A Fast-Lock, Jitter Filtering All-Digital DLL Based Burst-Mode Memory Interface

Masum Hossain, Farrukh Aquil, Pak Shing Chau, *Member, IEEE*, Brian Tsang, Phuong Le, Jason Wei, Teva Stone, Barry Daly, *Member, IEEE*, Chanh Tran, John C. Eble, *Member, IEEE*, Kurt Knorpp, and Jared L. Zerbe

***Abstract*—A 800 Mb/s to 3.2 Gb/s memory interface is designed that achieves 30% improved energy efficiency by eliminating idle mode power completely. The link is similar to a standard DDR architecture with the addition of a fast-lock DLL on the memory side that wakes up from 0 mW and locks within 3 clock cycles consuming 24 mW with residual timing error less than 33 mUI. Following initial lock, the DLL operates in a closed loop to compensate for V,T drift consuming 6 mW @ 1.6 GHz including a replica buffer. By incorporating an injection locked oscillator inside the loop, the DLL provides PLL like high frequency input jitter filtering, and corrects ±10% DCD without an additional duty cycle correction loop.**

***Index Terms*—Burst mode, digital DLL, memory, injection locking, fast locking, TDC.**

I. Introduction

PORTABLE devices with wireless connectivity are today's consumer mobile computing platform. Demand for higher processing power in such a device is mainly driven by mobile apps to support consumer convenience, connectivity such as social networking, security and productivity. With battery life already one of the significant challenges in this environment, power requirements in hand held devices have already outpaced the improvements in battery technology, increasing the need for energy efficiency. One major source of inefficiency in existing DDR architectures is the usage of a DLL in DRAM. The main purpose of the DRAM DLL is to compensate the skew introduced by the clock distribution network such that strobe (DQS) and data (DQ) are aligned to the global clock (CK) signal at the output pin. On the controller side the strobe is then shifted by 1/2 unit interval (UI) to sample the received data signal and is then synchronously moved into the internal clock domain. Satisfying this timing relationship over process, supply voltage and temperature variation allows the link to meet critical I/O timing. Since this timing relationship between DQ, DQS and CK needs to be satisfied only during read operations, the DLL can be potentially powered down during other modes of operations such as pre-charge, active stand by etc. Since a DRAM spends significant amount of time in these idle interface modes, keeping the DLL 'ON' causes significant power inefficiency in the system. Mobile centric LPDDR solutions are attempting to improve this situation by completely removing the DLL from DRAM. However, this also leaves uncompensated PVT variation on the DRAM side stressing timing margins at high data rates. As a result LPDDR solutions are falling behind in maximum achievable bandwidth compared to standard DDR solutions.

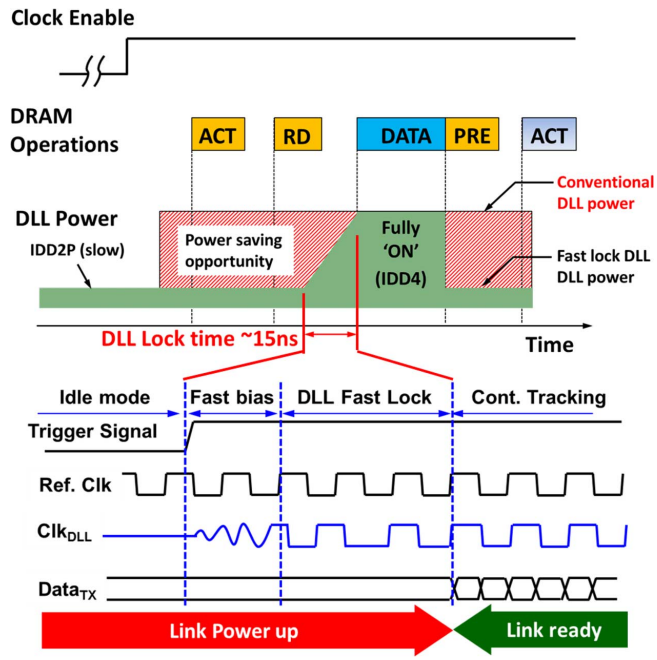


Fig. 1. CAS/RAS based fast lock DLL operation and power consumption during different modes of DRAM. Breakdown of DLL on time with wakeup and lock sequence.

An alternative approach is to keep the DLL 'ON' only during 'read' and keep it powered down in all other modes (Fig. 1). An active command (ACT) or a read command (RD) can be used as a trigger to wake up the DLL. To avoid a performance penalty, the triggering-read operation should not add any extra latency. Therefore, the DLL needs to wake up from nearly 0 mW and lock within 10 to 15 ns, preferably with residual error less than 1/32 UI. Based on the study reported in [1], for 10%

Manuscript received August 21, 2013; revised November 03, 2013; accepted December 05, 2013. Date of publication February 04, 2014; date of current version March 24, 2014. This paper was approved by Guest Editor Hideyuki Kabuo.

M. Hossain is with Rambus Inc., Sunnyvale, CA 94089 USA, and also with the University of Alberta, Edmonton, Alberta T6G 2R3, Canada (e-mail: masum@ualberta.ca).

F. Aquil, P. S. Chau, B. Tsang, P. Le, J. Wei, T. Stone, B. Daly, C. Tran, J. C. Eble, K. Knorpp, and J. L. Zerbe are with Rambus Inc., Sunnyvale, CA 94089 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2013.2297403

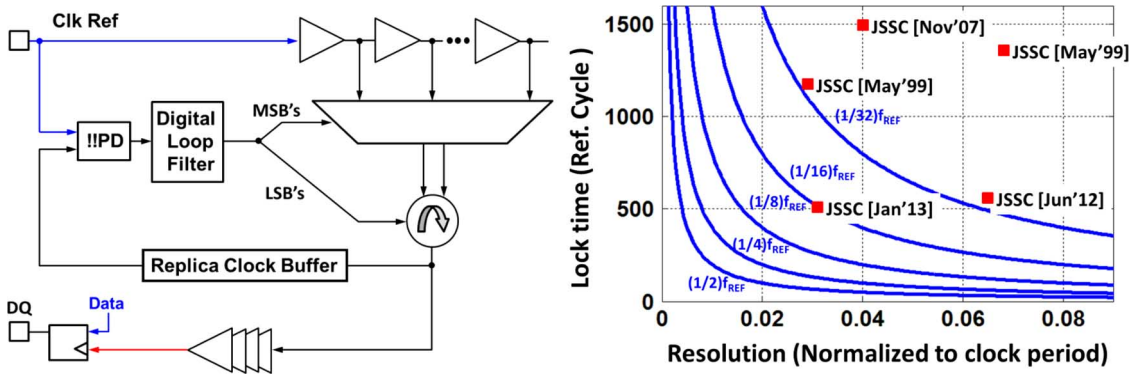


Fig. 2. Digital version of the feedback DLL described in [1]. Lock time vs.resolution tradeoff for different digital clock frequency.

TABLE I
DIFFERENT DRAM STATES FOR 40% AND 10% CPU UTILIZATION AND POTENTIAL SAVINGS WITH FAST LOCK DLL

DRAM Power State	% Time spent for diff. CPU utilization 40%	10% [1]	Savings with Fast lock DLL
IDD0,IDD1 (Active power)	0.1	0.7	80 %
IDD2P (Power down)	**41**	**50.46**	**80%**
IDD2N (Power down)	**20**	**9.26**	**80%**
IDD3P (Standby)	2.5	5.28	0%
IDD3N (Standby)	**22**	**5.56**	**80%**
IDD4R (Read)	0.6	0.14	0%
IDD4W (write)	1.0	0.24	0%
IDD6 (self refresh)	10	27.47	0%

CPU utilization, power down mode $(\mathrm{IDD_{2P}})$ constitutes more than 50% of its operating time, making the case for fast lock DLL quite promising. Note that in this case 'CPU utilization ratio' is defined as percentage of time CPU is not idle. Potential power savings at different modes of DRAM operation are summarized in Table I. For a 40% CPU utilization case, 30% improvement in power efficiency can be achieved by simply eliminating 'power down' and 'standby' mode DLL current. However, such fast lock and fine resolution DLL specifications are not achievable from a standard architecture; therefore, significant improvement in the DLL is needed. Based on this motivation the paper is organized in the following way: Section II introduces a hybrid DLL approach based on a brief review of the prior art, Section III explains different fast lock features of the DLL, and Section IV provides theoretical analysis with simulation results for different noise transfer functions of the proposed architecture in continuous mode of operation. Implementation and measured results are provided in Section V.

II. A HYBRID DLL APPROACH

Most standard DRAMs use closed loop DLLs. A replica of the actual buffer is inserted in the feedback path of the DLL. While locked, the VCDL's delay $(\mathrm{T_{VCDL}})$ contribution complements the buffer delay $(\mathrm{T_{buffer}})$ to complete a full cycle $(\mathrm{T_{REF}})$; therefore, the timing relationship can be mathematically described as: $\mathrm{T_{buffer}} + \mathrm{T_{VCDL}} = \mathrm{NT_{REF}}$, where N is an integer number starting from 1. Early designs used an analog implementation [2] and subsequently were replaced with digital versions [3] for better design portability, verification and smaller foot print. Note that these advantages come at the cost of additional quantization jitter due to the digital nature of the phase detector. Assuming the DLL does not lock in a hierarchical way [3], the lock time of this loop $(\mathrm{T_{lock}})$ is mainly determined by phase update rate and phase step size: $\mathrm{T_{lock}} = \mathrm{T_{REF}}/(\mathrm{f_{dig}} * \Delta\Phi)$. Phase update rate is process dependent, usually set by the maximum frequency $(\mathrm{f_{dig}})$ at which the digital loop filter can be synthesized meeting the timing requirement over PVT. On the other hand phase step size $(\Delta\Phi)$ is set by the dithering jitter limit of the system. Therefore, the jitter requirement sets the lock time; it usually requires $500+$ cycles to achieve low residual jitter (1/32 UI or 1/64 $\mathrm{T_{REF}}$ for DDR link). Although this lock time is sufficient to meet the present DDR4 specifications [4], it becomes exceedingly difficult to lock within the CAS/RAS latency even with an aggressive digital clock rate (Fig. 2).

A synchronous mirror delay (SMD) based approach provides significantly faster lock time by employing a time to digital converter (TDC). As shown in Fig. 3, the TDC directly measures the delay of the clock distribution buffer in the form of a code word 'D' by sampling the reference clock with N phases generated from an N stage delay line. Ideally, the TDC takes the buffer delay and one clock cycle to generate the TDC code, thereby achieving lock within 3 to 5 clock cycles. The generated TDC code is used to select the nth stage output from N delay stages such that $\mathrm{n}(\Delta\mathrm{T}) = \mathrm{T_{REF}} - \mathrm{T_{buffer}}$. Here, $\Delta\mathrm{T}$ is the resolution of the TDC defined as $\Delta\mathrm{T} = \mathrm{T_{REF}}/\mathrm{N}$. Therefore, the required resolution is achieved by choosing a proportional number of stages at the cost of higher power consumption. Compared to regular feedback type DLLs, power consumption increases in SMDs by more than $10\times$ to achieve a similar resolution (Fig. 3). This penalty becomes even higher as the delay line is used twice to implement the 'mirror' effect. Since most

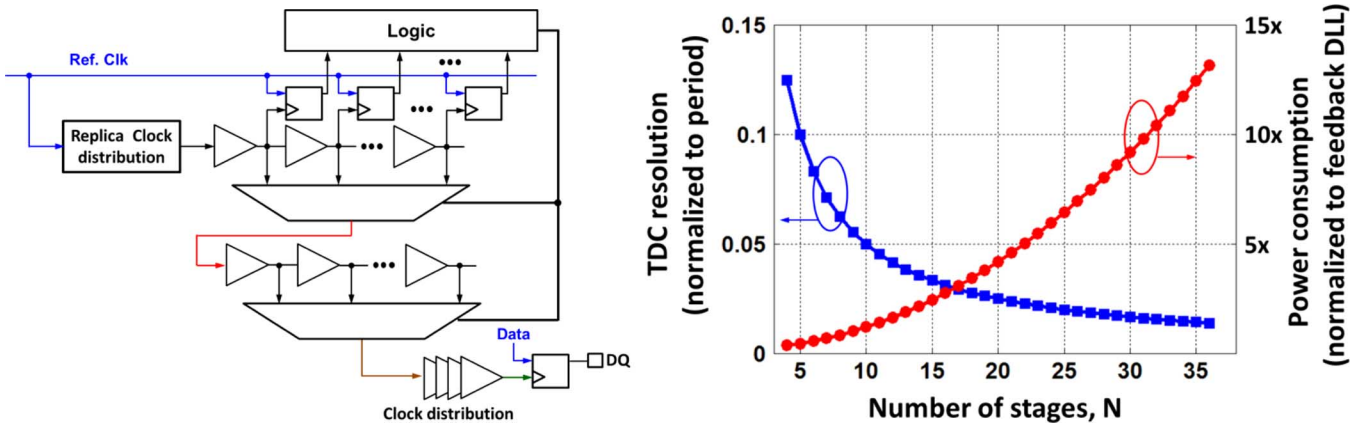


Fig. 3. Synchronous mirror delay structure as in [4] with power and resolution tradeoff.

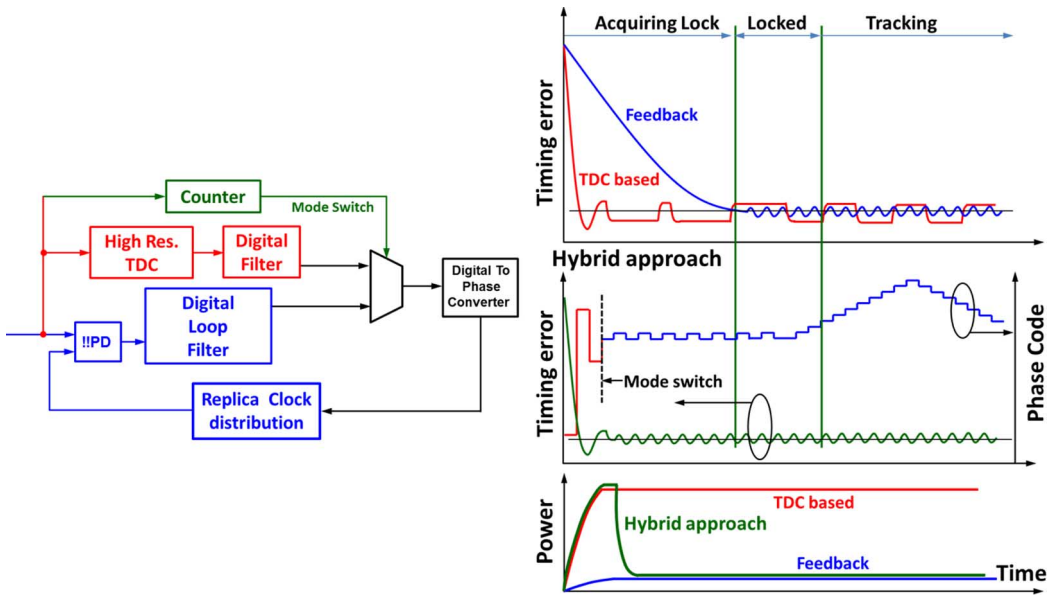


Fig. 4. A hybrid DLL with conceptual lock time and power consumption.

of these power consuming blocks are directly in the clock path, the architecture as described in [5] does not allow us to turn off power consuming devices after initial lock to save power. SMDs are mostly avoided in standard DDR interfaces despite their impressive lock time [6] due to prohibitive power consumption and often incorrect locking when the cycle time becomes shorter than buffer delay. To summarize, existing closed loop DLLs suffer from long lock time, and TDC based open loop solutions can lock fast but consume significantly higher power, thereby negating any power saving opportunity of the CAS/RAS triggered architecture.

Based on the above discussion, a rational approach leads to a hybrid architecture that wakes up in 'fast lock' mode, and once the output phase is locked then the DLL operates in a lower power 'continuous tracking' mode (Fig. 4). Such an approach has been explored in [7] where a coarse TDC is used to obtain the coarse delay setting in two clock cycles and then the analog loop takes another 10 clock cycles to achieve phase locking. Therefore, the total time needed for the phase locked clock to appear on the 'DQS' pin is $2T_{buffer} + 12\ T_{REF}$. Although this lock time exceeds the 15 ns target, the lock time is significantly better than a conventional approach. Based on the similar hybrid concept, the proposed architecture uses several techniques to meet the design target [8]. First, a two-step TDC is used to improve resolution such that the DLL can achieve locking in 3 reference clock cycles with residual error less than 1/32 UI. Second, to achieve power savings the DLL uses a fast bias technique [9] to wake up from a 0 mW idle-state (excluding leakage). In addition, the fast lock and continuous tracking loops are designed to be isolated and independent such that power consuming parts can be turned off to bring down the continuous mode power to 6 mW. In fast lock mode when high resolution TDC is employed, the DLL consumes significantly higher power ($\sim$24 mW). However, the DLL operates in this fast lock mode for only a short period of time ($\sim$15 ns), and the remaining time of the read mode the DLL operates in lower power continuous mode. Therefore, average power remains very close to 6 mW for a practical burst length during read operation. Third, a fully digital solution allows simpler mode switching—the fast lock mode generates a 6-bit phase code to deskew the DLL

output clock and align it with the reference clock. Then the DLL is switched over to a continuous tracking mode where the generated TDC code is used as an initial phase code for the phase accumulator. The DLL continues to operate tracking V, T variation by incrementing and decrementing the accumulator output by one phase step at a time. Unlike the analog voltage controlled delay unit described in [6], a phase rotating digital to phase converter allows infinite capture range. Compared to an analog or mixed-signal hybrid solution, the all-digital implementation enables simpler mode switch and reduces design and verification time. However, all-digital solutions also suffer from additional jitter sources such as quantization noise and power supply induced jitter (PSIJ). The proposed solution addresses these issues as described in Section IV. Finally, a duty cycle correction scheme is also used that is compatible with the 15 ns wakeup time.

III. Hybrid DLL Design

The main components designed to enable fast lock operation are the TDC, the code conversion logic and the digital-to-phase converter. The continuous tracking loop includes a bang-bang phase detector, digital loop filter and digital-to-phase converter. Note that the digital-to-phase converter is shared by both modes to reduce area overhead and achieve a glitch-less mode switch.

A. Two-Step TDC

A straightforward design of a TDC with $(1/64)\mathrm{T_{REF}}$ resolution would require a 32 stage delay line with 9.8 ps delay/stage and 64 samplers at 3.2 Gb/s data rate. This delay/stage requirement exceeds the FO4 delay in the desired 40 nm LP CMOS process and consumes more than 50 mW. To reduce power consumption, a two-step TDC is used (Fig. 5). First, the reference clock is passed through a coarse TDC implemented with an 8 stage differential delay line providing 39 ps resolution. A 16-bit raw output is generated from this TDC and then converted to a 4-bit binary output used to select two adjacent phases that bracket the reference clock. These two phases are then blended together to generate four phases with 9.8 ps spacing. The phase blending approach allows us to achieve targeted resolution without over stressing the FO4 delay of the process. Using these finely spaced phases, the reference clock is then re-sampled to generate a 4-bit raw output that is eventually converted to a 2-bit binary output. The 4-bit coarse and 2-bit fine outputs are combined as MSBs and LSBs to generate a final 6-bit TDC output. The two-step TDC solution requires a total of 20 samplers, 8 current starved buffers for phase blending, and the 8 stage delay line (39 ps/stage) for the coarse TDC. This relaxed delay/stage requirement and simpler implementation translates to $2.2\times$ improvement in TDC power consumption at the cost of one extra reference cycle in TDC conversion time. Simulated DNL of the phase blender in less than 0.25 LSB over different process corners, and including mismatch it is still better than 0.5 LSB. The delay in the phase blender must be replicated in the reference path using a dummy version as shown in the Fig. 5.

B. Direct Complementary Code Mapping

The code generated from the TDC represents the buffer delay by indicating the nth phase code out of N phase codes per clock cycle and then multiplying by $\Delta\mathrm{T}$; i.e., $\mathrm{T_{buffer}} = \mathrm{n}(\Delta\mathrm{T})$. Here, $\Delta\mathrm{T}$ is the resolution of the TDC defined as $\Delta\mathrm{T} = \mathrm{T_{REF}/N}$. This code needs to be converted to its complementary form such that the converted code represents the delay $(\mathrm{N}-\mathrm{n})\Delta\mathrm{T}$. Complementary code conversion can be implemented by subtracting 6 bit TDC code from the phase code corresponding to complete cycle, $\mathrm{T_{REF}} = 2^6 - 1 = 63$. As a result, the delay from CK to DQS can be written as

$$(\mathrm{N}-\mathrm{n})\Delta\mathrm{T} + \mathrm{T_{buffer}} = \mathrm{N}(\Delta\mathrm{T}) = \mathrm{T_{REF}} \tag{1}$$

Therefore the clock at DQS is phase locked with the reference at CK. One possible implementation of this concept is shown in Fig. 6(a). It is useful to identify the three components of lock time. First, the time required for the TDC to generate a valid code is $\mathrm{T_{buffer}} + 3\mathrm{T_{REF}}$, where $\mathrm{T_{buffer}}$ is the delay time for the 1st edge to appear at the output of the replica buffer and then the coarse and fine TDC each take one cycle to generate the TDC code with the code being held in the following cycle. Second, the coarse and fine codes are combined and then converted to a complementary form—this conversion takes 2 to 3 clock cycles. Finally, the locked edge appears at the output after $\mathrm{T_{buffer}}$ propagation delay. Adding all these components, the total time required for a valid clock edge to appear at DQS is $2\mathrm{T_{buffer}} + 6\mathrm{T_{REF}}$.

Clock distribution delay $\mathrm{T_{buffer}}$ depends on many parameters that cannot be affected by the DLL design, including physical length of the interface. However, the remaining part of the delay, especially code conversion latency, can be eliminated by modifying the phase interpolation architecture as shown in Fig. 6(b). In the conventional approach, the input to the DLL remains fixed, whereas the output is selected from the available phases at the delay line output. In the proposed approach, the output is fixed at the end of the delay line whereas the input point is variable. The delay through a conventional phase interpolator can be expressed as: $\alpha T_n + \beta T_{n+1}$, where α and β are the weighting factors of n-th and $(n+1)$th stage's output respectively. When the same code is applied to the modified structure as shown in Fig. 6(b) the total delay through the phase interpolator and clock distribution buffer can be written as:

$$\mathrm{T_{DQS}} = \alpha\mathrm{T_{P-1}} + \beta\mathrm{T_P} + \mathrm{T_{buffer}} \tag{2}$$

Note that in this case selecting the $(n+1)$th stage translates to a different delay: $\mathrm{T_P} = \mathrm{T_{REF}} - \mathrm{T_{n+1}}$. Therefore, mapping the delay in terms of $\mathrm{T_n}$ yields

$$\mathrm{T_{DQS}} = \alpha(\mathrm{T_{REF}} - \mathrm{T_n}) + \beta(\mathrm{T_{REF}} - \mathrm{T_{n+1}}) + \mathrm{T_{buffer}} \tag{3}$$

Since $\alpha + \beta = 1$, and α and β are the TDC codes that represent buffer delay, $\alpha\mathrm{T_N} + \beta\mathrm{T_{N+1}} = \mathrm{T_{buffer}}$. This relationship simplifies the delay expression given in (3):

$$\mathrm{T_{DQS}} = \mathrm{T_{REF}} - (\alpha\mathrm{T_N} + \beta\mathrm{T_{N+1}}) + \mathrm{T_{buffer}} = \mathrm{T_{REF}} \tag{4}$$

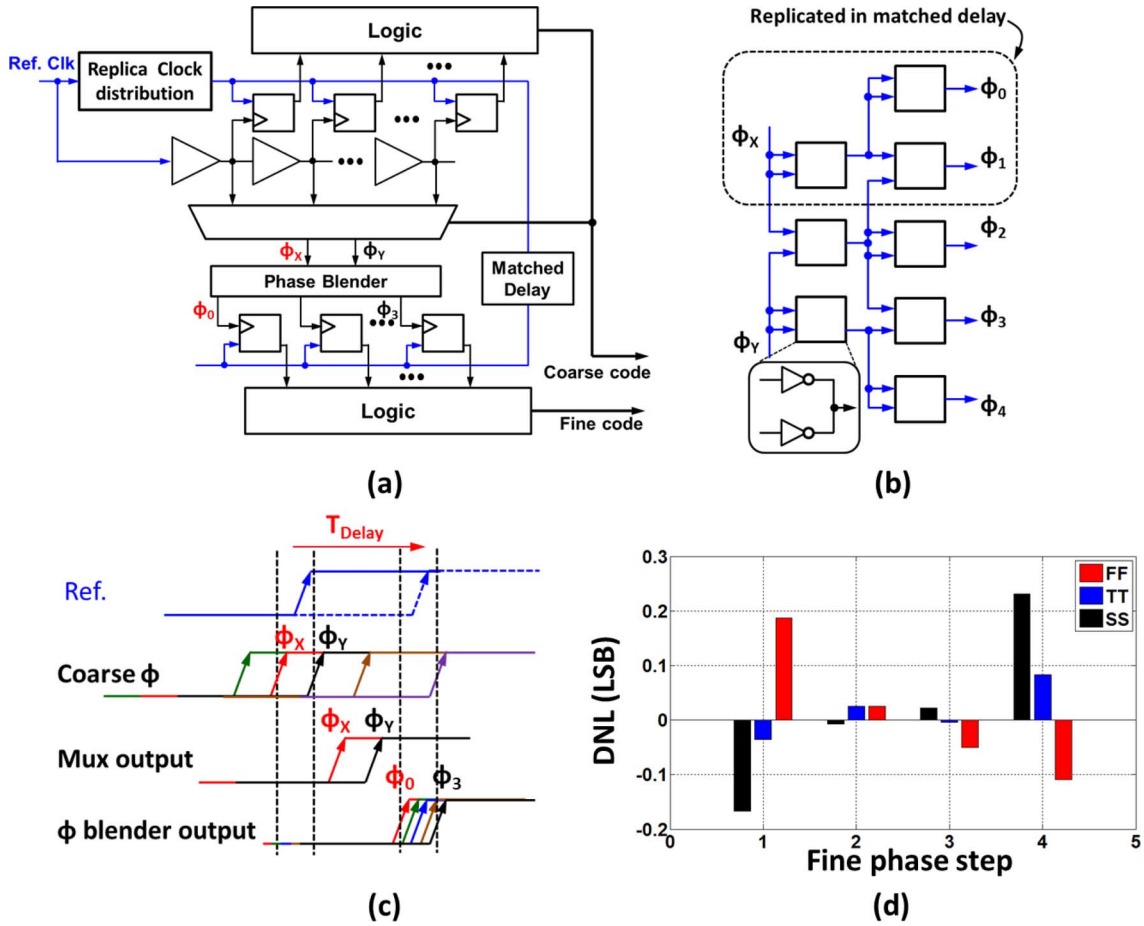


Fig. 5. (a) Two step TDC; (b) phase blender; (c) functionality elaborated in timing diagram; (d) DNL.

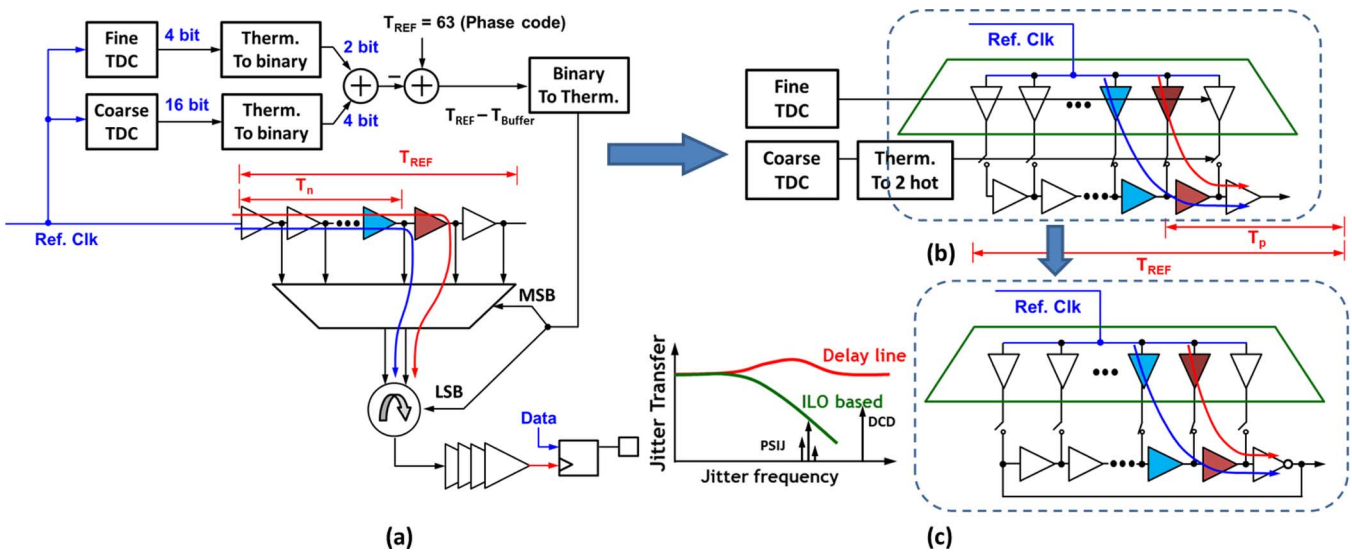


Fig. 6. Improvement on lock time and digital to phase converter: (a) conventional implementation; (b) direct complementary mapping; (c) converting the delay line to ILO.

This simply indicates that CK and DQS signals are phase locked. Since the structure itself provides complementary mapping of the code, code conversion logic is not needed and saves $3T_{REF}$ from the lock time.

C. ILO Based Digital to Phase Converter

As explained in the previous section, the modified architecture already provides significant improvement in lock time. This advantage can be further extended by feeding back the output

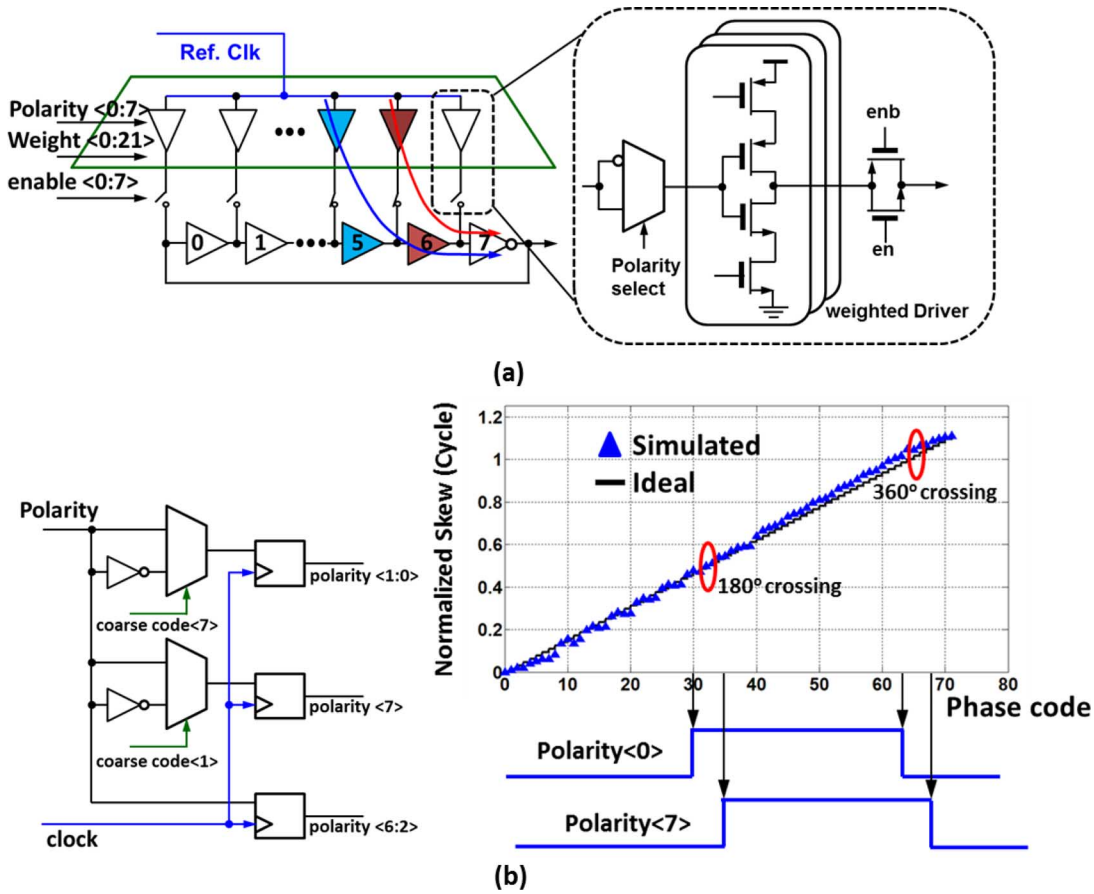


Fig. 7. (a) Glitch-less infinite phase rotation scheme. (b) Polarity switching timing.

of the delay line to the input in the form of an injection-locked oscillator (ILO) (Fig. 6(c)). Since an ILO has the property of low pass jitter transfer, this rather simple modification is effective in filtering high frequency input noise without adding any extra power or complexity. The ILO based phase interpolator has been well explored by [10], [11]. The working principle is simple—the injection point in the ILO keeps shifting while the output node remains fixed. This achieves different amounts of phase shift with relatively coarse resolution, 180°/N, set by the number of stages in the ILO, N. Therefore, increasing number of stages in the ILO improves the phase resolution at the cost of increased power consumption. In this design, an 8 stage ring oscillator was found to be a reasonable compromise between power consumption and coarse resolution of 22.5°. The 8 stage ring VCO also has a wide tuning range from 400 MHz to 1.6 GHz. To achieve finer resolution, two adjacent stages are injected at the same time with variable strength. This technique improves the resolution by $4\times$. Although the delay line only covers up to 180°, it has been shown that with input or output polarity selection it is possible to cover infinite range like a regular phase mixer [10]. However, a momentary glitch may occur while crossing the 180° boundary by polarity inversion. Note that this glitch is less of a concern for calibration based deskew scheme where the phase is 'frozen' after the calibration and during the functional mode. However, in DLL applications continuous tracking is required during functional mode, therefore such a glitch is not acceptable. A simple solution is to avoid the glitch issue by moving the polarity selection to each individual injection stages (Fig. 7). Rather than using a single polarity signal each injection point has its independent polarity selection. In the proposed method the digital loop filter's output phase code is observed, and based on the phase code the state machine predictively updates the polarity of the following injection stages. Let's consider 180° boundary crossing, this requires updating the polarity of 0th and 7th stage depending clock wise or anti clock wise rotation. Coarse code $\langle 7 \rangle$ 'high' indicates that injector 6th and 7th are enabled, and if the phase rotation continues injector $\langle 0 \rangle$ can be enabled next. Note that only two injectors are 'active' at any given time (in this case 6th and 7th), and the remaining 6 injectors are disabled thus the DLL output is insensitive to their injection signals. Therefore, polarity $\langle 0 \rangle$ is flipped based on coarse code $\langle 7 \rangle$ before injector $\langle 0 \rangle$ is enabled, and allows glitch-less boundary crossing. Similarly, polarity $\langle 7 \rangle$ is updated based on coarse code $\langle 1 \rangle$ before injector $\langle 7 \rangle$ is enabled.

In a conventional approach, resolution of the TDC is ensured by locking N stage delay line to $\mathrm{T_{REF}}$. This can either be a continuously running DLL or a periodic delay calibration loop. Similar approach is also needed for conventional Digital to phase conversion. In this implementation 8 identical stages are used for both TDC delay line and ILO. In the ILO 8th stage output is connected back to the 1st stage input whereas in the TDC 8th stage output is terminated with dummy load. Therefore, by calibrating the ILO frequency to be same as input frequency, we can ensure the resolution of the digital to phase converter as well as the TDC. ILO 'free running' frequency is cali-

brated using a digital counter same as [9]. ILO lock range is sufficiently wide, therefore small frequency offset is usually tolerable without significance performance impact. Note that if the data rate is changed, ILO free running frequency needs to be re-calibrated to match the new data rate similar. Rate change protocol in a DRAM interface usually allows sufficient time for such calibration.

D. Duty Cycle Correction

There are two primary sources of duty cycle error present in the DLL, input duty cycle error and duty cycle distortion added by the clock distribution buffer due to mismatch. One simple duty cycle correction method is to observe the duty cycle error at the output of the clock distribution network and accordingly correct the duty cycle at the input by adding DC offset. However, such loops take a long time (on the order of micro seconds) to converge making them less suitable for burst mode applications.

Mismatch profiles (both device and parasitics) are to the 1st order voltage and temperature independent. Since the clock distribution network remains unchanged, a simple one time calibration code is sufficient to correct the duty cycle distortion from the clock distribution buffer. However, input duty cycle error is unknown and unpredictable. Therefore, it can't be corrected through one time calibration. Rather than using a closed loop correction which takes longer to lock, the jitter filtering property of ILO to correct input DCD is used. Since the ILO has a low pass jitter transfer function and DCD is essentially very high frequency jitter, most of the DCD will be filtered out. If the input duty cycle error is within $\pm 10\%$, the ILO's jitter filtering is sufficient to provide nearly 50% duty cycle clock at the output.

However, it is also possible to completely remove duty cycle error using pulse injection. In this injection scheme a narrow pulse is generated from the reference clock $(\mathrm{ref}(t))$ by first delaying it $(\mathrm{ref}(t-\Delta T))$ and then passing through a NAND gate as shown in Fig. 8. Since the pulse is generated from only 'rising' or 'falling' edges, it is periodic to the reference cycle but now completely independent of the duty cycle. For example, even if the input duty cycle is 60/40, if the injection pulses are generated from only rising or falling edges these pulse will be separated by reference period as shown in the figure. The VCO itself provides 50% duty cycle clock, and these periodic reference pulses are then used to lock the phase by correcting the 'zero crossing' in each cycle of the VCO. Therefore, this scheme can provide a phase locked clock with nearly 50% duty cycle regardless of any reasonable input DCD (Fig. 9). To accommodate this duty cycle correction technique, a pulse generator and ILO need to be added following the DLL increasing overall power consumption. Since memory interfaces have well defined DCD specification, the pulse injector and additional ILO for DCD correction is avoided in this implementation.

IV. DLL Jitter Analysis

Compared to a conventional DLL architecture using a VCDL, the proposed DLL utilizes an ILO (Fig. 10). This significantly changes the jitter transfer properties of the DLL. To evaluate these changes analytically, phase domain equivalent models with dominant noise sources are shown in Fig. 10. For simplicity, the open loop (i.e., TDC) part is omitted in this model. Major sources of noise in DLLs are high frequency input jitter, quantization noise due to digital loop, and supply induced jitter in the DLL and clock distribution buffers. Bang-bang phase detectors are inherently non-linear and their gain depends on input jitter amplitude. Assuming Gaussian input jitter, the phase detector gain can be linearized as $\mathrm{K_{BB}} = 1/\sigma\sqrt{2\pi}$, where σ is the standard deviation of the input jitter distribution [12]. The digital loop filter is simply modeled as decimation factor $\mathrm{K_D}$ and an equivalent continuous time equivalent $L(s)$ of discrete time digital to phase converter $L(z^{-1}) = K_D K_{\mathrm{DPC}}/(1-z^{-1})$. Here, $\mathrm{K_D} = 1/2^2$ and $\mathrm{K_{DPC}}$ is programmable from $1/2^6$ to $1/2^9$ to reduce dithering jitter at the cost of DLL bandwidth.

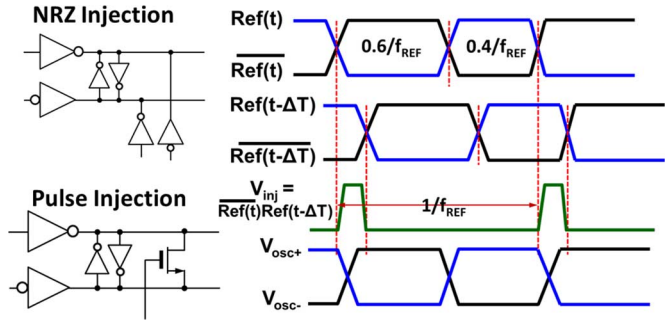


Fig. 8. Circuit schematics to implement NRZ and pulse injection methods. Duty cycle correction using pulse injection method.

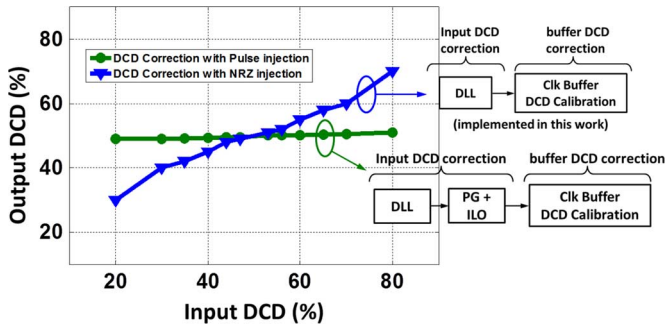


Fig. 9. Duty cycle distortion correction range with two injection methods.

Different noise transfer functions are derived in Appendix I. Based on the phase domain model of the proposed DLL, a summary and comparison with standard DLL is provided in Table II. In a conventional DLL input jitter transfer is all pass with negligible peaking (less than 1 dB) caused by the signal latency added by the delay line:

$$\frac{\Phi_{\mathrm{out}}}{\Phi_{\mathrm{in}}} = \frac{1 + s\left(\frac{1}{\omega_{\mathrm{DLL}}}\right)e^{-\Delta T s}}{\left(1 + \frac{s}{\omega_{\mathrm{DLL}}}\right)} \tag{5}$$

For digital implementation, ω_{DLL}, the pole in the DLL, can be approximated as $\omega_{\mathrm{DLL}} \approx K_{BB} K_D K_{\mathrm{DPC}}$. For the given loop filter parameters and a 1.6 GHz reference clock frequency, the pole is around 4 MHz. Replacing this delay line with an ILO adds an additional pole to this transfer function:

$$\frac{\Phi_{\mathrm{out}}}{\Phi_{\mathrm{in}}} = \frac{1 + s\left(\frac{1}{\omega_{\mathrm{DLL}}} + \frac{1}{\omega_{\mathrm{ILO}}}\right)}{\left(1 + \frac{s}{\omega_{\mathrm{DLL}}}\right)\left(1 + \frac{s}{\omega_{\mathrm{ILO}}}\right)} \tag{6}$$

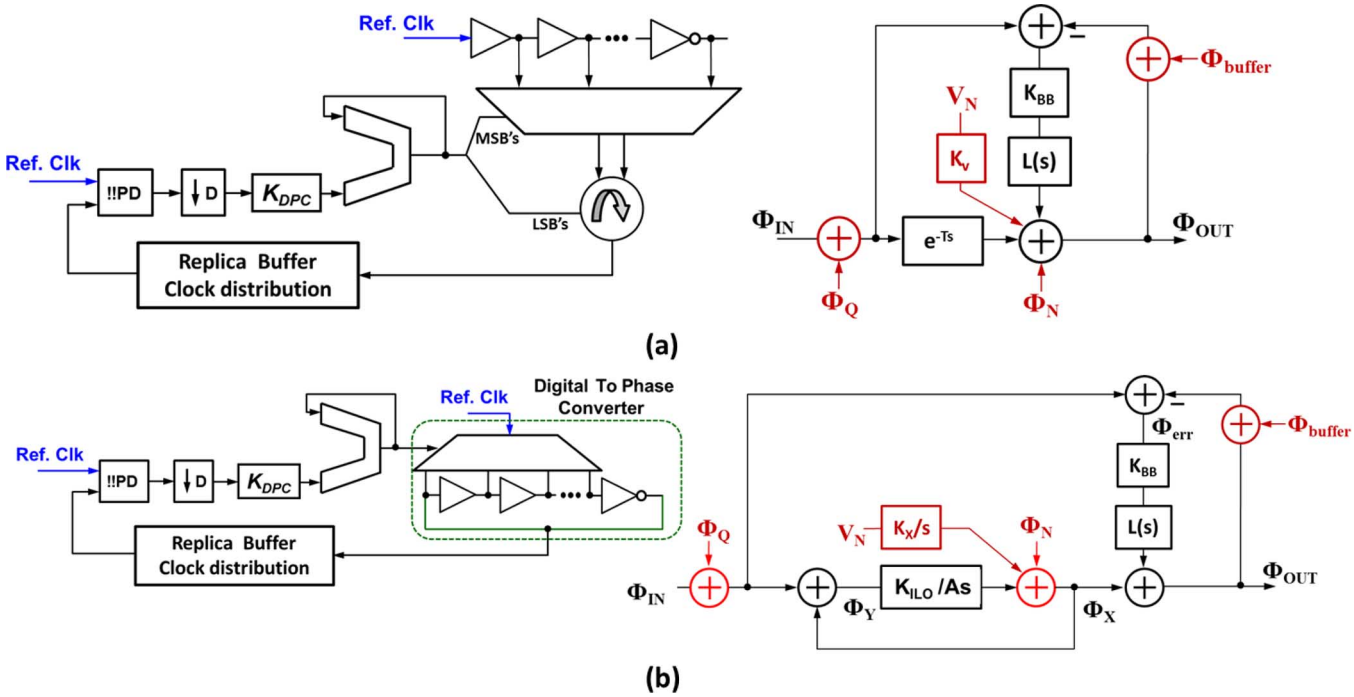


Fig. 10. DLL and its equivalent linearized phase model with noise sources: (a) conventional DLL; (b) proposed DLL.

TABLE II
COMPARISON OF DIFFERENT NOISE TRANSFER FUNCTIONS

	conventional DLL	**Proposed DLL**
Random Noise $\frac{\Phi_{out}}{\Phi_n}$	$\frac{s/\omega_{DLL}}{1+s/\omega_{DLL}}$	$\left(\frac{s/\omega_{DLL}}{1+s/\omega_{DLL}}\right)\left(\frac{s/\omega_{ILO}}{1+s/\omega_{ILO}}\right)$
supply Noise $\frac{\Phi_{out}}{v_n}$	$K_v\left(\frac{s/\omega_{DLL}}{1+s/\omega_{DLL}}\right)$	$\frac{K_x}{K/A}\left(\frac{s/\omega_{DLL}}{1+s/\omega_{DLL}}\right)\left(\frac{1}{1+s/\omega_{ILO}}\right)$
Quantization Noise $\frac{\Phi_{out}}{\Phi_Q}$	$\frac{1+s\left(\frac{1}{\omega_{DLL}}\right)e^{-\Delta Ts}}{\left(1+\frac{s}{\omega_{DLL}}\right)}$	$\frac{1+s\left(\frac{1}{\omega_{DLL}}+\frac{1}{\omega_{ILO}}\right)}{\left(1+\frac{s}{\omega_{DLL}}\right)\left(1+\frac{s}{\omega_{ILO}}\right)}$
Buffer Noise $\frac{\Phi_{out}}{\Phi_{Buffer}}$	$\frac{s/\omega_{DLL}}{1+s/\omega_{DLL}}$	$\frac{s/\omega_{DLL}}{1+s/\omega_{DLL}}$

Here, ω_{ILO} is the pole of the ILO and is defined as the ratio between injection strength K, and VCO's injection sensitivity A. For this particular implementation injection strength is set by the ratio between the strength of injection and oscillation buffer, $K = W_{\mathrm{osc}}/W_{\mathrm{inj}}$. Injection sensitivity for a n-stage ring oscillator is defined as $\mathrm{A} = \mathrm{n}/(2\omega_0)\ \sin(2\pi/\mathrm{n})$. Note that the zero is now a function of both ω_{DLL} and ω_{ILO}. Therefore to avoid peaking, the two poles should be separated as far as possible, $\omega_{\mathrm{ILO}} \gg \omega_{\mathrm{DLL}}$. Fortunately, ILOs are known to have high tracking bandwidth, making the transfer function behave like 1st order low pass filter with dominant pole at ω_{ILO}.

Random noise is less of a concern in conventional DLLs since there is no jitter accumulation in a delay line. Compared to that, feedback loops using voltage controlled oscillators (for example, PLLs) have poorer random jitter performance due to jitter accumulation in the oscillator. Therefore, RJ performance is evaluated by predicting and measuring phase noise performance. As derived in Appendix I, random jitter is high pass filtered by both injection locked loop and delay locked loop:

$$\frac{\Phi_{\mathrm{out}}}{\Phi_n} = \left(\frac{s/\omega_{\mathrm{DLL}}}{1+s/\omega_{\mathrm{DLL}}}\right)\left(\frac{s/\omega_{\mathrm{ILO}}}{1+s/\omega_{\mathrm{ILO}}}\right) \tag{7}$$

At low frequencies, $\omega < \omega_{\mathrm{DLL}}$ random noise is high pass filtered by both DLL and ILO. At mid frequencies, $\omega_{\mathrm{DLL}} < \omega < \omega_{\mathrm{ILO}}$ random noise is only filtered by the ILO, and only at very high frequencies $\omega > \omega_{\mathrm{ILO}}$ does the VCO's phase noise appear at the output. Similar to [8], DLL phase noise can be expressed by shaping VCO phase noise (S_{VCO}) and input phase noise (S_{REF}) profiles with above transfer functions:

$$S_{\mathrm{out}} = \left|\frac{\Phi_{out}}{\Phi_{\mathrm{in}}}\right|^2 S_{\mathrm{REF}} + \left|\frac{\Phi_{\mathrm{out}}}{\Phi_n}\right|^2 S_{\mathrm{VCO}} \tag{8}$$

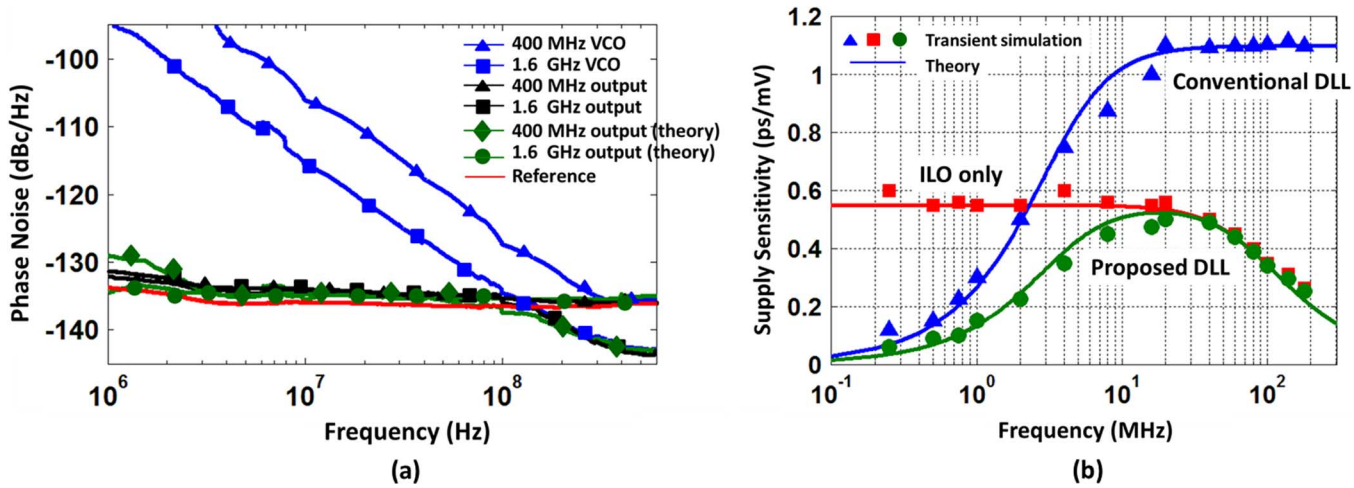


Fig. 11. DLL noise performance. (a) Measured and predicted phase noise from the expression given in (8). (b) Simulated and predicted supply sensitivity from (9), (10), and (11).

Since ILO tracking bandwidth exceeds 80 MHz, the VCO adds very little phase noise to the reference signal (Fig. 11). Measured results also demonstrate good agreement between measured and predicted phase noise from DLL phase noise expression.

Supply noise induced jitter is a major source of timing noise in high speed interfaces. Similar to random noise, supply noise also does not accumulate in a DLL. Therefore, it is high pass filtered without peaking:

$$\left(\frac{\Phi_{\text{out}}}{v_n}\right)_{\text{DLL}} = K_v \left(\frac{s/\omega_{\text{DLL}}}{1+s/\omega_{\text{DLL}}}\right) \tag{9}$$

Where K_v is the raw sensitivity of the delay line; for this design, it is 1.1 ps/mV. However, within the DLL bandwidth, the loop compensates supply induced jitter by adjusting VCDL delay; therefore, the sensitivity is better within the DLL loop bandwidth. On the other hand a standalone ILO demonstrates low-pass supply noise transfer:

$$\left(\frac{\Phi_{\text{out}}}{v_n}\right)_{\text{ILO}} = \frac{K_x}{K/A}\frac{1}{1+s/\omega_{\text{ILO}}} \tag{10}$$

In an ILO supply noise pulls the VCO similar to the injection pulling mechanism. However, the reference injection path is significantly stronger and opposes the supply pulling, thus reducing its effect by the ratio $K_x/(K/A)$ resulting in a lower supply sensitivity of 0.55 ps/mV. At higher frequencies, $\omega > \omega_{\text{ILO}}$, the ILO is less sensitive to both reference injection and supply pulling. When compared to a conventional DLL's supply sensitivity, the ILO's low frequency sensitivity is poor. However, at mid and high frequencies ILOs achieve better supply sensitivity compared to a DLL. Fortunately, in the proposed DLL we see the benefit of both solutions. Supply sensitivity of the proposed DLL can be written as:

$$\frac{\Phi_{\text{out}}}{v_n} = \frac{K_x}{K/A}\left(\frac{s/\omega_{\text{DLL}}}{1+s/\omega_{\text{DLL}}}\right)\left(\frac{1}{1+s/\omega_{\text{ILO}}}\right) \tag{11}$$

The above expressions are in good agreement with the transient simulation results shown in Fig. 11. At low frequencies, the feedback loop corrects the supply induced jitter whereas at higher frequencies outside the ILO's tracking bandwidth it is insensitive to supply pulling. In the mid frequency range, the ILO improves the DLL's sensitivity by a factor of $2\times$. Compared to other VCO based closed loop solutions such as PLLs, the proposed solution provides significantly better supply rejection. This is mainly because noise accumulation is limited to one cycle only since there is a correction pulse injected at every cycle.

Similarly quantization noise transfer function can be approximated as:

$$\frac{\Phi_{\text{out}}}{\Phi_Q} \approx \frac{1}{\left(1+\frac{s}{\omega_{\text{ILO}}}\right)} \tag{12}$$

This provides 1st order filtering of the quantization noise. During steady state the digital loop filter output dithers between 2 to 3 phase codes causing the recovered clock to suffer additional 'bang-bang' jitter. Typically, bang-bang jitter takes a significant part of the timing budget, and its main components are phase step size, DNL and loop latency. Note that 'bang-bang' jitter is periodic and its frequency is the same as, or a sub-harmonic of, the digital loop filter's clock frequency. A conventional DLL with all-pass jitter transfer does not reduce this noise. But in the proposed DLL it is possible to filter out part of the periodic noise by appropriately selecting ω_{ILO}. The digital loop filter's clock frequency scales with technology, therefore, shifting the periodic jitter to higher frequencies and making the ILO's quantization noise filtering more effective.

The clock buffer jitter for both conventional and proposed solutions remain the same; in both cases they are shaped with 1st order high pass filter with cutoff frequency set by the DLL bandwidth:

$$\frac{\Phi_{\text{out}}}{\Phi_{\text{buffer}}} = \left(\frac{\frac{s}{\omega_{\text{DLL}}}}{1+\frac{s}{\omega_{\text{DLL}}}}\right) \tag{13}$$

V. Implementation and Measured Results

The complete DLL implementation is shown in Fig. 12. The fully digital implementation results in a compact implementation and also enables physical design flexibility. Switching from

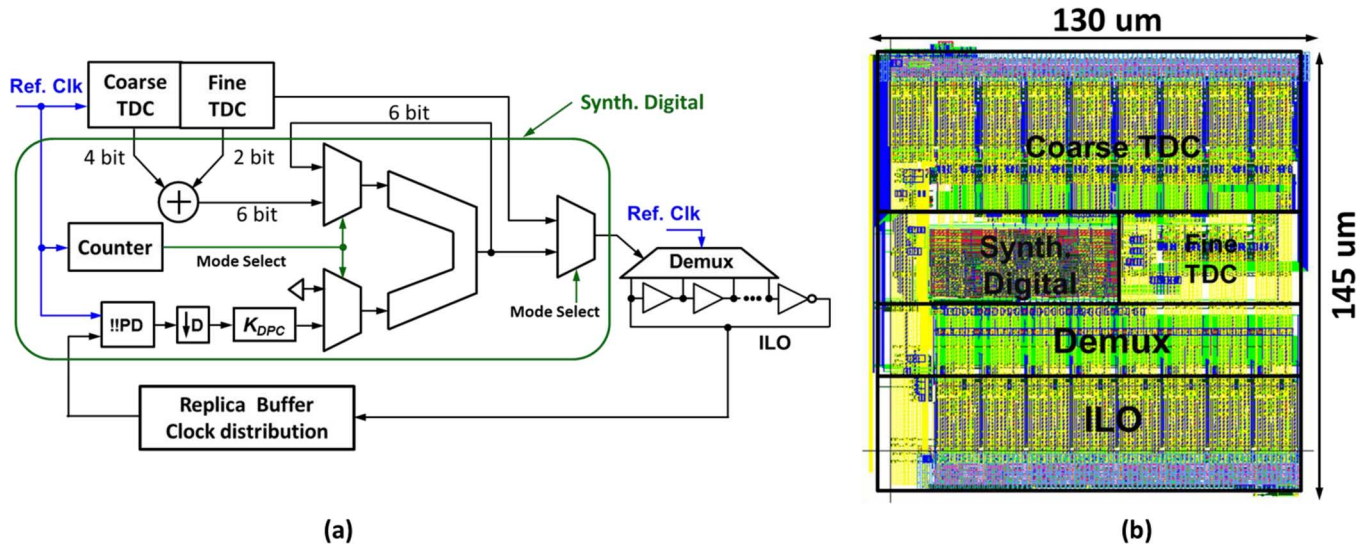


Fig. 12. DLL implementation. (a) Top level block diagram. (b) Layout in 40 nm CMOS.

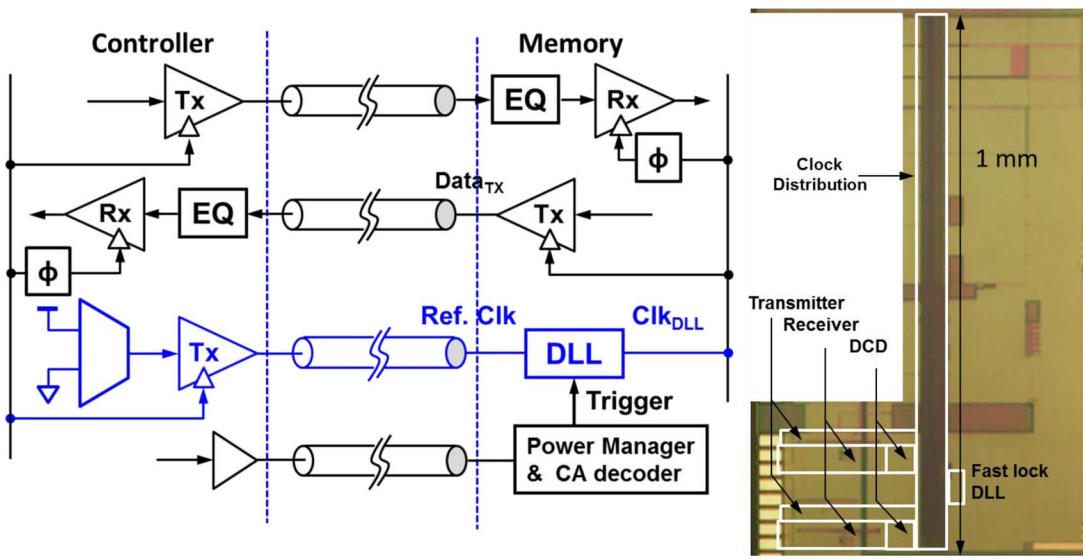


Fig. 13. Complete link implementation using the DLL in 40 nm CMOS.

fast lock to continuous tracking is pre-programmed by a counter preset to specify the number of reference cycles the DLL will spend in fast lock mode before switching. Since the TDC takes only 2 clock cycles to generate the code and 1 clock cycle to hold it, the default counter setting is 3 cycles. DLL bandwidth and averaging are controlled by gain factor 'K_{DPC}', which is set by adjusting the accumulator size. Jitter filtering bandwidth is set by programming injection strength. This allows the DLL to optimize link performance for different input jitter. To evaluate the system level performance, a complete link using this DLL is built in 40 nm CMOS (Fig. 13). Although a typical memory interface width is $8\times$ or $16\times$, for simplicity this prototype implements only two lanes. However, a 1 mm long clock distribution network is implemented to represent a typical DRAM clock distribution load. The fast locking transient is shown in Fig. 14. Reference clock and DLL output clocks are initially 180° out of phase. Once the DLL is activated with the falling edge trigger it takes 12.58 ns for the output clock to lock to the input reference. The experiment is repeated for different initial phase error and in all cases phase error reduces to 1/64 clock cycle within 13 ns (Fig. 14(b)). The link wakeup transient is shown in Fig. 15. In addition to phase lock time, fast bias turn-on takes an additional 2 ns for bias voltages to settle before link operation is initiated. The link readiness after fast locking is evaluated by transmitting and recovering a 128 bit pattern with sufficient timing margin (Fig. 15(b)). Continuous tracking mode DLL output clock (DQS) and data (DQ) are shown in Fig. 16. Jitter performance of the ADDLL (31 ps @ 3.2 Gb/s) is comparable to analog implementations. Low-pass jitter transfer characteristics are demonstrated in Fig. 17, with specific jitter transfer for modulation frequency of 400 MHz shown in Fig. 17. Duty cycle improvement is demonstrated in Fig. 18 where the ILO was able to correct $\pm 10\%$ duty cycle error. To evaluate the benefit of active tracking, we evaluated timing margin for three cases: First, the DLL was turned 'ON' to find the correct skew setting and then the DLL phase code is 'frozen' at that value. No supply noise is added and in addition most of the unused circuits are disabled to minimize supply noise. Although this is

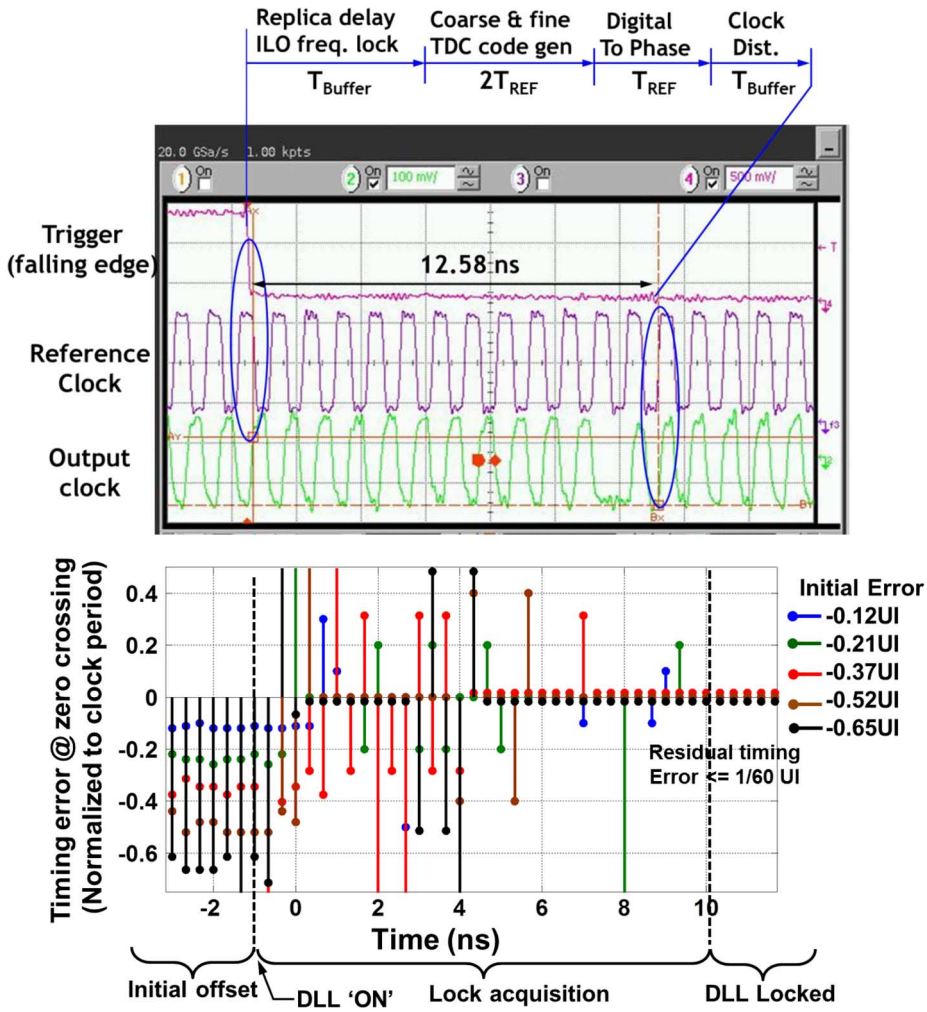


Fig. 14. DLL locking transient with initial error. Timing error vs.time for different initial error.

TABLE III
COMPARISON OF DIFFERENT FAST LOCK TECHNIQUES

Reference	J. Kim JSSC'00	Dehng JSSC'01	Liang TCAS'08	O'Mahony JSSC'10	M. Kim CICC'10	This Work
Architecture	TDC + Feedback (Analog)	SAR	Binary search	Tracking ADC + Feedback	Two step SMD	Two step TDC
Lock time	12 cycle	14 cycle	16 cycle	>16 Cycle	3 to 10 Cycle	3 cycle

rather unrealistic case, it allows us to quantify the effect of the supply noise or DLL self-generated noise. In the second case $\pm 10\%$ triangular supply noise was applied without enabling the DLL. This shifts the eye causing the timing margin to degrade as shown in Fig. 19. In the third case, when the DLL is enabled, the shift in the data eye is corrected as the DLL cancels out skew caused by supply noise. Although the unfiltered part of the PSIJ and self-generated noise of the DLL reduces timing margin compared to the noiseless case, there is significant improvement over an open loop solution.

Previous fast-lock solutions include different digital algorithms such as successive approximation and binary search. Although these solutions can significantly improve lock time, they still take up to 10 reference cycles (Table III). Since these designs were not targeting DRAM clock skew compensation, clock buffer delay is not included in the reported lock time. Therefore, in a direct comparison ignoring clock buffer latency, the proposed two-step TDC based proposed solution guarantees locking within 3 reference clock cycles while consuming low power and area. The proposed DLL solution is also compared with other existing DLLs in Table IV. Existing DLL solutions are feedback (both analog and digital) based to achieve good resolution consuming low power. However, such loops take a very long time (500+ cycles) to lock requiring the DLL to remain 'ON' all the time (including during idle modes). TDC based solutions can improve lock time significantly, but

TABLE IV
PERFORMANCE COMPARISON OF DIFFERENT DLL

	K. Lee et al JSSC'07	M. Kim CICC'10	H. Lee et al JSSC'12	This Work
DLL Architecture	Feedback (Analog)	SMD (digital)	Feedback (digital)	Hybrid (digital)
Data /Clock Rate	3.2 Gb/s	100 MHz to 1 GHz	1 GHz	800Mb/s To 3.2Gb/s
Lock time (Cycle) Link Wakeup time	1500	3	512	3 12 ns
Idle Power Active Power	------ 80 pJ/Hz (DL+Dist.)	0 64 pJ/Hz (DLL Only)	6.1 pJ/Hz 6.1 pJ/Hz (DLL only)	0 3.75 pJ/Hz (DLL only)
Area	0.3 mm^2	0.21 mm^2	0.10 mm^2	0.05 mm^2
Clock Jitter Tx Data Jitter	31 ps	14 ps (w/o buffer)	75 ps	31.6 ps 44.4 ps
TDC Resolution		12 ps		4.88 ps
Technology	90 nm (DRAM)	0.18 um (SOC)	4x CMOS (DRAM)	40 nm (SOC)

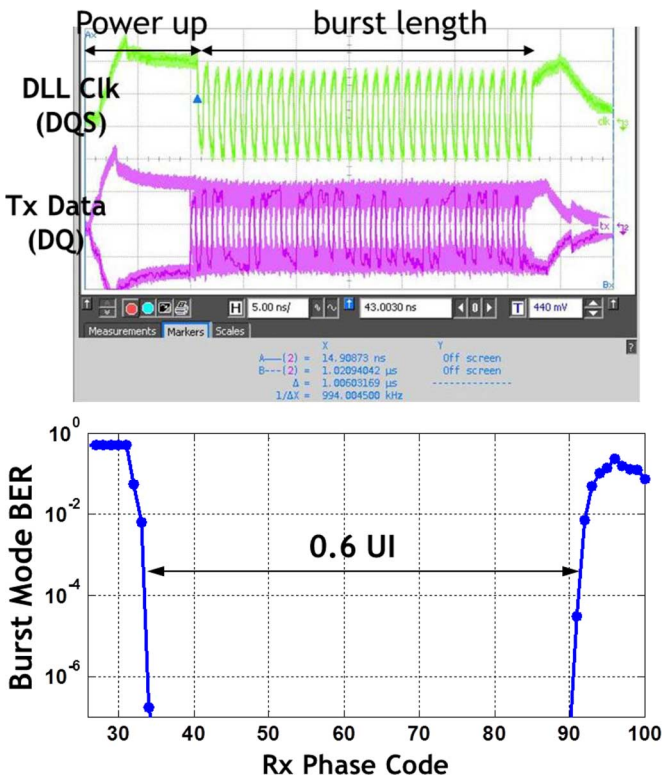


Fig. 15. Link wakeup transient with link burst mode BER on a 128 bit pattern.

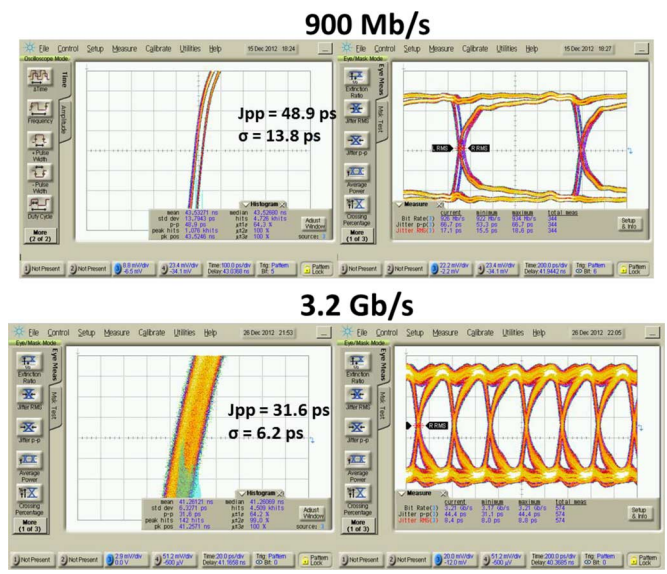


Fig. 16. Continuous mode performance. DQS (left side) and DQ (right side) at 900 Mb/s and 3.2 Gb/s.

their power consumption is prohibitive. This proposed hybrid solution combines the benefits of both feedback and TDC approaches achieving small area, low power, fast-lock and excellent jitter performance in a single design.

VI. CONCLUSION

Several power saving benefits of LPDDR mode can be achieved in regular DDR by simply changing the regular DLL to fast lock DLL. But improving lock time without significant power penalty requires a hybird approach where the DLL wakes up in fast lock mode and once the lock is achieved, the DLL switches over to lower power mode. Although the fast lock mode consumes more power, it is only 'ON' for a very short duration. Therefore, this architecture can achieve both fast locking and continuous tracking consuming low power. Two step TDC can achieve both fast lock and good resolution within affordable power budget. Furthermore, it is useful to be able filter out high frequency reference clock jitter including DCD. This jitter filtering can be easily added to the conventional DLL by simply using ILO as phase shifter. Such techniques can enable high-bandwidth DDR solutions to achieve LPDDR like power savings.

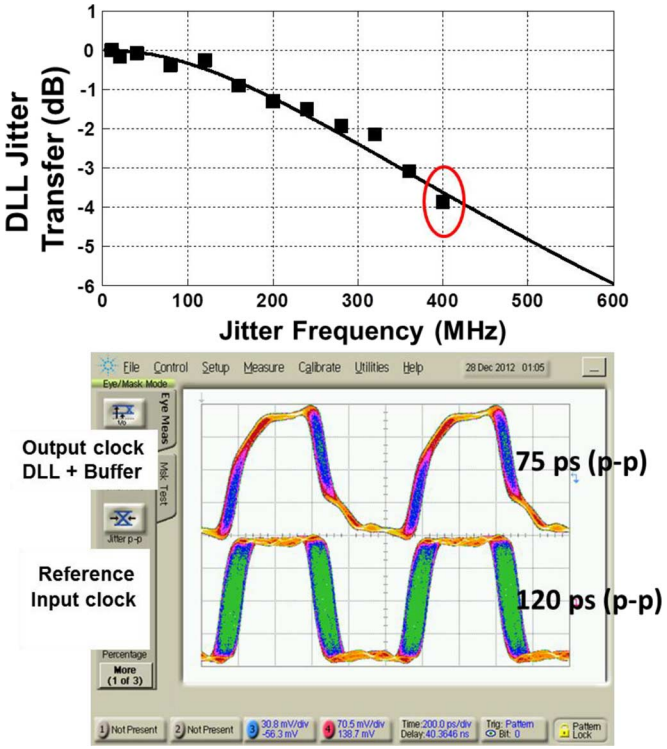


Fig. 17. DLL jitter transfer with DLL input and output clock phase modulated at 400 MHz.

APPENDIX I

I. Input Jitter Transfer

ILO's output phase can be written as:

$$\Phi_x = \Phi_y \frac{K}{As} = (\Phi_{\text{in}} - \Phi_x)\frac{K}{As}$$

Here, the injection strength K is defined as:

$$K = \frac{I_{\text{inj}}}{I_{\text{OSC}}} = \frac{W_{\text{osc}}}{W_{\text{inj}}}$$

and vco injection sensitivity A is defined as:

$$A = \frac{n}{2\omega_0} \sin\left(\frac{2\pi}{n}\right)$$

Substituting these variables, ILO's jitter transfer is derived as a first order low-pass filter:

$$\frac{\Phi_x}{\Phi_{\text{in}}} = \frac{K/As}{1 + K/As} = \frac{1}{1 + s/\omega_{\text{ILO}}} \tag{14}$$

Output phase is essentially the ILO's output with added phase shift resulted from the phase error shaped by the DLL loop gain $\mathrm{K_{BB}L(s)}$, where $\mathrm{L(s)}$ is the continuous time equivalent of the discrete time digital loop filter $\mathrm{L(z^{-1})}$.

$$\Phi_{\text{out}} = \Phi_{\text{err}} K_{BB} L(s) + \Phi_x$$
$$\Phi_{\text{out}}(1 + K_{BB}L(s)) = \Phi_{\text{in}} K_{BB} L(s) + \Phi_x$$

For intuitive understanding, it is useful to rewrite this expression in terms of DLL pole ω_{DLL}. For digital implementation the pole can be approximated as $\omega_{\text{DLL}} \approx K_{BB}K_D K_{\text{DPC}}$.

$$K_{BB}L(s) = \frac{\omega_{\text{DLL}}}{s}$$

Using this notation, we can re-write the output phase as a function of ILO and DLL poles.

$$\Phi_{\text{out}} = \left(\frac{\omega_{\text{DLL}}/s}{1 + \omega_{\text{DLL}}/s} + \frac{1}{(1 + \omega_{\text{DLL}}/s)(1 + s/\omega_{\text{ILO}})}\right)\Phi_{\text{in}}$$
$$\frac{\Phi_{\text{out}}}{\Phi_{\text{in}}} = \frac{1 + s\left(\frac{1}{\omega_{\text{DLL}}} + \frac{1}{\omega_{\text{ILO}}}\right)}{\left(1 + \frac{s}{\omega_{\text{DLL}}}\right)\left(1 + \frac{s}{\omega_{\text{ILO}}}\right)}$$

II. Random Noise Transfer

Random noise of the VCO appears at the output after being shaped by the ILO's inherent feedback structure:

$$\Phi_x = \Phi_y \frac{K}{As} + \Phi_n = -\Phi_x \frac{K}{As} + \Phi_n$$

After re-arranging, random noise transfer turns out to be high pass filtered up to ILO's tracking bandwidth:

$$\frac{\Phi_x}{\Phi_{\text{in}}} = \frac{1}{1 + K/As} = \frac{s/\omega_{\text{ILO}}}{1 + s/\omega_{\text{ILO}}} \tag{15}$$

Similar to the previous case, output phase can be expressed as combination of ILO's output phase with DLL's additional phase adjustment

$$\Phi_{\text{out}} = -\Phi_{\text{out}} K_{BB} L(s) + \Phi_x$$
$$\frac{\Phi_{\text{out}}}{\Phi_x} = \frac{1}{1 + \omega_{\text{DLL}}/s} = \frac{s/\omega_{\text{DLL}}}{1 + s/\omega_{\text{DLL}}} \tag{16}$$

From (15) and (16), random noise transfer function can be written as:

$$\frac{\Phi_{\text{out}}}{\Phi_n} = \left(\frac{s/\omega_{\text{DLL}}}{1 + s/\omega_{\text{DLL}}}\right)\left(\frac{s/\omega_{\text{ILO}}}{1 + s/\omega_{\text{ILO}}}\right)$$

III. Supply Noise Transfer

Similar to PLL, ILO frequency is also pulled by the supply noise by the factor K_x Hz/V. Therefore, the resultant phase shift K_x/s is shaped by the loop.

$$\Phi_x = \Phi_y \frac{K}{As} + v_n \frac{K_x}{s} = -\Phi_x \frac{K}{As} + v_n \frac{K_v}{s}$$

Rearranging this expression, we find ILO's supply noise transfer function

$$\frac{\Phi_x}{v_n} = \frac{K_x/s}{1 + K/As} = \frac{K_x}{K/A}\frac{1}{1 + s/\omega_{\text{ILO}}} \tag{17}$$

Output phase is then written as

$$\Phi_{\text{out}} = -\Phi_{\text{out}} K_{BB} L(s) + \Phi_x$$
$$\frac{\Phi_{\text{out}}}{\Phi_x} = \frac{1}{1 + \omega_{\text{DLL}}/s} = \frac{s/\omega_{\text{DLL}}}{1 + s/\omega_{\text{DLL}}} \tag{18}$$

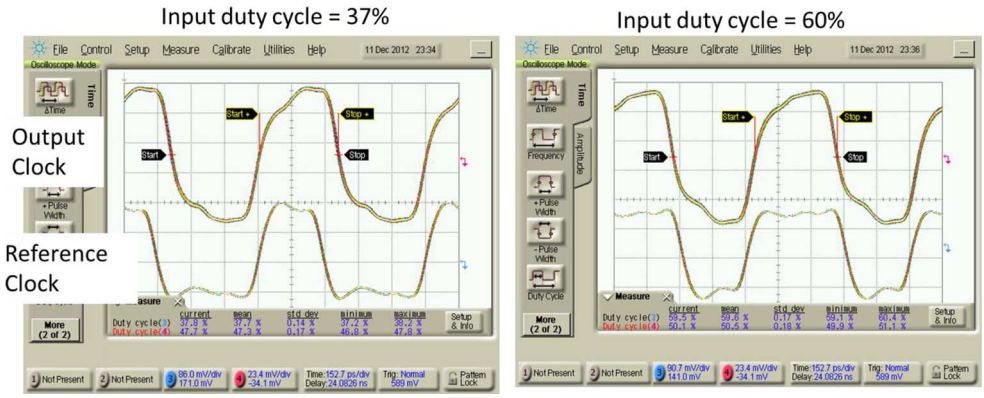


Fig. 18. Input and output duty cycle of the proposed DLL.

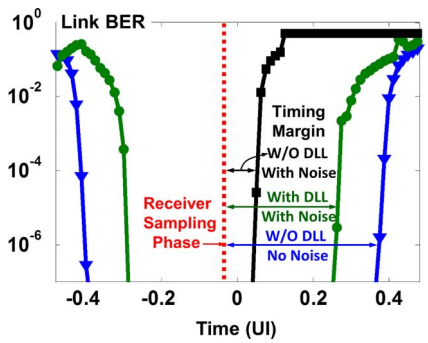


Fig. 19. Link timing margin in the presence of supply noise with and without DLL.

Combining (17) and (18) supply noise transfer function can be written as:

$$\frac{\Phi_{\text{out}}}{v_n} = \frac{K_x}{K/A}\left(\frac{s/\omega_{\text{DLL}}}{1+s/\omega_{\text{DLL}}}\right)\left(\frac{1}{1+s/\omega_{\text{ILO}}}\right).$$

REFERENCES

[1] R. Can *et al.*, "Save Power and Improve Efficiency in Virtualized Environment of Data Center by Right Choice of Memory," White paper, Samsung Semiconductor & Microsoft, May 2011.

[2] K. Donnelly *et al.*, "A 660 MB/s interface megacell portable circuit in 0.3 m–0.7 m CMOS ASIC," *IEEE J. Solid-State Circuits*, vol. 31, no. 12, pp. 1995–2003, Dec. 1996.

[3] B. Garlepp *et al.*, "A portable digital DLL for high-speed CMOS interface circuits," *IEEE J. Solid-State Circuits*, vol. 34, no. 5, pp. 632–644, May 1999.

[4] K. Sohn *et al.*, "A 1.2 V 30 nm 3.2 Gb/s/pin 4 Gb DDR4 SDRAM with dual-error detection and PVT-tolerant data-fetch scheme," *IEEE J. Solid-State Circuits*, vol. 48, no. 1, pp. 168–177, Jan. 2013.

[5] T. Saeki *et al.*, "A 2.5 ns clock access, 250 MHz 256-Mb SDRAM with synchronous mirror delay," *IEEE J. Solid-State Circuits*, vol. 31, no. 11, pp. 1656–1668, Nov. 1996.

[6] M. Kim and L. Kim, "A 100 MHz-to-1 GHz open-loop ADDLL with fast lock-time for mobile applications," in *Proc. IEEE Custom Integrated Circuits Conf., CICC 2010*, San Jose, CA, USA, Sep. 2010, pp. 1–4.

[7] J. Kim *et al.*, "A low-jitter mixed-mode DLL for high-speed DRAM applications," *IEEE J. Solid-State Circuits*, vol. 35, no. 10, pp. 1430–1436, Oct. 2000.

[8] M. Hossain *et al.*, "A 400 MHz–1.6 GHz fast lock, jitter filtering ADDLL based burst mode memory interface," in *Symp. VLSI Circuits Dig.*, Kyoto, Japan, Jun. 2013.

[9] J. Zerbe *et al.*, "A 5.6 Gb/s 2.4 mW/Gb/s bidirectional link with 8 ns power-on," in *Symp. VLSI Circuits Dig.*, Kyoto, Japan, Jun. 2011, pp. 82–83.

[10] M. Hossain and A. Chan Carusone, "CMOS oscillators for clock distribution and injection-locked deskew," *IEEE J. Solid-State Circuits*, vol. 44, no. 8, pp. 2138–2153, Aug. 2009.

[11] F. O'Mahony, B. Casper, M. Mansuri, and M. Hossain, "A programmable phase rotator based on time-modulated injection-locking," in *Symp. VLSI Circuits Dig.*, Honolulu, HI, USA, Jun. 2010.

[12] J. Sonntag and J. Stonick, "A digital clock and data recovery architecture for multi-gigabit/s binary links," in *Proc. IEEE Custom Integrated Circuits Conf., CICC 2005*, San Jose, CA, USA, Sep. 2005, pp. 532–539.

Masum Hossain received the Ph.D. degree at the University of Toronto, Toronto, ON, Canada, in 2010. Prior to that, he received the B.Sc. degree from the Bangladesh University of Engineering and Technology, India, and the M.Sc. degree from Queen's University, Kingston, ON, Canada, in 2002 and 2005, respectively.

He joined the faculty of the University of Alberta Department of Electrical and Computer Engineering in winter 2013. Before returning to academia, he has spent several years in industrial research. From 2008 to 2010, he was with Gennum Corp. in the Analog and Mixed Signal division where he focused on the development of world's highest capacity and most power-efficient cross point router solution. Following that, he joined Rambus Lab as a senior member of technical stuff, and focused on advanced equalization and clock recovery techniques for high-speed interfaces.

Dr. Hossain won the best student paper award at the 2008 IEEE Custom Integrated Circuits (CICC) Conference. He also won Analog Device's outstanding student designer award in 2010.

Farrukh Aquil received the B.S.E.E. degree from Mehran University of Engineering and Technology, Jamshoro, Pakistan, in 1992 and the M.S.E.E. degree from Concordia University, Montreal, Canada, in 2001.

From 2001 to 2008, he was with Infineon Technologies/Qimonda, where he worked on high-speed DRAM development. Between 2009 and 2010, he was with IBM, where he was involved in non-volatile memory research and development. From 2010 to 2012, he was with Rambus, Inc., Sunnyvale, CA, USA, as a Principal Design Engineer, where he was involved in low power mobile DRAM design and worked on high-speed CMOS I/O and DLL circuits.

Since September 2012, he has been with Qualcomm, San Diego, CA, USA, as a designer and architect of high-speed DDR PHY design.

Pak Shing Chau (S'89–M'91) was born in Hong Kong in 1966. He received the B.S. degree in computer system engineering from the University of Massachusetts, Amherst, MA, USA, in 1989, and the M.S. degree in electrical engineering from the University of California, Davis, CA, USA, in 1991.

From 1991 to 1994, he was at National Semiconductor and Chrontel Inc., where he worked on analog circuit designs. In 1994, he joined Rambus Inc., Mountain View, CA, USA, where he designed high-speed clock recovery and I/O circuits for synchronous chip-to-chip communication technologies, such as RDRAM and QRSL. In 2001, he joined Aeluros Inc., Mountain View, CA, USA, where he worked on the design of first generation 10 Gbps CMOS multi-protocol transceiver products and provided field support of the devices. In 2006, he returned to Rambus Inc., Los Altos, CA, USA, where he worked on the XDR DRAM interface design and is engaged in high-speed/low-power circuit design.

Brian Tsang was a principal engineer in the Systems Engineering Group at Rambus, where he divided his time between developing infrastructure and researching next-generation system architecture. He has ten years of experience in board design, package design, and software development, and is currently working at SiTime bringing MEMs oscillators to the market.

Phuong Le, photograph and biography not available at time of publication.

Jason Wei received the B.S. degree in electrical engineering from National Cheng Kung University, Taiwan, and the M.S. degree in electrical engineering from San Jose State University, San Jose, CA, USA.

He is currently a Senior Principal Engineer at Rambus Inc. developing technology for high-speed and low-power memory interface. Since joining Rambus in 1994, he has worked on high speed I/O, CDR, PLL and equalization circuits for a backplane SerDes, PCIE and chip-to-chip memory interface bus.

Teva Stone received the M.S. and B.S. in electrical engineering from North Carolina State University, Raleigh, NC, USA, in 1992 and 1990, respectively.

She was a design engineer at Rambus Inc, Chapel Hill, NC, USA, from 2003 to 2012 and has 20 years' experience which includes design of high-speed serial interface clocking for companies such as Velio Communications and National Semiconductor. She now works as an independent contractor.

Barry Daly (M'98) received the B.Eng. degree in electronic engineering from the Cork Institute of Technology, Cork, Ireland, in 1996.

He is a senior design engineer with Rambus Inc., Chapel Hill, NC, USA, working on high speed serial link and memory bus design. Since joining Rambus in 2002, he has worked on mixed-signal circuit design for high speed chip-to-chip communications and related clocking circuits.

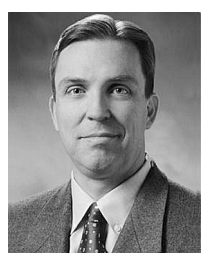
Chanh Tran received the B.S.E.E. degree from University of California, Berkeley, CA, USA, in 1989.

He is currently a Senior Engineering Manager at Rambus Inc. His group is responsible for high speed memory interface and high speed serial/parallel link development and production. Prior to Rambus, he worked at National Semiconductor as a design engineer in Data Acquisition System group.

John C. Eble (S'93–M'99) was born in Metairie, LA, USA, in 1971. He received the B.Cmp.E., M.S.E.E, and Ph.D. degrees in electrical engineering from Georgia Institute of Technology, Atlanta, GA, USA, in 1993, 1994, and 1998, respectively.

From 1998 to 2001, he worked on the EV7 high-speed I/O circuits in the Alpha Microprocessor Development Group, Compaq Computer Corporation, Shrewsbury, MA, USA. From 2001 to 2003, he was with Velio Communications as a circuit designer. In 2003 he joined Rambus Inc. where he has since specialized in the design of high-speed SERDES cells and next-generation memory signaling and clocking architectures. He has authored or co-authored over 30 technical publications and more than five patents and has contributed a book chapter on Off-chip Signaling. At Georgia Tech, he developed the original version of the Generic System Simulator (GENESYS) and received the Best Paper Award at the 1997 International ASIC Conference. He is currently the Director of US Design where his teams are focused on advanced development of high-performance and low-power memory and serial link interface technologies.

Kurt Knorpp, photograph and biography not available at time of publication.



Jared L. Zerbe was born in New York, NY, USA, in 1965. He received the B.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1987.

In 1987 he joined VLSI Technology, Inc. where he worked on custom and semi-custom ASICs and in 1989, he joined MIPS Computer Systems, where he designed high-performance CPU floating-point blocks. In 1992, he joined Rambus, Inc., Sunnyvale, CA, USA, where over 20 years he specialized in the design of high-speed I/O, PLL/DLL, clock-recovery, equalization and data-synchronization circuits, recently focusing on high-performance SerDes and fast power-on burst-mode low-power interfaces. He has taught courses at Berkeley and Stanford in high-speed I/O design and authored or co-authored over 40 IEEE conference and journal papers including forums and tutorials at ISSCC and VLSI Circuit Symposium. He has over 100 issued US patents and served on the program committee for DesignCon and VLSI Circuits Symposium from 2010–2013. In 2013 he joined Apple Inc.

Since 2011 Dr. Zerbe has been an associate editor for the IEEE JOURNAL OF SOLID-STATE CIRCUITS.