Chapter 13
Combinational Paths

Usually, outputs are always registered – just before being presented to the port. In many cases, inputs are also registered immediately after entering the block. In any case, most of the times, each signal entering an input gets registered at least once, before it comes out through an output port.

However, sometimes there might be paths from input to output, without encountering any register. Such paths are called combinational paths. Figure 13.1 shows an example of a combinational path.

13.1 set_max_delay

A combinational path can be constrained so that the delay on the path can be limited within an upper bound. This can be done through *set_max_delay* command. The SDC syntax for this command is:

```
set_max_delay    [-rise] [-fall]
                 [-from from_list]
                 [-to to_list]
                 [-through through_list]
                 [-rise_from rise_from_list]
                 [-rise_to rise_to_list]
                 [-rise_through rise_through_list]
                 [-fall_from fall_from_list]
                 [-fall_to fall_to_list]
                 [-fall_through fall_through_list]
                 delay_value
                 [-comment comment_string]
```

The options related to path and transition specification and comment are same as those explained in Chap. 11 for *set_false_path*, and thus a detailed explanation is omitted in this chapter. The *delay_value* specifies the upper limit of the allowed

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_13, © Springer Science+Business Media New York 2013

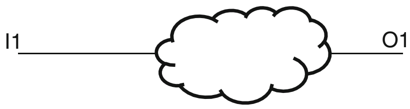

Fig. 13.1 Combinational path

delay for this combinational path. For example, if the path shown in Fig. 13.1 is allowed to have a maximum delay of *8ns*, the corresponding command would be:

set_max_delay -from [get_ports I1] -to [get_ports O1] 8.0

13.2 set_min_delay

If the path is required to have a lower bound for the delay, the requirement can be specified through set_min_delay command. The SDC syntax for the command is:

```
set_min_delay    [-rise] [-fall]
                 [-from from_list]
                 [-to to_list]
                 [-through through_list]
                 [-rise_from rise_from_list]
                 [-rise_to rise_to_list]
                 [-rise_through rise_through_list]
                 [-fall_from fall_from_list]
                 [-fall_to fall_to_list]
                 [-fall_through fall_through_list]
                 delay_value
                 [-comment comment_string]
```

The options for *set_min_delay* and *set_max_delay* are the same in meaning and syntax. The only way *set_min_delay* differs from *set_max_delay* is that this command specifies the lower bound on the delay through the path, while *set_max_delay* specifies the upper bound on the delay. Thus, the actual delay for the path has to be somewhere between *set_min_delay* and *set_max_delay*.

Usually, in most cases, there might be no need to specify *set_min_delay*. Only in some specific situation, where some hold requirements might need a minimal delay value, there would be a need for *set_min_delay* specification.

13.3 Input/Output Delay

A combinational path can also be constrained using *set_input_delay* and *set_output_delay*. The syntax and semantics of these commands and their options are described in Chap. 9 and are not being repeated here. In this section, we will describe how *set_input_delay* and *set_output_delay* can be used to constrain the delay for a combinational path.

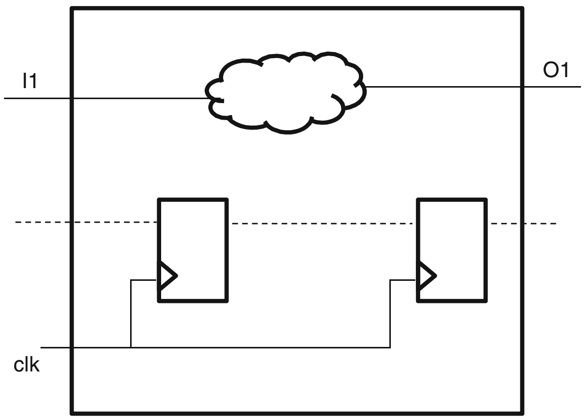

Fig. 13.2 Combinational path – no interaction with clock

Let us say that we want to constrain the combinational path shown in Fig. 13.1 to have a maximum delay of *8ns*.

13.3.1 Constraining with Unrelated Clock

Let us also say that this same block also has a clock declaration (say *CLK*) with a period of *12ns*. This clock may not have any relationship with this combinational path, as shown in Fig. 13.2.

So, out of the period of *12ns*, a duration of *8ns* needs to be available for this combinational path. The remaining *4ns* can be distributed outside this block, through *set_input_delay* and *set_output_delay*. Say, an input delay of *3* and an output delay of *1* can be specified. The distribution of *3* and *1* among *set_input_delay* and *set_output_delay* is not important, as long as the total of input delay and the output delay specification is *4*. Thus, the following set of commands can achieve a combinational path delay of maximum *8ns*:

```
create_clock -name CLK -period 12 [get_ports clk]
set_input_delay -max -clock CLK [get_ports I1] 3.0
set_output_delay -max -clock CLK [get_ports O1] 1.0
```

The risk with this style of constraining a combinational path is, if for some reason, the clock period is modified, the combinational path delay also gets modified, even though there might be no relation between the combinational path and the clock.

13.3.2 Constraining with Virtual Clock

Instead of using a clock which is being declared for this block for some other purpose, a virtual clock can be declared, just for constraining the combinational path.

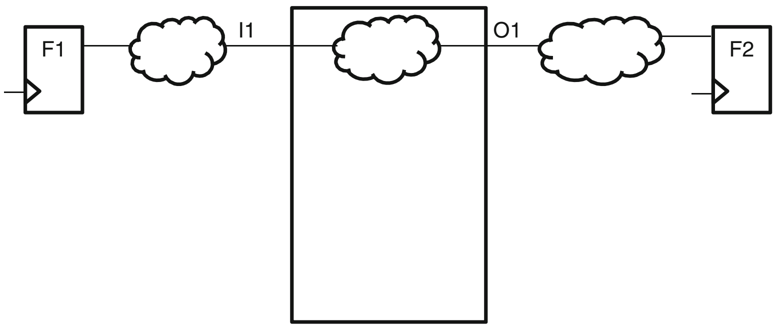

Fig. 13.3 Combinational path in the context of launching/capturing flop

We can choose whatever period we want for this virtual clock – as long as the period is more than the required max delay for the combinational path. And, the excess can be distributed among input and output delays. The following set of constraints provides one example possibility:

create_clock -name vCLK -period 15
set_input_delay -max -clock vCLK [get_ports I1] 3.0
set_output_delay -max -clock vCLK [get_ports O1] 4.0

A virtual clock has been declared with a period of *15*. Notice that there is no design object associated with the *create_clock*, thus making the clock to be virtual. With the period of *15*, there is an excess of *7ns (15–8)*. This excess has been distributed between *set_input_delay* and *set_output_delay*.

13.3.3 Constraining with Related Clock

Let us look at the same path, but this time, we also consider the circuit around this block, to show the launching and the capturing flops also – which lie outside this block. Figure 13.3 shows an example.

The launch and the capture flops lie outside the specific block. They could be lying in a different block, or they could be a part of the top-level glue logic. The input delay constraint can be specified with respect to the clock that launches data from *F1*. The input delay specified should be the delay from the launch flop till the input pin *I1*.

The output delay constraint can be specified with respect to the clock that captures data in *F2*. The output delay specified should be the delay from the output pin *O1* till the capture flop *F2*.

The clocks (which trigger *F1* and *F2*) themselves may or may not be feeding into this block. If these specific clocks are not feeding into the block, a corresponding virtual clock would need to be created.

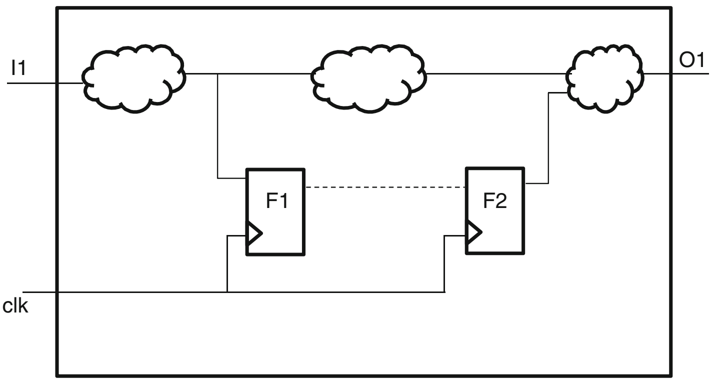

Fig. 13.4 An input/output is part of combinational as well as registered path

Among the three approaches discussed for constraining using input/output delays, this approach using the related clocks is the best. It correctly reflects the design scenario, and the allowed delay through the combinational block can then be determined automatically by the tools.

13.4 Min/Max Delay Versus Input/Output Delay

In terms of timing implication, usage of either style (viz., *set_max_delay/set_min_delay* or either of the three styles of *set_input_delay/set_output_delay*) has the same effect. However, it is preferable to constrain a combinational path using *set_input_delay/set_output_delay* combination. Let us consider the circuit in Fig. 13.4, where an input and output are part of both a combinational path as well as a registered path.

Let us say, the clock period is *15ns*. Let us say, the input arrives at *I1 at 4ns* after the clock edge. Let us say, the output *O1* has to travel for *3ns*, before it gets captured in the destination flop.

So, the delay outside the block is *7ns*. Thus, the combinational path can have a delay of *8ns* maximum. This can be specified as:

set_max_delay -from [get_ports I1] -to [get_ports O1] 8.0

Because of the input feeding into a register, there needs to be an input delay also on *I1* (in order to time *I1* to *F1* path), which will be specified as:

set_input_delay -max -clock CLK [get_ports I1] 4.0

And, in order to time *F2* to *O1* path, there needs to be an output delay also on *O1*, which will be specified as:

set_output_delay -max -clock CLK [get_ports I1] 3.0

The *set_input_delay* and the *set_output_delay* have to be anyways specified, because of *I1* and *O1*'s involvement in registered path. With these two specifications, the combinational path anyways gets constrained to *8ns*. So, there is no need for the explicit specification of *set_max_delay*.

Assuming that the *set_max_delay* is anyways specified (to *8ns*), it might be expected that these input and output delay specifications should not impact the combinational path delay (viz., *8ns*). However, in reality, the arrival time at the input and the output external delays will get counted as part of the combinational path!!!!

So, the combinational path is specified a limit of *8ns* (through *set_max_delay*). Out of that, *7ns* is contributed by the input and output delays. Thus, only *1ns* is left for the actual path. This is not what was intended. So, it is possible that for a combinational path constrained through *set_max_delay*, the effective allowed delay gets modified due to an input/output delay specification.

Now, if an input to output path is purely combinational, we would have a choice of either using only a *set_max_delay* or a combination of *set_input_delay* and *set_output_delay*. Here, since there is no need for *set_input_delay* and *set_output_delay*, it might appear as if *set_max_delay* alone is sufficient and is harmless. This is true. However, if the design gets modified so that *I1* or *O1* get involved in a registered path, they will also warrant a *set_input_delay/set_output_delay* specification. Now, this new specification of *set_input_delay/set_output_delay* ends up inadvertently modifying the max allowed delay for the combinational path.

Hence, from an ease-of-maintenance perspective, it is better to use input/output delay combination, rather than *set_max_delay*. Though the discussion was presented in terms of *set_max_delay* and *set_input_delay/set_output_delay* with *-max* specification, the same discussion holds true for *set_min_delay* and *set_input_delay/set_output_delay* with *-min* specification.

13.5 Feedthroughs

The word *feedthrough* has more than one meaning – depending upon the context. In the context of this chapter, we use the term to refer to specific types of combinational paths, wherein an input signal directly reaches an output port, without any circuit. The delay for a feedthrough path is just the wire delay inside the block. A feedthrough path often spans several consecutive blocks. The discussion mentioned in this section in the context of a feedthrough is equally applicable for other combinational paths also, if they happen to span through multiple consecutive blocks.

Let us consider a feedthrough path which spans across four consecutive blocks, as shown in Fig. 13.5.

Let us assume that the total path delay from *S*(ource) to *D*(estination) is supposed to be within *13ns*. Let us assume that the delay within each block is supposed to be maximum *2ns* and that the time of flight from one block to another block can be maximum *1ns*. The time of flight for *S* to *B1* and from *B4* to *D* also has a maximum limit of *1ns* each. So, the total path delay stays within *13ns*.

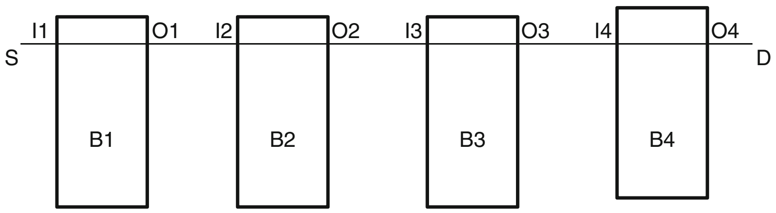

Fig. 13.5 Feedthrough path spanning multiple blocks

We will now apply constraint on all these blocks. One way is to specify the following for each of the blocks. The actual port names have to be specified in the command below:

set_max_delay -from <input port> -to <output port> 2.0

However, we have seen in the previous section that it is better to specify *set_input_delay/set_output_delay*, rather than *set_max_delay*.

For *B1*, the signal arrives at *I1* within *1ns*. After it comes out of *O1*, it has to travel for a max of *10ns* (*3* blocks * *2ns* each + *4* top-level routing * *1ns* each).

For *B2*, the signal arrives at *I2* within *4ns* (*1* block * *2ns* + *2* top-level routing * *1ns* each). After it comes out of *O2*, it has to travel for a max of *7ns* (*2* blocks * *2ns* each + *3* top-level routing * *1ns* each).

For *B3*, the signal arrives at *I3* within *7ns* (*2* blocks * *2ns* each + *3* top-level routing * *1ns* each). After it comes out of *O3*, it has to travel for a max of *4ns* (*1* block * *2ns* + *2* top-level routing * *1ns* each).

For *B4*, the signal arrives at *I4* within *10ns* (*3* blocks * *2ns* each + *4* top-level routing * *1ns* each). After it comes out of *O4*, it has to travel for a maximum of *1ns* (*1* top-level routing * *1ns*).

Assuming a clock *CLK* has already been created with a period of *13ns*, the constraints would be specified as:

For *B1*:
set_input_delay -max -clock CLK [get_ports I1] 1.0
set_output_delay -max -clock CLK [get_ports 01] 10.0

For *B2*:
set_input_delay -max -clock CLK [get_ports I2] 4.0
set_output_delay -max -clock CLK [get_ports 02] 7.0

For *B3*:
set_input_delay -max -clock CLK [get_ports I3] 7.0
set_output_delay -max -clock CLK [get_ports 03] 4.0

For *B4*:
set_input_delay -max -clock CLK [get_ports I4] 10.0
set_output_delay -max -clock CLK [get_ports 04] 1.0

It should be seen that the delay through each of the block gets constrained to a max of *2ns*.

13.5.1 Feedthroughs Constrained Imperfectly

Let us say that after the above is tried, for some reason, the timing for *B2* could not be met. Say, the delay for this block could not be reduced below *2.5ns*. So, for some other block, the delay has to be reduced. Say, for *B4*, the delay can be reduced to *1.5ns*. So, the total delay for the whole feedthrough path remains the same. However, the *set_input_delay* and *set_output_delay* for the individual blocks would need to be modified. This will change the arrival time for *B3* and *B4* (arrives *0.5ns* later), and external time on the output side of *B3* and *B2* (external required time is *0.5ns* lesser), thus changing many input/output delays – including for blocks like *B3* – for which there was no change in the routing/delays inside it.

Often, for such paths, where a feedthrough passes through several blocks, many designers do not necessarily specify the actual arrival time for input and external required time for output. Rather, they would choose a pair of input and output delay values such that the delay inside the block is the desired value. For the example described, the output delay for *B2* would be reduced by *0.5ns*. And, the input delay for *B4* would be increased by *0.5ns*. The constraints for *B3* would be left unchanged. Though, it still means *2ns* inside *B3*, the *set_input_delay/set_output_delay* no longer represents the actual time of arrival or the actual time needed to travel after coming out of the block.

Usually, on some very high-performance designs, e.g., processors – several blocks are designed as hard-IPs. In order to not impact the timing due to routing on higher layers, these IPs provide feedthrough paths. A path going from one block to another could feedthrough several IPs. Such designs might have such characteristics, where the input and output delays are different from the actual values.

13.6 Point-to-Point Exception

As shown in Sect. 13.4, usually for port to port paths, *set_input_delay* and *set_output_delay* combination is preferred compared to *set_max_delay*, even if the path is purely combinational.

Sometimes, a path segment on an entire path inside the design might need to be constrained to a special value. *set_max_delay* might be very useful for such point-to-point exceptions. Figure 13.6 shows a simple double-flop synchronizer due to an asynchronous clock domain crossing.

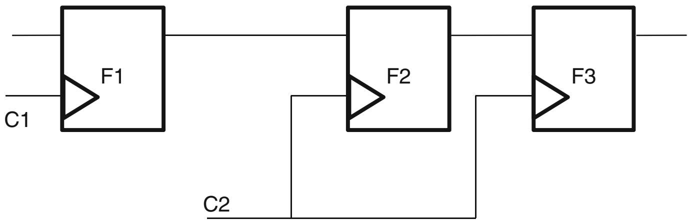

Fig. 13.6 Simple double-flop synchronization

The path from *F1* to *F2* should not be timed. This can be achieved through either *set_clock_groups* or *set_false_path*. Or, a user might put a *set_multicycle_path*. The path from *F2* to *F3* gets constrained due to clock period defined for *C2*. Usually, no logic is put between *F2* and *F3*. It is possible that placement and routing tools can place these flops far apart or take a long route, since they see the complete clock period available for this path. If the path from *F2* to *F3* is long, then the whole purpose of putting a direct path without any other logic is lost. The effectiveness of the synchronization is reduced (means MTBF (mean time between failures) will not increase as much as desired). Designers usually want that the delay from *F2* to *F3* should be very small – much smaller than the allowed clock period. So, they will constrain this path using *set_max_delay*, e.g.,

set_max_delay -from F2 -to F3 <value>

13.7 Path Breaking

Before applying a *set_max_delay* or a *set_min_delay*, the designer should understand that if the constrained portion does not start from a timing start point or end at a timing end point, these constraints break the path, at both ends of the path segment. Figure 13.7 shows a design excerpt, which has several paths, namely, *F1→F3; F1→F4; F2→F3; F2→F4*.

Each of these paths will be timed. However, let us say, for some reason, a *set_max_delay* or a *set_min_delay* is specified *-from I1/Z -to I2/A*. Now, a timing path that starts from *F1* (or *F2*) will stop at *I1/Z*, which has become a new start point. It is very important to understand this path breaking nature of *set_min_delay* and *set_max_delay*. Because of this characteristic, the paths *F1→F3; F1→F4; F2→F3 and F2→F4* will not be timed any more.

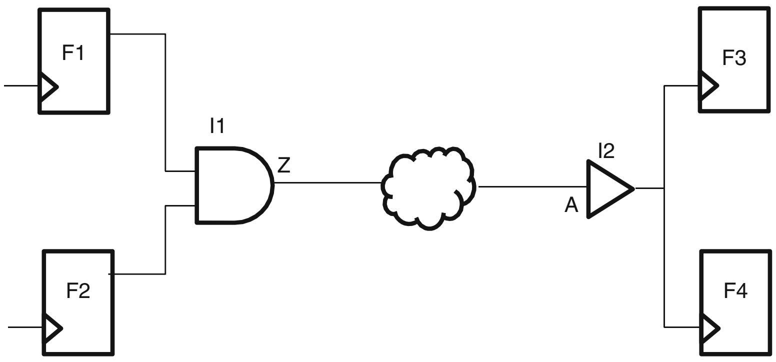

Fig. 13.7 Path breaking

13.8 Conclusion

A combinational path can be constrained using *set_min_delay* and *set_max_delay*. If the paths span from an input port to an output port, it is better to constrain the path using *set_input_delay* and *set_output_delay* combination. In general, since in most cases, the interest is in making sure that the delays are lesser than a desired value, so, *set_max_delay* is used more often than *set_min_delay*.