# A Multiple-Rotating-Clock-Phase Architecture for Digital Data Recovery Circuits using Verilog-A

S. I. Ahmed
Carleton University
Department of Electronics
Ottawa, ON K1S 5B6
siahmed@doe.carleton.ca

Tad A. Kwasniewski
Carleton University
Department of Electronics
Ottawa, ON K1S 5B6
tak@doe.carleton.ca

## ABSTRACT

This paper presents an oversampling Data Recovery (DR) architecture using Verilog-A that employs a novel Multiple-Rotating-Clock-Phase (MRCP) concept for its operation. The MRCP-DR architecture is a variant of the eye-tracking DR architecture [7]. Multiple rotating clock phases, supplied by a Delay-Locked Loop (DLL), establish a window for detecting data edges. As a result, the window width becomes robust against Process, Voltage and Temperature (PVT) variations. The MRCP architecture is tolerant of jitter on the local, blind, free-running oscillator that operates at approximately the incoming data frequency. Behavioral blocks are described and functional simulations are presented using the Verilog-A/Spectre/Cadence platform. The Verilog-A test benches allow the designer to perform system-level what-if analyses and make area, power and performance estimates.

**Index Terms**: **Data Recovery, Verilog-A, Behavioral Modeling, Jitter Tolerance, Tracking, MRCP**

## 1. INTRODUCTION

Clock and Data Recovery (CDR) circuits are used to recover serial data from an incoming data stream that often follows a Non-Return to Zero (NRZ) signaling scheme. The main component of a CDR or the Data Recovery (DR) circuit is the Phase-Locked Loop (PLL)[1]. Therefore, CDR circuits are broadly classified according to the type of the Phase Detector (PD) used in the PLL. Linear CDR circuits use a Hogge's style Phase Detector [2] and numerous such implementations (full-rate and reduced-rate) have been presented in literature. Non-linear CDR circuits employ Bang-Bang Phase Detectors [3] (BBPDs) that have become popular more recently [4], [5] for multi-Gb/s applications. The BBPD-based CDR circuits operate at data rates at which a flip-flop can be fabricated in a target technology and ameliorate many circuit-level challenges associated with linear CDR architectures [6], [8].

Traditional CDR circuits (linear or non-linear) perform a 2*x*-oversampling of the incoming data. One of the clock edges is aligned with a PLL to a data edge as the other clock edge samples the data. As an extension of this idea, the 3*x*-oversampling CDR architectures attempt to:

- track the data eye and sample at the centre of the eye [7],
- track the data edges and place the sampling clock in the middle of two data edges [8], or
- perform a blind oversampling of the data with an odd oversampling ratio and subsequently choose the most likely symbol algorithmically [9], [10].

The preceding examples are only a representative subset of oversampling DR architectures [14].

In Section 2, the MRCP architecture, which is a variant of the eye-tracking DR architecture [7], is described using Verilog-A blocks [11]. The simulation results are presented in Section 3 followed by the conclusions drawn from this work. Previously, we used Matlab/Simulink [12] to explore the feasibility of the MRCP-DR architecture [13], [14]. Further comments about the choice of Verilog-A as the simulation platform appear in Appendix A.

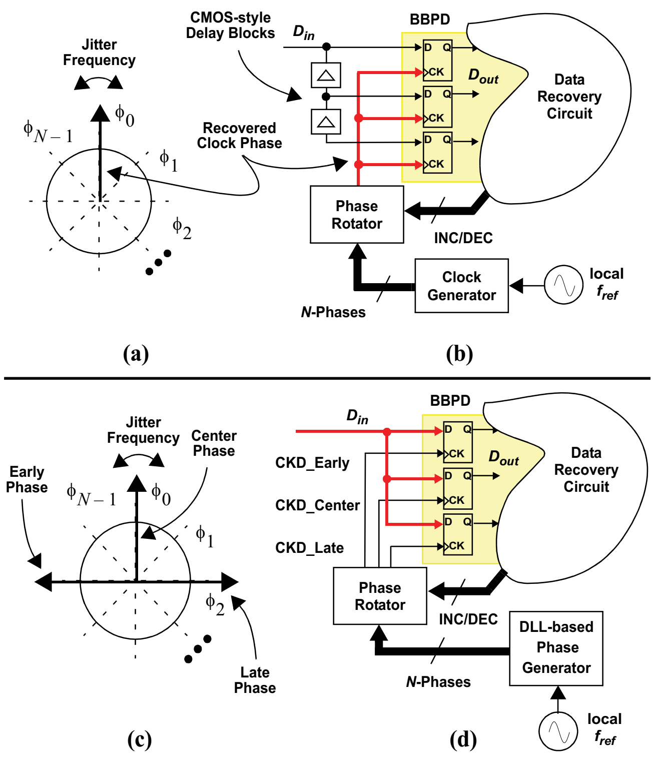


**Figure 1. Data recovery circuit concept (a) One rotating clock phase [7], (b) DR circuit [7], (c) Three rotating clock phases [14] (d) Multiple-Rotating-Clock-Phase architecture [14]**

## 2. MRCP ARCHITECTURE

The eye-tracking architecture uses CMOS-style delay elements to create multiple delayed versions of the incoming data that is subsequently recovered using one rotating clock phase (as shown in Figure 1(a) and Figure 1(b)). For further details and design guidelines, the reader is referred to [7]. In contrast, the MRCP architecture uses three rotating clock phases supplied by a DLL-based phase generator and the original data stream to achieve the same functionality (as shown in Figure 1(c) and Figure 1(d)). This is done to maintain a stable width for the data edge detection window in the presence of PVT variations. The width of the data edge detection interval is a critical parameter for realizing a predictable jitter tolerance [13]. The block diagram of the MRCP-DR architecture appears in Figure 2. The following sub-sections illustrate the Verilog-A implementation of the MRCP architecture. Behavioral primitives are used to maintain circuit-structure and reduce simulation time while behavioral code is used for the Phase Rotator (PR) block.

### 2.1. Bang-Bang Phase Detector

The UP and DN outputs from the BBPD provide an indication of the early/late data transitions. The BBPD was modeled as a circuit structure with behavioral flip-flops and gates. Three parameters were specified for all flip-flops and gates at the top-level: propagation delay, rise-time and fall-time. The flip-flops were modified from the ones available in the *ahdlLib* to provide a reset signal, a configurable supply and a choice of input/output levels. The (Early/Center/Late) clock phases were derived from a Voltage-Controlled Delay Line (VCDL) that was part of a DLL in order to maintain a constant phase spacing. A simple CMOS-style delay was created using the *absdelay(.)* function to study its effect on the performance of the original architecture. It should be noted that the delays cannot be chosen arbitrarily and will be dictated by the speed of the target technology.

### 2.2. Digital Filter

The INC/DEC outputs shown in Figure 2 delay/advance the phase and are never asserted simultaneously [7]. This block is clocked by a delayed half-rate clock to provide phase change decisions. Two variants of the filter were tested during the simulation cycle. The first one was the original filter [7] implemented as a circuit structure with behavioral gates and flip-flops. The second variant was a majority voting circuit based on a up/down counter that controlled the phase rotating circuit. Due to the anticipated circuit overhead in the count decoding circuitry, we decided to use the original filter in conjunction with the phase rotator.

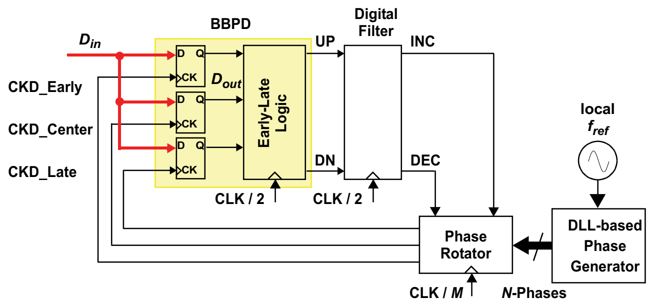


**Figure 2. MRCP-DR architecture block diagram**

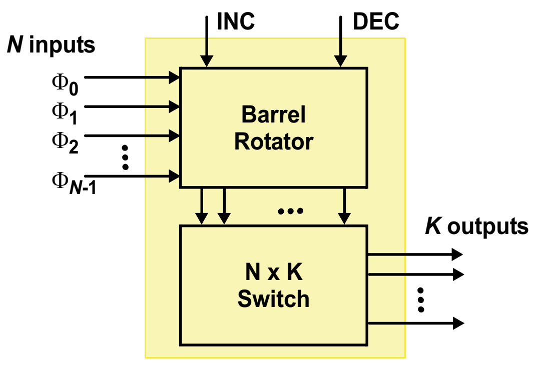


**Figure 3. Phase rotator concept diagram**

### 2.3. Phase Rotator

In order to provide the original architecture with infinite phase tracking ability, one of several available clock phases was chosen using a cyclic phase pointer [7]. This block operates at a much lower frequency (CLK/*M*) where *'M'* is the update interval (measured in number of bits). Therefore, it can use Current-Mode Logic (CML) library cells. For the actual circuit, a (left/right) rotating register made of flip-flops could be used to select one available phase at a time.

The conceptual diagram for a *K*-phase rotator is shown in Figure 3. Consider a barrel rotator [17] that can rotate its contents left or right as a result of the INC/DEC command respectively. One needs to enable any one output (any one but only one from every row) upon start-up to select a single phase. As the jitter accumulates, a decision is made by the BBPD/Digital Filter to either advance/delay (DEC/INC) the phase. A PR block encoded as a behavioral module was used for this purpose. For the modified PR block with the Early/Centre/Late clock phases, the code is a logical extension of the a single PR case and appears in Listing 1. Depending on the direction of the jitter as signaled by INC/DEC, the clock phases are selected on a rotating basis from the eight available DLL phases. The outputs from the PR code are decisions only. The actual switching process would depend on the chosen circuit implementation of the switches.

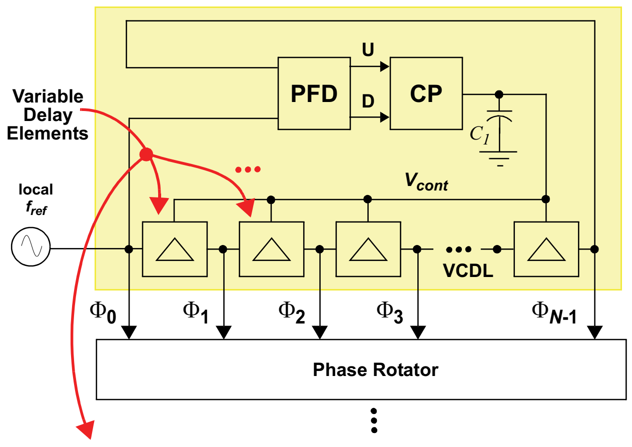
In the current version of the design, dynamic phase spacing between Early/Centre/Late clocks is not targeted, but simple changes can be made to the Verilog-A code/circuit to incorporate this functionality. This would enable the designer to get an estimate of the horizontal eye-opening limited by the phase resolution of the VCDL. An important note is that the switching should happen at the point of the smallest slope of the available phases in order to reduce glitching.

## 2.4. DLL-based Phase Generator

The DLL block diagram is shown in Figure 4 along with a code fragment for the variable delay block. The DLL is a classical structure with a Phase-Frequency Detector (PFD)

**Listing 1. Multiple Clock Phase Rotator (with static spacing)**

```
// ----- Conference     :   BMAS 2005, San Jose, CA, Sept. 22-23, 2005
// ----- Author         :   Syed Irfan Ahmed

`include "discipline.h"
`include "constants.h"
`define MAX_PHASES 8

module ph_rot_new_3x (clk, inc, dec, reset, pose, posc, posl);
input clk, inc, dec, reset;
output pose, posc, posl;
electrical clk, inc, dec, reset, pose, posc, posl;
parameter real tdel_rot = 10p  from (-1:inf);
parameter real tr_rot = 10p  from (-1:inf);
parameter real tf_rot = 10p from (-1:inf);
parameter real vdd_rot = 1.8 from (0:2.5);
parameter real vss_rot = 0  from (-1:2.5);
parameter integer init_count = 1 from (0:32);
parameter phase_step = 2 from (1:4);

real threshold;
integer countc, countl, counte;     // center, late, early

analog begin
   threshold = vdd_rot/2.0;

   @ ( initial_step or cross (V(reset) - threshold, +1) ) begin
         countl  = init_count;
         countc = (init_count + phase_step) % `MAX_PHASES;
         counte = (init_count + phase_step + phase_step) % `MAX_PHASES;

   end      // --------------------------------------- End of initialization

@ ( cross(V(clk) - threshold, +1)) begin

      if (V(inc) >= threshold && V(dec) < threshold && V(reset) < threshold) begin
           countl = countl - 1 ;
                  if (countl < 1)  begin
                        countl = `MAX_PHASES;
                  end
           countc = countc - 1 ;
                  if (countc < 1)  begin
                        countc = `MAX_PHASES;
                  end
           counte = counte - 1 ;
                   if (counte < 1)  begin
                        counte = `MAX_PHASES;
                   end
      end  // --------------------------------------- End of INC operation

      else if (V(dec) >= threshold && V(inc) < threshold && V(reset) < threshold) begin
          countl = countl + 1;
                   if (countl > `MAX_PHASES)  begin
                         countl = 1;
                   end
           countc = countc + 1 ;
                   if (countc > `MAX_PHASES)  begin
                         countc = 1 ;
                   end
           counte = counte + 1 ;
                   if (counte > `MAX_PHASES)  begin
                        counte = 1 ;
                   end
      end   // ---------------------------------------- End of DEC operation

end //     ----------------------------- end cross V(clk)

      //   ------------- Phase change (decision only) ouputs start here

       V(posc) <+  transition (countc, tdel_rot, tr_rot, tf_rot);
       V(pose) <+  transition (counte, tdel_rot, tr_rot, tf_rot);
       V(posl)  <+  transition (countl, tdel_rot, tr_rot, tf_rot);

end     //  ---------------------end analog statements
`undef MAX_PHASES
endmodule
```



```
// ------------ 'analog' section for a Variable Delay Element
// ------------ vddval=supply voltage, static_offset= parameter, real
analog begin
    td = ( (min_delay - max_delay)/vddval) * V(vcont) + max_delay + static_offset;
       V(vout) <+ absdelay(V(vin), td );
end
```

**Figure 4. DLL-based phase generator block diagram (Main code fragment for the variable delay element shown)**

and a Charge Pump (CP). This circuit-like arrangement offers the possibility of injecting noise and jitter at any node without additional modeling complexity. One can simulate the effect of noise and/or jitter on the individual stages, the control voltage node ($V_{cont}$) and the local clock generator. We have also introduced a *static_offset* parameter in the variable delay blocks at the block level for added simulation flexibility. More realistic models included non-linear effects but required a detailed discussion; therefore, a simple linear model is being used here for the variable delay blocks.

## 2.5. Jitter Tolerance Simulation Test Bench

Figure 5 shows the jitter tolerance test bench from [15] that we have included for completeness. Briefly, a jitter sinusoid (JSIN) phase modulates a carrier at the data frequency producing a jittered clock (JCLK). This JCLK signal drives a variable-length Linear Feedback Shift Register (LFSR) generating a Pseudo-Random Bit Sequence (PRBS) of the specified length. This jittered data is subsequently recovered by the MRCP-DR circuit. An ERROR output is generated using a replica-LFSR circuit after lowpass filtering. A novel jitter tolerance simulation technique has also been presented in [15] that significantly reduces simulation times and is applicable to any kind of CDR circuit.

## 3. SIMULATION RESULTS

### 3.1. Assumptions and Functional Verification

The timing parameters for gates and flip-flops have been derived from a 0.18-µm CMOS design [16]; $t_r = t_f = 100$ ps, $T_{cq}$=75 ps for all flip-flops and gate delays=50 ps. The blocks in our model are capable of CMOS or CML input and output

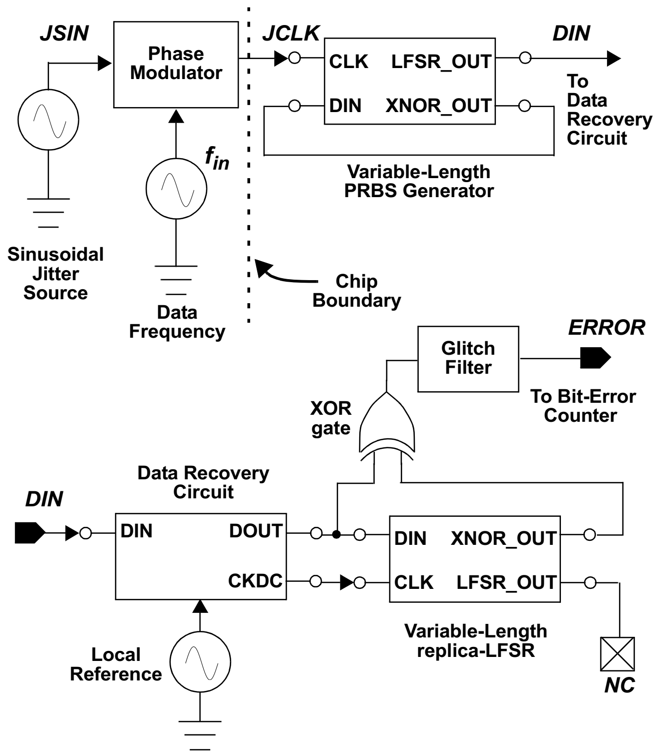


**Figure 5. Jitter tolerance test bench [15]**

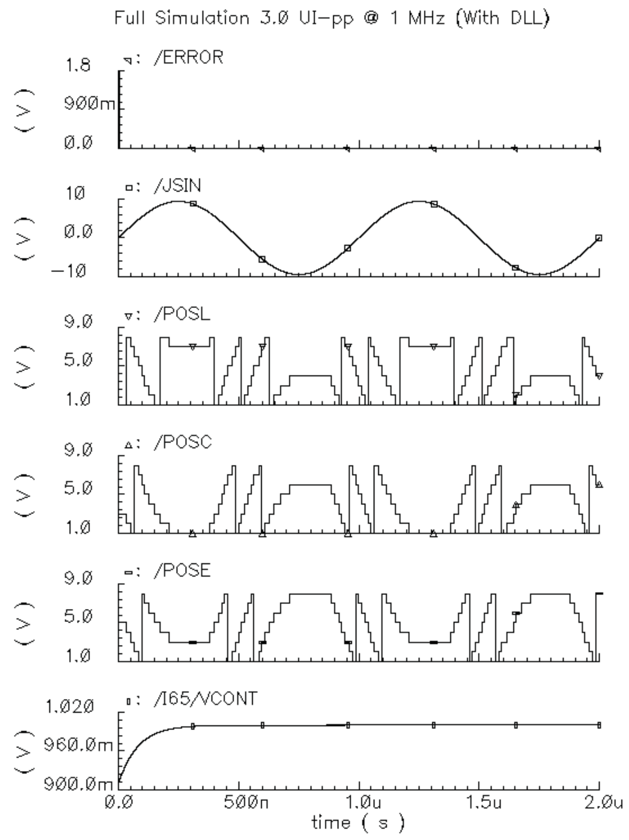


**Figure 6. Functional simulation of the MRCP-DR Circuit (with a DLL-based phase generator)**

levels and we are using these parameters as a guideline only. The update interval ($M$) for the multiple clock phase rotator is 16 bits. The adjacent clock phase separation is constant at 0.25 UI (for an optimal jitter tolerance performance [13]) and the total number of phases ($N$) is eight. The data rate is 2.5 Gb/s and a $2^{11}$-1 PRBS length is chosen.

Figure 6 shows a typical 2.0 μs functional simulation for the MRCP-DR architecture (5000 bits are not shown). The signals POSL, POSC, and POSE represent the phase number of the selected (Late/Center/Early) clock phases respectively. The ERROR signal records no transient bit errors except at start-up as the system tracks a 3.0 UI-pp (Unit Interval, peak-to-peak) jitter causing sinusoid (JSIN) with a frequency of 1.0 MHz. The three clock phases rotate three times in each direction for every cycle of the JSIN signal as the data is recovered. The start-up transient for the DLL is also visible. An initial condition was applied at the $V_{cont}$ node in order to reduce the simulation time (see Figure 4).

The equivalence of the original scheme and the MRCP-DR architecture for jitter tolerance has been verified previously [14]. The interested reader can compare the jitter tolerance results presented later with the ones presented in [7] for a non-rigorous proof of the validity of our models.

## 3.2. Estimated Area and Power Consumption

An advantage of having structural blocks is that the area and power can be estimated relatively easily. With the foregoing assumptions [16], the total power dissipation would be of the order of 15 mW @ 2.5 Gbps for a 0.18-μm design and around 10 mW for a 90-nm design. A conservative estimate puts the area at 120 μm*120 μm for the DR core. Out of this area, approximately 50% is used for the PR circuit and the DLL biasing circuitry. The DLL and its biasing circuitry could be shared between many data lanes on chip, although this remains to be verified. A SKILL language procedure could expedite calculations if the circuit were made up entirely of structural blocks.

## 3.3. Jitter Tolerance vs. PRBS Length

The jitter tolerance simulation results vs. PRBS length as parameter were presented in [15] and will not be repeated here. The same paper briefly comments about the simulation time requirements and the accuracy of BER measurements using transient simulations only. We are using Sun Microsystems Blade-1500 workstations running Solaris 9 with 2 GB of RAM. The simulation time is approximately 5.5/3 minutes with/without a DLL for every microsecond of DR operation.

## 3.4. Jitter Tolerance at Different Data Rates

Here we simulate the case where the same DR circuit is used at data rates of 1.25 Gb/s and 2.5 Gb/s. The top-level parameters for flip-flops and gates remain the same; only the data

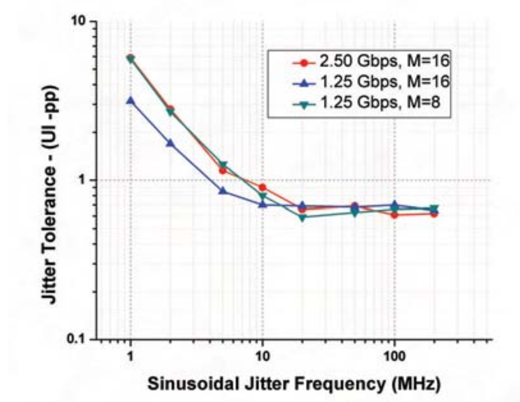


**Figure 7. Jitter tolerance at different data rates**

rate and DLL parameters are changed. Figure 7 shows the relevant simulation results. The CML library cells can be systematically optimized for propagation delay, rise-time and fall-time to work at multiple frequencies. In addition, library cell specifications for upcoming technologies can also be derived rapidly using a flexible top-level test bench.

## 3.5. Effect of Unequal DLL Phase Spacing

To simulate the effect of a biasing problem, we add a static delay offset to one of the variable delay cells in the DLL. The eight ideal phases are separated by 50 ps @ 2.5 GHz. Figure 8 shows that the architecture is tolerant of 20 ps static phase offset in a DLL delay cell. This is because the data detection window is rotating randomly and the effect of one incorrectly biased delay cell is dithered out. Rapid degradations occur in jitter tolerance with offsets beyond 25 ps.

## 3.6. Deviation of Local Clock Frequency

A static frequency offset may result from the design of the local clock synthesizer or a slow variation in temperature. Figure 9 shows the result of the local clock frequency deviation on the jitter tolerance of the MRCP-DR circuit. The POSE, POSC and the POSL staircases (seen in Figure 6) move in one direction only to compensate for the static frequency offset but are not shown due to lack of space.

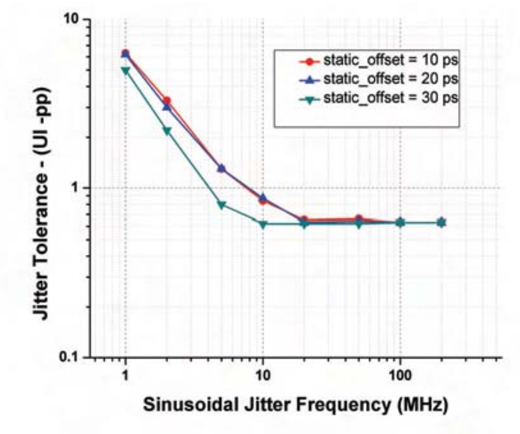


**Figure 8. Effect of unequal DLL phase spacing**

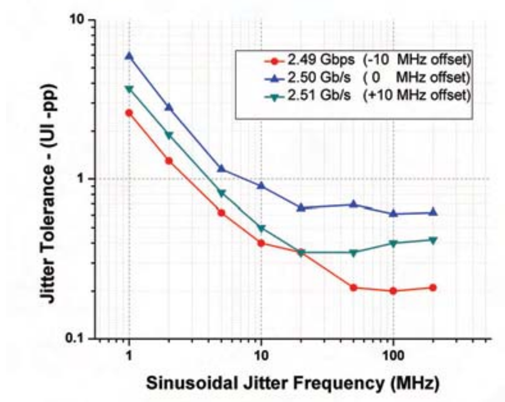


**Figure 9. Effect of local clock frequency deviation**

In a related scenario, the local clock can acquire a correlated high-frequency jitter from the incoming data. To simulate this effect, we apply a jitter of 0.1 UI-pp @ 100 MHz on the local clock while the DR circuit tracks a 3.0 UI-pp wander as shown earlier in Figure 6. Figure 10(a) shows a simulation with an ideal 2.5 GHz local clock. One can observe the bimodal distribution on the histogram of the DOUT signal as

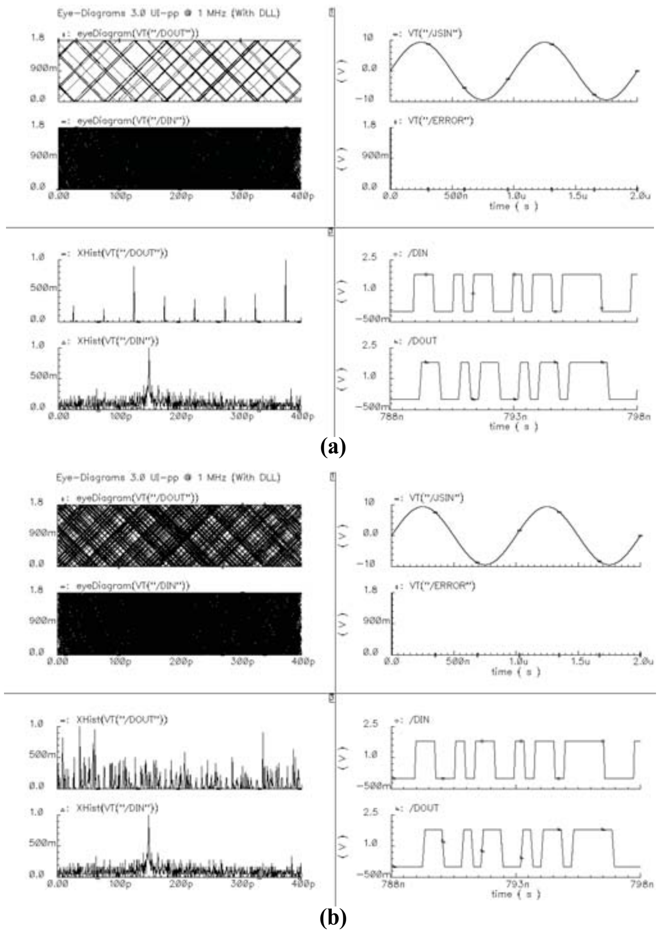


**Figure 10. Effect of high-frequency jitter on the local clock (a) No added jitter (b) 0.1UI-pp sinusoidal jitter@ 100MHz**

the eight ideal clock phases recover it as required. The bimodal distribution is due to the sinusoidal JSIN signal. Figure 10(b) shows the same scenario with the jitter applied. The eye diagram shows almost total closure and the jitter histogram [18] on DOUT is also dithered, but there are no transient bit errors as a result. This simulation can determine the robustness of the DR circuit in a noisy digital environment.

In general, if the accumulation of phase is greater than the phase spacing (0.125 UI=50 ps) during every update period ($M$=16 bits=6.4 ns) or 50 ps/6.4 ns, the BER would degrade rapidly. For this 1st-order Bang-Bang CDR architecture, the jitter contribution from all sources is bounded by this upper limit for a good design.

## 4. CONCLUSIONS

- A Multiple-Rotating-Clock-Phase Data Recovery Circuit was implemented using Verilog-A. The architecture is a variant of [7] and has an all-digital core. The data edge detection interval is established by using rotating clock phases tapped from a DLL-based phase generator and is robust against PVT variations. The local clock generator is blind to the incoming data frequency.
- Functional simulations and Verilog-A code excerpts were presented for the relevant blocks. Initial estimates were made for area and power of the DR circuit.
- Jitter Tolerance simulation results were presented for various data rates and other relevant scenarios. The flexibility of a Verilog-A test bench allows the designer to do what-if analyses at the early architectural stages.
- The MRCP architecture is tolerant of unequal phase spacings in the DLL-based phase generator or from any other clock source since the data detection window is rotating randomly with the incoming jitter.

## ACKNOWLEDGMENTS

The authors would like to thank the Ministry of Science and Technology, Government of Ontario, NSERC, Altera Canada and Carleton University for their financial support. We would also like to extend our gratitude to all colleagues who helped us with their comments and suggestions.

## REFERENCES

[1] B. Razavi, in *Monolithic Phase-Locked Loops and Clock Recovery Circuits, Theory and Design,* IEEE Press 1996

[2] C. R. Hogge, “A Self-Correcting Clock Recovery Circuit,” *IEEE J. Lightwave Tech.,*vol. 3, Dec. 1985, pp. 1312-1314

[3] J. D. H. Alexander, “Clock Recovery from Random Binary Data,” *Elect. Lett.,* vol. 11, Oct. 1975, pp. 541-542

[4] B. Razavi, “Challenges in the Design of High-Speed Clock and Data Recovery Circuits,” *IEEE Comms. Mag.*, Aug. 2002, pp. 94-101

[5] J. Lee, K. S. Kundert, B. Razavi, “Analysis and Modeling of Bang-Bang Clock and Data Recovery Circuits,” *IEEE J. Solid State Circuits*, vol 39, no. 4, Apr. 2004, pp. 613-621

[6] R. Walker, “Clock and Data Recovery for Serial Digital Communications,” *ISSCC Short Course*, Feb. 2002

[7] Y. Miki *et. al*, “A 50-mW/ch 2.5-Gb/s/ch Data Recovery Circuit for the SFI-5 Interface with Digital Eye-Tracking,” *IEEE J. Solid-State Circuits,* vol. 39, no. 4, Apr. 2004, pp. 613-621

[8] S-H. Lee, M.-S. Hwang, *et al.*, “A 5-Gb/s 0.25-μm CMOS Jitter-Tolerant Variable-Interval Oversampling Clock/Data Recovery Circuit,” *IEEE J. Solid-State Circuits*, vol. 37, no. 12, Dec. 2002, pp. 1822-1830

[9] S. Kim, K. Lee, D-K. Jeong, D. D. Lee, A. G. Nowatzyk, “An 800 Mbps multi-channel CMOS serial link with 3x oversampling,” *IEEE Proc. CICC ‘95,* pp. 451-455

[10] C-K. K. Yang, R. Farjad-Rad, M. A. Horowitz, “A 0.5-μm CMOS 4.0 Gbit/s serial link transceiver with data recovery using oversampling,” *IEEE J. Solid-State Circuits*, vol. 33, no. 5, May 1998, pp. 713-722

[11] *Cadence Verilog-A Language Reference,* ver. 5.1, January 2004

[12] *Matlab User Manuals*, ver. 7.0 R14, 2005, Mathworks Inc.

[13] S. I. Ahmed, Tad A. Kwasniewski, “An All-Digital Data Recovery Circuit Optimization Using Matlab/Simulink,” *International Symposium on Circuits & Systems, ISCAS 2005*, May 23-26th, Kobe, Japan

[14] S. I. Ahmed, Tad A. Kwasniewski, “Overview of Oversampling Clock and Data Recovery Circuits,” *Canadian Conference on Electrical and Computer Engineering, IEEE CCECE05,* Saskatoon, May 2005

[15] S. I. Ahmed, Kent Orthner, Tad A. Kwasniewski, “Behavioral Test Benches for Digital Clock and Data Recovery Circuits using Verilog-A,” *Custom Integrated Circuits Conference, CICC 2005*, Sept. 18-21, 2005, San Jose, California, in press

[16] M. Usama, Tad A. Kwasniewski, “Design and Comparison of CMOS Current Mode Logic Latches,” *International Symposium on Circuits & Systems, ISCAS 2004*, vol. 4, pp. 353-356

[17] K. Martin, *Digital Integrated Circuit Design*, Oxford University Press, ISBN 0-19-512584-3, 2000

[18] Using ‘Calculator Functions v 0.71 050429’, an add-on to the Cadence Calculator, © 2002-2005, advICo Microelectronics GmbH, Germany

## APPENDIX A

### Verilog-A vs. Matlab

This architecture was initially conceived and explored using Matlab/Simulink ([13], [14]). Matlab is a proven tool for the architectural exploration of DSP/Communications/Control systems. However, a Simulink model using built-in blocks for logic gates and flip-flops works to an un-realistically high frequency unless delay blocks are manually inserted. The simulations can thus become more accurate, but the rise- and the fall-times are still not modeled easily.

Consider an output stage with a load capacitance (due to geometry, fan-out or parasitics) that causes an *RC*-style delay through the stage. The current available to charge/discharge this load capacitance determines the rise-time/fall-time of this stage. A circuit-aware language, like Verilog-A, is required for simulating these circuit-level effects. It has built-in definitions for *flow*/*potential* quantities (current/voltage, for the *discipline ‘electrical’*). With its *transition* and *slew* waveform filters, Verilog-A/(AMS) can simulate such post-layout effects during the early design stages. It, therefore, provides a realistic transition between concept-stage modeling and the final chip. Multi-domain simulations and compatibility with major vendor supported IC-design flows make it a natural choice for mixed-signal, Systems-on-Chip (SoC) development as well.