Chapter 7
Clock Groups

When a design has more than one clock, the timing of such a design depends not just on the frequency of clocks but also on the relation the clocks share with each other. Synchronous clocks are clocks which share a deterministic phase relationship. More often than not, synchronous clocks share the same source.

On the other hand, asynchronous clocks are clocks which don't share a fixed phase relationship. Let us consider Fig. 7.1 – if the two clocks *C1* and *C2* are generated from different sources, then they are treated as asynchronous.

The section of the design driven by each of these clocks forms a clock domain. The signals that interface between these clock domains driven by asynchronous clocks are called asynchronous clock domain crossings or abbreviated as *CDC*.

In this chapter, we will understand how to specify the relation between clocks which are asynchronous in nature and how to group them into domains. But first, let us try to understand the timing impact on a design with multi-frequency clocks.

7.1 Setup and Hold Timing Check

Let us consider Fig. 7.1. In this simple circuit, there is a launch flop (*F1*) that launches data that is captured by the capture flop (*F2*). As described in Chap. 3, setup is defined as the time by which data needs to be available before the active edge of clock, and hold is the time for which the data must remain stable after the active edge of the clock, so that data is properly registered by the flip-flop.

The same concept can be extended for the design in Fig. 7.1. The design would need to ensure that data on the active edge of the launch flop (*F1*) is captured by the closest following active edge of the capture flop (*F2*). This is called the *setup timing check*. Figure 7.2 shows the waveform of the clocks for the design.

Let us assume that t_F is the delay from *Clock* to *Q* pin of launch flop (*F1*) and t_C is the delay within the combination cloud. This means data arrives at flop *F2 at* time t_F+t_C. Let us also assume that edges of clocks *C1* and *C2* are perfectly aligned and

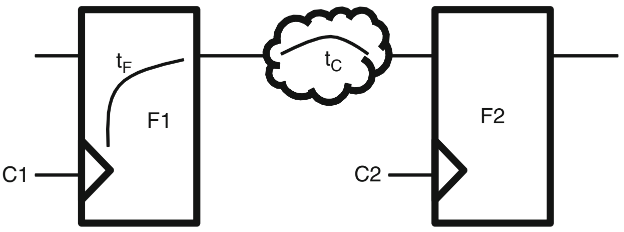

Fig. 7.1 Asynchronous clock domain crossing

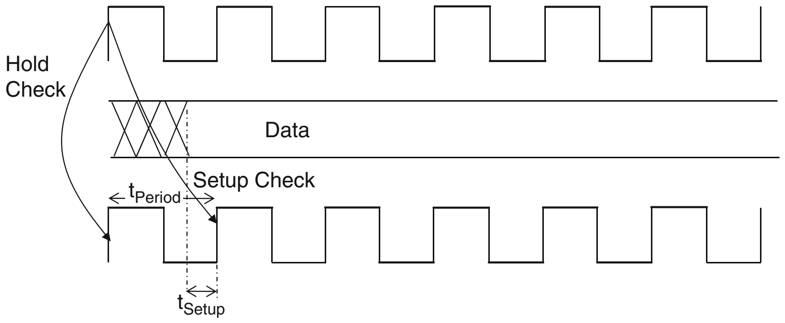

Fig. 7.2 Waveform for interacting clocks

the setup requirement of capture flop (*F2*) is t_{Setup}. Since the next clock arrives at *F2 at* the next edge, which is t_{Period} (*period* of clock *C2*), then for data from flop *F1* to be captured by *F2*, the data must arrive at least t_{Setup} time before the next active edge of *F2*. This setup timing check imposes an upper bound on the timing requirement for the signal to arrive at *F2* and can be represented as:

$$t_F + t_C < t_{Period} - t_{Setup}$$

Once the setup requirement is met, for the data to be properly captured the hold requirements have to meet as well. This is measured by the *hold timing check*, which ensures the hold timing is met between the active edge of the launch clock and the same edge of the capture clock. For the same design, since $t_F + t_C$ is the time required for the data to reach flop *F2*, the time at which the data arrives must be more than the hold time (t_{Hold}) of flop *F2*, so that the current data does not corrupt the previous data. This hold timing check therefore imposes a lower bound on the timing requirement for the signal to arrive at *F2* and can be represented as:

$$t_F + t_C > t_{Hold}$$

This was a rather simple case, since we assumed clocks *C1* and *C2* had perfectly aligned edges. The equations get just a little more complicated if the edges are not aligned (though, they still originate from the same source). If t_L is the time for the

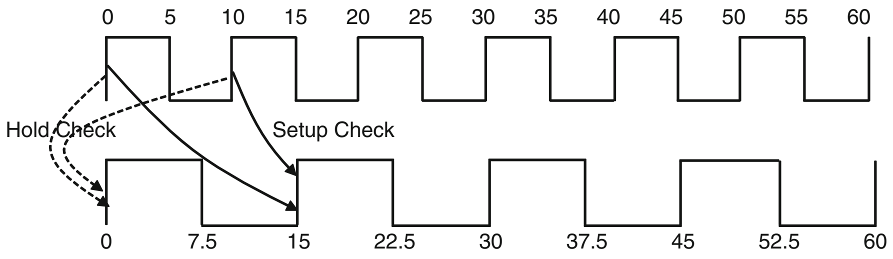

Fig. 7.3 Waveform for fast to slow clocks

clock to reach the launch flop from its source and t_Z is the time for the clock to reach the capture flop from its source, then setup and hold timing check would be:

$$t_L + t_F + t_C < t_Z + t_{Period} - t_{Setup}$$
$$t_L + t_F + t_C > t_Z + t_{Hold}$$

On the other hand, if the two interacting clocks have different frequencies, then depending on their respective frequency values, the active edge of flop (*F1*) and closest following active edge of capture flop (*F2*) may vary in every clock cycle. Here are few representative examples to analyze these further.

7.1.1 Fast to Slow Clocks

For Fig. 7.1, let us consider the case when the period of the launch clock is less than the period of the capture clock. Let us further assume that *C1* has a period of *10ns* with a *50%* duty cycle and *C2* has a period of *15ns* with a *50%* duty cycle. Let the clocks be represented as:

create_clock -period 10 -name C1 -waveform {0 5} [get_pins F1/CK]
create_clock -period 15 -name C2 -waveform {0 7.5} [get_pins F2/CK]

Figure 7.3 shows the waveform of these clocks. From this it will be evident that the waveforms repeat themselves after *30ns*. Thus, any analysis has to be done only within *30ns* window. For the setup timing check, the launch/capture combinations within the window occur at

1. Launch edge at *0* and capture at *15ns*.
2. Launch at *10ns* and capture at *15ns*.
3. Launch at *20ns* and capture at *30ns*.

Out of these, the second pair is the most restrictive and is considered for setup. Similarly, if we compute all the hold check pairs within the window, we will find that the worst case combination for hold corresponds to the launch edge at *0* and capture edge at *0*. So, both edges at *0* are chosen for hold check. This ensures that data at time unit *0 at* the launch flop is not registered by capture flop at time *0*.

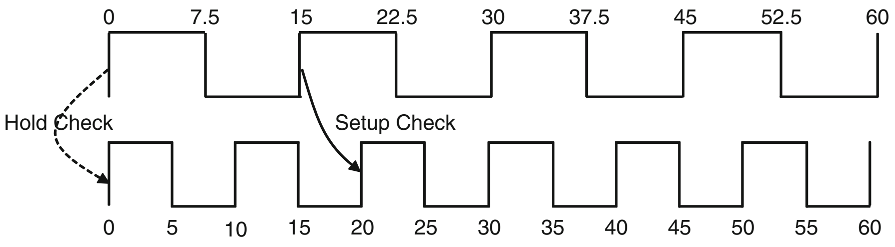

Fig. 7.4 Waveform for slow to fast clocks

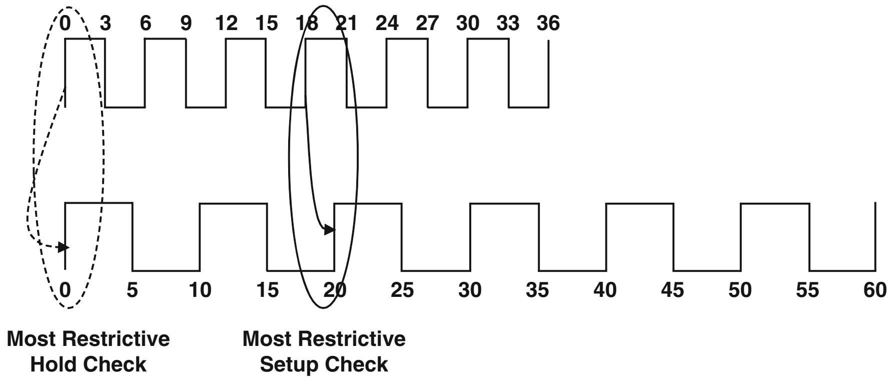

Fig. 7.5 Waveform for clocks that are not integer multiples

7.1.2 Slow to Fast Clocks

Let us look at another example of most restrictive check being used. If the period of clocks *C1* (*15*) and *C2* (*10*) are reversed, then once again all edge-pair combination till time *30ns* are considered and the most restrictive pair is used. Thus, setup timing check should be done between the launch edge at *15ns* and capture edge at *20ns*. Similarly, the most restrictive hold check is determined, which is still at time *0* for both edges. Figure 7.4 shows the waveform in this case.

7.1.3 Multiple Clocks Where Periods Synchronize in More than Two Cycles

Let us consider Fig. 7.5 where the clocks take several more cycles to realign. Let period of clock *C1* be *6ns* and period of clock *C2* be *10ns*. Assuming the clock edges are aligned at time $t = 0$, the next time their edges will align will be at time $t = 30$, which is the LCM of the two clock periods.

As it can be seen from the waveform, there are a number of edges where you can perform setup and hold check. But the most restrictive setup check is when launch is at *18ns* and capture is at *20ns*. Similarly the most restrictive hold check is when both edges are at *0*.

7.1.4 Asynchronous Clocks

As it is evident from these examples, these checks can get pretty complicated for multiple-frequency clocks. If the clocks don't share a phase relationship, the arrival of the launch clock and capture clock will not be deterministic relative to each other. This means setup and hold timing requirement could potentially vary in every cycle. This becomes a big timing problem when analyzing asynchronous clocks, if there is a signal in the data path driven by these clocks that may be interacting and creating an asynchronous clock domain crossing. This can potentially lead to certain issues like metastability. In Fig. 7.1, if the input of the flip-flop *F2* is changing while it is being captured by flip-flop *F2*, then the output of *F2* could be unstable for a certain period of time. This is called *metastability* which needs to be resolved using synchronizers. The main problem with asynchronous *CDC* is as follows: With each edge pair, there is a different timing requirement. So, at some time or other, there will be very little margin. And, since checks are supposed to be made on most restrictive pair, hence, there will be at least some edge, which will violate!

To prevent implementation tools from spending time unnecessarily to meet the timing on such paths, it is generally recommend to identify such crossings. This is achieved using *set_clock_groups*.

7.2 Logically and Physically Exclusive Clocks

Sometimes, you would have designs where clocks may not be talking to each other depending on how the design is architected. Let us consider Fig. 7.6; here the two clocks irrespective of their source don't interact with each other, even though they coexist in the design. These clocks are considered to be logically exclusive.

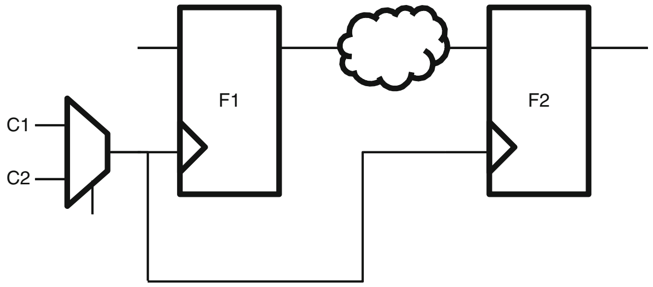

Fig. 7.6 Logically exclusive clocks (*C1* and *C2*)

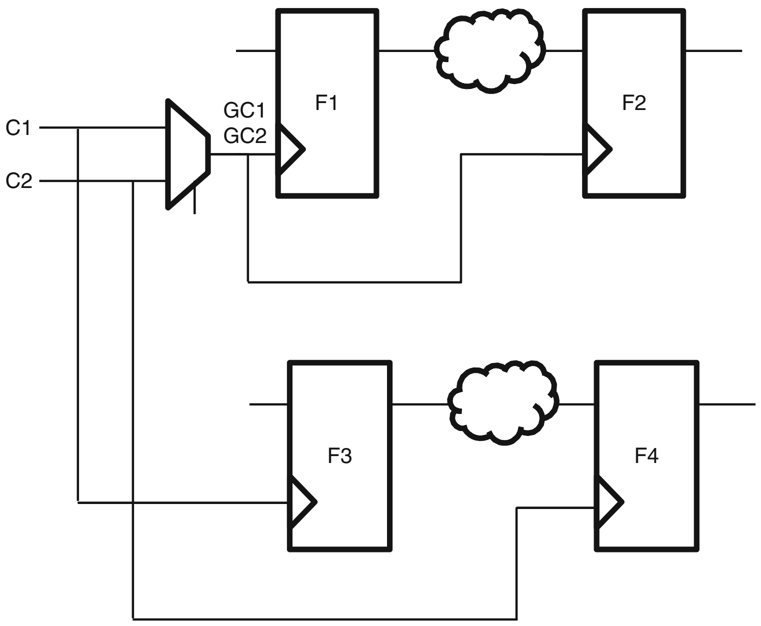

Fig. 7.7 Physically exclusive clocks (*GC1* and *GC2*)

Let us consider Fig. 7.7; here the clocks *C1* and *C2* are logically exclusive; however, the two generated clocks *GC1* and *GC2* are exclusive, but they cannot coexist together on the same net. Thus, clocks *GC1* and *GC2* are considered to be physically exclusive.

7.3 Crosstalk

When clocks are mutually exclusive, even though they don't talk, there could be interference between the signals resulting in unwanted effect. This is typically a problem seen in deep submicron technology and could be because of a number of reasons like lower geometry's requirement for higher routing density, interaction between devices, or coupling capacitance between signals. This results in a phenomenon called *crosstalk*. Let us consider Fig. 7.8.

In this figure the coupling capacitance between neighborhood nets results in unwanted and unexpected activity on the signals. This activity could be a glitch that can impact timing. The signal that is impacted is called the victim and signal that is the cause is called the aggressor. The crosstalk can affect the timing of the victim signal, if the aggressor switches at the same time as the victim. Depending upon the

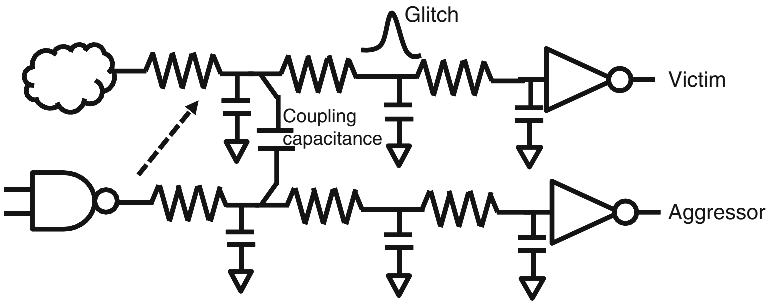

Fig. 7.8 Glitch due to crosstalk

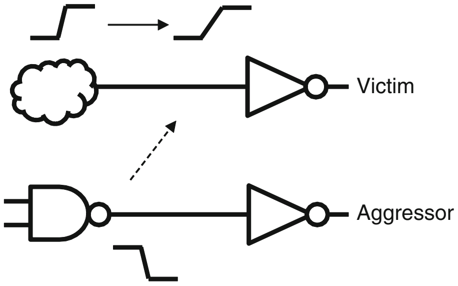

Fig. 7.9 Victim slew deterioration on account of crosstalk

direction of the switching for the aggressor and the victim, the transition at the victim could be slower (impacting setup relationship) or faster (impacting hold relationship). This is referred as the timing window relationship between the aggressor and victim and indicates the period of overlapping time when switching of aggressor and victim can potentially coincide.

Since crosstalk affects timing, it has a direct impact on setup and hold timing check. Let us consider Fig. 7.9, which is the schematic representation of Fig. 7.8 without the resistance and capacitance. If the aggressor net has a switching in the direction opposite to that of the victim, the slew on the victim net can deteriorate, thereby increasing its delay. This will impact the setup timing. Similarly, a switching on the aggressor net in the same direction as the victim can improve the slew of the adjacent victim net reducing its delay. This will impact hold timing.

From a signal integrity perspective, if mutually exclusive clocks have no crosstalk issue, then they are considered to be physically exclusive.

Most STA tools provide a way to measure the integrity of a signal in a design framework. There are books just on signal integrity and crosstalk analysis, and we will not be covering this in detail here. The concept is being introduced since certain SDC commands provide directives for crosstalk analysis.

7.4 set_clock_group

Based on what we have looked so far, correct setup and hold requirements ensure timing for reliable data capture. However for asynchronous clocks it could be tedious and impossible to meet the requirement given that the phase relationship of the clocks is not deterministic. For mutually exclusive clocks it makes no sense to try to meet the requirement, since the clocks don't talk to each other. In order to indicate to timing tools to ignore any timing paths or crosstalk analysis between asynchronous or mutually exclusive clocks, SDC provides the *set_clock_groups* command. The BNF grammar for the command is:

```
set_clock_groups [-name group_name]
                 [-group clock_list]
                 [-logically_exclusive]
                 [-physically_exclusive]
                 [-asynchronous]
                 [-allow_paths]
                 [-comments comment_string]
```

The *-name* option is used to provide a unique name for clock group. The clocks are divided into groups which are specified using *-group* option.

The *-logically_exclusive* option is used when clocks are mutually exclusive but can have a coupling interaction between them. The grouping between clocks in Fig. 7.6 can be represented as:

```
create_clock -period 10 -name C1 -waveform {0 5} [get_ports C1]
create_clock -period 20 -name C2 -waveform {0 12} [get_ports C2]
set_clock_groups -logically_exclusive -group C1 -group C2
```

Though the aforementioned *set_clock_groups* is technically correct, the authors recommend to create a combinational generated clock from *C1* and *C2* and then set up the clock group relation between them. This helps reuse in case the design is modified at a later stage such that clocks *C1* and *C2* start interacting in another part of the design (among *F3* and *F4*) as shown in Fig. 7.7. This would be modified as:

```
create_clock -period 10 -name C1 -waveform {0 5} [get_ports C1]
create_clock -period 20 -name C2 -waveform {0 12} [get_ports C2]
create_generated_clock -name GC1 \
-source [get_ports C1] [get_pins mux1/A] -combinational
create_generated_clock -name GC2 \
-source [get_ports C2] [get_pins mux1/B]  -combinational
set_clock_groups -logically_exclusive -group GC1 -group GC2
```

The *-physically_exclusive* option is used when the clocks don't coexist in the design. The grouping between the clocks in Fig. 7.7 can be represented as:

```
create_clock -period 10 -name C1 -waveform {0 5} [get_ports C1]
create_clock -period 20 -name C2 -waveform {0 12} [get_ports C2]
create_generated_clock -name GC1 -divide_by 1 \
```

-source [get_pins mux1/A] [get_pins mux1/Z] -combinational
*create_generated_clock -name GC2 -divide_by 1 *
-source [get_pins mux1/B] [get_pins mux1/Z] -combinational -add
set_clock_groups -physically_exclusive -group GC1 -group GC2

As it can be seen, the timing between flops *F1* and *F2* doesn't have to be considered for the combination of *F1* being driven by *C1* and *F2* by *C2* and vice versa, but clocks *C1* and *C2* also drive flops *F3* and *F4,* and so, we cannot simply apply

set_clock_groups -logically_exclusive -group C1 -group C2

This command will disable timing paths between *F3* and *F4* for the clocks *C1* and *C2*. By defining a combinational generated clock at the output of the mux, the timing tool is given the directive to disable localized timing path analysis between flops *F1* and *F2* for the relevant clocks, without impacting flops *F3* and *F4*.

If you define multiple clocks on the same design object (using *-add* option), they should be physically exclusive. Another scenario when clocks are physically exclusive is when both system clock and test clock are applied on the same port.

The *-asynchronous* option is used when the clocks don't share a phase relationship with each other. It should be understood that asynchronous crossings also need synchronizers, purely for functional reliability. Synchronizers are not being dealt in this book, since the scope of the book is limited to timing aspects.

The options *-logically_exclusive*, *-physically_exclusive*, and *-asynchronous* are mutually exclusive. You can use only one option in a single *set_clock_groups* command. However you can specify relationships between clocks in multiple commands which could be different.

Each of these three options indicates that timing paths between clock groups must not be considered. However for crosstalk analysis, they have a different meaning. If the clock group is *logically_exclusive*, then crosstalk analysis between clocks is computed like any two synchronous clocks. If the clock group is *physically_exclusive*, then no crosstalk analysis is done between the clocks. If the clock group is *asynchronous*, the clocks are assumed to have an infinite timing window where the aggressor and victim can switch together.

When clock groups are defined asynchronous and the users want to maintain the crosstalk analysis but don't want to disable timing paths between clock, then that is achieved using *-allow_paths* option. This option can only be used with *-asynchronous* option. This is generally used only in the context of signal integrity checks and not used in STA.

You can have more than one group in a single *set_clock_groups* command. The list of clocks in a group is meant to be logically exclusive or physically exclusive or asynchronous to all the clocks in other groups. If only one group is specified, then it indicates all clocks in that group are logically exclusive or physically exclusive or asynchronous to the rest of the clocks in the design. One of the most important things to note is this command only specifies relationship between clocks in different groups. No relationship is implied for the clocks in the same group. Let us consider the command below:

set_clock_groups -asynchronous -group [get_clocks {clk1 clk2 clk3}]
-group [get_clocks {clk4 clk5 clk6}]

This command implies:

1. *clk1* is asynchronous to *clk4, clk5*, and *clk6.*
2. *clk2* is asynchronous to *clk4, clk5,* and *clk6.*
3. *clk3* is asynchronous to *clk4, clk5,* and *clk6.*
4. No relation can be assumed among *clk1, clk2,* and *clk3.*
5. No relation can be assumed among *clk4, clk5,* and *clk6.*

7.5 Clock Group Gotchas

While specifying the clock group the designer must be careful about the following things:

1. If you define clocks within a group, it doesn't mean they are synchronous. The relationship among clocks within a group could be defined elsewhere (say in another *set_clock_group* command or by the tool default).
2. Defining the clock group with incorrect option (*-physically_exclusive, logically_exclusive, -asynchronous*) may not impact timing since all effected timing paths are ignored, but it will impact your signal integrity analysis.
3. Just because you have defined a clock group relationship between a master clock and other clocks in the design, it doesn't mean that relationship is inherited by the generated clocks which have been derived from the master clock. All relationships should be explicitly specified.
4. The best way to remember clock grouping is
   (a) If two or more clocks coexist in the design, but there is no phase relationship, then they are specified as *-asynchronous* in *set_clock_group*.
   (b) If two or more clocks coexist in the design, but there is a circuit to select only one among these, then they are specified as *-logically_exclusive* in *set_clock_group*.
   (c) If two or more clocks cannot coexist in the design, then they are specified as *-physically_exclusive* in *set_clock_group*.

7.6 Conclusion

As much as we would like all clocks in a design to be in a single domain, the reality is multiple clock domains are inevitable. We looked at how we can ignore timing paths between domains that don't necessarily interact or which need not be timed, even if they interact. In the next chapter we will look at other clock characteristics that have to be considered for clocks.