

2.6 Reset Design Strategy

Many design issues must be considered before choosing a reset strategy for an ASIC design, such as whether to use synchronous or asynchronous resets, will every flip-flop receive a reset etc.

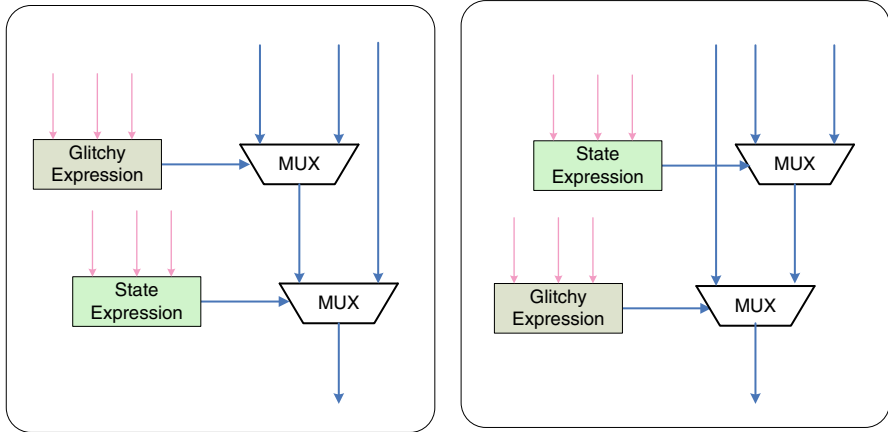


Fig. 2.28 Data path re-ordering to reduce switching propagation

The primary purpose of a reset is to force the SoC into a known state for stable operations. This would avoid the SoC to power on to a random state and get hanged. Once the SoC is built, the need for the SoC to have reset applied is determined by the system, the application of the SoC, and the design of the SoC. A good design guideline is to provide reset to every flip-flop in a SoC whether or not it is required by the system. In some cases, when pipelined flip-flops (shift register flip-flops) are used in high speed applications, reset might be eliminated from some flip-flops to achieve higher performance designs.

A design may choose to use either an Asynchronous or Synchronous reset or a mix of two. There are distinct advantages and disadvantages to use either synchronous or asynchronous resets and either method can be effectively used in actual designs. The designer must use an approach that is most appropriate for the design.

2.6.1 Design with Synchronous Reset

Synchronous resets are based on the premise that the reset signal will only affect or reset the state of the flip-flop on the active edge of a clock. In some simulators, based on the logic equations, the logic can block the reset from reaching the flip-flop. This is only a simulation issue, not a real hardware issue.

The reset could be a “late arriving signal” relative to the clock period, due to the high fanout of the reset tree. Even though the reset will be buffered from a reset buffer tree, it is wise to limit the amount of logic the reset must traverse once it reaches the local logic.

Figure 2.29 shows one of the RTL code for a loadable Flop with Synchronous Reset. Figure 2.30 shows the corresponding hardware implementation.

```
module load_syn_ff ( clk, in, out, load, rst_n );  
  input clk, in, load, rst_n;  
  output out;  
  
  always @(posedge clk)  
    if (!rst_n)  
      out <= 1'b0; // sync reset  
    else if (load)  
      out <= in; // sync  
  
endmodule
```

Fig. 2.29 Verilog RTL code for loadable flop with synchronous reset

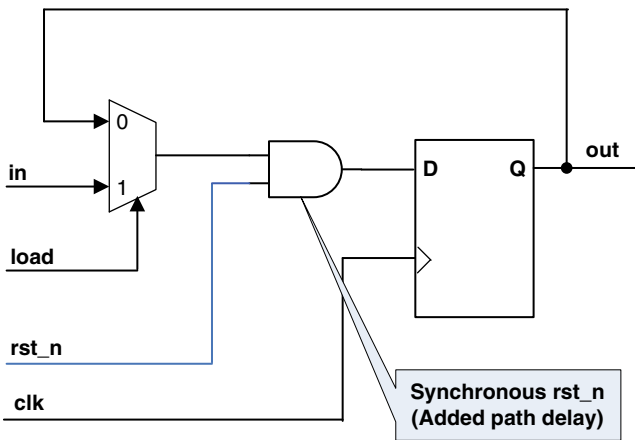


Fig. 2.30 Loadable flop with synchronous reset (hardware implementation)

One problem with synchronous resets is that the synthesis tool cannot easily distinguish the reset signal from any other data signal. The synthesis tool could alternatively have produced the circuit of Fig. 2.31.

Circuit shown in Fig. 2.31 is functionally identical to implementation shown in Fig. 2.30 with the only difference that reset AND gates are outside the MUX. Now, consider what happens at the start of a gate-level simulation. The inputs to both legs of the MUX can be forced to 0 by holding “rst_n” asserted low, however if “load” is unknown (X) and the MUX model is pessimistic, then the flops will stay unknown (X) rather than being reset. Note this is only a problem during simulation. The actual circuit would work correctly and reset the flop to 0.

Synthesis tools often provide compiler directives which tell the synthesis tool that a given signal is a synchronous reset (or set). The synthesis tool will “pull” this signal as close to the flop as possible to prevent this initialization problem from occurring.

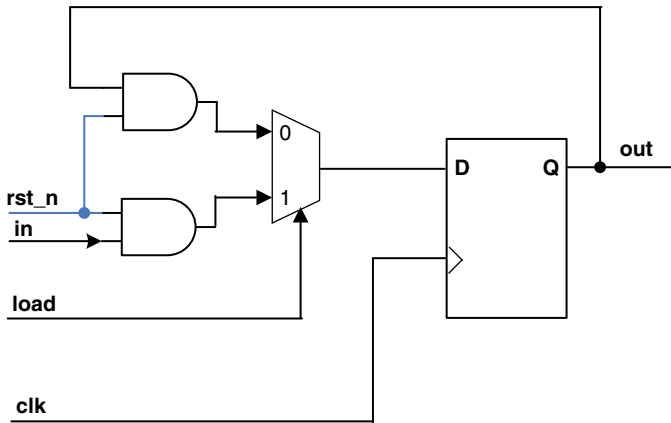


Fig. 2.31 Alternate implementation for loadable flop with synchronous reset

It would be recommended to add these directives to the RTL code from the start of project to avoid re-synthesizing the design late in the project schedule.

2.6.1.1 Advantages of Using Synchronous Resets

1. Synchronous resets generally insure that the circuit is 100% synchronous.
2. Synchronous reset logic will synthesize to smaller flip-flops, particularly if the reset is gated with the logic generating the Flop input.
3. Synchronous resets ensure that reset can only occur at an active clock edge. The clock works as a filter for small reset glitches.
4. In some designs, the reset must be generated by a set of internal conditions. A synchronous reset is recommended for these types of designs because it will filter the logic equation glitches between clocks.

2.6.1.2 Disadvantages of Using Synchronous Resets

Not all ASIC libraries have flip-flops with built-in synchronous resets. Since synchronous reset is just another data input, the reset logic can be easily synthesized outside the flop itself (as shown in Figs. 2.30 and 2.31).

1. Synchronous resets may need a pulse stretcher to guarantee a reset pulse width wide enough to ensure reset is present during an active edge of the clock. This is an issue that is important to consider when doing multi-clock design. A small counter can be used that will guarantee a reset pulse width of a certain number of cycles.
2. A potential problem exists if the reset is generated by combinational logic in the SoC or if the reset must traverse many levels of local combinational logic. During simulation, depending on how the reset is generated or how the reset is applied

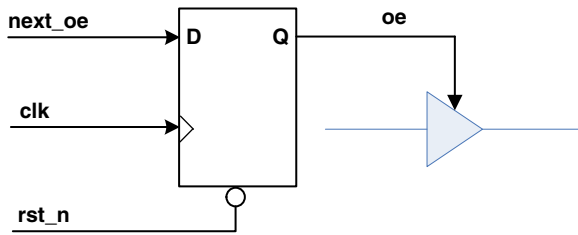


Fig. 2.32 Asynchronous reset for output enable

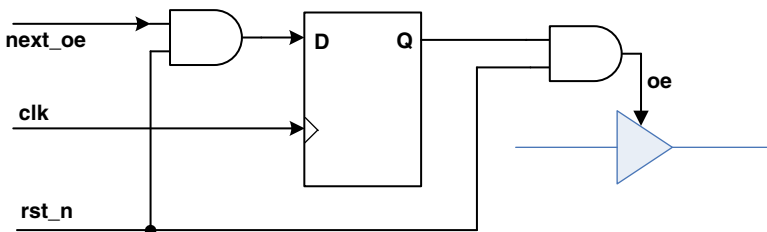


Fig. 2.33 Synchronous reset for output enable

to a functional block, the reset can be masked by X's. The problem is not so much what type of reset you have, but whether the reset signal is easily controlled by an external pin.

3. By its very nature, a synchronous reset will require a clock in order to reset the circuit. This may be a problem in some case where a gated clock is used to save power. Clock will be disabled at the same time during reset is asserted. Only an asynchronous reset will work in this situation, as the reset might be removed prior to the resumption of the clock.

The requirement of a clock to cause the reset condition is significant if the ASIC/FPGA has an internal tristate bus. In order to prevent bus contention on an internal tristate bus when a chip is powered up, the chip should have a power-on asynchronous reset as shown in Fig. 2.32.

A synchronous reset could be used; however you must also directly de-assert the tristate enable using the reset signal (Fig. 2.33). This synchronous technique has the advantage of a simpler timing analysis for the reset-to-HiZ path.

2.6.2 Design with Asynchronous Reset

Asynchronous reset flip-flops incorporate a reset pin into the flip-flop design. With an active low reset (normally used in designs), the flip-flop goes into the reset state when the signal attached to the flip-flop reset pin goes to a logic low level.

```
module load_asyn_ff ( clk, in, out, load, rst_n);  
  input clk, in, load, rst_n;  
  output out;  
  
  always @(posedge clk or nedge rst_n)  
    if (!rst_n)  
      out <= 1'b0; // sync reset  
    else if (load)  
      out <= in; // sync  
  
endmodule
```

Fig. 2.34 Verilog RTL code for loadable flop with asynchronous reset

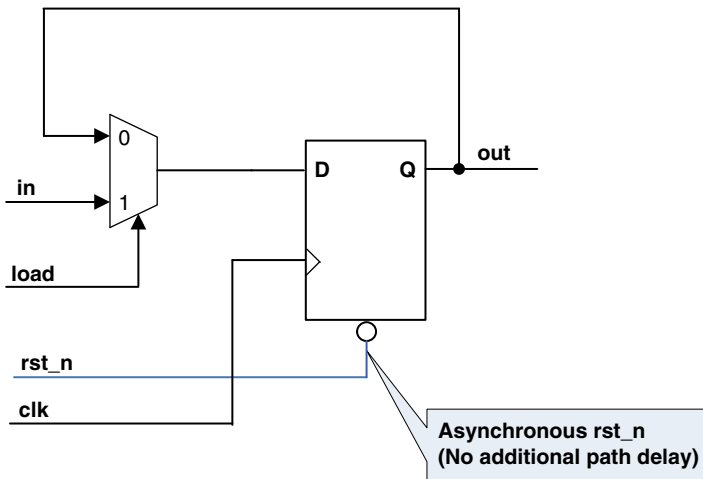


Fig. 2.35 Loadable flop with asynchronous reset (hardware implementation)

Figure 2.34 shows one of the RTL code for a loadable Flop with Asynchronous Reset. Figure 2.35 shows corresponding hardware implementation.

2.6.2.1 Advantages of Using Asynchronous Resets

1. The biggest advantage to using asynchronous resets is that, as long as the vendor library has asynchronously reset-able flip-flops, the data path is guaranteed to be clean. Designs that are pushing 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. Using an asynchronous reset, the designer is guaranteed not to have the reset added to the data path (Fig. 2.35).

```

module dff_set_reset ( clk, in, out, rst_n, set_n);
  input clk, in, rst_n, set_n;
  output out;

  always @(posedge clk or nededge rst_n or negedge set_n)
    if (!rst_n)
      out <= 1'b0; // async reset
    else if (!set_n)
      out <= 1'b1; // async set
    else
      out <= in;
endmodule

```

Fig. 2.36 Verilog RTL for the flop with async reset and async set

2. The most obvious advantage favoring asynchronous resets is that the circuit can be reset with or without a clock present. Synthesis tool tend to infer the asynchronous reset automatically without the need to add any synthesis attributes.

2.6.2.2 Disadvantages of Using Asynchronous Resets

1. 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 [30].
2. The biggest problem with asynchronous resets is that they are asynchronous, both at the assertion and at the de-assertion of the reset. The assertion is a non issue, the de-assertion is the issue. 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 SoC could be lost.
3. 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. Often glitch filters needs to be designed to eliminate the effect of glitches on the reset circuit. If this is a real problem in a system, then one might think that using synchronous resets is the solution.
4. 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. One approach to eliminate this is to use distributed reset synchronizer flip-flop.

2.6.3 Flip Flops with Asynchronous Reset and Asynchronous Set

Most synchronous designs do not have flop-flops that contain both an asynchronous set and asynchronous reset, but at times such a flip-flop is required.

Figure 2.36 shows the Verilog RTL for the Asynchronous Set/Reset Flip Flop.

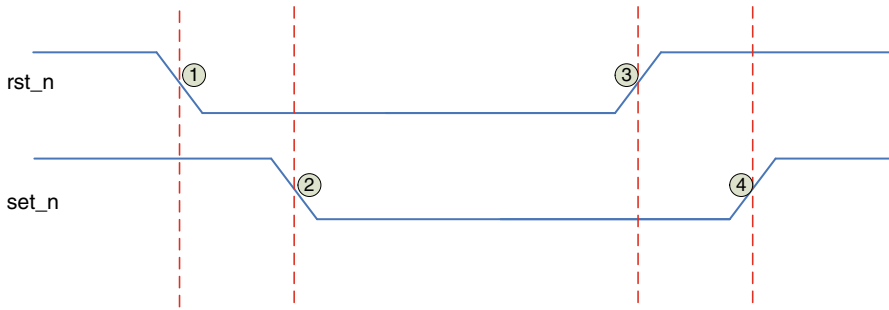


Fig. 2.37 Timing waveform for any asynchronous set/reset condition

```

// Add Compiler specific directive to
// ignore the following block during synthesis
always @(rst_n or set_n)
if (rst_n && !set_n) force q = 1;
else release q;
// End the compiler directive here
endmodule

```

Fig. 2.38 Simulation model for flop with asynchronous set/reset

Synthesis tool should be able to infer the correct flip flop with the asynchronous set/reset but this is not going to work in simulation. The simulation problem is due to the always block that is only entered on the active edge of the set, reset or clock signals.

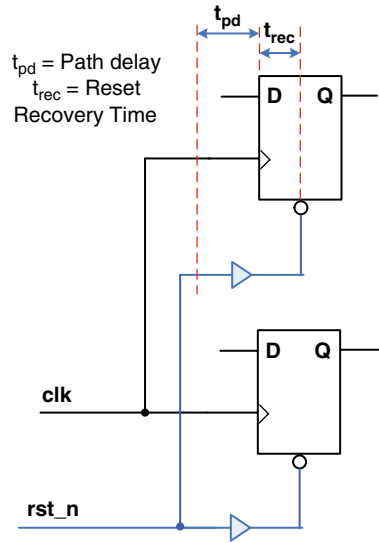
If the reset becomes active, followed then by the set going active, then if the reset goes inactive, the flip-flop should first go to a reset state, followed by going to a set state (Timing waveform shown in Fig. 2.37).

With both these inputs being asynchronous, the set should be active as soon as the reset is removed, but that will not be the case in Verilog since there is no way to trigger the always block until the next rising clock edge. Always block will be only triggered for 1 and 2 events shown in Fig. 2.37 and would skip the events 3 and 4.

For those rare designs where reset and set are both permitted to be asserted simultaneously and then reset is removed first, the fix to this simulation problem is to model the flip-flop using self-correcting code enclosed within the correct compiler directives and force the output to the correct value for this one condition. The best recommendation here is to avoid, as much as possible, the condition that requires a flip-flop that uses both asynchronous set and asynchronous reset.

The code shown in Fig. 2.38 shows the fix that will simulate correctly and guarantee a match between pre- and post-synthesis simulations.

Fig. 2.39 Asynchronous reset removal recovery time problem



2.6.4 Asynchronous Reset Removal Problem

Releasing the Asynchronous reset in the system could cause the chip to go into a metastable unknown state, thus avoiding the reset all together. Attention must be paid to the release of the reset so as to prevent the chip from going into a metastable unknown state when reset is released. When a synchronous reset is being used, then both the leading and trailing edges of the reset must be away from the active edge of the clock.

As shown in Fig. 2.39, there are two potential problems when an asynchronous reset signal is de-asserted asynchronous to the clock signal.

1. Violation of reset recovery time. Reset recovery time refers to the time between when reset is de-asserted and the time that the clock signal goes high again. Missing a recovery time can cause signal integrity or metastability problems with the registered data outputs.
2. Reset removal happening in different clock cycles for different sequential elements. When reset removal is asynchronous to the rising clock edge, slight differences in propagation delays in either or both the reset signal and the clock signal can cause some registers or flip-flops to exit the reset state before others.

2.6.5 Reset Synchronizer

Solution to asynchronous reset removal problem described in Sect. 2.6.4 is to use a Reset Synchronizer. This is the most commonly used technique to guarantee correct reset removal in the circuits using Asynchronous Resets. Without a reset synchronizer, the usefulness of the asynchronous reset in the final system is void even if the reset works during simulation.

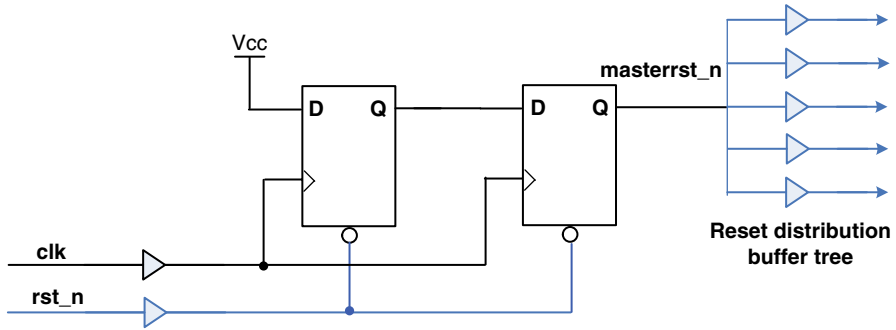


Fig. 2.40 Reset synchronizer block diagram

The reset synchronizer logic of Fig. 2.40 is designed to take advantage of the best of both asynchronous and synchronous reset styles.

An external reset signal asynchronously resets a pair of flip-flops, which in turn drive the master reset signal asynchronously through the reset buffer tree to the rest of the flip-flops in the design. The entire design will be asynchronously reset.

Reset removal is accomplished by de-asserting the reset signal, which then permits the d-input of the first master reset flip-flop (which is tied high) to be clocked through a reset synchronizer. It typically takes two rising clock edges after reset removal to synchronize removal of the master reset.

Two flip-flops are required to synchronize the reset signal to the clock pulse where the second flip-flop is used to remove any metastability that might be caused by the reset signal being removed asynchronously and too close to the rising clock edge.

Also note that there are no metastability problems on the second flip-flop when reset is removed. The first flip-flop of the reset synchronizer does have potential metastability problems because the input is tied high, the output has been asynchronously reset to a 0 and the reset could be removed within the specified reset recovery time of the flip-flop (the reset may go high too close to the rising edge of the clock input to the same flip-flop). This is why the second flip-flop is required.

The second flip-flop of the reset synchronizer is not subjected to recovery time metastability because the input and output of the flip-flop are both low when reset is removed. There is no logic differential between the input and output of the flip-flop so there is no chance that the output would oscillate between two different logic values.

The following equation calculates the total reset distribution time

$$T_{rst_dis} = t_{clk-q} + t_{pd} + t_{rec}$$

where

- t_{clk-q} = Clock to Q propagation delay of the second flip flop in the reset synchronizer
- t_{pd} = Total delay through the reset distribution tree
- t_{rec} = Recovery time of the destination flip flop

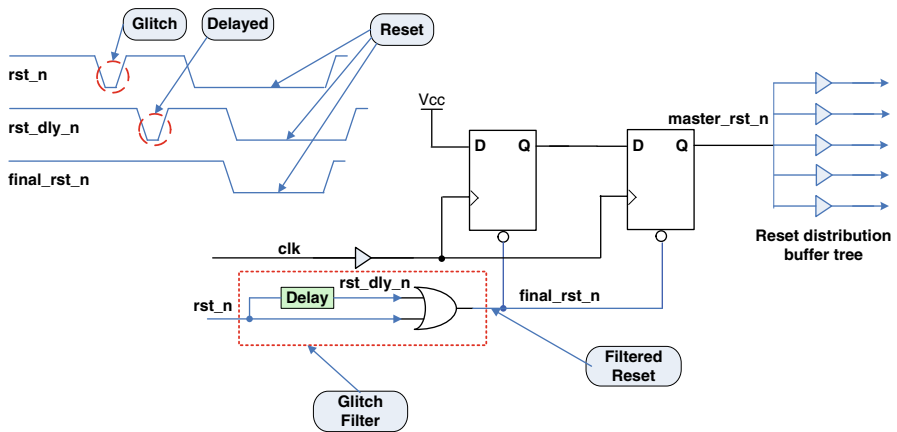


Fig. 2.41 Reset glitch filtering

2.6.6 Reset Glitch Filtering

Asynchronous Reset are susceptible to glitches, that means any input wide enough to meet the minimum reset pulse width for a flip-flop will cause the flip-flop to reset. If the reset line is subject to glitches, this can be a real problem. A design may not have a very high frequency sampling clock to detect small glitch on the reset; this section presents an approach that will work to filter out glitches [30]. This solution requires a digital delay to filter out small glitches. The reset input pad should also be a Schmidt triggered pad to help with glitch filtering. Figure 2.41 shows the reset glitch filter circuit and the timing diagram.

In order to add the delay, some vendors provide a delay hard macro that can be hand instantiated. If such a delay macro is not available, the designer could manually instantiate the delay into the synthesized design after optimization. A second approach is to instantiate a slow buffer in a module and then instantiated that module multiple times to get the desired delay. Many variations could expand on this concept.

Since this approach uses delay lines, one of the disadvantages is that this delay would vary with temperature, voltage and process. Care must be taken to make sure that the delay meets the design requirements across all PVT corners.