

# Pattern Generator Model for Jitter-Tolerance Simulation

---

**Paul Muller**  
**Yusuf Leblebici**

École Polytechnique Fédérale de Lausanne (EPFL)

---

**Version 1, 18 July 2006**

This paper discusses the behavioral modeling of a pattern generator including various types of data jitter for jitter tolerance analysis of high-speed serial link receivers during the design phase. The presented model can be used both during the system-level design exploration and the following transistor-level design phases. First, jitter tolerance of clock recovery circuits is explained and the different natures of jitter applied to the data test pattern are introduced. Then, a behavioral model of the serial data pattern generator is presented, which includes these jitter components and delivers the jitter-affected data used to stimulate the receiver's clock recovery circuit. Finally, the detailed VHDL-AMS model is introduced and its parameter definitions are discussed.

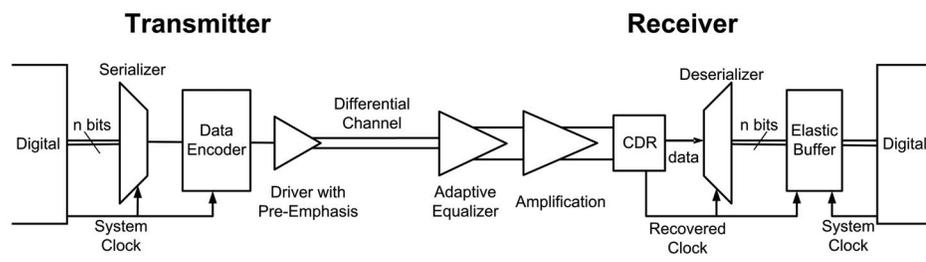
*Last updated on August 25, 2006. You can find the most recent version at [www.designers-guide.org](http://www.designers-guide.org). Contact the author via e-mail at [paul.muller@a3.epfl.ch](mailto:paul.muller@a3.epfl.ch).*

*Permission to make copies, either paper or electronic, of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage and that the copies are complete and unmodified. To distribute otherwise, to publish, to post on servers, or to distribute to lists, requires prior written permission.*

## 1 Introduction

While serial data communications have been available in the wide, metro and local area network domains for many years, these links nowadays also appear *in the box*, i.e. in between and inside personal computers. Unlike in the before-mentioned domains, the data transmitted by these short distance links (e.g. SerialATA) is retimed in each receiver, resetting the jitter budget of the link. The retiming ability of the link is therefore determined by the jitter contributions of the different link elements and the receiver's capability to eliminate incoming jitter, called jitter tolerance (JTOL).

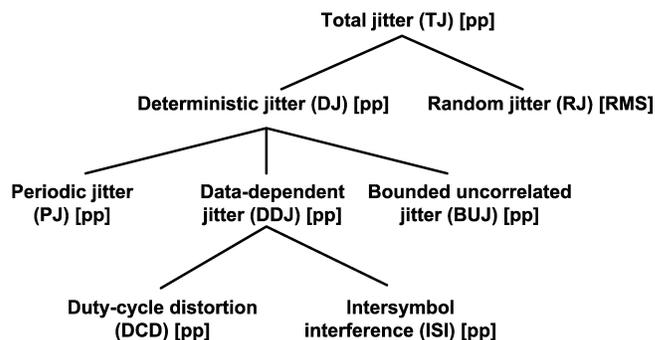
FIGURE 1 Short-distance serial data link.



As the serial data stream is transmitted over a single data link without clock forwarding (Figure 1), the receiver must extract the timing information from the data stream to retime the data. The accuracy of this operation, performed in the clock and data recovery (CDR) block, is limited by the jitter affecting the received data edges. The JTOL specification being verified on the manufactured samples, it is desirable to include a method for JTOL estimation in the system-level verification step of the design flow.

## 2 Specification of Jitter Tolerance

FIGURE 2 Jitter contributions [1].

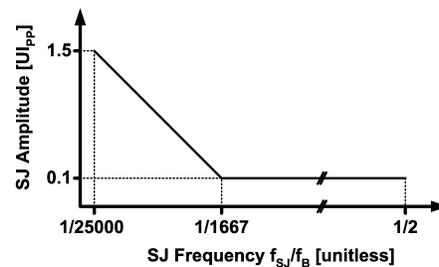


The jitter present in a data link can be decomposed into several components, as shown in Figure 2. Random jitter (RJ) is due essentially due to device noise sources (e.g. thermal noise) and modeled by a Gaussian distribution. Deterministic jitter (DJ) is composed of data-dependent jitter (which includes duty-cycle distortion and intersymbol interfer-

ence), bounded uncorrelated jitter and periodic jitter [1]. Bounded uncorrelated jitter is due to coupling from neighboring channels or digital switching noise, which explains the absence of correlation with the received data signal. Periodic jitter, due to electromagnetic interference, is to be distinguished from sinusoidal jitter, introduced below. The jitter tolerance link specification defines the amount of deterministic and random jitter in presence, at several test points of the link (e.g. the CDR input).

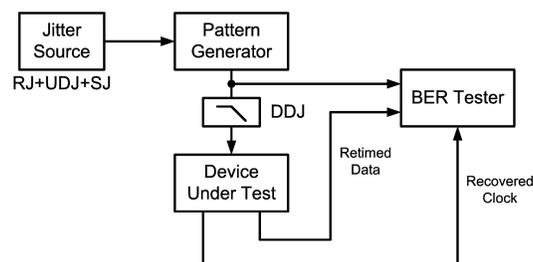
In addition to these jitter quantities covering all types of jitter affecting the input signal of the clock recovery circuit, an additional jitter component has to be considered. Jitter tolerance is assessed by superimposing to the previously discussed jitter components a sinusoidal jitter (SJ) contribution, which amplitude and frequency can be adjusted for the purpose of the measurement. In order to be considered compliant to the specifications, a receiver must guarantee a given bit error rate (BER, typically  $10^{-12}$  or  $10^{-15}$ ) for  $\{SJ_{amp}, SJ_{freq}\}$  pairs exceeding the jitter tolerance mask shown in Figure 3.

FIGURE 3 Jitter tolerance mask [2].



While high-level estimations based on the CDR topology (e.g. [3]) are an efficient way for calculation a large number of points in the  $\{SJ_{amp}, SJ_{freq}\}$  space, more CPU time demanding behavioral time-domain simulations are required to guarantee the coverage of system-level aspects like connectivity, block interoperability, etc. A simulation configuration for JTOL analysis is defined based on the actual JTOL measurement set-up suggested by [2] (Figure 4). Such configurations usually consider data-dependent jitter (DDJ) due to signal bandwidth limitation and uncompensatable deterministic jitter (UDJ) as two separable components of the total deterministic jitter budget.

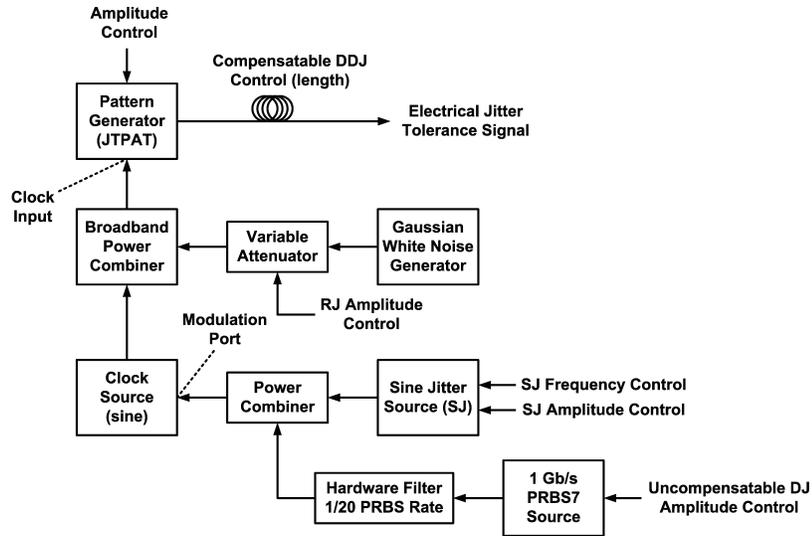
FIGURE 4 Jitter tolerance analysis configuration.



### 3 The Fibre-Channel JTPAT Source

According to common communication protocol specifications like Gigabit Ethernet, FibreChannel™ or InfiniBand™, the pattern generator in combination with the jitter source can be implemented as shown in Figure 5.

FIGURE 5 Block diagram of the electrical signal tolerance source (based on [2]).



Some of the contributions to deterministic jitter, especially those related to bandwidth limitations, can be compensated for by signal processing manipulations like pre-distortion and equalization. To obtain a realistic jitter distribution, deterministic jitter is separated in a compensatable data dependent part (DDJ), which is implemented by a filter at the output of the pattern generator, and an uncompensatable part (UDJ), which is generated through a filtered PRBS data sequence. Common values for the different jitter contributions are presented in Table 1.

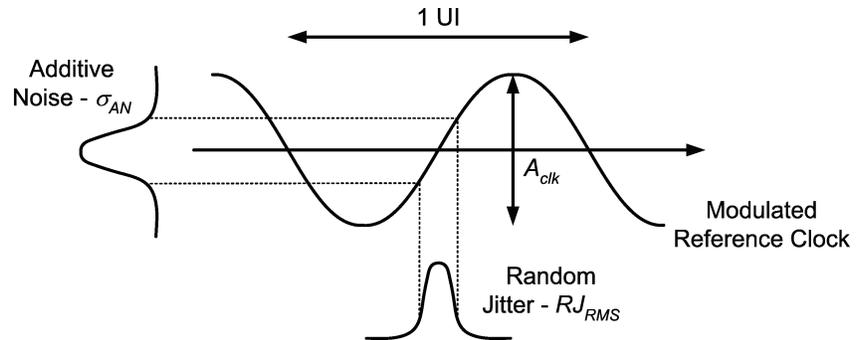
TABLE 1 Common values for the different jitter contributions.

Jitter Nature	Amplitude
Deterministic Jitter	0.4 UI <sub>pp</sub>
Random Jitter	0.3 UI <sub>pp</sub> = 0.021 UI <sub>RMS</sub> @ BER = 10 <sup>-12</sup>
Sinusoidal Jitter Amplitude	swept according to Figure 3

The UDJ component is combined with the sinusoidal jitter and applied to the modulation input of the clock source that serves as a time base for the pattern generator. For further reading on the validity of sinusoidal jitter testing at large jitter amplitudes, the interested reader is referred to [4].

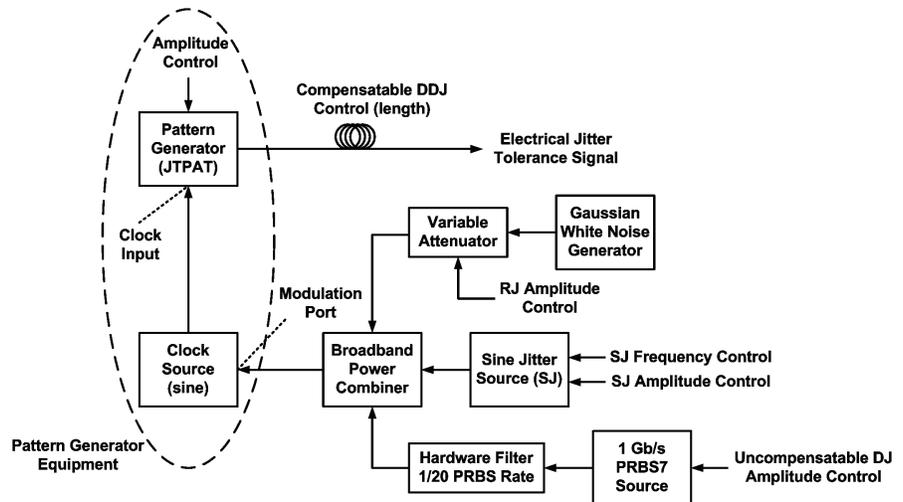
Random jitter is added in power to the modulated clock signal, which is then applied to the clock input of the pattern generator. The conversion from additive noise to jitter is

FIGURE 6 Additive noise to random jitter conversion (based on [5])



illustrated in Figure 6. While this standardized approach has been implemented in the behavioral model included in appendix, the reader should be aware that the principle of adding random noise is arguable due to the resulting invalid additional clock edges, as discussed in [5]. For this reason, an alternative implementation has been proposed (Figure 7), in which RJ is first added to SJ and UDJ. This summed jitter component is then applied to the phase modulation input of the clock source. A further advantage of this method during physical testing is the removal of an external reference clock source to be modulated. Modification of the included model to implement this second configuration is left for future work.

FIGURE 7 Block diagram of the electrical signal tolerance source with alternative RJ application.



The pattern generator of the standardized jitter tolerance source for system compliance verification generates a so-called compliant jitter tolerance pattern (CJTPAT), composed of the data words representing the worst case condition for jitter tolerance measurement of the receiver. The compliant term indicates that the effective stress pattern is embedded into data frames including delimiters, fill words and CRC, which is necessary when physical links are tested. In both cases described in Figures 5 and 7, use of the jitter tolerance pattern (JTPAT), which corresponds to the payload of the CJTPAT, is recom-

mended for verification of the CDR. While alternative suggestions of test pattern to be used have been published [6,7], we decided to use the standardized JTPAT test pattern from [2]. As already mentioned, the generated output pattern is bandwidth limited by a first order filter at the output to implement DDJ.

## 4 Behavioral Modeling

Language-based modeling is an appropriate way of generating data patterns affected by timing jitter. The resulting model can be used both with behavioral models for system-level verification including the CDR, as well as during transistor-level verification in a mixed-language environment. The proposed model has been implemented in VHDL-AMS (IEEE 1076.1) and uses both analog and event-driven elements of this language. The VHDL-AMS code of the complete model is included in Appendix A. In the following, some implementation details of the different subsections are addressed.

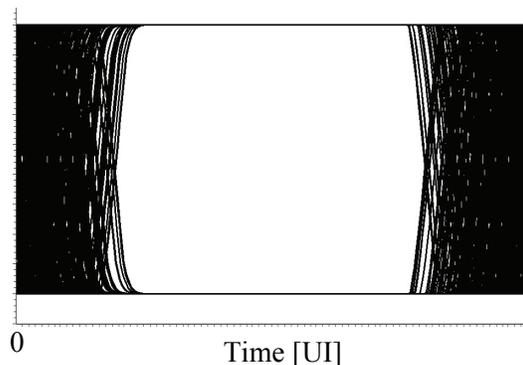
### 4.1 UDJ Source

The uncompensatable deterministic jitter source, designated by UDJ, is built of a 1.0Gb/s PRBS7 pattern generator, which data is sent through a 50MHz bandwidth 1st-order low-pass filter. After combination with the sinusoidal jitter component, this signal modulates the phase of the reference clock. However, due to low-pass filtering, the peak amplitude of the UDJ signal is lower than the input PRBS7 pattern. Considering a periodic pattern of maximum run length gives a first-order estimate of the peak-peak amplitude of the output pattern, (1). In this expression,  $7T_{ck}$  represents the longest run length of a PRBS7 pattern, while the filter time constant is given by  $\tau=20T_{ck}/2\pi$ . To compensate for this undesired attenuation, a gain block is added at the filter output. In the final model, the gain value is numerically adjusted by simulation to improve the amplitude compensation.

$$\frac{v_{UDJ}}{v_{PRBS7}} = \tanh\left(\frac{7T_{ck}}{2\tau}\right) \cong \tanh(1.1) \cong 0.8 \tag{1}$$

Figure 8 shows the output eye diagram of the pattern generator when only uncompensatable deterministic jitter is applied to the data.

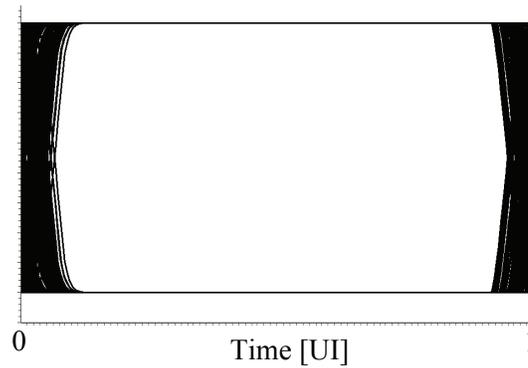
FIGURE 8 VHDL-AMS JTPAT data for UDJ = 0.4UI



## 4.2 Sinusoidal Jitter Source

As already mentioned, SJ is added to UDJ before application to the modulation input of the clock source. When setting the parameters of the SJ component, care should be taken to avoid the sinusoidal jitter frequency to result from an integer or rational division of the clock frequency. Otherwise the sinusoidal jitter contribution may be insufficient covered by the simulation. Figure 9 shows the output eye diagram of the pattern generator when sinusoidal jitter of amplitude 0.1 UI and frequency  $0.0211 f_{\text{clk}}$  is applied to the data.

FIGURE 9 VHDL-AMS JTPAT data for  $SJ_{\text{amp}} = 0.1 \text{ UI}$  and  $SJ_{\text{freq}} / f_{\text{clk}} = 0.0211$ .



## 4.3 Power Combiner

After combination of the sinusoidal jitter component and the UDJ component, this signal is applied to the modulation input of the reference clock generator. The power combiners are modeled as Thévenin-equivalent voltage sources with a source resistance of  $50\Omega$  driving a shared load resistance of  $50\Omega$ , where the ideal voltage sources deliver twice the output voltage, as usual in high-frequency measurement equipment. An example with two voltage sources is shown in Figure 10. For a number  $n$  of sources driving the load according to this principle, the output signal of this configuration is given by (2).

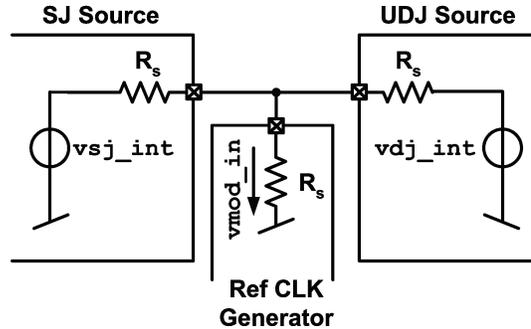
$$v_{\text{mod\_in}} = \sum_{i=1}^n \frac{2v_n}{1+n} \quad (2)$$

A sinusoidal clock source is used to allow proper phase modulation by the previously mentioned components, (3). All voltages are centered on  $v_{\text{mid}}$  (typically  $V_{\text{DD}}/2$ ) and  $A_{\text{clk}}$ ,  $A_{\text{mod}}$  are the respective amplitudes of the reference clock and the voltage at the modulation input.  $f_{\text{clk}}$  is the nominal clock frequency and  $k_{\text{VCO}}$  is the modulation gain of the clock source.

$$v_{\text{sin\_clk}} = v_{\text{mid}} + A_{\text{clk}} \sin(2\pi f_{\text{clk}} t + d\phi) \quad (3)$$

$$\text{with } d\phi = 2\pi \frac{v_{\text{mod\_in}} - v_{\text{mid}}}{A_{\text{mod}}} k_{\text{VCO}}$$

FIGURE 10 Equivalent schematic of the power combiners



#### 4.4 Random Jitter Source

Additive white Gaussian noise (AWGN) is generated using the well-known Box-Muller transform (4) and the uniformly distributed random number generator available in VHDL-AMS. If  $r_{u1}$  and  $r_{u2}$  are two uniformly distributed random variables,  $r_g$  obtained by the Box-Muller transform follows a Gaussian distribution of mean  $\mu$  and variance  $\sigma$ .

$$r_g = \mu + \sigma \sqrt{-2 \ln r_{u1}} \cos(2\pi r_{u2}) \tag{4}$$

This noise signal is added to the clock signal using a power combiner as described above. Care has to be taken to obtain the desired jitter amplitudes in the output data stream. The required AWGN variance for a desired amount of random jitter  $RJ_{RMS}$  is given by (5), where  $A_{clk}$  is the amplitude of the clock reference. The drawbacks of this approach and the possible replacement by phase modulating random noise have already been discussed.

$$\sigma_{AN} = RJ_{RMS} \frac{1 UI}{\pi A_{clk}} \tag{5}$$

#### 4.5 Data Dependent Jitter

As already mentioned, the resulting pattern is bandwidth limited by a first order filter of time constant  $\tau=1/2\pi f_c$ , resulting in compensatable deterministic jitter. Considering a periodic pattern centered around its DC value, the filter bandwidth required to obtain a given amount of deterministic jitter is given by (6), where  $T_B = f_B^{-1}$  is the bit period and  $T_{MIN}$  and  $T_{MAX}$  respectively the shortest and longest run length in the pattern.

$$DDJ_{pp}[UI] = \frac{\tau}{T_B} \ln \left( \frac{1 + e^{\frac{-T_{MIN}}{\tau}}}{1 + e^{\frac{-T_{MAX}}{\tau}}} \right) \tag{6}$$

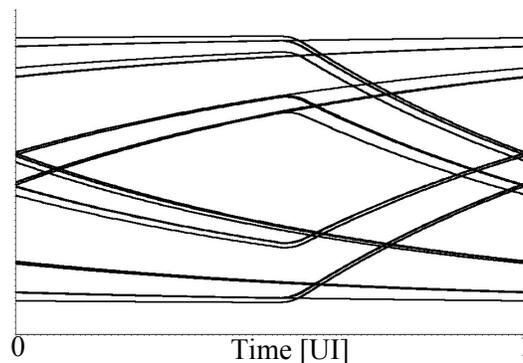
Convenient cut-off frequency settings for some common values of deterministic jitter are proposed in Table 2. However, applying the JTPAT or a PRBS signal results in more severe jitter due to the transitions between long and short CID sequences, which are difficult to estimate mathematically. Some simulated correspondence points between cut-

off frequency and DDJ for the JTPAT are also given in Table 2. Figure 11 shows the pattern generator output when the bandwidth corresponds to 2/10 of the data rate.

TABLE 2 Filter cut-off frequencies and associate DDJ values.

$f_c / f_B$	DDJ - periodicity assumption [UI <sub>pp</sub> ]	DDJ - JTPAT [UI <sub>pp</sub> ] (simulated)
0.1256	0.50	
0.1416	0.4	
0.1600	0.30	
0.1644		0.41
0.2000	0.20	0.27
0.2688	0.10	
0.4000	0.031	0.033
0.8000	0.001	0.001

FIGURE 11 VHDL-AMS JTPAT data for  $f_c = 0.2 f_B$ .



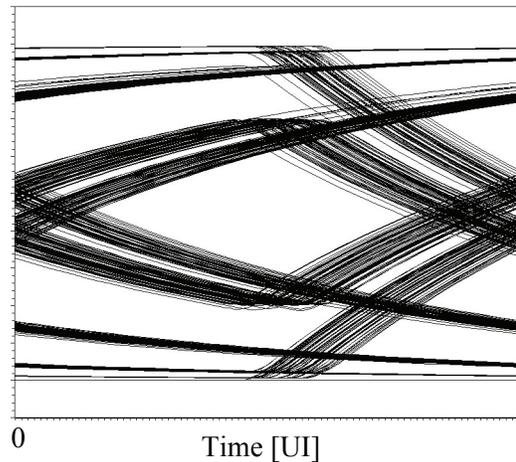
#### 4.6 Jitter Tolerance Test Pattern

The implemented data pattern is the standard jitter test pattern (JTPAT) composed of a sequence of 10 D30.3 characters, followed by 3 D21.5 characters with the appropriate running disparity required for DC balance. These characters correspond to the 8b/10b encoded form of 7Eh and B5h bytes respectively. The first character represents a low transition density, while the B5h character is a string of alternating bits, testing the high-frequency performance of the receiver. It should be mentioned that the maximum run length of the 8b/10b coding scheme is not stressed by this pattern. As already mentioned, alternative test patterns have also been published.

#### 4.7 Simulation Results

Figure 12 shows the eye diagram of the data output by the JTPAT data source with different jitter components applied. Of course the simulation time step influences the accuracy of the obtained eye opening. Simulation of a 1ms pattern at 2.5Gb/s (2'500 bits) with a maximum time step of 2 fs on a SPARC Ultra 10 (400MHz CPU) takes approximately 7 minutes.

FIGURE 12 VHDL-AMS JTPAT data for  $SJ_{amp}=0.1UI_{PP}$ ,  $SJ_{freq}/f_{clk}=0.0211$ ,  $UDJ=0.1UI_{PP}$ ,  $f_c(DDJ)=0.2f_B$  and  $RJ=0.001UI_{RMS}$ .



## 5 Conclusion

This paper introduces the needs for simulating the jitter tolerance of clock recovery circuits for serial data link receivers. The different jitter components to be considered are discussed and a modeling approach is proposed. Finally, a VHDL-AMS time-domain model of a data source for jitter tolerance simulation is proposed. The data pattern issued by this model can be used both during system-level verification as in transistor-level simulation of the CDR circuit.

### 5.1 If You Have Questions

If you have questions about what you have just read, feel free to post them on the *Forum* section of *The Designer's Guide Community* website. Do so by going to [www.designers-guide.org/Forum](http://www.designers-guide.org/Forum).

---

## Appendix A VHDL-AMS Model of the JTOL Data Source

---

```
-----  
-- Block Name:    FC_JTOL_SRC_NS  
-- File Name:    fc_jtol_src_ns.vhd  
-- Author:       Paul Muller  
-- Date:        18.07.06  
-- Rev:         1.0  
-----  
-- Modification History (rev, date, author):  
-- 1.0, 18.07.06, PMr  
-- Official release for DG web site  
-----  
-- Please post any questions to  
-- http://www.designers-guide.org/Forum  
-----  
  
-----  
-- Entity Declarations  
-----  
  
library IEEE;  
use IEEE.MATH_REAL.all;  
library IEEE_PROPOSED;  
use IEEE_PROPOSED.electrical_systems.all;  
entity sj_src is  
    port (  
        sj_freq : in real;  
        sj_amp  : in real;  
        freq    : in real;  
        terminal sj_out : electrical);  
end sj_src;  
  
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.MATH_REAL.all;  
library IEEE_PROPOSED;  
use IEEE_PROPOSED.electrical_systems.all;  
library LIB_AMS;  
use LIB_AMS.all;  
entity dj_src is  
    port (  
        udj_amp : in real;  
        nreset  : in std_logic;  
        terminal dj_out : electrical);  
end dj_src;  
  
library IEEE;  
use IEEE.MATH_REAL.all;  
library IEEE_PROPOSED;  
use IEEE_PROPOSED.electrical_systems.all;  
library LIB_PCK;  
use LIB_PCK.general.all;  
entity rj_src is
```

```
    port (
        rj_amp : in real;
        terminal rj_out : electrical);
end rj_src;

library IEEE;
use IEEE.MATH_REAL.all;
library IEEE_PROPOSED;
use IEEE_PROPOSED.electrical_systems.all;
library LIB_AMS;
use LIB_AMS.all;
entity sin_clk_src is
    port (
        freq : in real;
        terminal mod_in : electrical;
        terminal sin_clk : electrical);
end sin_clk_src;

library IEEE;
use IEEE.std_logic_1164.all;
library IEEE_PROPOSED;
use IEEE_PROPOSED.electrical_systems.all;
library LIB_AMS;
use LIB_AMS.all;
entity clk_gen is
    port (
        freq : in real;
        sj_amp : in real;           -- Peak-peak value
        sj_freq : in real;
        udj_amp : in real;         -- Peak-peak value
        nreset : in std_logic;
        terminal sin_clk : electrical);
end clk_gen;

library IEEE;
use IEEE.std_logic_1164.all;
entity cjtpat_core is
    port (
        clk : in std_logic;
        rst_b : in std_logic;
        data_out : out std_logic);
end cjtpat_core;

library IEEE;
use IEEE.std_logic_1164.all;
library IEEE_PROPOSED;
use IEEE_PROPOSED.electrical_systems.all;
library LIB_AMS;
use LIB_AMS.all;
entity cjtpat_gen is
    port (
        nreset : in std_logic;
        terminal clk_in : electrical;
        cjtpat : out std_logic);
```

```

end cjtpat_gen;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.MATH_REAL.all;
library IEEE_PROPOSED;
use IEEE_PROPOSED.electrical_systems.all;
library LIB_AMS;
use LIB_AMS.all;
entity fc_jtol_src_ns is
  generic (
    data_amp : real := 1.8;    -- data amplitude peak-peak
    data_vcm : real := 0.9);  -- common-mode voltage
  port (
    freq : in real;
    sj_amp : in real;         -- Peak-peak value
    sj_freq : in real;
    rj_amp : in real;        -- RMS value
    udj_amp : in real;       -- Peak-peak value
    ddj_fc : in real;
    nreset : in std_logic;
    terminal data_outp, data_outn : electrical);
end fc_jtol_src_ns;

-----
-- Architecture Declarations
-----

architecture bhv of sj_src is
  constant sj_tau : time := 10ps;
  quantity vsj_out across isj_out through sj_out;
  quantity vsj_int : voltage;
begin
  -- All quantities considered in a 50 Ohm environment
  -- Voltage source is 2 * effective voltage
  -- sj_amp is considered a PP voltage
  -- output signal is centered around Vdd/2=0.0V
  vsj_int == 2.0 * (0.0 + 0.9 * 0.5 * sj_amp * sin(MATH_2_PI*sj_freq*freq*NOW));
  vsj_out == vsj_int + isj_out * 50.0;
end bhv;

architecture bhv of dj_src is
  constant rt, ft : real := 20.0e-12;
  constant num : real_vector := (0 => MATH_2_PI * 50.0e6);
  constant den : real_vector := (0 => MATH_2_PI * 50.0e6,
                                  1 => 1.0);

  signal sck, sena : std_ulogic := '0';
  signal sdata : std_ulogic := '0';
  signal dj_prbs_real : real := 0.0;
  signal dj_clk, dj_prbs : std_logic := '0';
  quantity dj_out_quant : voltage := 0.0;
  quantity vdj_int : voltage := 1.8;
  quantity vdj_out across idj_out through dj_out;

```

```

begin
  -- purpose: Generates Clock for DJ PRBS Gen
  -- type : combinational
  -- inputs : dj_clk
  -- outputs: dj_clk
  gen_dj_clk: process (dj_clk)
  begin -- process gen_dj_clk
    dj_clk <= not dj_clk after 0.5 ns;
  end process gen_dj_clk;

  -- DJ PRBS Generator
  gen_dj_prbs : entity LIB_AMS.prbs7(dig)
    port map (
      clk           => dj_clk,
      sck           => sck,
      sena          => sena,
      rst_b         => nreset,
      sdata         => sdata,
      data_out      => dj_prbs);

  -- purpose: D2A Conversion for DJ stimulus
  -- type : combinational
  -- inputs : dj_prbs
  -- outputs: dj_prbs_real
  dj_prbs_d2a: process (dj_prbs)
  begin -- process dj_prbs_d2a
    if dj_prbs = '0' then
      -- division by 2 as udj_amp is a PP value
      -- multiplication factor to compensate for loss of
      -- amplitude due to incomplete settling
      -- PRBS7 at 1.0 Gb/s has 2.2tau when filtered with
      -- fc = 50MHz (was 1.0587, the 1.0877)
      dj_prbs_real <= 0.0 - 0.9 * udj_amp / 2.0 * 1.1667;
    else
      dj_prbs_real <= 0.0 + 0.9 * udj_amp / 2.0 * 1.1667;
    end if;
  end process dj_prbs_d2a;
  dj_out_quant == dj_prbs_real*ramp(rt, ft);
  -- All quantities considered in a 50 Ohm environment
  -- Voltage source is 2 * effective voltage
  vdj_int == 2.0 * dj_out_quant*ltf(num, den);
  vdj_out == vdj_int + idj_out * 50.0;
end bhv;

architecture bhv of rj_src is
  constant rj_tau : time := 10ps;
  signal rj_out_sig : real := 0.0;
  signal mean : real := 0.0;
  signal sigma : real := 0.0;
  quantity vrj_out across irj_out through rj_out;
  quantity vrj_int : voltage;
begin
  -- purpose: Gaussian White Noise Generator
  -- type : combinational

```

```

-- inputs :
-- outputs: rj_out
gen_rj: process
    variable seed1 : positive := 19824;
    variable seed2 : positive := 184;
    variable x : real := 0.0;
begin -- process gen_agwn
    sigma <= 2.0 * 1.8 * MATH_PI * rj_amp;-- Baumer, "XAUI Jit Specs"
    awgn(seed1, seed2, mean, sigma, x);
    rj_out_sig <= x;
    wait for rj_tau;
end process gen_rj;
-- All quantities considered in a 50 Ohm environment
-- Voltage source is 2 * effective voltage
vrj_int == 2.0 * rj_out_sig;
vrj_out == vrj_int + irj_out * 50.0;
end bhv;

architecture bhv of sin_clk_src is
    constant kvco : real := 1.5;
    constant PI_OVER_2 : real := MATH_2_PI / 4.0;-- test
    quantity vsin_clk across isin_clk through sin_clk;
    quantity vmod_in across imod_in through mod_in;
    quantity dph : real;
    quantity inst_ph : real := 1.0;
    quantity vsin_clk_int : voltage;
begin
    -- All quantities considered in a 50 Ohm environment
    -- Voltage source is 2 * effective voltage
    vmod_in == imod_in * 50.0;
    -- Voltage is centered around 0.0V and scaled to full - scale
    dph == vmod_in * kvco * MATH_2_PI / 0.9;
    vsin_clk_int == 2.0 * (0.0 + 0.9 * sin(MATH_2_PI*freq*NOW+dph));
    vsin_clk == vsin_clk_int + isin_clk * 50.0;
end bhv;

architecture str of clk_gen is
    terminal mod_in : electrical;
begin
    i_sj_src: entity LIB_AMS.sj_src(bhv)
        port map (
            sj_amp      => sj_amp,
            sj_freq     => sj_freq,
            freq        => freq,
            sj_out      => mod_in);

    i_dj_src: entity LIB_AMS.dj_src(bhv)
        port map (
            udj_amp     => udj_amp,
            nreset      => nreset,
            dj_out      => mod_in);

    i_sin_clk_src: entity LIB_AMS.sin_clk_src(bhv)
        port map (

```

```

        freq      => freq,
        mod_in    => mod_in,
        sin_clk   => sin_clk);
end str;

architecture dig of cjtpat_core is
    constant d30p3_pat : std_logic_vector(19 downto 0) :=
        "10000111000111100011";
    constant d21p5_pat : std_logic_vector(19 downto 0) :=
        "10101010101010101010";
    signal counter : natural := 100;
begin

    gen_proc : process (clk, rst_b)
    begin
        if rst_b = '0' then
            data_out <= '0';
        elsif clk'EVENT and clk='1' then
            if counter < 100 then
                data_out <= d30p3_pat(counter mod 20);
            else
                data_out <= d21p5_pat(counter mod 20);
            end if;
            if counter = 130 then
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process gen_proc;
end dig;

architecture bhv_str of cjtpat_gen is
    constant clk_thresh1 : real := -0.05;
    constant clk_thresh2 : real := 0.05;
    signal sq_clk : std_logic := '0';
    quantity vclk_in across iclk_in through clk_in;
begin
    vclk_in == iclk_in * 50.0;

    -- purpose: Detects zero crossings of the reference clock
    -- type : combinational
    -- inputs : sin_jit_clk
    -- outputs: sq_jit_clk
    edge_det_clk: process (vclk_in'above(clk_thresh1),
        vclk_in'above(clk_thresh2))
    begin -- process edge_det_clk
        if not vclk_in'above(clk_thresh1) then
            sq_clk <= '0';
        elsif vclk_in'above(clk_thresh2) then
            sq_clk <= '1';
        end if;
    end process edge_det_clk;
end architecture bhv_str;

```

```

-- CJTPAT Generator
gen_cjtpat : entity LIB_AMS.cjtpat_core(dig)
  port map (
    clk           => sq_clk,
    rst_b         => nreset,
    data_out      => cjtpat);
end bhv_str;

architecture bhv of fc_jtol_src_ns is
  constant rt, ft : real := 20.0e-12;
  terminal clk_in : electrical;
  signal cjtpat : std_logic := '0';
  signal cjtpat_real : real := 0.0;
  quantity cjtpat_quant : voltage;
  quantity vdata_out across idata_out through data_outp to data_outn;
begin

  i_clk_gen : entity LIB_AMS.clk_gen(str)
    port map (
      sj_freq      => sj_freq,
      sj_amp       => sj_amp,
      udj_amp      => udj_amp,
      freq         => freq,
      nreset       => nreset,
      sin_clk      => clk_in);

  i_rj_src : entity LIB_AMS.rj_src(bhv)
    port map (
      rj_amp       => rj_amp,
      rj_out       => clk_in);

  i_cjtpat_gen : entity LIB_AMS.cjtpat_gen(bhv_str)
    port map (
      nreset       => nreset,
      clk_in       => clk_in,
      cjtpat       => cjtpat);

  cjtpat_d2a: process (cjtpat)
  begin
    if cjtpat = '0' then
      cjtpat_real <= data_vcm - data_amp / 2.0;
    else
      cjtpat_real <= data_vcm + data_amp / 2.0;
    end if;
  end process cjtpat_d2a;
  cjtpat_quant == cjtpat_real*ramp(rt, ft);
  vdata_out'dot / (MATH_2_PI * ddj_fc) + vdata_out == cjtpat_quant;
end bhv;

-----
-- Block Name: General Functions Package
-- File Name:general.vhd
-- Author:Paul Muller
-- Date:16.07.04

```

```
-- Rev: 1.0
-----
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.math_real.all;

package general is

    procedure awgn (
        variable seed1, seed2 : inout positive;
        signal mean, sigma : in real;
        variable x : out real);

end general;

package body general is
procedure awgn (
    variable seed1, seed2 : inout positive;
    signal mean, sigma : in real;
    variable x : out real
) is
    variable r1, r2 : real:= 0.0;
begin
    UNIFORM(seed1, seed2, r1);
    UNIFORM(seed1, seed2, r2);
    x := mean + sigma * (sqrt(-2.0 * log( r1 )) * cos( 2.0 * MATH_PI * r2 ));
end procedure awgn;
end general;
```

---

## References

- [1] N. Ou *et al.*, "Jitter models for the design and test of Gbps-speed serial interconnects", *IEEE Design & Test of Computers*, vol.21, issue 4, pp. 302 - 313, July-Aug. 2004
- [2] INCITS Technical Committee T11, "Fibre Channel - Methodologies for Jitter and Signal Quality Specification - MJSQ", Draft Technical Report, rev.14, June 2004
- [3] P. Muller, A. Tajalli, M. Atarodi, Y. Leblebici, "Top-Down Design of a Low-Power Multi-Channel 2.5-Gbit/s/Channel Gated Oscillator Clock-Recovery Circuit", in *Design, Automation and Test in Europe (DATE)*, vol. 1, pp. 258 - 263, 2005.
- [4] A. Gupta, B. Tailor and J. Francis, "The Deficiencies in Measuring Input Jitter Tolerance (IJT) using Sine wave modulated Jitter", IEEE 802.3 Higher Speed Study Group, March 2001
- [5] H. Baumer, J. van Engelen, "XAUI TX/RX Jitter Specifications", *IEEE P802.3ae 10Gb/s Ethernet Task Force*, July 2001  
<http://www.ieee802.org/3/ae/public/documents/XAUIjittercomments.pdf>
- [6] E. Lida and B. Shahar, "8b/10b ISI Killer Pattern Presentation & Comparison With CJTPAT", *IEEE P802.3ae 10Gb/s Ethernet Task Force*, January 2001,  
<http://www.ieee802.org/3/ae/public/documents/testpat.pdf>

- [7] D. Cecchi, "Analysis, Characterization, Design Tools and Techniques", *InfiniBand Trade Association*, High Speed Signaling WG, June 2001