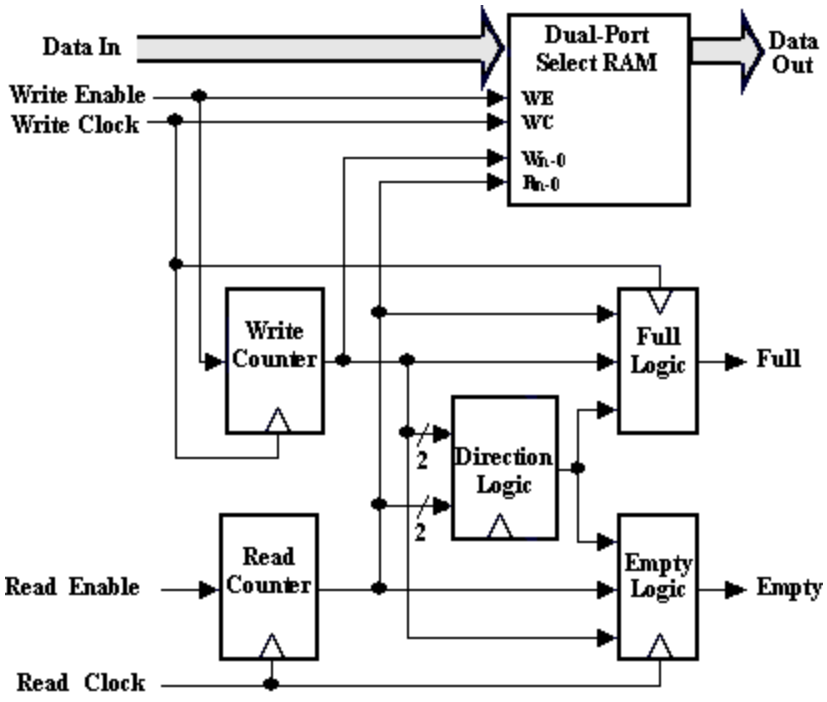
techXclusives

Asynchronous FIFO in Virtex-II™ FPGAs

By Peter Alfke
Director, Applications Engineering - San Jose

A FIFO is a popular memory structure that solves data-rate differences in many systems. Most FIFOs have independent write and read clocks to move data across clock boundaries. This requires some attention to asynchronous design details, as described below.

Given a dual-ported RAM, like the BlockRAMs available in FPGA devices, the core FIFO design is trivial. One port is configured as write port, addressed by a write-address counter, the other port is the read port, addressed by the read-address counter. Both counters must count by the same algorithm, but they need not be binary counters. Linear-Feedback-Shift-register (LFSR) counters used to be popular, but Gray counters are best for the general case of asynchronous FIFOs, as described below.



FIFO Block Diagram

Figure 1

The only difficult design detail is the proper and reliable handling of the two exceptional cases, FIFO FULL and FIFO EMPTY. FULL means that the next write, if allowed, would overwrite the oldest not-yet-read entry; EMPTY means that the next read, if allowed, would read stale data a second time.

Either of these conditions must raise its own flag, telling the system to cope with the potential overflow or underflow. Designers can cheat on the Full flag and raise it early, thus sacrificing some FIFO depth, but there is no alternative to the proper timing of the EMPTY flag, since the system needs to read even the very last word stored in the FIFO.

There are two problems with FULL and EMPTY:

- First, both conditions are indicated by the identity of read and write addresses. Therefore, something else has to distinguish between FULL and EMPTY. A simple solution divides the address space into four quadrants and decodes the two MSBs of the two counters (works in binary or Gray, but the equations are different) together in two 4-input look-up tables. If the write counter is one quadrant behind the read counter, this indicates a "possibly going full" situation, and sets a direction latch. If the write counter is one quadrant ahead of the read counter, this indicates a "possibly going empty" situation, and resets the direction latch. The direction latch eliminates the ambiguity of the address identity decoder.

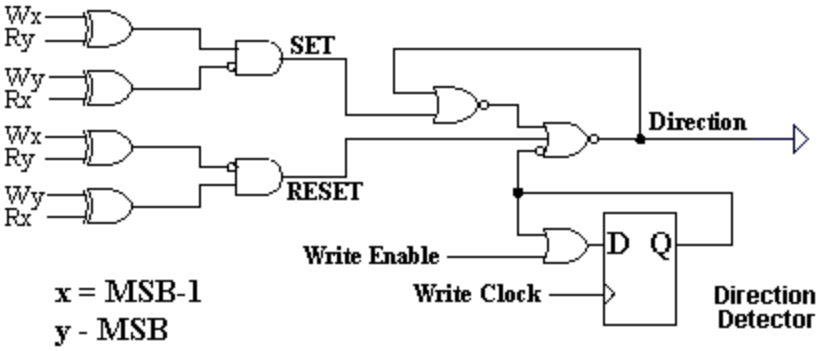


Figure 2

- The second, and more difficult, problem stems from the asynchronous nature of the write and read clocks. Comparing two counters that are clocked asynchronously can lead to unreliable decoding spikes when either or both counters change multiple bits "simultaneously". The preferred solution is to make both counters count in a Gray sequence, where only one bit changes from one count to the next. Any decoder or comparator will then switch only from one valid output to the next one, with no danger of spurious decoding glitches.

Virtex™ logic makes it very easy to design high-speed synchronous Gray counters, taking advantage of the built-in binary carry structure. A binary counter is implemented by cascading a string of vertical slices, using the carry chain. The flip-flops in the adjacent slice form the Gray counter, clocked by the same clock (and clock enable) as the binary counter, but its D-inputs driven by the XOR of the two binary counter D-signals, one of the same rank, and one of the next higher significance:

Dgi = Dbi XOR Db(i+1), Dgmax = Dbmax

where "i" designates the binary significance, and "Dg" and "Db" denote the Gray and binary counter's D-input, respectively. The Gray counters thus operate in parallel with the binary counters, are not pipelined, and do not suffer from any decoding glitches. There is never a false "equal" output, but there can be an arbitrarily short output glitch when counter A matches B, but B increments very soon after. The resulting spike either creates a Full or Empty signal, or it does not. Either case is acceptable.