A 3.2 Gb/s CDR Using Semi-Blind Oversampling to Achieve High Jitter Tolerance

Marcus van Ierssel, *Member, IEEE*, Ali Sheikholeslami, *Senior Member, IEEE*, Hirotaka Tamura, *Member, IEEE*, and William W. Walker, *Member, IEEE*

***Abstract*—A hybrid CDR is presented that embeds a 5× blind-oversampling CDR within a conventional phase-tracking CDR. This hybrid CDR has a jitter tolerance that is the product of the individual jitter tolerances. In this implementation, the jitter tolerance of a phase-tracking CDR alone is increased by a factor of 32 at frequencies below its loop filter's bandwidth, while maintaining the high-frequency jitter tolerance of a 5× blind-oversampling CDR. Measured data from a 0.11 μm CMOS test chip at 2.4 Gb/s confirm a 200 UI peak-to-peak jitter tolerance for a 200 kHz jitter. The test chip operates from 1.9 Gb/s to 3.5 Gb/s with a BER less than 10^{-11}, consuming 115 mW at 2.4 Gb/s.**

***Index Terms*—Blind oversampling, clock and data recovery (CDR), jitter tolerance, oversampling, phase tracking.**

I. Introduction

JITTER tolerance refers to the maximum amplitude of sinusoidal jitter (as a function of frequency) that can be tolerated without causing data recovery errors. High jitter tolerance is essential to applications such as spread-spectrum clocking [1], [2] where the clock embedded in the received data can deviate from its nominal phase by as many as a few hundred UI. Increased jitter tolerance also introduces flexability into the tradeoff between jitter tolerance, transfer, and generation. The traditional phase-tracking CDR has a jitter tolerance that is inversely proportional to jitter frequency, within the bandwidth of its loop filter [3]. While increased jitter tolerance can be achieved through increased loop bandwidth, this may not be practical due to loop stability concerns, and increased jitter transfer [4].

Instead of increasing the loop bandwidth, one alternative is to change the CDR architecture. In [5], an analog phase shifter (DLL) is embedded within a phase-tracking CDR. This DLL absorbs jitter before it reaches the CDR, allowing it to operate under jitter conditions that would cause a conventional phase-tracking CDR to fail. Such an architecture can increase the jitter tolerance by factor equal to the phase-shift range of the DLL. However, DLLs have a limited phase shift range, only 2–3 UI in this implementation.

Manuscript received October 4, 2006; revised February 28, 2007.

M. van Ierssel and A. Sheikholeslami are with the Department of Electrical Engineering, University of Toronto, Toronto, ON M5S 3G4 Canada (e-mail: vane@eecg.toronto.edu; ali@eecg.toronto.edu).

H. Tamura is with Fujitsu Laboratories Ltd., Kanagawa 223-8522, Kawasaki, Japan.

W. W. Walker is with Fujitsu Laboratories of America, Inc., Sunnyvale, CA 94085 USA (e-mail walker@fla.fujitsu.com).

Digital Object Identifier 10.1109/JSSC.2007.905233

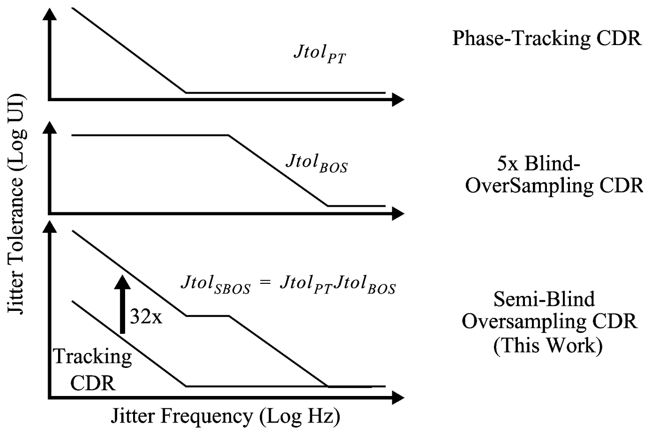


Fig. 1. CDR jitter tolerance.

We propose to expand this phase shift range by substituting the analog phase shifter (DLL) with a digital phase shifter, in the form of a blind-oversampling CDR [6]. We refer to this hybrid combination of phase tracking and blind-oversampling CDRs as a *semi-blind oversampling* CDR [7]. As shown in Fig. 1, the semi-blind oversampling technique produces a jitter tolerance, $Jtol_{\mathrm{SBOS}}$, equal to the product of the phase tracking jitter tolerance, $Jtol_{\mathrm{PT}}$, and the blind oversampling jitter tolerance, $Jtol_{\mathrm{BOS}}$, thereby increasing the low-frequency jitter tolerance by the phase shift range (32 UI in our design). This paper provides the theory and detailed implementation of this technique, as well as measured results for a design that was fabricated after the publication of [7].

To provide context for our proposed semi-blind oversampling technique, Section II derives the jitter tolerance of a phase-tracking CDR and that of a blind-oversampling CDR. Section III conceptually describes the proposed semi-blind oversampling CDR, and derives equations for its jitter tolerance. Section IV describes the implementation of the semi-blind oversampling CDR. The simulated and measured results of its implementation are then discussed in Section V. Finally, Section VI summarizes the contributions of this work.

II. Jitter Tolerance in the Phase-Tracking and Blind-Oversampling CDR

A. Phase-Tracking Jitter Tolerance

A phase-tracking CDR employs feedback to keep the recovered clock in phase with the clock embedded in the received data, as shown in Fig. 2. The recovered clock is then used to sample (recover) received data using a sampler. Under ideal

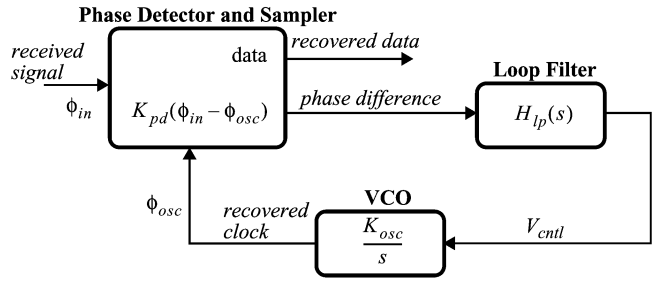


Fig. 2. Phase-tracking CDR.

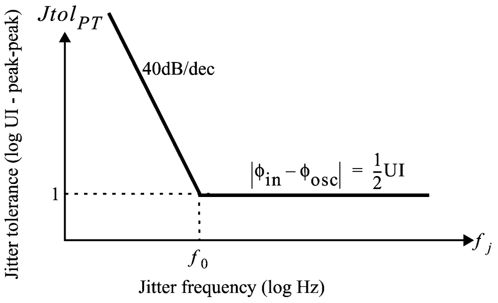


Fig. 3. Jitter tolerance of a phase-tracking CDR under ideal conditions.

conditions, with no ISI or clock jitter, error-free data recovery is achieved when the received data is sampled within 1/2 UI of the nominal sampling point—the center of the data eye. In terms of the input clock phase, $\phi_{\rm in}$, and the recovered clock phase, $\phi_{\rm osc}$, the condition for error-free data recovery is expressed as

$$|\phi_{\rm in} - \phi_{\rm osc}| < \frac{1}{2}{\rm UI}. \tag{1}$$

The jitter tolerance of this CDR is the largest sinusoidal amplitude of $\phi_{\rm in}$ that satisfies (1). For a charge-pump CDR using a series *RC* filter [8], the phase transfer function can be written as

$$\frac{\Phi_{\rm osc}(s)}{\Phi_{\rm in}(s)} = \frac{1+sRC}{1+sRC+\dfrac{s^2C}{K_{\rm pd}K_{\rm osc}}} \tag{2}$$

where $\Phi(s)$ is the Laplace transform of $\phi(t)$.

Combining (2) and (1) and solving for twice the largest amplitude of $\Phi_{\rm in}(s)$, we derive the peak–peak phase-tracking (PT) jitter tolerance:

$$Jtol_{\rm PT} = \left(\frac{K_{\rm pd}K_{\rm osc}}{s^2C} + \frac{K_{\rm pd}K_{\rm osc}R}{s} + 1\right){\rm UI}. \tag{3}$$

In a critically damped loop, the jitter tolerance has two poles at the origin and two zeros at $2\pi f_0 = \sqrt{K_{\rm pd}K_{\rm osc}/C}$, as pictorially shown in Fig. 3. For this type of CDR, the only way to increase the jitter tolerance is to increase f_0, moving the tolerance curve to the right. However, as in all closed-loop feedback systems, increasing f_0 degrades the stability of the loop [4].

Instead of increasing f_0, the semi-blind oversampling CDR proposed in this paper achieves a higher jitter tolerance by embedding a blind-oversampling CDR within a phase-tracking

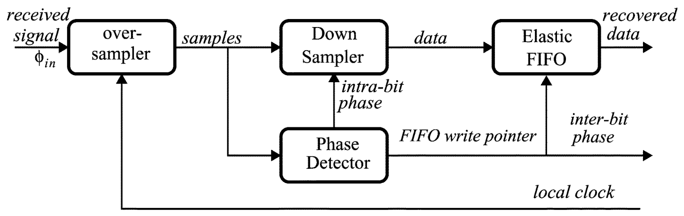


Fig. 4. Blind-oversampling CDR.

CDR. In Section II-B, we examine the jitter tolerance of a blind-oversampling CDR.

B. Blind-Oversampling Jitter Tolerance

Unlike the phase-tracking CDR, the blind-oversampling CDR does not attempt to track the clock embedded in the received data. Instead, it oversamples the received data using a local clock uncorrelated with the embedded clock. As shown in Fig. 4, this feed-forward architecture determines the embedded clock phase from the transitions in the samples. The recovered phase is then used to select (down-sample) the data bits within a window of oversampled data. These bits then enter an elastic FIFO. For every additional bit period that the local clock lags the embedded clock, an additional bit must be delayed (stored) by the FIFO. This makes the occupied FIFO depth proportional to this phase difference with a resolution of 1 UI, a property which will be exploited later for inter-bit phase-detection.

Blind-oversampling has an important feature that we exploit to overcome the sampling restriction (1) of the phase-tracking CDR. Because phase detection and data selection occur after sampling, there is no longer an inherent restriction on the phase difference between the local clock and the clock embedded in the received data. These clocks may differ by many UI, as long as the FIFO is large enough to absorb the difference.

While this CDR has no inherent limit on the phase *difference*, there is a limit of the *rate* of phase change. The phase change between transitions in the received signal must be less than 1/2 UI, otherwise it becomes impossible to identify the number of symbols between transitions. Because oversampling quantizes the phase, this translates to a maximum phase change of 2/5 UI between transitions for 5× oversampling.

The jitter tolerance of a blind-oversampling CDR with a 5× oversampling ratio is illustrated in Fig. 5, where the tolerable sinusoidal jitter amplitude is plotted as a function of jitter frequency. At high jitter frequencies, the jitter period is much smaller than the time between transitions, $t_{\rm trans}$ (upon which the phase detector relies), hence a transition might not occur during one jitter period. At these frequencies, the jitter tolerance is 2/5 UI peak-to-peak.

As the jitter-frequency decreases, the CDR begins to track phase changes in the received signal not exceeding 2/5 UI between transitions, or

$$\left|\frac{d}{dt}\Phi_{\rm in}(t)\right| < \frac{2}{5}\frac{\rm UI}{t_{\rm trans}}. \tag{4}$$

To find the resulting jitter tolerance, we apply a sinusoidal jitter source, $\phi_{\rm in}(t) = 2\pi A_j \sin(2\pi f_j t)$ (in radians), to the CDR, where A_j is the jitter amplitude in units of UI. Inserting

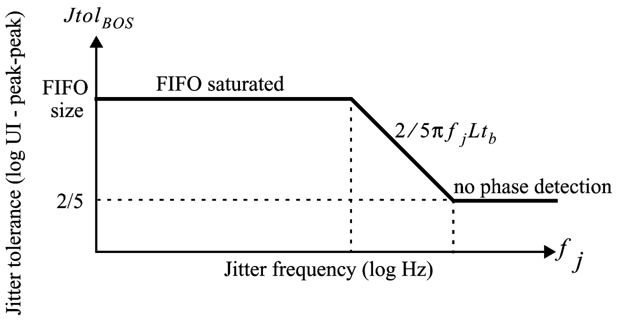


Fig. 5. Blind-oversampling CDR jitter tolerance.

this into (4) and maximizing the derivative gives the following condition for the jitter amplitude:

$$2\pi f_j A_j < \frac{2}{5}\frac{\text{UI}}{t_{\text{trans}}}. \tag{5}$$

This relation shows that the maximum jitter amplitude is a function of the time between transitions. We must satisfy (5) for all possible values of t_{trans}, including the worst-case t_{trans} of Lt_b, where L is the runlength of the input sequence and t_b is the bit time. Therefore, the peak–peak Blind-OverSampling (BOS) jitter tolerance becomes

$$Jtol_{\text{BOS}} = 2A_j = \frac{2}{5\pi f_j L t_b}\text{UI}. \tag{6}$$

The jitter tolerance thus increases at 20 dB/decade, with decreasing frequency, until the CDR's FIFO becomes saturated. At and below this frequency, the peak jitter tolerance remains constant and equal to the FIFO size.

III. Proposed Semi-Blind Oversampling CDR

The contribution of this work is the use of the blind-oversampling CDR to overcome the sampling time restriction of the phase-tracking CDR, as described by (1). This is illustrated in Fig. 6, where a blind-oversampling CDR replaces the sampler in a phase-tracking CDR. We term this hybrid design a *semi-blind* oversampling CDR, as the local clock used for oversampling is no longer blind, but instead tracks phase change within the bandwidth of the phase-tracking loop. However, where a phase-tracking CDR has to maintain a phase difference between received and local clocks of less than 1/2 UI, as specified in (1), the new phase detector allows a phase difference equal to the peak jitter tolerance of the blind-oversampling CDR, or

$$|\phi_{\text{in}} - \phi_{\text{osc}}| < \frac{1}{2}Jtol_{\text{BOS}}. \tag{7}$$

This is similar to the embedding of a DLL inside a PLL as described in [5], in which an analog phase-shifter shifts the phase of the received signal *before* it is applied to a conventional phase-tracking CDR. This approach increases the allowable phase-difference between the recovered and embedded clocks by an amount equal to the tuning range of the analog phase-shifter, which is limited to approximately 2 UI. In contrast, our design uses a blind-oversampling CDR as a digital phase-shifter that increases the allowable phase-difference by an amount equal to its FIFO size. Since the FIFO size can be chosen by the designer, the allowable phase-difference becomes merely another design parameter.

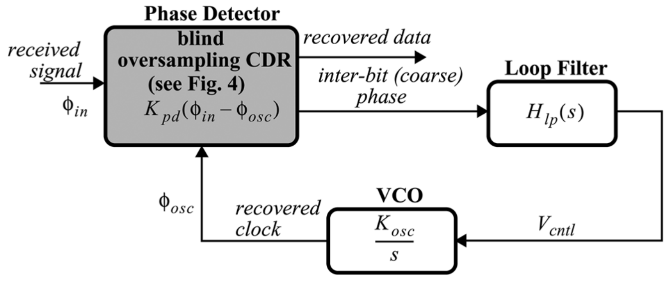


Fig. 6. Semi-blind oversampling CDR.

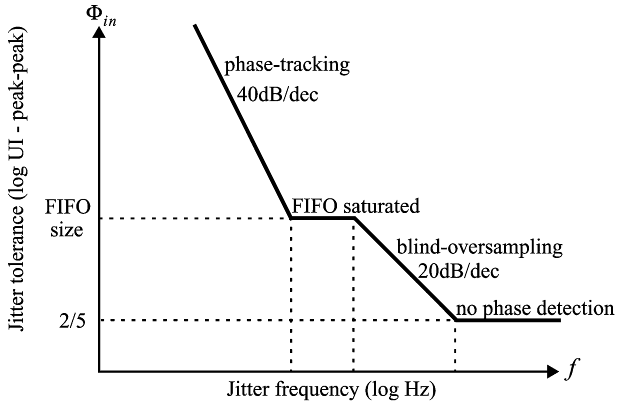


Fig. 7. Semi-blind oversampling CDR jitter tolerance.

Combining (7) with the phase transfer function of the phase-tracking CDR, (2), and solving for twice the maximum value of $\Phi_{\text{in}}(s)$, we obtain the peak–peak jitter tolerance of this Semi-Blind OverSampling (SBOS) CDR:

$$\begin{aligned} Jtol_{\text{SBOS}} &= Jtol_{\text{BOS}}\left(\frac{K_{\text{pd}}K_{\text{osc}}}{s^2C} + \frac{K_{\text{pd}}K_{\text{osc}}R}{s} + 1\right) && (8)\\ &= Jtol_{\text{BOS}}Jtol_{\text{PT}}. && (9)\end{aligned}$$

Thus, the jitter tolerance is the product of the blind-oversampling and phase-tracking tolerances, as shown in Fig. 7. Because the jitter tolerance of the blind-oversampling CDR at low frequencies saturates at the FIFO size, the resulting jitter tolerance at these frequencies is a factor of the FIFO-size higher (32 in our implementation) than that of the conventional phase-tracking CDR.

While this hybrid architecture increases the jitter tolerance of the phase tracking CDR, it does so with little effect on jitter transfer and jitter generation. The jitter transfer characteristic remains that of the original phase tracking CDR. The jitter generation characteristic also remains that of a phase-tracking CDR. However, for small jitter amplitudes, the CDR behaves as if it uses an early/late phase detector, while for large jitter amplitudes, the multi-level phase output of the blind-oversampling CDR (the FIFO depth) produces behaviour consistent with the use of a linear phase-detector.

IV. Design Implementation

Fig. 8 shows the block diagram of our semi-blind oversampling CDR. A 20-phase 800 MHz VCO is used to 5× over-

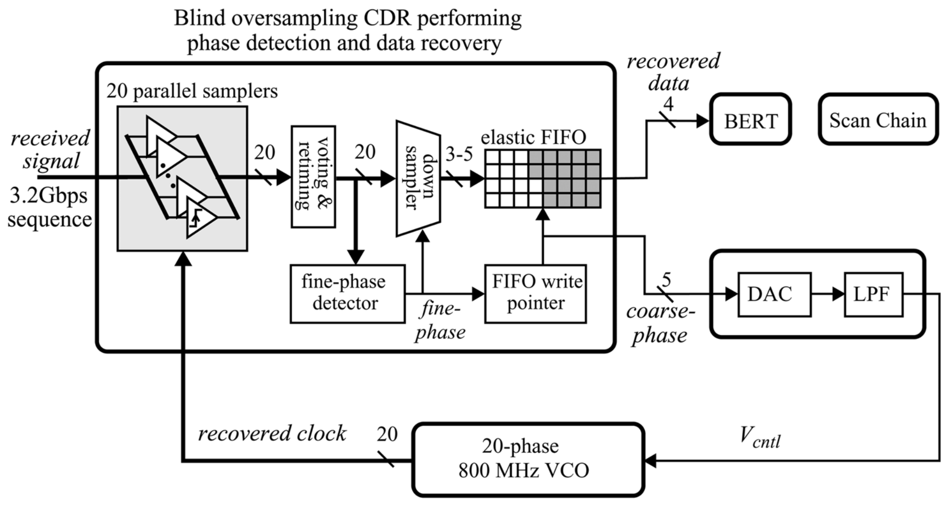


Fig. 8. Hybrid CDR design.

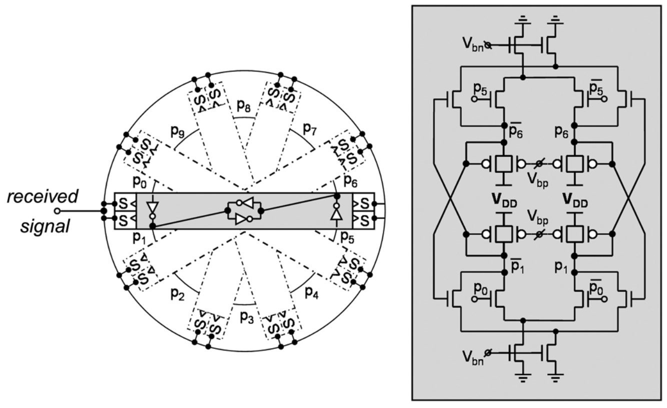


Fig. 9. VCO and samplers schematic.

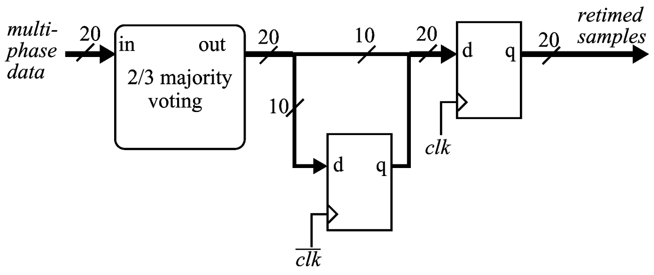


Fig. 10. Retiming and sample voting.

sample a 3.2 Gb/s sequence. Therefore, in one period of the 800 MHz clock, we collect a total of 20 samples, corresponding to 4 UI in the received data. These 20 samples are then aligned to a single clock edge by the retiming block and passed to the fine-phase detector. The fine phase is the intra-bit component (modulo 1 UI) of the phase difference between the clock embedded in the received data, and the recovered clock. The fine-phase detector uses the location of transitions within the window of 20 samples to determine the fine phase. The fine phase is then used by the down-sampler to pick the data bits among the 20 samples. Due to the changing fine phase, the downsampled data size from a 20-sample window can be between 3 and 5 bits, as explained later. This data is then written into an 8×4 elastic FIFO where up to a total of 32 UI peak–peak jitter (the FIFO size) is absorbed.

Since the occupied depth of the FIFO is an indication of the phase difference between embedded and recovered clocks, the FIFO write pointer can be used as a measure of the *coarse phase*. This coarse phase is measured in integer multiples of UI and complements the fine phase. Phase tracking in this CDR is implemented by passing the *FIFO write pointer* (coarse phase) to a 5-bit DAC, which is then low-pass filtered to provide the VCO control voltage. The loop is then closed by feeding the recovered clock to the blind-oversampling component, creating a hybrid CDR that is no longer blind, but tracks low-frequency jitter with higher tolerance.

Sections IV-A–G describe the detailed implementation of the various blocks in Fig. 8.

A. VCO/Samplers

Fig. 9(a) shows the circuit implementation of the 20-phase VCO using a 10-stage ring oscillator. Although the figure shows single-ended signals for clarity, all signals are differential. Small extra buffers cut halfway across the ring, creating sub-feedback loops that enable increased VCO frequency [9]. The CDR samplers are also integrated with the VCO, shown as the small blocks marked “S” on the periphery of the ring, eliminating the need to distribute a multi-phase clock. Once sampling has occurred, the remaining CDR blocks use only a single clock. Fig. 9(b) shows the transistor-level schematic of the shaded block in Fig. 9(a). The top and bottom halves of the schematic are mirror images of each other. In each half, an nMOS differential pair for a main VCO buffer and a sub-feedback buffer share common pMOS loads. The inputs of the sub-feedback differential pairs are connected to the outputs on the other side, while the inputs to the main VCO buffers are driven by the outputs of the previous stages.

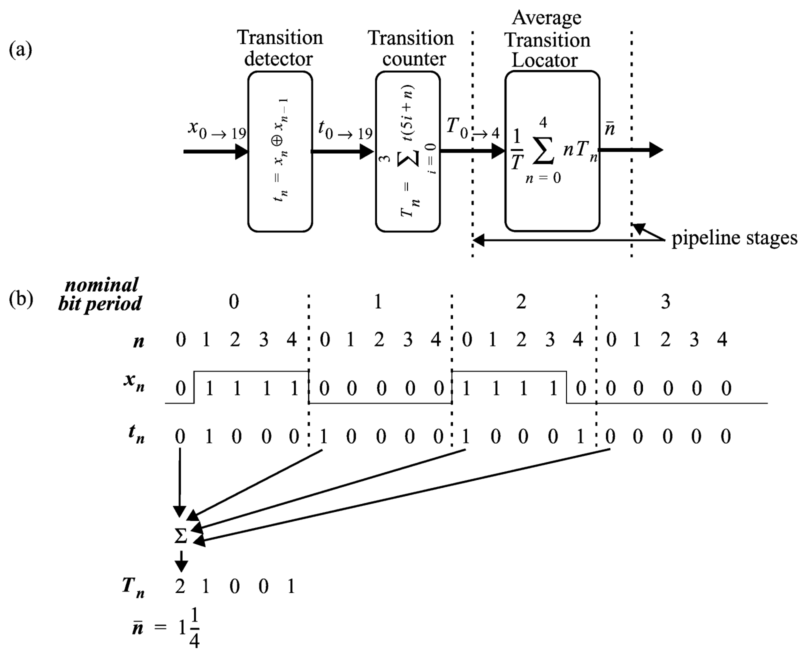


Fig. 11. Fine-phase detector. (a) Block diagram. (b) Example of data sequence.

B. Retiming and Voting

The 20 samples from the VCO samplers are aligned to 20 evenly-spaced clock phases. The retiming block, shown in Fig. 10, applies the samples to a voting block, discussed below, and then aligns the voted samples to a single clock phase (one of the 20), common to all other digital logic in the CDR.

The retiming block includes a 2/3 majority sample voting block which aims to reduce the effect of noise in the input samples. For each sample, the voting block outputs the value shared by the majority (two or more) of a three-sample window centered on that sample. The purpose of the voting block can best be illustrated with an example: While the sequence of samples "00001111" is unchanged after voting, the sequence "00010111" becomes "00001111" after voting. This change eliminates the extra transitions that result from noise near data transitions. These extra transitions can upset the fine-phase detector, and eliminating them improves the BER, as we will see later.

C. Fine-Phase Detector Basics

The fine-phase detector determines the intra-bit clock phase, or fine phase, based on data transitions within a window of 20 samples spanning four nominal bit periods. In general, these transitions can occur on any of five fine phases (0 to 4, representing 0/5 to 4/5 UI), consistent with a 5× oversampling rate. The fine-phase detector determines which fine phase within one window is most likely to have resulted in the distribution of transitions across the 5 fine phases. This fine phase is later used to downsample the data within that window.

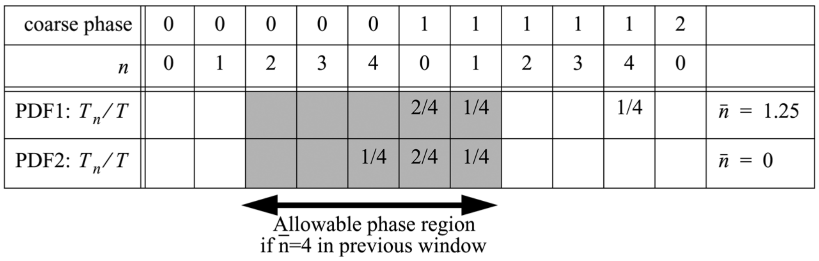
A block diagram of the fine-phase detector is shown in Fig. 11(a). The input to the fine-phase detector is the oversampled and retimed window of 20 samples, x_n. Each sample is XORed with its preceding sample to locate the data transitions, t_n. The number of transitions (0–4) occurring during each fine phase are then totalled to find the transition counts, T_n. The final block finds the fine phase, $\bar{n}$, which is essentially a weighted average of the transition phases, n, using the T_n as weights. If no transitions occur, the fine phase from the previous window is retained.

Fig. 11(b) shows an example sequence occurring during one window. On top, the figure shows the 20 sample-periods broken down into four nominal bit-periods, each having five samples with fine phases, n, of 0 to 4. Under this is the sequence of samples, x_n, and the transitions, t_n. The transitions occurring on each fine phase are then added to form the transition totals, T_n, below. This process is illustrated for T_0. The computation of $\bar{n}$ from these transition totals is described next.

As discussed in Section II-B, the 5× blind-oversampling CDR can track phase changes occurring at a rate of less than 2/5 UI between transitions. Since our system uses a $2^{31}-1$ PRBS sequence, the maximum runlength of the sequence is 31 bits. This results in at most 32 UI between transitions, occurring over eight windows (of 4 UI each). Over one window, this gives a maximum phase change of 2/5 UI divided by 8, or 1/20 UI. Because we measure the fine phase in discrete increments of 1/5 UI, this will usually not be measurable within one window, and as such can be approximated as constant over one window. As a result, we can treat each transition within one window as a measure of the same phase with an added component of random jitter. This allows us to construct a fine phase probability density function (PDF)[1] in the form of T_n/T, where T is the total number of transitions. The expected fine phase, or average transition phase, can be calculated as

$$\bar{n} = E[n] = \frac{1}{T}\sum_{n=0}^{4} nT_n. \tag{10}$$

[1]Technically, this should be called probability mass function (PMF) as it applies to a *discrete* random variable.



coarse phase	0	0	0	0	0	1	1	1	1	1	2	
n	0	1	2	3	4	0	1	2	3	4	0	
PDF1: T_n/T			▨	▨	▨	2/4	1/4			1/4		$\bar{n} = 1.25$
PDF2: T_n/T			▨	▨	1/4	2/4	1/4					$\bar{n} = 0$

Allowable phase region
if $\bar{n}$=4 in previous window

Fig. 12. Transition count PDFs and their ambiguity in fine phase due to phase-wrapping.

There is an ambiguity, however, in the direct application of (10) that can result in multiple values of $\bar{n}$ for the same set of T_n/T. This ambiguity is illustrated in Fig. 12 where two different PDFs are associated with the same set of T_n/T. In each PDF T_4/T has a fine phase of 4, although occurring at two different coarse phases. Blindly applying (10) to both PDFs produces $\bar{n} = 1.25$. However, as shown in the figure, knowledge of the coarse phase shows that $\bar{n} = 0$ is the correct result for PDF2.

The above ambiguity can be resolved by applying the constraint that the change in fine phase between two transitions cannot exceed 2/5 UI. This constraint restricts the composite phase of the T_n to a 1 UI contiguous region centered around the fine phase from the previous window. In the above example, if the previous fine phase was 4, this would be represented by the shaded region in Fig. 12. Given a set of T_n and the previous fine phase, this allows only one interpretation of the respective coarse phases. Only PDF2 in Fig. 12 satisfies this constraint. Once the correct PDF is constructed, its fine phases must be unwrapped to reflect their *relative* phases. In this example, this would mean treating the fine phases in the shaded region as $\{2, 3, 4, 5, 6\}$, not $\{2, 3, 4, 0, 1\}$. After unwrapping, we can apply (10) to determine $\bar{n}$.

While the transition detector and transition counter of Fig. 11(a) can be implemented in a straightforward fashion, a brute-force arithmetic approach to averaging and phase-unwrapping in the average transition locator results in a circuit with a delay in excess of the short cycle times required in a CDR operating in the Gb/s regime. To speed up the computation, we use a slightly different viewpoint of averaging, as outlined next.

D. Fine-Phase Detector Detailed Implementation

The implementation of the average transition detector is based on the observation that (10) can be rewritten as

$$\sum_{n=0}^{4} (n - \bar{n})T_n = 0. \tag{11}$$

Accordingly, if we define

$$f(\hat{n}) = \sum_{n=0}^{4} (n - \hat{n})T_n \tag{12}$$

then $f(\hat{n})$ is a decreasing function of $\hat{n}$ with a single zero at $\hat{n} = \bar{n}$. Therefore, if we decrease $\hat{n}$ from its maximum value, there will be an $\hat{n}$ for which the sign of $f(\hat{n})$ changes. By simultaneously calculating $f(\hat{n})$ *between* all possible integer values

$n-\hat{n}$	-3.5	-2.5	-1.5	-0.5	0.5	1.5	2.5	3.5
$g(n-\hat{n})$	-8	-4	-2	-1	1	2	4	8

Fig. 13. $g(n - \bar{n})$ as a function of .

of $\hat{n}$ (at 0.5, 1.5, 2.5, 3.5), we can identify $\bar{n}$ (rounded to the nearest integer value) as laying between the values of $\hat{n}$ for which $f(\hat{n})$ changes polarity. For example, if $f(1.5) = 1.5$ and $f(0.5) = -1.5$, then we can conclude that $\bar{n} = 1$.

Unfortunately, the evaluation of (12) for multiple $\hat{n}$ requires a large layout area, and results in an implementation that is still too slow for our application. However, we can achieve a simple and fast implementation by replacing $n - \hat{n}$ in (12) with a monotonic function, $g(n - \hat{n})$

$$f(\hat{n}) = \sum_{n=0}^{4} g(n - \hat{n})T_n \tag{13}$$

yet maintain similar properties. The function we use, illustrated in Fig. 13, increases by a factor of 2 on each side of $n - \hat{n} = 0$. This reduces the calculation complexity of (13) as all multiplications now reduce to simple bit-shifts and, as will be shown, partial results can be shared between the calculations for the various values of $\hat{n}$. This simplification produces almost identical results to (12) with a few minor differences that are discussed later.

The above substitution of $g(n - \hat{n})$ allows the calculation of $\bar{n}$ with a simple implementation. Fig. 14 shows an example of such an implementation for PDF2 in Fig. 12. It consists of five identical blocks, one for each phase in the allowable phase region such that the phase is unwrapped with previous fine phase ($\bar{n} = 4$) at the center. Each of these blocks simultaneously contributes a portion of the computation required to calculate $f(\hat{n})$ at ***all*** four intermediate values of $\hat{n}$.

Running up and down, partial sums from the previous block are multiplied by two using a bit shift, and then added to the T_n of the local block. The resulting new partial sum is then passed on to the next block, where the process is repeated. As a result, at any point a T_n from m stages away is multiplied by 2^m. Therefore, the partial sum at any point gives the contribution towards $f(\hat{n})$ from all the T_n on the side of the block from which that partial sum originated. To fully calculate $f(\hat{n})$, the partial sum running in one direction is subtracted from the partial sum from the other direction (shown by the value in brackets between partial sums), keeping only the most-significant bit (MSB) to indicate the polarity. The MSBs bounding each phase are then

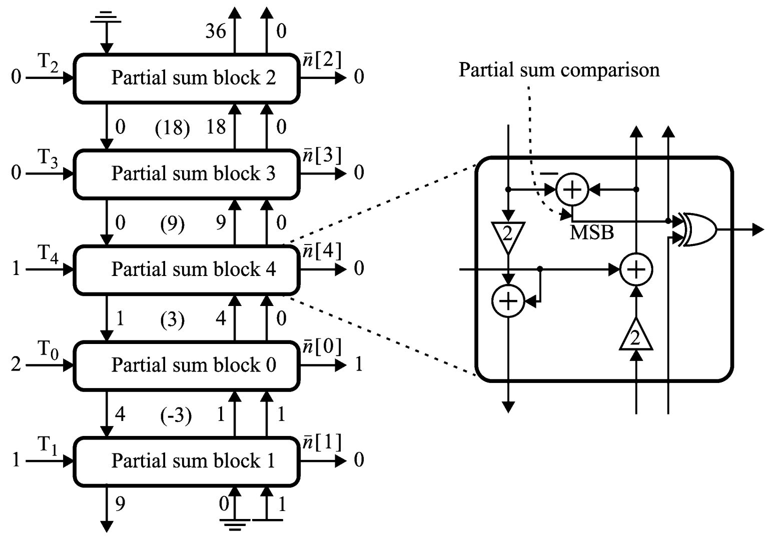


Fig. 14. Average transition locator implementation.

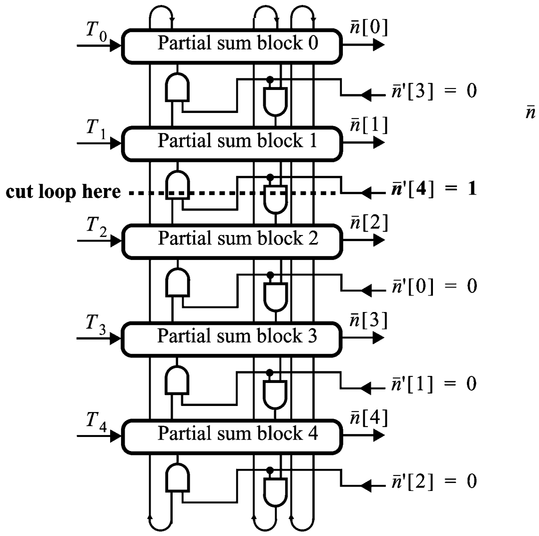


Fig. 15. Average transition locator with phase unwrapping.

XORed to generate an average fine phase output (one-hot encoded) where the polarity of $f(\hat{n})$ changes.

To complete the implementation, we must ensure that the previously discussed phase-unwrapping has been performed. This can easily be done as shown in Fig. 15 where the five stages in the phase detector are connected in a loop. This loop is cut between blocks 1/2 UI away from the previous fine phase output using AND gates driven by the previous one-hot fine phase. This cut creates a 1 UI contiguous region around the previous phase, as shown in Fig. 12(b). Once cut, the result is the 5-stage system of Fig. 14.

To determine the loss of accuracy of a fine-phase detector using $g(n - \hat{n})$ instead of $n - \hat{n}$, we used a behavioural simulation written in C. This simulation compared the fine phase outputs of phase-detectors using both methods, over all 126 possible unique combinations of one to four transitions. The results matched in all but eight cases. Of these mismatches, two were pathological transition counts that do not happen during normal operation. Another three cases were boundary cases where the fine phases lay exactly halfway between discrete fine phases and could be rounded to one side or the other with identical rounding errors. The final three cases represent a phase-detection error of 1/30 UI. For these three cases, the tracking ability of the blind oversampling drops from 2/5 UI to (2/5 − 1/30) UI between transitions. The reduction in tracking ability translates to a proportional decrease in the high-frequency jitter tolerance, due to the blind-oversampling component described in Section II-B. However, it has no effect on the low-frequency jitter tolerance where the FIFO is saturated and the blind-oversampler is no longer limited by the rate of phase-change. The loss at high-frequencies is a small price given the increased bit-rate permitted by the simplified implementation.

Despite its simple and fast architecture, the average transition locator shown in Fig. 15 is still the speed bottleneck of the CDR. The critical path starts at the register generating the old fine phase, $\bar{n}'$, then proceeds to the AND gates which cut the loop of partial-sum blocks. It then goes through these blocks and comes out as the new fine phase, $\bar{n}$, going back to the register. While the average transition locator is bracketed by pipeline stages, as shown in Fig. 11(a), the critical path described above cannot be pipelined, limiting the current implementation to a bit-rate of 3.5 Gb/s.

E. Downsampler

The downsampling block, shown in Fig. 16, selects the data bits amongst the 20 samples in a window, guided by the fine phase. Four 5–1 multiplexers then provide 5× downsampling, using the fine phase to pick the sample in the center of the data eye, which is 0.5 UI, or 2.5 samples, from the average transition location.

Under typical conditions, we downsample only 4 bits from the 20 samples. However, depending on the relationship between the sampling phases of the current and previous windows, there exist 6 cases (among a total of 25) for which we must pick an additional bit or drop a bit, ending up with 3 or 5 bits. The graph on the left side of Fig. 17 shows the downsampled

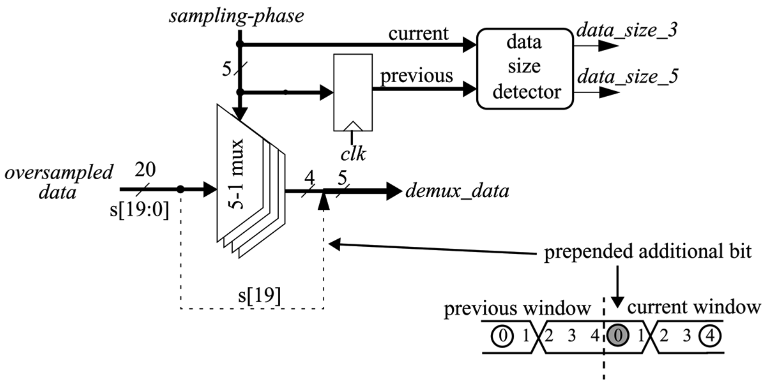


Fig. 16. Downsampler.

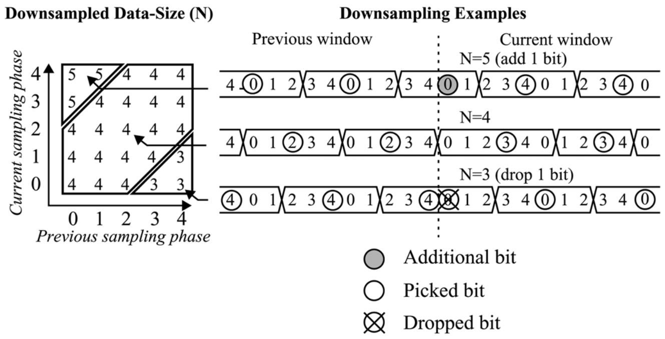


Fig. 17. Data-selection implementation.

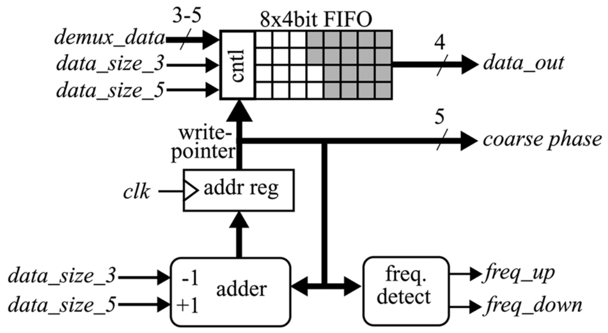


Fig. 18. Elastic FIFO block.

data-size as a function of the current and previous sampling phase. On the right side of Fig. 17 are three examples showing the different downsampled data sizes. The different sizes arise due to the sampling phase drifting across the boundary between sample windows. When the sampling phase drifts forward across this boundary, as shown in the bottom example, an oversampled data-bit straddling the window boundary is demultiplexed twice-once in each window. When this happens, we drop a bit. When the sampling phase drifts backward across this boundary, as shown in the top example, the bit straddling the window boundary is not demultiplexed in either window. In this case, we add a bit.

To deal with the additional-bit condition, the first bit in the 20-sample window is prepended to the downsampled data after the 5–1 multiplexer in Fig. 16, as shown by the dashed line bypassing the multiplexers. This 5-bit *demux_data* is then sent to the elastic FIFO. However the number of these bits used depends on the downsampled data size. When the data size is 4 bits (neither *data_size_3* nor *data_size_5* asserted), the 4 bits from the multiplexer are used and the prepended bit is discarded; When an additional bit results in a 5-bit data size, all 5 bits are used; When a bit is deleted, resulting in a 3-bit data size, both the prepended bit and following bit are discarded. The data size is determined based on the current and previous *sampling-phase* by the data-size detector of Fig. 16, which implements the mapping function shown in Fig. 17.

F. Elastic FIFO

The elastic FIFO block, shown in Fig. 18, accepts 3–5 bits from the downsampler block during each clock cycle, while outputing 4 bits. It has a depth of 32 bits, and hence is capable of absorbing up to 32 UI peak–peak phase difference between the received data and the local sampling clock.

In addition to this primary function, the elastic FIFO block also provides the coarse phase output for the CDR's loop filter. The FIFO write-pointer provides a measure of the phase offset, in UI, between the embedded clock and the local clock. As such, the FIFO write-pointer is used directly as a coarse phase output, used by the phase-tracking portion of the CDR. In addition, this coarse phase output is used for frequency detection during CDR start-up. During start-up, when the embedded and local clocks are at different frequencies, the constant phase shift will cause the FIFO to repeatedly overflow or underflow. If the two or more sequential overflows or underflows occur, then the frequency detector will output *frequency_up* or *frequency_down* pulses, respectively.

G. DAC/LPF

The coarse phase and frequency up/down signals control current sources connected to the CDR's loop filter as shown in Fig. 19. The coarse phase input (0–31) drives a 5-bit current DAC which is *RC* filtered to create the VCO control voltage, V_{cntl}.

The current output of DAC is $I_{\mathrm{DAC}} = (\text{coarse phase} - 15.5)I_{\mathrm{step}}$, where I_{step} is the current resolution. A coarse phase offset of 15.5 biases the DAC around a half-full (half-empty?)

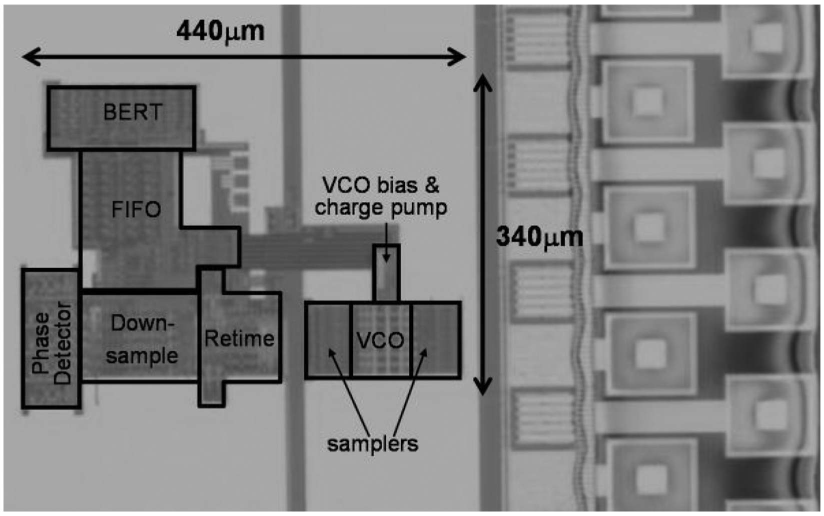


Fig. 20. Test chip die photo.

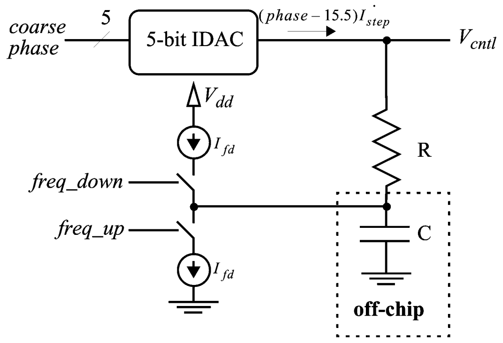


Fig. 19. DAC and loop filter.

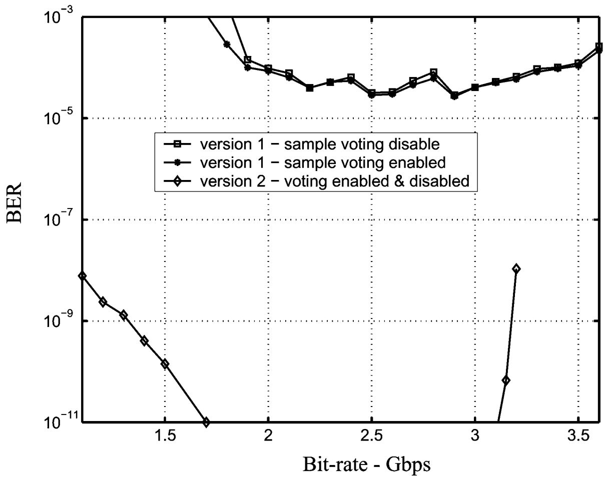


Fig. 21. Bit error rate versus bit-rate.

FIFO. Because one coarse phase step equals 1 UI, which equals 1/4 of a clock cycle, the resulting phase detector gain is $K_{\mathrm{PD}} = 2I_{\mathrm{step}}/\pi$. An off-chip current reference is used to set I_{step}.

The frequency up/down signals control switches which source or sink a current of I_{fd} directly onto the loop filter capacitor, providing quick frequency lock.

The loop dynamics of the CDR are those of a typical charge-pump PLL [8], characterized by (2). In this loop, the CDR's VCO was measured and found to have $K_{\mathrm{osc}} = 30$ Grad/s/V, and we selected $R = 200\ \Omega$, $C = 1.5$ nF and $I_{\mathrm{step}} = 1.2\ \mu$A to produce a system with a characteristic frequency, f_0, of 0.6 MHz and a quality factor of $Q = 0.85$.

V. Measured Results

A test chip was designed and fabricated in a 0.11 μm CMOS process to demonstrate the feasibility and performance of the CDR. Fig. 20 shows a die photo of the design, measuring 440 μm× 340 μm, excluding pads. Two iterations of this design were fabricated, as supply noise problems degraded the BER of the first version. The measured results of both iterations are discussed here.

Fig. 21 shows the bit-error rate of our CDR as a function of bit-rate. This figure shows the first design iteration CDR to be functional between 1.9 Gb/s and 3.5 Gb/s. The low-frequency limit is a result of the VCO tuning range, while simulation shows the high-frequency limit to be due to a critical path in the fine-phase detector. In the midband the average BER is 4×10^{-5}, far worse than commercially acceptable levels of 10^{-12}. While our measurement results show that the sample-voting scheme described in Section IV provides a 15% improvement in BER, these results still fail to meet commercial standards.

During testing, two observations led us to conclude that the source of the poor BER performance was supply noise. First, it was observed that the midband BER was roughly 10^{-3} with a 50 Ω load connected to the recovered clock output, and 4×10^{-5} without. As the loading of this clock output only directly affects the final drive stage in the chip's pad ring, we concluded that supply noise was the cause of the BER degradation. Our second observation related to the JTAG-like scan-chain that is used to configure the CDR, and to monitor the built-in BERT. In our first design iteration it was necessary to continuously shift out the BERT error-counter via the scan-chain to determine the BER. This BER measurement was found to be correlated with the frequency of the scan-chain clock. By lowering this frequency from 166 MHz to 500 kHz, we observed the BER to drop from

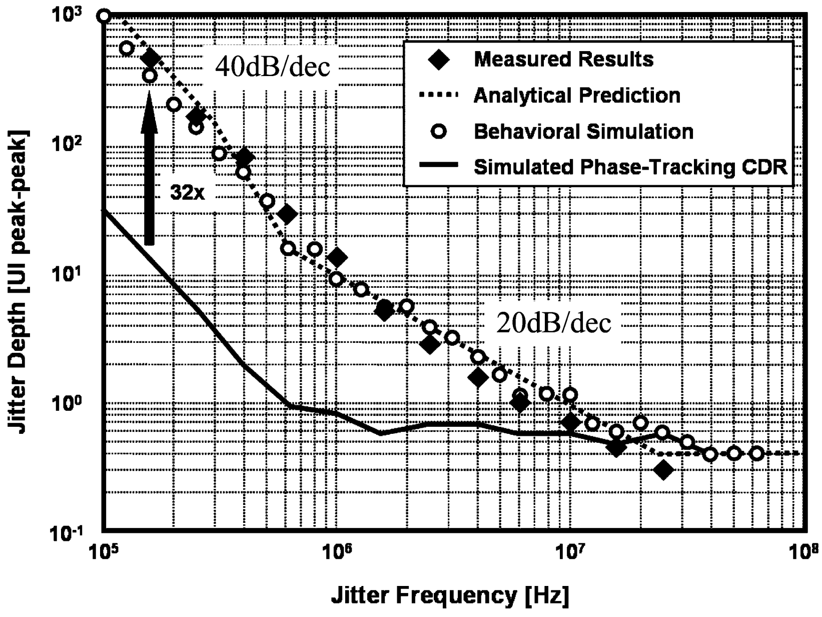


Fig. 22. Jitter tolerance results.

4×10^{-5} to 4×10^{-6}. This scan-chain snakes through the entire design, and we suspect supply noise from the scan-chain clock buffers and the chain's shift-registers as the cause of BER degradation in this case. The first design iteration results presented in Fig. 21 were obtained with an unloaded recovered clock output, and a scan-chain clock-rate of 166 MHz.

To address the supply noise issues described above, we fabricated a second iteration of our CDR design that was modified to reduce supply noise. To reduce the amount of generated supply noise we added an enable signal to the recovered clock output, and introduced a method to observe the BERT error-counter without clocking the scan-chain. To better suppress the remaining supply noise we increased the area of on-chip decoupling gate capacitance from 0.002 mm^2 to 1 mm^2, and increased the average metal width connecting the circuit to the global power-grid by roughly a factor of three. In addition to these design changes, we also redesigned the testbench PCB used to improve impedance matching in the transmission-lines, as well as reducing the total length of coax-cable between the PCB and the PRBS data-source from over 3′ to 6″.

The measured results of the second design iteration are also shown in Fig. 21. The low- and high-frequency limits are still the same as those of the first iteration, however, the BER in the midband improved to better than 10^{-11}, which was the limit of our measurement capability. In this version of the design, there was no observable difference in BER with the sample-voting scheme enabled or disabled. This suggests that in the first design iteration the voting scheme was compensating for samples that had been perturbed by supply noise.

Fig. 22 shows measured results for the jitter tolerance of the first CDR design iteration, and compares them against behavioural-simulation results and analytical predictions, all in close agreement. The measured results from the second iteration are almost identical. The behavioural simulations were performed in Verilog, repeated at each frequency for 10^5 cycles with increasing jitter amplitude until an error was observed. The analytical prediction is based on an allowable phase change of 2/5 UI over a run length of 32 ($2^{31}-1$ PRBS), a FIFO of 32 bits, and a loop filter with an *RC* time constant of 300 ns. With these parameters, the corner frequency of the jitter tolerance of the phase-tracking CDR exceeds that of the oversampling counterpart. This produces a -60 dB/decade region where the jitter tolerance of the phase tracking component is enhanced by the oversampling component with a benefit that increases towards 32× with decreasing frequency. Below this frequency range the jitter tolerance is that of a phase-tracking CDR with the full 32× enhancement, while above this range the jitter tolerance is determined only by the oversampling CDR. All results are measured at 2.4 Gb/s due to the limited tuning bandwidth of the VCO used to frequency-modulate the clock. The CDR, however, is fully operational from 1.9 to 3.5 Gb/s.

To compare the jitter tolerance of the hybrid CDR against that of a purely phase-tracking CDR, we simulated the latter by reducing the FIFO size to 1-bit. The result confirms that the hybrid CDR has a low-frequency jitter tolerance that is 32 times that of the phase-tracking CDR, as shown in Fig. 22.

A. Power Consumption

The power consumption of the CDR was measured to be 55 mW from the analog supply and 60 mW from the digital supply, for a total of 115 mW from a 1.2 V supply. The analog supply feeds the VCO, samplers, and current DACs in the loop filter. The digital supply feeds the remaining circuits. The digital power is primarily due to the sense-amp flip-flops and the clock that drives them. Approximately 1/3 of this digital power is due to the BERT, which is required only for test purposes. The remaining digital power could be reduced still further through the selective use of lower-power digital logic, as the current design exclusively uses sense-amp flip-flops and a form of differential pass-gate logic [10].

TABLE I
SUMMARY OF DESIGN CHARACTERISTICS

Maximum data rate	3.2Gbps
Process	0.11μm CMOS
Design size	440μm x 340μm
Supply voltage	1.2V
Power consumption	115mW

VI. Conclusion

This paper described the design of a CDR that combines the advantages of blind-oversampling with those of traditional phase-tracking CDRs. The recovered clock in a traditional phase-tracking CDR must track the embedded clock within 1/2 UI; otherwise the CDR will recover data from the wrong symbol period. By embedding a blind-oversampling CDR within a phase-tracking CDR, the recovered clock may deviate in phase from the embedded clock by the jitter tolerance of the blind-oversampling CDR. The recovered clock from the phase-tracking CDR is then used to clock the blind-oversampling CDR, resulting in a semi-blind oversampling CDR having a jitter tolerance equal to the product of the jitter tolerances of phase-tracking and blind-oversampling CDRs. At low jitter frequencies, the jitter tolerance of the blind-oversampling CDR is equal to the size of its FIFO-32 in our implementation. Measured results of our fabricated design show the low-frequency jitter tolerance of the CDR to be 32 times that of a conventional phase-tracking CDR. The characteristics of the fabricated design are summarized in Table I.

References

[1] M. Kokubo *et al.*, "Spread-Spectrum clock generator for serial ATA using fractional PLL controlled by $\Delta\Sigma$ modulator with level shifter," in *IEEE ISSCC 2005 Dig. Tech. Papers*, Feb. 2005, pp. 160–161.

[2] M. Hsieh and G. Sobelman, "Clock and data recovery with adaptive loop gain for spread spectrum serdes applications," in *Proc. IEEE ISCAS 2005*, May 2005, pp. 4883–4886.

[3] L. M. DeVito, "A versatile clock recovery architecture and monolithic implementation," in *Monolothic Phase-Locked Loops and Clock Recovery Circuits*, B. Razavi, Ed. New York: IEEE Press, 1996, pp. 405–430.

[4] B. Razavi, *Monolithic Phase-Locked Loops and Clock Recovery Circuits*. New York: IEEE Press, 1996.

[5] T. Lee and J. Bulzacchelli, "A 155-MHz clock recovery delay- and phase-locked loop," *IEEE J. Solid-State Circuits*, vol. 27, no. 12, pp. 1736–1746, Dec. 1992.

[6] J. Kim and D. K. Jeong, "Multi-gigabit-rate clock and data recovery based on blind oversampling," *IEEE Commun.*, pp. 68–74, Dec. 2003.

[7] M. van Ierssel, A. Sheikholeslami, H. Tamura, and W. W. Walker, "A 3.2 Gb/s semi-blind-oversampling CDR," in *IEEE ISSCC 2006 Dig. Tech. Papers*, Feb. 2006, pp. 334–335.

[8] D. Johns and K. Martin, *Analog Integrated Circuit Design*. New York: Wiley, 1997.

[9] L. Sun *et al.*, "A 1.25-GHz 0.35-μm monolithic CMOS PLL based on a multiphase ring oscillator," *IEEE J. Solid-State Circuits*, vol. 36, no. 6, pp. 910–916, Jun. 2001.

[10] F. Lai and W. Hwang, "Design and implementation of differential cascode voltage switch with pass-gate (DCVSPG) logic for high-performance digital systems," *IEEE J. Solid-State Circuits*, vol. 32, no. 4, pp. 563–573, Apr. 1997.

Marcus van Ierssel (S'92–M'07) received the B.A.Sc. degree in electrical engineering from the Division of Engineering Science in 1992, the M.A.Sc. degree in the Department of Electrical and Computer Engineering in 1995, and the Ph.D. degree in the Department of Electrical and Computer Engineering in 2007, all at the University of Toronto, Toronto, ON, Canada.

He is currently working at Snowbush Microelectronics as an analog IC designer, focusing on PLL and SERDES design.

Ali Sheikholeslami (S'98–M'99–SM'02) received the B.Sc. degree from Shiraz University, Shiraz, Iran, in 1990, and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, ON, Canada, in 1994 and 1999, respectively, all in electrical and computer engineering.

In 1999, he joined the Department of Electrical and Computer Engineering, University of Toronto, where he is currently an Associate Professor. His research interests are in the areas of analog and digital integrated circuits, high-speed signaling, and VLSI memory design (including SRAM, DRAM, CAM, and FeRAM). He has collaborated with industry on various VLSI design research in the past few years, including work with Nortel and Mosaid, Canada, and with Fujitsu Labs. He spent his 2005–2006 research sabbatical year with Fujitsu Labs of Japan and Fujitsu Labs of America. He presently supervises two active research groups in the areas of high-speed signaling and VLSI memories. He has co-authored several journal and conference papers (in both areas), in addition to three U.S. patents on VLSI memories.

Dr. Sheikholeslami served on the Memory Subcommittee of the IEEE International Solid-State Circuits Conference (ISSCC) from 2001 to 2004, and on the Technology Directions Subcommittee of the same conference from 2002 to 2005. He presented a tutorial on ferroelectric memory design at the ISSCC 2002. He was the program chair for the 34th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2004) held in Toronto, Canada. He is a registered professional engineer in the province of Ontario, Canada. He received the Best Professor of the Year Award in 2000, 2002, and 2005 by the popular vote of the undergraduate students in the Department of Electrical and Computer Engineering, University of Toronto. In 2006, he received the Early Career Teaching Award in recognition of his "superb accomplishment in teaching" from the Faculty of Applied Science and Engineering at the University of Toronto.

Hirotaka Tamura received the B.S., M.S., and Ph.D. degrees in electronic engineering from Tokyo University, Tokyo, Japan, in 1977, 1979, and 1982, respectively.

In 1982, he joined Fujitsu Laboratories, Ltd., Kawasaki, Japan, where he was engaged in research on Josephson devices and other exploratory devices. In 1995, he moved into the area of CMOS circuit design. After working on multi-gigabit DRAMs and ferroelectric nonvolatile memories, he got involved in CMOS high-speed signaling. His current interests cover the circuit topology and architecture of high-speed CMOS interfaces.

William W. Walker (M'79) received the A.B. degree in physics and applied mathematics in 1976, and the M.S.E.E. in 1978, both from the University of California at Berkeley.

From 1978 to 1983, he was a Staff Engineer at IBM Corporation, in East Fishkill, NY, and Burlington, VT, where he was involved in the development of the LDD MOS transistor. From 1984 to 1991 he was a Senior Engineer at Integrated CMOS Systems, Inc., Sunnyvale, CA. From 1991 to 2000, he was an Engineering Manager at Hal Computer Systems, Inc., Campbell, CA, where his group developed CAMs, SRAMs, PLLs, and Register Files for the first 64-bit SPARC microprocessors. Since 2000, he has been with Fujitsu Laboratories of America, where he is currently Vice President in charge of the Circuits and Devices Innovation Group. His research interests include high-speed and low-power digital circuits for microprocessors, millimeter-wave CMOS, and high-speed wireline CMOS circuits for computer backplanes and optical communications.