



XILINX

ALL PROGRAMMABLE™

Defining Clock Groups

Lauren Gao

Clock Interactions

➤ Synchronous

- Two clocks have a fixed phase relationship
 - They share common circuitry (common node)
 - They share the same primary clock (same initial phase)

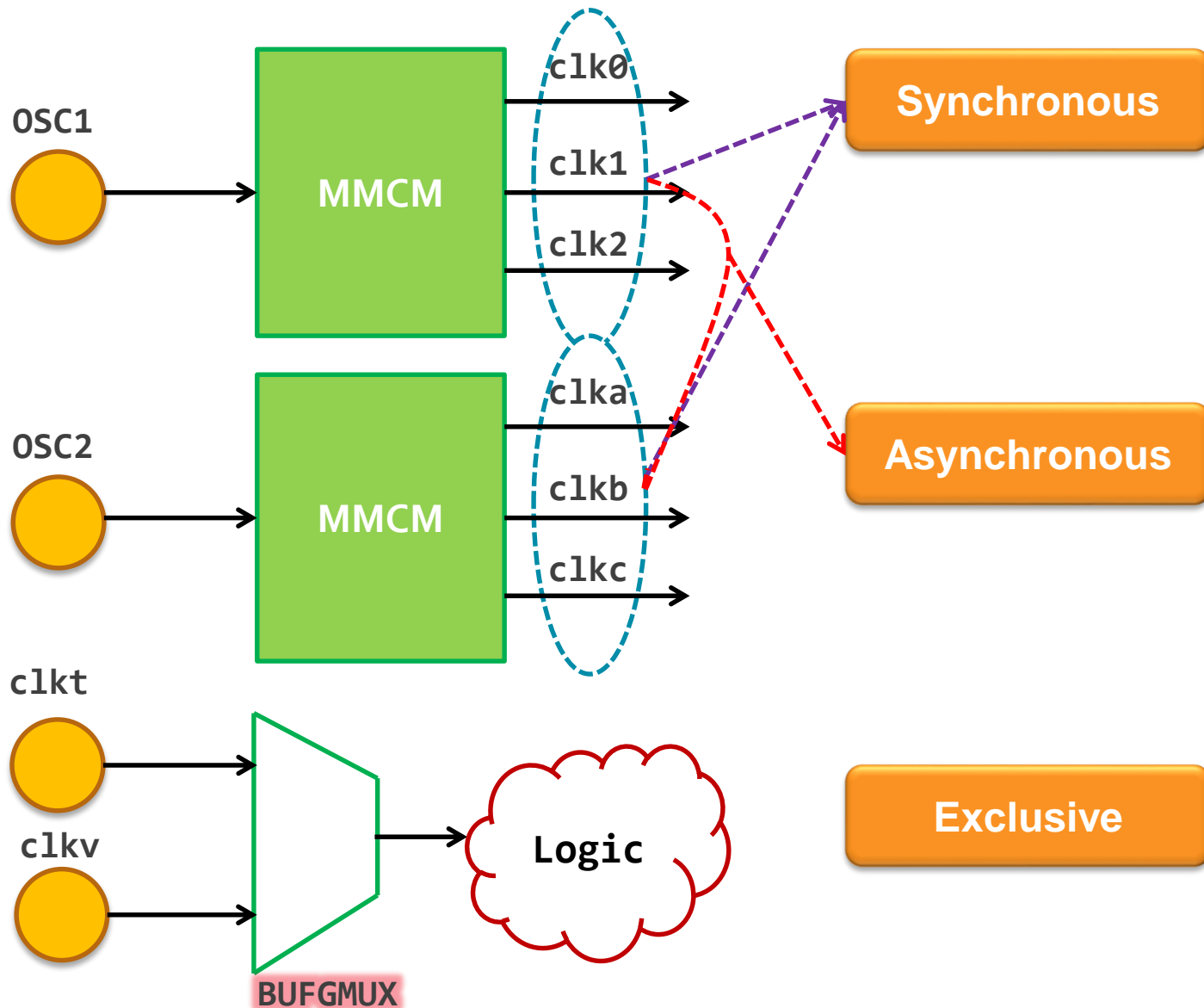
➤ Asynchronous

- Two clocks do not have a fixed phase relationship
 - They do not share any common circuitry in the design and do not have a common primary clock
 - They do not have a common period within 1000 cycles (unexpandable) and the timing engine cannot properly time them together

➤ Exclusive

- Two clocks propagate on a same clock tree and reach the same sequential cell clock pins but cannot physically be active at the same time
- Logically exclusive
 - Two clocks are defined on different source roots
- Physically exclusive
 - Two clocks are defined on the same source root by "create_clock -add"

Clock Interactions Examples

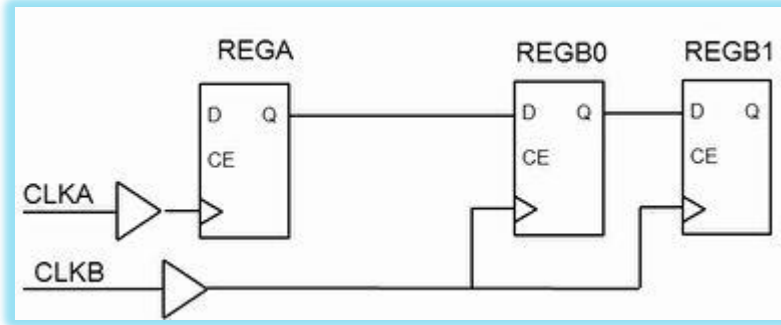


Clock Group Constraint Types

- `set_clock_groups [-name arg] [-logically_exclusive]`
`[-physically_exclusive] [-asynchronous] [-group args] [-quiet]`
`[-verbose]`
- Use `set_clock_groups` to create clock exception constraint
 - `set_clock_groups -asynchronous`
 - `set_clock_groups -logically_exclusive`
 - `set_clock_groups -physically_exclusive`

Asynchronous Clock Groups

- Use **set_clock_groups -asynchronous** to efficiently constrain such clocks



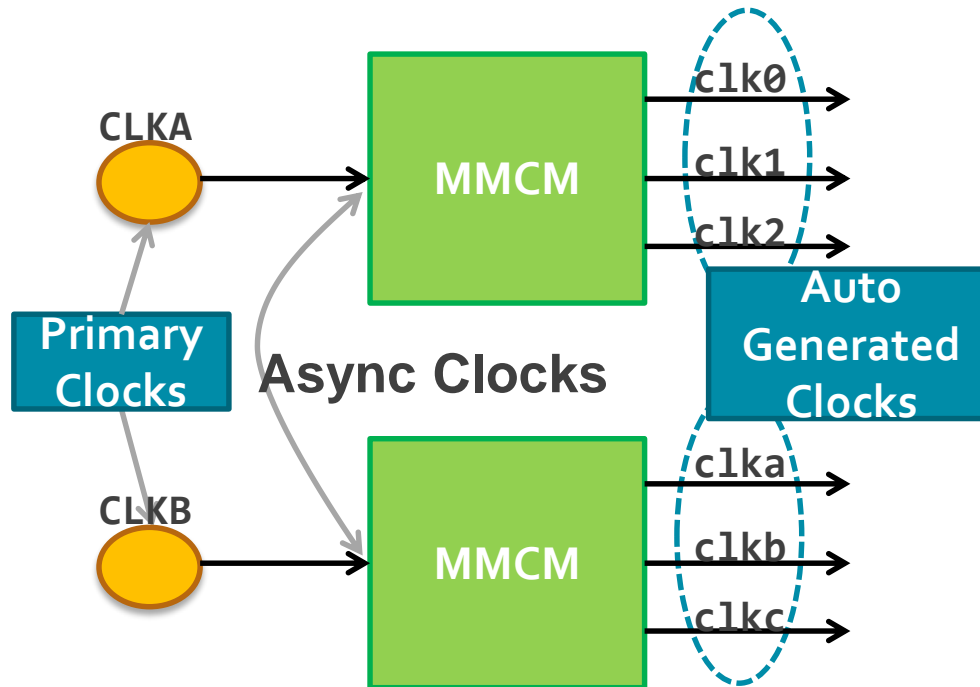
- *CLKA,CLKB can be from different ports*
- *Or different MMCM*

```
create_clock -name CLKA -period 10.0 [get_ports CLKA]
create_clock -name CLKB -period 5.0 [get_ports CLKB]
set_clock_groups -async -group CLKA -group CLKB
```



```
set_false_path -from [get_clocks CLKA] -to [get_clocks CLKB]
set_false_path -from [get_clocks CLKB] -to [get_clocks CLKA]
```

Asynchronous Clock Groups



BEWARE: This overrides any *set_max_delay* constraints!

```
create_clock -name CLKA -period 10.0 [get_ports CLKA]
create_clock -name CLKB -period 5.0 [get_ports CLKB]
set_clock_groups -async \
-group [get_clocks -include_generated_clocks CLKA] \
-group [get_clocks -include_generated_clocks CLKB]
```

Asynchronous Clock Groups

➤ Example:

- The clocks `clk50` and `clk100` are synchronous to each other
- The clocks `clk33` and `clk66` are synchronous to each other
- The clocks `clk50` and `clk100` are asynchronous to the clocks `clk33` and `clk66`

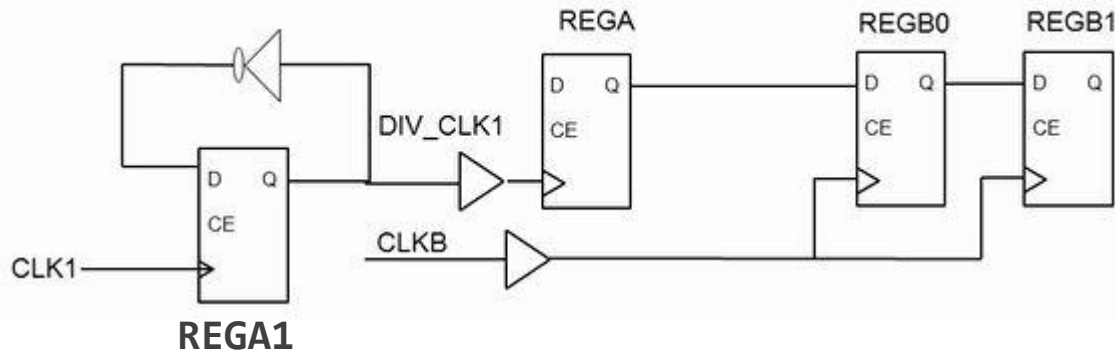
➤ The constraint for the clock groups would be:

```
set_clock_groups -async \  
-group {clk50 clk100} -group {clk33 clk66}
```

Asynchronous Clock Groups

➤ In the below figure

- The clocks **CLK1** and **DIV_CLK1** are synchronous to each other
- The clocks **CLK1** and **DIV_CLK1** are asynchronous to **CLKB**



➤ Solution

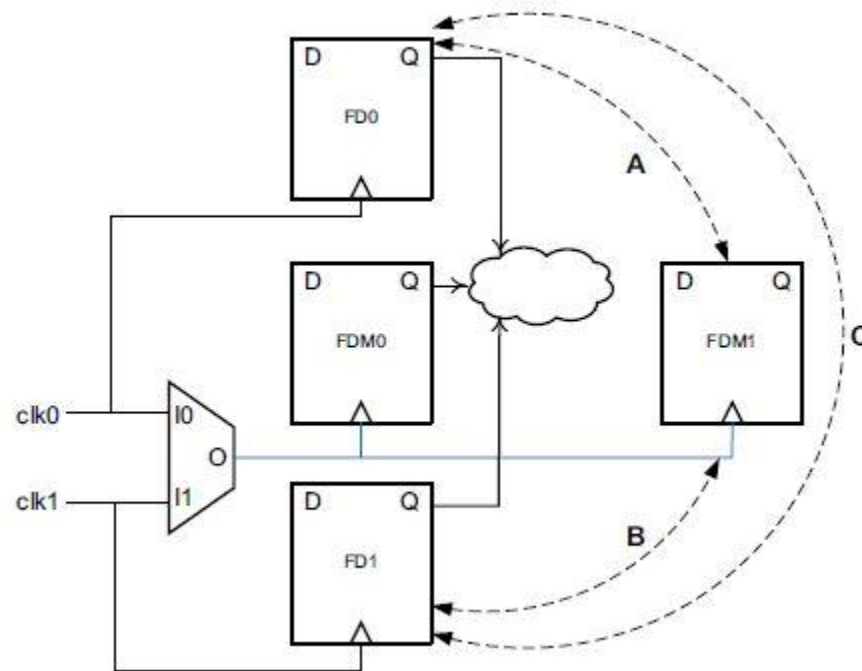
```
create_generated_clock -name DIV_CLK1 -source \  
[get_pins REGA1/C] -divide_by 2 [get_pins REGA1/Q]  
set_clock_groups -async -group {CLK1 DIV_CLK1} -group {CLKB}
```


Logically Exclusive Clock Groups

➤ Guideline

- Logically exclusive clocks shouldn't interact outside the MUX
- Logically exclusive clocks are defined on different source roots

Logically Exclusive Clock Groups

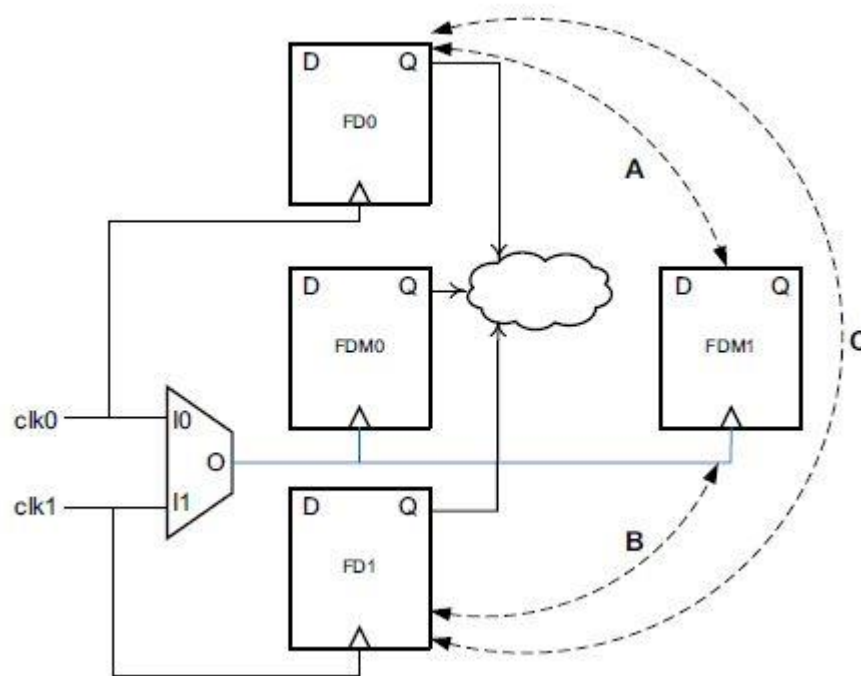


➤ Case in which the paths A, B, and C do not exist

```
set_clock_groups -logically_exclusive -group clk0 -group clk1
```

Logically Exclusive Clock Groups

- Case in which only the paths A or B or C exist



```
create_generated_clock -name clk0mux -divide_by 1 \  
-source [get_pins mux/I0] [get_pins mux/O]  
create_generated_clock -name clk1mux -divide_by 1 \  
-add -master_clock clk1 -source [get_pins mux/I1] [get_pins mux/O]  
set_clock_groups -physically_exclusive \  
-group clk0mux -group clk1mux
```

Physically Exclusive Clock Groups

➤ Guideline

- Physically clocks cannot physically exist at the same time
- Physically clocks are defined on the same source root

Physically Exclusive Clock Groups

- The FPGA you are implementing must go on two different versions of PCB boards. To ease the FPGA version management, the requirement is to have a single bitfile for both boards. The only difference between the boards is that the wbClk has a different frequency depending on the configuration:
- Configuration A:
 - wbClk is a 100MHz clock
 - bftClk is a 200MHz clock
 - All CDCs between these clock domains can be treated as asynchronous
- Configuration B:
 - wbClk is a 150MHz clock
 - bftClk is a 200MHz clock
 - All CDCs between these clock domains can be treated as asynchronous

Solutions

```
create_clock -name wbClk_A -period 10.0 [get_ports wbClk]
create_clock -name wbClk_B -period 6.667 [get_ports wbClk] -add
create_clock -name bftClk -period 5.0 [get_ports bftClk]
set_clock_groups -physically_exclusive -name two_clk_grps \
-group wbClk_A -group wbClk_B
set_clock_groups -async -name my_async_clks \
-group [get_clocks "wbClk_A wbClk_B"] -group bftClk
```

Things to Keep in Mind...

- Say you have 10 clocks in your design (clk1, clk2, ..., clk10)
 - Let us assume that clocks clk1, clk2 and clk3 are synchronous and you create the following constraint
 - `set_clock_groups -async -group {clk1 clk2 clk3}`
 - The clocks clk1, clk2 and clk3 are synchronous with each other but are asynchronous to all other clocks in the design
 - The other clocks (clocks clk4, clk5...) are analyzed as synchronous to each other
- The above constraint is the same as
 - `set_clock_groups -async -group {clk1 clk2 clk3} -group {clk4 clk5 ...clk10}`
- User generated clocks are not automatically grouped in the same group as the master!
 - In the above example if clk1_gen is a user generated clock of clk1 they you'd need to add the clk1_gen to the same group as clk1
 - `set_clock_groups -async -group {clk1 clk1_gen clk2 clk3}`