Understanding *Setup and Hold Times* *One Key to Successful Designs*

Using synchronous design techniques is one essential key to creating reliable designs.

by Claude Gaschet, Xilinx Field Applications Engineer
Reptronic (France), claude@xilinx.com

The Xilinx software technology provides all the information you need to create reliable, high performance designs, but you should make sure that you don't violate two of the most important parameters: setup and hold time.

The Basics

Synchronous elements such as D flip-flops (DFF's) accept the data present on their D inputs when the clock transitions. However, the data must be stable prior to the clock edge (setup time, T_{su}) and maintained after the same clock edge (hold time, T_h); T_h can safely be ignored with Xilinx FPGAs (T_h is 0). See Figure 1.

If the setup and hold times are not violated, the data on the D input is transferred to the Q output, after the flip-flop clock-to-output delay (T_{cko}). However, if T_{su} or T_h timing is not met, the Q output is indeterminate.

Controlling Logic Paths

A real design is usually made of thousands of flip-flops, with several levels of logic between them, and moderate to high utilization of routing

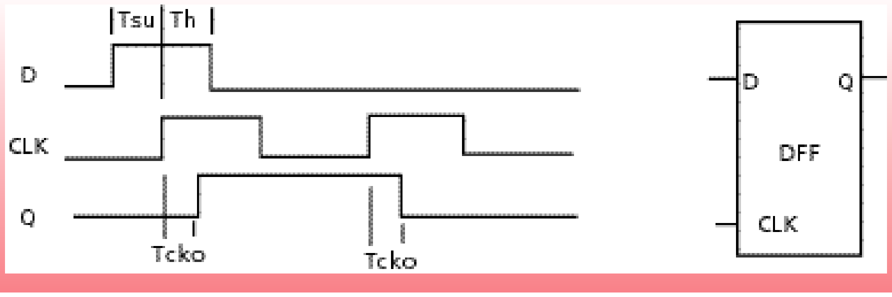

Figure 1 - Setup and hold timing.

resources. All those paths, from a data source element (PAD, FF, RAM...) to a receiving element (PAD, FF, RAM...), are known as logic paths.

Controlling those paths (giving the implementation tools the adequate directives to meet the timing criteria) is one key to success. In the Xilinx software tools, there are several graphical tools (Constraints Editor, FloorPlanner) to help you pass the right control information to the Implementation Engine. Another set of verification tools (the GUI or command-line-driven Timing Analyzer or TRCE) give you everything you need to check your results. But, what if some of your timing criteria are not met? Is your design failing because the timing constraint is not possible to meet ? If so why? Is it failing because of a setup time violation or a hold time violation? And, how do you fix the problem?

If your timing is OK, you'll get a Timing Analyzer message (in the .TWR report) that says "All Timing Constraints met." If your timing is not OK, you'll see messages that tell you why (such as "...clock frequency too high," or "...too many logic levels for the requested frequency," or " ...too much routing delays versus logic delays,") and you will be given the actual timing difference.

Two Examples of Clock Distribution

Consider a logic path starting at the Q output of a flip-flop, going through a number of logic stages, and ending at the D input of a another flip-flop, as shown in Figure 2. The safe operation is specified by the equation:

$$T_{clk} \geq T_{cko} + \text{(logic and routing delays)} + T_{su}$$

The Best Case - Minimal Clock Skew

In an ideal design, there will be minimal clock skew (all flip-flops see the clock edge at the

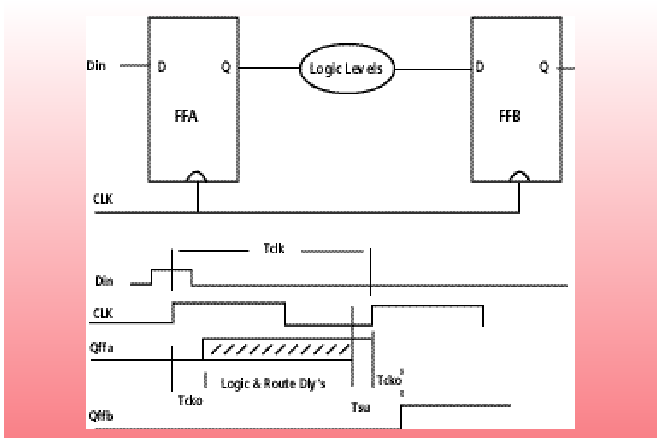

Figure 2 - Basic timing model.

same time, within a few hundred picoseconds). For synchronous designs, the optimum clock distribution is obtained when you use one of the dedicated clock routing resources attached to a global buffer (BUFG). Depending upon the FPGA you select, you have from two to eight BUFGs available.

For a given clock frequency, and for a given FPGA, you only have to control the number of logical levels (logic delays) the signal passes through, and make sure that this sum of delays does not exceed the clock period.

How to Correct the Problem

Figure 3 shows an excerpt of a report, showing a timing error. The report shows that we missed our timing goal (the slack value is negative, -0.662ns). We have a 67% logic budget for a 33% route budget (Xilinx recommends the opposite. A 50/50 ratio is generally OK for small design, and the bigger the chip, the bigger the recommended route budget (40/60 or 30/70).

The best solution is to decrease the number of logic levels; this will remove one combinatorial logic delay (T_{ilo}) and one net delay. An alternative would be to simply use a faster speed grade, because the you only need to gain 0.66ns.

```
Timing constraint:TS_CLK = PERIOD TIMEGRP "CLK"5 nS HIGH 50.000 % ;
1 item analyzed,1 timing error detected.
 Minimum period is 5.662ns.
________________________________________________________

Slack:-0.662ns path DLY1 to $Net00012_ relative to
5.000ns delay constraint

Path DLY1 to $Net00012_ contains 4 levels of logic:
Path starting from Comp:CLB_R5C4.K (from $Net00004_)
To Delay type Delay(ns) Physical Resource
Logical Resource(s)
_______________________________________________________

CLB_R5C4.XQ Tcko 1.192R DLY1
 $I3/$1I37
CLB_R5C4.F1 net (fanout=1) 0.530R DIN_Q
CLB_R5C4.X Tilo 0.959R DLY1
 $I8
CLB_R5C5.F1 net (fanout=1) 0.832R DLY1
CLB_R5C5.X Tilo 0.959R DLY2
 $I9
CLB_R6C5.F1 net (fanout=1) 0.522R DLY2
CLB_R6C5.K Tick 0.668R $Net00012_
 $I10
 $I7/$1I37
________________________________________________________-

Total (3.778ns logic,1.884ns route) 5.662ns (to $Net00004_)
 (66.7% logic,33.3% route)
```

Figure 3 - A .TWR report showing a timing violation.

The Worst Case - Significant Clock Skew

In designs where the number of clocks exceeds the dedicated BUFG clock resources, some of the clocks will need to use the non-dedicated, standard, routing resources which will introduce clock skew. It's best to assign the fastest, high fanout, clocks to the dedicated BUFG routing resources. How does clock skew affect the design? And, how do you minimize its negative effects?

In Figure 4, the FFB clock is delayed from the FFA clock (skewed) and there is a direct connection from the Q output of FFA to the D input of FFB.

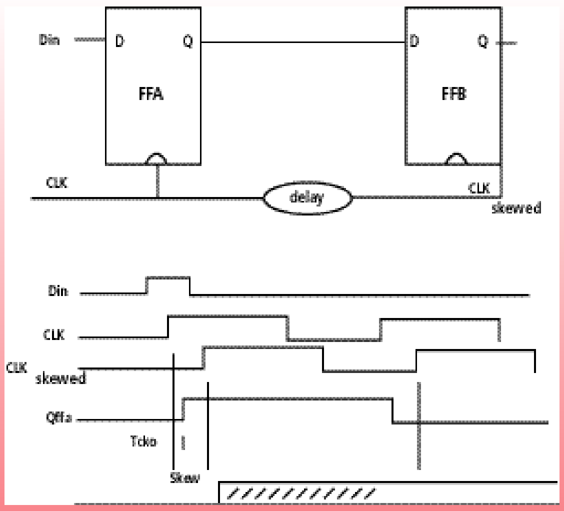

Figure 4 - Synchronous circuit with clock skew.

For FFB to clock in the correct data, you must respect the following condition :

$$T_{cko}(FFA) \geq SKEW$$

However, due to the clock delay, the D output of FFA may be unstable, changing, when FFB receives the clock edge, causing FFB to latch the wrong data. The CLK frequency does not affect the problem, therefore reducing the CLK speed won't solve the problem.

Once again, the .TWR report will give you the detailed information you need to identify and correct the problem. In the report, shown in Figure 5, you can quickly see that you missed your timing goal (the report shows negative slack, -3.234ns). The summary gives a (logic + route delay) of 3.06ns for a 6.6ns clock skew; The message "6.592ns skew between D_186 and D_188" should alert you to a probable data hold violation.

```
Slack: -3.234ns path D_186 to D_188 relative to
 6.592ns skew between D_186 and D_188

Path D_186 to D_188 contains 2 levels of logic:
Path starting from Comp: CLB_R45C25.S0.CLK (from clk_0)
To Delay type Delay(ns) Physical Resource
 Logical Resource(s)
-------------------------------------------------
CLB_R45C25.S0.YQ Tcko 1.065F D_186
 genck_0__genreg_185__ff
CLB_R45C10.S0.BY net (fanout=1) 2.127F D_186
CLB_R45C10.S0.CLK Tckdi -0.166F D_188
 genck_0__genreg_186__ff
-------------------------------------------------
Total (0.899ns logic, 2.127ns route) 3.026ns (to clk_0)
 (29.7% logic, 70.3% route)
```

Figure 5 - Timing report showing clock skew.

How to Correct the Problem

The easiest solution is to compensate for the clock skew by adding dummy logic in the data path to delay the data.

Another solution is to propagate clock and data in the opposite direction, as shown in Figure 6, (this may require manual edits, using the FPGA_Editor). You'll still get skew, and it will be reported in the .TWR, but you won't experience data loss.

Compared to the first equation, now you have the following relationship:

$$T_{clk}\text{-skew} = T_{cko} + \text{(logic and routing delays)} + T_{su}$$

This simply means that you now have less time for logic and routing delays, and you'll have to be careful not to violate the receiving flip-flop setup time, especially for high frequency operation.

Of course, the best solution would be to use BUFG to distribute the clock, which will eliminate the skew entirely.

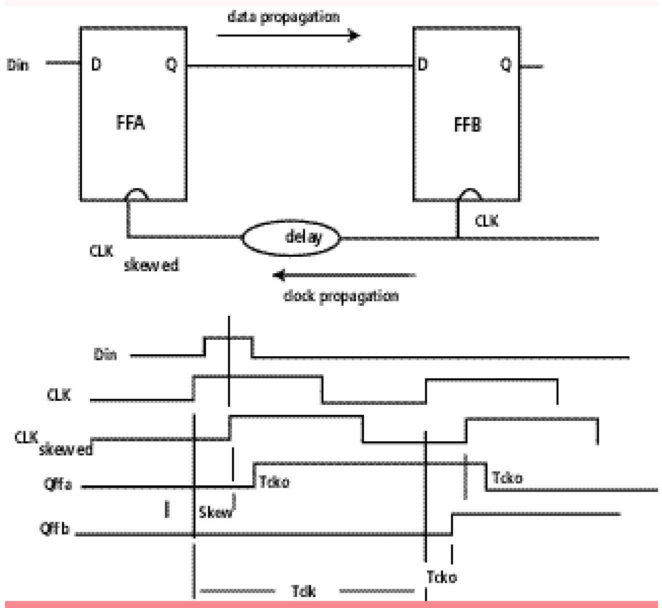

Figure 6 - Controlling the effects of clock skew through propagating the clock in the opposite direction.

Conclusion

The techniques discussed here apply to any type of logic design, not just programmable logic. However, it's easy to get reliable results with Xilinx components because they include special clock distribution networks (BUFGs). In our newer families (Virtex, Virtex-E, SPARTAN-II) we also include Delay Locked Loops that eliminate skew for both on-chip and off-chip clocks. Σ

If you need more help, go to www.support.xilinx.com