Jitter Tolerance Estimation of a 3X Oversampling CDR using Event-Driven Simulation

Nathan Kiddinapillai
Department of Electronics
Carleton University
Ottawa, Canada
kkiddi@doe.carleton.ca

Tad Kwasniewski
Department of Electronics
Carleton University
Ottawa, Canada
tak@doe.carleton.ca

Abstract—This paper presents a system level jitter tolerance estimation of a 3X oversampling Clock and Data Recovery (CDR) circuit used in high speed serial data communication receivers. It is critical to know the amount of jitter that can be tolerated by the CDR in order to recover the data with satisfied bit error ratio (BER) performance. The jitter tolerance of the CDR is estimated by an event-driven simulation model, developed in Matlab. The theoretical jitter tolerance value is derived analytically for different jitter frequencies. The simulated results show a very close match to the theoretical values. We also compared the simulation time for the event-driven and the conventional fixed-time-step models.

Keywords—Event-Driven; CDR; oversampling; jitter tolerance; clock and data recovery;

I. INTRODUCTION

Clock and data recovery is an important function in high frequency serial data communication receivers, such as optical communication systems, backplane routing and chip-to-chip interconnects [6]. The data received at the receiver contains jitter therefore the sampling points may not be aligned with the center of the data eye thus causing bit errors in the recovered data. A CDR circuit extracts the clock timing information, which defines a sampling point close to the center of the data eye and retimes the data such that the associated jitter is removed. It is important to know the amount of jitter (sinusoidal) that can be tolerated by the CDR without on set of errors or exceeding the specified BER [3]. This is defined as the jitter tolerance. The jitter tolerance is a function of frequency and is usually specified in unit interval (UI). One UI corresponds to one baud period.

Figure 1 shows the jitter tolerance variation of a CDR against jitter frequency (dotted line) where \( A \) is the high frequency jitter tolerance and \( f_c \) is the jitter corner frequency. The solid line in Figure 1 shows the jitter tolerance mask for the Synchronous Optical Network / Optical Carrier-48 (SONET/OC-48) standard. The jitter tolerance of a CDR has to be in the acceptable region, which is the region above the jitter tolerance mask. The jitter tolerance is an important CDR performance parameter that could be combined with system’s random jitter to estimate its predicted BER performance through “bathtub curve”.

In this paper, first the operating principle of the proposed CDR is described using system level block diagrams. The sinusoidal jitter tolerance performance of the CDR is analyzed analytically. In order to derive the jitter tolerance variation of the CDR, similar to that shown in Figure 1, an event-driven simulation model was developed in Matlab.

II. PROPOSED CDR

A CDR with an oversampling ratio of three (3X) that uses a threshold decision technique to achieve high jitter tolerance performance is proposed. The system level block diagram of the CDR is shown in Figure 2. The CDR consists of a phase detector (PD), a phase rotating signal generator (PRSG), a phase rotator (PR) and a multiplexer (MUX).

A. CDR Operating Principle

Incoming data is sampled by three clock phases (\( Clk1 \), \( Clk2 \) and \( Clk3 \)), separated by \( T/3 \), where \( T \) is the nominal symbol period of the data. One of three clock phases is dynamically selected as the data sampling clock phase (DSCP) and is used to retime the data. The DSCP is continuously adjusted (rotated) by the CDR so that the DSCP is kept close to the center of the data eye to reduce the BER.

B. Phase Detector

At the phase detector the data samples are first used to detect the data transition point. The data transition points are compared with the DSCP point to produce a phase error. The phase error is defined as the phase difference between the rising edge of the middle phase, which is the second phase after
the data transition, and the rising edge of the DSCP. In the case of zero jitter, an ideal case, middle phase will be aligned with the DSCP, thus producing a zero phase error. The phase error is assumed to be zero in the absence of data transitions.

According to the proposed technique, the phase detector generates a request signal L (or R) if the data transition point is lagging (or leading) with respect to its ideal position, by more than $T/3$. The L and R signals indicate that the current DSCP will be rotated left (advanced in time), or right (retarded in time), respectively. If the magnitude of the phase error is less than $T/3$, no request signal will be produced, thus the current DSCP will be unchanged. The $T/3$ is considered to be the decision threshold of the CDR.

C. Phase Rotating Signal Generator (PRSING)

The CDR will not rotate the DSCP point in each symbol period even though the request signal L or R is produced. According to the proposed decision technique, the DSCP point will be rotated once in each timing window of eight symbol periods. Updating the DSCP point once in eight symbol periods leads to high frequency jitter averaging, which is the most common way to improve the jitter tolerance at high jitter frequencies. A timing window of eight symbol periods is selected to ensure that at least two data transition points are captured within the timing window for $2^7 - 1$ PRBS (PRBS 7) data.

The PRSG collects all the L and R request signals within each timing window and stores them. A command signal (Rot. sig) is generated, if the PRSG meets one of the following conditions within each timing window,

- found at least one L signal but no R signal
- found at least one R signal but no L signal

If both L and R signals are present within any timing window, no command signal will be produced.

D. Phase Rotator (PR)

The phase rotator receives the request signal L (or R), and the Rot. sig, and rotates the DSCP point left (or right). For example, the current DSCP point is Clk2, and it receives both Rot. sig and R, the new DSCP point will be Clk3, (see Fig. 2). If the phase rotator receives either the L or R signal but the command signal is not asserted the current DSCP will be unchanged.

E. Multiplexer (MUX)

The MUX receives the DSCP point for each symbol period and selects one of three data samples as the recovered data.

F. Mathematical analysis of jitter tolerance of proposed CDR

A sinusoidal jitter at a jitter frequency of $f_j$ with a peak jitter amplitude of $A_j$ UI can be expressed as

$$\phi_j = 2\pi A_j \sin(2\pi f_j t) \text{ rads} \quad (1)$$

By differentiating (1) with respect to time, the phase changing speed (PCS) can be obtained as,

$$PCS = A_j (2\pi)^2 f_j \cos(2\pi f_j t) \text{ rads/sec} \quad (2)$$

From (2) the maximum phase changing speed (MPCS) is evaluated in terms of normalized jitter frequency ($F_j$), defined as the ratio between the jitter frequency and the baud rate.

$$MPCS = 2\pi A_j F_j \text{ UI/UI} \quad (3)$$

The guaranteed number of data transitions is one per seven symbol periods for PRBS 7. Since the phase is updated by 1/3 UI once in eight symbol periods for the 3X oversampling CDR, the phase change speed is 1/3 UI in thirteen symbol periods. This phase change speed should be larger than the MPCS determined in (3) in order to avoid bit errors. The maximum peak-to-peak low frequency jitter amplitude ($J_{TOL-LOW}$) that can be tolerated is obtained as

$$J_{TOL-LOW} = \frac{1}{39 \pi F_j} \text{ UI} \quad (4)$$

At high frequencies the jitter period is smaller than that at low jitter frequencies so the transition may not occur within one jitter period. Between two transitions, which is larger than the jitter period, average phase change is equal to $A_j$. This average phase change must be less than the phase change limit of 1/3 UI for 3X oversampling CDR. Therefore the jitter tolerance at high jitter frequency ($J_{TOL}$) is obtained as

$$J_{TOL} = \frac{2}{3} \text{ UI} \quad (5)$$

On a log scale, the jitter tolerance at a low frequency (see (4)) has a linear variation with the slope of -1(-20 dB/decade).

From (4) and (5), the jitter corner frequency ($F_c$), the intersection between the jitter tolerance curve at low jitter frequency and high jitter frequency, is obtained.

$$F_c = \frac{1}{26\pi} \quad (6)$$
III. JITTER TOLERANCE ESTIMATION OF THE CDR

A. Simulation setup and procedure

Figure 3 shows the simulation setup to find the bit error produced by the CDR. First, a sinusoidal jitter is generated for a given jitter frequency and amplitude which is then combined with the PRBS data to produce the jittered data. The CDR receives the jittered data and three clock signals, and outputs the recovered data. The recovered data is compared with the original data at a bit error tester to produce the bit errors.

B. Estimation of Jitter Tolerance

Since the jitter tolerance estimation is performed for the sinusoidal jitter (not random jitter) simulation runs until the occurrence of the first bit error. To find the jitter tolerance at a given jitter frequency \((f_j)\), a jitter amplitude is set at an initial value \((A_i)\), and the bit error tester output is observed. If any bit errors are detected, the jitter amplitude is reduced by a small step, \(\Delta A\), and the simulation runs again. The amplitude at which errors stop to appear would be the jitter tolerance at that jitter frequency. Alternatively, if the initial amplitude produces no bit error, the jitter amplitude is stepwise increased by \(\Delta A\) until the CDR produces a bit error. The corresponding amplitude would be the jitter tolerance at that jitter frequency.

In order to obtain the jitter tolerance over a frequency range, the frequency is changed by small frequency steps, \(\Delta f\), to define a set of discrete frequencies. The above procedure is repeated for each discrete frequency value to get the corresponding jitter tolerance.

C. Conventional simulation

Conventional simulation tools use a time sweep with a certain time step to calculate the signal level at each time point [1], which is called as the fixed-time-step simulation [5]. A simulation requires a sufficiently small time step to ensure that good timing resolution is achieved. A small time step results in a large number of signal points that need to be evaluated. Therefore the simulation time will be dramatically increased for a simulation such as the one used to find the jitter tolerance [2].

In the jitter tolerance simulation, the accuracy of the jitter tolerance value depends on the amplitude and frequency step size used. The smaller the step size the better the accuracy. However, a smaller step size will require more iterations, thus increase the simulation time.

IV. CONCEPT OF EVENT-DRIVEN SIMULATION

In an event-driven simulation, only the time instants of interest are used. In an oversampling CDR, even though the real data and clock signals have values at all instants of time, the transition time points are considered to be the time points of interest. By using only the time point of interest, the simulations are no longer driven by a regular increment of time. They are driven by the events at the transition points. Event-driven simulations require much less time than conventional fixed-time-step simulations to perform the same task [4] [5].

In an event-driven simulation model, each event is defined by three fields: event time, event index, and event parameter [1]. Event time is the exact time instant at which the event is to be executed. Event index indicates the functional block in which the event is generated and executed. For example, the event index “PD_index” indicates that events are from the phase detector (PD). An event parameter is used to control different event tasks by assigning different event parameter values.

An event driven simulation model has three parts, as shown in Figure 4 [1]: the event generator, the event dispatcher and the event handler.

A. Event generator

All the events are generated by one of the functional blocks represented by the event handler except for the events that are generated at the start of simulation by initializing each functional block at time zero with parameter of zero. The Matlab code for initializing the functional blocks is shown in List 1, based on [1]. The Matlab function ‘Addevent’ is called to generate the event at time zero by passing the parameter of zero.

List 1. Matlab code to initialize the functional blocks
event(‘initialize’) %reset passing parameter of zero at time zero
event(‘Addevent’, @time=0, PRBS_generator, parameter = 0)
event(‘Addevent’, @time=0, Phase_Detector, parameter =0)
event(‘Addevent’, @time=0, Phase_Rotator, parameter = 0)
event(‘Addevent’, @time=0, Data_MUX, parameter =0)
B. Event Dispatcher

The event dispatcher receives all of the event requests in the order in which they are generated according to the functional operation of the CDR. Within the event dispatcher an event management function sorts the events in an order by which events are to be executed and stores them in a queue.

C. Event Handler

The event handler executes the events in the order in which they appear in the queue. The earliest event will be executed first. Once an event is executed by the event handler that event will be deleted from the queue. In addition to executing events, the event handler is also responsible for generating the subsequent event according to the CDR operation.

Each functional block is modeled by writing a Matlab function. A Matlab code for a functional block is shown in List 2 [1]. Each function is defined by two arguments: time and parameter. Different cases are used to perform different event tasks, for example, at ‘case 0’ events are initialized. Different parameter values (cases) are used to control different type of event tasks.


```
function Block_Name(time, parameter)
persistent var1, var2
switch parameter
  case 0  %initialize the block
    <initialization>
  case 1
    <processing of event type 1>
  ...
  Case N
    <processing of event type N>
end
```

V. SIMULATION RESULTS

Table 1 compares the simulation time for conventional fixed-time-step and event-driven Matlab models. The fixed-time-step model uses a time step value of 1/100 of baud period. In both cases the simulation is run for 1000 baud period intervals.

<table>
<thead>
<tr>
<th></th>
<th>Conventional fixed-time-step</th>
<th>Event-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation time</td>
<td>120 s</td>
<td>4 s</td>
</tr>
</tbody>
</table>

The event-driven simulation is executed for 20,000 data symbols, and thus total of 60,000 samples are processed. The variation of jitter tolerance against jitter frequency is obtained. The simulated result and theoretical analysis (solid line) are presented in Figure 5. The simulation results show a very close match to the theoretical values. The simulated high frequency jitter tolerance of 0.65 UI is very close to the theoretical value of 0.66 UI from (5). The normalized jitter corner frequency obtained from simulation (0.014) is close to the calculated value of 0.012 from (6). For a longer PRBS sequence (here PRBS 7) the error might be larger.

VI. CONCLUSION

In this paper, the jitter tolerance variation of the proposed 3X oversampling CDR is estimated using event-driven simulation model, developed in Matlab. The jitter tolerance of the CDR is mathematically analyzed and the theoretical values, equations and results are presented for different jitter frequencies. The simulation results show a close match with the theoretical values. It was also found that event-driven simulation is 30 times faster than conventional simulation with fixed-time-step of 1/100 of baud period. The event-driven simulation is becoming more popular particularly in modeling of mixed signal CMOS designs.

REFERENCES