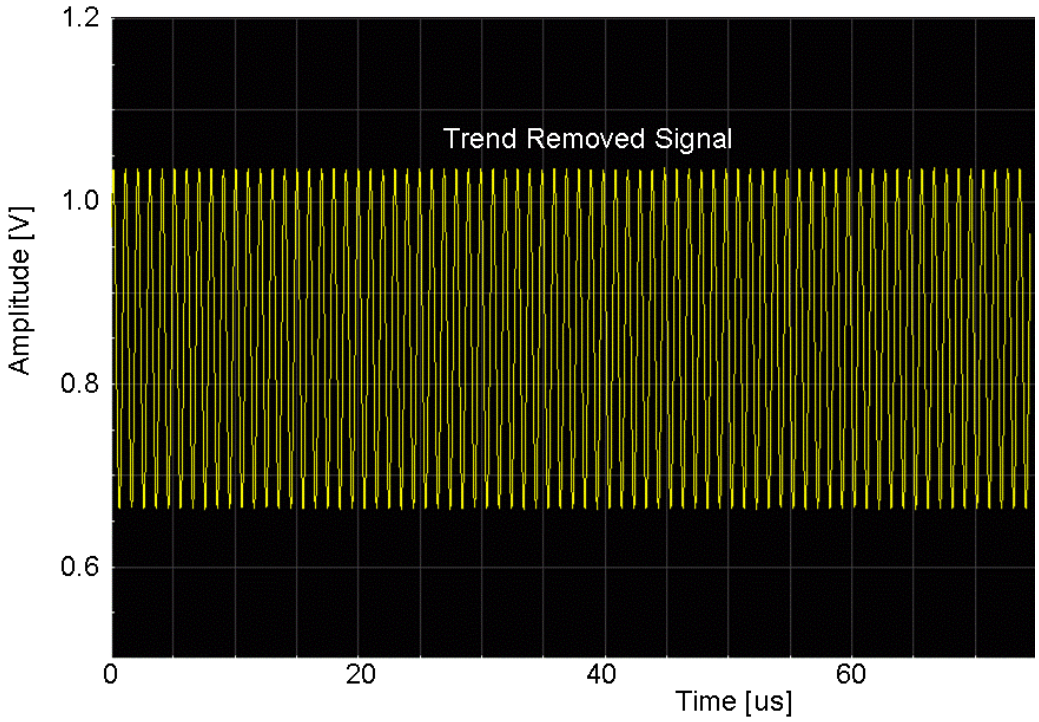


**Figure 6: Trend Removed UTP Waveform**

This waveform contains no DC trend anymore so that FFT can reveal a good-looking spectrum as Figure 7 shows, and eventually you could calculate a reasonable SNR from the spectrum.

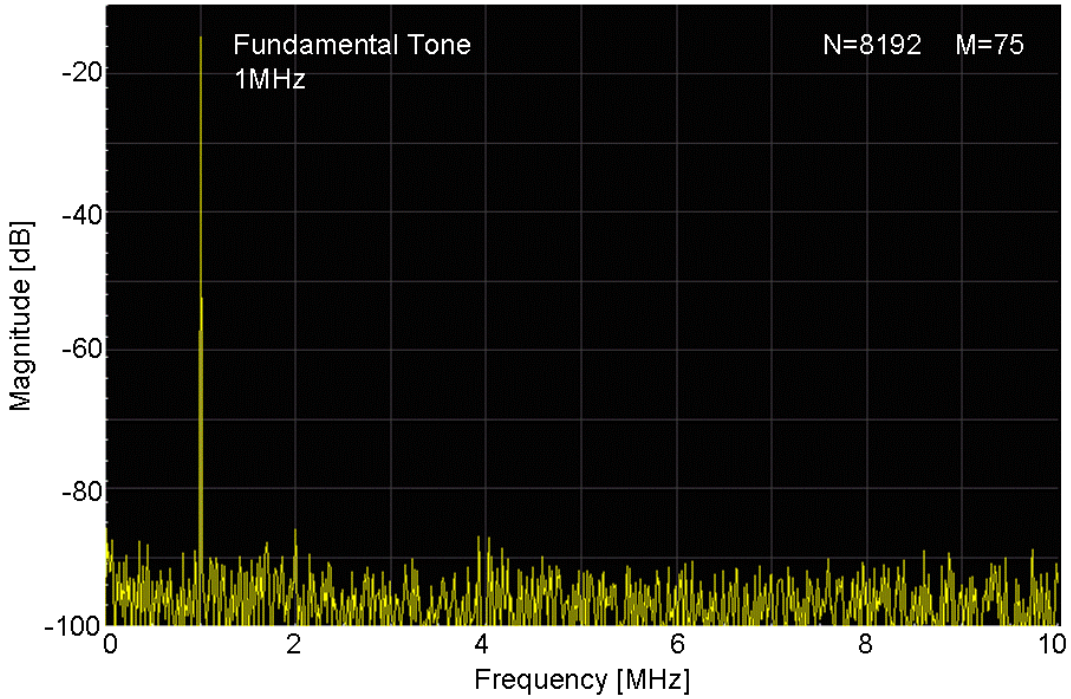


**Figure 7: FFT Spectrum for the Trend Removed Signal**

The target signal is a single sinusoidal waveform in this example so that the sine fit routine can simply and effectively extract the sinusoidal waveform. DSP_SIN_FIT() can address a single tone signal. If the target signal would be dual-tone or multi-tone, this API cannot cope with it. Then you can apply DSP_FFT() instead and extract major tone signals one by one. List 2 describes the usage of DSP_FFT() as the substitute of DSP_SIN_FIT(). The estimated signal by List 2 appears as the light-blue line in Figure 8. Then the least square curve fit estimates the DC drift as the red line in Figure 9. The difference between Figures 4 & 5 and Figures 8 & 9 is if the mean of DC offset is taken account of in advance or not. Data processing after that is exactly the same as the way in the previous sine fit method. (Lines 19 through 23 in List 1)

```
01:   INT           i,N,M;
02:   DOUBLE        dA,dP,dQ,dX,dY;
03:   ARRAY_D       dVdata,dWave,
04:   ARRAY_COMPLEX CSp;
05:
06:   //  dVdata[] is the original waveform data
07:   //  dVdata[] contains N points and M cycles of sine waveform.
08:
09:   DSP_FFT(dVdata,CSp,RECT);   // FFT with no window
10:   dX=CSp[M].real();           // Bin #M vector
11:   dY=CSp[M].imag();           //
12:   dA=sqrt(dX*dX+dY*dY);       // Amplitude
13:   dQ=atan2(dY,dX);            // Phase Offset
14:   dP=2.0*M_PI*M/N;            // Phase Increment of the vector
15:   dWave.resize(N);            // Estimated Waveform Container
16:   for (i=0;i<N;i++) dWave[i]=dA*cos(dP*i+dQ);   // Use cosine
17:
```

**List 2: Signal Estimation by FFT**

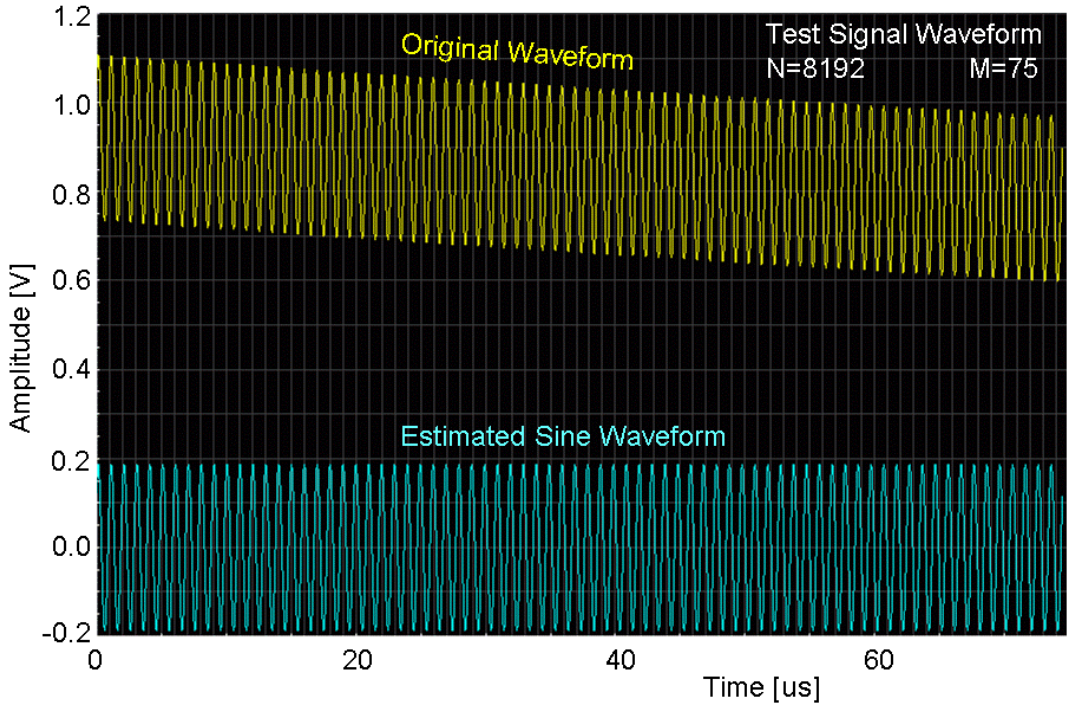


**Figure 8: Signal Estimation by FFT**

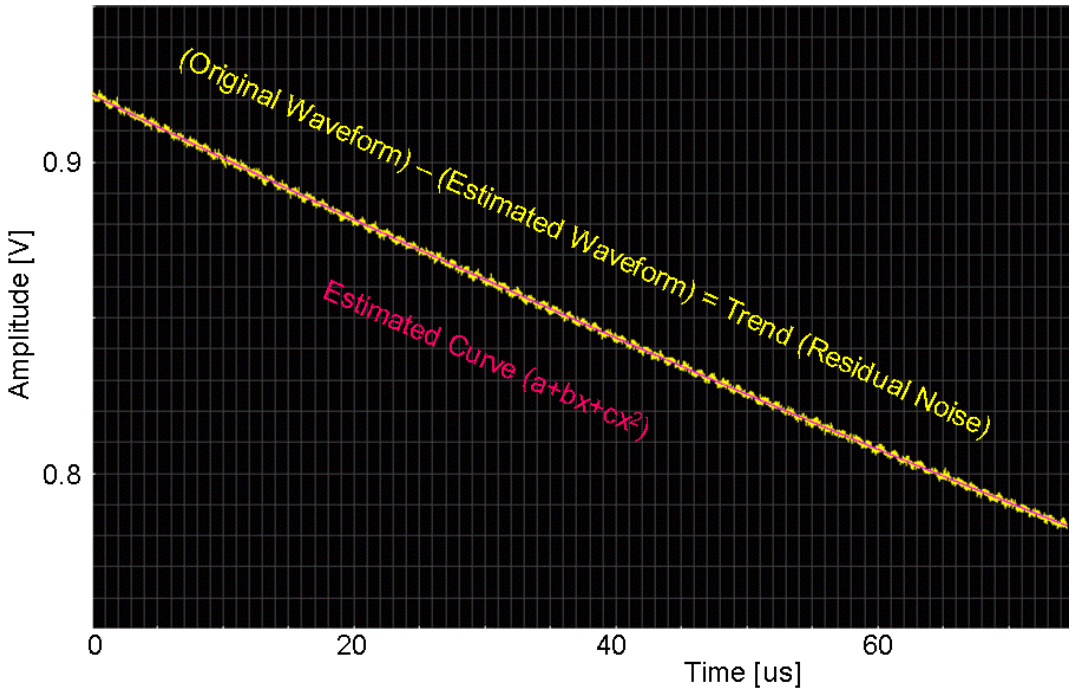


**Figure 9: Trend Estimation by Residual Noise**