Summary of Xilinx WP275

This document explains a design technique that allows FPGA designs to function in smaller sizes by minimizing the use of LUTs (Look-Up Tables).

Single and Two Level Logic

Simple logic functions in Xilinx FPGAs are implemented using LUTs and flip-flops. Even though small FPGA devices contain thousands of LUT and flip-flop pairs, keeping the design compact provides benefits in performance and efficiency.

A diagram of a circuit

AI-generated content may be incorrect. A white background with black text

AI-generated content may be incorrect.

As seen in the example above, it was able to perform simple single logic operation using a single LUT. This is the optimum way that can be used in simple logic operations. In short, we obtain output using LUT and flip-flop connected to it in simple logic operations.

A diagram of a diagram

AI-generated content may be incorrect.

As seen in the example above, if the number of variables (input variables) in our simple logic operation increases, we can see that the LUT allows up to a maximum of four inputs. If our input variables are more than four, the number of LUTs used for the design doubles. It should be noted that when two LUTs are used here, it costs twice as much as single level logic.

Adding a Reset

They have a known state due to the fact that the global reset addition has the initial state in Xilinx FPGA devices, making this global reset unnecessary most of the time. It should be noted that flip-flops support special inputs such as set, reset and clock enable controls in the design. In this section, apart from the relationship between the number of inputs shown in simple logic operations and our use of LUTs, it informs about the design by giving information about the inputs that the flip-flops to which the LUTs are connected can receive.

Adding More Controls

After explaining that flip-flops have control inputs such as set, reset and clock enable in the upper header, it opens our design by showing the inputs of our flip-flop by taking our example with four inputs similar to the examples given above.

A white screen with black text

AI-generated content may be incorrect.

Our design code is given above.

A diagram of a circuit

AI-generated content may be incorrect.

Figure 4 is the best design that the tool can build on the FPGA for our code and as can be seen it uses two LUTs.

Do Not Mix Your Drinks

The flip-flops can support both asynchronous and synchronous reset and set controls. However, asynchronous and synchronous controls cannot exist on the same flip-flop, the tool has to choose one of them. Asynchronous reset alweys takes priority and forces the selection.

A diagram of a diagram

AI-generated content may be incorrect.

There may be cases where the same flip-flop may include two reset options. The first one is global reset and the second one is local reset (for ex, a BCD counter that must roll over to 0 following a 9). These are the options. If in this case the global reset is asynchronous and the local reset is synchronous, the system will need to use a LUT to realize the local reset at twice the cost and with twice the performance.

Synchronous Design

If global reset will definitely be used in the design, at least synchronous reset can be used and synchronous can be used if it works better.

A white background with black text

AI-generated content may be incorrect.

A diagram of a circuit

AI-generated content may be incorrect.

We can see that our design uses two LUTs even when synchronous reset is used.

Get Your Priorities Right

For a successful implementation it is necessary to understand the inherent priority order of flip-flops. To understand this order of precedence, it is necessary to examine the datasheet.A table with text and numbers

AI-generated content may be incorrect.

With this table, we can see that R (reset) has the highest priority, S (set) has the second priority and clock enable has the lowest priority. After understanding these priorities, we can realize a more optimum design by shaping our design accordingly.

A diagram of a function

AI-generated content may be incorrect.

As can be seen from the Figure 7, our design will be healthier if the priority order is this way, but Asynchronous flip-flop is called FDCPE.

Writing Sympathetic Code

When we learn flip-flop priority rules, since flip-flops are the same, we can easily write optimized codes with the working conditions of our design.

A white background with black text

AI-generated content may be incorrect.

The figure above shows a sympathetic code written in FDRSE.

A diagram of a computer

AI-generated content may be incorrect.

Figure 8 is the most optimized version of our design with four inputs and other features of the flip-flop from the beginning. We can recall that other codes written for this design used two LUTs for the design and used larger sizes. It is clear that the code written with attention to priorities gives us a smaller footprint in terms of size.

Summary

In general, writing code by addressing priorities will reflect positively on us in terms of our design. In exceptional cases we can ignore priorities, but in general sympathetic code writing should be preferred. It will be in our best interest to remove the global asynchronous reset state in our designs.