Delay Window Blind Oversampling Clock and Data Recovery Algorithm with Wide Tracking Range

Travis Bartley*, Shuji Tanaka*, Yutaka Nonomura†, Takahiro Nakayama‡ and Masanori Muroyama*
*Microsystem Integration Center, Tohoku University, Sendai, Miyagi, Japan
†Power Electronics Research Division, Toyota Central R&D Labs., Inc., Nagakute, Aichi, Japan
‡Partner Robot Division, Toyota Motor Corp., Toyota, Aichi, Japan

***Abstract*—A new blind oversampling clock and data recovery (BO–CDR) algorithm is proposed. It has high tolerance to low–frequency jitter (14.8 unit intervals at 10 kHz, measured at 640 Mbps) and is suitable for systems where the receiver clock has high drift with respect to the transmission. The algorithm is capable of recovering data over a wide tracking range or when the precise oversampling rate (β) is not known a priori, for any real–valued oversampling rate, $\beta \geq 3$, making this BO–CDR algorithm the first to not require integer–valued β. To demonstrate the utility of the algorithm, two implementations are designed and evaluated. The first is used in a low–power, low–data rate sensor node IC with a low–performance single phase clock source. The second is a high–speed receiver with a multiple phase clock source implemented on FPGA. The CDR core consists of just 47 logic cells and 19 registers and has an estimated power consumption of 0.70 mW at 640 Mbps. The properties of this CDR algorithm make it appropriate for a wide range of applications in serial communication.**

I. Introduction

Clock and data recovery (CDR) is an essential operation in serial communication. Of the various types of CDR architectures, blind oversampling based CDRs offer some advantages. First, data recovery can be provided with low latency because data sampling can lock to the serial signal almost immediately [1]. Oversampling CDRs can also achieve data rates of multiple Gbps, and are inherently stable. They are applicable to both burst–mode and continuous–mode data transmission, and can be implemented using digital cells alone, making BO–CDR designs portable and easy to implement.

Existing blind oversampling CDR (BO–CDR) schemes begin with the assumption that the sampling rate is an exact multiple of the transmission data rate. Two well–known BO–CDR algorithms are direct phase picking (DPP) and averaged phase picking (APP) [2]. A sampling rate of $3R$ or $5R$ is common, where R is the bit rate. The decision window is always β input samples wide, where β is the oversampling rate (integer-valued in this case). This means that 1 bit of data is recovered every β input samples, without exception. There are a few drawbacks of recovering data in this manner.

If there is a slight mismatch between the transmission rate and the sampling rate, there will be a proportional rate of bit slips. This requirement for an accurate frequency match means that these CDR schemes are sensitive to drift or low–frequency jitter in either the clock or data signals. Also, the minimum achievable bit error rate (BER) is limited by the accuracy of both transmitter and receiver clocks. This sensitivity to timing variations implies that these algorithms have narrow tracking ranges. They can perform well in basic laboratory tests with frequency-matched reference clock and data sources, but will have high BER in real-world environments where the reference clock and input data drift with respect to each other. This is a critical drawback in almost all BO-CDR algorithms that is rarely mentioned in the literature. There has been a BO–CDR developed to overcome this problem [3]. However, it requires $\beta = 5$. This means that about 67% more samples must be processed for the same amount of recovered data as compared to $\beta = 3$, with a proportional impact to resource consumption and the maximum data rate. Also, the maximum packet size is inversely proportional to the frequency offset between the transmitter and receiver clocks.

The second drawback to the existing BO–CDR systems is that there is no flexibility in the oversampling rate. β is determined at design–time and cannot be reconfigured without changing the clock speed. The third limitation is that these systems only function correctly when β is an integer value. On the other hand, an analog phase–locked loop (PLL) CDR can be designed to function correctly for drifting data, or for a wide tracking range. These are significant drawbacks for BO-CDR that are not present in PLL CDR circuits. However, it is noteworthy that their limitations are not due to some inherent property of digital systems, but are due to the algorithms themselves. The algorithm proposed in this article avoids these drawbacks by using variable–length delay windows to track the bit boundary of the input data. Section II describes the delay window CDR algorithm at the abstract level. Section III provides implementation-specific information. Section IV describes the experimental setup and Section V provides the experimental results. The paper is concluded in Section VI.

II. Algorithm

Existing BO–CDRs track the bit boundary based on the timing of the data transitions, but they are limited to how far they can track with respect to the reference clock due to the fact that the decision window has fixed width. The proposed delay window (DW) CDR algorithm also tracks the incoming data based on edges. However, the width of the DW can be changed to adapt to the input stream. Whenever an edge is detected or a DW expires, a new DW is started. The duration of the DW is calculated based on a function which will be explained below.

If the DW ends before an edge is detected, this means that the data did not transition in the expected time frame. As a result, 1 bit of data is recovered, and a new DW is set. When an edge does occur, this event immediately triggers the circuit to store 1 bit of data and start a new DW, forcing the current DW to expire. By tracking the data in this manner, the DW will remain phase–locked to the incoming data stream, even in the presence of significant data or clock drift. Rather than having a decision window of fixed width, as with APP and DPP, a DW of variable width offers a significant advantage: it is not locked to the reference clock and can drift early or late to track with the input data.

A. Single Phase Sampling Clock

Based on this idea, the architecture for the DW CDR was developed (Fig. 1). M is defined as the number of input samples processed per core clock cycle. In the case of a single phase sampling clock ($M = 1$), the signal widths of d, e, and $phasesel$ (defined below) are all 1 bit each. The input data, d^*, must be synchronized to the core clock (clk_{rx}) by a synchronizer (in the single phase case) or a sampler (in the multiple phase case) [4]. d^* is routed to the inputs of a register and XOR gate for edge detection. The $timer$ signal is used to determine when a DW expires, and is decremented once per clock cycle. The logic that drives the $timer^*$ signal takes the current $timer$ value, the number of DWs since the last edge, p^* (where $p^* = 0, 1, 2, ...$), and the edge signal, e, as input (Fig. 2). If an edge occurs, p^* is set to "0" and the $timer$ register is loaded with a value corresponding to the duration of the next DW. If either $timer$ reaches "0" or an edge occurs, the $phasesel$ signal indicates to the data extraction circuit that a new bit, d_0, is ready to be stored. By starting a new DW when an edge is detected, the timing information in the input is used to correct the DW phase–lock. A suitable reset value for $timer$, p and $phasesel$ is 0. Even with no reset, these variables will enter known states after an edge is detected.

B. Multiple Phase Sampling Clock

To increase the bit rate of a digital CDR circuit, the first option might be to increase the clock frequency. However, there is a limit to how much the clock frequency can be increased. Another way to increase the bit rate is for the CDR to process more samples per clock cycle. This is a well–known principle of BO–CDR, and a generic architecture including a sampler, phase selector, data extractor and output fifo has previously been established (Fig. 3). Based on this, a multiple phase DW CDR was also implemented. The sampler used was a simple shift register–based circuit with M^2 D flip–flops [5]. The multiple phase architecture resembles that of the single phase, except that the $timer$ reload and decision logic must process multiple input bits each clock cycle.

C. Delay Window Duration Function

The function used to determine the DW durations is critically important to the performance of this scheme. If the function is too inaccurate, a DW may expire too early, causing

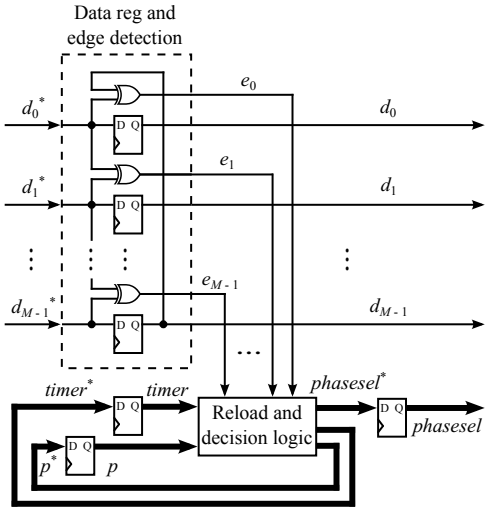

Fig. 1. Architecture for delay window CDR core, generalized for both single phase and multiple phase operation. All registers shown are in the core clock domain (clk_{rx}). The most significant bit in d is the newest sampled data, and the least significant bit is the oldest.

for $i = M - 1$ to 0 **do**
  if $e_i == 1$ **then**
    $p^* \leftarrow 0$
    $phasesel_i^* \leftarrow 1$
    $timer^* \leftarrow T_{p^*} - 1$
  else
    if $timer == 0$ **then**
      $p^* \leftarrow p + 1$
      $phasesel_i^* \leftarrow 1$
      $timer^* \leftarrow T_{p^*} - 1$
    else
      $p^* \leftarrow p$
      $phasesel_i^* \leftarrow 0$
      $timer^* \leftarrow timer - 1$
    end if
  end if
end for

Fig. 2. Register reload and decision logic pseudocode.

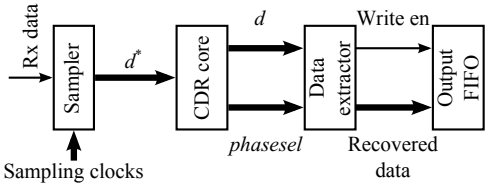

Fig. 3. Architecture for overall BO–CDR circuit.

1 bit to be repeated, or expire too late, causing 1 bit to be deleted. Theoretically, the DW duration could be calculated based on whatever factors the designer determined to be important. The tradeoff is that the complexity of the $timer$ reload and decision logic impacts the maximum frequency of the circuit, as well as the size and power consumption.

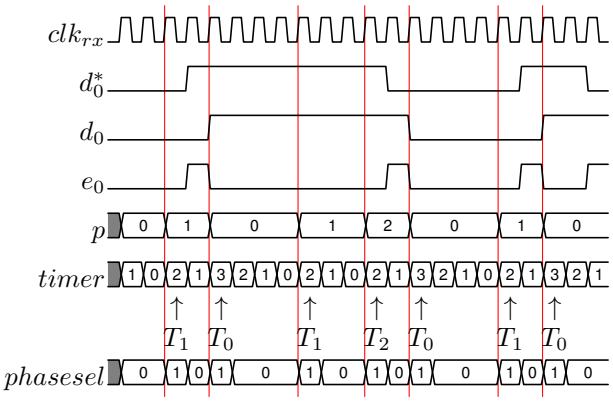

Fig. 4. Example timing diagram for single phase DW CDR with $\beta = 3$. Vertical lines show where new DWs begin.

For the implementations presented in this paper, a decision function was developed which has excellent jitter tolerance, while also having a low impact to circuit complexity. In this case, the DW width is determined by two factors: β and p^*. The initial duration (in input samples) of a DW is defined as T_{p^*} according to the equation below, with an example in Fig. 4.

$$T_{p^*} = floor(1.5\beta) \text{ for } p^* = 0$$

$$T_{p^*} = floor((p^* + 1.5)\beta) - floor((p^* + 0.5)\beta) \text{ for } p^* > 0$$

This equation was derived by determining the times when subsequent edges are expected to occur after an initial edge $(1\beta, 2\beta, ...)$. The midpoints between these times were determined, and this equation sets the DW to expire near these midpoints $(1.5\beta, 2.5\beta, ...)$. Any DW that begins after an edge must be approximately 1.5β samples long in order to end at the next midpoint. Any DW that does not begin after an edge must be approximately β samples long to end at the next midpoint since it starts near a midpoint. However, if $T_{p^*} = floor(1.5\beta)$ for $p^* = 0$ and $T_{p^*} = floor(\beta)$ for $p^* > 0$ were used for delay window duration function, the algorithm would quickly accumulate timing errors for non–integer values of β. The problem is that β is a real number, but the width of the DW can only be an integer number of samples. This causes a rounding error to be introduced into the DW timing. The more the error accumulates, the more likely a bit slip will occur. For this reason, the calculation of T_{p^*} must take into account the timing error caused by rounding from previous T_{p^*} calculations.

The equation given for T_{p^*} accounts for this. For example, suppose $\beta = 3.5$, and an edge occurs, followed by many samples with no input transition. The T_{p^*} sequence will then be $5, 3, 4, 3, 4, ...$, with 3 and 4 continuing to repeat. Because the average of 3 and 4 is 3.5, this sequence will not result in rounding error accumulation over time. The function has been derived such that that timing errors are minimized for all real values of β. It is also straight–forward to implement this function in hardware. The floor function is performed by truncating the fractional bits of the operation. Also, it is possible to use only integer operations by shifting the binary values of the variables left before the calculation, and shifting the binary result right by a corresponding amount. For example, if β is stored as a binary number with 5 integer bits and 3 fractional bits, this is identical to storing an 8 bit number with value 8β. The 8β value is then operated on using integer operations, and the result is divided by 8 by truncating the three least significant bits.

D. Oversampling Rate Estimation

If the exact frequency of the receiver clock or transmission rate is not known, the oversampling rate must be estimated for CDR. For this reason, a training sequence or preamble can be used to determine the oversampling rate. Once the beginning of the preamble is detected, the receiver begins counting clock cycles. By the end of the preamble, the receiver has counted the duration of the preamble in receiver clock cycles. If the length of the preamble in unit intervals (UI) is also known, β can be calculated as the number of receiver clock cycles divided by the length of the preamble in unit intervals. In this way, an estimation of β can be made during the preamble, which allows successful data recovery of the remainder of the packet.

III. Implementation

The DW CDR was implemented for two different applications to demonstrate the broad flexibility of the algorithm. The first implementation was for a receiver in a low–power sensor node system [6]. The system consists of several sensor nodes on a shared bus. Each sensor node circuit has its own on–chip RC-based clock generator. The sensor node clocks are therefore sensitive to process, voltage and temperature variations as well as noise. Not only is the clock rate expected to not precisely match the designed frequency, but it is also expected to change depending on operating conditions. Likewise, communication between circuits is expected to have drift between the transmitter and receiver. Meanwhile, the data rate requirement of the system is relaxed at 50 Mbps. Because the data rate is relatively low, a single phase receiver was used to reduce circuit size and complexity. Transmission data is sent in a formatted packet, which has a 9 bit preamble for β estimation, as described above. The packet is encoded using 4B5B and NRZI to ensure that the transmission is run-length limited, which increases the reliability of communications.

The designed tracking range was $3 \leq \beta \leq 9$, and simulation testing demonstrated correct functionality over this range. The sensor node IC was fabricated in the TSMC 180 nm CMOS Mixed Signal Process and assembled together with a sensor node bus ribbon and a relay node FPGA. The communication systems including the CDR core functioned as designed for both the relay node and sensor nodes. Because of the highly integrated design of the sensor node IC, it was not possible to perform in–depth evaluation of the performance of the CDR.

IV. Experiment

For a more critical analysis of the algorithm's performance, a second implementation was developed on a Cyclone IV

FPGA EP4CGX150CF23C7N on the HuMANDATA ACM–024C and ZKB–105C boards. The architecture was designed as in the "Multiple Phase Delay Window CDR" section above, and comparable phase selection circuits for APP (shift register depth: $W = 12$) and DPP were used for direct comparison. The bit rate was 640 Mbps. Although the internal circuit was able to reach a higher data rate, the speed was ultimately limited by the general purpose low–voltage differential signaling I/O of the FPGA.

Each of the three algorithms were designed with $\beta = 3$ and $M = 12$. The FPGA board was then connected by SMA cables to the Agilent N4903B J bit error rate tester (BERT) for jitter tolerance testing. The N4903B has calibrated and integrated jitter injection which was used to add sinusoidal jitter (SJ) and periodic jitter (PJ) with sinusoidal characteristic to the test data. The use of SJ and PJ depends on the jitter frequency, and the BERT switches automatically between the two sources. The SJ and PJ were both injected into the data signal which was sent to the design under test (DUT). The reference clock sent to the DUT was not distorted by any jitter source, and the data signal was injected with SJ and PJ.

V. Results

The results of the jitter tolerance tests of DPP, APP and DW CDR are plotted together (Fig. 5). The maximum magnitude of jitter that the BERT can provide is also shown. The target BER was 10^{-9} with 95% confidence level, and the data pattern used was a $2^{31} - 1$ pseudorandom binary sequence. Both DPP and APP jitter tolerance remains well below 1 UI throughout the testing range. DW CDR, on the other hand, is able to track the input when it is injected with low frequency jitter, with a maximum jitter tolerance of 14.832 UI. Overall, DPP has the worst jitter tolerance and DW CDR has the best. In terms of size and power, DW CDR and DPP have approximately the same impact (table I). APP, with its much higher computational complexity and size, consumes the most power. FPGA and CDR core power estimations were calculated using the Quartus II PowerPlay Tool.

VI. Conclusion

The DW CDR algorithm was developed and evaluated. Through analysis, it can be seen that it has several advantages over existing BO–CDRs. First, it has a flexible oversampling range. It can be designed for any real oversampling rate at or above 3. Also, the tracking range can be made arbitrarily wide since the maximum oversampling rate can be arbitrarily high. This means a single circuit can be designed to operate over a range of bit rates without the need for redesign or change in reference clock frequency. It is also highly tolerant to low–frequency SJ and PJ or drift. These are significant advantages over DPP, APP and other BO–CDR algorithms. DW CDR also retains the benefits of other BO–CDR algorithms due to its all–digital nature. It can be easily transferred between process technologies and designs, and is low–sized and energy–efficient. The greatest advantage of the delay window CDR algorithm is its flexibility. As demonstrated, it can be designed for low–power systems with high timing variability, as well as high–speed systems with more accurate clocks. Trade–offs between oversampling rate range, BER, data rate, size, power and complexity requirements can be addressed by selecting appropriate design parameters.

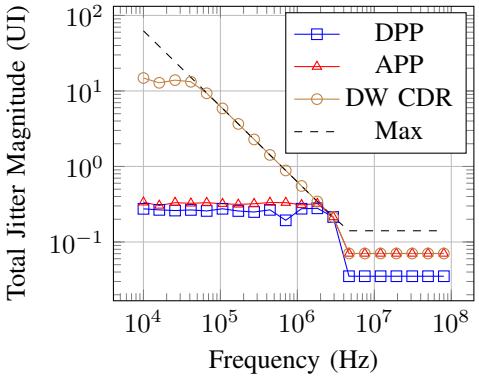

Fig. 5. Jitter tolerance for sinusoidal and periodic jitter. The proposed algorithm (DW CDR) shows significant improvement in low-frequency jitter tolerance over compared algorithms.

TABLE I
Comparison of CDR algorithms

Algorithm	DPP	APP	DW CDR
Logic cell combinationals	22	386	47
Logic cell registers	20	438	19
Board–level current at 3.3 V (mA)	293	297	293
Est. FPGA power (mW)	169.4	173.80	166.69
Est. CDR core power (mW)	0.57	6.75	0.70

Acknowledgment

This study was performed in the R&D Center of Excellence for Integrated Microsystems, Tohoku University under the program "Formation of Innovation Center for Fusion of Advanced Technologies" supported by Special Coordination Funds for Promoting Science and Technology. The authors would also like to thank VDEC at the University of Tokyo for supporting the evaluation.

References

[1] M.–T. Hsieh, and G. Sobelman "Architectures for Multi–Gigabit Wire–Linked Clock and Data Recovery," *Circuits and Systems Magazine, IEEE*, vol. 8, no. 4 , pp. 45–57, Dec, 2008.

[2] J. Kim, and D.–K. Jeong "Multi–Gigabit–Rate Clock and Data Recovery Based on Blind Oversampling," *Communications Magazine, IEEE*, vol. 41, no. 12 , pp. 68–74, Dec, 2003.

[3] S.–H. Park *et al.*, "A Single–Data–Bit Blind Oversampling Data-Recovery Circuit With an Add–Drop FIFO for USB2.0 High–Speed Interface," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 55, no. 2, pp. 156–160, Feb. 2008.

[4] C. Cummings, "Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog," *Synopsys Users Group*, (Boston, USA), Sep. 2008.

[5] N. Sawyer "Data Recovery," *Xilinx Application Note*, no. 224, Sep. 2000.

[6] M. Muroyama *et al.*, "Tactile Sensor Network System with CMOS–MEMS Integration for Social Robot Applications," *Smart Systems Integration International Conference and Exhibition*, (Vienna, Austria), Mar. 2014.