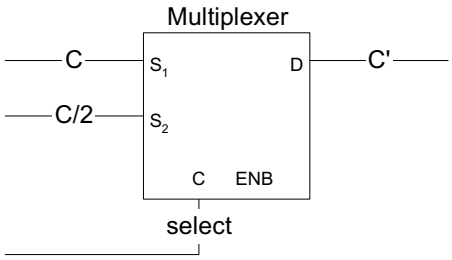
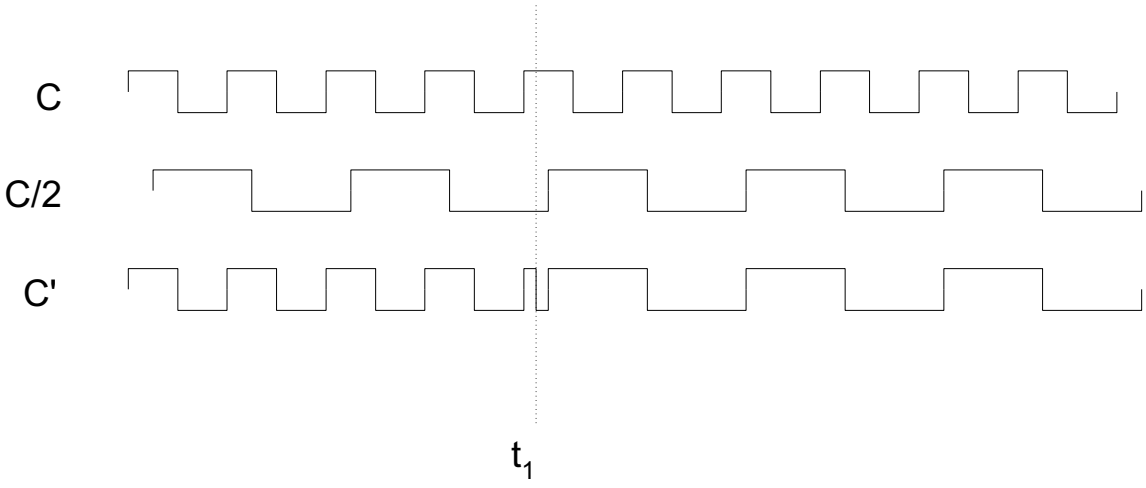
SWITCHING ASYNCHRONOUS CLOCKS

Several months ago, while designing a UART, I was faced with the following problem. You have a clock C, and its derivatives C/2, C/3, C/4 ... C/n. Some logic works off a clock C', where C' is one of C or its derivatives. The selection of the clock is done based on a register R, and the selection can be made at any time. In other words, R is not updated synchronously with C'. The problem is this: How do you switch the clock in such a way that the logic that works off C' does not have a pulse width violation?

This problem is shown in Figure 1.



A: Simple multiplexer based clock switching scheme



B: Waveform for A shows glitches.

Figure 1 An unsuccessful clock switching scheme.

This is the general problem with switching clocks – if the switching instant is not "chosen" carefully, the resultant clock will have narrow pulses (high or low or both, see time t_1 in Figure 1 above) that could throw succeeding logic into metastability. This article demonstrates a simple solution that can handle any two asynchronous clocks, switching them in such a way that a pulse width violation can be guaranteed not to occur.

Back to our problem. We have a clock C and *several* derivatives C/2, C/3,… C/N that need to be switched. This problem is solved by breaking it down into two steps. We begin by partitioning the set of clocks into two classes: the primary clock C, and the derived clocks C_d = {C/2, C/3,…C/N}. Switching between the derived clocks is easy because we can devise a mechanism that works off C. In fact, this is accomplished by having a loadable counter that is clocked by C. This loadable counter loads a value, counts down to 0, and toggles a synchronous output. Changing a clock would simply mean changing the load value. Synchronizing the load value will mean that for a few clocks, the counter may not see the correct load value – but it eventually will.

Switching between C and any member of the class C_d is a different problem and is best tackled by solving for the general case of two asynchronous clocks C1 and C2, which have no frequency or phase relationship. Given that glitches or spikes are not acceptable, we propose to shut off one clock before starting the other. Further, this shutting and starting must occur during a "dead" period of the clocks. For instance, C1 is turned off during a low period, and C2 is turned on during its low period. A scheme for this is shown in Figure 2.

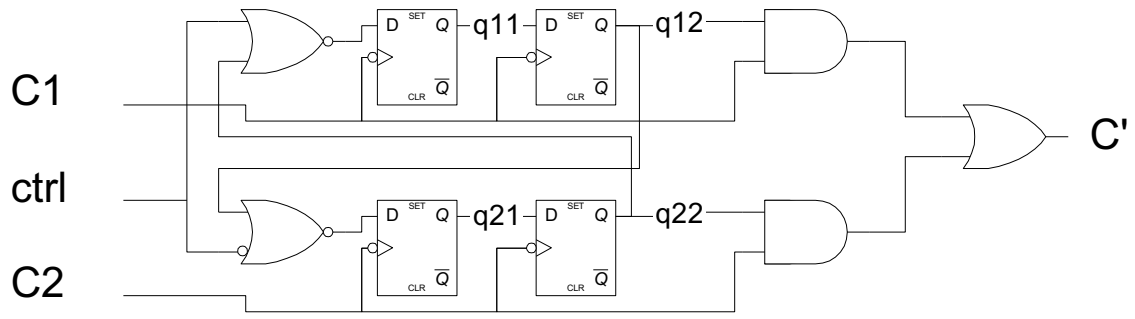


Figure 2 Clean switching of two clocks.

In this circuit, assume that initially all flip-flops are clear. It is easy to set this up with flip-flops that clear asynchronously on power on. Also assume that ctrl = 0. At power ON, and as long as the asynchronous clear is asserted, C′ = 0. This is a subtle advantage of this circuit – namely that there can never be a removal time violation (reset removal to clock active edge) on any logic that works off C′. Once the asynchronous reset is removed, q11 becomes 1 at the first falling edge of C1, and at the next edge, q12 is set, making C′ = C1, and this state persists indefinitely until the value of ctrl is changed. When ctrl is pulled high, presumably asynchronously with respect to C1, q11 goes low if there is no setup

or hold violation. After one more falling edge, q12 goes low, thereby shutting off C′ and holding it low. If q11 had a timing violation that caused it to go metastable, the worst that could happen is that C′ takes additional time before it shuts off. Only after q12 = 0 (or that C′ shuts down) does the C2 chain start up. There could be a further violation on q21, but the worst effect that this could have is that C′ takes longer to start up with C2. In other words, metastable failures could extend the dead period of the clocks, but never cause glitches or spikes.

This circuit has a subtle timing problem that could occur – even though the chances of this happening seem rather remote. To understand how this circuit can be made to fail, consider the case when the delay of C2 through the AND gate to the OR gate to C′ is greater than the delay from the negative edge of C2 through q22 through the AND gate through the OR gate to C′. If this is the case, while turning C2 on, there could be a spike on C2. Of course, since the circuit is symmetric, the same problem exists on C1. To make sure this does not happen, the clock skew between the clocks at q12(q22) and its corresponding AND gate must be less than the clk-to-Q time for q12(q22). Normally ASIC clock skews are lesser than t_cq, but the prudent designer will make certain that this is indeed so, especially for FPGAs. The same problem would occur if the AND gate is not delay-symmetric; that is, the delay from one input to the output is not equal to the delay from the other.

Extending the concept to switching three or more asynchronous clocks is trivial and shown in

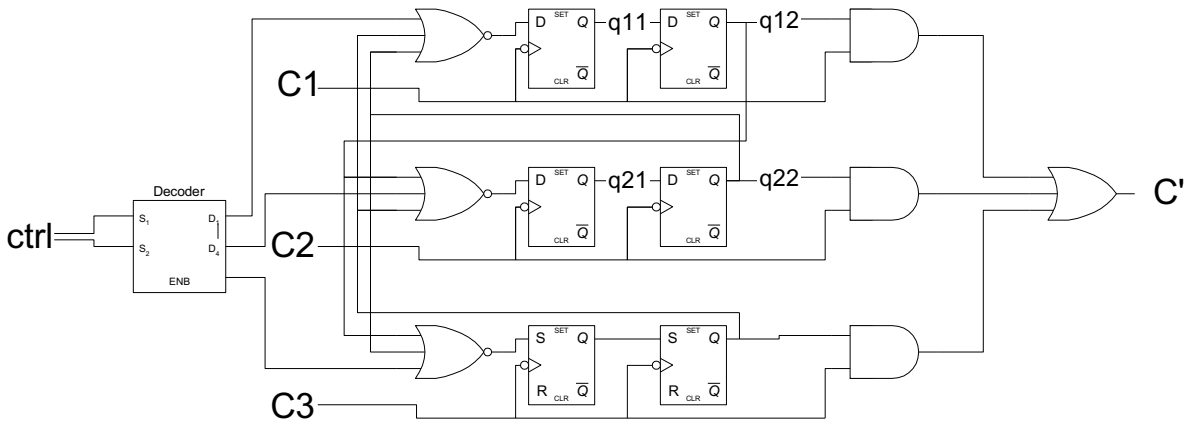


Figure 3 Extension of the concept to three clocks.