A Verilog-A Cycle-to-cycle Jitter Measurement Module

Fuding Ge

fudingge@yahoo.com

1. Definition of Cycle-to-cycle Jitter

The cycle-to-cycle jitter of a clock signal is defined as the r.m.s. variation (standard deviation) in the period, which can be described by the following equation:

$$\sigma_{CTC} = \sqrt{\frac{1}{M}\sum_{i=1}^{M,M\to\infty}(t_i - t_{avg})^2}$$

Where t_i is the instantaneous period of the signal and t_{avg} is the average period value of the clock signal[1]. Using the so-called short-cut formula we can rewrite the above equation as:

$$\sigma_{CTC} = \sqrt{\frac{1}{M}\sum_{i=1}^{M,M\to\infty} t_i^{\,2} - (t_{avg})^2} = \sqrt{\frac{1}{M}\sum_{i=1}^{M,M\to\infty} t_i^{\,2} - (\frac{1}{M}\sum_{i=1}^{M,M\to\infty} t_i)^2}$$

We use this equation and transformed it into a VerilogA module to calculate the cycle-to-cycle jitter. The VerilogA code is shown in the following:

2. Verilog-A Module

```
// ////////////////////////////////////////////
// Cycle-to-cycle jitter meter             /////
// Copyright by Fuding Ge, All right reserved //
////////////////////////////////////////////////

// This verilogA model output the rms cycle-to-cycle jitter of the input
// clock signal. The definition of the cycle-to-cycle jitter is the rms
// variation in itd period.

// JitCTC=sqrt((sum(ti-tavg)**2)/M) where M is the number of periods of test.
// M should be as large as possible. ti is the instantaneous and tavg
// is the average of the periods: tavg=(t1+t2+...+tM)/M

// In this model we use the following equation:
////////////////////////////////////////////////
// JitCTC=sqrt((sum(ti)**2)/M-(tavg)**2)     ///
```

[1] Here we divide the sum by M instead of more formal mathematical definition:

$$\sigma_{CTC} = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M,M\to\infty}(t_i - \bar{t})^2}$$


```
////////////////////////////////////////////////

// You need to change the value of hlfvcc to match the power supply
// of the clock signal. In this version vcc=2.5 and hlfvcc=1.25

`include "constants.h"
`include "discipline.h"

nature  Time
        abstol = 1e-25;
        access = TT;
        units = "ps";
        blowup = 1.0e10;
endnature

nature  Number
        abstol = 1e-3;
        access = NUM;
        units = "";
        blowup = 1.0e200;
endnature

discipline time_current
        potential Time;
        flow Current;
enddiscipline

discipline number_current
        potential Number;
        flow Current;
enddiscipline


////////////////////////////////////////////////////////

module Jittermeter(vin,jitter,num_period);
input vin;
output jitter,num_period;

electrical vin;
time_current jitter;
number_current num_period;


   real tlatest;
   real tearly;
   real period_early, period_latest;

   real period_square;
   real period_square_sum;
   real period_square_avg;

   real period_total;
   real period_avg;
   real period_avg_square;

   real jitter_val;
   real hlfvcc;
   integer counter;
   integer counter_begin;

////////////////////////////////////
   analog begin

      ///////   initialize the parameters///////////////
      @ ( initial_step ) begin
        tearly=0.0;
        tlatest=0.0;
        hlfvcc=1.25;          //Change your hlfvcc here !
        counter =0;
        counter_begin=20000;
```

```
        period_early=0.0;
        period_latest=0.0;

        period_square=0.0;
        period_square_sum=0.0;
        period_square_avg=0.0;

        period_total=0.0;
        period_avg=0.0;
        period_avg_square=0.0;
      end


////////////////////////////////////////////////////////////////
      @ ( cross ((V(vin)-hlfvcc),+1 )) begin
         tlatest = $realtime*1e12;
         period_latest  = tlatest-tearly; //Current period value
         tearly = tlatest;
         period_early = period_latest;
         counter = counter +1;

if (counter >= counter_begin+2) begin

         period_square = period_latest*period_latest;
         period_square_sum = period_square_sum + period_square;
         period_square_avg =  period_square_sum/(counter-counter_begin-1);

         period_total = period_total + period_latest;
         period_avg = period_total/(counter-counter_begin-1);
         period_avg_square =  period_avg*period_avg;

         jitter_val = period_square_avg - period_avg_square;
end
end
         ////////////////////////////////////

      //// output jitter value and number of measure periods /////
         if  (counter-counter_begin < 10000)
            begin
                 TT(jitter) <+ 0;
                 NUM(num_period) <+ 0;
            end
         else
            begin
                TT(jitter) <+ sqrt(jitter_val);
                NUM(num_period) <+ counter-counter_begin;
            end

    end
endmodule
--------------------------------------------------------------------------
```

3. Adjacent cycle jitter

Herzel (*Frank Herzl and Behzad Razavi, "A study of oscillator jitter due to supply and substrate noise", IEEE Trans. Circuit and system – II, Vol46 (1), PP56-62, 1999.*) defined the cycle-to-cycle jitter as the variance between successive periods:

$$\sigma_{\mathrm{CTC},CC} = \sqrt{\frac{1}{M}\sum_{i=1}^{M,M\to\infty}(t_{i+1}-t_i)^2}$$

For white noise source, two successive periods are uncorrelated, and since adjacent cycle jitter represents the difference between two periods, it is twice as large as the variance of one period, so:

$$\sigma_{CTC,CC} = \sqrt{2}\sigma_{CTC}$$

This jitter is sometimes called Rambus Jitter because Rambus use this definition. We can also use this equation to implement jitter simulation.

4. Simulation Results

The following figure shows the simulation results of the cycle-to-cycle jitter using Spectre. It can be seen that the accuracy of the transient analysis is very important. If the accuracy is set to be libral, the jitter is about 5 ps, but if the accuracy is set to be conservative or moderate, the jitter is only about 0.3 ps. It can also be seen that there are not much difference between moderate and conservative accuracy.

Also please noted that I begin to measure the jitter after 100us, which is needed for the PLL to get at the stead state.