Tips for better constraint and design

KEK IPNS E-sys
Ryotaro Honda, Yun-Tsung Lai

Content

Electronics System Group

Placement constraint

- IOB constraint, ASYNC_REG constraint (exchanging signals with the outside of FPGA)
- Relative placement constraint (RLOC)

Timing constraint

- Clock definition and grouping
- multicycle_path constraint
- The case of switching clocks with MUX

Explanation on constraint≒There are many types of related UGs explaining Vivado.

- UG with direct relation: UG903, UG912。
- UG with relation to Vivado: UG894, UG895, UG896, UG899, UG901, UG904, UG905, UG906, UG908

~1500 pages in total.
Let's focus on common cases and explain them.

TIPS for better design

- Reset distribution
- Huge fanout and driver duplication
- Re-consider the CLB register insertion
- Stop using magic numbers
- Beware of large IPs
- In case of timing violation

Digression

Expertise in the constraint largely determines how elaborate FW can be made.

signal exchange with the outside of FPGA

Electronics System Group

About timing analysis

Signal exchange inside FPGA

- Vivado knows all the parameters (Jitter, etc. are given correctly). We should just leave it to Vivado usually.
- The circuits controlling the amount of delay in FPGA internal signals are not within intermediate level.

signal exchange with the outside of FPGA

Output

- We want all the sync signals to be output from the FPGA PAD at the same timing.
- All you have to do is placing a register on the IO block.
  - The path from the last register to PAD is equivalent to extending the wiring pattern on the board.
  - If the last register is not fixed in the IO block, the circuit will change the pattern length every time it is synthesized.

Input

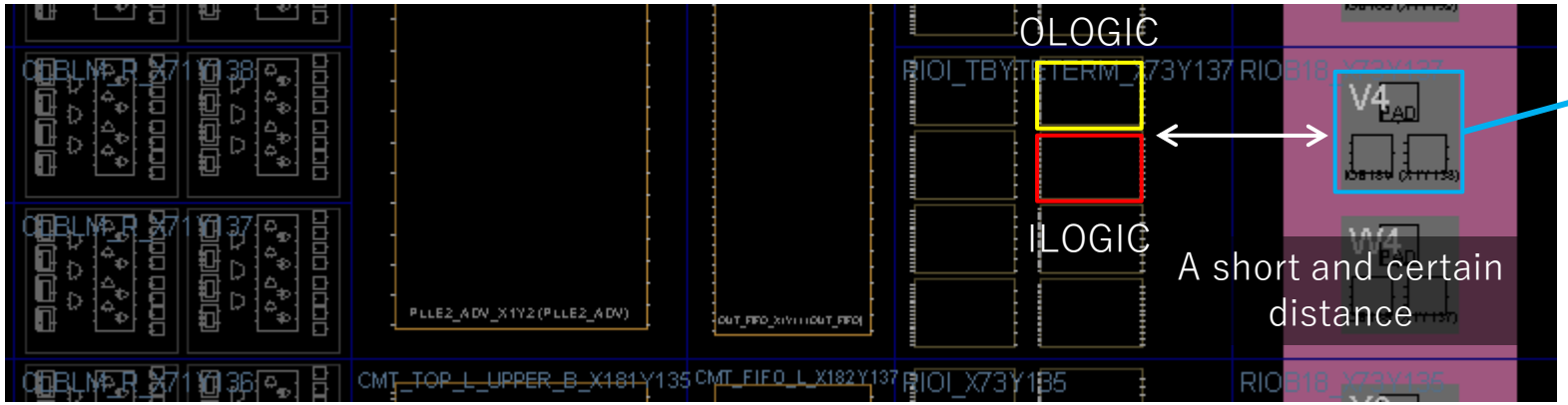
- Even if a register is placed in an IO block, I'm concerned about input timing variations between signals.
  - Constraints give the time difference between signals?
    Is the giving time difference really accurate?
  - We will omit this for now.
- Again, consider how to capture using the elements on the IO block.

⇒ If necessary, refer to UG903 for IO delay's constraint.

Consider the cases for low speed (up to about 50-100 MHz) and high speed.
In any case, we assume that we have to rely on the clock buffer for skew adjustment, and exclude the cases where timing can be adjusted in any way with a high-speed clock such as 1 MHz or less.

openit
Open source consortium of Instrumentation

Output to outside of FPGA



Low-speed (up to about 50-100 MHz) bus signals, etc.
For data

- Place last register on OLOGIC => use **IOB constraint**

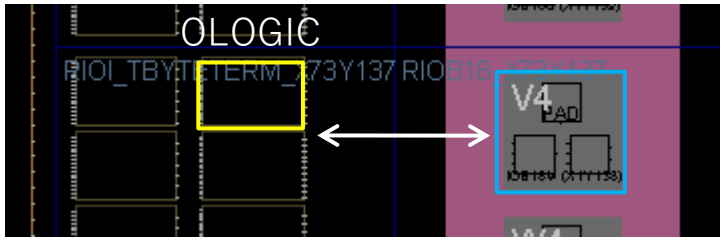
For clock

- Configure OLOGIC to ODDR primitive, and output

High speed or serial communication

- Use OSERDES with clock forward obediently.

Output to outside of FPGA



A register is placed on OLOGIC.
It looks like a simple FF on the HDL code and cannot be distinguished from CLB registers.

IOB Constraints with XDC

- In-code constraints can also be done in HDL, but not easy to re-use it, so I try to apply IOB in XDC.
- The device driving the top port has to be FF.。

```
set_property IOB TRUE [get_ports SIGIN]
set_property IOB TRUE [get_ports SIGOUT]
```

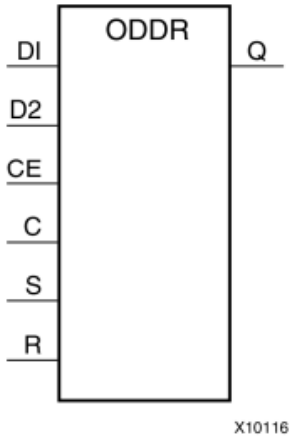
Port name at top-level

Output to outside of FPGA

Electronics System Group

ODDR

- A primitive to output data at double-data-rate.
- If D1 is HIGH and D2 is LOW, the same phase clock will be output.
  If it is reversed, the opposite phase clock will be output.



Port Descriptions

Port	Direction	Width	Function
Q	Output	1	Data Output (DDR): The ODDR output that connects to the IOB pad.
C	Input	1	Clock Input: The C pin represents the clock input pin.
CE	Input	1	Clock Enable Input: When asserted High, this port enables the clock input on port C.
D1 : D2	Input	1 (each)	Data Input: This pin is where the DDR data is presented into the ODDR module.
R	Input	1	Reset: Depends on how SRTYPE is set.
S	Input	1	Set: Active-High asynchronous set pin. This pin can also be Synchronous depending on the SRTYPE attribute.

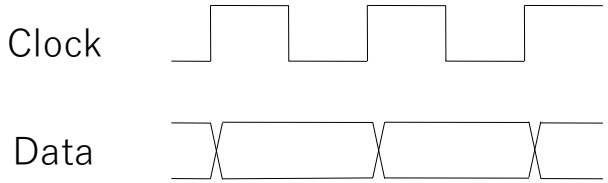
Why use the ODDR primitive to output the clock?

- OLOGIC does not (probably) have a dedicated clock output primitive.
- With ODDR, it is possible to output a clock that has been skew-adjusted with other data using IOB constraints.

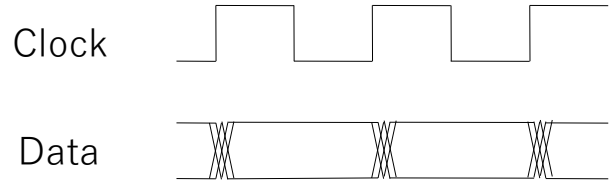
Input from outside of FPGA

The case with both data and source clock inputs
(Data acquisition is Done with an external clock)

When output data



When input data



In circuits where wire delays are not aligned, timing **may be** off.
(This does not happen if the circuit is properly equal in length.)

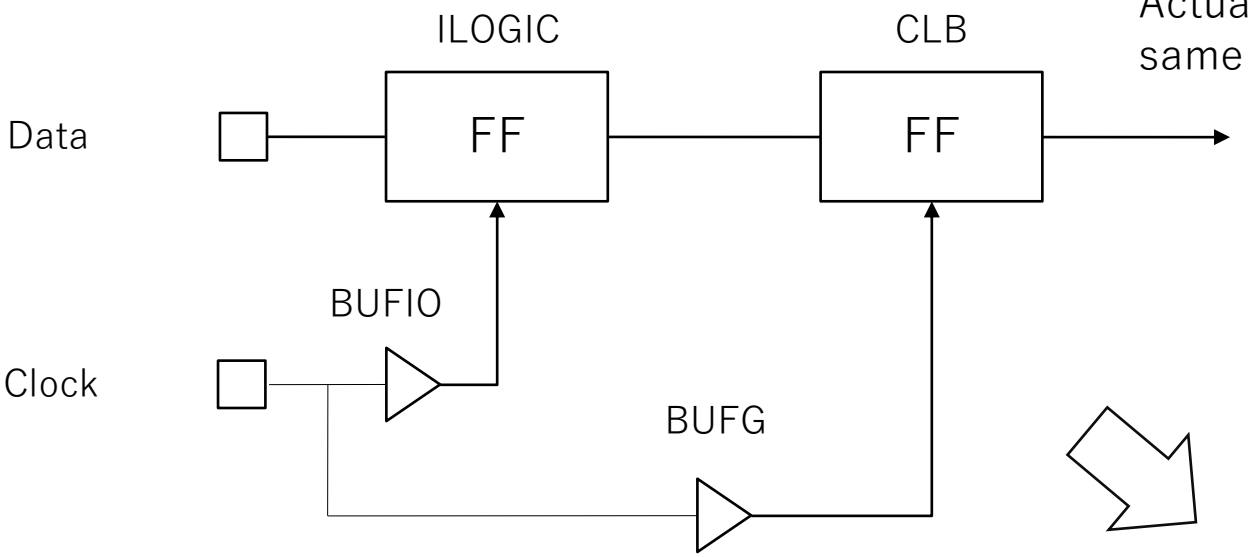
Even if the same phase is dangerous, there is enough margin if you capture the opposite phase up to about 50-100 MHz.

With higher speed, better to simply use ISERDES.

Input from outside of FPGA

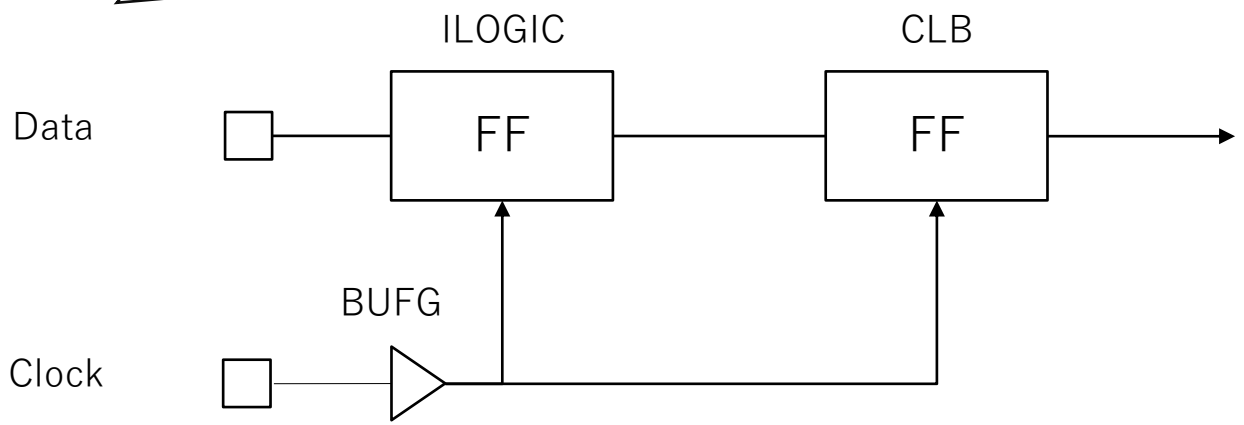
If the source sends an anti-phase clock.

Put the first FF in ILOGIC with IOB constraint



S redundant circuit just for explanation.
Actually, the clock outputs by BUFIO and BUFG are the same (adjusted to be in phase), so only BUFG is sufficient.

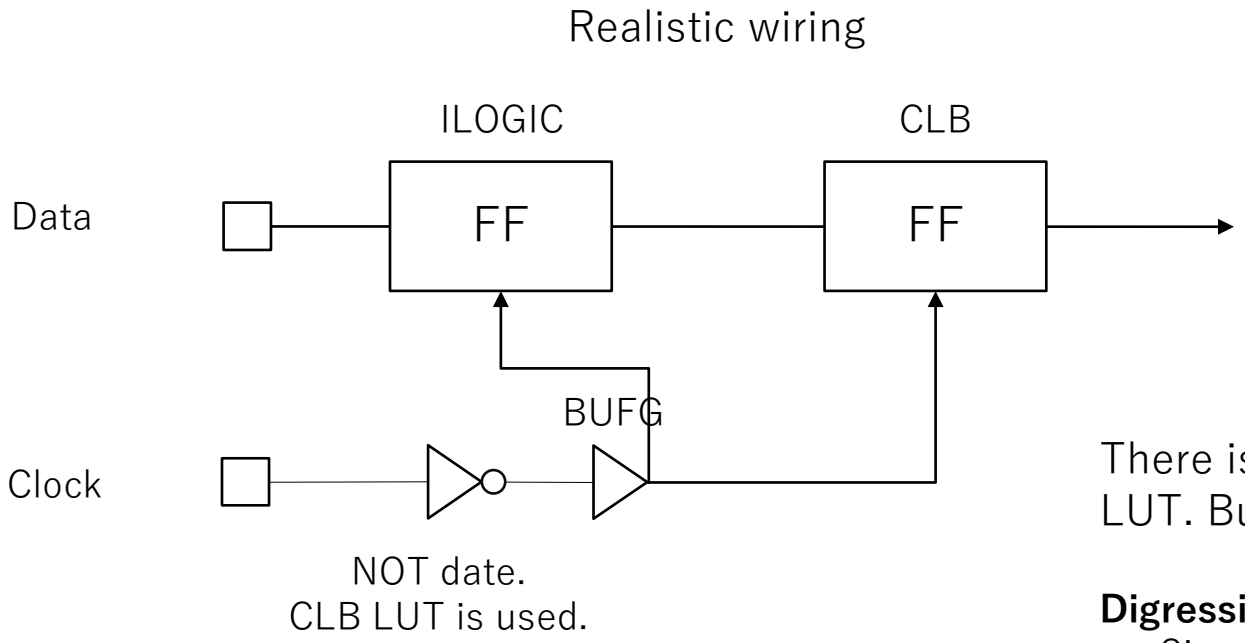
Realistic wiring



The clock input must be the CC pin.
(probably an error)

Input from outside of FPGA

If the source does not send an anti-phase clock.



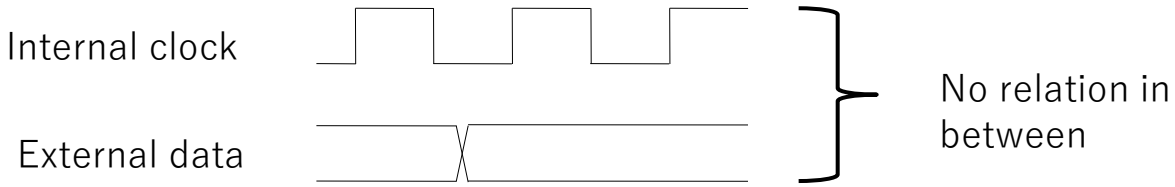
There is a delay of several hundred ps to exceed the LUT. But if it is up to about 100 MHz, it does not matter.

Digression

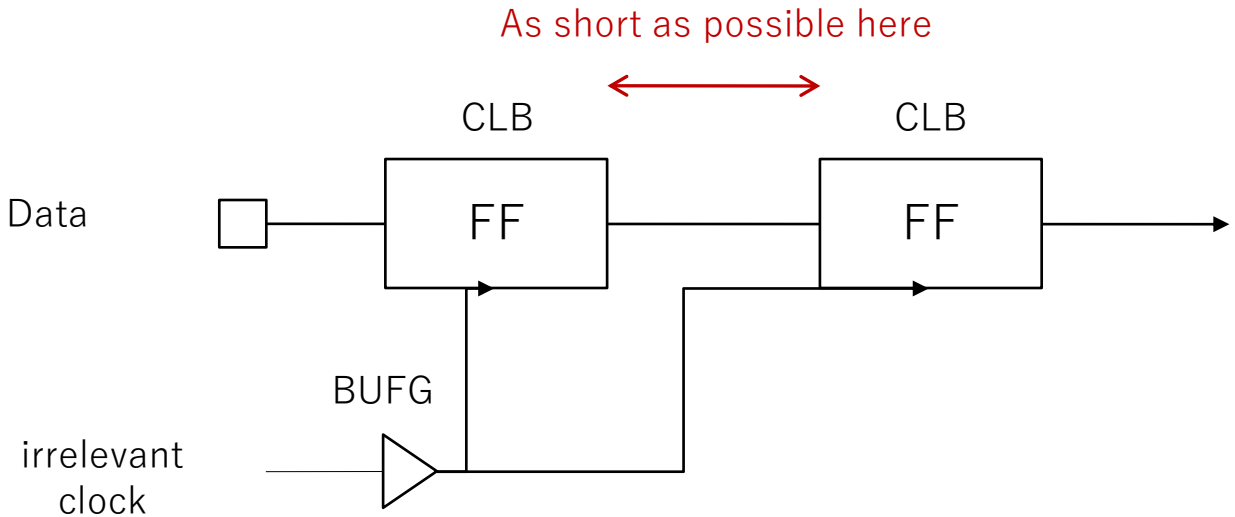
- Since fabric resources cannot intervene between PAD and BUFIO with dedicated wiring, it is impossible to create a redundant configuration with NOT.

Input from outside of FPGA

When no synchronous clock input from the source side.
(Capture using an unrelated system clock)
In this case, the data may be just timing inputs or very slowly transitioning data.



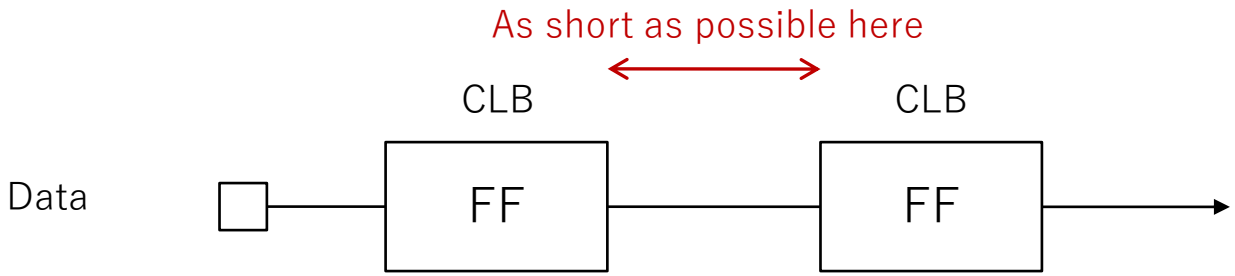
The first thing to consider is meta-stable countermeasures.
Use double or triple FF.



Input from outside of FPGA

Electronics System Group

ASYNC_REG constraint



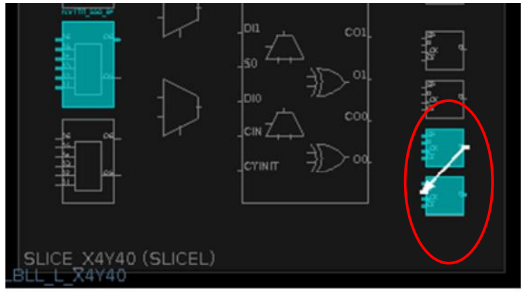


図 3-2: レジスタのグループ化　Make registers in group

With the ASYNC_REG constraint, it tries to place register elements in the same slice as much as possible.

Verilog-HDL

```
(* ASYNC_REG = "TRUE" *) reg sync_0, sync_1;
always @(posedge clk) begin
sync_1 <= sync_0;
sync_0 <= en;
...
```

VHDL

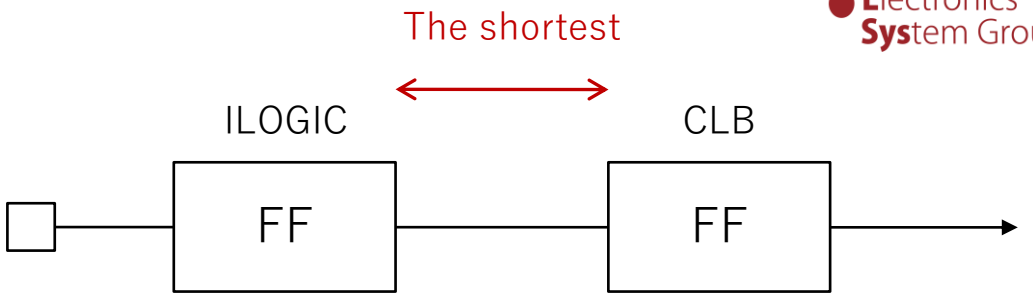
```
  attribute ASYNC_REG : string;
  attribute ASYNC_REG of sync_ff : label is "true";

begin

  sync_ff : process(clk)
  begin
    if(clk'event and clk = '1') then
      reg_sync0  <= data;
      reg_sync1  <= reg_sync0;
    end if;
```

It is used not only for asynchronous inputs from the outside but also for switching asynchronous clock domains inside FPGA.
May be used whenever an asynchronous signal is received.

Input from outside of FPGA

ASYNC_REG and IOB constraint

- With both ASYNC_REG and IOB constraint at the same time, IOB is prior.



However, priority should be given to countermeasures against metastability.
Consider if IOB is really needed.

- ➢ The wiring delay inside the FPGA is several ns at most.
- ➢ Usually a negligible value when capturing with an asynchronous clock.

If really want to use them at the same time (Please consider if really needed)

- There is a way to prepare triple FFs: apply an IOB constraint to the first FF, and manually place the remaining two FFs in nearby slices with LOC constraints.
  - Because the ASYNC_REG constraint alone does not guarantee that ILOGC will use a nearby slice.

Metastable and trouble events

Electronics System Group

In my experience, FSM is the most vulnerable for metastability (Finite State Machine)
When the signal is used for state control, it becomes metastable...

- Fall into an undefined state.
- Deadlock
- No action is taken.

, etc. Lots of trouble encountered before.

Of course, we should be careful about signals that directly control the FSM, but metastability can propagate, so be sure to take metastability countermeasures for asynchronous signals.

FSMs is also vulnerable to asynchronous resets.

Electronics System Group

LOC, RLOC

openit
Open source consortium of Instrumentation

LOC constraint and BEL constraint

Electronics System Group

LOC

- A placement constraint specifying absolute position. Fix the logic cell to the SITE.
  - When we want to place a specific cell closest to the IO (example: IOB and ASYNC_REG above)
  - With complicated design, too many degrees of freedom, and cells that should be placed close to each other are far apart.
    - However, if go that far, better to consider using PBLOCK.
- Rather than using constraints in the code, you probably use the floor planner to specify the position from the GUI.
  - Try it later.

BEL

- Specifies which internal resource to be used for the CLB SLICE.
  - BEL is not usually needed.
  - I use it to control delay amount of ps order, but it's an asynchronous usage.

In principle, you can leave the P&R to Vivado.
Avoid LOC constraints in any way.

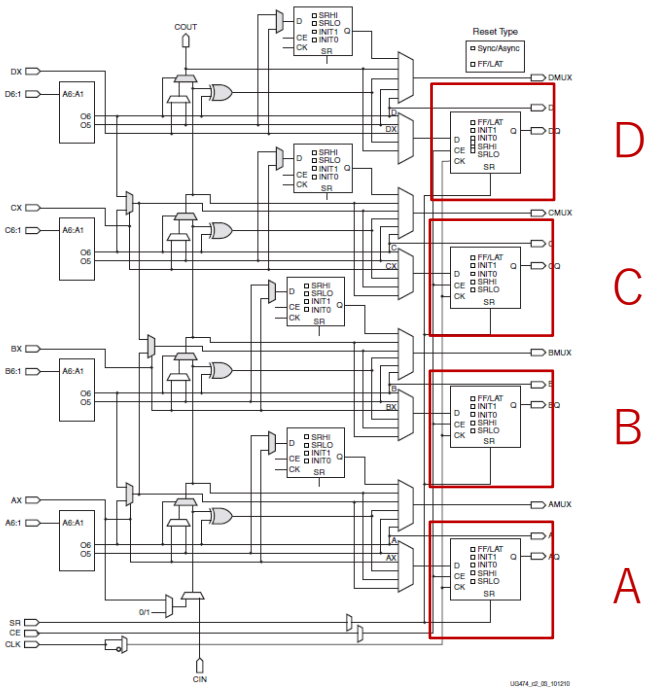


図 2-4 : SLICEL の図

openit
Open source consortium of Instrumentation

RLOC constraint

Electronics System Group

RLOC

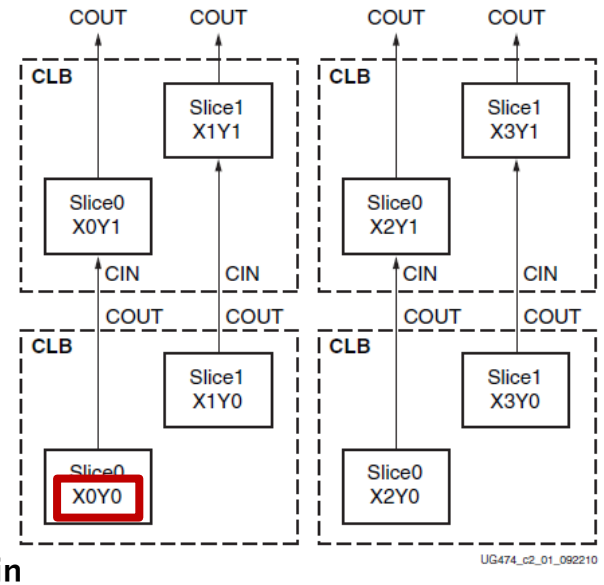
- A placement constraint specifying the relative position of the cell.
  Vivado decides where is the origin.
  - Use RLOC_ORIGIN additionally if you want to determine the origin.
- Cannot be specified in XDC, only in code.
- Enough to control the timing of a TDC with 1 ns accuracy using a 4-phase clock.
  - **Relatively close ⇔ Roughly the same wiring delay amount**

VHDL

```
attribute rloc        : string;
attribute rloc     of u_FDCE_A     : label is "X0Y0";
attribute rloc     of u_FDCE_B     : label is "X1Y0";
attribute rloc     of u_FDCE_C     : label is "X0Y1";
attribute rloc     of u_FDCE_D     : label is "X1Y1";
```

Verilog-HDL

```
(* RLOC = "X0Y0", HU_SET = "h0" *) FD sr0 (.C(clk), .D(sr_1n), .Q(sr_0));
(* RLOC = "X0Y0", HU_SET = "h0" *) FD sr1 (.C(clk), .D(sr_2n), .Q(sr_1));
(* RLOC = "X0Y1", HU_SET = "h0" *) FD sr2 (.C(clk), .D(sr_3n), .Q(sr_2));
(* RLOC = "X0Y1", HU_SET = "h0" *) FD sr3 (.C(clk), .D(sr_4n), .Q(sr_3));
(* RLOC = "X0Y0", HU_SET = "h1" *) FD sr4 (.C(clk), .D(sr_5n), .Q(sr_4));
(* RLOC = "X0Y0", HU_SET = "h1" *) FD sr5 (.C(clk), .D(sr_6n), .Q(sr_5));
(* RLOC = "X0Y1", HU_SET = "h1" *) FD sr6 (.C(clk), .D(sr_7n), .Q(sr_6));
(* RLOC = "X0Y1", HU_SET = "h1" *) FD sr7 (.C(clk), .D(inrn), .Q(sr_7));
```



In this figure, this is the origin

図 2-2 : CLB およびスライス間の行とカラムの関係
Row and column relation between CLBs and slices

I will actually try IOB and LOC.
(RLOC together with timing constraints)

If there are multiple origins in one HDL, specify the group with HU_SET.
RLOC is a constraint on the group specified by HU_SET, so it's generated by default even if not specified.

openit
Open source consortium of Instrumentation

Timing constraint

Clock definition and clock group

```
create_clock -period 10 -name clk_sys -waveform {0.000 5.000} [get_ports CLKP]

create_generated_clock -name tdc0 [get_pins u_mmcm/inst/mmcm_adv_inst/CLKOUT0]
create_generated_clock -name tdc1 [get_pins u_mmcm/inst/mmcm_adv_inst/CLKOUT1]
create_generated_clock -name tdc2 [get_pins u_mmcm/inst/mmcm_adv_inst/CLKOUT2]
create_generated_clock -name tdc3 [get_pins u_mmcm/inst/mmcm_adv_inst/CLKOUT3]
```

Source clock

The Clocks generated by MMCM from the source clock

- All the used clocks have to be defined.
- When using MMCM or PLL, Vivado automatically defines both the source clock and the generated clock.
  - But, since it is hard to understand and the serial number is attached, I re-define it myself.

```
create_generated_clock -name clk_div2 -source [get_ports CLKP] -divide_by 2 [get_pins instance_name/pin_nDame]
```

- Don't forget to define a user-defined clock which is divided by 2 by FF.

If we want a list of clocks automatically generated by Vivado
- Load the synthesized design, and run report_clocks in the Tcl Console.
- It can also be used to check if any user-defined clock is missing.

Clock definition and clock group

Why define?

- Necessary to tell Vivado how to specify synchronization/asynchronization between clocks.

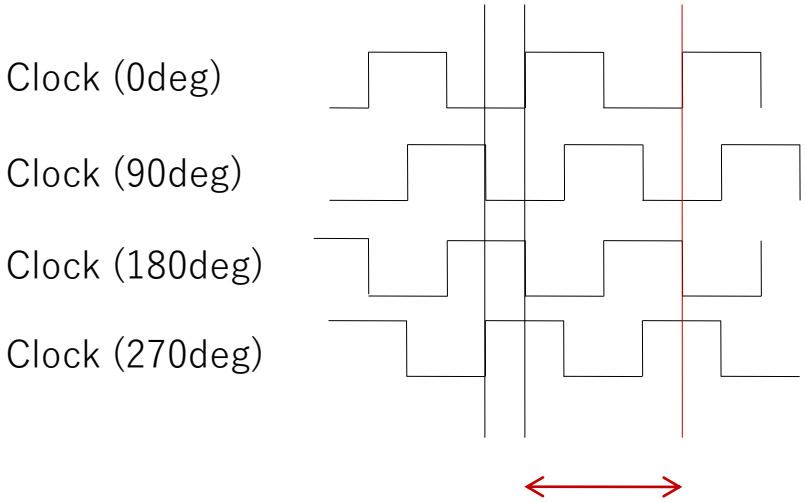
asynchronous constraint with set_clock_group

- By default, all the defined clocks are estimated in the timing analysis as synchronous.
- Clocks that are not synchronous to begin with, and synchronous clocks but may be excluded from timing analysis, are asynchronously specified.

```
set_clock_groups -name async_sys -asynchronous \
    -group {clk_tdc_0 clk_tdc_90 clk_tdc_180 clk_tdc_270} \
    -group clk_sys \
    -group clk_gtx \
    -group clk_spi \
    -group clk_icap \
    -group clk_10mhz
```

In between, the clocks grouped together are included in the timing analysis.
Groups are excluded from analysis as asynchronous.

multicycle_path



- For example, when using clocks with the same frequency and different phases in every 90 degrees as shown on the left, consider switching from 270 degrees to 0 degrees domain.
- At this time, the allowable time is only 1/4 of the original frequency, and timing is not met in most cases.

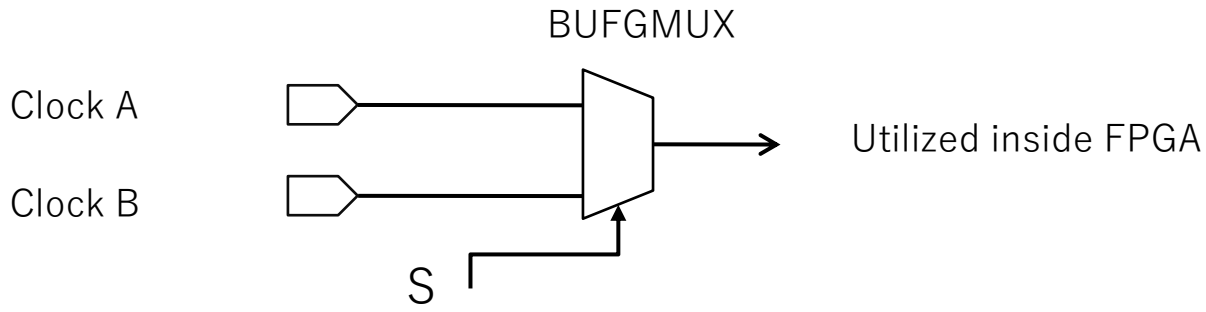
```
set_multicycle_path -setup -from [get_clocks tdc3] -to [get_clocks tdc0] 2
```

- Therefore, use the multicycle_path constraint, when switching from 270 degrees to 0 degrees, the edge after multiple times is used instead of the next clock edge.
  - The example above is timed to ensure that it is captured correctly on the second edge.
  - Allowable latency is 5/4 cycles.

When switching multiple clocks with BUFGMUX

Clocks A and B have the same frequency but are from different sources.

- EX: A is from a local oscillator, for debugging. B is a master clock distributed from upstream, for main logic.



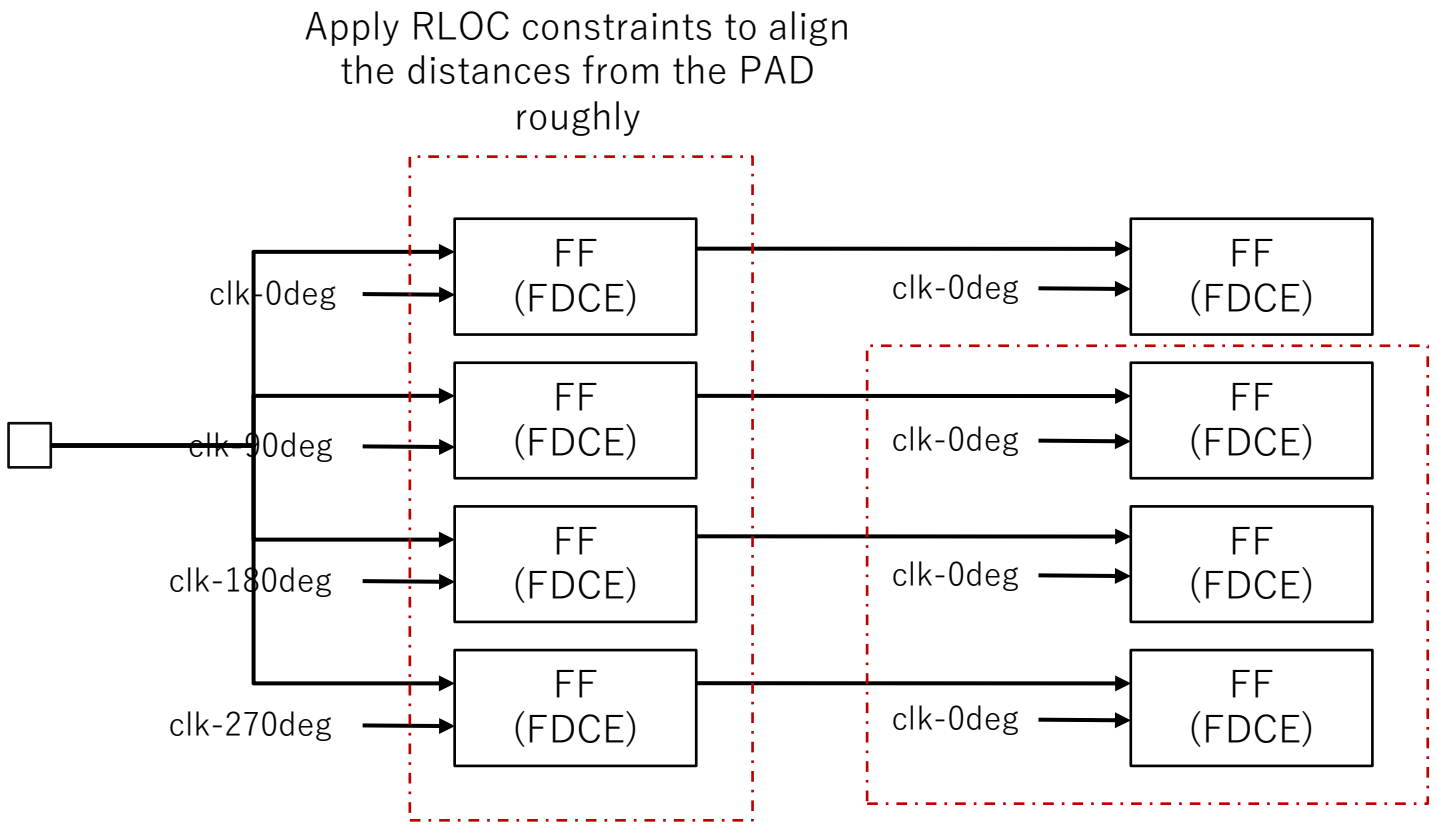
- For some reasons, Vivado tries to perform a timing analysis between clocks A and B and gives a timing violation.
- We need to specify that those two are identical and exclusive, and only one of them needs to be analyzed.

```
# Clock definition
create_clock -period 10.000 -name clk_osc -waveform {0.000 5.000} [get_ports CLKOSC_P]
set_input_jitter clk_osc 0.030

create_clock -period 10.000 -name clk_hul -waveform {0.000 5.000} [get_ports CLKHUL_P]
set_input_jitter clk_hul 0.030

set_case_analysis 0 [get_pins u_BUFGMUX_inst/S]

set_clock_groups -name async_input -physically_exclusive -group [get_clocks clk_osc] -group [get_clocks clk_hul]
```

S input of MUX. It means to analyze only the case of S=0.

Indicate physically exclusive with set_clock_group

Now let's try with RLOC, multicycle_path, and clock_group again.
Here, we will implement the first four FFs when making a TDC using a 4-phase clock and the next clock transfer.



A clock domain crossing happens
Specify the ulticycle_path for the transition from 270 degrees to 0 degrees.

Tips for better design

About distributing reset

Prerequisite knowledge

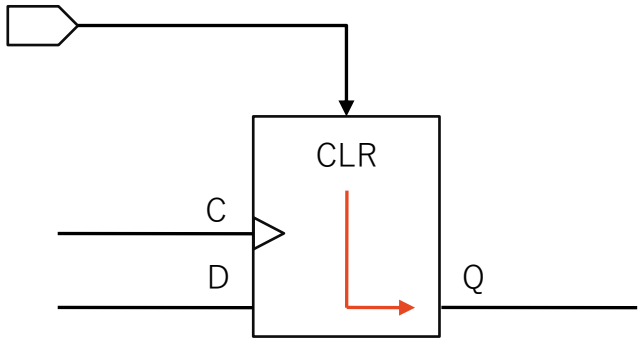
Xilinx uses the following notation for CLB register asynchronous (synchronous) reset (set).

- (Asynchronous) Clear (CLR)
- (Asynchronous) Preset (PRE)
- (Synchronous) Reset (R)
- (Synchronous) Set (S)

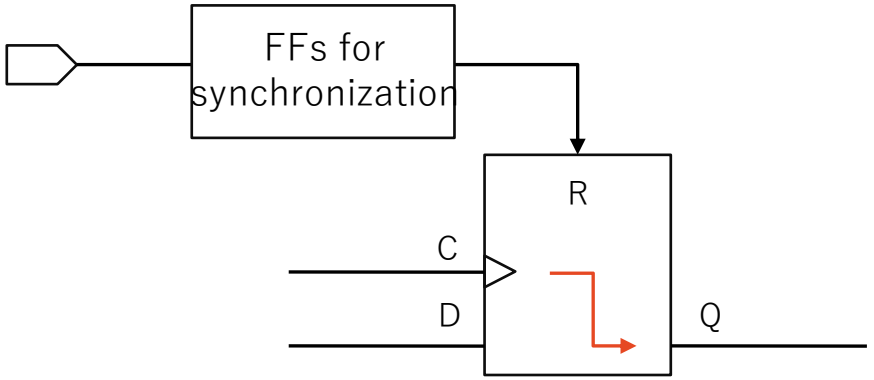
⇨ FDCE in Primitive name
D-type FF element with clock enable and asynchronous reset (clear)

We have used several way to distribute reset in previous examples.
Roughly speaking, the following two types

Asynchronous reset



Synchronous reset



It is clear that an asynchronous reset circuit with unknown transition timing of Q is risky.
In principle, do not use asynchronous resets (sets).
This is easy, but there are more complicated situations in reset distribution.

About distributing reset

The content here is extracted from the Xilinx white paper (WP231).
It is a rather old document which does not work with the current Xilinx FPGA,
but very well written in general terms.

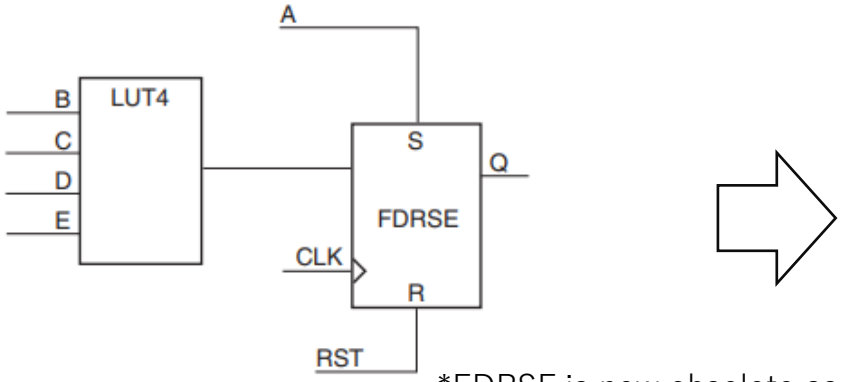
Asynchronous reset (set) and synchronous reset (set) are optimized differently when using CLB resources

- Q must transit synchronously with CLK.
- R and S, which are synchronous to CLK, can be used as part of a combinational circuit.

VHDL	Verilog

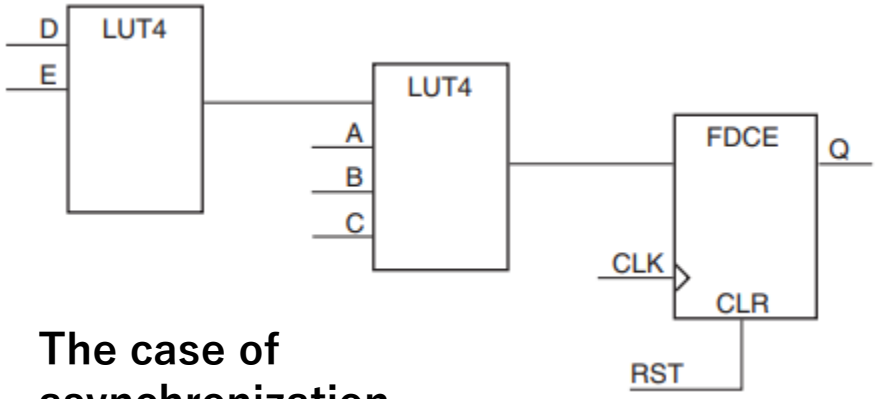
process (CLK)
begin
  if (rising_edge(clk)) then
  if (RST = '1') then
  Q <= '0';
  else
  Q <= A or (B and C and D and E);
  end if;
  end if;
end process;
	always @(posedge CLK)
	  if (RESET)
	  Q <= 1'b0;
	  else
	  Q <= A | (B & C & D & E);

The case of synchronization



*FDRSE is now obsolete as a primitive

*In the current FPGA, the LUT has 6 inputs, so this is not the case now.



The case of asynchronization

Open source consortium of Instrumentation

About distributing reset

Electronics System Group

That reset signal is really needed? Unnecessary resets harms optimization

- Series FFs (CLB registers) can be replaced with SRL16 (implemented in LUTs) or distributed RAM.
  - SRL and RAM have no reset input. It is not optimized if there is reset behavior description.

- There is one type of reset signal which can be accepted by one slice.
  - Distributing multiple resets ensures that leaf cells with different resets are split into different slices, consuming extra fabric area.
  - It's the same for clocks. To increase the utilization efficiency of the fabric, it is better to reduce the number of clock domains as much as possible.

- If trying to distribute an asynchronous reset to a built-in register in a dedicated resource which only accepts synchronous resets, it can cause some functionalities to seep into the fabric.

(Perhaps it is a case of estimating dedicated resources by behavior description in HDL)

About distributing reset

Electronics System Group

Remove unnecessary reset distributions

- Bus and sequence control signals need to transit to 0 correctly at the reset timing. But for data lines, for example, they just flow and disappear when the control signal goes to 0, so no need to distribute resets.
  - (Maybe. Case dependent.)

- By the way, what happens when power on without a reset?
  - Xilinx FPGAs have a signal distributed to all primitives on power up.
    It is called the Global Set Reset (GSR) signal.
  - With GSR input, the output of each primitive is initialized to the value specified by INIT.

About distributing reset

What to be careful about if you could handle it completely with just a synchronous reset (set)?

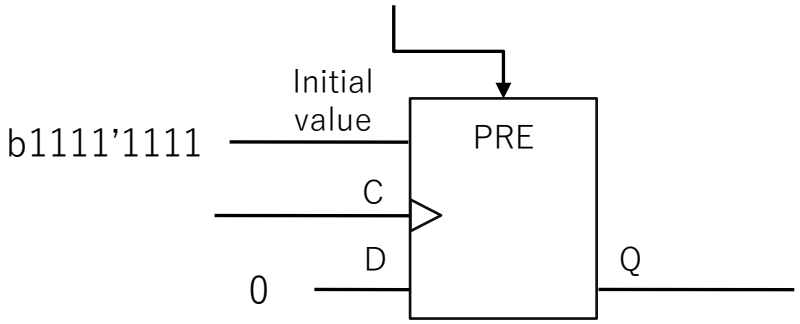
- Timing analysis is required for all reset paths. A large FW may reduce the operation speed.
- Requires large fanout to get the same signal to multiple locations
  - Remove unnecessary resets.
  - Use BUFGs (Global Clock Lines) to distribute reset.
    - Worth a try if there is still extra BUFG.
- Always re-synchronize the reset signal between asynchronous clock domains.

In fact, there is also a wat to actively distribute GSR.
To do so, we use a primitive called STARTUP~, which controls the configuration of the FPGA.
It's a pretty maniac solution. If you are considering using it, read each UG carefully before using it.

About distributing reset

When an asynchronous reset is really needed
(e.g. loss of clock due to reset)

- There is a technique that the reset entry is asynchronous and the exit is synchronous.



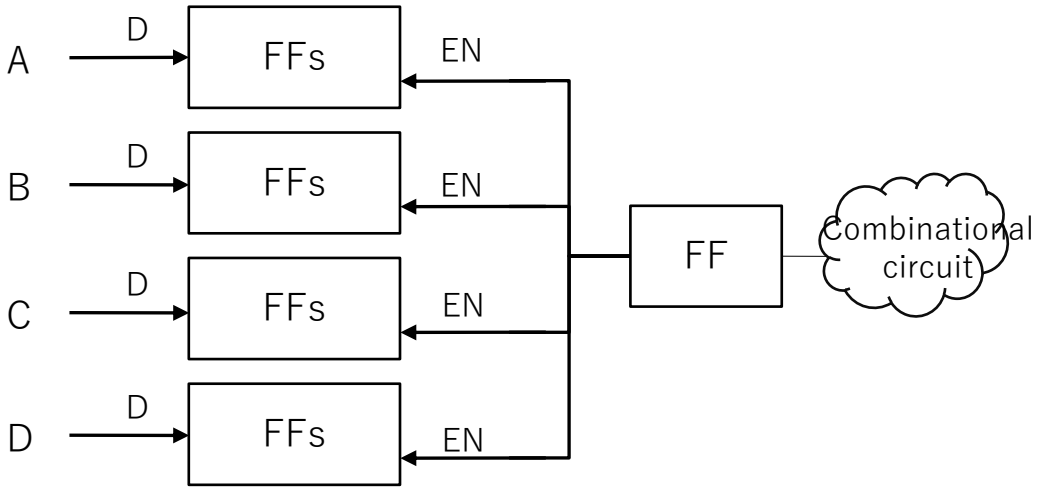
A shift register of a length 8 bits that presets b1111'1111 when an asynchronous set comes in
(The length depends on number of clock cycles required)

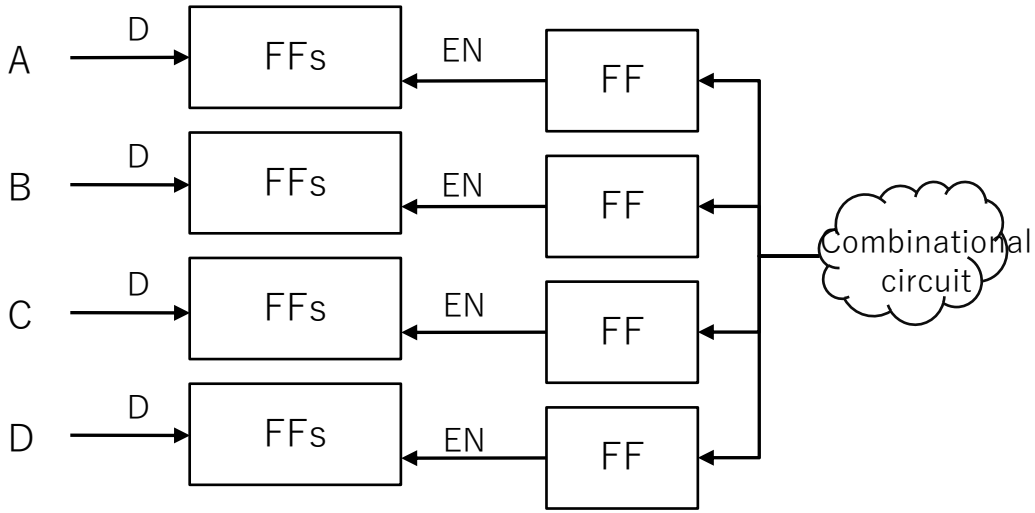
Distribute Q to asynchronous reset

```
-- Reset sequence --
sync_reset  <= reset_shiftreg(kWidthResetSync-1);
u_sync_reset : process(rst, clk)
begin
  if(rst = '1') then
    reset_shiftreg  <= (others => '1');
  elsif(clk'event and clk = '1') then
    reset_shiftreg  <= reset_shiftreg(kWidthResetSync-2 downto 0) & '0';
  end if;
end process;
```

Duplication of large fanout and driver

Which way can use more fabric when distributing the same signal to many endpoints?
(Wiring delay always exists in the signal transmission)





- The right one looks better, but even if you write HDL like this, it is often combined into one fanout due to optimization. (Driver duplication and deletion is a kind of optimization)
- Specify KEEP in **EQUIVALENT_DRIVER_OPT constraint** can inhibit driver integration for the same signal.
  It cannot be constrained in HDL. It is done for cells in XDC.

UG912

- For a circuit which takes matrix coincidence between detector segments, it often distributes the same signal to multiple endpoints.

Consider inserting CLB registers

Electronics System Group

- For the resources other than CLB, they are stored in specific rows within the FPGA and cannot be moved.
- Signal transmission between dedicated resources causes a large wiring delay and it can be a cause of lowering the operating frequency.
  - Throughput is improved by appropriately inserting registers between dedicated resources and providing relay points in the fabric (trade-off with latency).

As a design point

- Design so that the logic will not collapse even if the pipeline register is retrofitted.
- Allow to add additional delays.

How to write the source code

- Use the transfer of signal line names (There are pros and cons for this, so let's decide the coding rules)

```
din_moduleB   <= dout_moduleA;
```

Simple change of signal line name
(Description is often seen in Xilinx example code)

```
process(clk)
begin
  if(clk'event and clk= '1') then
    din_moduleB   <= dout_moduleA;
  end if;
end process;
```

Registers can be added without defining new signal lines.
Even when adding modules, it reduces the amount of rewriting.

On the other hand…
With a larger number of signal lines to be defined, the readability is worse.
Decide on your own coding style.

Stop using magic numbers

Electronics System Group

A magic number is a solid number that you don't know the meaning at first glance.

In HDL, we are used to do it for bus width descriptions and preset values, etc.
It greatly makes the readability worse for others.

Both Verilog-HDL and VHDL have a way to include predefined constants,
so please avoid magic numbers.

About large IP (logic)

EX: implement a large FIFO like below.

- Input data width: 32-bit
- Input data width: 8-bit
- Depth for writing: 65536
- Clock speed for writing: 125 MHz
- Clock speed for reading: 250 MHz

Built-in FIFO cannot be used because the data width is non-symmetric aspect ratios.

Now let's implement it, and wire the input to VIO and the output to ILA.
It's a very simple (or almost nothing) circuit, and the frequency isn't that high.
Just one thing... read depth is expected to be very large.

Will P&R pass?

About large IP (logic)

Got a timing error (Running in Vivado 2020.1, Windows 10)

Timing	Setup | Hold | Pulse Width
Worst Negative Slack (WNS):	-0.571 ns
Total Negative Slack (TNS):	-4.788 ns
Number of Failing Endpoints:	16
Total Number of Endpoints:	8558
Implemented Timing Report	

The arrival point is hidden and do not know what kind of signal it is.
In any case, it is impossible to change the design because it is inside the IP.

Summary

Name	Path 21
Slack	-0.571ns
Source	u_fifo0/U0/inst_fifo_gen/gconvfifo.rf/grf.rf/gntv_or_sync_fifo.mem/gbm.gbmg.gbmga.ngecc.bmg/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[56].rar
Destination	<hidden> (rising edge-triggered cell FDRE clocked by clk_sys {rise@0.000ns fall@2.000ns period=4.000ns})
Path Group	clk_sys
Path Type	Setup (Max at Slow Process Corner)
Requirement	4.000ns (clk_sys rise@4.000ns - clk_sys rise@0.000ns)
Data Path Delay	4.234ns (logic 2.162ns (51.066%) route 2.072ns (48.934%))
Logic Levels	5 (LUT3=1 LUT6=2 MUXF7=1 MUXF8=1)
Clock Path Skew	-0.307ns
Clock Un...rtainty	0.065ns

About large IP (logic)

This is an intentional timing error.
The reason is that the circuit on the output side has become huge due to the asymmetric data width.

What to do?
If the FIFO depth cannot be changed due to the data size, and the asymmetric data width is also required…
Let's divide it into:

- Huge built-in FIFO for data buffering
- BRAM FIFO for small data width conversoin

and implement it.

About large IP (logic)

Then, the timing error disappeared with the all necessary functions.

This is just one possible case.
A large and complicated circuit never raises the operating frequency.

Especially when there is a propagation delay from the lower bits to the upper bits,
be careful with circuits with large counters.
(For example, RAM pointer incrementing)

Try to make design as simple as possible, and make each circuit in small-scale.

If timing violation

Every day, many people are fighting with Worst Negative Slack (WNS).
There's no one-size-fits-all solution, but let's consider some cases.
(Assuming false_path and clock_group constraints are correct)

The case when the fabric usage is not too high (many free clock regions)
Since the degree of freedom in placement and wiring is low, it is expected that the operating frequency limit will be reached.

- First, explore the cause of the delay from the timing summary.
  - If mainly due to logic, there is room to insert pipeline registers.
- It is possible that the reset signal cannot arrive within the required time.
  - We will work with a combination of the techniques described so far.

Summary

Name	Path 21
Slack	-0.571ns
Source	u_fifo0/U0/inst_fifo_gen/gconvfifo.rf/grf.rf/gntv_or_sync_fifo.mem/gbi
Destination	<hidden> (rising edge-triggered cell FDRE clocked by clk_sys {ris
Path Group	clk_sys
Path Type	Setup (Max at Slow Process Corner)
Requirement	4.000ns (clk_sys rise@4.000ns - clk_sys rise@0.000ns)
Data Path Delay	4.234ns (logic 2.162ns (51.066%) route 2.072ns (48.934%))
Logic Levels	5 (LUT3=1 LUT6=2 MUXF7=1 MUXF8=1)
Clock Path Skew	-0.307ns
Clock Un...rtainty	0.065ns

If timing violation

Electronics System Group

High amount of fabric usagage (less than 70-80% slice usage for 200T FPGA)

There is a high degree of freedom in placement and routing, and there is a possibility that the correct answer has not been reached.

More difficult from low utilization with a huge FPGA.

- Try the ideas that make extensive use of the fabric, such as inserting pipeline registers and duplicating drivers to distribute the same signals widely.
- If still not working, consider removing the logic and wiring, especially the reset wiring.
- Consider whether there are functions implemented in CLB which can be transferred to dedicated blocks such as DSP and BRAM.

Correction by subtraction may produce better results than correction by addition (which increases the degree of freedom).
(It is sometimes necessary to remove excess pipeline registers.)
It is nice to promote fabric optimization to reduce utilization.

When slice usage exceeds 70-80%

It becomes extremely difficult to maintain the operating frequency.

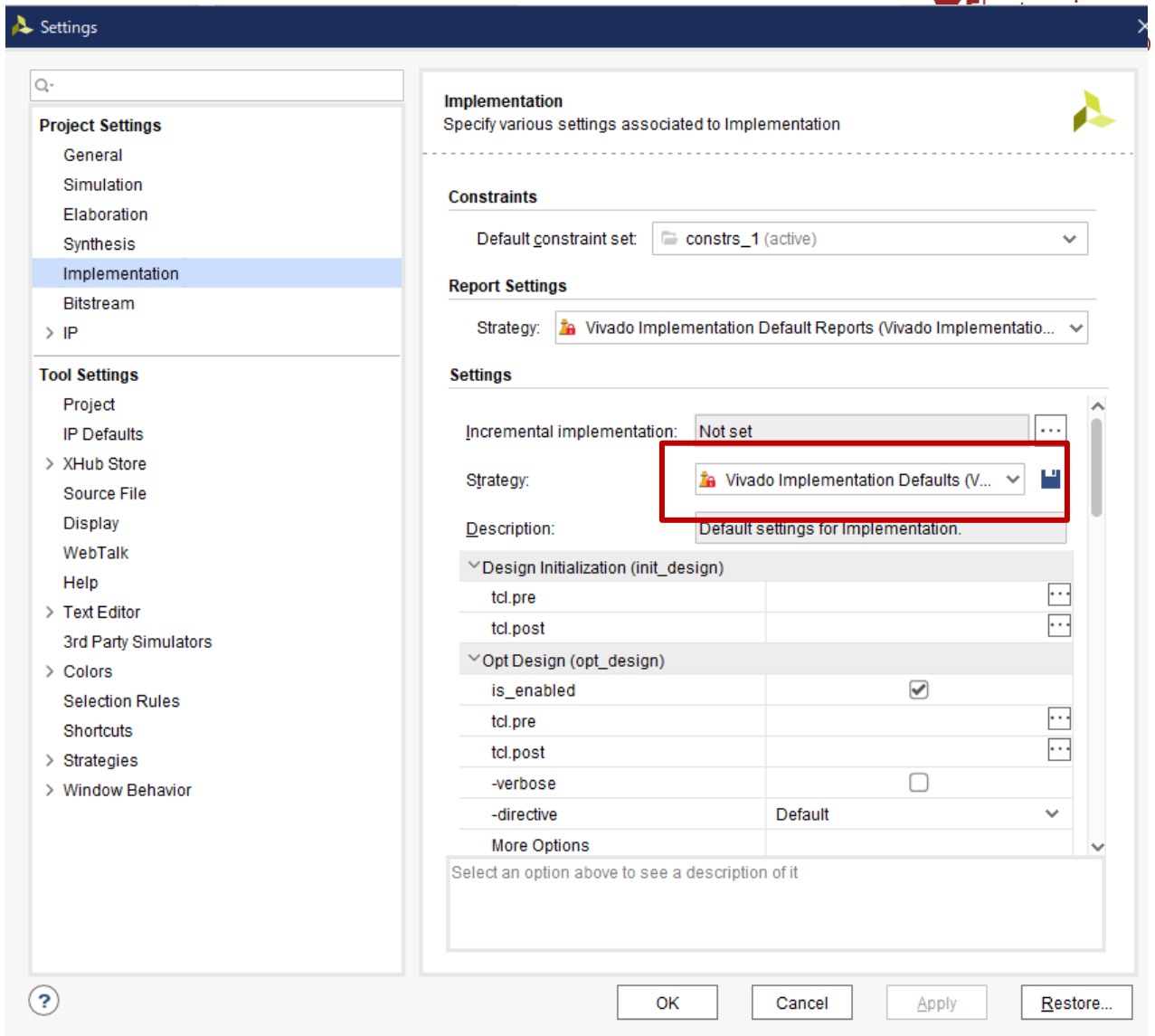
- **First, I want to select an FPGA so that this kind of situation does not occur.**
- Vivado cannot reach the correct answer unless the degree of freedom is lowered by restricting the area where logic cells can be used by making full use of constraints such as PBLOCK.
- Too difficult for beginners.

If timing violation

- If all else fails, one option is to change the P&R (and synthesis) strategy.
- There are various strategies, such as space saving, power saving, and high performance, and it is effective when it seems that the timing will be met shortly.

In principle, I develop with the default strategy.
(Because FW in this industry often makes small improvements over and over again)
Considering selling it out, you can develop with the desired strategy from the beginning.

Re-make the design until it deterministically meets timing, rather than risking timing violations every time of code fix.



Open source consortium of Instrumentation