2.4 Impact of reset on area

Insisting on defining a global set/reset condition for every flip-flop may seem like good design practice but, it can often lead to a larger and slower design. The reason for this is because certain functions can be optimized according to the fine-grain architecture of the FPGA, but bringing a reset into every synchronous element can cause the synthesis and mapping tools to push the logic into a coarser implementation.

The next sections describe a number of different scenarios where the reset can play a significant role in the speed/area characteristics and how to optimize accordingly.

2.4.1 Resources without reset

An optimized FPGA resource will not be used if an incompatible reset is assigned to it. The function will be implemented with generic elements and will occupy more area.

In the example, the SRL16 shift-register was not used to implement the shift register as the RTL implements a synchronous reset which is not supported.

2.4.2 Resources without set

Similarly, certain resources such as a multipliers in most FPGAs do not have set resources and if used result to additional logic being synthesized.

2.4.3 Resources without asynchronous reset

DSPs and other multifunction resources are typically not flexible to varying reset strategies.

In the example, the DSP core available in the FPGA device could not be used as they only support synchronous resets.

2.4.4 Resetting RAM

Resetting RAM is usually poor design practice, particularly if the reset is asynchronous. In the example, the synthesis tool synthesis the RAM with an asynchronous reset with smaller distributed RAM blocks, additional decode logic to create the appropriate-size RAM, and additional logic to implement the asynchronous reset. If a synchronous reset is used, only a single RAM module gets synthesized.

2.4.5 Utilising set/reset flip-flop pins

Most FPGA vendors have a variety of flip-flop elements available in any given device, and given a particular logic function, the synthesis tool can often use the set and reset pins to implement aspects of the logic and reduce the burden on the look-up tables. The primary reason synthesis tools are prevented from performing this class of optimizations is related to the reset strategy. Any constraints on the reset will not only use available set/reset pins but will also limit the number of library elements to choose from.

In *setReset.sv* a resettable flip-flop was used for the asynchronous reset capability, and the OR gate was implemented in discrete logic as seen in Figure 2.4.5a.

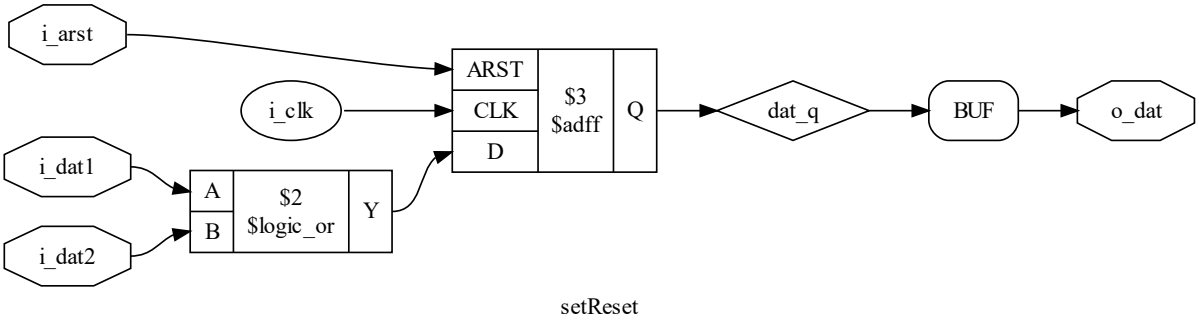

Figure 2.4.5a: FF with an asynchronous reset.

If the reset is removed and the synthesis tool chooses a resettable flop a single FF will be synthesized as shown in Figure 2.4.5b.

TODO: Figure out how to make Yosys select a SRST FF to replicate Figure 2.16 in the book.

We can take this one step further by using both synchronous set and reset signals. If we have a logic equation to evaluate in the form of: o_Dat = !i_dat3 & (i_dat1 || i_dat2). This is implemented in *setRest_2.sv* and the synthesized design is shown in Figure 2.4.5c.

TODO: Figure out how to make Yosys select a SRST FF with a set to replicate Figure 2.17 in the book.

In general, avoid using set or reset whenever possible when area is the key consideration.