The "Flancter"

How to set a status flag in one clock domain, clear it in another, and never have to use an asynchronous clear for anything but reset.

by Rob Weinstein
Senior Member of Technical Staff,
Memec Design Services

There are times when it's important to generate a status flag that is set by an event in one clock domain and reset by an event in a different clock domain. Using a D-type flip-flop, where a "1" is clocked in from one clock domain and its asynchronous reset is pulsed by logic in the second clock domain, is the time-honored method to achieve this function. While there is nothing logically wrong with this, it introduces other problems such as combinational logic driving an asynchronous reset pin, uncertainty in timing constraint boundaries, and muddying the global reset function.

Here, I present an alternative method for generating a multiple clock domain flag register that mitigates these problems. It's called the "Flancter" (named by my colleague, Mark Long), and is shown in Figure 1.

As you can see, it's made up of two D-type flip-flops, an inverter, and an exclusive OR (XOR) gate. Notice that the asynchronous reset inputs to the flip-flops are shown unconnected for clarity only. Normally, these would be tied to the global set/reset net in the system.

Operation of the Flancter is simple; when FF1 is clocked (rising edge of SET_CLK while SET_CE is asserted), OUT goes high. When FF2 is clocked (rising edge of RESET_CLK while RESET_CE is asserted), OUT goes low. Note that this circuit must be used in an interlocked system where the flip-flops won't be continuously clocked by the two clock domains. Also, the output must be synchronized with additional flip-flops to mitigate metastability when crossing clock domains (more about this later).

How It Works

To explain its operation, I like to rearrange the circuit as shown in Figure 2.

This is the same circuit as shown in Figure 1, but untwisted so that the two inputs to the XOR gate are clearly visible. You can see that the XOR gate's upper input is labeled Q1, while its lower input is labeled Q2. Also, Q1 and Q2 are the Q outputs of FF1 and FF2, respectively. Now for the trick part of this magic trick: the D input to FF1 comes from an inverter, so whenever FF1 is clocked, Q1 assumes the opposite state of Q2 and the output of the XOR

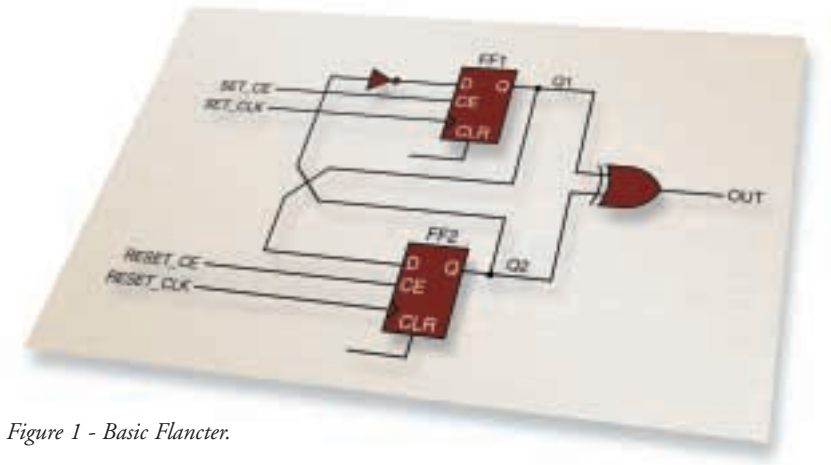

Figure 1 - Basic Flancter.

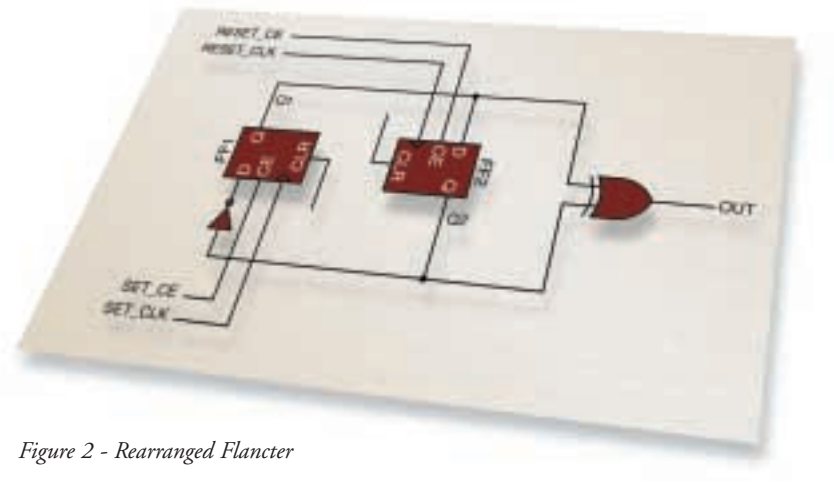

Figure 2 - Rearranged Flancter

gate will go high. When FF2 is clocked, Q2 becomes the same as Q1, and the output will go low. In summary, clocking FF1 causes OUT to go high and clocking FF2 causes OUT to go low.

A timing diagram will help describe the Flancter's operation. Figure 3 shows the basic timing diagram.

The basic points of interest in the timing diagram are:

- SET_CLK and RESET_CLK are asynchronous to each other.
- At point A, the rising edge of SET_CLK while SET_CE is high causes Q1 to go high because it gets the inverted value of Q2. Also, OUT goes high because it is the XOR of Q1 and Q2.
- At point B, the rising edge of RESET_CLK while RESET_CE is high causes Q2 to go high because it gets the value of Q1. Also, OUT goes low because it is the XOR of Q1 and Q2.
- At point C, Q1 again gets the inverted value of Q2, causing OUT to go high.
- At point D, Q2 goes low because it gets the value of Q1, causing OUT to go low.

So What's Wrong With It?

There are a few things wrong with the Flancter from the start. One problem is that it uses two flip-flops to create a single flag bit. This is a minor fault when you consider that most FPGAs have an abundant supply of flip-flops. A more serious issue is how to use the output. Remember that the output can change synchronously to either clock domain. You need to resynchronize the output to whichever clock domain needs to see it; often both clock domains. It is common to use two flip-flops in series as a metastability-resistant synchronizer.

However, the most serious drawback to the Flancter is that operating the set and reset flip-flops must be mutually exclusive in time. This means that when logic in clock domain 1 sets the Flancter, it doesn't attempt to set the Flancter again until it sees that it has been reset. Likewise, the logic in clock domain 2 never attempts to reset the Flancter unless it sees that it has been set. Establishing this kind of interlocked protocol guarantees that both of the Flancter's flip-flops won't be clocked simultaneously (or within each other's setup and hold time windows).

Applications of the Flancter

There are many applications of the Flancter, but a very common application is interfacing a microprocessor to an FPGA. Typically, the microprocessor and FPGA logic run on separate clocks. When the microprocessor writes a control register within the FPGA, the Flancter can be used as a status flag to tell an internal state machine that new data is available.

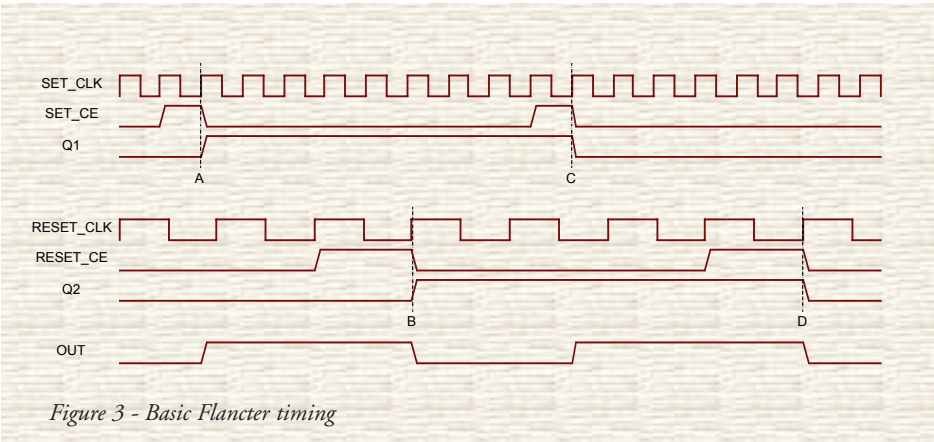

Figure 3 - Basic Flancter timing

Likewise, a state machine within the FPGA can use the Flancter to generate an interrupt to the microprocessor that is subsequently cleared by a read or interrupt-acknowledge cycle from the microprocessor, as shown in Figure 4.

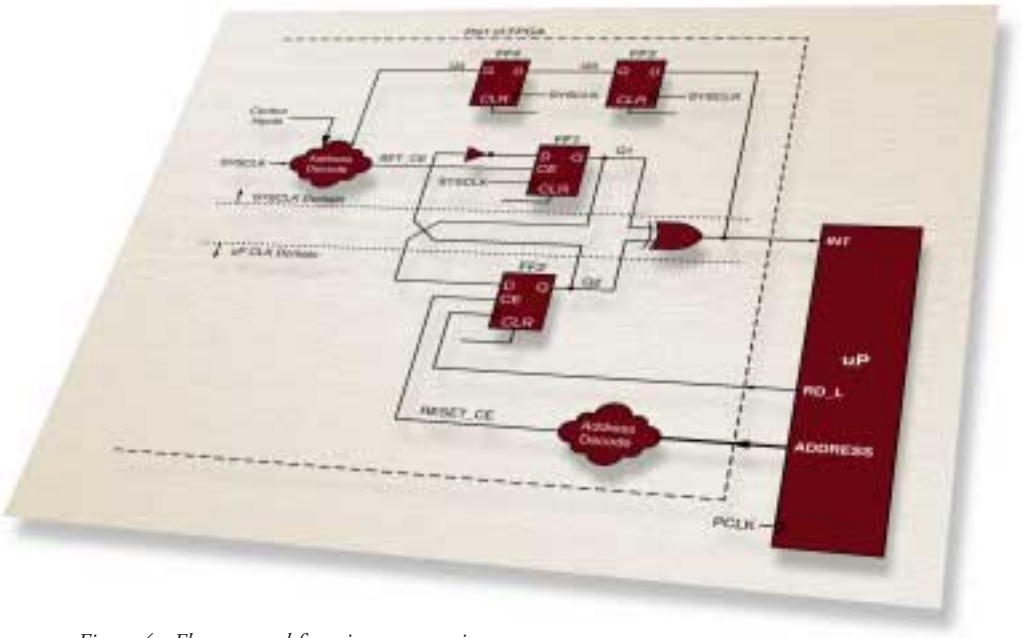

Figure 4 - Flancter used for microprocessor interrupt

Things to notice in Figure 4:

- The Flancter is made up of FF1, FF2, the inverter, and the XOR gate.
- The state machine, FF1, FF3, and FF4 are all synchronous to SYSCLK.
- The microprocessor (µP) runs off its own clock, PCLK.
- The state machine pulses SET_CE for one SYSCLK cycle when it needs to request an interrupt.
- The microprocessor performs a read cycle from a predefined address to reset the interrupt. Although not shown, reading from this address may also cause a status register to be driven onto the microprocessor's data bus allowing simultaneous reading of status and resetting of interrupt.
- FF3 and FF4 are resynchronizing flip-flops used to filter any metastable logic conditions from propagating into the state machine.
- The particular microprocessor used in this example employs metastable resistant techniques on its INT input.
- The interrupt sequence is defined such that setting and resetting the interrupt flag cannot occur simultaneously.

The following timing diagram helps illustrate the operation:

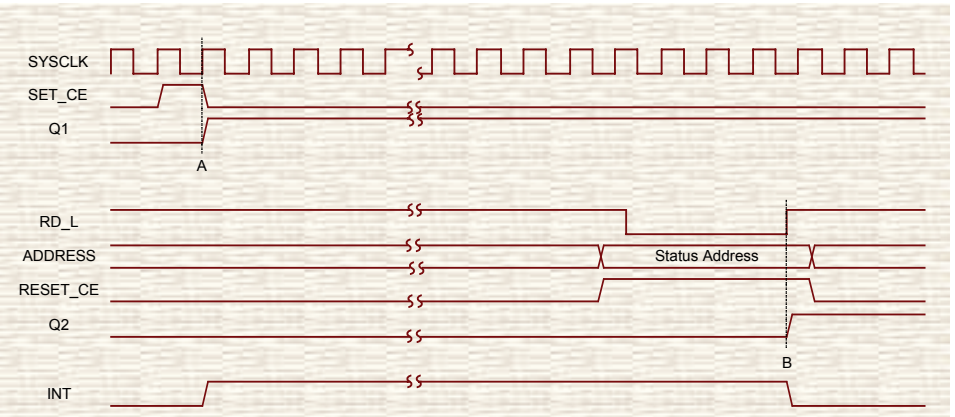

Figure 5 - Flancter interrupt timing diagram

Things to notice in Figure 5:

- The FPGA's state machine sets the interrupt request (INT) at point A.
- Sometime later, the microprocessor responds to the interrupt by reading a status register, thus resetting the interrupt request at point B.

Variations on a Flancter

While the basic Flancter is very simple, many useful variations are possible. I describe some of the more interesting variations in the Flancter app-note available on our website, http://www.memecdesign.com/resources/guides/. Some of the variations you'll find there include the 3-way Flancter, the n-way Flancter, the async-reset emulating Flancter, and the default-to-active-high Flancter. The app-note also has VHDL and Verilog listings for the basic Flancter.

Conclusion

I turned 37 this year and I got to thinking that all the "greats" did their best work long before they reached my age. As I look back on my design career, I realize that I haven't designed any circuits or developed any theorems that will bear my name like the Pierce Oscillator or Shannon's Sampling Theorem. The best I can do is to offer the Flancter as my legacy. Perhaps someday, the Weinstein-Flancter will be found in the indexes of engineering tomes right between Watt-Hour and Wien-Bridge.