

Asynchronous Resets

Asynchronous resets are the preferred reset approach. However, asynchronous resets alone may be dangerous. Many engineers like the idea of being able to apply the reset to their circuit and have the logic go to a known state. The biggest problem with asynchronous resets is the reset release, also called reset removal. Asynchronous reset flip-flops incorporate a reset pin into the flip-flop design. The reset pin is typically at active low (the flip-flop goes into the reset state when the signal attached to the flip-flop reset pin goes to a logic low level)

Coding Style and Example Circuit

The Verilog code shows the correct way to model asynchronous reset flip-flops. The reset is part of the sensitivity list. For Verilog, adding the reset to the sensitivity list is what makes the reset asynchronous. In order for the Verilog simulation model of an asynchronous flip-flop to simulate correctly, the sensitivity list should only be active on the leading edge of the asynchronous reset signal. Hence, as shown in the example, the always procedure block will be entered on the leading edge of the reset, then the 'if' condition will check for the correct reset level. In Verilog simulation, if the sensitivity list were sensitive to more than just the active clock edge and the reset leading edge, the simulation model would be incorrect. Additionally, only the clock and reset signals can be in the sensitivity list. If other signals are included, the simulation model would not be correct for a flip-flop. For VHDL, including the reset in the sensitivity list and checking for the reset before the "if clk'event and clk = 1" statement makes the reset asynchronous. Also, note that the reset is given priority over any other assignment (including the clock) by using the **if/else** coding style. Because of the nature of a VHDL sensitivity list and flip-flop coding style, additional signals can be included in the sensitivity list with no ill effects directly for simulation and synthesis. However, good coding style recommends that only the signals that can directly change output of the flip-flop should be in the sensitivity list. These signals are the clock and the asynchronous reset. All other signals will slow down simulation and be ignored by synthesis.

```
module async_resetFFstyle (q, d, clk, rst_n);
output q;
input d, clk, rst_n;
reg q;
    // Verilog-2001: permits comma-separation
    // @(posedge clk, negedge rst_n)
    always @(posedge clk or negedge rst_n)
        if (!rst_n)
            q <= 1'b0;
        else
            q <= d;
endmodule
```



Advantages of Asynchronous Resets:

The advantage in using asynchronous resets is that, as long as the vendor library has asynchronously resettable flip-flops, the data path is guaranteed to be clean. Designs that push the limit for data path timing, cannot afford to have added gates and additional net delays in the data path due to logic inserted to handle synchronous resets. This argument does not hold if the vendor library has flip-flops with synchronous reset inputs. Using an asynchronous reset, the designer is guaranteed not to have the reset added to the data path.

The Reuse Methodology Manual (RMM) suggests that asynchronous resets are not to be used because they cannot be used with cycle based simulators. This is not true. The basis of a cycle-based simulator is that all inputs change on a clock edge. Since timing is not part of cycle-based simulation, the asynchronous reset can simply be applied on the inactive clock edge. For DFT, if the asynchronous reset is not directly driven from an I/O pin, then the reset net from the reset driver must be disabled for DFT scanning and testing, this is required for the synchronizer circuit. Some designers claim that static timing analysis is very difficult to perform with designs using asynchronous resets. The reset tree must be timed for both synchronous and asynchronous resets to ensure that the release of the reset can occur within one clock period. The timing analysis for a reset tree must be performed after layout to ensure this timing requirement is met. The biggest problem with asynchronous resets is that they are asynchronous both at the assertion and at the desertion of the reset. If the asynchronous reset is released at or near the active clock edge of a flip-flop, the output of the flip-flop could go metastable and thus the reset state of the ASIC could be lost. Another problem that an asynchronous reset can have, depending on its source, is spurious resets due to noise or glitches on the board or system reset. If this is a real problem in a system, one might think that using synchronous resets is the solution, but a similar problem exists for synchronous resets, if these spurious reset pulses occur near a clock edge, the flip-flops can still go metastable.