Clock Recovery in Serial-Data Systems

Ransom Stephens, Ph.D.

Abstract:

The definition of a bit period, or unit interval, is much more complicated than it looks. If it were just the reciprocal of the data rate we'd be in worse trouble with jitter than we already are. In this installment of Jitter 360, we investigate the true identity of the unit interval and how serial-data systems use a recovered clock instead of an independent reference clock. The investigation will reveal the key features of clock recovery that affect the bit error ratio, namely bandwidth and peaking – including which part of the jitter-spectrum matters most.

It's convenient to think of a receiver, as shown in Figure 1, as a device that uses a clock to position the sampling point in time so that a comparator can decide whether the signal voltage at that time is greater or less than a decision threshold value. If it's greater, the receiver identifies a logic 1, and if it's less, a logic 0. Setting the voltage decision threshold is simple – the threshold is almost always zero for differential systems – but the time-position of the sampling point is much trickier. This is where clock recovery comes in.

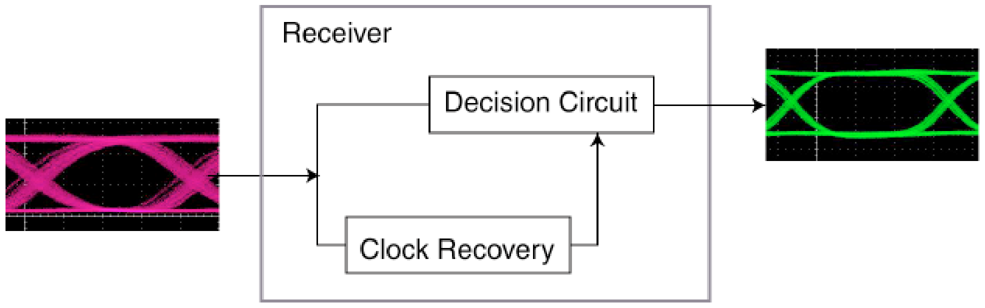


Figure 1: Simple diagram of a serial-data receiver.

Consider a simple system with an absolute external reference clock, Figure 2a. If we tune the relative phase of the clock so that it's in phase with the incoming data transitions and then have the receiver sample on falling clock edges, the sampling point should be the center of every bit, Figure 2b. The unit interval in this system is exactly the reciprocal of the nominal data rate. Usually when we speak of the timing of bits, this is what we have in mind. It's easy to think about, but it has some problems.

The first problem is that providing an absolute external clock to both the transmitter and receiver requires an extra data line and an expensive clock low jitter clock. Oops, did I say expensive?

The biggest problem, though, is that having the perfect external clock *increases* the Bit Error Ratio (BER)!

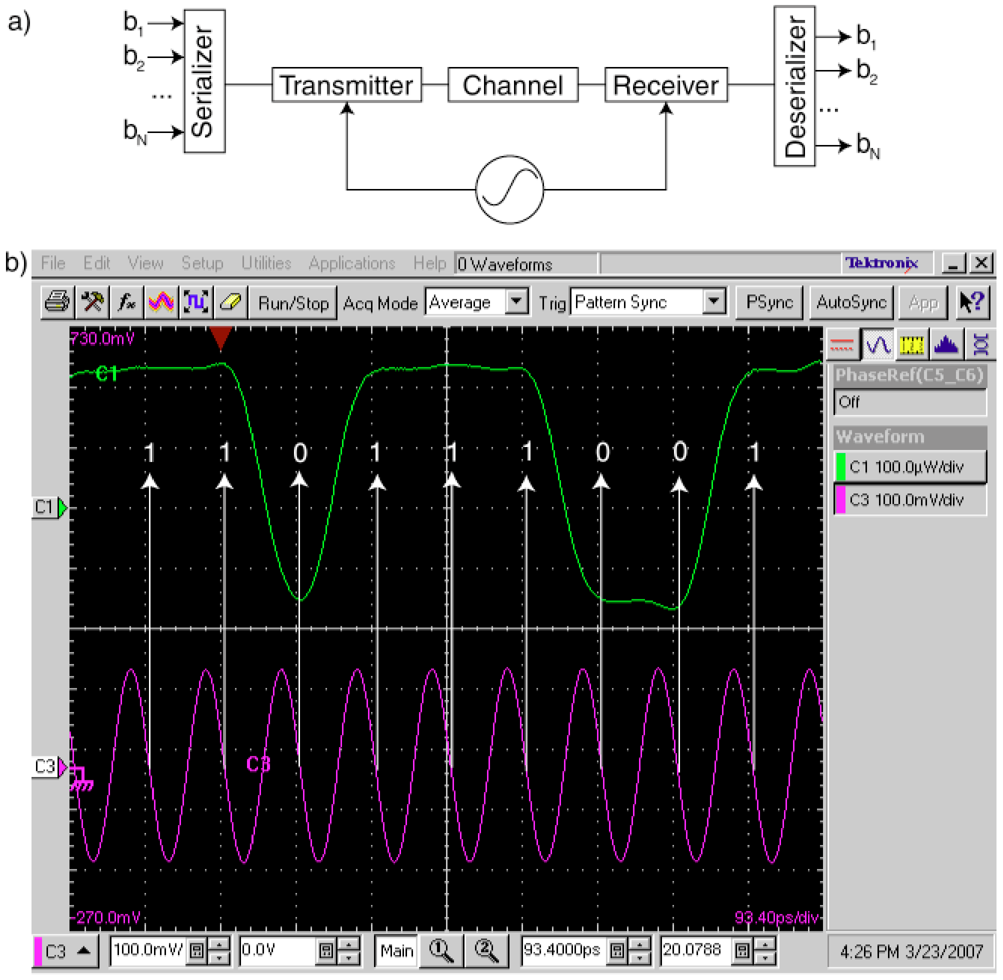


Figure 2: (a) System with an external clock and (b) setting the time-position of the sampling point.

What would happen if we could set the sampling point half a bit after the time at which logic transitions *actually* occur instead of half a bit after the time at which they *should* occur? Jitter would never cause an error!

In this ideal world, we could trigger on a logic transition and sample half a bit period later. The sampling point would have exactly the same jitter as the data and the signal would never fluctuate across the sampling point. The only price we would pay is a more complicated definition of a unit interval.

We can get close to such a utopian world by recovering the clock from the data itself. An infinite bandwidth clock recovery system would trigger the clock signal on a data transition and the sampling point timing would have the same jitter as the data. If the data and clock have the same jitter, then they dance in harmony and bits are identified not at ideal times but at the best times – the jitter on the clock *tracks* the jitter on the data and the BER isn't affected by jitter.

In real life, with a finite bandwidth clock recovery circuit, the low frequency jitter tracks the data. Only jitter at frequencies above the clock recovery bandwidth causes errors. It gets better, not only does clock reconstruction reduce the BER, but it allows the use of clocks that have lots of jitter on them (i.e., cheaper clocks) and it doesn't require a trace or cable to get a clock signal from the transmitter to the receiver (i.e., cheaper design).

Clock Recovery

There are two basic types of clock recovery, those that are more analog in nature, like Phase-Locked Loops (PLL), and those that are more digital in nature. Digital, in this context means that the clock is reconstructed from multiple discrete samples rather than the continuous analog data signal. The Phase Interpolator (PI) is a well known example, though there are many proprietary techniques. The biggest practical difference between PLLs and PIs is cost and the biggest theoretical difference is ease of parameterization and modeling. PIs usually have a faster lock time than PLLs, use less power, and require less surface area resulting in less expensive designs. Like any circuit element, a well designed PI can do its job just fine, certainly as well as a well designed PLL. The problem with PIs is that their nonlinear behavior is difficult to anticipate making it much harder to debug a troubled PI-based clock recovery system than one that is PLL-based.

Since transfer functions for whole families of PLLs can be written down in terms of a few simple parameters, where PIs can have different parameterizations for nearly every design, technology standards committees specify clock recovery in terms of PLLs. One approach is to use a simple second-order PLL with transfer function, $H(s)$

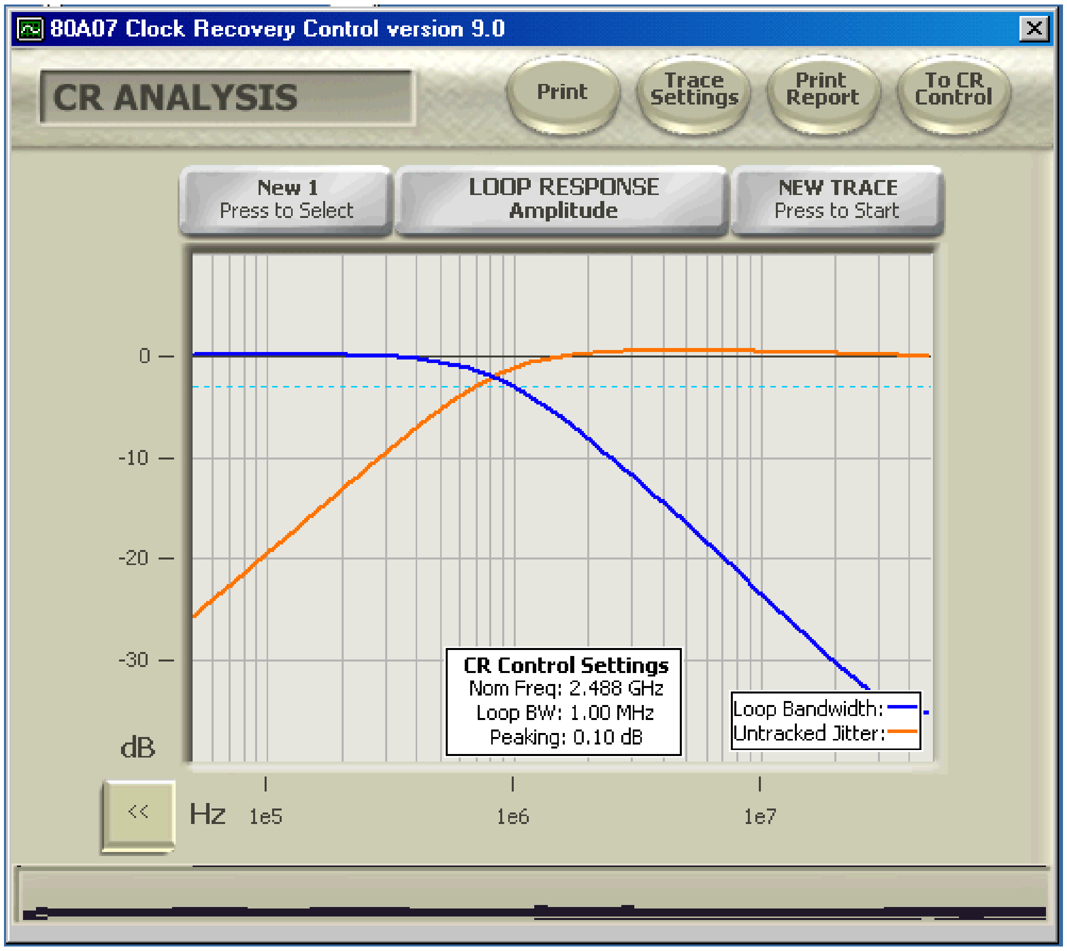
$$H(s) = \frac{2s\zeta\omega_n + \omega_n^2}{s^2 + 2s\zeta\omega_n + \omega_n^2} \, . \tag{1}$$

There are two key parameters associated with this PLL, the peaking, given by ζ, and the natural frequency ω_n. The bandwidth is given by

$$\omega_{3\mathrm{dB}} = \omega_n \sqrt{1 + 2\zeta^2 + \sqrt{(1+2\zeta^2)^2 + 1}} \, . \tag{2}$$

The two parameters determine what range of jitter frequencies common to both the clock and data signals. Jitter at frequencies *below* the bandwidth cutoff doesn't cause errors unless the peaking, ζ, is too high. PLL peaking causes jitter frequencies near the cutoff to be *amplified* and show up in the recovered clock at greater amplitudes than in the data signal.

First order PLLs are a simple alternative which don't have any peaking. From the perspective of a standards body there is a tradeoff in which way to model clock recovery. The ideal would be to use an arbitrarily complex model so that every parameter could be specified – but no one would understand the specification. The second order PLL modeled by Eq. (1) provides a compromise allowing both the bandwidth and peaking to be constrained by the specification.



CR Control Settings
Nom Freq: 2.488 GHz
Loop BW: 1.00 MHz
Peaking: 0.10 dB

Figure 3: The frequency response of a 2nd order PLL.

Figure 3 shows the frequency response of a CDR based on a second order PLL. By reconstructing the clock from the data, jitter in the frequency band below the cutoff is common to both. When the clock signal and data signal jitter together in harmony, that jitter doesn't cause errors. The wider the CDR bandwidth, the smaller is the BER-relevant jitter-frequency band. Given the transfer function for a CDR, it's straightforward to implement in on a real time oscilloscope – a nice way to separate the relevant jitter and analyze the signal as it would be seen by a receiver's decision circuit.

Figure 4 shows two eye diagrams. The data signal in each is identical, the only difference is the bandwidth of the recovered clock used to trigger the oscilloscope. A very narrow bandwidth clock recovery shows a great deal of jitter. As the bandwidth of the recovered clock increases, the clock tracks jitter on the signal and, in Figure 4b there is almost no effective jitter.

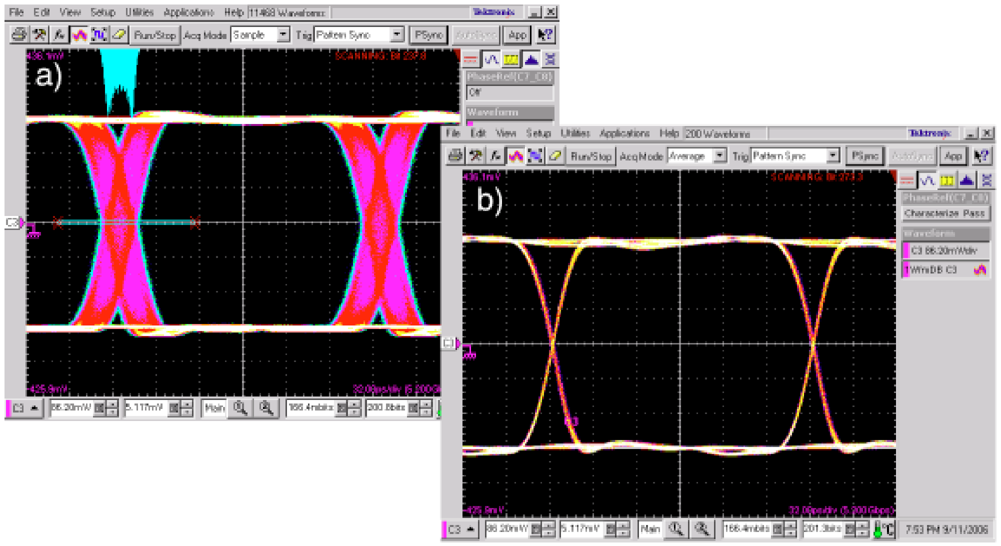


Figure 4: Eye diagrams of the same signal with (a) very small clock recovery bandwidths and (b) very wide clock recovery bandwidth.

Figure 5 is a block diagram of a PLL-based Clock Data Recovery (CDR) circuit. The data signal is split into two paths. One path guides the logic signal to the decision circuit (DC), and the other path goes to a PLL clock recovery circuit. The PLL is composed of a phase-frequency detector, a low-pass filter, and a voltage-controlled oscillator (VCO). The VCO locks to a phase and frequency in such a way as to minimize the difference in its phase and the phase of data-transitions. The recovered clock signal is then used as the time reference in the decision circuit.

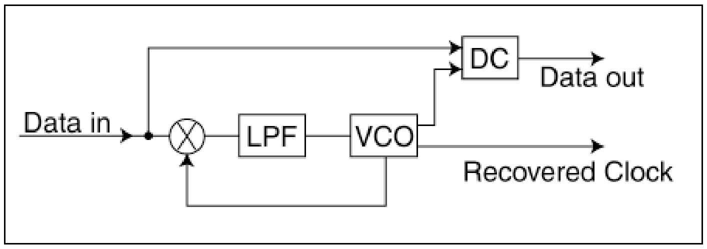


Figure 5: Phase-locked loop based Clock Data Recovery (CDR) circuit.

There are a couple of fairly obvious requirements of data signals in these embedded clock systems. In order to extract a clock signal of the appropriate frequency, the data signal cannot have extremely long runs of consecutive identical bits. Rather, the data must have plenty of transitions to drive the phase detector and tune the VCO. Second, the overall signal must be DC-balanced. If there aren't enough transitions, the VCO may not be able to lock on the data signal and, even if it locks, it will drift. To ensure enough transitions, data signals must be encoded. For example, "8B/10B encoding" is a common technique that encodes each 8 bits of data into a 10 bit symbol. The low 5 bits of data are encoded into 6 bits and the following 3 bits are encoded into 4 bits so that the transmitted signal is DC-balanced (same number of zeros as ones) and has no more than 4 consecutive identical bits.

Figure 6 is a block diagram of a Phase Interpolator (PI). PIs sample a data rate reference clock and compare it to the incoming data to fix the relative phase of the sampling point. Figure 6 is a simple configuration. A distributed 100 MHz reference clock is multiplied up to the data rate by a simple PLL. The data rate clock is sampled to determine that pair of reference phases that surround logic transitions in the data signal. The sampling point of the decision circuit is then determined by interpolating between those two reference phases. Since PIs require a data-rate reference clock, most designs include a distributed system reference clock.

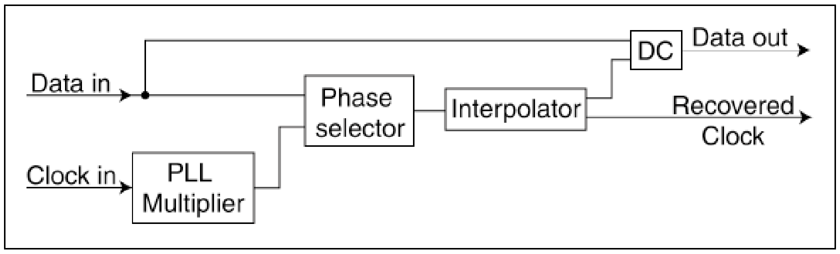


Figure 6: Phase Interpolator block diagram.

Delay

To reap the greatest rewards from the use of clock recovery, it is important that the relative delay of the data signal and the recovered clock be small. Remember, if the clock and data both have the same jitter at the decision circuit, then that jitter doesn't cause errors. But if the jitter on the clock differs from the jitter on the data, then we just end up with more jitter and a larger BER. If there is a delay between the reconstructed clock and the data, then the transitions don't share the same jitter. They're like two dancers listening to different music. If there's appreciable delay, rather than have the clock and data signals dance in harmony, the clock steps on the data's feet.

Actually, it's not quite as bad as I've made it out. Figure 3 shows that the low-frequency end of the jitter spectrum is rendered irrelevant to the BER in a CDR-based system. A delay of a few bit periods between the data and the clock may not have much effect on the BER since the tracked jitter is at a frequency much lower than the data rate. Still, we must be aware that delay can cause problems… especially in systems that use spread-spectrum clocks.

Spread Spectrum Clocking

Spread Spectrum Clocking (SSC) is a technique used to smear radiated energy into larger frequency bands through low frequency clock modulation. Smearing the energy reduces the peak energy at any given frequency, making it easier to comply with regulations that limit emitted power in narrow frequency bands. Typically, the clock is modulated at 33 kHz from zero to -0.5% of the nominal data rate in either a triangle wave or a “Hershey’s Kiss” waveform.

At the receiver, the CDR bandwidth should generously cover the modulation bandwidth and the delay must be small enough that any SSC deviations that leak through are tracked. The problem introduced by delay is exacerbated at the transition from positive to negative going triangle FM. The peculiarities of SSC, how it’s implemented, how it’s affected by peaking and how it is best analyzed to reduce BER is a subject worthy of a paper all its own. In the next paper of this series, *Part 6: Reference Clock Jitter and Data Jitter*. we’ll discuss more aspects of SSC.

Conclusion

There are many reasons for using clock recovery in serial-data systems and reducing the BER caused by jitter is a very good one.

In most existing specifications, and nearly every foreseeable specification, for technology over about 3 Gb/s, clock recovery elements are required to have a bandwidth above a specified minimum – to encourage data-jitter/clock-jitter tracking – and peaking below a specified maximum – to limit jitter amplification. For example, FibreChannel 4X, at 4.25 Gb/s requires a loop bandwidth of 2.55 MHz with maximum peaking of 0.3 dB. PCI Express Generation II (at 5 Gb/s) requires a loop bandwidth between 5 and 16 MHz with peaking no larger than 1 dB or loop bandwidth between 8 and 16 MHz with maximum peaking of 3 dB. In the latter case, the wider loop bandwidth affords greater peaking; separate specifications are provided for greater design flexibility.

In the part six of this series, we’ll return to the topic of clock recovery as we discuss the role of the reference clock itself. Stay tuned…