

How To Set a Status Flag in One Clock Domain, Clear It in Another, and Never, Ever Have to Use an Asynchronous Clear for Anything but Reset

Overview

There are times when it is 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 is made up of two D-type flip-flops, an inverter, and an

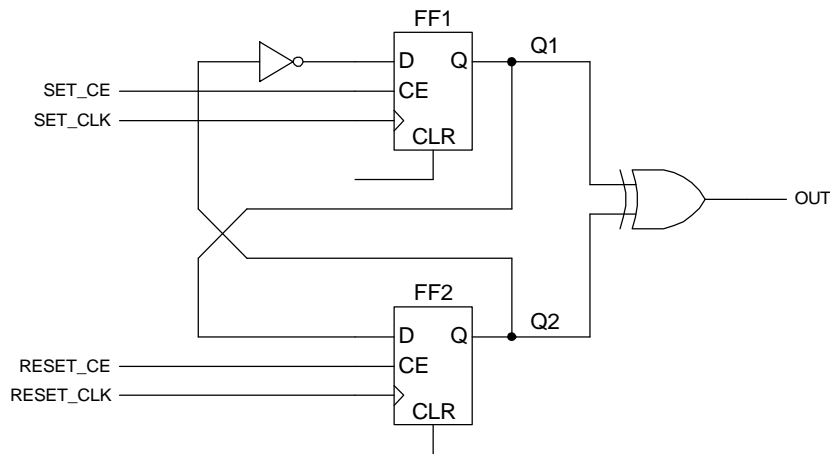


Figure 1 - Basic Flancter.

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. Contrariwise, 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.

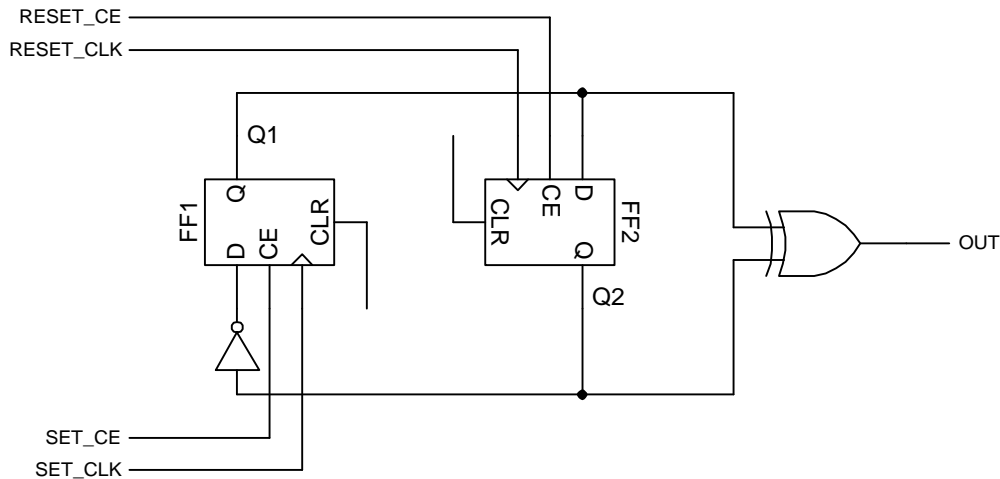


Figure 2 - Rearranged Flancter.

This is exactly 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 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:

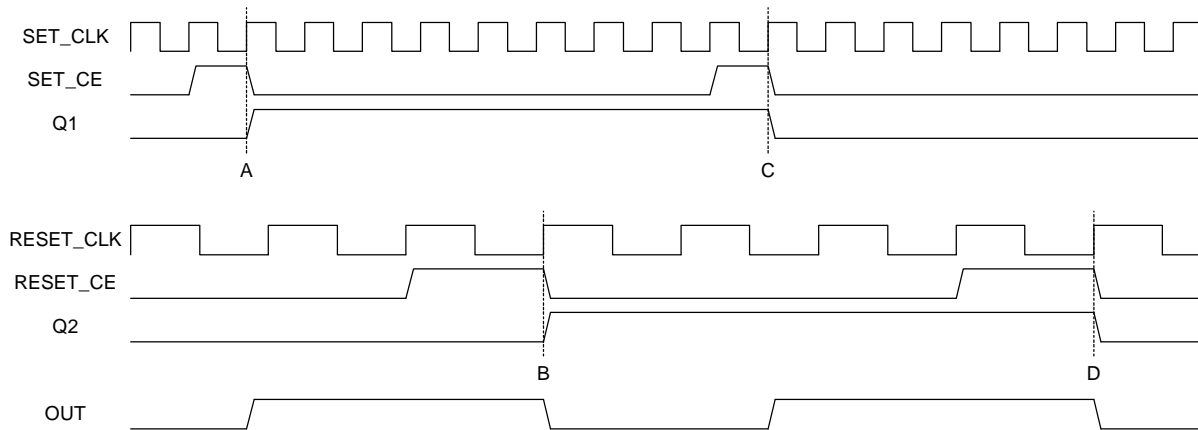


Figure 3 - Basic Flancter Timing.

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. 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.

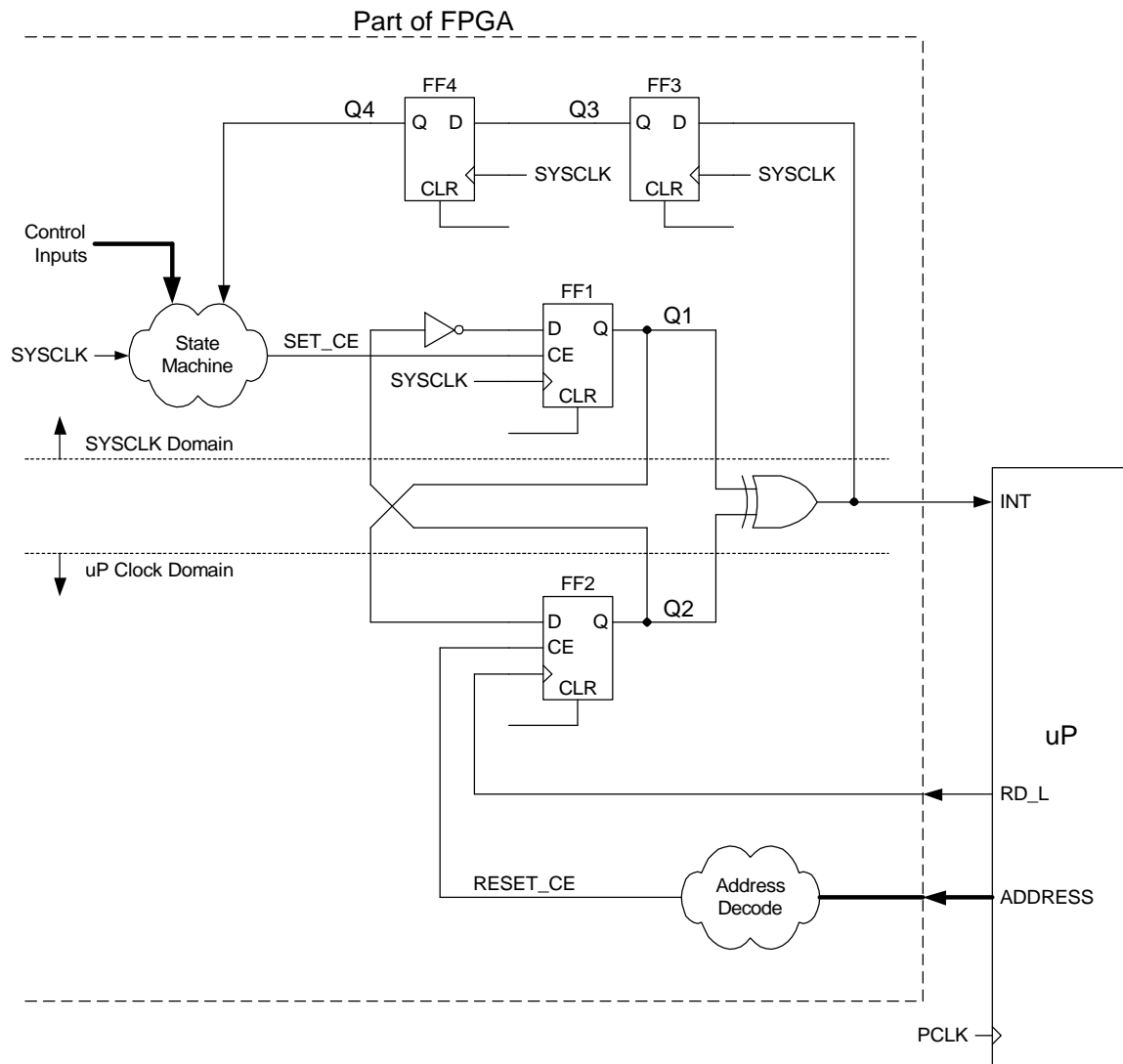


Figure 4 - Flancter Used for Microprocessor Interrupt.

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 (uP) 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:

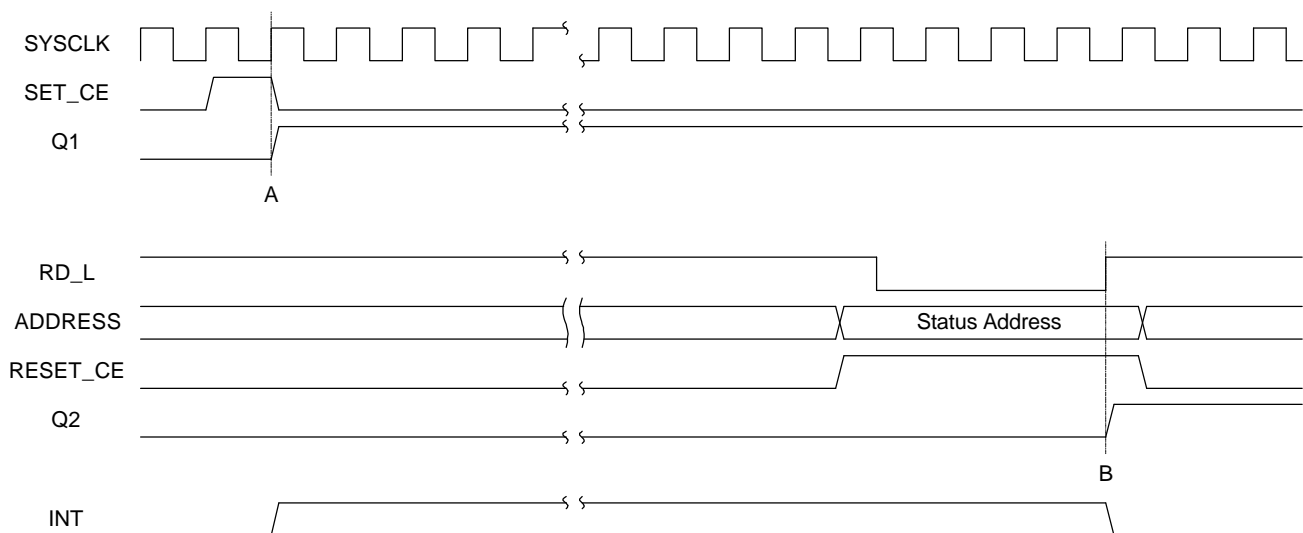


Figure 5 - Flancter Interrupt Timing Diagram.

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.

HDL Examples

Verilog Example

```
//-----  
// FileName      : flancter.v  
// Author       : Rob Weinstein  
// -----  
  
module flancter (  
    // Inputs  
    ASYNC_RESET,  
    SET_CLK,  
    SET_CE,  
    RESET_CLK,  
    RESET_CE,  
    // Outputs  
    FLAG_OUT  
);  
  
input  ASYNC_RESET;  
input  SET_CLK;  
input  SET_CE;  
input  RESET_CLK;  
input  RESET_CE;  
output FLAG_OUT;  
  
// Internal Registers  
reg    SetFlop;  
reg    RstFlop;  
  
// Output assignments  
assign FLAG_OUT = SetFlop ^ RstFlop;  
  
// The Set flip-flop  
always @(posedge SET_CLK or posedge ASYNC_RESET)  
begin : set_proc  
    if (ASYNC_RESET)  
        SetFlop <= 0;  
    else if (SET_CE)  
        SetFlop <= ~RstFlop;    // Flops get opposite logic levels.  
end  
  
// The Reset flip-flop  
always @(posedge RESET_CLK or posedge ASYNC_RESET)  
begin : reset_proc  
    if (ASYNC_RESET)  
        RstFlop <= 0;  
    else if (RESET_CE)  
        RstFlop <= SetFlop;    // Flops get the same logic levels.  
end  
  
endmodule
```

VHDL Example

```
-----  
-- FileName       : flancter.vhd  
-- Author        : Rob Weinstein  
-----  
  
library IEEE;  
use IEEE.std_logic_1164.all;  
  
entity flancter is  
    port (  
        ASYNC_RESET   : in  std_logic;  
        SET_CLK       : in  std_logic;  
        SET_CE        : in  std_logic;  
        RESET_CLK     : in  std_logic;  
        RESET_CE      : in  std_logic;  
        FLAG_OUT      : out std_logic  
    );  
end flancter;  
  
architecture flancter of flancter is  
    signal SetFlop      : std_logic;  
    signal RstFlop      : std_logic;  
  
begin  
  
    --The Set flip-flop  
    set_proc:process(ASYNC_RESET, SET_CLK)  
    begin  
        if ASYNC_RESET = '1' then  
            SetFlop <= '0';  
        elsif rising_edge(SET_CLK) then  
            if SET_CE = '1' then  
                -- Flops get opposite logic levels.  
                SetFlop <= not RstFlop;  
            end if;  
        end if;  
    end process;  
  
    --The Reset flip-flop  
    reset_proc:process(ASYNC_RESET, RESET_CLK)  
    begin  
        if ASYNC_RESET = '1' then  
            RstFlop <= '0';  
        elsif rising_edge(RESET_CLK) then  
            if RESET_CE = '1' then  
                -- Flops get the same logic levels.  
                RstFlop <= SetFlop;  
            end if;  
        end if;  
    end process;  
  
    FLAG_OUT <= SetFlop xor RstFlop;  
  
end flancter;
```

Synthesis Result

The Verilog example was run through Synplify and the resulting HDL Analyst RTL view is shown in Figure below:

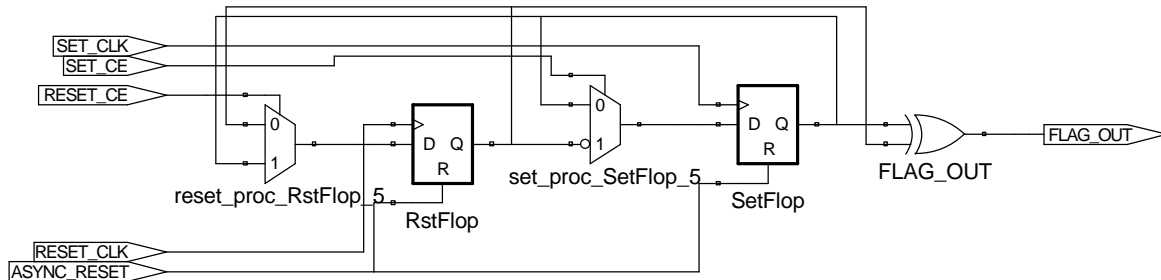


Figure 6 - Synthesized Flancter.

Note that in the RTL view, clock-enabled flip-flops are shown as multiplexers in front of D-type flip-flops.

Variations on a Flancter

While the basic Flancter is very simple, many useful variations are possible. I describe some of the more interesting variations below.

Flancter Variation #1

One interesting variation lets you use a Flancter in lieu of a D-type flip-flop whose async clear input is used to save a clock cycle. In some single-clock-domain systems, the original designer relied on an async clear to immediately reset a flip-flop so that it would be reset by the next rising clock edge. A variation of a Flancter can give you the same functionality without relying on an async clear input. Figure 7 shows this variation:

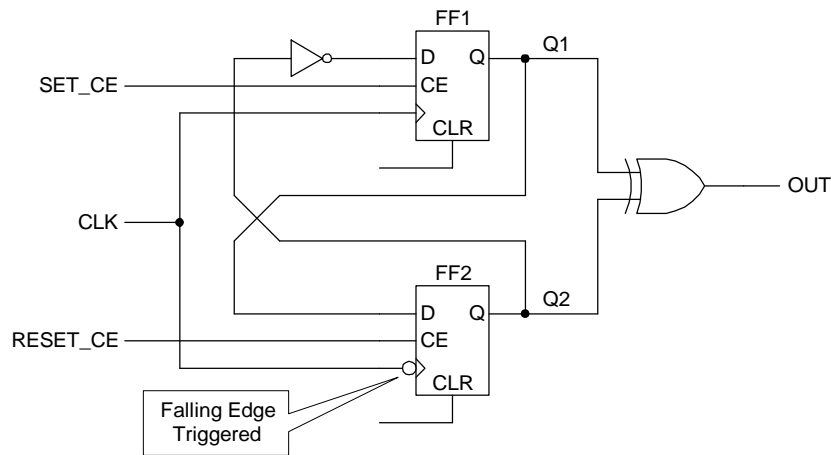


Figure 7 - Flancter Variation #1.

Note that this is the same as the Basic Flancter, but it uses a single clock input, CLK, and the reset flip-flop is clocked on the falling edge of that clock.

Its operation is depicted in the timing diagram in Figure 8:

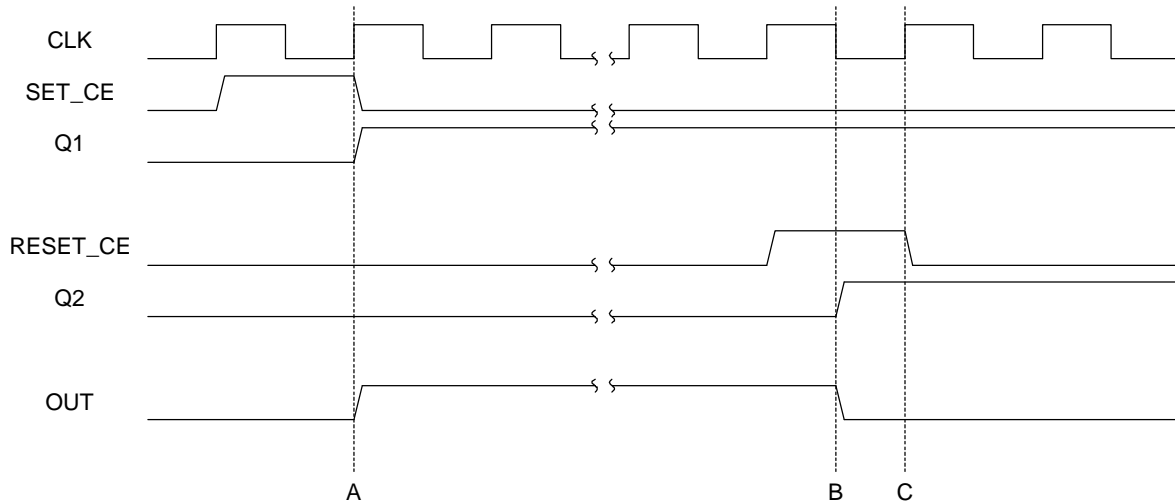


Figure 8 - Flancter Timing for Variation #1.

Things to notice in Figure 8:

- OUT goes high on the rising edge of CLK at point A in response to SET_CE.
- Sometime later, RESET_CE is pulsed for one CLKcycle, synchronous to the rising edge of CLK, but...
- OUT responds to RESET_CE by going low on the *falling* edge of CLK at point B.
- OUT is already low by the next rising edge of CLK at point C.
- If a synchronous reset had been used, OUT would still be high at point C.

Flancter Variation #2

Another variation on the Flancter is to have OUT come up high after global reset. This is easily achieved by making sure that either FF1 or FF2 (but not both) uses an async preset instead of an async clear. Figure 9 shows ~~what~~ I mean:

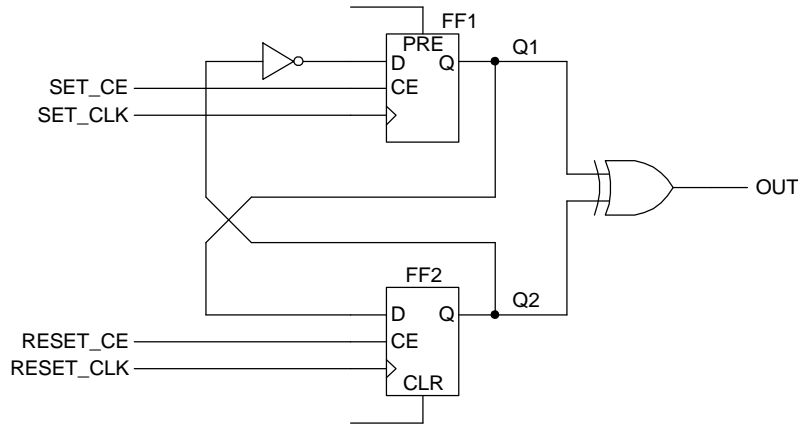


Figure 9 - Flancter Variation #2.

Flancter Variation #3

This variation operates in three different clock domains.

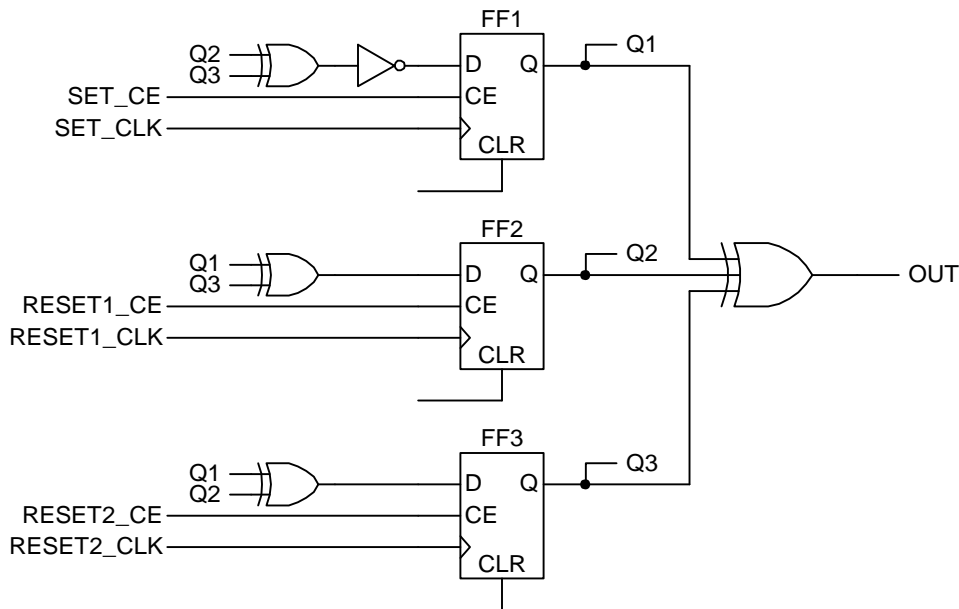


Figure 10 - 3-Way Flancter.

The particular configuration shown in Figure 10 is set by one clock domain and reset by either of two other clock domains. Note that the flipflop connected to the setting domain, FF1, has an inverter on its D input, while the flipflops connected to the resetting domains, FF2 and FF3, do not. This variation works equally well with two setting domains and one resetting domain. The rule is to use inverters on the flip-flops in the setting domains and no inverters on the flipflops in the

resetting domains. The inputs to the XOR gates are simply the Q outputs of all the registers except the one we are driving. For instance, the D input of FF1 uses the XOR of Q2 and Q3. Note that Q1 is not used in the input function to FF1. Likewise, FF2 is based on Q1 and Q3, but not Q2 and FF3 is based on Q1 and Q2, but not Q3.

A Flancter variation can be built with any number of setting and resetting clock domains; this is an n -way Flancter. The rules for building an n -way Flancter are:

- Use n flip-flops, one for each clock domain (FF1...FF n).
- The output of the n -way Flancter is simply the XOR of all of the Q outputs.
- The D input to any particular flipflop (call it FF m), is the logical XOR of the Q outputs of all the flip-flops except the one we are driving (Q1...Q n except Q m).
- An inverter is inserted at the D inputs of a flip-flop based on whether that particular flip-flop is used for setting or resetting the n -way Flancter. Use inverters for setting flip-flops or no inverters for resetting flipflops.
- Designing a protocol in which only one flipflop is clocked at a time is the biggest problem with the n -way Flancter. Remember, the Flancter is only reliable when the flip-flops are never clocked simultaneously.

Summary

I turned 37 this year and I got to thinking that all the "greats" did their bestwork 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.