Chapter 14
Modal Analysis

Today's designs are very complex. They are "System on a Chip" in the real sense. The same chip performs multiple functions at different points of time. Within the chip also, there are portions in the design which behave one way in one use mode and behave differently in another use mode.

14.1 Usage Modes

A portion of the design might have one requirement for one kind of operation. And, for a different kind of operation the same portion of the design might have a different requirement.

The best example could be a design in the video entertainment segment. In the video world, user experience is a major requirement. In order to provide a real-like user experience, performance becomes the key factor. On the other hand, when the user is not using the device for a video application, performance is no longer important. Rather, it is more important to conserve battery life (thus, power) – even if performance has to be scaled down significantly.

Thus, parts of the device could have changing requirements – depending upon what mode the device is currently in. Individually, each part of the design has to meet the requirements of each of the individual modes.

14.2 Multiple Modes

For the sake of simplicity, let us say, a device has two major usage scenario – represented as modes *M1* and *M2*. Let us further assume, there are two parts – *P1* and *P2* in the design, for which the timing requirements change depending upon whether the device is being used in mode *M1* or *M2*.

S. Gangadharan and S. Churiwala, *Constraining Designs for Synthesis and Timing Analysis: A Practical Guide to Synopsys Design Constraints (SDC)*,
DOI 10.1007/978-1-4614-3269-2_14, © Springer Science+Business Media New York 2013

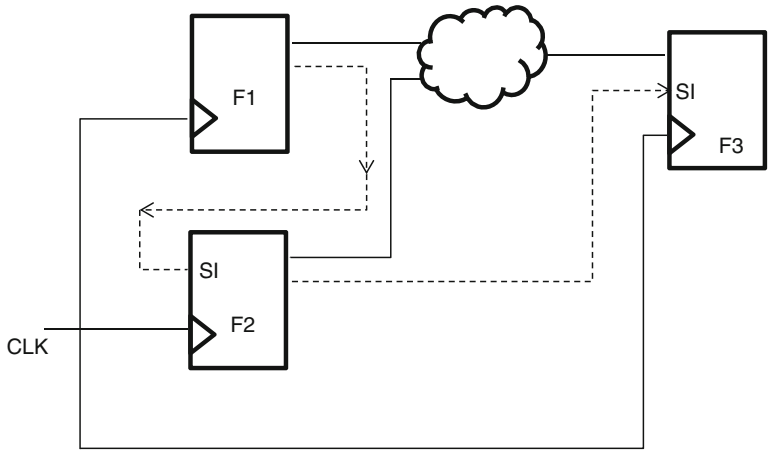

Fig. 14.1 Functional and test mode

Now, *P1* has to meet the requirements of both the modes, *M1* and *M2*. Thus, *P1* has to be designed to meet the most restrictive of the requirements. Similarly, *P2* has to be designed to meet the most restrictive of the requirements among *M1* and *M2*. However, there is no situation when *P1* will be operating in mode *M1* while *P2* would be operating in mode *M2*. Both *P1* and *P2* will together operate in mode *M1* or in *M2*.

Let us consider the circuit shown in Fig. 14.1.

The paths shown with solid lines indicate functional paths – which are active when the circuit is in normal operation. The paths shown with dotted lines indicate scan paths – which are active during Scan Shift. The same *CLK* port is used for *SystemClock* during functional mode and *TestClock* for Scan mode.

The *SystemClock* usually operates at a higher frequency, say a period of *10ns*; while *TestClock* usually operates at a lower frequency, say a period of *40ns*.

The path $F1 \rightarrow F3$ is a functional path and should meet the timing corresponding to *10ns* period. The path $F1 \rightarrow F2/SI$ is a scan path and should meet the timing corresponding to *40ns*.

We need to specify *SystemClock* so that path $F1 \rightarrow F3$ gets timed correctly. We also need to specify *TestClock* so that path $F1 \rightarrow F2$ gets timed. Since both *SystemClock* and *TestClock* share the same port, both clocks will be declared at the same location – which is *CLK* port. Now, during timing analysis, each of the paths will get analyzed corresponding to both *SystemClock* as well as *TestClock*. Thus, path $F1 \rightarrow F2$ will be timed corresponding to *SystemClock* also – which is an over-kill. The path will be forced to meet *10ns*, when *40* is good enough.

However, at any time the device will be in only one mode – either it will be in normal operation or it will be under scan mode. If it is in functional mode, the path $F1 \rightarrow F2$ is not of interest. And, when the path $F1 \rightarrow F2$ is of interest, the device is in scan mode.

In such situations, we can define two different modes for the device. We could define a functional mode. In this mode, it analyzes paths $F1 \rightarrow F3$ and $F2 \rightarrow F3$ using *SystemClock*. And, we could define another mode for scan. In this mode, it analyzes path $F1 \rightarrow F2$ using *TestClock*.

14.3 Single Mode Versus Merged Mode

A user could write the constraints for each mode individually, or write a set of constraints which are combined for multiple modes.

Usually front end designers who write the RTL code to represent the functionality find it easier to comprehend the design in terms of various functional modes. It comes more naturally for them to think of the design in terms of functional mode. Hence, they prefer to write the constraints for each mode individually.

In Sect. 14.6, we will see some of the challenges that arise due to individual mode constraints. Because of those challenges, the backend designers tend to merge the constraints. For them, the design is usually less about the functionality. They look at the design as a network of logic elements, and don't tend to think in terms of individual functional modes.

14.4 Setting Mode

When an SDC represents a single mode, certain points in the design can be fixed at specific values that are unique characteristics of that mode. The SDC command for setting a specific value is *set_case_analysis*. The SDC syntax for the command is:

set_case_analysis value port_pin_list

where, value can be *0/1/rising/falling*.

The command fundamentally conveys that for the current analysis assume that a given object is at the specified value or transition.

For putting a device into a specific mode, sometimes just one *set_case_analysis* might be sufficient. And, sometimes, a set of several *set_case_analysis* might be needed to put the device into a specific mode.

Figure 14.2 shows the same circuit as Fig. 14.1 – but with some more details.

For this example to be analyzed in the functional mode, the clocks will be declared as:

create_clock -name SysClk -period 10 [get_ports CLK]

In addition, we should apply

set_case_analysis 0 [get_ports SE]

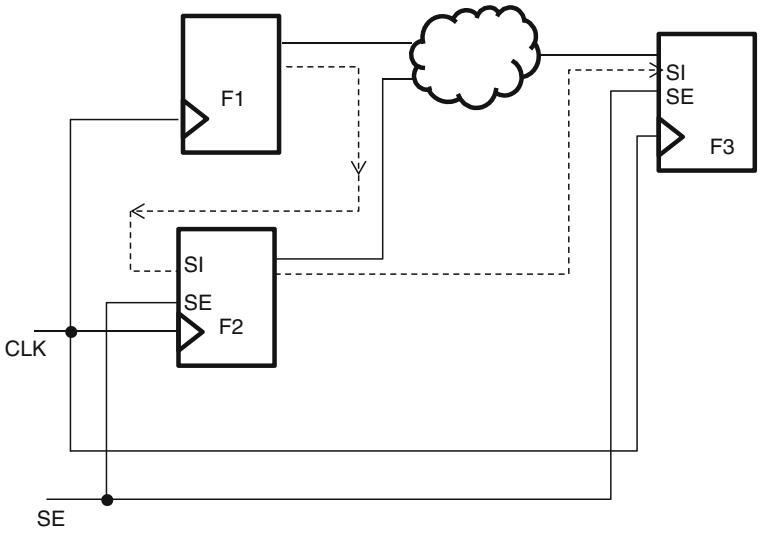

Fig. 14.2 Scan pins shown for the previous circuit

The flop models contain the information that when *SE* pin is *0*, only *D* pin can be sampled. Thus, the paths to *SI* pin will not be analyzed, but the paths to *D* pin will be analyzed. Thus, path $F1 \rightarrow F2$ will not be analyzed in this mode, because this path reaches *SI* pin of *F2*.

On the other hand, if we want the example to be analyzed in the scan mode, the corresponding commands would be:

create_clock -name TstClk -period 40 [get_ports CLK]
set_case_analysis 1 [get_ports SE]

Again, because flop model contains the information that when *SE* pin is *1*, only *SI* pin can be sampled. Thus, the paths to *D* pin will not be analyzed but the paths to *SI* pin will be analyzed. Thus path $F1 \rightarrow F2$ will be analyzed in this mode. Also, the path from $F2 \rightarrow F3$'s *SI* pin will also be analyzed.

Let us consider a block, which has several possible modes of operation. There is a configuration register of 8 bits whose setting decides the specific mode of operation. In such a case, all 8 bits of the register might need to be set in order to put the device in the mode of interest. Example commands could be something like:

set_case_anlaysis 0 [get_pins config_reg[0]/Q]
set_case_anlaysis 1 [get_pins config_reg[1]/Q]
set_case_anlaysis 1 [get_pins config_reg[2]/Q]
set_case_anlaysis 0 [get_pins config_reg[3]/Q]
set_case_anlaysis 1 [get_pins config_reg[4]/Q]
set_case_anlaysis 0 [get_pins config_reg[5]/Q]
set_case_anlaysis 1 [get_pins config_reg[6]/Q]
set_case_anlaysis 0 [get_pins config_reg[7]/Q]

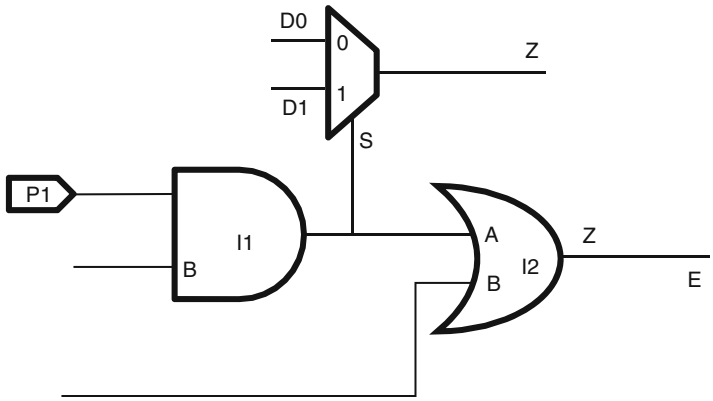

Fig. 14.3 Case analysis impact on paths being timed

In order to decide the *set_case_analysis* settings, we first need to decide the mode of operation for which the analysis has to be done. After that we need to decide the control pins/ports which influence the mode of operation for the device. Then, we need to specify those pins/ports to be at the values which will put the device into that mode of operation. Usually, *set_case_analysis* is applied only on ports or on register output pins. Usually, the register pins are used to set values on configuration registers. Even though, the syntax allows the values to be applied on any pin.

set_case_analysis command prevents certain paths from participating in timing analysis. This prevention happens in multiple ways. First, the specific pin being constant does not originate or transmit any transition. Second, the values specified through *set_case_analysis* propagate to the rest of the design – thus putting additional constants in the design. Third, these constants (either specified directly through *set_case_analysis* or after propagation) disable certain paths from being timed. Circuit shown in Fig. 14.3 provides an example of how the values applied through *set_case_anlaysis* propagate and disable certain paths from participating in timing analysis.

Let us assume that in order to set the device into a specific mode, the following constraint has been specified:

set_case_analysis 0 [get_ports P1]

So, any path involving a transition on *I1/A* no longer participates in timing analysis, as *I1/A* is always held at a constant value. A transition on *I1/B* will also not reach *I1*'s output pin. Hence, any path involving a transition on *I1/B* will also not participate in timing analysis. The value of *0* on *I1/A* propagates to the output of the *AND* gate and then to the *Sel* pin of the *MUX*. Because *MUX*'s *Sel* pin is held at constant, so, paths through this pin will also not be timed. And, paths through *D1* pin of the *MUX* will also not participate – because, *MUX*'s *Sel* pin being at *0* means *D1* will not reach the output. Only the paths through *D0* to output of *MUX* and the *I2/B* to *I2/Z* will be timed.

14.5 Other Constraints

Using the *set_case_analysis* command we can set specific points in a design to fixed logical values which characterize a specific mode of operation. Besides setting the logical values, the constraints for a specific mode also means setting other constraints like clock definitions, input and output delays etc. also which are specific to the intended mode of operation. Thus for the same input port, in one mode, it could have one input delay and in another mode, it could have another input delay.

For example, let us say, an input port receives data with respect to one clock in one mode of operation and data with respect to another clock in another mode of operation. In this case, for each mode, the input would be constrained with respect to one clock only (corresponding to that mode).

Or, an input might receive signals at different time in different mode of operation. In such cases also, the input delay specified in a specific mode is usually the value corresponding to that mode of operation.

In short, while writing constraints for a specific mode, the constraints are written as if that is the only mode in which the device will operate.

14.6 Mode Analysis Challenges

The advantage of analyzing individual modes is that certain timing paths which are never possible in the actual device operation get excluded from timing analysis, e.g., referring to Fig. 14.1, $F1 \rightarrow F2$ path in functional mode need not be timed. However, mode analysis also has its own challenges.

14.6.1 Timing Closure Iterations

Let us consider a design with four different modes – *M1*, M2, M3, and *M4*. The design has to meet the timing for each mode individually. The designer will synthesize the design for any one mode – say *M1*. Now, if timing analysis is done for *M1*, the timing might be clean. The same design also needs to be analyzed and made timing clean for mode *M2*. It is possible that certain paths which are applicable in *M2* were not to be analyzed in *M1*. These paths might potentially fail timing, when subjected to timing analysis in mode *M2*. So, some fixes will have to be made into the design – so that these paths also start meeting the timing. After mode *M2* is also timing clean, the timing analysis will need to be done for mode *M3*. Once again, it is possible that some paths valid in mode *M3* might fail the timing. So, some fixes will have to be made once again – so that these paths also start meeting the timing. Similarly, analysis will be needed for mode *M4*, which might cause some more fixes.

By now, each of the modes has been individually analyzed and where needed, fixes were also made. However, as part of making these fixes, the design has been altered. Any timing analysis done before the design was last altered is no longer

valid. Thus, timing analysis will need to be done once again for each of the modes after the last update.

When we redo the timing analysis for mode *M1* – it is possible that some path might fail timing now. During the previous iteration of mode *M1*, this same path was meeting the timing. As part of making fixes for other modes, it is possible that this path was diverted through a longer route. And, as part of fixing this, it is possible that some other path is diverted through a longer route – which could potentially cause some other mode to fail.

Thus, the whole process goes into a loop, where a fix of a path in one mode causes another path applicable in another mode to have broken timing. The fundamental problem is that implementation tools have been made to see only a subset of the paths at any given time and they try to meet only those paths. In the process, they might deteriorate paths which are not being seen by them in the current mode. At this time, the tools are not able to see that the paths which are being deteriorated could be important in another mode, causing a failure in that other mode.

Today, with complete systems on a single chip, many of the designs have more than ten modes. So, there are too many analysis required; and there is always a risk of this going into a loop. This loop is commonly referred as timing closure iterations.

As a solution to this problem, many designers try to combine various modes into a single hypothetical mode. The concept is called Mode Merge. We will read more about it in the next chapter.

14.6.2 Missed Timing Paths

We have seen earlier in this chapter, that when we apply some *set_case_analysis*, certain paths get excluded from the timing analysis. However, it is expected that each timing path in the design is there for some specific purpose, and each path should be required to meet some timing in some mode or other. Typically, a design has millions of timing paths. In each mode, thousands of paths may get excluded from timing analysis.

However, each timing path should get covered in at least some mode or the other. There is no good way of knowing that each path been covered. Effectively, there is a risk that some specific path got excluded from each of the mode settings and was not timed at all in any of the modes. This could happen because the *set_case_analysis* used for some mode were incorrect; or because a certain mode was not considered for analysis.

14.7 Conflicting Modes

Because of the problem related to timing closure iteration mentioned above, sometimes, users will set different parts of the design in different modes, which might even be conflicting. They do this so that not many different modes have to be created. Let us consider the circuit shown in Fig. 14.4.

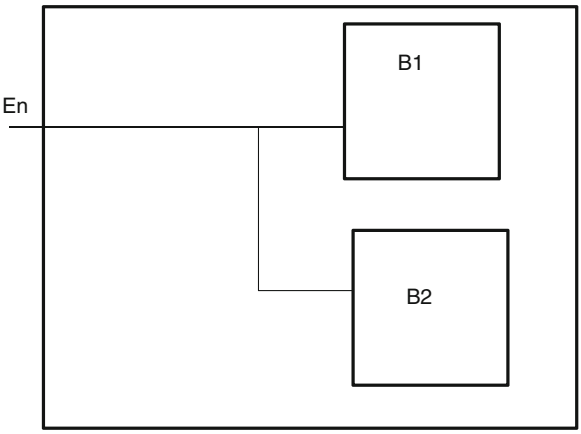

Fig. 14.4 Conflicting mode settings

Let us assume that the design pin *En* controls the operational mode of the design. Hence, for two different runs, the *case_analysis* settings would be:

set_case_analysis 0 [get_ports En]

and

set_case_anlaysis 1 [get_ports En]

As mentioned in the previous section, this will mean doing timing analysis twice. Sometimes, this could also cause iterations through the implementation tools.

Let us further assume that the block *B1* has the most restrictive timing when its pin *En* is at *0*, while the block *B2* has the most restrictive timing when its input pin *En* is *1*. In such a case, some designers prefer to specify:

set_case_analysis 0 [get_ports En]
set_case_anlaysis 1 [get_pins B2/En]

It should be noticed that it is never possible to have the above situation in the design, since *B2/En* is being driven directly by the *En* port. However, this allows the design to be put into the most restrictive situation and do the analysis only once.

Similarly, sometimes, conflicting values are set at flops. A *set_case_analysis* set somewhere could propagate a *0 at* a specific flop's input, while the flop's output might have a *1* set at it. Again, not something that is actually possible in the design. However, this covers the situation, where the timing was supposed to be most restrictive in one condition till the flop; and after the flop it is most restrictive in an opposite condition.

Merging of several SDC files belonging to different modes into one SDC file is dealt in more detail in the next chapter. This example has been given mostly to show that sometimes, mode settings could be made in a conflicting manner, even though, logically these situations may never occur in the actual design.

14.8 Mode Names

It should be noted that several times in this chapter, we refer to mode names. However, SDC does not provide any mode naming convention/command. Individual tools might still provide a mechanism to provide a name to the mode. In the context of SDC, any name for the mode is mostly for understanding of the user – as to which functionality does he want to cover, by the corresponding *set_case_analysis* commands.

14.9 Conclusion

Mode analysis allows a user to restrict the analysis to specific operational situations only, rather than considering all possible combinations of paths and situations, some of which might never happen in the design. Mode analysis makes it easier for the designer to write constraints only for specific operational modes. However, dealing with only a subset of paths for one mode, without any consideration for the other paths, which will be meaningful in other modes often causes a long iterative loop through the timing closure.

For most designs, front end designers generate the SDC file specific to individual modes. However, the backend engineers merge several modes into one constraint file, so that the implementation tools can look at the whole set of requirements in one go.