Chapter 6
Generated Clocks

Most complex designs require more than one clock for its functioning. When there are multiple clocks in a design, they would need to interact or share a relationship. Asynchronous clocks are clock signals that don't share a fixed phase relationship. Having only asynchronous clocks in the design makes it really hard to meet setup and hold requirements when multiple clock domains are interacting. We will explain about this in Chap. 7 as to why it is so. Synchronous clocks share a fixed phase relationship. More often than not synchronous clocks originate from the same source.

Today's SoCs (System on a chip) contain heterogeneous devices within the same chip. This could include very high-speed processors as well as low-speed memories all on the same chip. These elements working at different speeds are usually triggered by different clocks. Each portion operating on its own clock could bring in asynchronicity in the design. This may result in several clocks being derived from one master clock. Such clocks are referred to as *generated clocks* or *derived clocks*. These clocks can be generated in multiple ways:

1. Clock dividers
2. Clock multipliers
3. Clock gating

6.1 Clock Divider

A clock divider generates a clock of higher period and lower frequency compared to the original source clock. A typical example of a clock divider is a 2-bit ripple counter. Figure 6.1 shows the circuit of a ripple counter. For this circuit, if the period of the clock at the input of the first flop is *10ns*, then waveform generated at the LSB (least significant bit) is divided by 2, which means it has a period of *20ns*. For the same design, the waveform at the MSB (most significant bit) is divided by 4, which means it has a period of *40ns*.

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_6, © Springer Science+Business Media New York 2013

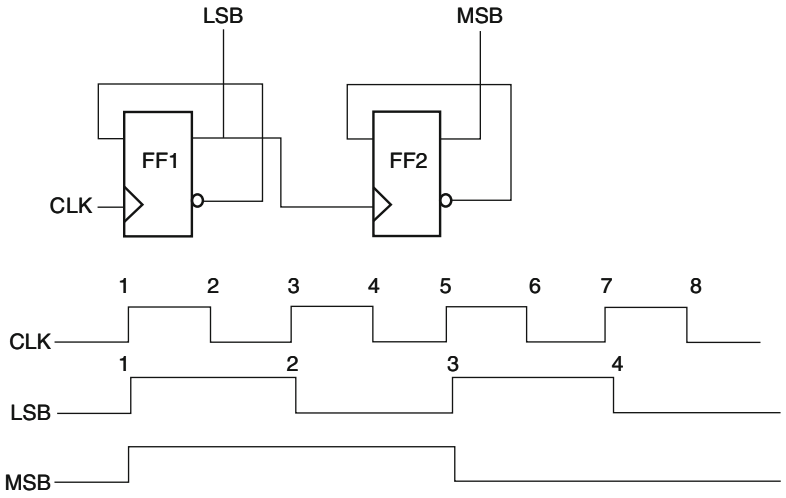


Fig. 6.1 2-bit ripple counter

6.2 Clock Multiplier

A clock multiplier is a circuit where frequency is increased and clock period is decreased for faster clocking rate. This technique is typically used in microprocessors and on internal busses so as to improve the overall throughput of the processor and is generally used in conjunction with internal cache memories. Figure 6.2 shows the circuit of a simple clock multiplier, where the clock frequency is doubled. The circuit is simple implementation of clock and its delayed version. The delay can be introduced in the line by use of buffers and invertors.

It is more common to use *PLLs* (phase-locked loops) to achieve frequency multiplication. This usage of PLLs has been mentioned in Chap. 17.

6.3 Clock Gating

Clock-gating technique has become very popular since mid-1990s to reduce power consumption. Power in a circuit is consumed when a flop or register in the design switches state due to a clock trigger. However in a design portions of the logic may not be getting used at certain times. During that stage, disabling clock to those portions of the design reduces the switching power. This is achieved by having enable logic before the clock and such a clock is called gated clock. Figure 6.3 shows the example of a gated clock.

We can also use clock gating to obtain divided clocks with waveforms similar to those shown in Fig. 6.1. The concept of clock gating can be extended to create clock pulses. Let us consider Fig. 6.4 where the clock is gated via a chain of odd number

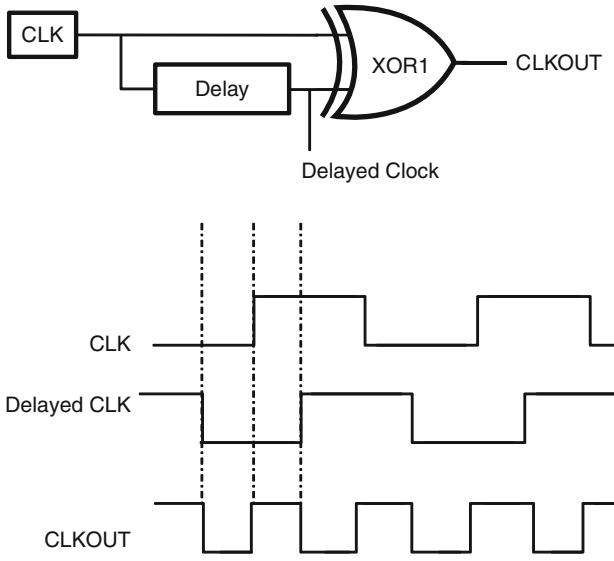
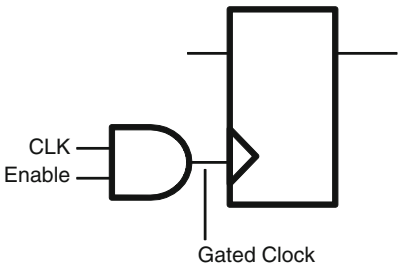


Fig. 6.2 A simple clock multiplier

Fig. 6.3 A gated clock



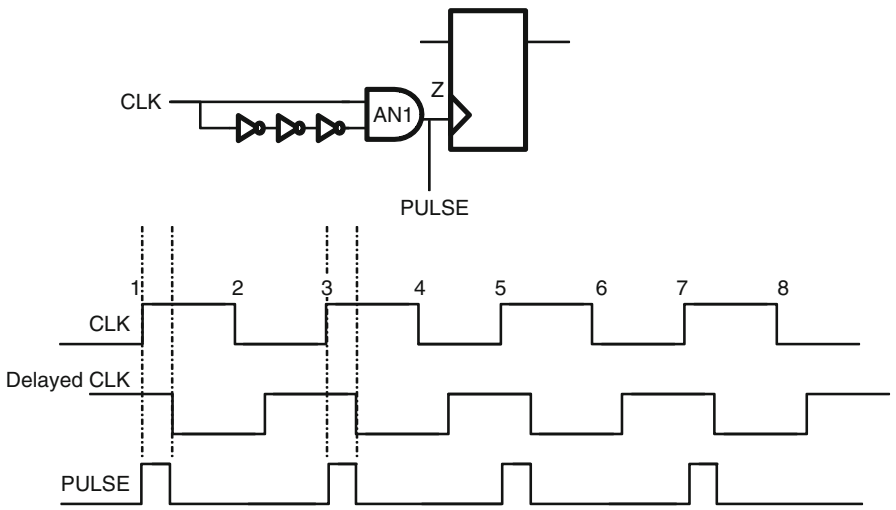


Fig. 6.4 A gated clock to generate pulse

of inverters. Depending on the delay on the chain of inverters, a pulse will be generated. This method is used to improve performance and reduce power as well. It is important to ensure that clock gating is typically done with only one clock.

6.4 create_generated_clock

The SDC command for specifying derived clocks in a design is *create_generated_clock*. The BNF grammar for the command is

create_generated_clock [source_objects]
-*source* clock_source_pin
[-*master_clock* master_clock_name]
[-*name* generated_clock_name]
[-*edges* edge_list]
[-*divide_by* factor]
[-*multiply_by* factor]
[-*invert*]
[-*edge_shift* shift_list]
[-*duty_cycle* percent]
[-*combinational*]
[-*add*]
[-*comment* comment_string]

6.4.1 Defining the Generated Clock Object

create_generated_clock is generally specified on design objects where the clock is actually available after division or multiplication or any other form of generation. These design objects called source objects can be port, pin, or net. When defining a clock on a net, ensure that net has a driver pin or the port. Otherwise the clock will not have a source. These are the points from where generated clocks can propagate into the circuit.

6.4.2 Defining the Source of Generated Clock

The source pin of a generated clock is specified using the -*source* option. This indicates the master clock source pin from which the generated clock is derived. For

example, in Fig. 6.1, the generated clock is defined for LSB and MSB, and the source of the generated clock is defined at CLK.

It is better to understand the difference between a source object and the source of the generated clock. Source object refers to the point where the generated clock (or clock) is being specified, while source of the generated clock refers to the point which acts as a reference from which the generated clock has been obtained.

As indicated in Chap. 5, a source object can have more than one clock. If the master clock source pin has more than one clock in its fanin, then the generated clock must indicate the master clock which causes the generated clock to be derived. This is specified using the *-master_clock* option. This option takes the name of the SDC clock that has been defined to drive the master clock source pin. Once a generated clock has been defined, the clock characteristics (waveform, period, etc.) would be derived by the tool, based on the characteristics of the waveform at the source.

For a clock to be generated from a specific source, it is important that the source has to somehow influence the generated clock. One of the commonly committed mistakes while specifying generated clock is to specify a source which doesn't fanout to the generated clock. Effectively, this means the waveform of the generated clock has been specified as a function of the waveform at a source pin that does not even influence the generated clock! Many implementation tools do not catch this and it results in incorrect clock waveforms being used for the generated clock during STA.

6.4.3 Naming the Clock

Like the primary clock, a generated clock is also identified by its name. This is specified as string using the *-name* option. When *-name* is not specified, tools might assign a name on their own. To establish dependency on the generated clock, any subsequent SDC command simply refers to the generated clock name.

6.4.4 Specifying the Generated Clock Characteristic

The characteristic of a generated clock can be specified using one of the three options:

1. *-edges* – this is represented as a list of integers that correspond to the edge of the source clock from which the generated clock has been obtained. The edges indicate alternating rising and falling edge of the generated clock. The edges must contain an odd number of integers and should at the very minimum contain 3 integers to represent one full cycle of the generated clock. The count of edge

starts with "1" and this number ("1") represents the first rising edge of the source clock.
2. *-divide_by* – this represents a generated clock where the frequency has been divided by a factor, which means the period is multiplied by the same factor.
3. *-multiply_by* – this represents a generated clock where the frequency has been multiplied by a factor, which means the period is divided by the same factor. It should be noted that though clocks are defined using period characteristic, the multiply_by and divide_by are specified using frequency characteristic in mind (which is inverse of period).

In general any generated clock represented using *-divide_by* or *-multiply_by* options can also be represented using *-edges* option. However the vice versa is not true. Let us consider Fig. 6.1; assuming the *create_clock* is defined for *CLK*, the generated clock can be defined at *LSB* and *MSB*.

create_clock -period 10 -name CLK [get_ports CLK]
create_generated_clock -name LSB -source [get_port CLK]
-divide_by 2 [get_pins FF1/Q]
create_generated_clock -name MSB -source [get_pins FF1/Q]
-divide_by 2 [get_pins FF2/Q]

The generated clocks at *LSB* and *MSB* can also be represented using the *edges* option as:

create_generated_clock -name LSB -source [get_ports CLK]
-edges {1 3 5}[get_pins FF1/Q]
create_generated_clock -name MSB -source [get_pins FF1/Q]
-edges {1 3 5}[get_pins FF2/Q]

In the *LSB* case, the edges *{1 3 5}* indicate the edge number of the specified source clock *CLK* to which the fall and rise edges of the generated clock are aligned. For the *MSB*, since the edges are aligned to *LSB* (which is the source); hence, the edge specification is the same.

The same waveform for *MSB* can be generated using *CLK* as the source. In this case, the edge would be *{1 5 9}*.

create_generated_clock -name MSB -source [get_ports CLK]
-edges {1 5 9} [get_pins FF2/Q]

The edge specification of *MSB* depends on the edge of the source, which in this case is the primary clock *CLK*.

When a generated clock defined using *-divide_by* or *-multiply_by* options need to be inverted, then it can be specified using the *-invert* option. Let us consider Figs. 6.5 and 6.6 which have different flavors of the divide-by-two circuits. Now depending on how the generated clock is defined (inverting or non-inverting), the characteristic of the generated clock can change.

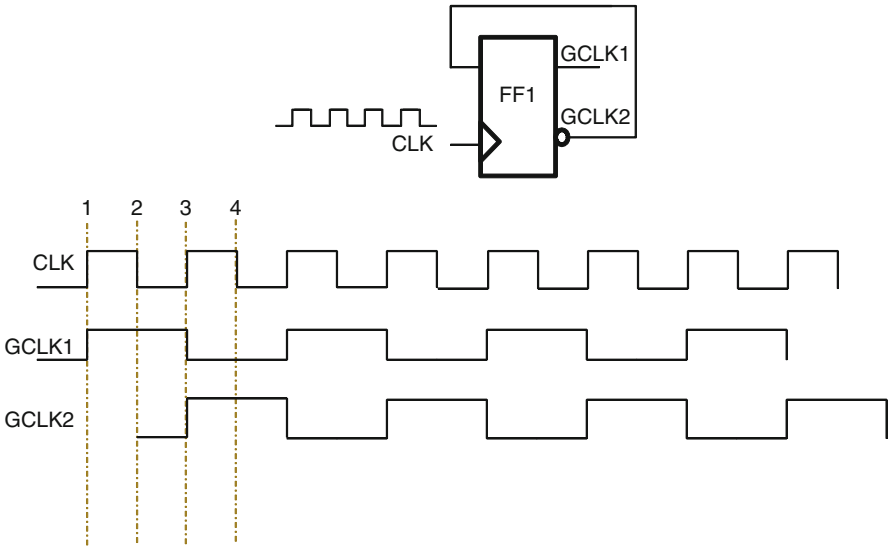


Fig. 6.5 Divide-by-two circuit with a non-inverting clock

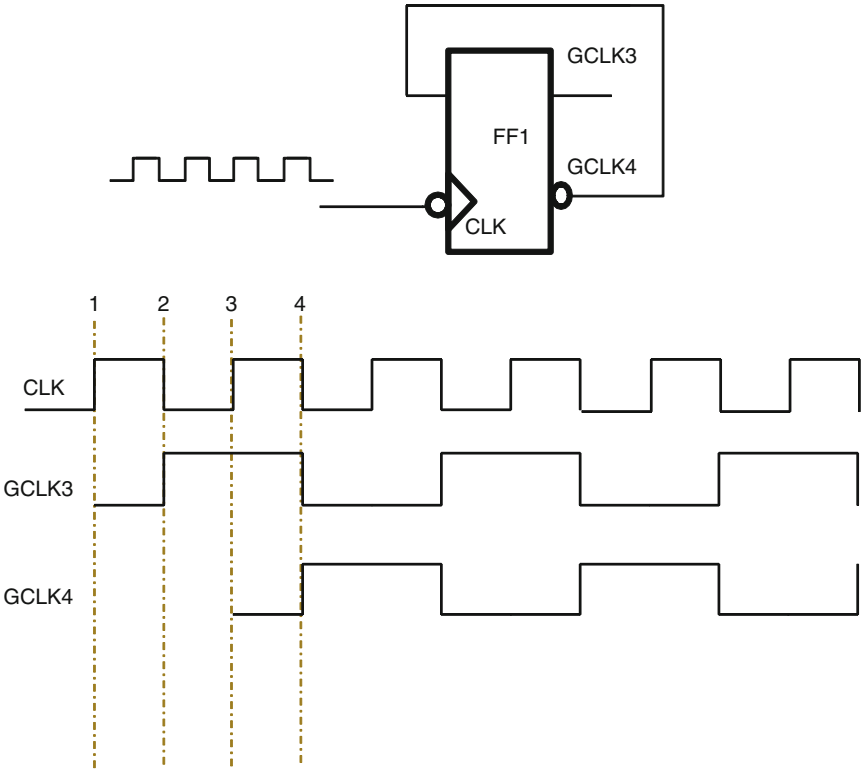


Fig. 6.6 Divide-by-two circuit with an inverting clock

In Fig. 6.5, the divider is triggered by positive edge of the source clock, and the generated clock *GCLK1* is defined as:

create_generated_clock -name GCLK1 -source [get_ports CLK]
-divide_by 2 [get_pins FF1/Q]

The clock *GCLK2* is an inverted version of *GCLK1*; this is therefore defined as:

create_generated_clock -name GCLK2 -source [get_ports CLK]
-divide_by 2 -invert [get_pins FF1/QBAR]

It should be noted that the presence of *-invert* does not change the edge of the source clock at which generated clock will have a transition. It only impacts whether the generated clock will start with a rising transition or a falling transition.

However in Fig. 6.6, the divider is triggered by negative edge of the source clock; in this case, the generated clock *GCLK3* is defined as:

create_generated_clock -name GCLK3 -source [get_ports CLK]
-edges { 2 4 6} [get_pins FF1/Q]

The clock *GCLK4* is an inverted version of *GCLK3*; this is therefore defined as:

create_generated_clock -name GCLK4 -source [get_ports CLK]
-edges { 4 6 8} [get_pins FF1/QBAR]

As it can be seen that *GCLK3* and *GCLK4* can be represented using the *-edges* option. Specifying it any other way will result in inconsistency between the actual circuit and the waveform as represented by the SDC command. This is the most commonly made mistake in defining generated clocks.

Similarly, the *CLKOUT* in Fig. 6.2 can be represented as:

create_generated_clock -name CLKOUT -source [get_ports CLK]
-multiply_by 2 [get_pins XOR1/Z]

When defining a clock where frequency is multiplied, the duty cycle can be specified using the *-duty_cycle* option. This option has meaning only with *multiply_by* option and represents the percentage of the pulse width when the multiplied clock is *1*. For example, *CLKOUT* in Fig. 6.2 can also be represented as below, indicating a 50 % duty cycle.

create_generated_clock -name CLKOUT -source [get_ports CLK]
-multiply_by 2 [get_pins XOR1/Z] -duty_cycle 50

Let us consider the Fig. 6.4, where a high pulse has been generated and the pulse width depends on the delay in the chain of the invertors. In this case, edge *1* of the clock triggers both the rising and falling edge of the pulse. This is represented as:

create_generated_clock -name PULSE -source [get_ports CLK]
-edges { 1 1 3} [get_pins AN1/Z]

Depending on the kind of pulse, the edge specification may change. For example, in Fig. 6.4, if the *AND* gate is replaced by a *NAND* gate, then it will result in a rise-edge-triggered low pulse. This would be represented as:

create_generated_clock -name PULSE_N -source [get_ports clk]
-edges { 1 3 3} [get_pins NAND1/Z]

This is because the first edge of the clock will result in a falling edge and then a rising edge of the low pulse and since *-edges* represent the order in terms of rising and falling, so it is represented as *{1 3 3}*. This implies the rising edge of the generated clock will happen due to edge *1* of the source clock. The next falling edge of the generated clock will happen due to edge *3* of the source clock followed by the next rising edge which also is on the edge *3* of the source clock. This falling edge is actually in the next pulse of the generated clock.

6.4.5 Shifting the Edges

The edges of a generated clock may need to be moved by time units to indicate shift. For example, in Fig. 6.4, if the delay through the chain of inverters is *2ns*, then the high pulse can be accurately represented as:

create_generated_clock -name PULSE -source [get_ports clk]
-edges { 1 1 3} -edge_shift {0 2 0} [get_pins AN1/Z]

The *-edge_shift* option takes a list of floating point numbers, which represents the shift in each edge in terms of time units. This option must have the same number of arguments as the number of edges to represent the shift of each edge of the generated clock. The above command now implies, on the generated clock:

Rising edge happens at the first edge of the source clock.
Falling edge happens at *2ns* after the first edge of the source clock.
Next rising edge happens at third edge of the source clock.

Similarly, for a low pulse, the representation would be

create_generated_clock -name PULSE_N -source [get_ports clk]
-edges { 1 3 3}-edge_shift {2 0 2} [get_pins NAND1/Z]

This command implies, on the generated clock

Rising edge happens at *2ns* after the first edge of the source clock.
Falling edge happens on third edge of the source clock.
Next rising edge happens at *2ns* after the third edge of the source clock.

The shift can be a positive or negative number. Use of *-edges* and *-edge_shift* can be used to model arbitrarily complex generated clocks.

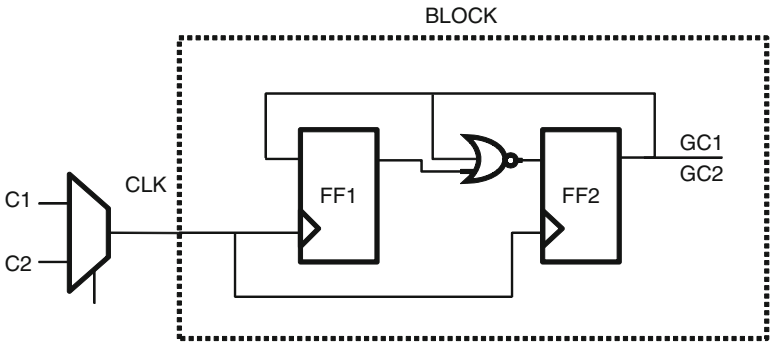


Fig. 6.7 Block driven by off-chip multiplexer with two clocks

6.4.6 More than One Clock on the Same Source

As described in Chap. 5, there can be more than one clock defined at a point. Or, for a given source, multiple clocks could be reaching the source. Typically one generated clock is defined per clock reaching the specified source. If there is more than one clock converging on the source specified for the generated clock, then the generated clock derived from this clock source pin could have characteristics corresponding to either of the clocks reaching the source. Thus, we would need to specify which of the clocks should be used to determine the characteristics of the generated clock. Let us consider the block in Fig. 6.7.

Assume that the *CLK* port is driven outside the block by a multiplexer on which two clocks with two different characteristics converge. This *CLK* port can act as a source for a clock divider circuit inside the block. In order to model the clock constraints for such a block and the generated clock for divider circuit, the designer would have to specify multiple generated clocks on the same object. This is represented as:

```
create_clock -name C1 -period 10 [get_ports CLK]
create_clock -name C2-period 15 [get_ports CLK] -add

# The following generated clock is based on C1's characteristics
create_generated_clock -name GC1 -divide_by 3 -source [get_port CLK]
-master_clock C1 [get_pins FF2/Q]

# The following generated clock is based on C2's characteristics
create_generated_clock -name GC2 -divide_by 3 -source [get_port CLK]
-master_clock C2 [get_pins FF2/Q] -add
```

Thus, even though the source for both the generated clocks is the same (viz., *CLK* port), the waveform for the two generated clocks are different. Alternately,

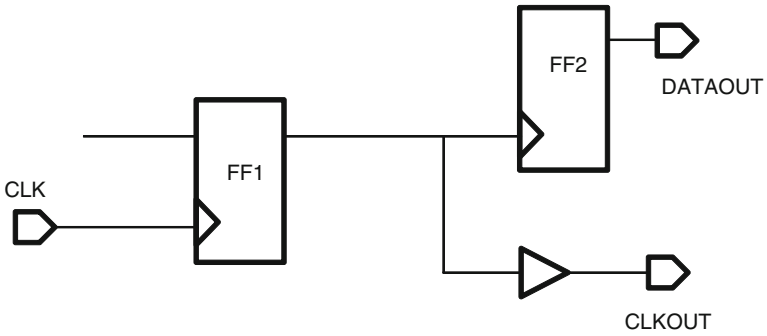


Fig. 6.8 Source-synchronous interface

there are two clock waveforms on the object *FF2/Q*. This makes sense logically. If the source of the divider has two different kinds of clocks on it, the divider output will also have two different kinds of clocks on it.

Under these conditions the user would need to specify an *-add* option, if he wants both the generated clocks to be considered for analysis by synthesis and STA. Since each clock is required to be identified by a unique name, it is mandatory to use *-name* option, when *-add* option is used. In this example for a generated clock, the specified source object had multiple clocks (either defined on it or reaching it). In such situations, it's not clear as to which of these clock characteristics should be used to create the waveform for the generated clock. The option *-master_clock* has been used to identify which of the clocks reaching the specified source object should be used for deriving the characteristics of the generated clock. When a user specifies multiple clocks (or generated clocks) on the same object but doesn't specify a *-add* option, the last constraint overrides the previous definitions.

It should be noted that *-source* uses design object on which clock is defined/ reaches, while master specifies the clock name.

6.4.7 Enabling Combinational Path

Let us consider Fig. 6.8 which represents a source-synchronous interface. In a source-synchronous interface, clock appears along with the data as an output. The advantage of this mechanism is that both clock and data are routed through similar traces and thus have very similar delays. At the receiver device, the incoming data is sampled with respect to the incoming clock. The actual trace delay is not of much importance as long as the delay differential on the two lines is close to 0. This mechanism provides an interface for high-speed data transfer.

In this figure the delay on the *DATAOUT* pin should be specified with respect to *CLKOUT*. In this case, a generated clock needs to be defined at CLKOUT. This is done using the *-combinational* option. When this option is specified, the generated

clock is considered to be of the same period as master clock pin, which is equivalent to a *divide_by 1*. It cannot be used with any other option. This can be represented as:

create_generated_clock -name CLKOUT -combinational
-source [get_pins FF1/Q [get_ports CLKOUT]

In some case, there may be more than one path from the source clock pin to the place where generated clock is defined. If these paths are sequential in nature, i.e., they pass through sequential elements like flip-flops or through a transparent latch, then generated clocks are generally considered safe the way they are traditionally defined. However in some cases, if there is path from the source pin to the generated clock, which is purely combinational, that coexists with the sequential path, then traditional definitions of *create_generated_clock* will fail. In such cases, it is important to block the sequential path because the combinational path is always active. That too is achieved by defining a generated clock with *-combinational* option.

In Chap. 11 on false paths, we will see how the various kinds of clocks can be used to disable certain clock paths from timing analysis, which help in improving the efficiency of STA tools.

6.5 Generated Clock Gotchas

Since clocks can be generated in multiple ways, it is a common source of mismatch between design functionality and timing specification. While specifying generated clock, the designer must be careful about the following things:

1. If you define a generated clock make sure it is actually generated by the specified source object. Conversely, if a flip-flip or register is driven by a clock which is in fanout of another clock, make sure there is *create_generated_clock* constraint set on it. A missing generated clock may result in unconstrained registers.
2. When multiple clocks converge on the source pin of a clock, make sure to specify the master clock with the generated clock definition.
3. If you are specifying more than one generated clock constraint on a pin because of multiple sources, make sure to use the *-add* option; otherwise, the last specified constraint would override.
4. Avoid clock convergence via multiple combinational paths as it can result in a pulse. If clocks converge via multiple paths (combinational and sequential), then it is important to disable the sequential path.

6.6 Conclusion

As with primary clocks, it is important to model generated clocks correctly. Failure to do so may result in increased timing closure iterations. If the characteristic of the generated clock as defined by the SDC constraint doesn't match the actual functionality

of the circuit, then these are extremely difficult to debug. In many cases, the design may meet the timing, but the hardware will exhibit a totally different behavior.

When generated clocks are defined, the clock characteristics are formed based on the clock characteristic at the source. It is usually possible to define the same characteristic directly through *create_clock* on the objects, rather than using generated clocks. From timing analysis perspective, as long as the characteristics are the same, it does not matter whether the clock was specified using *create_generated_clock* or using *create_clock*. However, whenever a clock is derived from another clock, it is always better to use *create_generated_clock*, rather than *create_clock*. It is easier to maintain and enhance, as modifying the source clock characteristic will directly impact the characteristic here. Also, using the correct constraint better mimics the design intent, which reduces the chances of errors as constraints are modified or enhanced – including migration across technologies and designs.

Further, when multiple clocks in a design interact, it is not enough to simply define the clocks correctly; it is also required to correctly define relationship between clocks. In the next chapter we will cover how you can effectively define such relationship between interacting clocks.