Jitter Transfer Characteristics of Delay-Locked Loops—Theories and Design Techniques

M.-J. Edward Lee, *Member, IEEE*, William J. Dally, *Fellow, IEEE*, Trey Greer, Hiok-Tiaq Ng, *Member, IEEE*, Ramin Farjad-Rad, *Member, IEEE*, John Poulton, *Senior Member, IEEE*, and Ramesh Senthinathan

***Abstract*—This paper presents analyses and experimental results on the jitter transfer of delay-locked loops (DLLs). Through a z-domain model, we show that in a widely used DLL configuration, jitter peaking always exists and high-frequency jitter does not get attenuated as previous analyses suggest. This is true even in a first-order DLL and an overdamped second-order DLL. The amount of jitter peaking is shown to trade off with the tracking bandwidth and, therefore, the acquisition time. Techniques to reduce jitter amplification by loop filtering and phase filtering are discussed. Measurements from a prototype chip incorporating the discussed techniques confirm the prediction of the analytical model. In environments where the reference clock is noisy or where multiple timing circuits are cascaded, this jitter amplification effect should be carefully evaluated.**

***Index Terms*—Delay-locked loop (DLL), injection locking, jitter peaking, jitter transfer, multiplying delay-locked loop (MDLL), phase-locked loop (PLL).**

I. Introduction

DELAY-LOCKED loops (DLLs) have been widely used as frequency synthesizers and clock deskewing circuits in radio-frequency (RF) transceivers [1], [15], interchip communication interfaces [2], [3], and clock distribution networks [4], [5]. Although these functions can also be performed with phase-locked loops (PLLs), DLLs are often preferred due to their ease of design, better immunity to on-chip noise, and stability. In particular, a phenomenon known as *jitter accumulation* makes PLLs more susceptible to power-supply and substrate noise [2], [6], [14]. In cases where a significant amount of noise-generating digital circuitry is present on the same chip, DLLs are preferred because any jitter created by the on-chip noise is completely corrected when a clean reference clock edge arrives at the input of the DLL [1], [3], [15]. In some cases, however, the reference clock itself might have significant jitter, and the utilization of a DLL does not always guarantee superior jitter performance compared to a PLL.

Jitter peaking refers to the amplification of jitter from an input to an output over a certain frequency band and is an important performance metric in systems where multiple PLLs or DLLs are cascaded (such as in repeaters and clock distribution networks). It has been widely known that the traditional PLL inherently has jitter peaking that cannot be eliminated [7]. Techniques to minimize it by overdamping the loop or to remove it altogether through an architectural modification have been discussed in the literature [7]. This paper shows that in a widely used configuration, a DLL also has jitter peaking that cannot be eliminated, contrary to previous analyses [11], [12]. It is shown that this jitter peaking trades off with the tracking bandwidth and, therefore, the acquisition time of the DLL. Furthermore, unlike a PLL, high-frequency jitter in the reference clock does not get attenuated in a DLL. We introduce two techniques—loop filtering and phase filtering—to attenuate the high-frequency jitter transfer of a DLL. When a DLL is compared with other clocking architectures (e.g., a PLL) for jitter performance, the effect of jitter amplification needs to be evaluated carefully because any advantage a DLL has in correcting the self-noise might be completely offset by the amplification of a noisy reference clock.

Section II provides a general background on different types of DLLs. Section III gives a z-domain model of the Type I DLL to show that jitter amplification exists, and a physical explanation is given to show why it exists. Section IV discusses several techniques to reduce high-frequency jitter transfer. Section V presents the experimental results, and Section VI concludes with a summary.

II. Background

Previous DLL designs can be divided into two categories, which we call Type I and Type II, according to their jitter transfer characteristics. Type I and Type II DLLs are shown in Fig. 1(a) and (b), respectively. This paper is focused on the Type I DLL, which, unlike the Type II DLL, always exhibits jitter peaking. Generally speaking, a DLL is a servomechanism in which a delay path is adjusted in order to produce a desired phase relationship between two signals. The distinction here is whether one signal is derived from the other. In a Type I DLL, the reference is compared with the delayed version of itself. This architecture is widely used in DLL-based frequency synthesizers [1], [3], [4], multiphase clock generators [8], [9], and clock deskewing circuits [2], [5]. In a Type II DLL, the reference is compared with the delayed version of an uncorrelated signal. This architecture is widely used in DLL-based clock recovery circuits [10]. In previous publications, the same analysis is used for both types of DLLs, although it is only valid for Type II DLLs. We briefly review this analysis here [11], [12].

Manuscript received May 28, 2002; revised November 21, 2002.

M.-J. E. Lee, T. Greer, H.-T. Ng, R. Farjad-Rad, J. Poulton, and R. Senthinathan are with Velio Communications, Inc., Milpitas, CA 95035 USA (e-mail: ed@velio.com).

W. J. Dally is with Velio Communications, Inc., Milpitas, CA 95035 USA, and also with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA.

Digital Object Identifier 10.1109/JSSC.2003.809519

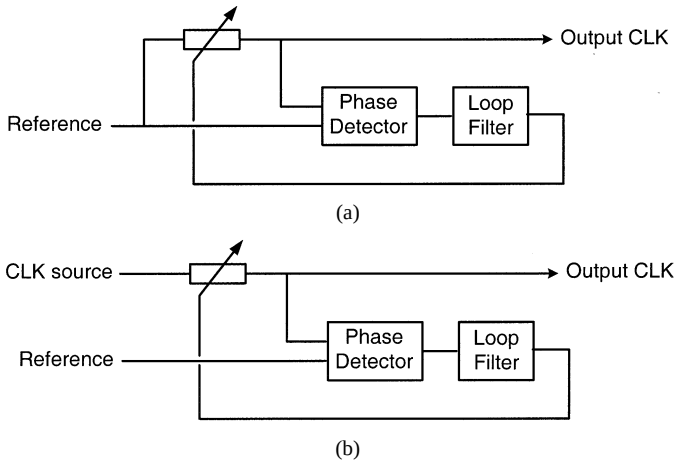


Fig. 1. DLL architecture. (a) Type I DLL. (b) Type II DLL.

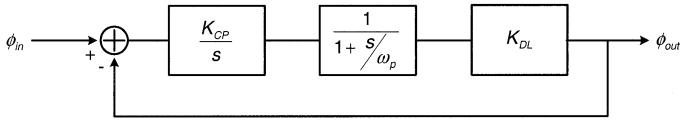


Fig. 2. S-domain model of DLL.

Fig. 2 shows the S-domain representation of a charge-pump DLL. Although in most cases the loop filter consists of only a capacitor (an integrator), in certain situations an extra pole (denoted by ω_p here) is introduced [3], [8], [9]. $K_{\rm DL}$ represents the delay line gain in radians per volt. $K_{\rm CP}$ represents the charge pump and loop filter gain. In terms of the charge pump current $I_{\rm CP}$ and the loop filter capacitance C_f, $K_{\rm CP}$ is equal to $I_{\rm CP}/2\pi C_f$. The jitter transfer is given by

$$\frac{\phi_{\rm out}}{\phi_{\rm in}} = \frac{1}{\frac{s^2}{\omega_p K_{\rm DL} K_{\rm CP}} + \frac{s}{K_{\rm DL} K_{\rm CP}} + 1}. \tag{1}$$

In cases where the loop filter consists of only a capacitor ($\omega_p \to \infty$), the jitter transfer contains a single pole ($K_{\rm DL}K_{\rm CP}$) and exhibits no jitter peaking. For the more general case, the jitter transfer is a second-order system and there is a possibility of jitter peaking. By overdamping the loop, however, jitter peaking is eliminated. This analysis also shows that high-frequency jitter is filtered out. In the next section, we show that these conclusions do not apply to Type I DLLs.

III. Jitter Amplification in Type I Delay-Locked Loops

The problem with the above analysis is that it does not take into account the fact that in Type I DLLs $\phi_{\rm out}$ is derived from $\phi_{\rm in}$. We use a z-domain analysis here to express this effect, although a more refined s-domain analysis that incorporates a delay relationship between $\phi_{\rm in}$ and $\phi_{\rm out}$ works as well. Fig. 3 shows the representation of Type I DLLs in the z domain. $K_{\rm DL}$ is the delay line gain in radians per cycle per volt, where a cycle refers to a sampling period (also a reference clock period). For the special case of a multiplying DLL [3], $K_{\rm DL}$ automatically includes the effect of the multiplication ratio because it refers to the accumulated phase over one reference clock cycle in the reference clock phase domain (i.e., one reference clock cycle is 2π). In some systems, the reference clock buffers before the

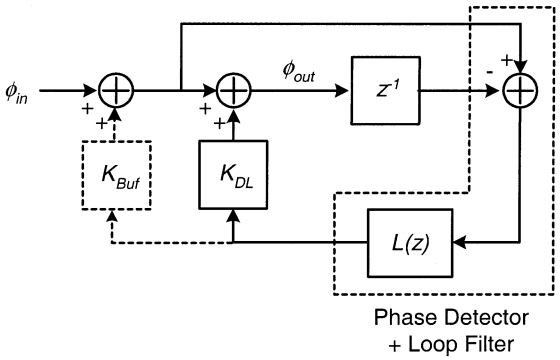


Fig. 3. Z-domain model of Type I DLL.

DLL are also under the control of the loop for signal conditioning and can be modeled by $K_{\rm Buf}$ in radians per volt. However, $K_{\rm Buf}$ is usually negligible compared to $K_{\rm DL}$ and can be safely ignored. The z^{-1} block represents the fact that the phase detector compares the current reference clock edge with the oscillator output derived from the previous reference clock edge. For the first part of the analysis, we assume that the loop filter consists of only a capacitor, as is commonly the case. In this case, $L(z)$ is given by

$$L(z) = \frac{K_{\rm CP} z}{z-1}. \tag{2}$$

$K_{\rm CP}$ is the charge pump and loop filter gain in volts per cycle per radian. In terms of the charge pump current, $I_{\rm CP}$, and the loop filter capacitance, C_f, $K_{\rm CP}$ is equal to $I_{\rm CP}T_i/2\pi C_f$, where T_i is the sampling period. The jitter transfer is given by

$$\frac{\phi_{\rm out}}{\phi_{\rm in}} = \frac{(1+K_{\rm CP}K_{\rm DL})z-1}{z-(1-K_{\rm CP}K_{\rm DL})} = \frac{(1+\alpha)z-1}{z-(1-\alpha)} \tag{3}$$

which contains a pole at $1-\alpha$ and a zero at $1/(1+\alpha)$. For nonzero α, jitter peaking can never be eliminated. Fig. 4 shows the magnitude of the jitter transfer up to half the sampling frequency for $T_i = 8$ ns, $K_{\rm DL} = 22$ radians per cycle per volt, and $K_{\rm CP} = 0.0034$ volts per cycle per radian (e.g., $I_{\rm CP} = 20$ μA, $C_f = 7.5$ pF). Jitter above this frequency gets aliased down because of the discrete-time nature of the system. The maximum jitter peaking can be calculated with $z = -1$ (at half the sampling frequency) and is given by

$$P_1 = \frac{2+\alpha}{2-\alpha}. \tag{4}$$

In this example, the maximum jitter peaking is 0.66 dB. It can be seen that a higher tracking bandwidth and a smaller lock time, both desirable features, increase the amount of jitter peaking. Assuming that the reference clock jitter is white, the root-mean-square (rms) jitter would be amplified by 0.63 dB. This is almost the same as the maximum jitter peaking and is due to the fact that all frequencies above the pole are amplified by the maximum jitter peaking.

Jitter peaking occurs in Type I DLL because it cannot distinguish between input clock jitter and output clock jitter. For example, when the phase comparator sees $\phi_{\rm in}$ lag $\phi_{\rm out}$, it could mean that $\phi_{\rm in}$ has a sudden lagging jitter or that the delay between $\phi_{\rm in}$ and $\phi_{\rm out}$ suddenly became smaller. The former requires that the delay be decreased and the latter increased in

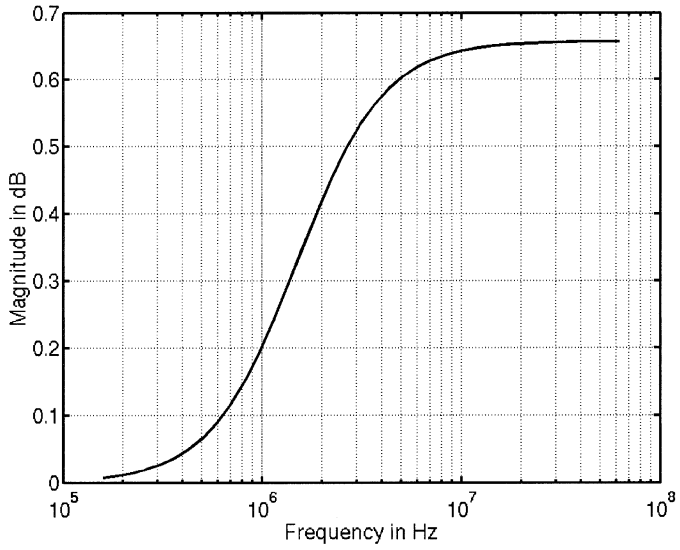


Fig. 4. Jitter transfer function of first-order Type I DLL.

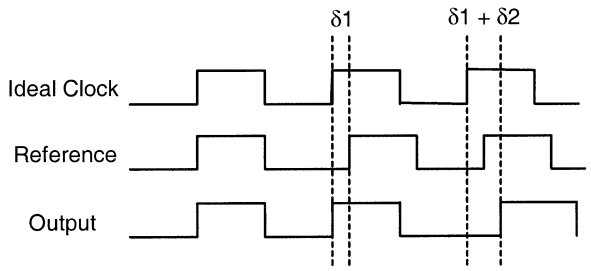


Fig. 5. Timing diagram illustrating jitter peaking.

order to prevent any jitter in $\phi_{\rm out}$. The two scenarios have conflicting requirements. Since the Type I DLL adjusts the delay between $\phi_{\rm in}$ and $\phi_{\rm out}$, the latter must be done in order to prevent positive feedback. This means that any sudden jitter in $\phi_{\rm in}$ is temporarily amplified until this jitter propagates to $\phi_{\rm out}$ and the loop reacts in the correct direction.

Fig. 5 depicts jitter peaking in the time domain. We use a simple case where the input and the output are the same frequency and the DLL attempts to lock them 360° out of phase. The input and output both follow the ideal clock until a positive phase step, δ_1, occurs in the input clock. Since this phase step has not propagated to the output, the phase detector sees an instantaneous phase difference between input and output and interprets it as a decrease in the delay. The delay is increased by δ_2, resulting in an overall phase jitter of $\delta_1 + \delta_2$ at the next output clock edge.

As mentioned earlier, jitter peaking can be reduced by decreasing the loop gain. This degrades the tracking bandwidth and the lock time. In DLL applications, tracking bandwidth is not very critical for canceling the self-noise of the circuit, as phase error is not accumulated over multiple reference cycles. In cases where lock time is important, a special turbo mode, in which the loop bandwidth is increased during acquisition, can be implemented. The steady-state loop gain should be minimized in order to reduce jitter peaking.

IV. Jitter Amplification Reduction

This section is concerned with the reduction of high-frequency jitter transfer in Type I DLLs. There are two benefits of these techniques. First, if the input noise spans a wide range of frequencies (e.g., white), the output jitter is significantly reduced. Second, high-frequency jitter is often more detrimental than low-frequency jitter. For example, in clocked digital circuits, cycle-to-cycle jitter is more important than peak-to-peak jitter since cycle-to-cycle jitter directly reduces the frequency of operation and stresses the clock distribution network by requiring the clock buffers to pass a skinnier pulse. A larger cycle-to-cycle jitter for a given amount of peak-to-peak jitter corresponds to a higher jitter frequency.

The first technique, called *loop filtering*, places an additional filter within the DLL. In some systems, this loop filter is inherent to the chosen architecture and does not need explicit design [3], [8], [9]. The second technique, called *phase filtering*, places a filter after the DLL.

A. Loop Filtering

As mentioned earlier, even when the reference clock jitter spans a wide range of frequencies, significant jitter amplification still occurs because all high-frequency jitter is amplified. One way to reduce high-frequency jitter amplification without compromising loop bandwidth is by introducing a high-frequency noise filter within the DLL. The simplest form is a first-order filter with a pole located at ω_p. In this case, $L(z)$ in Fig. 3 is given by

$$L(z) = K_{\rm CP}\frac{z}{z-1}\frac{(1-e^{-\omega_p T_i})z}{z-e^{-\omega_p T_i}} = K_{\rm CP}\frac{z}{z-1}\frac{(1-a)z}{z-a} \tag{5}$$

where $a = e^{-\omega_p T_i}$. The jitter transfer becomes

$$\begin{aligned}\frac{\phi_{\rm out}}{\phi_{\rm in}} &= \frac{1+Kg_z(z)}{1+Kg_p(z)}\\ K &= K_{\rm DL}K_{\rm CP}(1-a)\\ g_z(z) &= \frac{z^2}{(z-1)(z-a)}\\ g_p(z) &= \frac{z}{(z-1)(z-a)}.\end{aligned} \tag{6}$$

It is convenient to express the jitter transfer in the above form because root locus techniques can be used to derive the locations of the closed-loop poles and zeros as a function of the loop gain K, as shown in Fig. 6. The closed-loop poles and zeros start from the open-loop poles at $z=1$ and $z=a$ for $K=0$. As K increases, they move toward each other, with the closed-loop poles moving at a faster rate. Therefore, even when the loop is overdamped (before the closed-loop poles become complex), jitter peaking still exists, as shown in Fig. 7. Notice that a z-domain pole or zero z_o can be converted to an s-domain pole or zero ω_o by

$$z_o = e^{-\omega_o T_i} \tag{7}$$

for $0 < z_o < 1$. Therefore, a larger z_o gives a smaller ω_o and *vice versa*. The high-frequency jitter peaking now becomes

$$P_2 = \frac{a(2-K_{\rm CP}K_{\rm DL})+(2+K_{\rm CP}K_{\rm DL})}{a(2+K_{\rm CP}K_{\rm DL})+(2-K_{\rm CP}K_{\rm DL})} \tag{8}$$

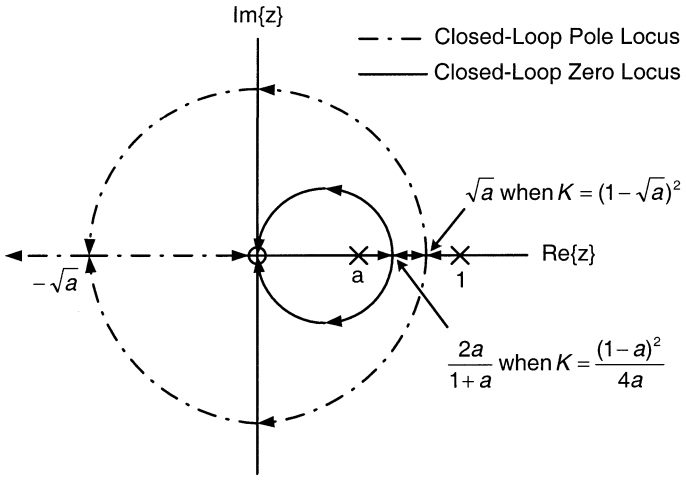


Fig. 6. Root loci of the closed-loop poles and zeros.

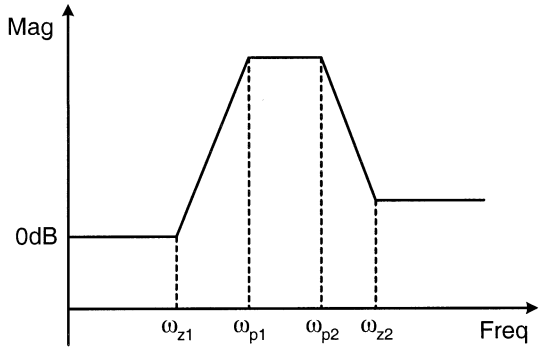


Fig. 7. Jitter peaking in second-order Type I DLL.

where the first terms in the numerator and the denominator are due to the added pole. Since $a < 1$ in order for the system to be stable, $P_2 > 1$. However, compared to a first-order system, the high-frequency jitter peaking has been reduced by

$$\frac{P_2}{P_1} = \frac{1 + (a/P_1)}{1 + aP_1} \tag{9}$$

where P_1 is given by (4). Fig. 8 shows the jitter transfer for the above example with an additional pole at 6.5 MHz, which makes the loop slightly overdamped. Because of the filtering by the additional pole, the maximum jitter peaking reduces slightly to 0.63 dB. More importantly, the high-frequency jitter peaking has been reduced to ∼0.1 dB. For white phase noise in the reference clock, the overall jitter amplification has been reduced to 0.18 dB.

Fig. 9 shows white noise amplification and the maximum jitter peaking versus the loop pole location. Although the maximum jitter peaking remains approximately the same, white noise amplification decreases significantly with a lower pole since more high-frequency components within the loop are filtered out. When the pole falls below the vertical line, however, the loop becomes increasingly oscillatory and unstable. The z-domain pole location where the damping factor equals to one and below which the loop begins to show oscillatory behavior, according to Fig. 6, is given by

$$a_c = \left(\frac{1 - K_{\mathrm{DL}}K_{\mathrm{CP}}}{1 + K_{\mathrm{DL}}K_{\mathrm{CP}}}\right)^2. \tag{10}$$

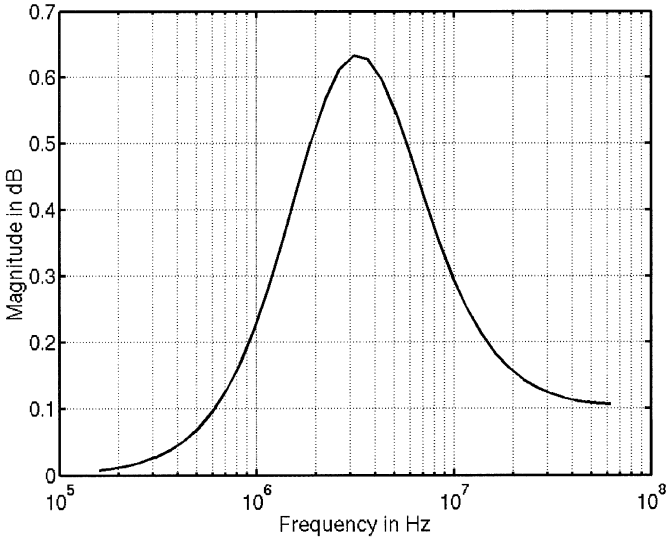


Fig. 8. Jitter transfer function of second-order Type I DLL.

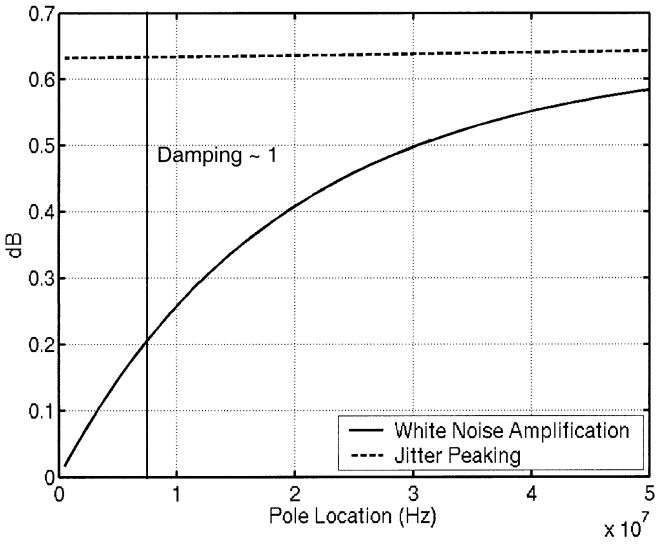


Fig. 9. Effect of the loop pole location on noise amplification and jitter peaking.

This expression can be converted to frequency (Hertz) using (7). The pole location should be set close to this value for reasonable input jitter filtering and overdamped loop dynamics.

Finally, it should be mentioned that a higher order loop filter further decreases the high-frequency jitter transfer. The jitter transfer can be calculated by a routine extension of the above analysis. For the sake of brevity, we will not expand on it further.

B. Phase Filtering

When a DLL is used as a clock multiplier (a multiplying DLL, or MDLL), it is advantageous to add a phase-domain filter to further reduce jitter transfer at high frequencies. One possibility of this phase filter is a PLL. To prevent excessive jitter accumulation, the PLL can be easily designed to have a high bandwidth as its sampling frequency is at the multiplied frequency. At the same time, high-frequency jitter can be significantly reduced. For example, with a 125-MHz reference clock

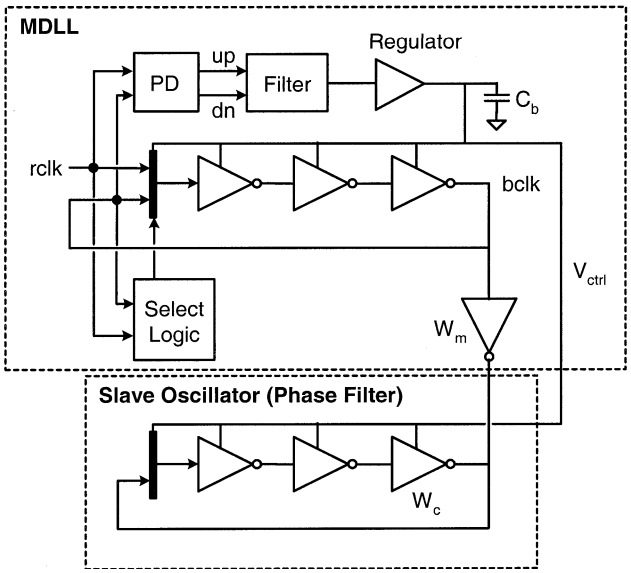


Fig. 10. Multiplying DLL and slave oscillator.

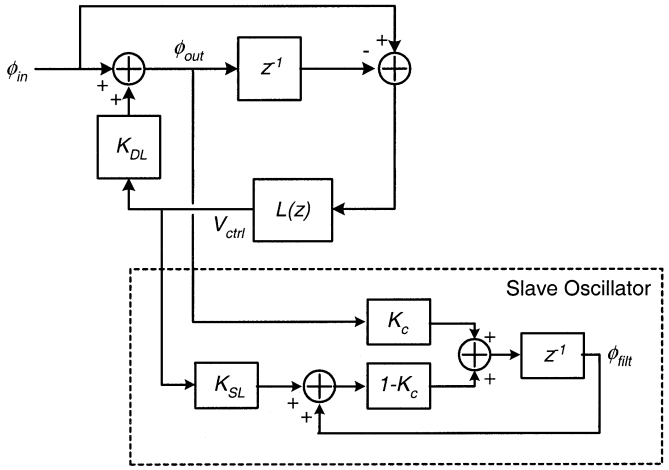


Fig. 11. Z-domain model of MDLL and slave oscillator.

and a multiplication factor of 8, the MDLL produces a 1-GHz clock, which is also the sampling frequency of the PLL. To provide a significant high-frequency jitter filtering while retaining a high tracking bandwidth to suppress jitter accumulation, the bandwidth of the PLL can be easily designed to be, for instance, 20 MHz (only 1/50 of the sampling frequency). This means that the DLL output jitter above this frequency is heavily attenuated.[1] On the other hand, a bandwidth of 20 MHz is also much higher than that which can be achieved if a PLL is used directly to multiply the reference clock. The stability of the PLL usually requires the bandwidth to be below 1/20 of the sampling frequency for adequate design margins. In our example, this would make the upper bound of a multiplying PLL's tracking bandwidth ~6 MHz.

The traditional second-order and third-order PLLs, however, add significant complexity. A simpler phase filter can be implemented using injection locking [13]. An implementation based on a CMOS ring oscillator is shown in Fig. 10 [3]. The reference clock $rclk$ is multiplied by an MDLL. For a multiplication factor of N, the delay line output is fed back (configured as a ring oscillator) for N cycles before the front-end multiplexer (mux) selects the reference clock. The select logic counts the number of $bclk$ edges and determines when to inject $rclk$ into the ring oscillator. A regulator adjusts the delay of the delay elements while isolating them from supply noise. (Please refer to [3] for implementation details of the MDLL.) The slave oscillator uses the same delay element as the MDLL and runs at nominally the same frequency. The injection strength is defined as $S_c = W_m/(W_c+W_m)$ and is approximately the amount of corrected phase per injection divided by the phase error between the master and slave oscillators. For small phase errors, the system can be modeled as shown Fig. 11. Note that since the sampling frequency of this z-domain model is equal to the reference clock frequency, the effective coupling coefficient in the sampling frequency domain, K_c, is related to the coupling coefficient in the multiplied frequency domain, S_c, by $K_c = 1-(1-S_c)^N$. We use a second-order DLL [see (5)] for the following analysis. The MDLL and the slave oscillator are coupled through two paths. The output of the MDLL $\phi_{\rm out}$ is injected into the slave oscillator and the control signal of the MDLL $V_{\rm ctrl}$ sets the oscillation frequency of the slave oscillator. $K_{\rm SL}$ represents the gain of the slave oscillator. If the coupling through $V_{\rm ctrl}$ is ignored for the moment, then the jitter transfer of the slave oscillator from $\phi_{\rm out}$ to $\phi_{\rm filt}$ is given by

$$\frac{\phi_{\rm filt}}{\phi_{\rm out}} = \frac{K_c}{z-(1-K_c)}. \tag{11}$$

This is a simply a first-order low-pass filter with a single pole located at

$$p_0 = \frac{-\ln(1-K_c)}{T_i} \quad \text{(rad/s)} \tag{12}$$

where T_i is the injection frequency. Notice that, again, because injection occurs at the multiplied frequency, the bandwidth of this phase filter can be high to suppress its own jitter accumulation while still filtering out a significant amount of high-frequency jitter. If we assume $K_{\rm SL}$ is equal to $K_{\rm DL}$, the coupling of the MDLL and the slave oscillator through $V_{\rm ctrl}$ modifies the jitter transfer of the slave oscillator as follows:

$$\frac{\phi_{filt}}{\phi_{\rm out}} = \frac{K_c}{z-(1-K_c)} \frac{1+\dfrac{K}{K_c}\,g_d(z)}{1+Kg_z(z)} \tag{13}$$

$$g_d(z) = \frac{z(z-(1-K_c))}{(z-1)(z-a)} \tag{14}$$

where K, $g_z(z)$, and a are given by (6). The first term in (13) is identical to (11). Root locus techniques similar to Fig. 6 can be used to show that the second term in (13) exhibits similar peaking behavior as the jitter transfer of the second-order Type I DLL described by (6) and Fig. 8. The important differences here are that the peaking starts earlier and the poles coincide with the zeros of (6). Thus, the peaking range for the second term of (13) is wider than that for (6). Equation (13) indicates that the addition of a slave oscillator not only filters out high-frequency

[1]It can be shown that if the DLL output jitter is assumed white, then the noise bandwidth for an overdamped second-order PLL is approximately 30 MHz in our example.

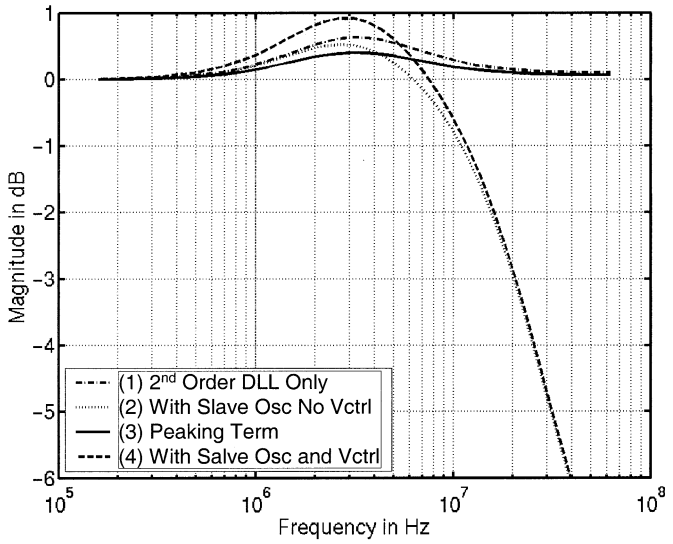


Fig. 12. Effect of the slave oscillator on the jitter transfer of second-order Type I DLL.

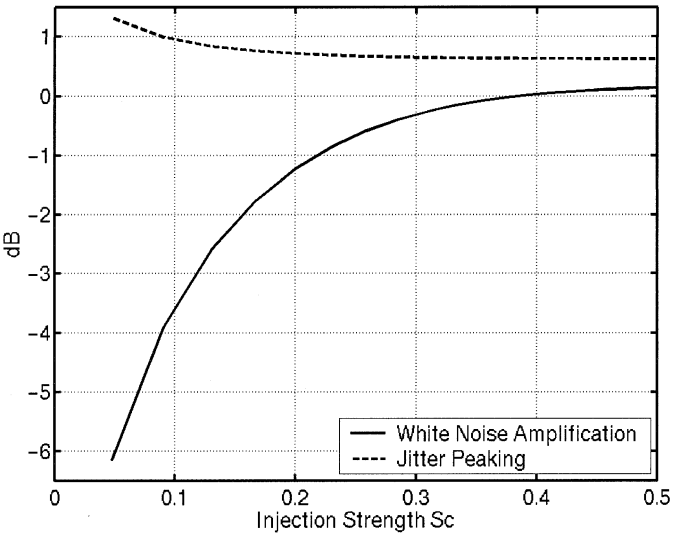


Fig. 13. Effect of the slave oscillator injection strength on noise amplification and jitter peaking.

jitter but also possibly increases jitter peaking, as is the case when a PLL is used for phase filtering. Fig. 12 compares the jitter transfer functions of

1) a second-order Type I DLL only;
2) a second-order Type I DLL plus a slave oscillator, without considering $V_{\rm ctrl}$ coupling;
3) the peaking term in (13) due to $V_{\rm ctrl}$ coupling;
4) a second-order Type I DLL plus a slave oscillator, with $V_{\rm ctrl}$ coupling.

The loop parameters are identical to the examples above. The injection strength S_c is 1/10, which corresponds to a slave oscillator bandwidth of about 20 MHz. Fig. 13 shows the maximum jitter peaking and white noise amplification versus the injection strength S_c. The lower bound of the injection strength is constrained by self-noise correction and lock range of the slave oscillator and the peaking due to the second term of (13), which increases as the injection strength decreases.

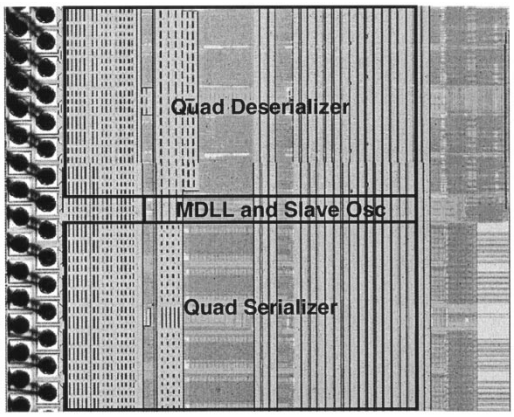


Fig. 14. Die photo of the MDLL and slave oscillator circuits.

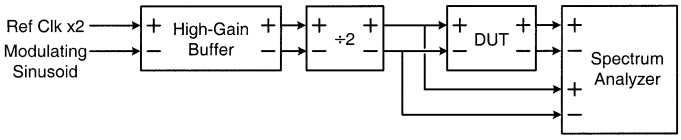


Fig. 15. Experimental setup of jitter transfer measurement.

Finally, it should be mentioned that the phase filtering techniques presented in this section filter out high-frequency jitter not only *transferred* from the reference clock but also *generated* within the DLL itself. The jitter generated within a multiplying DLL, for example, often contains more high-frequency components compared with that transferred from the reference clock. The reasons are the higher oscillation frequency and the spurious tones caused by the reference clock injection [1], [15]. The phase filtering techniques described here are equally applicable to these high-frequency noise sources.

V. Experimental Results

A system comprising a multiplying DLL with a slave oscillator, as shown in Fig. 10, has been fabricated in a 0.18-μm CMOS technology [3]. Fig. 14 shows the die micrograph of a clock multiplier composed of an MDLL and a slave oscillator. The clock multiplier supplies a maximum of 1.56-GHz clock source for four serializers and four deserializers running at a maximum bit rate of 3.125 Gb/s. The active area of the clock multiplier is 80 μm by 1 mm. The loop filter in the MDLL is a simple charge pump described by (2). Since the regulator creates an additional pole within the loop, the expected jitter transfer is similar to Fig. 8 without a slave oscillator and Fig. 12 with a slave oscillator. The design parameters are approximately the same as those in the examples presented above for a 125-MHz reference clock and a multiplication factor of 8.

Fig. 15 shows the experimental setup for jitter transfer measurement. A clock source with twice the required reference clock frequency and a modulating sinusoid are applied to a high-gain buffer. The resulting waveform contains both sinusoidal jitter and sinusoidal duty-cycle distortion since the modulating sinusoid varies the switching threshold of the high-gain buffer. A divide-by-two circuit is used to remove this duty-cycle distortion, resulting in a reference clock with the required frequency and phase modulation. A spectrum analyzer

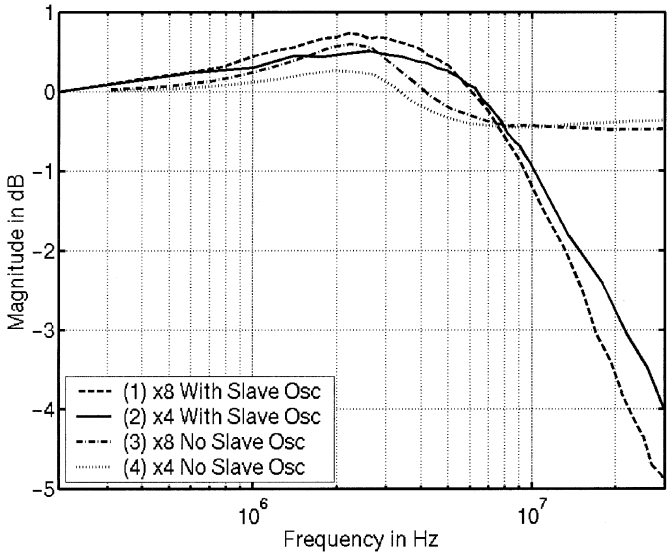


Fig. 16. Measured jitter transfer of the multiplying DLL with and without a slave oscillator.

measures the sideband amplitudes of the input and the output of the device under test (DUT) across a range of modulating frequency to obtain the jitter transfer function. Fig. 16 shows the measured jitter transfer functions. The four curves are for

1) a 125-MHz reference clock and a multiplication factor of 8 with a slave oscillator;
2) a 250-MHz reference clock and a multiplication factor of 4 with a slave oscillator;
3) a 125-MHz reference clock and a multiplication factor of 8 without a slave oscillator;
4) a 250-MHz reference clock and a multiplication factor of 4 without a slave oscillator.

As expected, a higher multiplication factor leads to a larger jitter peaking since the loop gain is larger. The peaking increases slightly when a slave oscillator is added. It also creates a 20-MHz pole in the jitter transfer, filtering out a significant amount of high-frequency jitter. These results are consistent with the predictions of our analytical model. The location of the pole indicates that the injection strength S_c is about 1/10. Notice that for the two cases without a slave oscillator, the high-frequency jitter transfer is smaller than that predicted by (8) and Fig. 8 for a second-order Type I DLL. This is due to the presence of higher poles within the DLL.

VI. Summary

Through a z-domain model, we have shown and verified that in Type I DLLs, jitter peaking always exists and high-frequency jitter does not get attenuated, as previous analyses suggest. This is true even in a first-order DLL and an overdamped second-order DLL. To avoid significant amplification of the reference clock jitter, the loop gain should be minimized. Jitter transfer attenuation techniques through loop filtering and phase filtering have also been discussed in detail. These techniques extend the usefulness of DLLs to cases where the reference clock is noisy. A multiplying DLL incorporating the techniques presented in this paper has been fabricated. The measurement results have been shown to be consistent with the prediction of our analytical model. The reference clock jitter amplification effect described in this paper should be carefully evaluated against the self-noise jitter accumulation effect described in previous literature [2], [6], [14] to determine the best overall jitter performance.

Acknowledgment

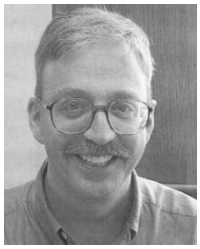
The authors would like to thank J. Edmondson, S. Gowni, T. Stone, R. Rathi, V. Radulescu, S. Tell, C. Okruhlica, K. Kinder, and M. Pimentel for their contributions to this work.

References

[1] G. Chien and P. Gray, "A 900-MHz local oscillator using a DLL-based frequency multiplier technique for PCS applications," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2000, pp. 202–203, 458.
[2] T. Lee *et al.*, "A 2.5-V CMOS delay-locked loop for an 18-Mbit 500-Megabyte/s DRAM," *IEEE J. Solid-State Circuits*, vol. 29, pp. 1491–1496, Dec. 1994.
[3] R. Farjad-Rad *et al.*, "A 0.2–2-GHz 12-mW multiplying DLL for low-jitter clock synthesis in highly integrated data-communication chips," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2002, pp. 76–77.
[4] A. Waizman, "A delay line loop for frequency synthesis of de-skewed clock," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 1994, pp. 298–299.
[5] S. Tam *et al.*, "Clock generation and distribution for the first IA-64 microprocessor," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1545–1552, Nov. 2000.
[6] B. Kim, T. Weigandt, and P. Gray, "PLL/DLL system noise analysis for low jitter clock synthesizer design," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 4, May 1994, pp. 31–38.
[7] T. Lee and J. F. Bulzacchelli, "A 155-MHz clock recovery delay- and phase-locked loop," *IEEE J. Solid-State Circuits*, vol. 27, pp. 1736–1746, Dec. 1992.
[8] M.-J. E. Lee, W. J. Dally, and P. Chiang, "Low-power, area efficient, high speed I/O circuit techniques," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1591–1599, Nov. 2000.
[9] G.-Y. Wei *et al.*, "A variable-frequency parallel I/O interface with adaptive power-supply regulation," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1600–1610, Nov. 2000.
[10] S. Sidiropoulos and M. Horowitz, "A semidigital dual delay-locked loop," *IEEE J. Solid-State Circuits*, vol. 32, pp. 1683–1692, Nov. 1997.
[11] J. G. Maneatis, "Low-jitter process-independent DLL and PLL based on self-biased techniques," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1723–1732, Nov. 1996.
[12] W. J. Dally and J. Poulton, *Digital Systems Engineering*. Cambridge, U.K.: Cambridge Univ. Press, 1998.
[13] P. Kinget *et al.*, "An injection-locking scheme for precision quadrature generation," *IEEE J. Solid-State Circuits*, vol. 37, pp. 845–851, July 2002.
[14] T. Weigandt, B. Kim, and P. Gray, "Analysis of timing jitter in CMOS ring oscillators," in *Proc. Int. Symp. Circuits and Systems*, vol. 4, June 1994, pp. 27–30.
[15] G. Chien, "Low-noise local oscillator design techniques using a DLL-based frequency multiplier for wireless applications," Ph.D. dissertation, Univ. of California, Berkeley, 2000.

M.-J. Edward Lee (M'01) received the B.S. degree in electrical engineering and computer sciences from the University of California, Berkeley, in 1997 and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 2000 and 2001, respectively.

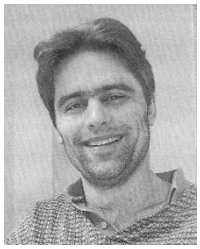
During the summer of 1996, he was with Hewlett-Packard Laboratories, Palo Alto, CA, working on an RF switched-mode power amplifier linearization technique. Since 2000, he has been with Velio Communications, Milpitas, CA, where he leads the next-generation gigabit serial I/O development. He has published more than ten conference and journal papers in the area of high-speed communications.



William J. Dally (M'80–SM'01–F'02) received the B.S. degree in electrical engineering from Virginia Polytechnic Institute, Blacksburg, the M.S. degree in electrical engineering from Stanford University, Stanford, CA, and the Ph.D. degree in computer science from the California Institute of Technology (Caltech), Pasadena.

He and his group have developed system architecture, network architecture, signaling, routing, and synchronization technology that can be found in most large parallel computers today. While at Bell Telephone Laboratories, he contributed to the design of the BELLMAC32 microprocessor and designed the MARS hardware accelerator. At Caltech, he designed the MOSSIM Simulation Engine and the Torus Routing Chip which pioneered wormhole routing and virtual-channel flow control. While a Professor of electrical engineering and computer science at the Massachusetts Institute of Technology, Cambridge, his group built the J-Machine and the M-Machine, experimental parallel computer systems that pioneered the separation of mechanisms from programming models and demonstrated very low overhead synchronization and communication mechanisms. He is currently a Professor of electrical engineering and computer science at Stanford University, where his group has developed the Imagine processor, which introduced the concepts of stream processing and partitioned register organizations. He has worked with Cray Research and Intel to incorporate many of these innovations in commercial parallel computers, and with Avici Systems to incorporate this technology into Internet routers, and he cofounded Velio Communications to commercialize high-speed signaling technology. He currently leads projects on high-speed signaling, computer architecture, and network architecture. He has published over 150 papers in these areas and is an author of the textbook *Digital Systems Engineering* (Cambridge, U.K.: Cambridge University Press, 1998).

Dr. Dally is a Fellow of the ACM and has received numerous honors including the ACM Maurice Wilkes Award.



Trey Greer received the B.S. degree in mathematics and physics from the University of the South, Sewanee, TN, in 1984 and the M.S. degree in computer science from the University of North Carolina at Chapel Hill in 1988.

Since 1988, he has been involved in various startups in the Chapel Hill and Bristol, U.K., areas. His research interests include parallel architectures for computer graphics and clock generation for high-speed data links.

Hiok-Tiaq Ng (S'93–M'01) was born in Penang, Malaysia. He received the B.A. degree in mathematics and physics from Whitman College, Walla Walla, WA, and the B.S.E.E. degree from the California Institute of Technology, Pasadena, both in 1992, and the M.S.E.E. degree from Carnegie Mellon University, Pittsburgh, PA, in 1994, working on low-voltage low-power current steering logic families. He received the Ph.D. degree from Arizona State University, Tempe, in 1999, working on low-voltage, highly-linear, high-frequency continuous-time filters using multistage operational amplifiers. His other research work involved designing read channel circuits for hard disk drives.

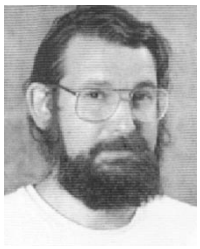
From 1994 to 1995, he was a mixed-signal circuit Design Engineer with Texas Instruments Incorporated, Dallas, TX. Since 1999, he has been with Velio Communications, Milpitas, CA, where he designs high-integration clock multiplier circuits for high-speed serial links.

Dr. Ng is a member of Sigma Xi and Tau Beta Pi and has received an SRC Inventor's Recognition Award.

Ramin Farjad-Rad (S'95–M'00) received the B.Sc. degree in electrical engineering in 1993 from Sharif University of Technology, Tehran, Iran, and the M.Sc. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1995 and 1999, respectively.

He was with SUN Microsystems Laboratories working on a 1.25-Gb/s serial transceiver for Fiber channel and Gigabit Ethernet standards in summer of 1995. During the summer of 1996, he was with LSI Logic, where he examined different multigigabits-per-second serial transceiver architectures. He was part of the founding team of Velio Communications, a communication chip manufacturer in Milpitas, CA, where he has been in charge of multigigahertz CMOS circuit development as the company's Research Lead since 1999. He holds nine patents and patents pending in the area of high-speed integrated circuits and has published more than ten conference and invited journal papers.

Dr. Farjad-Rad ranked third in the 20th International Physics Olympiad in 1989.



John Poulton (M'85–SM'90) received the B.S. degree in physics from the Virginia Polytechnic Institute, Blacksburg, in 1967, the M.S. degree in physics from the State University of New York at Stony Brook in 1969, and the Ph.D. degree in physics from the University of North Carolina at Chapel Hill in 1980.

From 1981 to 1999, he was a Researcher in the Department of Computer Science, University of North Carolina at Chapel Hill, where from 1995 he held the rank of Research Professor. He did research on VLSI-based architectures for graphics and imaging and was a principal contributor to the design and construction of several experimental high-performance graphics systems. Since 1999, he has served as Chief Engineer for Velio Communications, Milpitas, CA, where he is engaged in the development of gigabit signaling systems.

Ramesh Senthinathan received the B.S. degree in computer engineering from the State University of New York at Buffalo in 1984 and the M.S. and Ph.D. degrees in electrical engineering from the University of Arizona, Tucson, in 1986 and 1992, respectively.

He is currently Director of Engineering with Velio Communications, Milpitas, CA, responsible for technology and product development. From 1995 to 2000, he was with Intel Corporation as a Design Manager for the microprocessor group in Folsom, CA. He was responsible for all aspects of design and integration of Pentium® Pro (shrink), Pentium® Pro (1 Meg), and the flagship first Pentium® III microprocessor design. With Level One communication acquisition, he moved to the Intel DSL group as a Distinguished Engineer responsible for analog front-end and UDSL design. From 1993 to 1995, he was with Motorola as a Principal Engineer responsible for 16-bit DSP (56K Motorola DSP) design for cellular communications. He was a Staff Design Engineer in the IBM mainframe computer group in Fishkill, NY, from 1992 to 1993. During 1986 to 1989, he was a Design Engineer with the ASIC and microcontroller groups, Intel Corporation, Chandler, AZ.