techXclusives

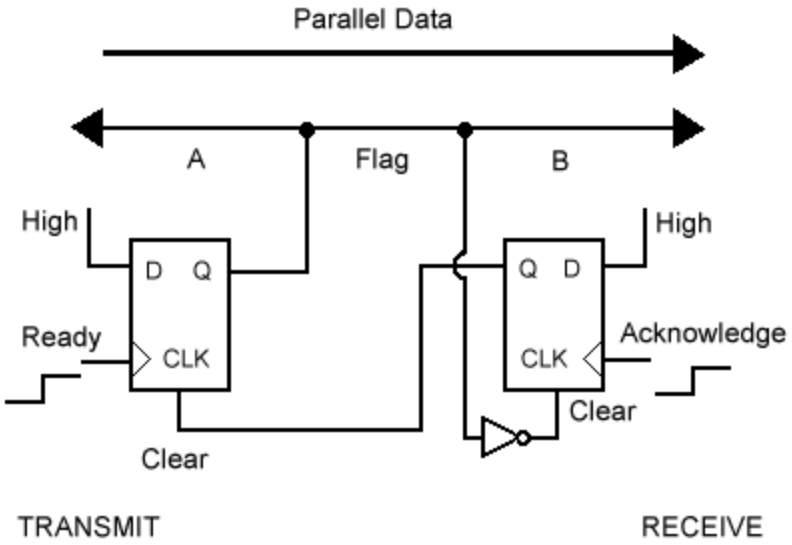
Moving Data Across Asynchronous Clock Boundaries

By Peter Alfke
Director, Applications Engineering

Synchronous single-clock systems are robust and easy to design, simulate, and debug. But in some situations, multiple unrelated clocks must access common data. Such asynchronous interfaces will operate reliably and predictably if the designer invests the necessary care.

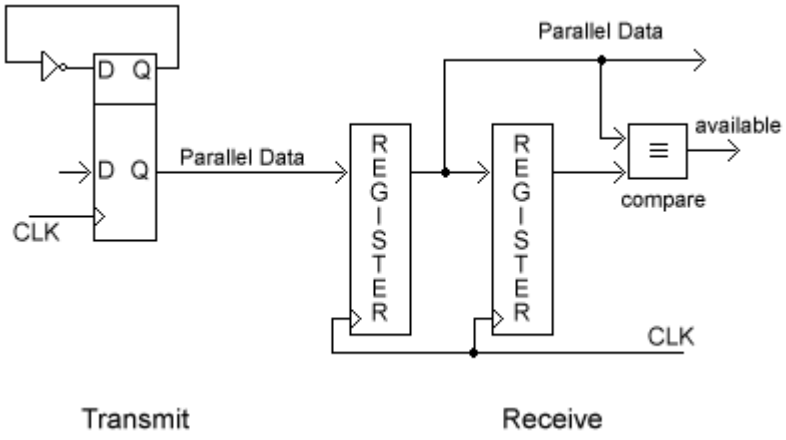
Two or more unrelated (i.e, asynchronous) clocks have a constantly changing phase relationship, and the designer must anticipate the worst possible condition, because it will inevitably occur sooner or later.

WARNING: Never cross a clock-domain boundary with more than one control interface. If you do, there will inevitably come the moment where the two controllers disagree; most systems cannot cope with that condition.



Here is a safe circuit that controls parallel date transfer from a transmitter on the left to the receiver on the right. The transmitter indicates available data by clocking (and thus setting) flip-flop A. This raises the Flag line that is monitored by both sides. As long as the Flag is High, the transmitter must maintain the data on the bus for the receiver to read. Having read the data, the receiver acknowledges this by clocking (and thus setting) flip-flop B. This, in turn, clears flip-flop A, pulls the Flag Low, and also clears flip-flop B. This inherently benign and safe race condition can (redundantly) be made even safer by adding delay to the inverter that resets flip-flop B.

This design works reliably with any arbitrary relationship or non-relationship between the transmit and receive clocks and their timing. This design is slowed down by the need to monitor and manipulate the Flag signal. The transmitter may present new data only when the Flag is High, and it must acknowledge this by setting flip-flop B.



If the receiver clock is always faster than the transmitter clock, the interface can be simplified and run open-loop, without any handshake. In the most general case, the transmitter puts data on the bus and adds one more bit that toggles for every new data word. This extra bit allows the receiver to separate a series of identical words. Depending on the system requirements, this extra bit may not be necessary.

The receiver constantly clocks the data into a dual-rank register and monitors the output of an identity comparator. Whenever the two registers are identical, both contain proper data. And the XOR of the control bit indicates that the data has changed. There is no need for a handshake, as long as the receive clock period is always shorter than the transmit clock period.

These circuits allow the transfer of parallel data between asynchronous clocks. In the most general cases of high-speed or variable-speed read and write clocks, an asynchronous FIFO is the best solution. (This will be described in my next techXclusive.)