A DLL-based Clock Data Recovery with a modified input format

Tae-Ho Kim, Sang-Ho Kim, and Jin-Ku Kang^{a)}

School of Electronics Engineering, Inha University,
253 Yonghyun-dong, Nam-Gu, Incheon 402–751, Korea

^{a)} jkang@inha.ac.kr

Abstract: This letter presents a DLL (Delay Locked Loop)-based CDR (Clock Data Recovery) design with a modified input data format. The proposed CDR recovers the clock and tracks the phase by the proposed training and real data patterns. The proposed input data formatting is done by inserting the ‘01’ pattern in every N -bit data. To prove the feasibility, a 2.4 Gbps CDR is designed and simulated. The training and the real data pattern were formatted as the 10B12B for a high-performance display interface. The CDR achieves less jitter due to the DLL structure. The proposed CDR with the 10B12B format consumes approximately 8 mA under 3.3 V power supply using 0.25 μm CMOS process.

Keywords: Clock and Data Recovery, DLL-based CDR, 10B12B format, CMOS

Classification: Integrated circuits

References

- [1] K. Yamaguchi, “A 2.0 Gb/s Clock-Embedded Interface for Full-HD 10 bit 120 Hz LCD Drivers with 1/5-Rate Noise-Tolerant Phase and Frequency Recovery,” *IEEE Int. Solid-State Circuits Conf.*, pp. 192–194, Feb. 2009.
- [2] X. Maillard, “A 900-Mb/s CMOS Data Recovery DLL using Half-Frequency Clock,” *IEEE J. Solid-State Circuits*, vol. 37, no. 6, pp. 711–715, June 2002.
- [3] A. L. Coban, “A 2.5–3.125-Gb/s Quad Transceiver with Second-Order Analog DLL-Based CDRs,” *IEEE J. Solid-State Circuits*, vol. 40, no. 9, pp. 1940–1947, Sept. 2005.

1 Introduction

In high-speed serial interface applications, conventionally a PLL (Phase Locked Loop)-based CDR is mostly used [1]. However, the PLL-based CDR consumes more power and generates more jitter due to the VCO (Voltage Controlled Oscillator). In addition, the PLL-based CDR requires the 8B10B encoding scheme for providing DC balance and the ample transition edges

for clock recovery. Therefore additional 8B10B decoding, comma detection and word alignment blocks are required. On the other hand, the DLL-based CDR usually consumes less power than PLL-based CDR. Also, the VCDL (Voltage Controlled Delay Line) in the DLL-based CDR does not accumulate jitter. However, conventional DLL-based CDRs require the external reference clock [2, 3]. Therefore, an additional clock line is needed between the transceiver and receiver.

The proposed DLL-based CDR uses a single data line with the proposed input data format and without an external clock. The proposed CDR recovers the clock and tracks the phase by the formatted training and real data patterns. In this 2.4 Gbps CDR design, the training data and real data pattern were formatted by inserting '01' in every 10 bit input data.

2 Architecture and operation

Fig. 1 (a) shows the proposed serial interface block and Fig. 1 (b) illustrates the block diagram of proposed DLL-based CDR. The transmitter (TX) block has the encoder and the serializer. The main block of the receiver (RX) has the proposed CDR. The proposed DLL-based CDR consists of clock recovery unit and data recovery block. In this design, the training data and real data pattern were formatted by inserting '01' in every 10 bit input data. The proposed 10B12B data format is applied since the circuit is designed for the high-definition display interface application with a gray level of 10 bit. For 2.4 Gbps CDR design with the proposed data format, the clock recovery unit needs to make a 200 MHz 12-phase clock (phase difference = 417 ps = 1 bit period of 2.4 Gbps). The data recovery block recovers the data and clock using 12-phase clock. Depending on targeted high-speed interfaces, the demultiplexing structure, the clock rate, number of multiphase clocks (Since input data is formatted by inserting '01' pattern in every N -bit, the number

Fig. 1. Proposed block diagrams and input data format
(a) Serial interface (b) DLL-based CDR (c) Input Data Format

of multiphase clocks to be generated is $N + 2$), and the input data rate can be varied for the optimal design conditions.

The formatted input patterns for the CDR are shown in Fig. 1 (c). The proposed formatting scheme is to insert the 2 bit ‘01’ pattern in every 10 bit data. The training pattern is actually a clock signal to be recovered in the CDR. The duty ratio of the training pattern is not a concern as far as the pattern provides the signal transition information. The formatted data is transmitted at 2.4 Gbps for handling actual 2 Gbps data rate. The proposed 10B12B format scheme can be expanded to $NB(N + 2)B$. If the N is increased, the data formatting overhead is decreased. But the number of delay cell in VCDL increases. So, the power consumption and the jitter are affected.

The input data transmission is done with two steps of operation and each step sends different data patterns. At the beginning of data transmission, the TX sends the formatted training pattern for a certain period. During the training period, the CDR in the receiver circuit is virtually operating as a DLL. Thus the proposed CDR generates a recovered clock and acquires the lock state of the clock in the training pattern period. After the TX sends the formatted training data pattern, the TX switches to send the formatted real data pattern. During the real data pattern period, the CDR still recovers the clock by tracing the inserted ‘01’ pattern continuously, and recovers the data. Transmission period of training pattern and real data pattern is controlled by the data control signal in the TX.

3 Clock and Data Recovery Unit

Fig. 2 (a) shows the clock recovery unit of the proposed CDR. While the TX is sending the training data pattern and path control signal ‘0’. The path control signal connects to the serial data stream (training data pattern) to VCDL. Then the clock recovery unit operates like a DLL. So, the training pattern is acting as a 200 MHz reference clock signal. Through the DLL operation, the clock recovery unit acquires the lock state. The lock state is achieved within hundreds of nanoseconds. Though the DLL goes to the lock state, the path maintains the pure DLL operation while the path control signal stays ‘0’. After training pattern period, the TX transmits the real data pattern and the path control signal goes to ‘1’. Then the MUX receives signal from the pulse generator. The pulse generator block extracts the clock and keeps tracking the clock phase by the inserted ‘01’ pattern. The path control signal is related with the data control signal in the transmitter. The window signal is produced for masking random data patterns and extracting only the signal transition (‘01’ pattern) information. While the formatted random input data patterns are provided, the clock recovery unit keeps extracting the transition information and generates the clock signal. The clock recovery unit makes the 12 multi-phase clocks by the VCDL. The data recovery block recovers the data using the 12 multi-phase clocks.

Fig. 2 (b) shows the ideal timing diagram of the window and pulse sig-

Fig. 2. (a) Block diagram of the clock recovery unit
(b) Timing diagram of window and pulse generator under lock state
(c) Block diagram of the data recovery

nal. The window signal is generated using the two multi-phase clocks (11th and 1st clock signals) by tapping two signals from the VCDL. The window generator can be realized by a single D-F/F with a reset. The window signal is marching with the inserted '01' pattern on the data stream. The pulse generator generates the 200 MHz clock by the combination of the window signal and the inserted '01' pattern. The pulse generator consists of two D-F/Fs and two delay cells. The first D-F/F of pulse generator makes about 0.9 ns pulse width by the window signal and the positive edge of inserted '01' pattern in every 5 ns. This pulse becomes enlarged to 2.3 ns pulse width by the second D-F/F. But if the delay is increased more than 2.5 ns, the next positive edge will be not generated by the feedback reset. So, the optimal pulse is generated with the 45% duty cycle. The duty ratio is not critical in this design because the data recovery is done with only rising edge of the clock. The critical point is the precise window position control relative to the '01' inserted pattern timing in the input data. In order to make sure that the window is positioned in the right place, a 'sliding window scheme' is

implemented. The scheme generates four different windows for capturing the ‘01’ inserted pattern under PVT variation. The four windows can be shaped by different combinations of multi-phase clock outputs from DLL. The optimal window is selected by measuring the relative position between the ‘01’ pattern and the windows.

Fig. 2 (c) shows the data recovery block. The data recovery starts to recover the data when the path control signal goes to ‘1’. The data recovery block consists of twenty one D-F/Fs and twelve buffers. The data recovery block recovers the 10 bit real data in parallel using the 10 different phase clocks of the 12 multi-phase clocks generated from the clock recovery unit. The D-F/F₁~10 recover the each data and the D-F/F₁₁~20 align the recovered data. The D-F/F₂₁ generates the start-end signal which indicates the start and end of recovered real data. The inserted ‘01’ pattern is not recovered. Thus the word alignment scheme is easily implemented without an additional logic compared to conventional 8B10B word alignment.

4 Simulation result

The proposed 2.4 Gbps CDR with the 10B12B data formatting has been designed and simulated using 0.25 μm CMOS. Fig. 3 (a) shows the simulated window, pulse signal from a serial data stream. The pulse width of window signal is approximately the 2 bit size of the serial data (about 0.9 ns). The frequency of the generated pulse signal is 200 MHz. Since the data recovery is done by only rising edge of the recovered clock, the duty ratio is not a concern. In order to investigate the intersymbol interference (ISI) effect, the prototype circuit includes an equalizer at the input. The input path can be connected to input buffer directly or through the equalizer by a digital switch. The ISI effect can be evaluated by measuring the data performance with the equalizer and without the equalizer. To obtain the simulation results on ISI effect, a modeled channel was included. The simulation shows any degradation of data recovery performance with ISI effect was not detected up to 2.7 Gbps data rate. Fig. 3 (b) shows the recovered data and the start-end signal, when the training pattern and the real random data pattern period are repeated. When the start-end signal goes to ‘0’ (training pattern period), the real data are not recovered. And if the start-end signal goes to ‘1’, the real random data are recovered. In the simulation, the TX transmitted the training pattern for 500 ns and then the TX sends the real data pattern for 1.5 μs , repeatedly.

The width of window signal tracking the inserted ‘01’ pattern has ideally 1UI. The window signal itself could have a jitter caused by DLL loop. Simulations show the peak-to peak jitter of recovered clock is 35 ps with the formatted real data pattern. In conventional serial high-speed interface, the transmitted data is usually modulated by a spread-spectrum clock for mitigating EMI. Then, the loop bandwidth of the DLL in the CDR should be designed carefully based on the modulation ratio and modulation frequency. The jitter tolerance of the proposed CDR is dependent on the DLL loop bandwidth and the recovered clock jitter. The proposed CDR used 3.3 V

Fig. 3. Simulated waveforms (a) Generated window and pulse signal (b) Retimed data

power supply and consumed approximately 8 mA current on simulation.

5 Conclusion

This letter presents a DLL-based CDR design with a modified input data format. The proposed CDR recovers the clock and tracks the phase by the training and real data patterns. The proposed input data formatting scheme is done by inserting the ‘01’ pattern every N -bit input. A 2.4 Gbps CDR is designed and simulated based on the proposed scheme. The training and the real data pattern were formatted to support high-performance display interfaces. The proposed 10B12B formatting scheme has a formatting overhead, but needs a simple word alignment operation. Thus the CDR shows less jitter and occupies the smaller chip area. The proposed DLL-based CDR consumes approximately 8 mA under 3.3 V power supply using 0.25 μm CMOS process.

Acknowledgments

This work is supported by the Ministry of Knowledge Economy (MKE) and Korea Industrial Technology Foundation (KOTEF) through the Human Resource Training Project for Strategic Technology. Authors also thank IT-SoC program from ETRI and the IDEC program.