IOSERDES

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

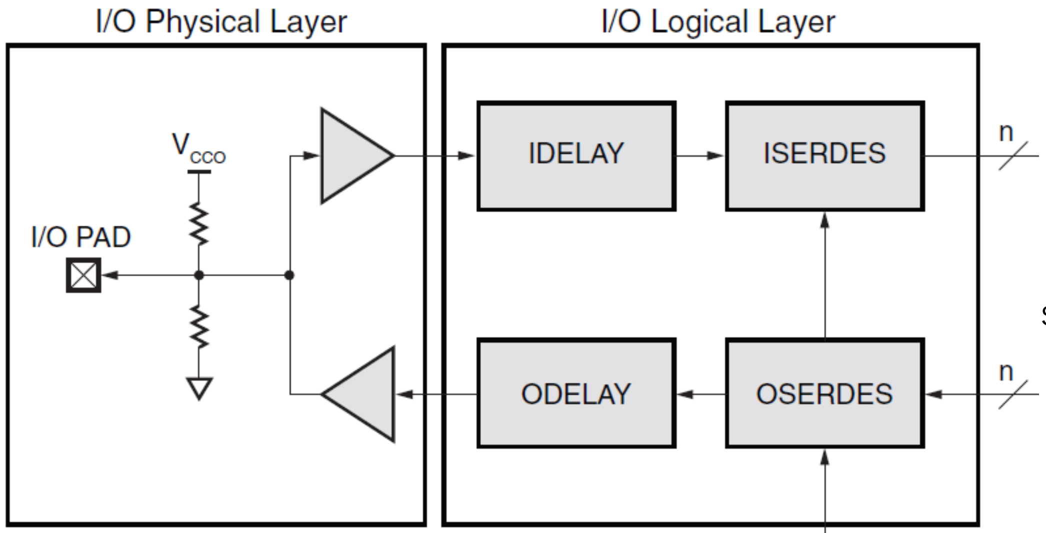
IOタイル

ILOGIC (resource)
De-serializer, ISERDES (Primitive)

- Convert 1-bit signal line into multiple-bit data (Data In)

Physical layer

- determine the physical properties of IO
    - Termination resistor (input)
    - Input common mode voltage
    - Output voltage
    - etc…



Basic I/O Block Structure for Single I/O

When the logical layer is not used, the signal line is directly connected from the IOB to the fabric.

ODELAY exsits only in HP bank.

Single data rate (SDR)

- Use only rising (or only falling) clock edges

Double data rate (DDR)

- Use both edges of the clock

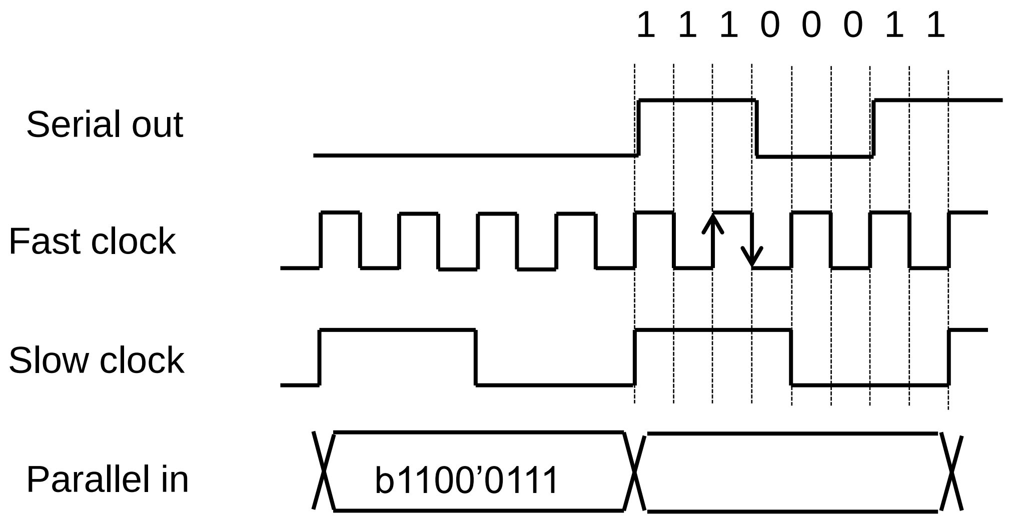
OLOGIC (resource)
Serializer, OSERDES (Primitive)

- Convert multi-bit data to 1-bit string signal (Data Out)

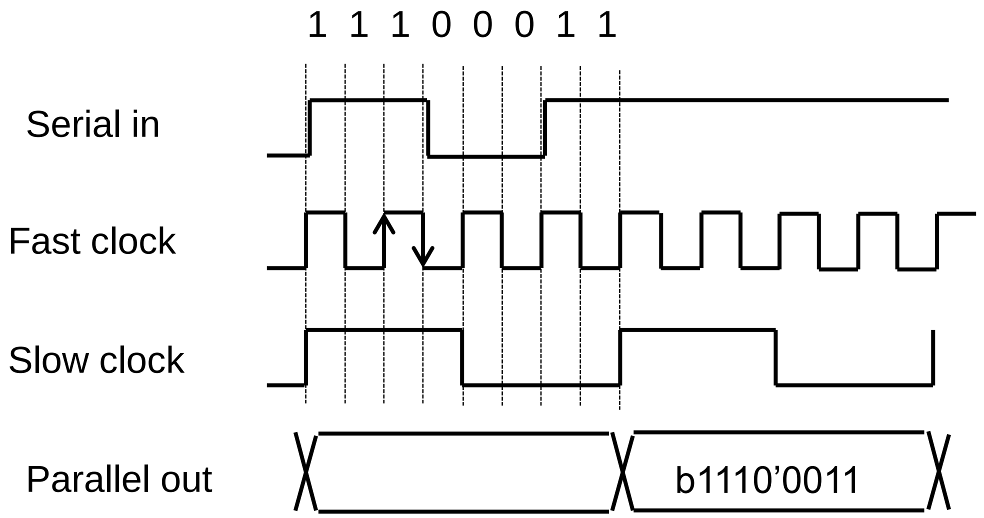
Open source consortium of Instrumentation

IOタイル

Parallel-to-Serial
(OSERDES)



Serial-to-Parallel
(ISERDES)



Skills targeted in this chapter

We will proceed with the actual circuit board.

IOSERDES

- Implement OSERDES and transmit 10-bit data at 200 MHz DDR (400 Mbps).
  - Easy for sending
- Receive the data from ISERDES and restore 10-bit data.
  - Learn to use BUFIO and BUFR.
  - Since the slow clock has a phase difference w.r.t. the data, shift the data by bit slip

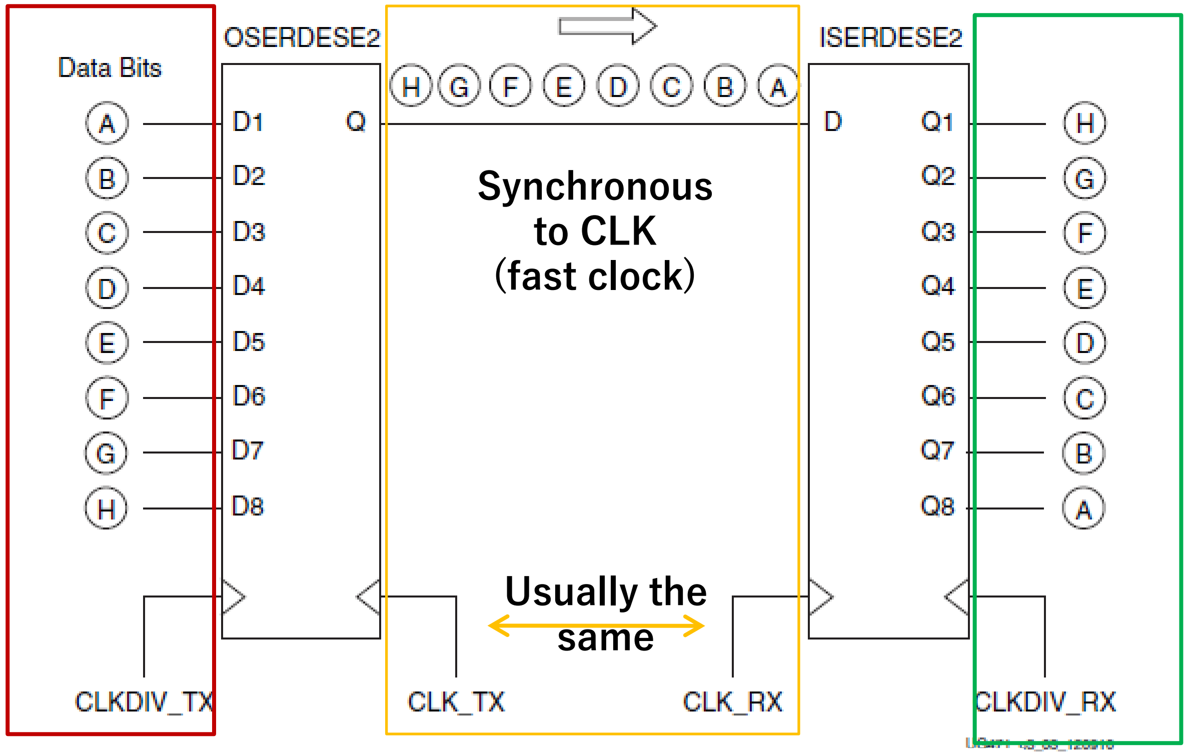
IDELAY

- The timing between data and fast clock is not always correct.
- A block for adjustment by giving a delay of several tens of ps to the data side.
- Check what happens when you add a delay using IDELAY.

IDELAYCTRL

- Learn the proper points since it is difficult for beginners.

Sending by OSERDES and receiving by ISERDES



Synchronous to CLK (fast clock)

Usually the same

Figure 3-3: **Bit Ordering on Q1–Q8 Outputs of ISERDESE2 Ports**

UG471

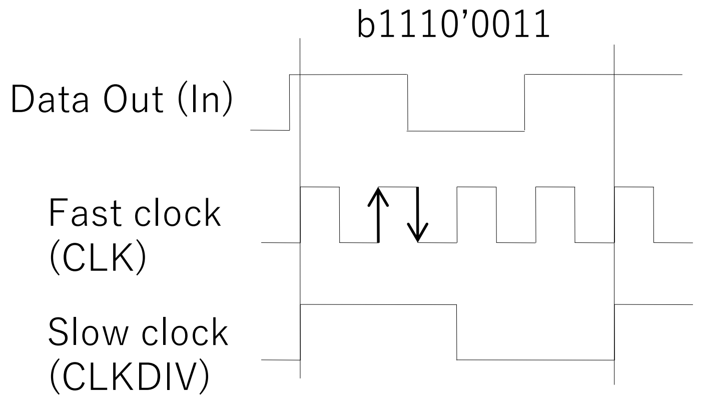
Parallel data is synchronous to CLKDIV (slow clock)

Parallel data is synchronous to CLKDIV

CLKDIV_RX is made from CLK_RX

Reverse the bit order
When generating in the IP catalog, it will be flipped on the sending side.
When implemented by hand, turn it over by yourself.

8-bit DDR IOSERDES

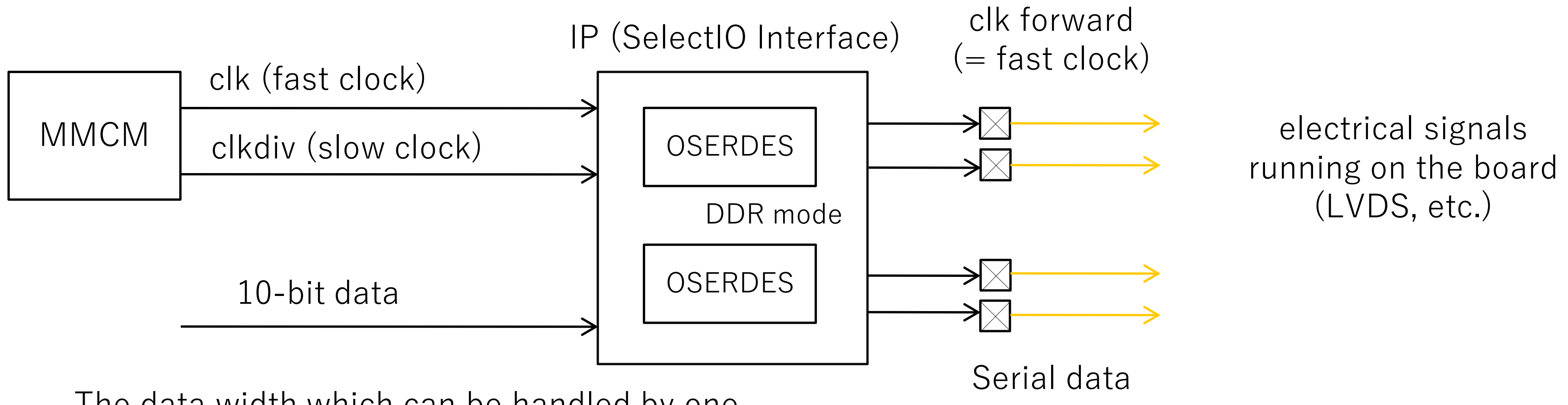


A single IOSERDES primitive can handle up to 8-bits.
For mow, cascade is needed.

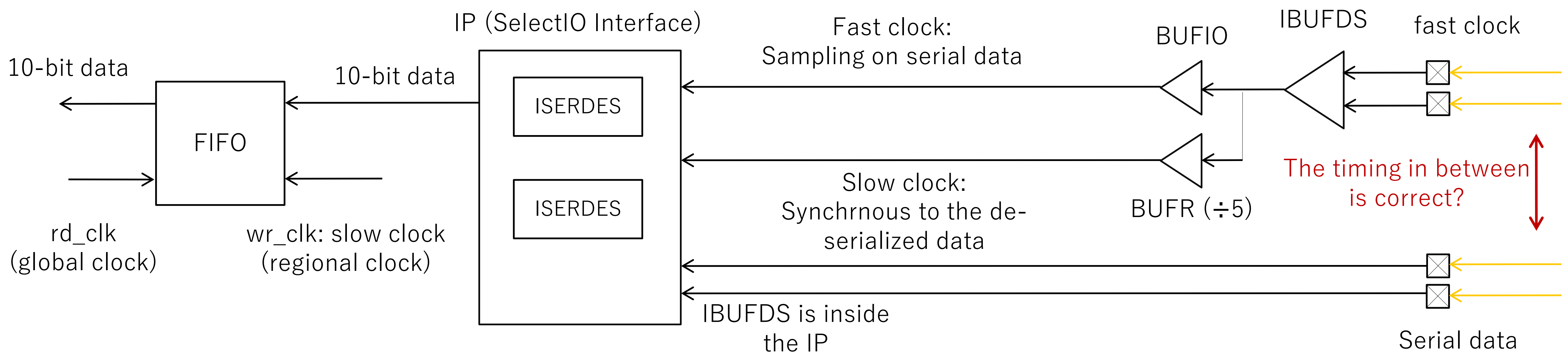
How to implement it?
First, try to make it with the IP Catalog.
Check what kind of HDL was generated,
It is expected to manually implement it eventually.

Realistic firmware construction

When sending 10-bit data using SERDES

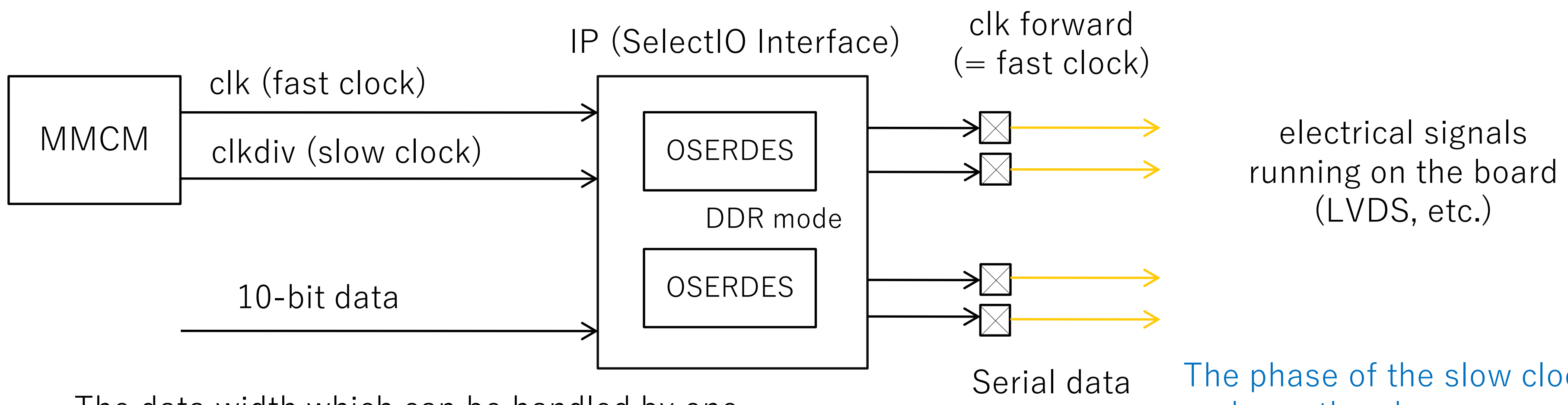


The data width which can be handled by one SERDES is up to 8-bit.
It can be expanded to 14-bit by cascading two.



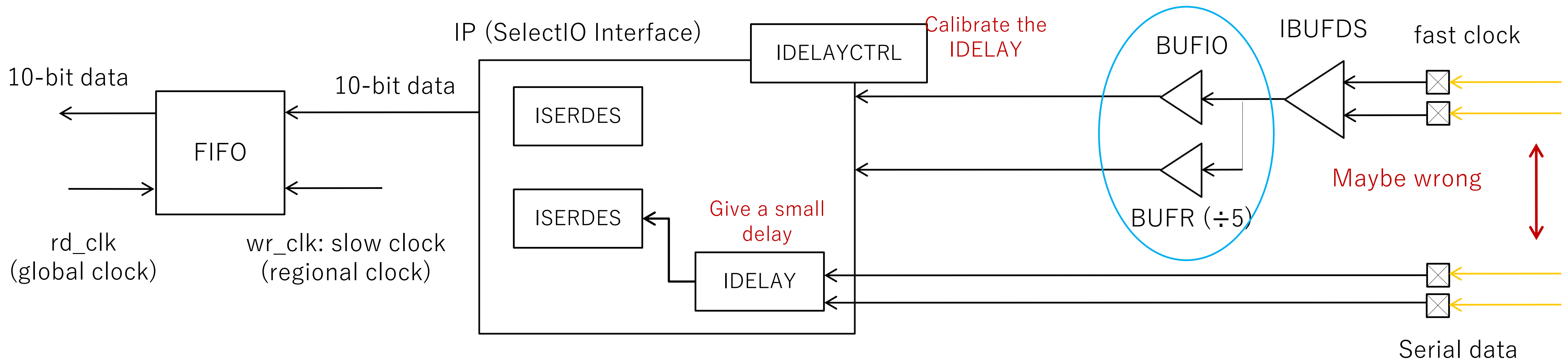
Realistic firmware construction

When sending 10-bit data using SERDES



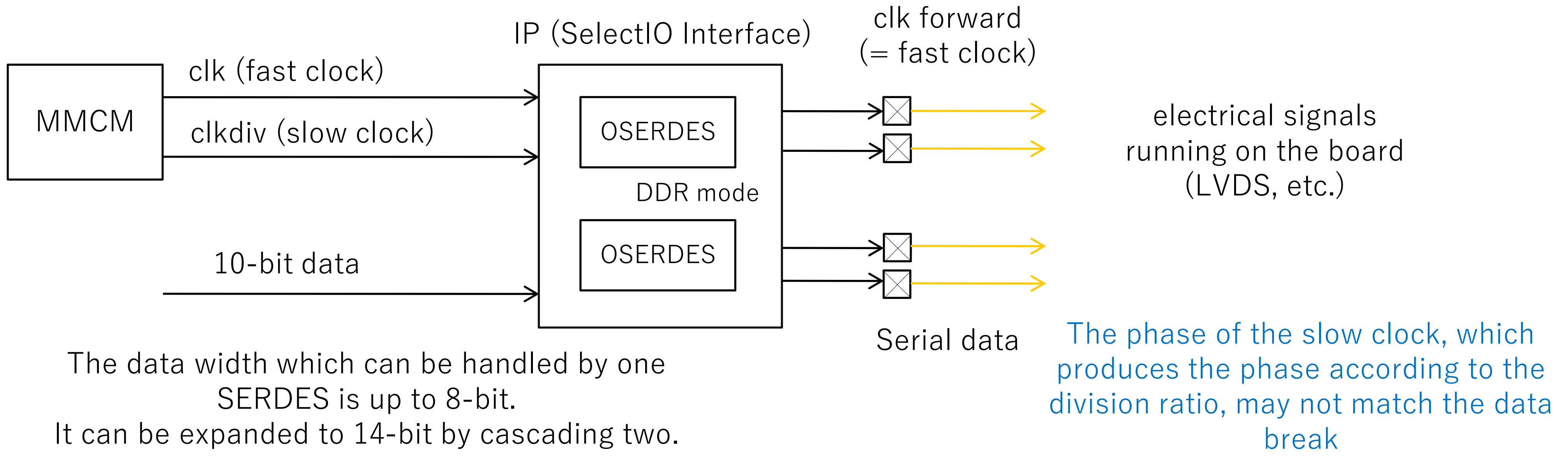
The data width which can be handled by one SERDES is up to 8-bit.
It can be expanded to 14-bit by cascading two.

The phase of the slow clock, which produces the phase according to the division ratio, may not match the data break



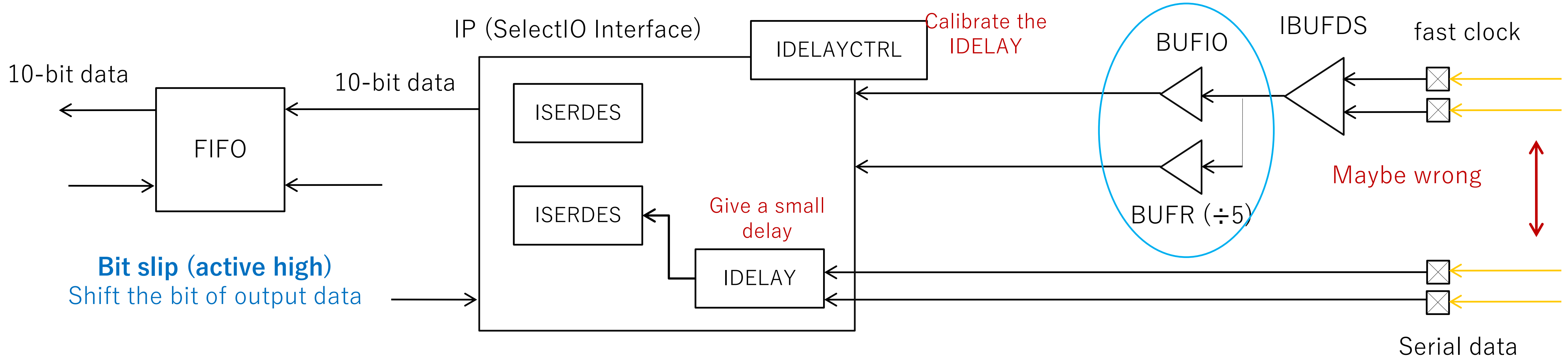
Realistic firmware construction

When sending 10-bit data using SERDES



The data width which can be handled by one SERDES is up to 8-bit.
It can be expanded to 14-bit by cascading two.

The phase of the slow clock, which produces the phase according to the division ratio, may not match the data break

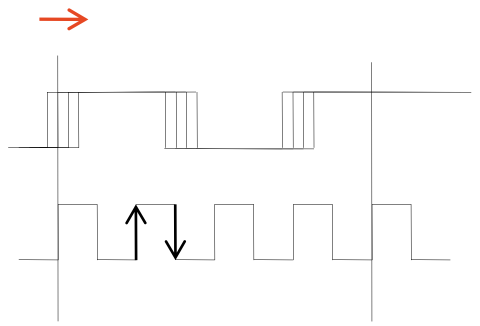


Bit slip (active high)
Shift the bit of output data

Diagram for IDELAY

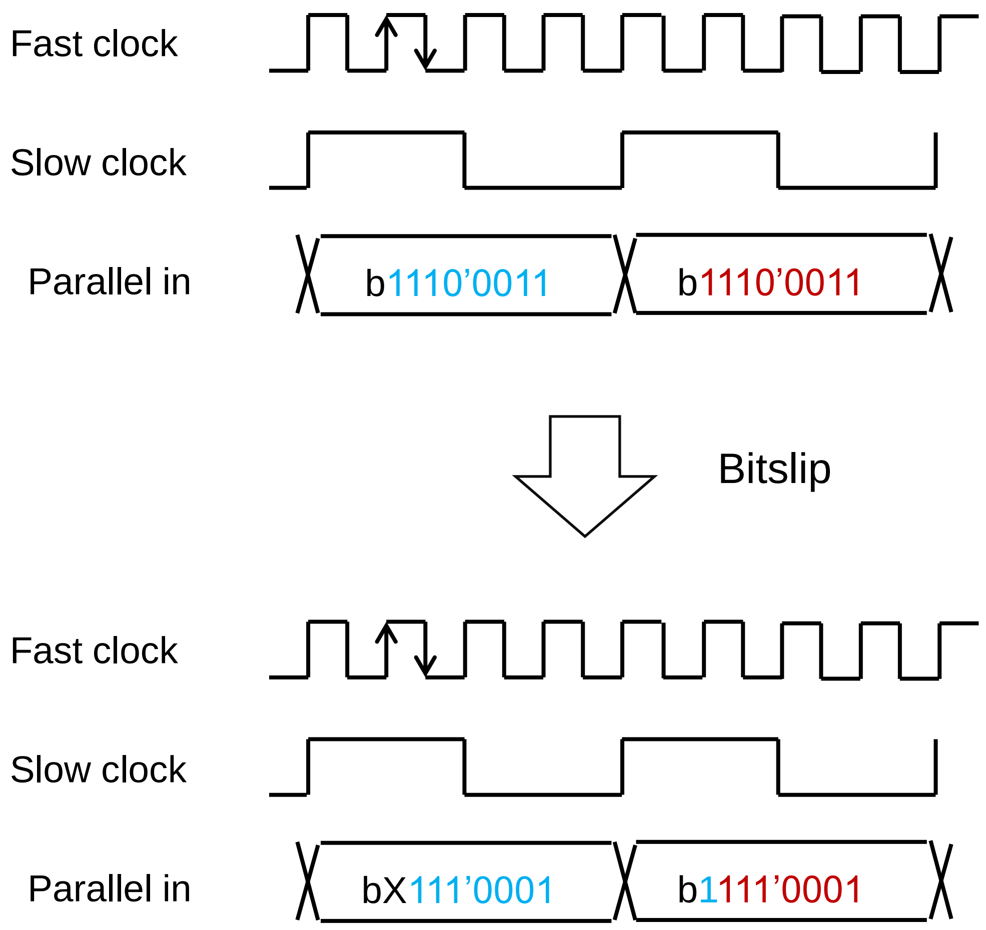
IDELAY

Changing the timing relative to
the clock edge
IDELAY

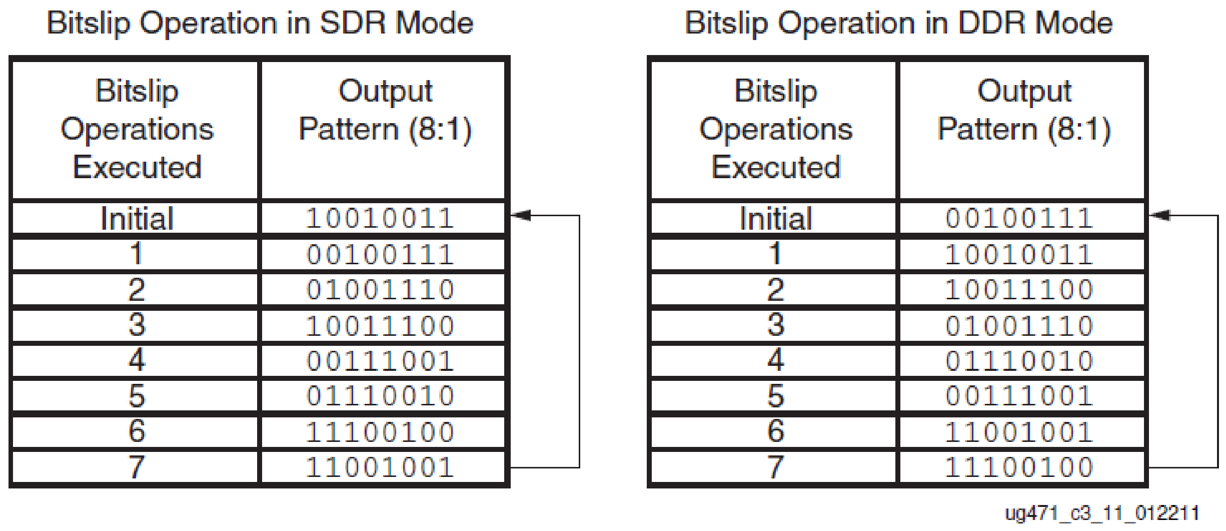


With **IDELAY,** it is also possible
to give a delay to the clock

Diagram for bitslip



Bit slip



Bitslip Operation in SDR Mode

Bitslip Operations Executed	Output Pattern (8:1)
Initial	10010011
1	00100111
2	01001110
3	10011100
4	00111001
5	01110010
6	11100100
7	11001001

Bitslip Operation in DDR Mode

Bitslip Operations Executed	Output Pattern (8:1)
Initial	00100111
1	10010011
2	10011100
3	01001110
4	01110010
5	00111001
6	11001001
7	11100100

ug471_c3_11_012211

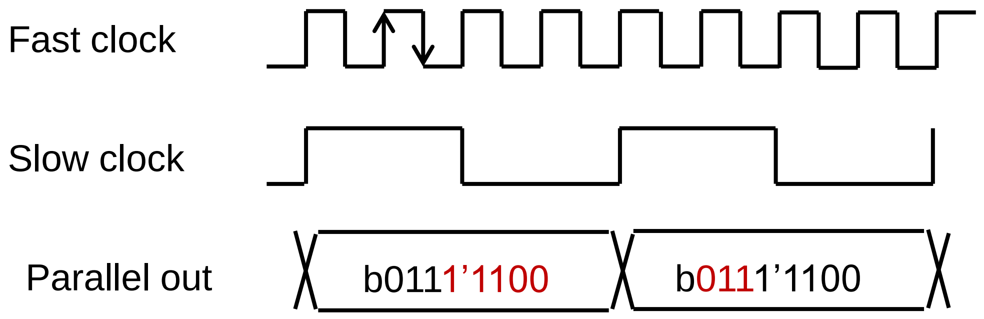
図 3-11 : Bitslip の処理例
Example of bitslip

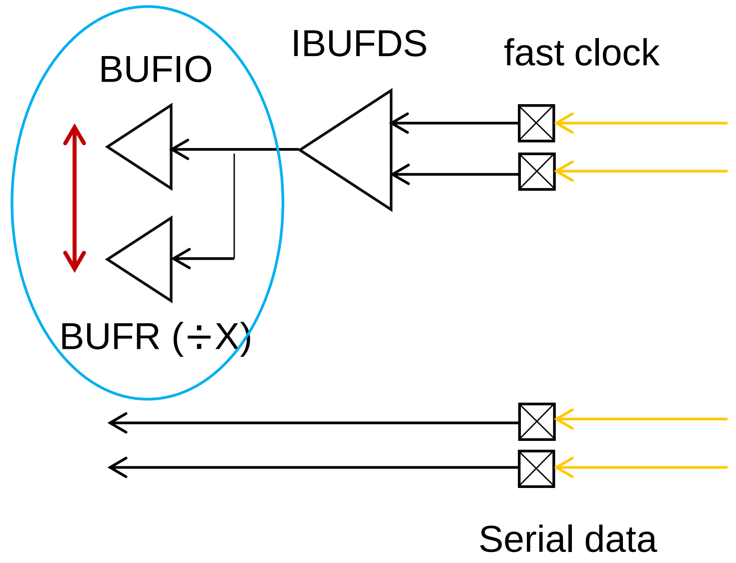
Bitslip works in a different way in SDR and DDR mode, and it's a little more complicated in DDR mode.

Why bitslip is necessary?

For example,
ADC sends 8-bit data (b1110’0011)...

The phase of the slow clock, which produces the phase according to the division ratio, may not match the data break



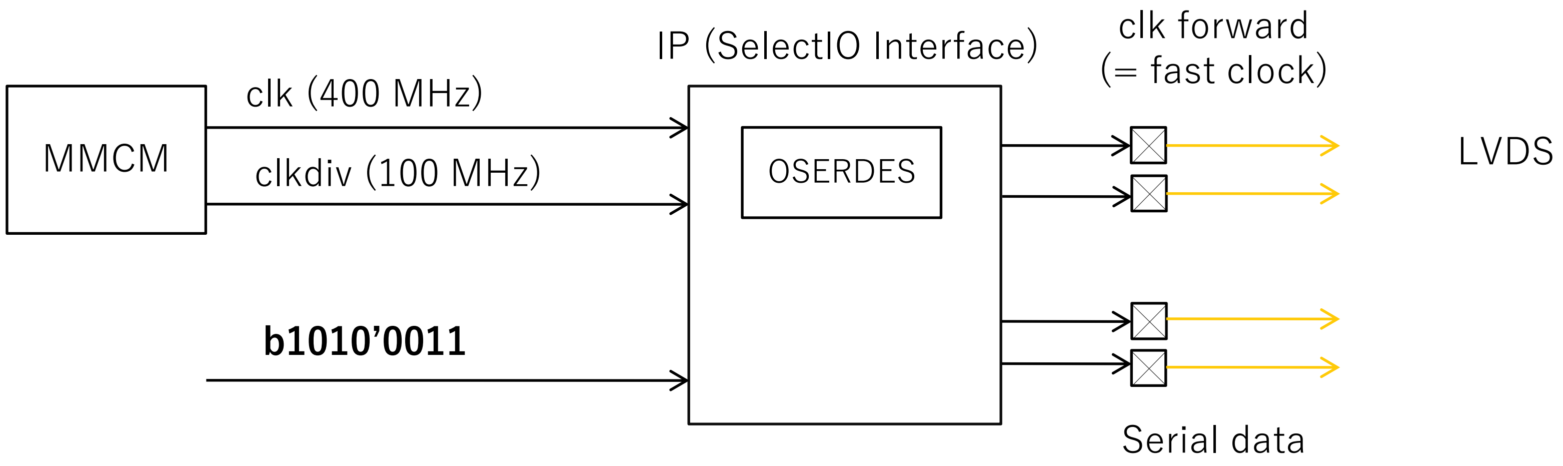


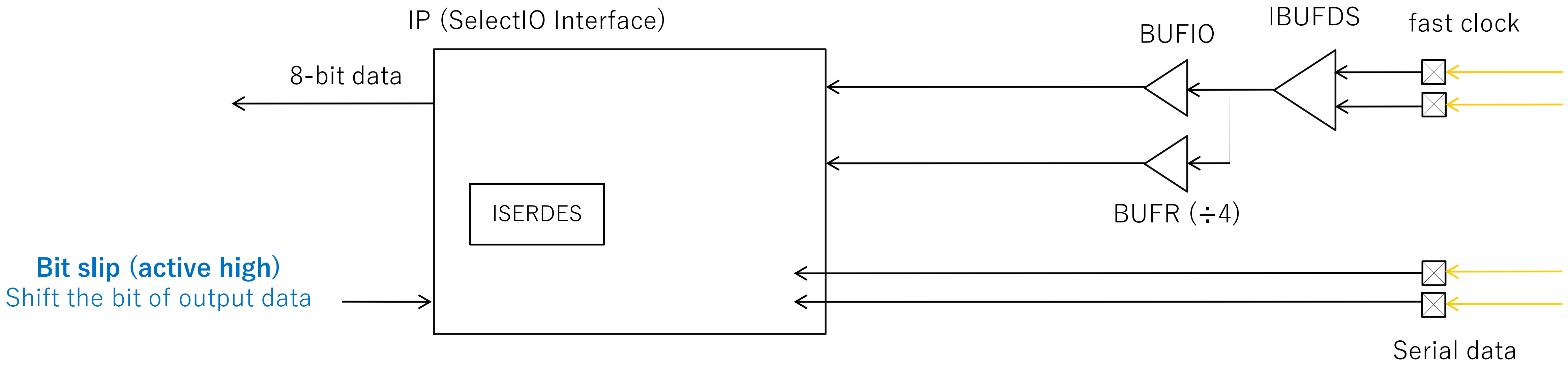
Data is broken since the phase of slow clock is not equal to the boundary of serial data.
Repeat bitslip to obtain the correct phase relation.

EX2

Exercise manual
EX2

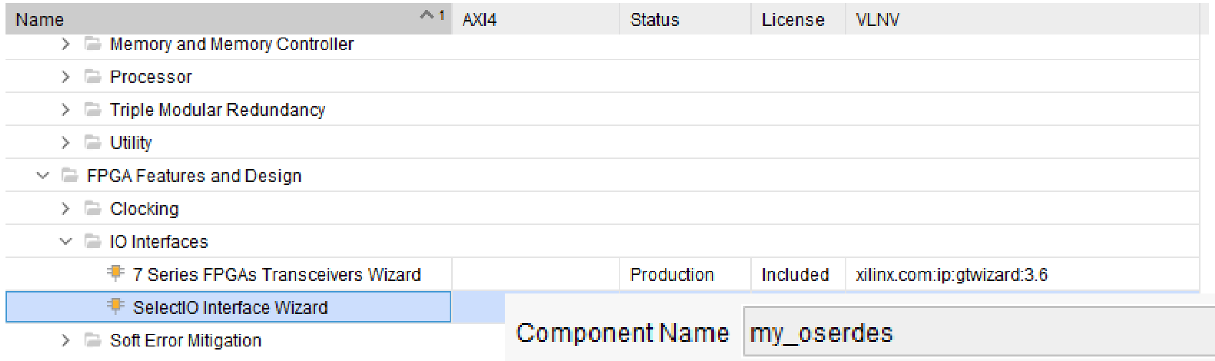
1. Output 8-bit fixed pattern data at 800 Mbps using OSERDES.
2. Make a loopback with a cable outside of the board.
3. Receive the data by ISERDES. Check if the received bit pattern is correct.



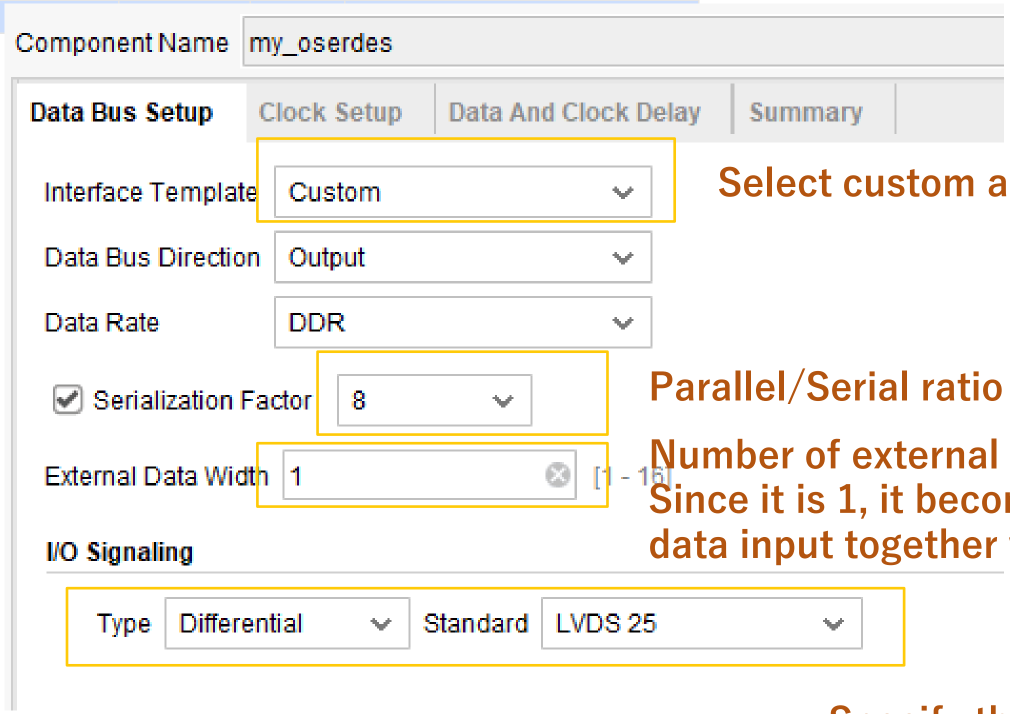


First, generate OSERDES

IP catalog



Name	AXI4	Status	License	VLNV
> Memory and Memory Controller				
> Processor				
> Triple Modular Redundancy				
> Utility				
⌄ FPGA Features and Design				
> Clocking				
⌄ IO Interfaces				
7 Series FPGAs Transceivers Wizard		Production	Included	xilinx.com:ip:gtwizard:3.6
SelectIO Interface Wizard				
> Soft Error Mitigation				



Component Name: my_oserdes

Data Bus Setup | Clock Setup | Data And Clock Delay | Summary

Interface Template: Custom

Data Bus Direction: Output

Data Rate: DDR

[✓] Serialization Factor: 8

External Data Width: 1 [1 - 16]

I/O Signaling

Type: Differential   Standard: LVDS 25

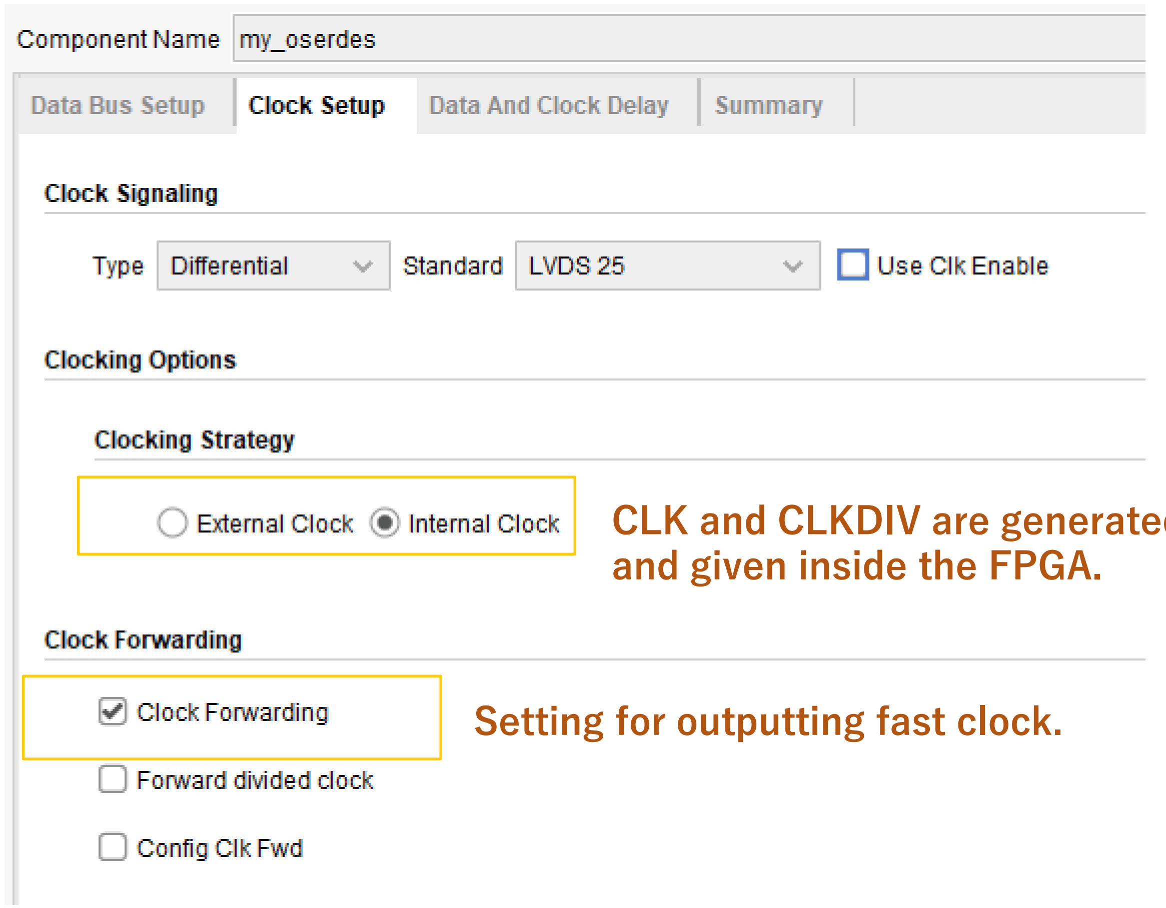
Select custom as it is self-made

Parallel/Serial ratio => 10:1

Number of external data ports.
Since it is 1, it becomes a 10-bit parallel data input together with the above.

Specify the signal standard

First, generate OSERDES



Component Name: my_oserdes

Data Bus Setup | **Clock Setup** | Data And Clock Delay | Summary

Clock Signaling

Type: Differential — Standard: LVDS 25 — ☐ Use Clk Enable

Clocking Options

Clocking Strategy

○ External Clock ● Internal Clock

CLK and CLKDIV are generated and given inside the FPGA.

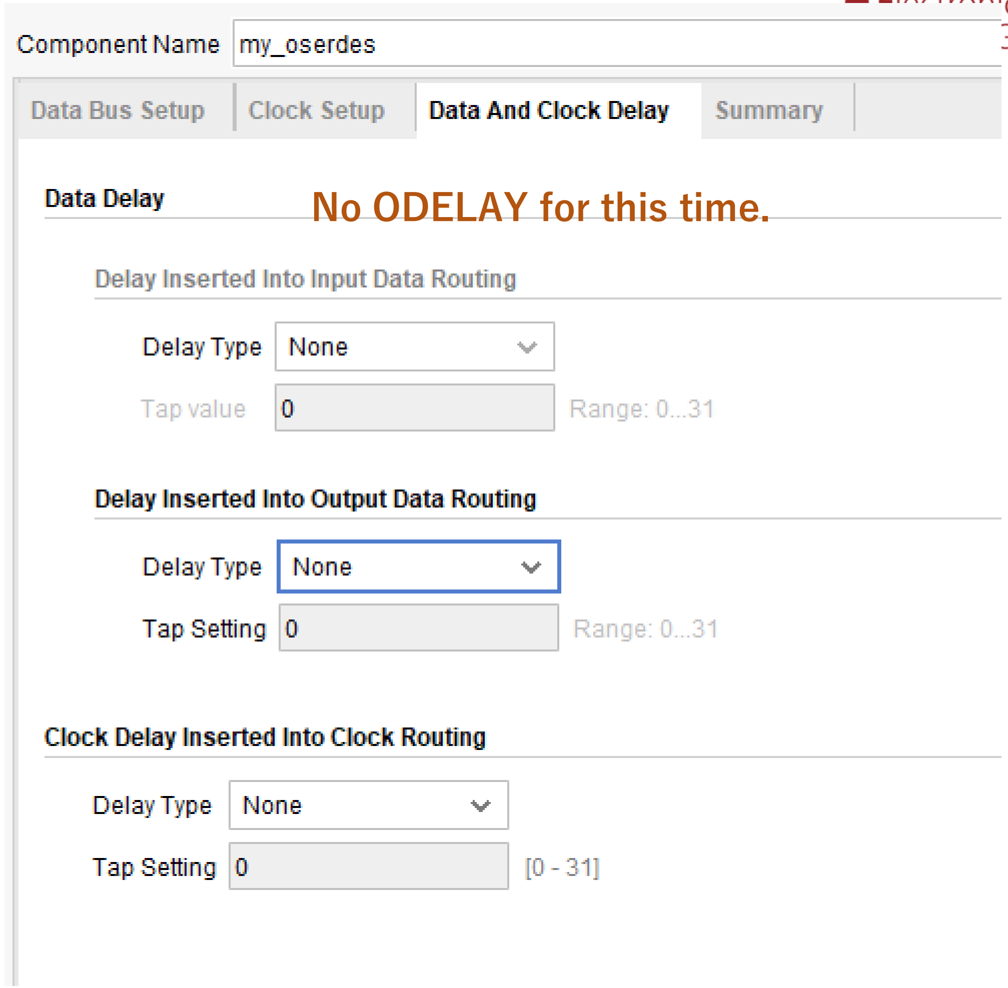
Clock Forwarding

☑ Clock Forwarding

☐ Forward divided clock

☐ Config Clk Fwd

Setting for outputting fast clock.



Component Name: my_oserdes

Data Bus Setup | Clock Setup | **Data And Clock Delay** | Summary

Data Delay

No ODELAY for this time.

Delay Inserted Into Input Data Routing

Delay Type: None

Tap value: 0 — Range: 0...31

Delay Inserted Into Output Data Routing

Delay Type: None

Tap Setting: 0 — Range: 0...31

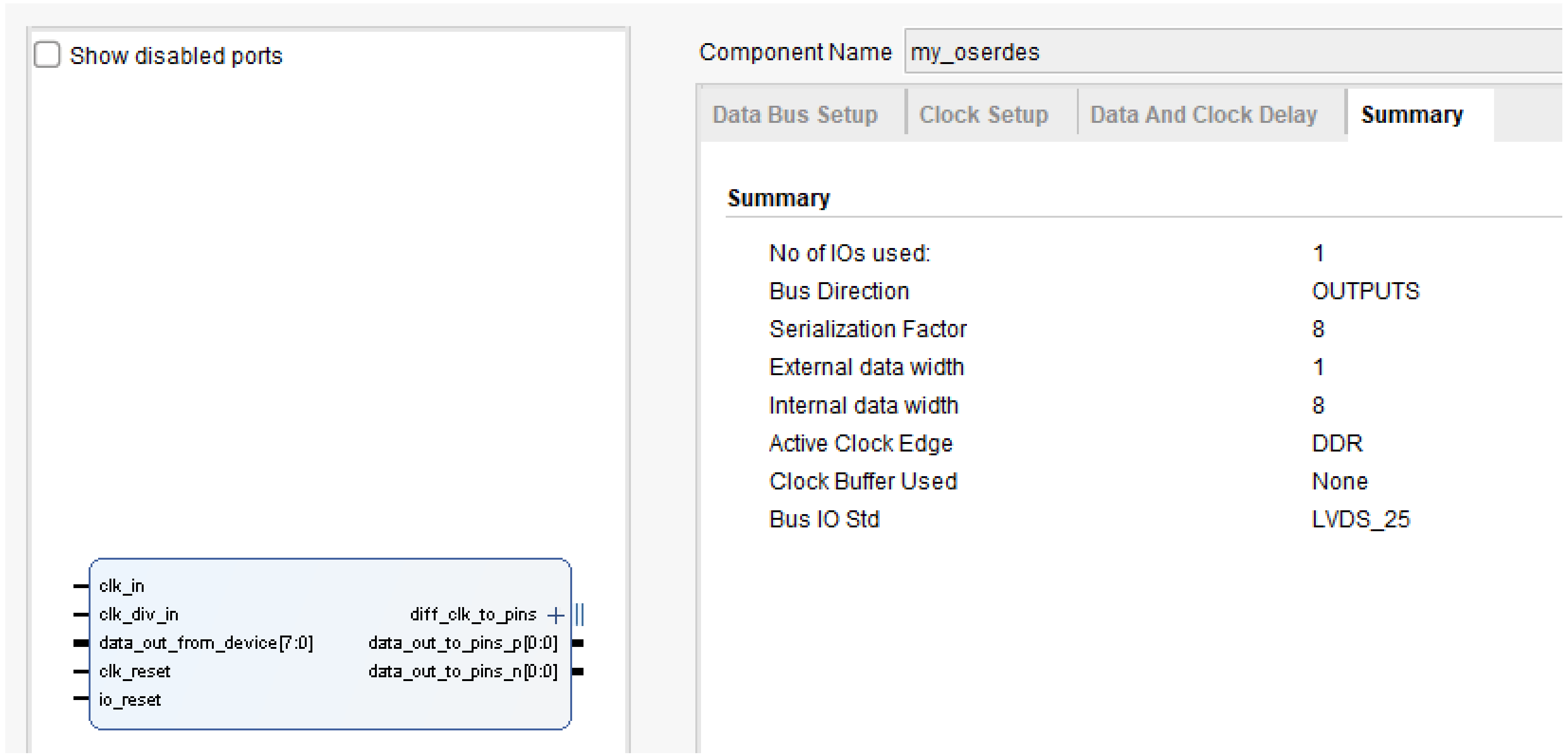
Clock Delay Inserted Into Clock Routing

Delay Type: None

Tap Setting: 0 — [0 - 31]

First, generate OSERDES

Like this?

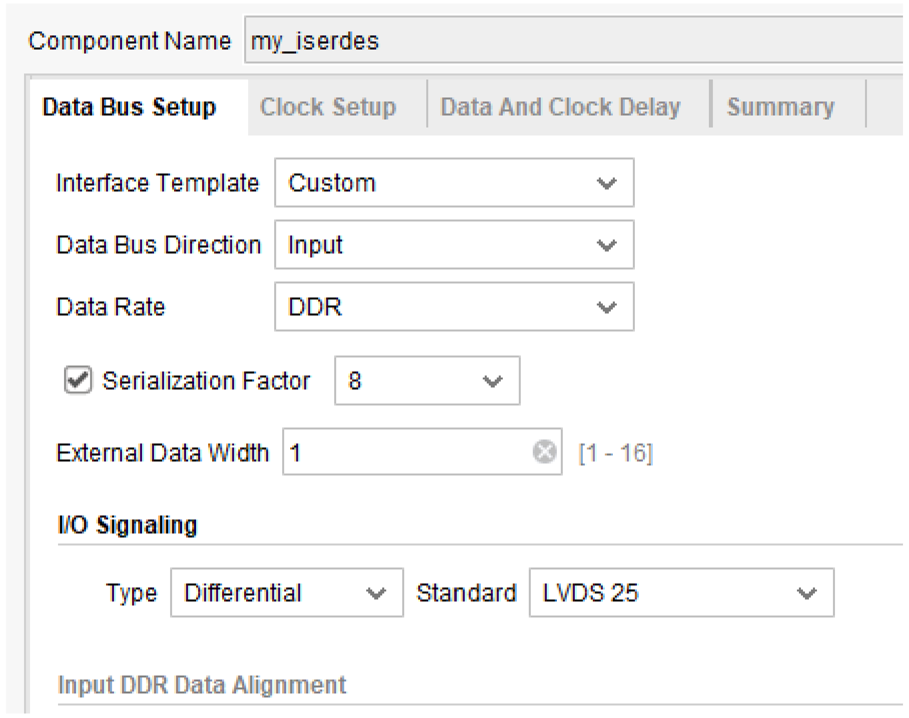


Component Name: my_oserdes

Summary

No of IOs used:	1
Bus Direction	OUTPUTS
Serialization Factor	8
External data width	1
Internal data width	8
Active Clock Edge	DDR
Clock Buffer Used	None
Bus IO Std	LVDS_25

Next, try to generate ISERDES without IDEALY



Component Name: my_iserdes

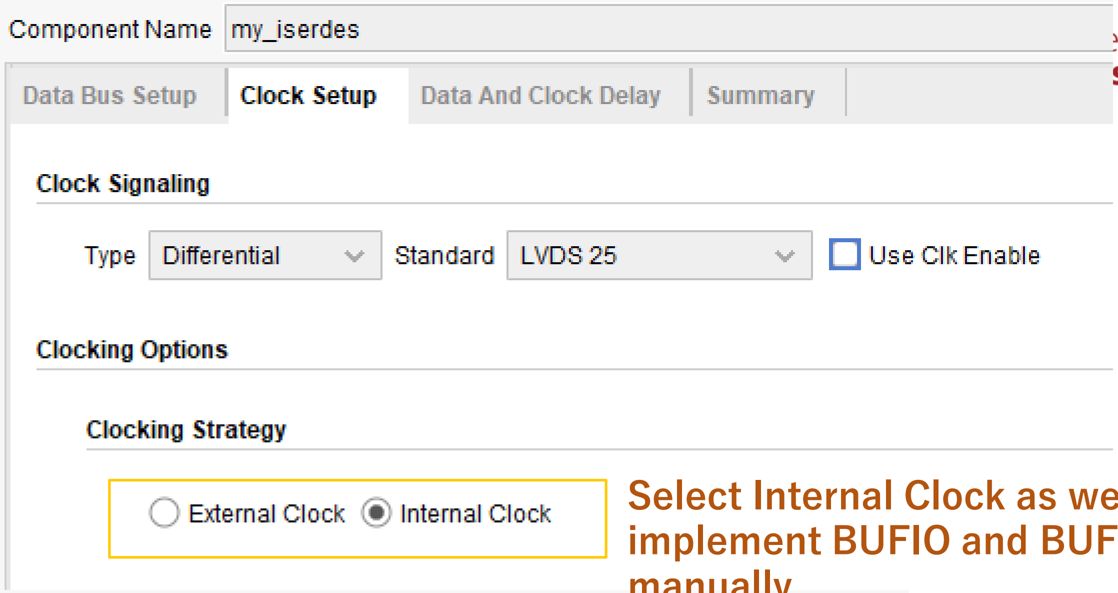
Data Bus Setup | Clock Setup | Data And Clock Delay | Summary

- Interface Template: Custom
- Data Bus Direction: Input
- Data Rate: DDR
- ✓ Serialization Factor: 8
- External Data Width: 1 [1 - 16]

I/O Signaling

- Type: Differential
- Standard: LVDS 25

Input DDR Data Alignment



Component Name: my_iserdes

Data Bus Setup | Clock Setup | Data And Clock Delay | Summary

Clock Signaling

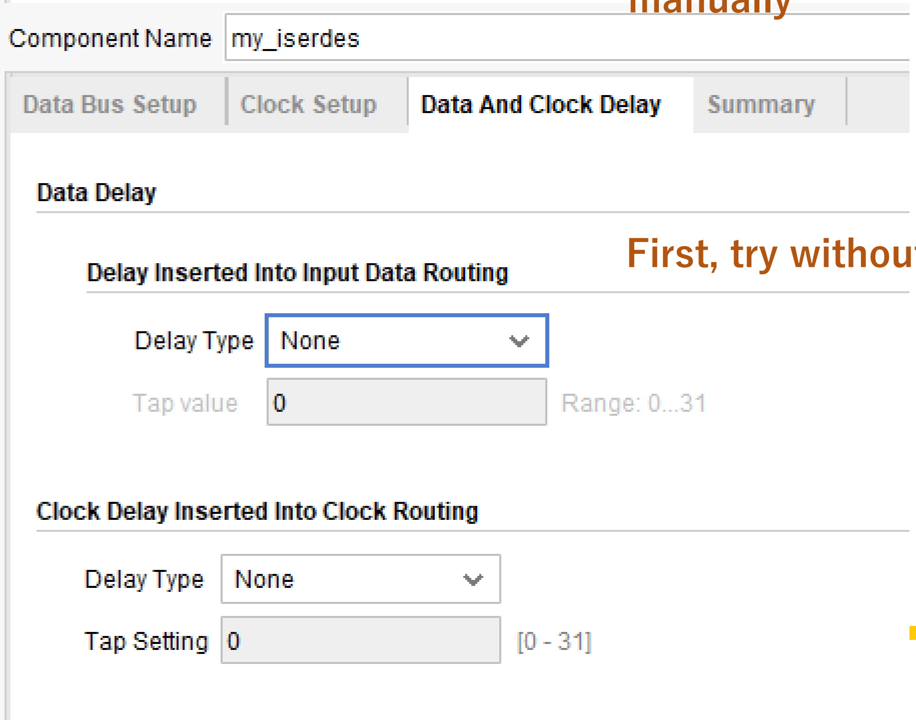
- Type: Differential
- Standard: LVDS 25
- ☐ Use Clk Enable

Clocking Options

Clocking Strategy

External Clock [✓ Internal Clock]

Select Internal Clock as we will implement BUFIO and BUFR manually



Component Name: my_iserdes

Data Bus Setup | Clock Setup | Data And Clock Delay | Summary

Data Delay

Delay Inserted Into Input Data Routing

- Delay Type: None
- Tap value: 0 Range: 0...31

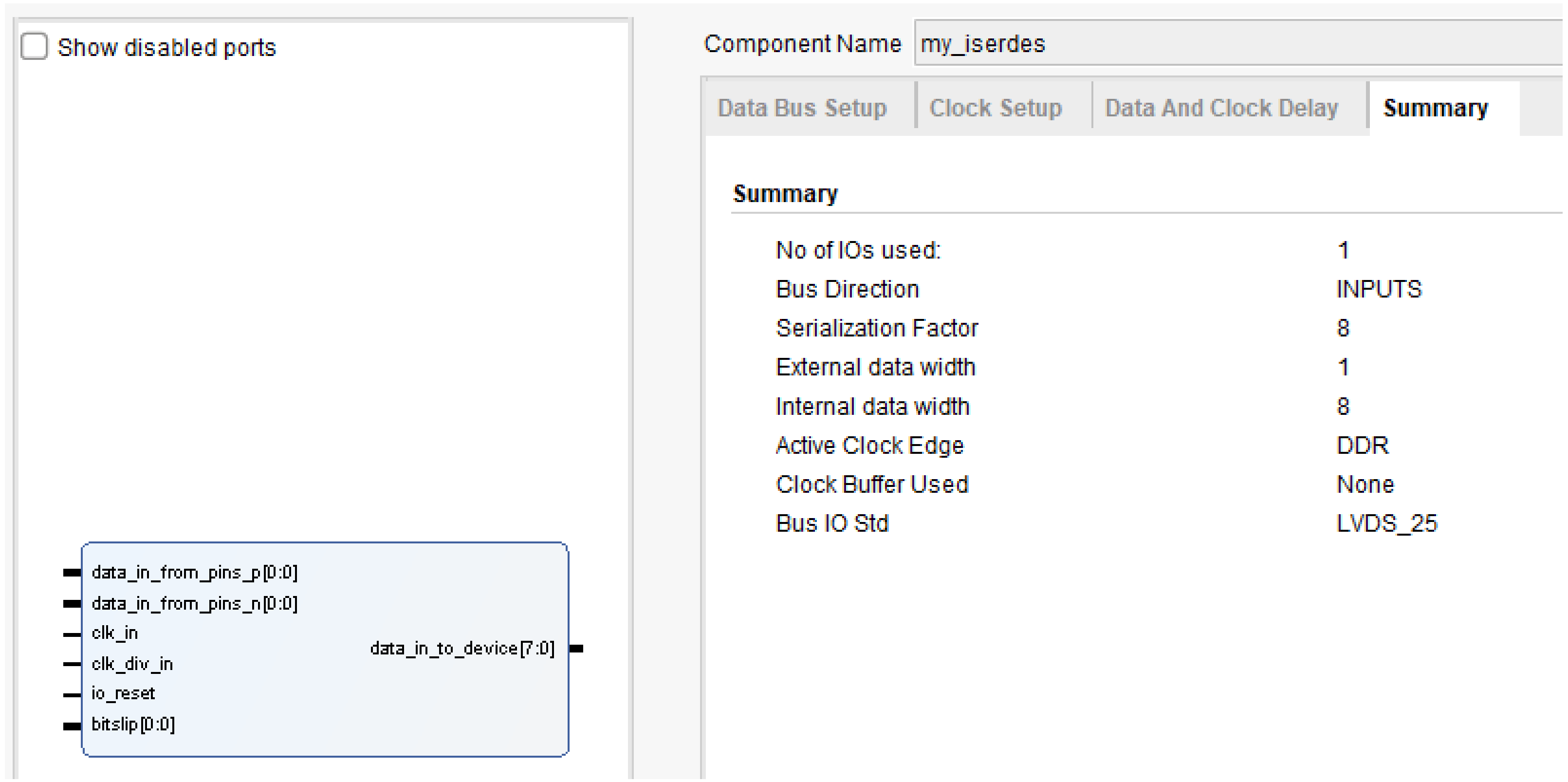
Clock Delay Inserted Into Clock Routing

- Delay Type: None
- Tap Setting: 0 [0 - 31]

First, try without IDELAY

Next, try to generate ISERDES without IDEALY

Like this?



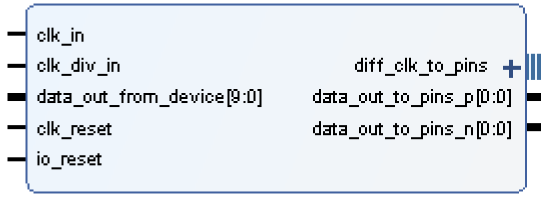
Component Name: my_iserdes

Summary

No of IOs used:	1
Bus Direction	INPUTS
Serialization Factor	8
External data width	1
Internal data width	8
Active Clock Edge	DDR
Clock Buffer Used	None
Bus IO Std	LVDS_25

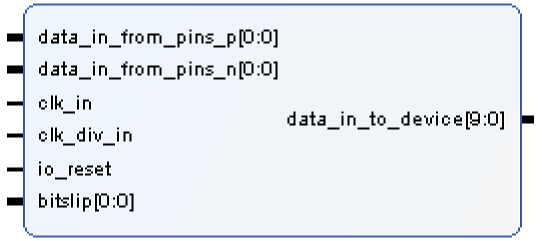
Wiring on HDL

- clk_in
  - 400 MHz (グローバルクロック)
- clk_div_in
  - 100 MHz (グローバルクロック)
- data_out_from_device
  - 8-bit data. Synchrous to 100 MHz clock.
  - **Constant: gives b1010’0011**
- clk_reset
  - Active high. Connect to the push button.
- io_reset
  - Active high. Connect to the push button.



- diff_clk_to_pins
  - Direct connect to top level port.
- data_out_to_pins_p/n
  - Direct connect to top level port.

- diff_clk_in
  - Direct connect to top level port.
- data_in_from_pins_p/n
  - Direct connect to top level port.
- clk_reset
  - open
- io_reset
  - Active high. Connect to the push button.
- bitslip
  - **Control with VIO probe_out.**
  - **VIO is driven by regional clock.**



- data_in_to_device
  - 8-bit data output.
  - **Connect to ILA.**
  - **Synchronized to regional clock.**

The ISERDES performs a bit shift every clock when bitslip is HIGH.
Make sure you get a one-shot pulse when the VIO button is pressed.

Is it done?

If a bit slip happens every time the push button is pressed, it is OK.

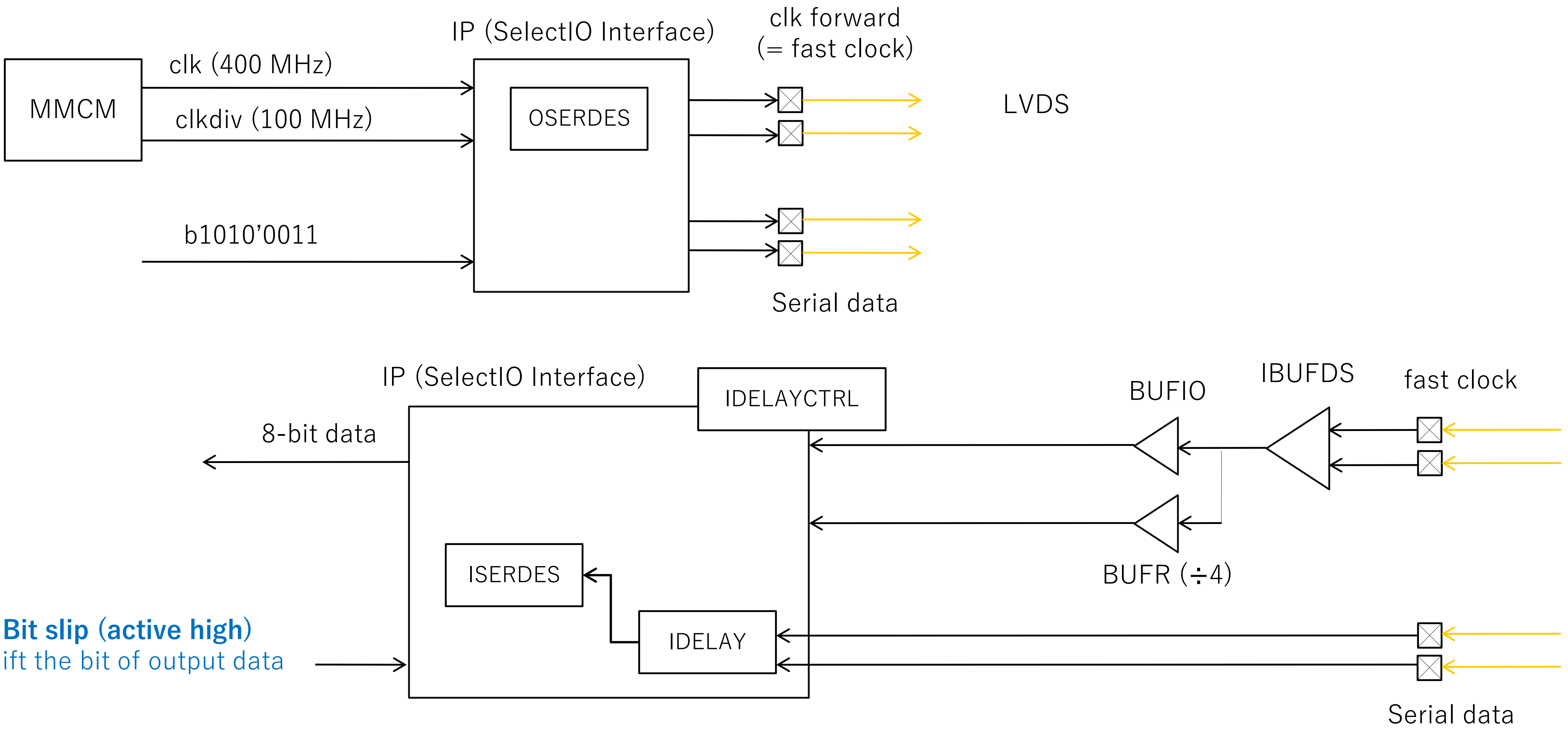
Let's keep doing bit shift until the received bit pattern matches the transmitted pattern.

Slides for advanced exercise
Skipped on the first day

Development topic H1

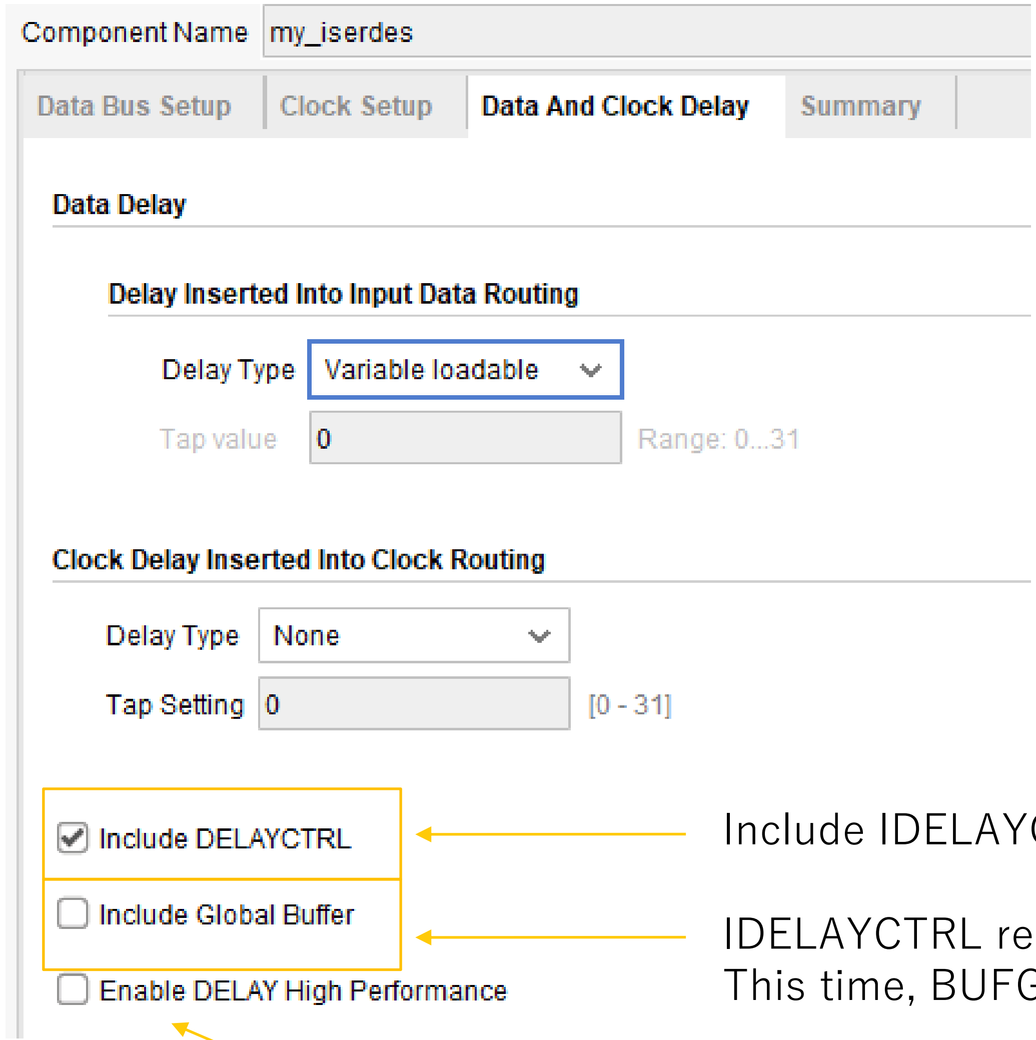
Exercise manual H1

1. Implement IDELAY before ISERDES



Bit slip (active high)
Shift the bit of output data

Implementing IDELAY



Component Name: my_iserdes

Data Delay

Delay Inserted Into Input Data Routing

Delay Type: Variable loadable

Tap value: 0 Range: 0...31

Clock Delay Inserted Into Clock Routing

Delay Type: None

Tap Setting: 0 [0 - 31]

- [✓] Include DELAYCTRL
- [ ] Include Global Buffer
- [ ] Enable DELAY High Performance

IDELAY can be adjusted in 32 steps (tap)

- Fixed: Gives a fixed delay.
- Variable: Raises or lowers by one using CE and INC signals.
- Variable loadable: Dynamically specify the Tap value.

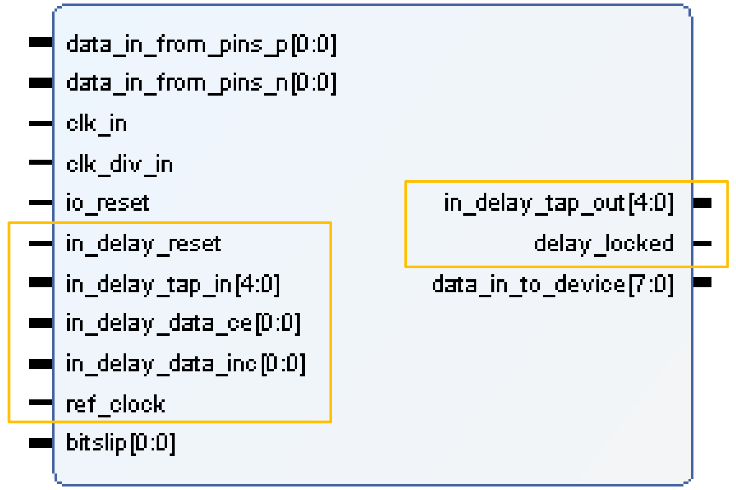
For this time, try with variable loadable.

Include IDELAYCTRL in the IP to calibrate IDELAY elements

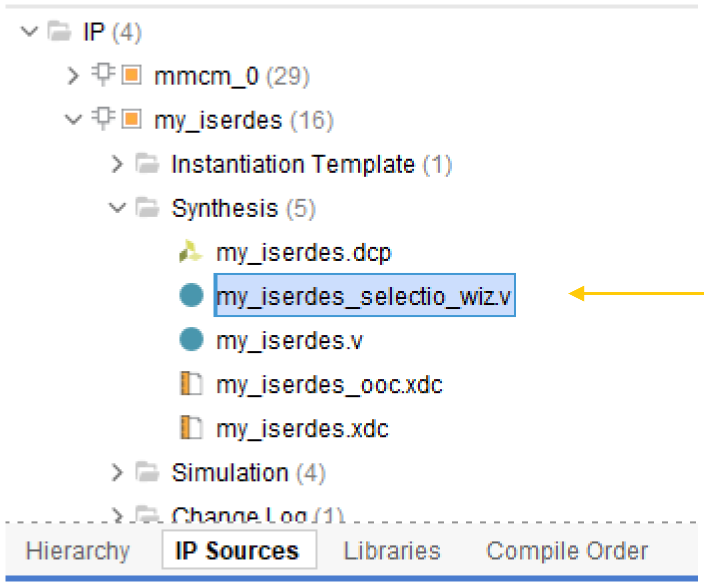
IDELAYCTRL reference clock must be driven by BUFG or BUFH.
This time, BUFG is specified on the MMCM side, so un-check it.

If this is specified, the jitter when passing through the IDELAY seems to be reduced.
It seems safer to turn ON when communicating close the limit speed.

Implementing IDELAY

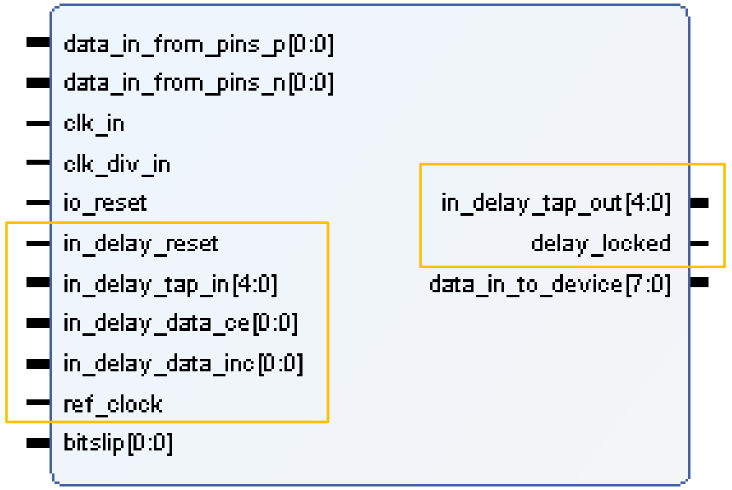


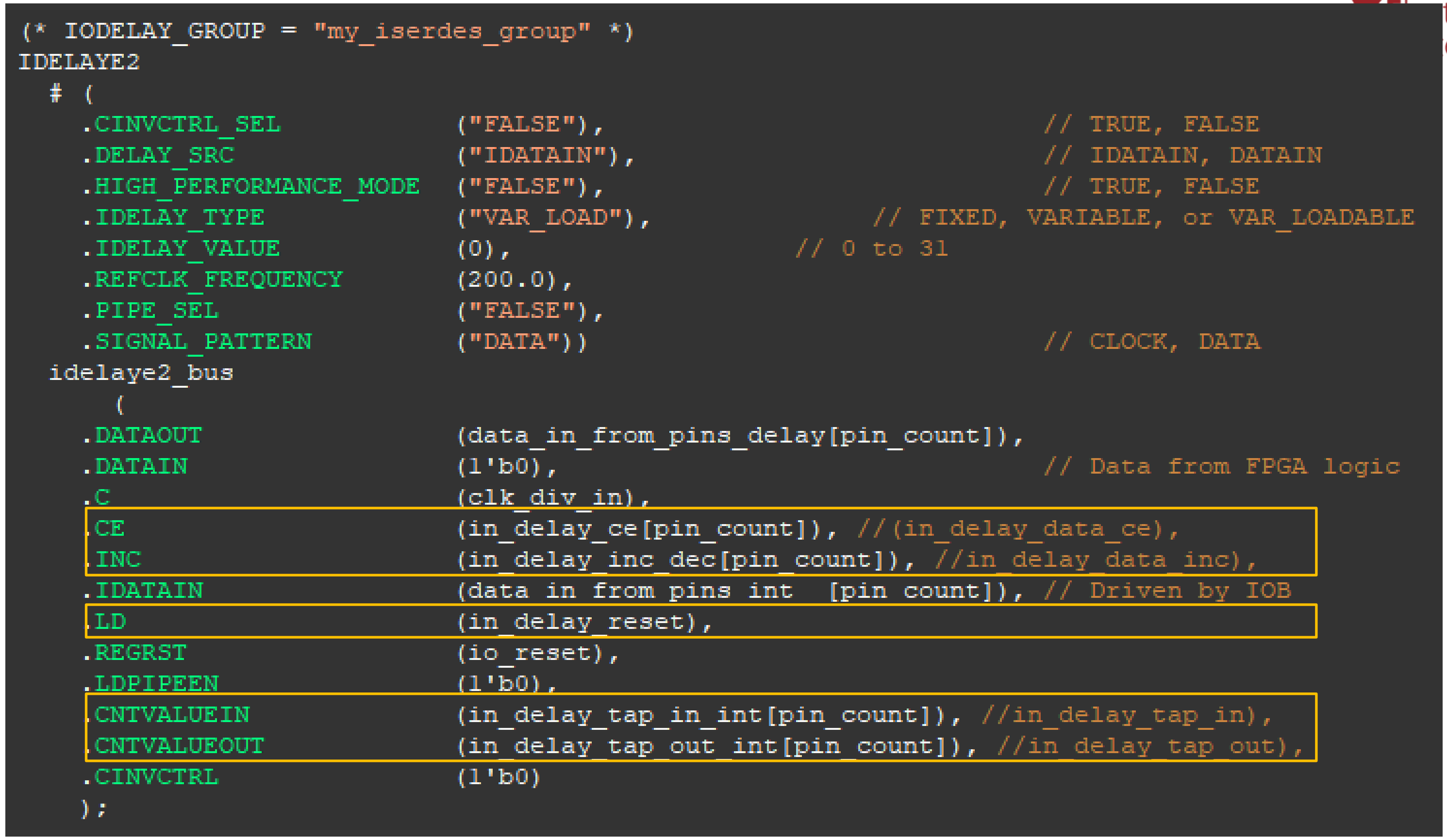
A new port has been added, but no idea about some of the functions.
Since there is an IP, read the generated HDL code and find out where it connects to.



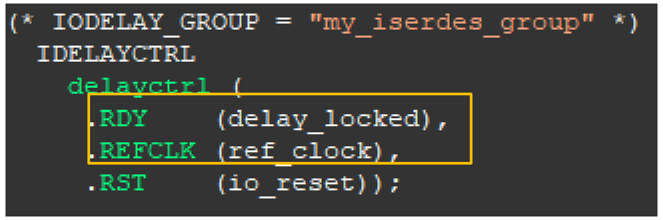
Code generated by the wizard for implementation.
You can think of the IP Catalog as providing examples of different dedicated block implementations.

Implementing IDELAY



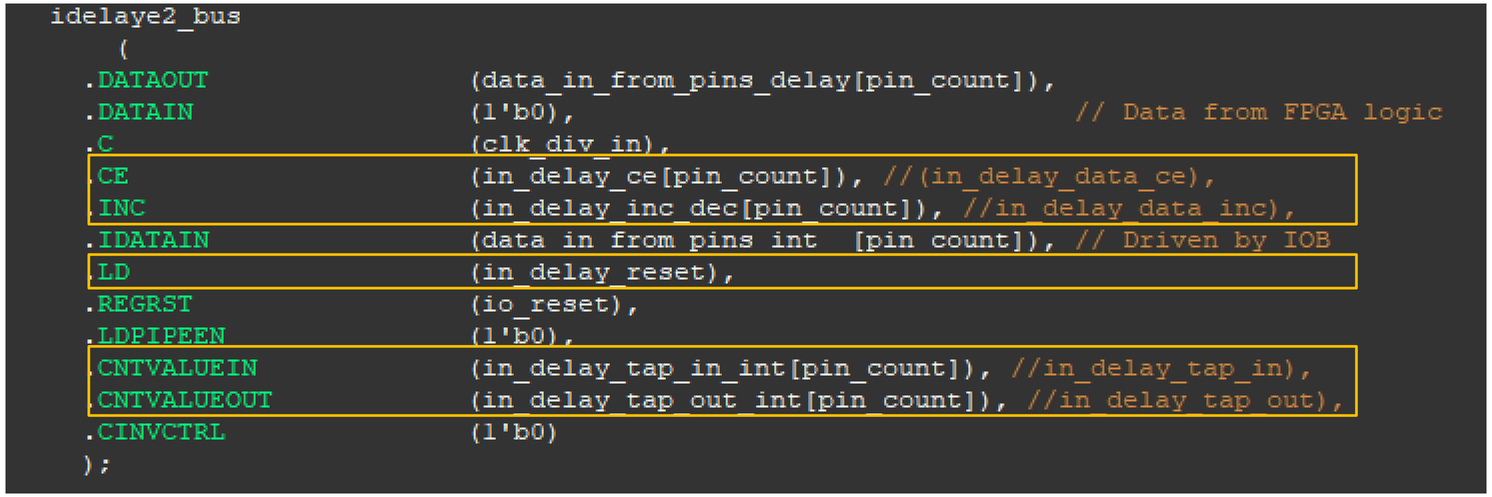


```
(* IODELAY_GROUP = "my_iserdes_group" *)
IDELAYE2
  # (
    .CINVCTRL_SEL           ("FALSE"),                          // TRUE, FALSE
    .DELAY_SRC              ("IDATAIN"),                        // IDATAIN, DATAIN
    .HIGH_PERFORMANCE_MODE  ("FALSE"),                          // TRUE, FALSE
    .IDELAY_TYPE            ("VAR_LOAD"),            // FIXED, VARIABLE, or VAR_LOADABLE
    .IDELAY_VALUE           (0),                // 0 to 31
    .REFCLK_FREQUENCY       (200.0),
    .PIPE_SEL               ("FALSE"),
    .SIGNAL_PATTERN         ("DATA"))                           // CLOCK, DATA
  idelaye2_bus
      (
    .DATAOUT                (data_in_from_pins_delay[pin_count]),
    .DATAIN                 (1'b0),                             // Data from FPGA logic
    .C                      (clk_div_in),
    .CE                     (in_delay_ce[pin_count]), //(in_delay_data_ce),
    .INC                    (in_delay_inc_dec[pin_count]), //in_delay_data_inc),
    .IDATAIN                (data_in_from_pins_int  [pin_count]), // Driven by IOB
    .LD                     (in_delay_reset),
    .REGRST                 (io_reset),
    .LDPIPEEN               (1'b0),
    .CNTVALUEIN             (in_delay_tap_in_int[pin_count]), //in_delay_tap_in),
    .CNTVALUEOUT            (in_delay_tap_out_int[pin_count]), //in_delay_tap_out),
    .CINVCTRL               (1'b0)
    );
```



```
(* IODELAY_GROUP = "my_iserdes_group" *)
 IDELAYCTRL
   delayctrl (
    .RDY     (delay_locked),
    .REFCLK  (ref_clock),
    .RST     (io_reset));
```

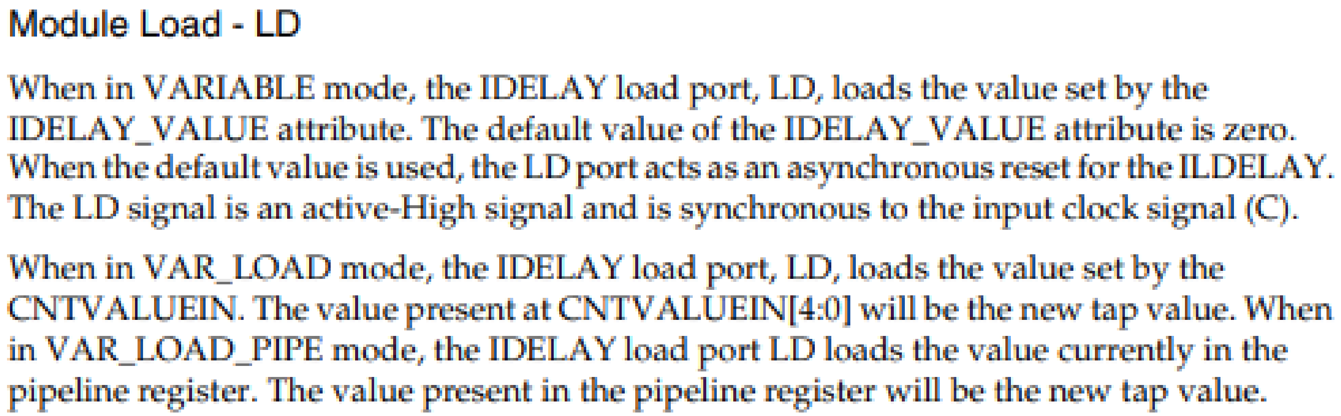
Implementing IDELAY



```
idelaye2_bus
    (
  .DATAOUT                  (data_in_from_pins_delay[pin_count]),
  .DATAIN                   (1'b0),                                 // Data from FPGA logic
  .C                        (clk_div_in),
  .CE                       (in_delay_ce[pin_count]), //(in_delay_data_ce),
  .INC                      (in_delay_inc_dec[pin_count]), //in_delay_data_inc),
  .IDATAIN                  (data_in_from_pins_int  [pin_count]), // Driven by IOB
  .LD                       (in_delay_reset),
  .REGRST                   (io_reset),
  .LDPIPEEN                 (1'b0),
  .CNTVALUEIN               (in_delay_tap_in_int[pin_count]), //in_delay_tap_in),
  .CNTVALUEOUT              (in_delay_tap_out_int[pin_count]), //in_delay_tap_out),
  .CINVCTRL                 (1'b0)
  );
```

CE and INC are clock (C) synchronous inputs.

- If CE is HIGH and INC is HIGH, the tap advances by one.
- If INC is LOW when CE is HIGH, one tap is returned.



Module Load - LD

When in VARIABLE mode, the IDELAY load port, LD, loads the value set by the IDELAY_VALUE attribute. The default value of the IDELAY_VALUE attribute is zero. When the default value is used, the LD port acts as an asynchronous reset for the ILDELAY. The LD signal is an active-High signal and is synchronous to the input clock signal (C).

When in VAR_LOAD mode, the IDELAY load port, LD, loads the value set by the CNTVALUEIN. The value present at CNTVALUEIN[4:0] will be the new tap value. When in VAR_LOAD_PIPE mode, the IDELAY load port LD loads the value currently in the pipeline register. The value present in the pipeline register will be the new tap value.

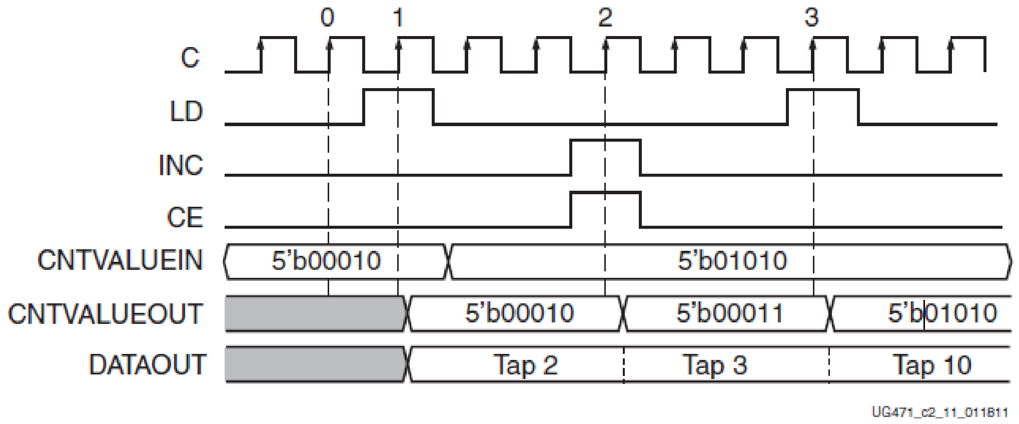
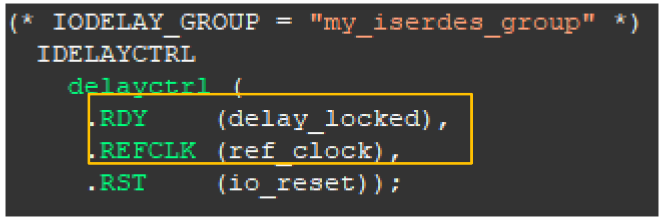


Figure 2-13: **IDELAY in VAR_LOAD Timing Diagram**

UG471

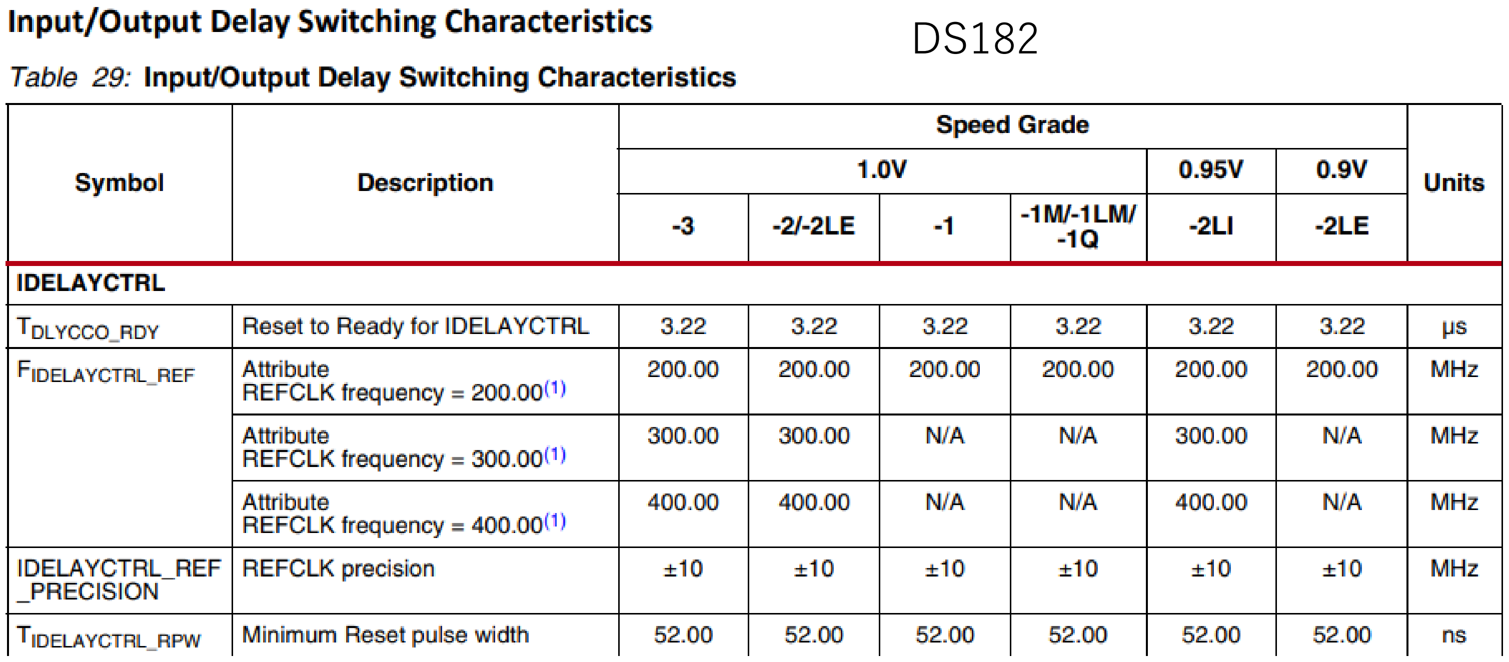
Implementing IDELAY



```
(* IODELAY_GROUP = "my_iserdes_group" *)
  IDELAYCTRL
    delayctrl (
      .RDY      (delay_locked),
      .REFCLK  (ref_clock),
      .RST      (io_reset));
```

- RDY: Indicates that the calibration is done.
- REFCLK: Reference clock for calibration.
  - ⇒No idea about the the frequency to input

DS182



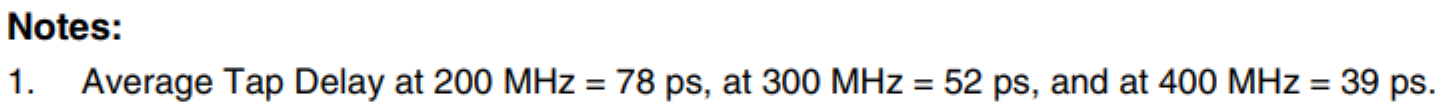
Input/Output Delay Switching Characteristics

Table 29: **Input/Output Delay Switching Characteristics**

Symbol	Description	Speed Grade						Units
		1.0V				0.95V	0.9V	
		-3	-2/-2LE	-1	-1M/-1LM/-1Q	-2LI	-2LE	
IDELAYCTRL								
T_{DLYCCO_RDY}	Reset to Ready for IDELAYCTRL	3.22	3.22	3.22	3.22	3.22	3.22	µs
$F_{IDELAYCTRL\_REF}$	Attribute REFCLK frequency = 200.00[1]	200.00	200.00	200.00	200.00	200.00	200.00	MHz
	Attribute REFCLK frequency = 300.00[1]	300.00	300.00	N/A	N/A	300.00	N/A	MHz
	Attribute REFCLK frequency = 400.00[1]	400.00	400.00	N/A	N/A	400.00	N/A	MHz
IDELAYCTRL_REF_PRECISION	REFCLK precision	±10	±10	±10	±10	±10	±10	MHz
$T_{IDELAYCTRL\_RPW}$	Minimum Reset pulse width	52.00	52.00	52.00	52.00	52.00	52.00	ns

The operation frequency depends on the speed grade!

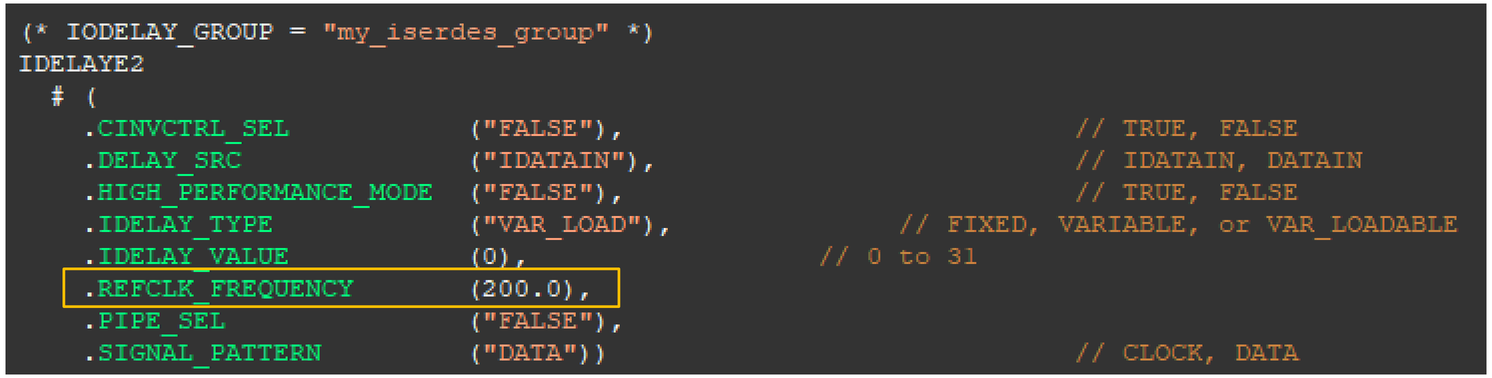
Please be careful when porting source code.



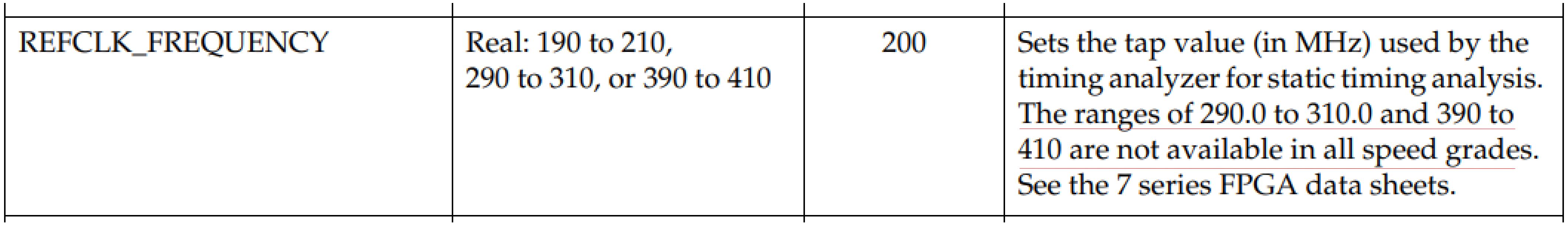
Notes:

1. Average Tap Delay at 200 MHz = 78 ps, at 300 MHz = 52 ps, and at 400 MHz = 39 ps.

Digression



```
(* IODELAY_GROUP = "my_iserdes_group" *)
IDELAYE2
  # (
    .CINVCTRL_SEL           ("FALSE"),                          // TRUE, FALSE
    .DELAY_SRC              ("IDATAIN"),                        // IDATAIN, DATAIN
    .HIGH_PERFORMANCE_MODE  ("FALSE"),                          // TRUE, FALSE
    .IDELAY_TYPE            ("VAR_LOAD"),             // FIXED, VARIABLE, or VAR_LOADABLE
    .IDELAY_VALUE           (0),             // 0 to 31
    .REFCLK_FREQUENCY       (200.0),
    .PIPE_SEL               ("FALSE"),
    .SIGNAL_PATTERN         ("DATA"))                           // CLOCK, DATA
```

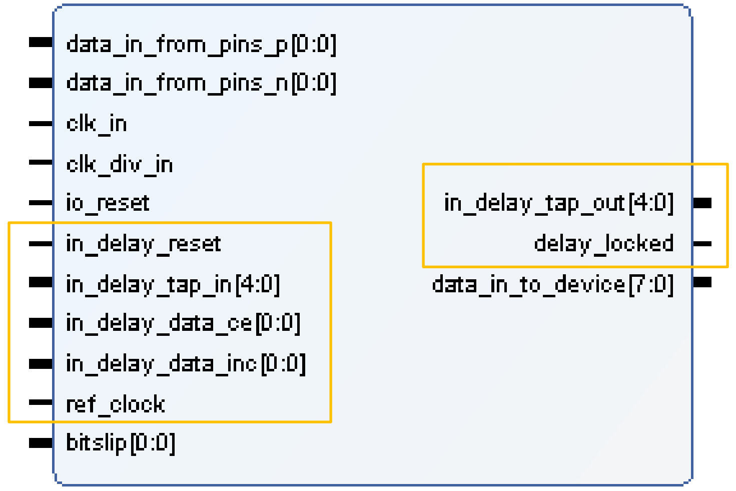


REFCLK_FREQUENCY	Real: 190 to 210, 290 to 310, or 390 to 410	200	Sets the tap value (in MHz) used by the timing analyzer for static timing analysis. The ranges of 290.0 to 310.0 and 390 to 410 are not available in all speed grades. See the 7 series FPGA data sheets.

UG471

Seems to be a contradiction here.
I imagine that this attribute value is used for static timing analysis, so even if you write 300 or 400, it will not be reflected in the analysis result.

Implementing IDELAY



In summary, it seems better to do the following for this example:

- in_delay_reset
  - Use VIO's probe_out to input a one-shot pulse.
  - Let's leave bitslip functionality.
- in_delay_tap_in
  - Given by VIO
- in_delay_data_ce/inc
  - 利用しない。
- ref_clock
  - 200 MHz clock passing through BUFG
- in_delay_tap_out
  - Connect to ILA
- delay_locked
  - Open

Let's modify the previous source codes and implement it.

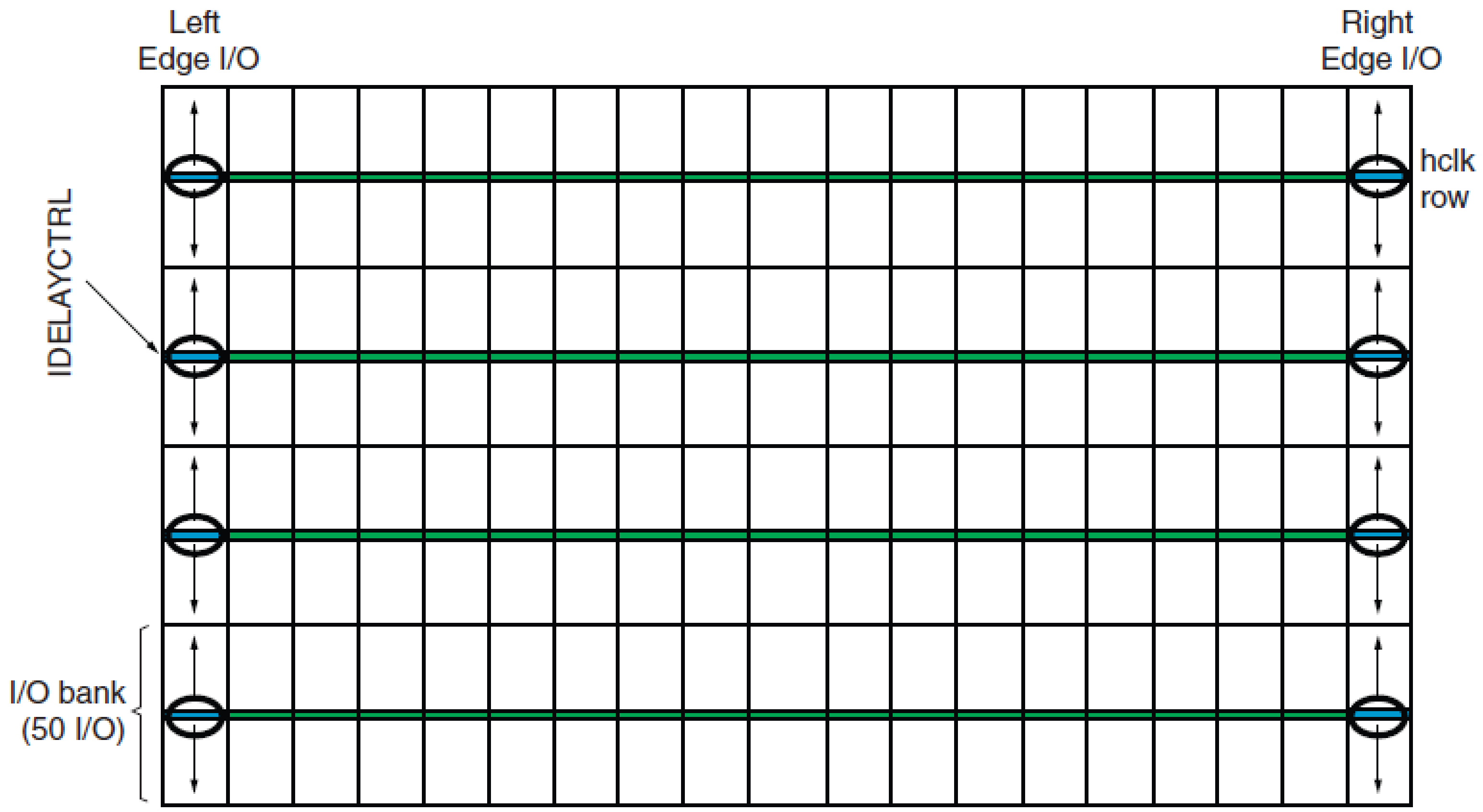
Is it done?

At 800 Mbps, it’s 1.25 ns per 1-bit.
IDELAY is 78 ps/tap and 32 taps, so 2.5 ns.
In other words, out of 32 taps, the timing will be once wrong.
(Actually, the tap range that seems to be incorrect)

When happened when trying it?

IDELAYCTRL

IDELAYCTRL exists in one IO column of every bank, and it calibrates IDELAY and ODELAY.



ug471_c2_14_021914

Figure 2-16: **Relative Locations of IDELAYCTRL Modules**

IDELAYCTRL

Use the IP which was made earlier to issue an intentionally error.

- Implement another my_iserdes and wire it to the IO pin of the same bank.

Point

- This IP has built-in IDELAYCTRL which is the only one in each bank.

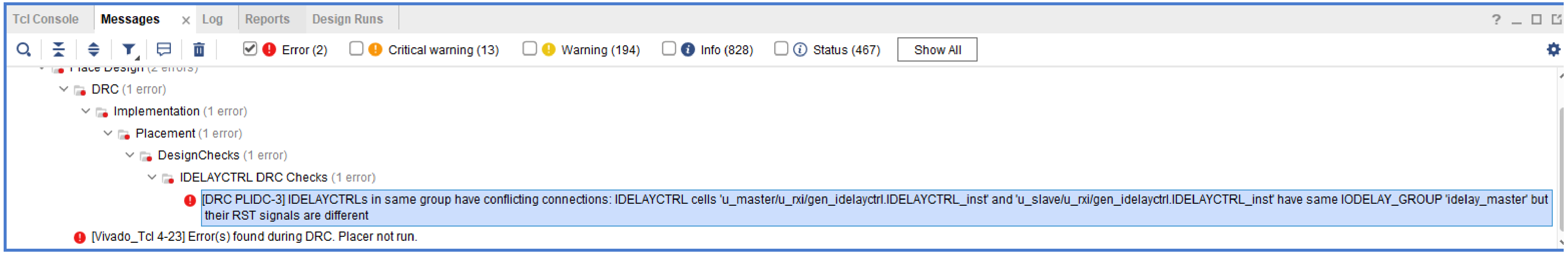
Try this myself.

IDELAYCTRL

An IDELAYCTRL of unknown usage was created in an unused IO column, but it was not an error.
(There might be an error if the unused area decreases.)

There must be an error this time.

- Make io_reset on my_iserdes1 a separate signal from my_iserdes0.



```
Place Design (2 errors)
  DRC (1 error)
    Implementation (1 error)
      Placement (1 error)
        DesignChecks (1 error)
          IDELAYCTRL DRC Checks (1 error)
            [DRC PLIDC-3] IDELAYCTRLs in same group have conflicting connections: IDELAYCTRL cells 'u_master/u_rxi/gen_idelayctrl.IDELAYCTRL_inst' and 'u_slave/u_rxi/gen_idelayctrl.IDELAYCTRL_inst' have same IODELAY_GROUP 'idelay_master' but their RST signals are different
  [Vivado_Tcl 4-23] Error(s) found during DRC. Placer not run.
```

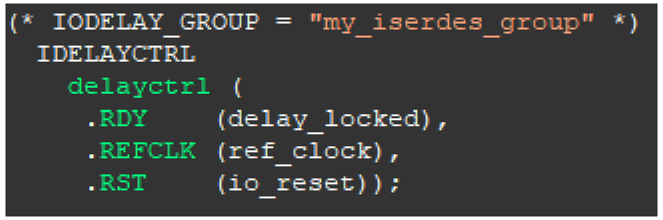
Should I remove IDELAYCTRL from IP?
Answer: Then, there is potential trouble.

The IODELAY_GROUP constraint is the key to this trouble.

IODELAY_GROUP

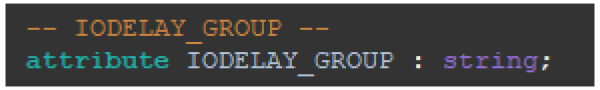
IODELAY_GROUP is a constraint to associate IDELAYCTRL with IDELAY and ODELAY elements, and it gives calibration correspondence.

Verilog-HDL

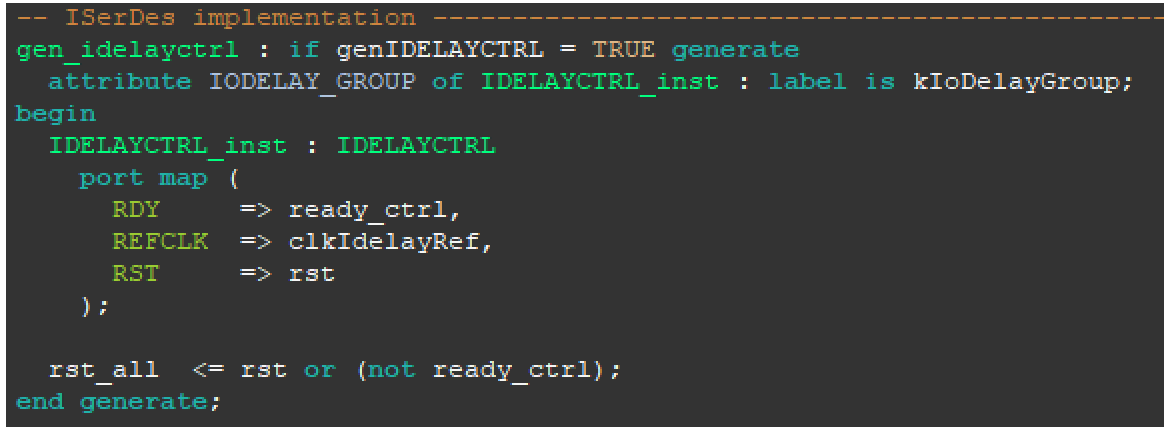


```verilog
(* IODELAY_GROUP = "my_iserdes_group" *)
  IDELAYCTRL
    delayctrl (
     .RDY     (delay_locked),
     .REFCLK (ref_clock),
     .RST     (io_reset));
```

VHDL



```vhdl
 -- IODELAY_GROUP --
 attribute IODELAY_GROUP : string;
```



```vhdl
-- ISerDes implementation ---------------------------------------------------
gen_idelayctrl : if genIDELAYCTRL = TRUE generate
  attribute IODELAY_GROUP of IDELAYCTRL_inst : label is kIoDelayGroup;
begin
  IDELAYCTRL_inst : IDELAYCTRL
    port map (
      RDY      => ready_ctrl,
      REFCLK   => clkIdelayRef,
      RST      => rst
    );

  rst_all   <= rst or (not ready_ctrl);
end generate;
```

Since it is a constraint providing correspondence relationship, it is necessary to name the group correctly.
But once it is generated by IP, it cannot be changed later.

Digression
I end up constraining things in HDL instead of XDC where in-code constraints are possible.
It's a bad design for reuse because it ties up the device and the code.

IODELAY_GROUP

- If there are multiple IDELAYCTRL-IODELAY combinations with the same group name and seem to have the same functionality, you can duplicate the IDELAYCTRL, and Vivado will implement it as the developer intended.
  - As seen earlier, even if you try to force to implement two IDELAYCTRLs in the same bank, it will do something.
  - However, there is the risk of potential errors.

- By changing the RST signal by an intentional error, it made it look like that it was not the same function IDELAYCTRL and prevented Vivado from handling it.

Perfect solution

- The programmer must write IODELAY_GROUP correctly.
- If IP is used, it will be the same IODELAY_GROUP unless you change the name to another IP.
  - For the firmware which only receives ADC data on SERDES, you can expect Vivado to replicate IDELAYCTRL.
  - For complex firmware which communicates with multiple external devices using IOSERDES, manual implementation while referring to IP HDL code is more flexible.