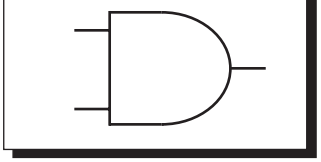

KISS those asynchronous-logic problems good-bye

Steve Knapp

Many organizations promote an effective philosophy dubbed KISS (for Keep It Simple, Stupid) to improve the success rate of complex operations. A variation of this theme promotes success in digital design: Keep It Strictly Synchronous.

The popularity and flexibility of array-based logic—whether a field-programmable gate array (FPGA), a complex programmable-logic device (CPLD) or a gate array—might tempt unwary designers into developing bad asynchronous habits. Because these devices all use programmable interconnect, different signal arrival times coupled with asynchronous logic invite a digital disaster.

Synchronous designs are inherently safer and easier to debug than asynchronous designs. Engineers can simply predict the behavior of synchronous systems and model them in simulation. The analysis comes down to the worst-case path between clock edges—a simple process, especially using a static timing analyzer. Asynchronous design involves analyzing all combinations of best- and worst-case signal paths over temperature, voltage and process—a far more onerous chore.

Consequently, synchronous designs work with a much wider variation in device timing parameters and over a broader temperature range than do most asynchronous designs. As process technology improves, circuit delays decrease. A vendor might ship faster devices that meet all datasheet specifications but behave differently than the old part. Problems with asynchronous logic usually don't appear until you've built a board. One batch of parts works fine, another batch fails the system test, while another batch seemingly fails intermittently in the field depending on operating conditions.

The scourge of gated clocks

One problem most vulnerable to process changes and common in programmable-logic and gate-array designs is the gated clock. The Japanese phrase *zen zen dame*, which loosely translated means "never, never do" applies in this situation. Glitches on a

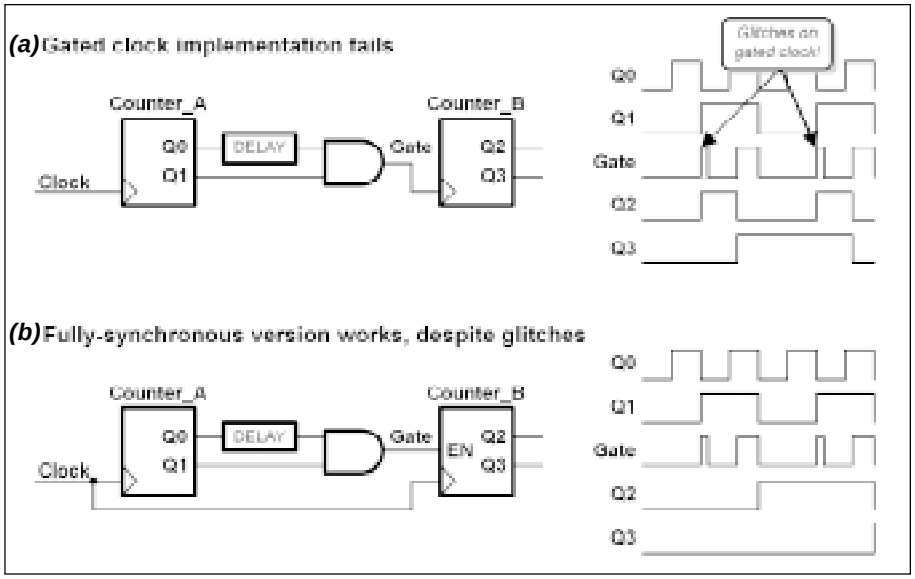

Fig 1—Gated clocks often have glitches due to differences in signal arrival times (a). Changing to a synchronous solution guarantees success(b).

Steven K Knapp is the founder and president of OptiMagic Inc (Aptos, CA, www.optimagic.com), a firm that develops intellectual property and design software for programmable logic. Prior to founding this firm he held various applications, engineering and management positions at Xilinx and Intel's former programmable-logic division.

EXPERT COLUMN
EDA

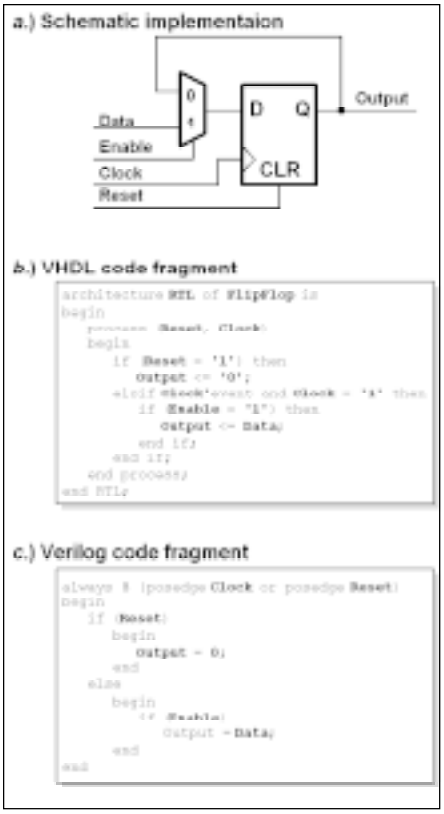

Fig 2—Some FPGAs have flip-flops with built-in clock enables, some don't. Building an equivalent solution is simple as implemented in schematic (a), VHDL (b) and Verilog (c).

gated clock often result from differences in arrival times for its input signals caused by routing delays inside a device (Fig 1). Further, gated clocks introduce additional delay in the clock path, which pushes out clock-to-output times and might introduce hold-time problems.

At first blush, you might think that you never use gated clocks. But all of the following situations involve a glitch-prone path through some combinatorial logic. For each circuit, there is an alternative, fully-synchronous implementation, usually involving a flip-flop with a Clock Enable input. In general, the clock for an internal flip-flop should only originate from either a device input or from another flip-flop. Consider these common situations:

• **Clocks derived from the terminal count (TC) of a counter.** Most circuits generate TC signals with an AND gate. This situation is common in applications where a high-speed clock is divided down and redistributed within a device.

• **Clocks derived from a decoder.** Many designs commonly use this logic to load various banks of flip-flops.

• **Clocks derived from a multiplexer output.** This situation is common in applications that select various clock frequencies.

The example in Fig 1a, presents a case where a gated input drives the Clock input of the second counter—a bad situation. The difference in routing delays causes glitches on the AND gate during specific states on Counter_A. These glitches cause incorrect clocking for Counter_B. A better implementation appears in Fig 1b, where both counters operate from the same clock source. The terminal count from Counter_A drives a Clock Enable signal on Counter_B. The clock path is clean and unfettered with asynchronous gates.

The solution in this example, and many others, includes a flip-flop with a Clock Enable input. In some FPGA devices, such as those from Xilinx and Lucent, the internal flip-flops have a built-in Clock Enable input. In other devices, you can easily build a clock input by including a 2:1 multiplexer in front of the data input (Fig 2). If designing circuits with VHDL or Verilog, note that not all synthesis packages automatically create a flip-flop using the built-in enable, even if one is available in the target technology.

A designer using CPLDs might wonder about the wisdom in using a product-term clock, which essentially is a gated clock. However, internal delays on a CPLD product term are more closely matched than routing delays inside an FPGA or gate array. However, you can still run into gated-clock problems on a product-term clock. Watch the inputs and their arrival times and be wary of potential glitches.

Global buffers aid clocking

Another clocking aid is a global clock buffer, a high-speed, low-skew, high-fanout clock-distribution network built into most CPLDs and FPGAs. These devices come with two or more global clock buffers, and some have as many as eight. If a design doesn't use them, these valuable resources are wasted. Ideally, a design should have one or two clock inputs. These clocks, typically with high fanout, should use the global clock buffers to simplify overall device design.

With most vendors' tools, you must specifically request a global buffer either using a special symbol in the schematic or by instantiating the buffer through VHDL or Verilog, but some tools automatically infer the use of a global buffer.

A downside of global clock buffers is that they're among the largest power consumers in an FPGA or CPLD at somewhere between 2-10 mW/MHz. For most designs, this level is second only to I/O switching in overall power consumption. However, the benefit of a clock buffer typically outweighs the extra power consumption and engineering

required to design without them. Luckily, in most devices you can connect as many flip-flops as desired to a buffer without consuming additional power. If a design requires the absolute minimum power and you decide not to use any global clock buffers, always be wary of clock-skew problems where the clock signal might arrive at one flip-flop before the others.

Many synchronous flip-flops, even with global clocks, use asynchronous Set and Reset inputs. Keep a watchful eye on these inputs. With modern fast devices, even a momentary glitch is enough to inadvertently change a flop's state. Fig 3 shows a common circuit that decodes the output of a counter, and the decoded output asynchronously resets the counter. The problem is that a decoded output might be too short to reliably reset all the flip-flops in the counter. A better approach is to use a counter with a synchronous reset.

As mentioned for clock inputs, asynchronous flip-flop inputs should originate from either device inputs or from other flip-flop outputs.

A few guidelines

A few simple guidelines define the Keep It Strictly Synchronous approach:

• Use as few clocks as possible. The ideal synchronous system has a single clock input.

• Avoid gated signals on any asynchronous flip-flop input including Clock inputs as well as asynchronous Set or Reset inputs.

• If available, use global clock buffers to distribute any high-fanout or skew-critical clock signals. If you're not using global buffers, always evaluate each separate clock path for clock-skew problems.

• If you find an asynchronous circuit, ask if it's possible to redesign the logic using a synchronous alternative.

• If you absolutely require an asynchronous circuit, always perform a thorough worst-case analysis. What happens if one of the signals arrives early and the other late? How about vice versa? Watch for glitches. ***PE&IN***

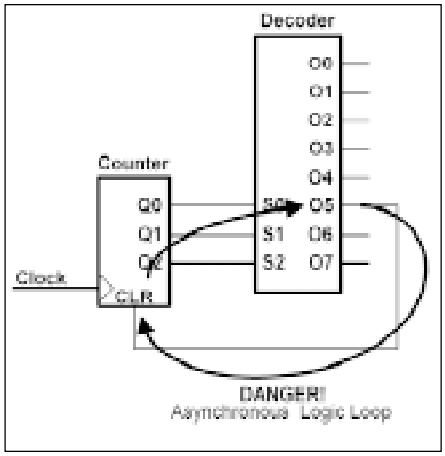

Fig 3—A common circuit demonstrates unreliable operation. The output from the decoder depends upon and also resets the counter flip-flops. The asynchronous reset signal might be too fast to reliably reset all of the counter flip-flops.

This space left intentionally blank.

Editorial Feedback
This article's value to me was:
High—266 Average—267 Low—268