Implement a simple digital-serial NRZ data-recovery algorithm in an FPGA

Jef Thoné and Bob Puers, ESAT-MICAS, Katholieke Universiteit Leuven, Leuven, Belgium; Edited by Martin Rowe and Fran Granville - June 11, 2009

Serial-data links embed clocks in their data streams, and those clocks must be recovered at the receiver end. This Design Idea describes a data/clock-recovery algorithm for an NRZ (non-return-to-zero), 1.5-Mbps data stream in a Xilinx Spartan XC3S200 FPGA. The algorithm employs a modified data-recovery application note (Reference 1). The application note uses the DCM (digital-clock manager) on the Xilinx Spartan and Virtex models, but this application uses a simplified algorithm that compares the data edges, if any, with internally generated clock edges, dynamically changing the data-input-to-data-output delay. The simplified algorithm allows integration in smaller CPLDs or FPGAs that lack a DCM (Figure 1).
Figure 1 A clock-recovery circuit in an FPGA recovers data in a 1.5-Mbps data stream.

The algorithm uses a 3-bit, free-running counter to generate the output clock, an 8-bit shift register to sample the serial data, seven XOR ports for edge detection, a 7-to-1 multiplexer with decoding for multiplexing the right-shift-register bit to the output, and some buffering registers. The algorithm runs at eight times the serial-data-stream speed, without a known phase relationship between both. It clocks the data into the shift register, which implies that, after eight clock cycles, the shift register will contain a rising edge; a falling edge; or, when the input data remains the same, no edge. The multiplexer does not take into account cases in which the shift register contains no edges or more than one edge.

The edge location is checked in the shift register using the XOR-port array, which compares shift-register bit 0 with bit 1, bit 1 with bit 2, and so on. Depending on the output of the XOR array, showing where the edge occurs, a certain bit of the shift register multiplexes to the output. This action ensures that the output clock always toggles around the middle of the output-data bits.

When there are slight differences in clock speed and serial-input-data speed—for example, in the case of clock jitter or clock tolerances—the data-input phase continuously changes with regard to the output-clock phase as the algorithm tries to track the input-data phase. In this case, the multiplexer has an overflow, which happens when shift-register bit 7 multiplexes to the output, the next bit is shift-register bit 1, or vice versa.

If bit 7 is output first—that is, the signal edge_select is 0100 0000—and the next selected bit is bit 1, with an edge_select of 0000 0001, a sudden phase jump in output data occurs. This phase jump is
-360°×7/8, or -315°. Because the next input-data bit already had shifted in completely in the shift register, you need to employ a double-output clock once, so that the register doesn't miss a data bit (circled area in Figure 2).

![Figure 2](image)

**Figure 2** Doubling a clock output prevents a backward phase jump.

When bit 1 is output, with an edge_select of 0000 0001, and the multiplexer jumps to bit 7, with an edge select of 0100 0000, a sudden phase jump in output data of 360°×7/8, or 315°, occurs. Because the shift-register data bit 7 is a delayed version of the last clocked bit, b1, the output clock must be stalled for one cycle. Otherwise, one bit too many will clock at the output (circled area in Figure 3). You can solve the overflow-phase jumps by gating the output clock using combinatorial logic.

![Figure 3](image)

**Figure 3** Stalling the output clock prevents a forward phase jump.

**Reference**