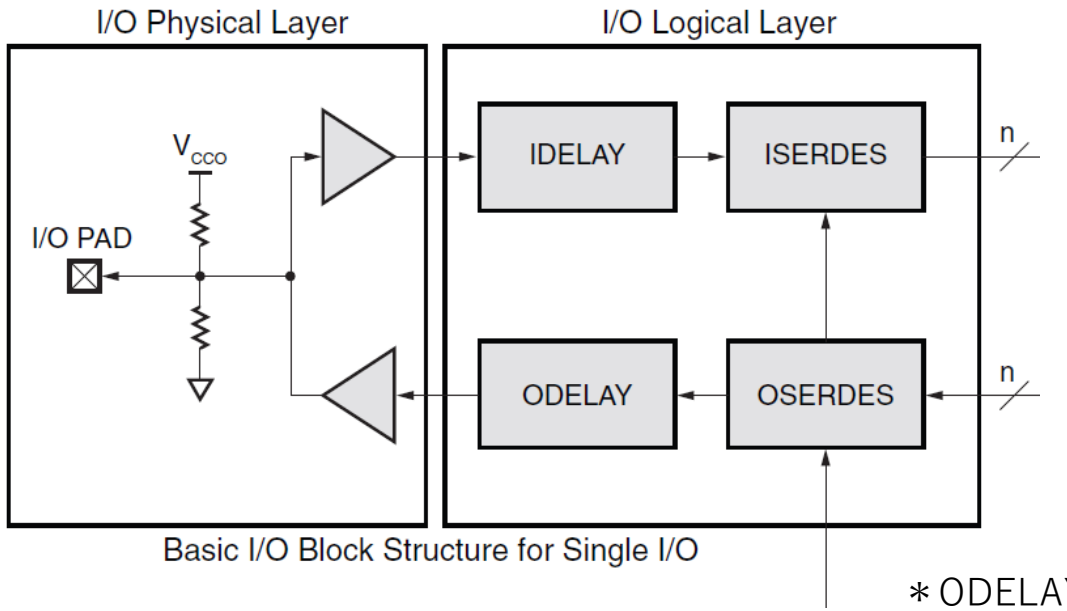
IOSERDES

KEK IPNS E-sys
本多良太郎

おさらい

ILOGIC（リソース）
De-serializer, ISERDES (Primitive)
- 1本の信号線から複数ビットのデータを取り出す (Data In)

Physical layer
- 入出力の物理的な性質を決定する
  - 終端抵抗（入力）
  - 入力同相電位
  - 出力電位
  - etc...



Basic I/O Block Structure for Single I/O

Logical layerを使用しない場合IOBから直接ファブリックへ信号線が繋がる

＊ODELAYはHPバンクにのみ存在

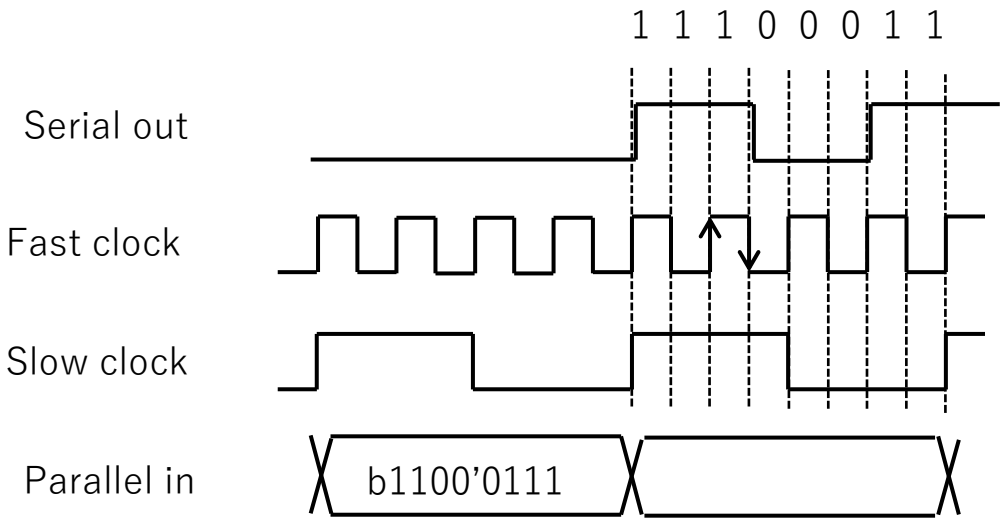
Single data rate (SDR)
- クロックエッジの立ち上がり (立下り)だけを利用する

Double data rate (DDR)
- クロックの両エッジを使う

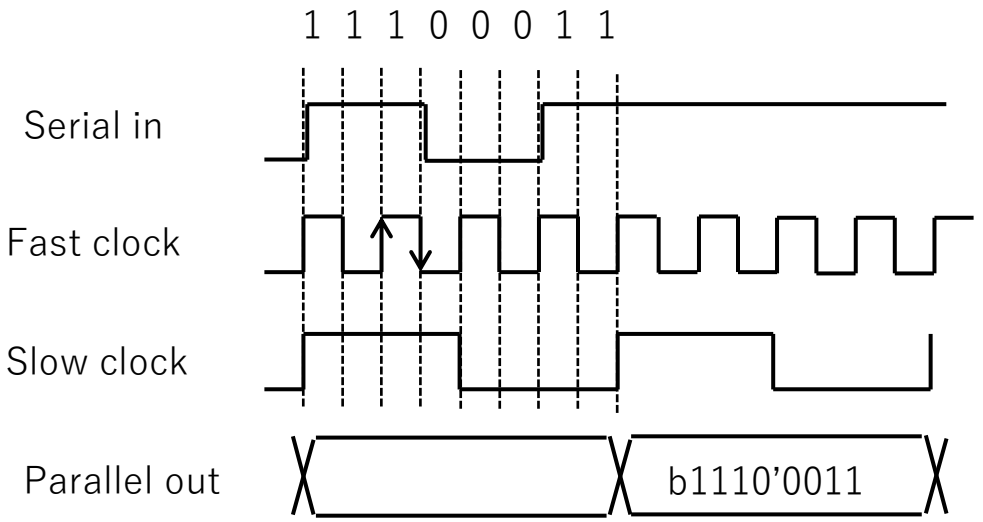
OLOGIC（リソース）
Serializer, OSERDES (Primitive)
- 複数ビットのデータを1ビット列の信号に変換する (Data Out)

IOタイル

Parallel-to-Serial (OSERDES)



Serial-to-Parallel (ISERDES)



この章で目指すスキル

実際の回路基板を使いつつ進めます。

IOSERDES
- OSERDESを実装して10-bitのデータを200 MHzのDDR (400 Mbps)で送信する。
  - 送信するのは簡単。
- ISERDESで送信したデータを受信して10-bitのデータを復元する。
  - BUFIOとBUFRの使い方を学ぶ。
  - データに対してslow clockが位相差を持ってしまうのでビットスリップでデータシフトをする。

IDELAY
- データとfast clock間のタイミングがいつも丁度いいとは限らない。
- データ側に数十psの遅延を与えて調整するブロック。
- IDELAYを使って遅延を与えた時に何が起きるのかを自分の目で確かめる。

IDELAYCTRL
- 初心者殺しなのではまるポイントを学ぶ。

OSERDESで送ってISERDESで受け取る

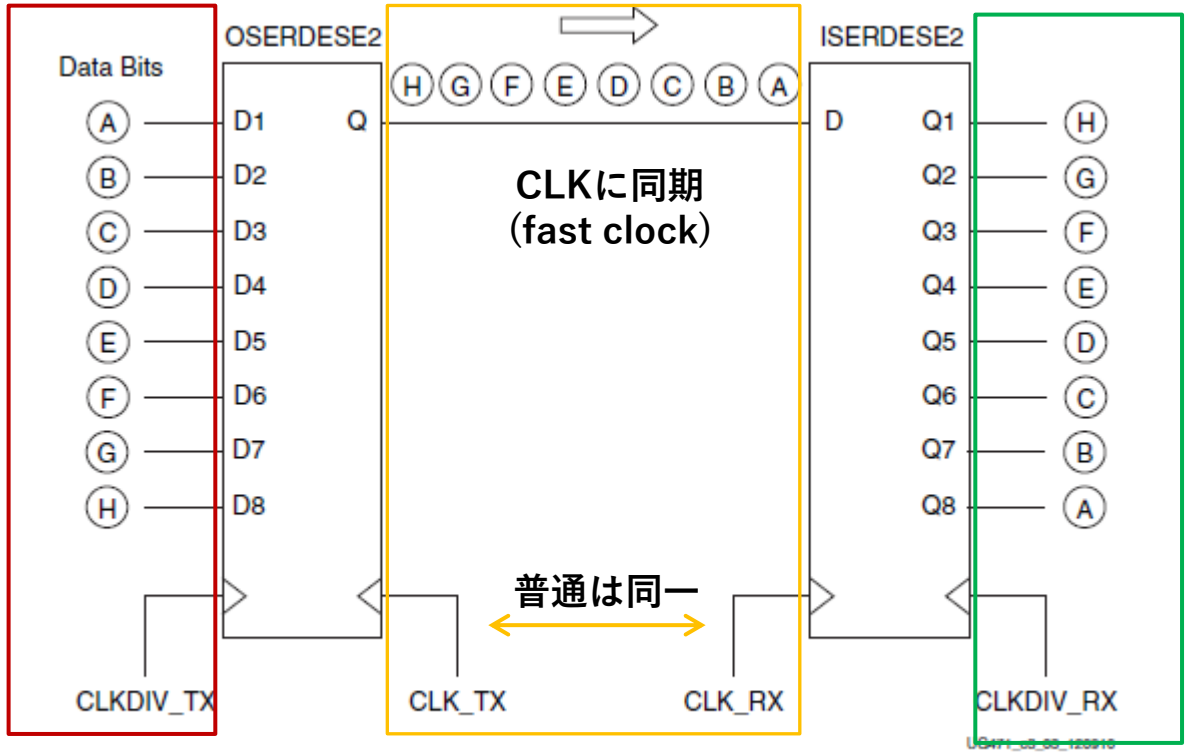


図 3-3 : ISERDESE2 ポートの Q1−Q8 出力のビット順序

UG471

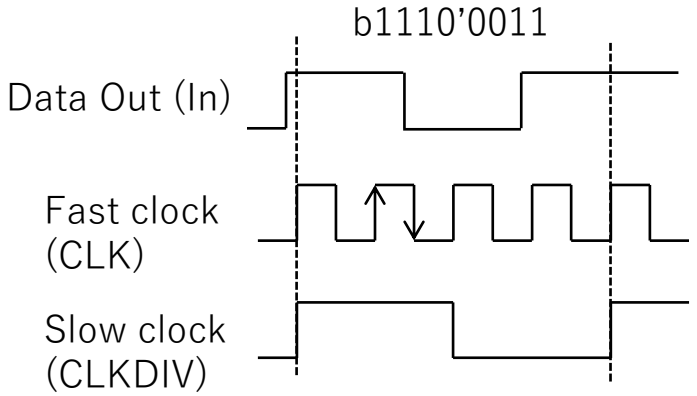
パラレルデータは CLKDIVに同期 (slow clock)

ビットオーダーが逆転する
IPカタログで生成するときは送信側で
ひっくり返してくれる。
手実装するときは自分でひっくり返す。

パラレルデータは CLKDIVに同期

CLKDIV_RXは
CLK_RXから作る

8-bit DDR IOSERDES

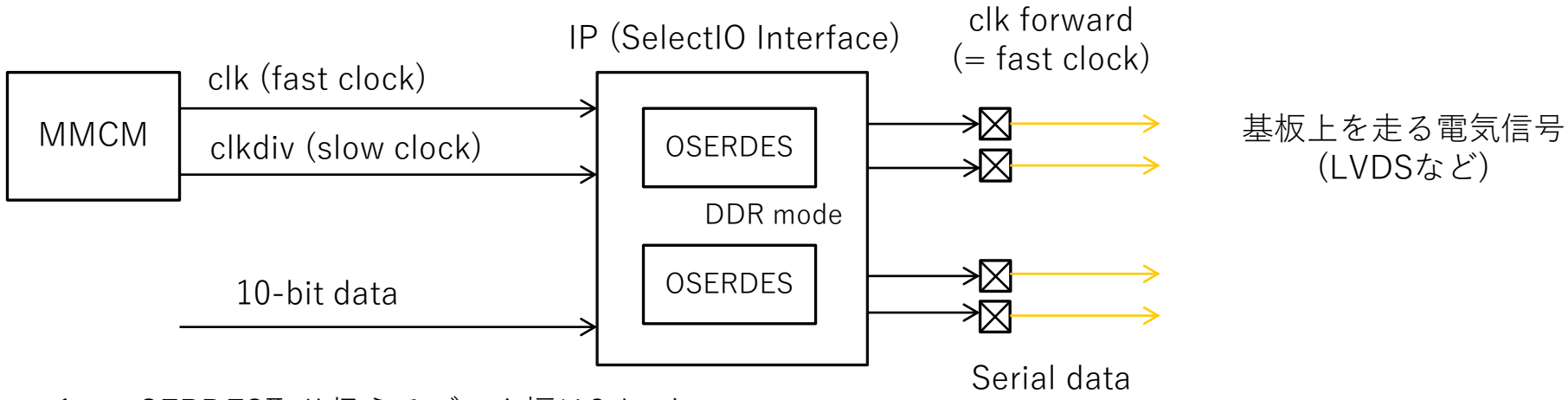


1つのIOSERDESプリミティブで処理
できるビット数は8-bitまで。
それ以上はカスケードが必要。

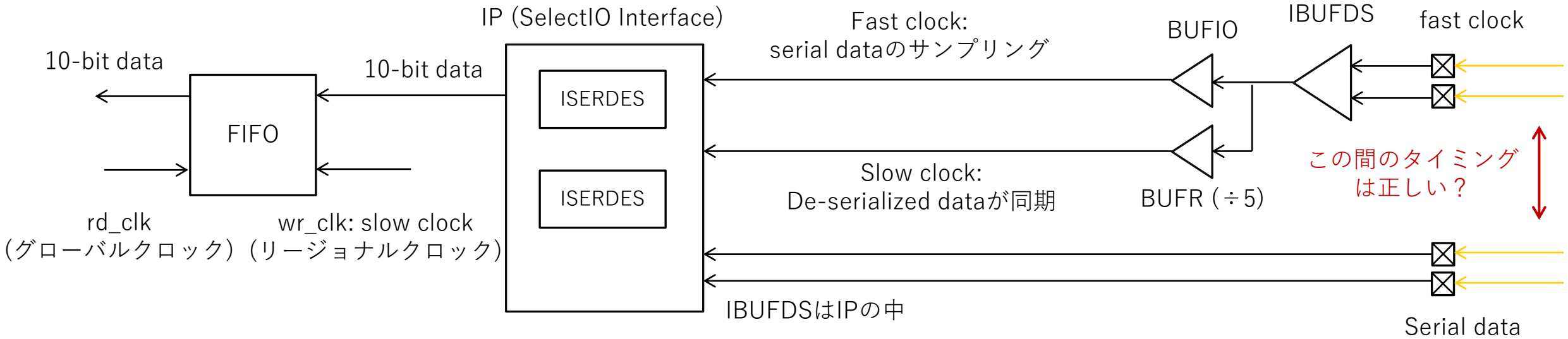
どうやって実装する？
最初はIPカタログで作ってみる。
どういうHDLが生成されたか確認して、
ゆくゆくは手実装も出来るようになるとよい。

現実的なFW構成

10-bitのデータをSERDESを使って送信する場合

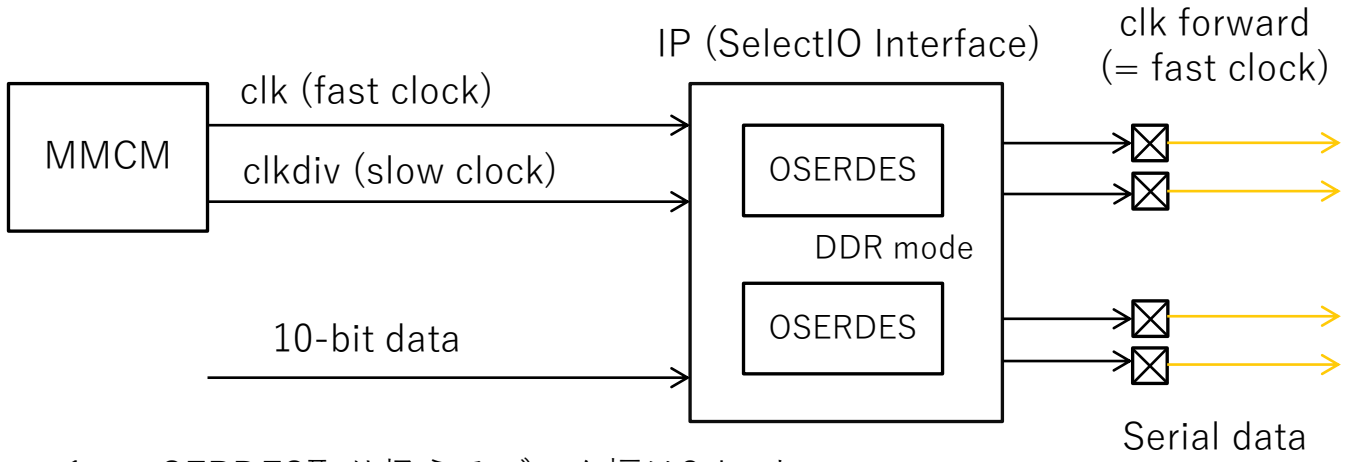


1つのSERDES取り扱えるデータ幅は8-bitまで。
2つカスケードして14-bitまで拡張可能。



現実的なFW構成

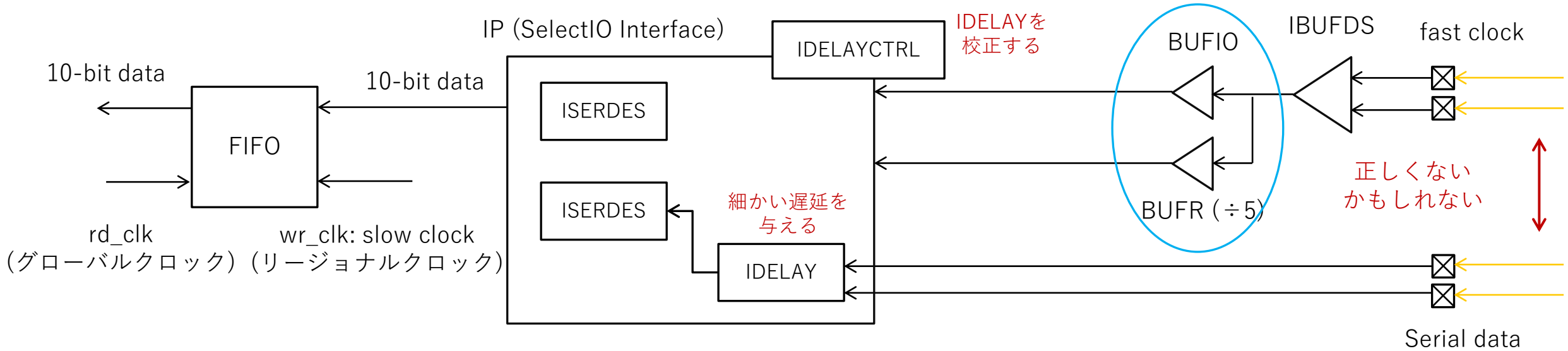
10-bitのデータをSERDESを使って送信する場合



基板上を走る電気信号
(LVDSなど)

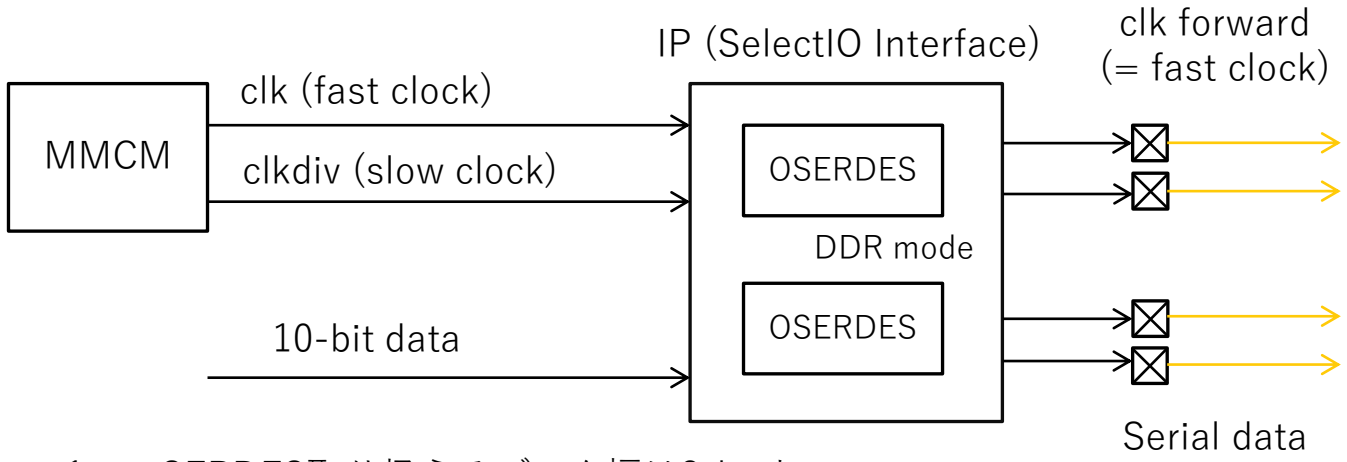
1つのSERDES取り扱えるデータ幅は8-bitまで。
2つカスケードして14-bitまで拡張可能。

分周比に応じた位相が出る
slow clockの位相がデータの切れ目と
一致していないかもしれない



現実的なFW構成

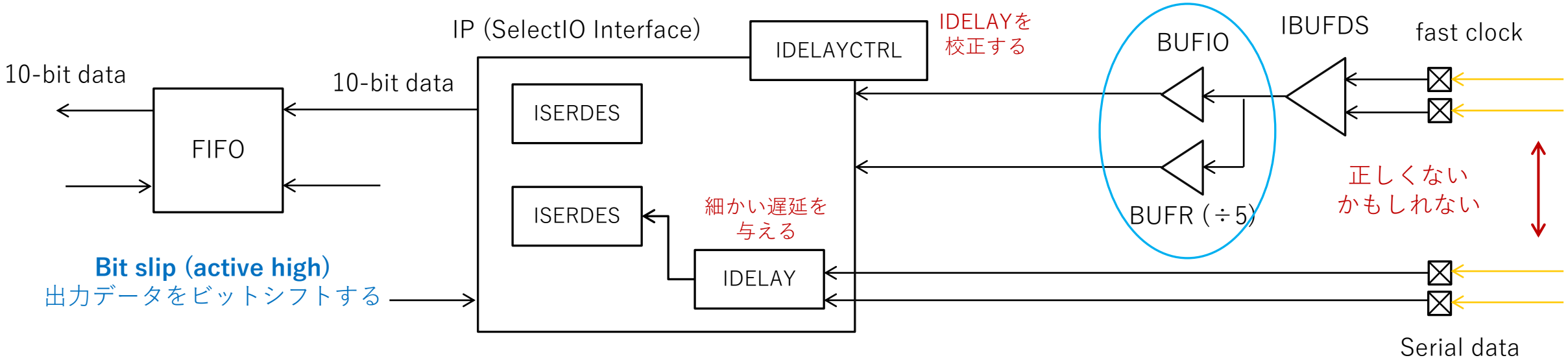
10-bitのデータをSERDESを使って送信する場合



基板上を走る電気信号
(LVDSなど)

1つのSERDES取り扱えるデータ幅は8-bitまで。
2つカスケードして14-bitまで拡張可能。

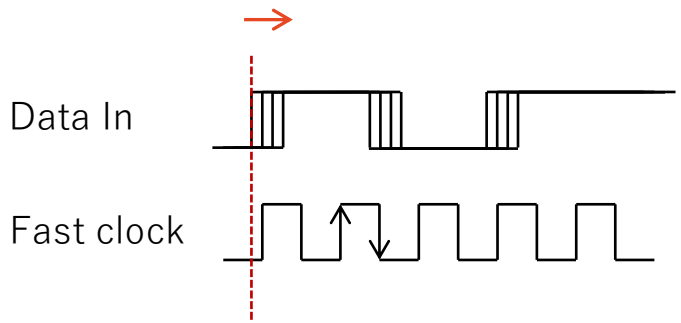
分周比に応じた位相が出る
slow clockの位相がデータの切れ目と
一致していないかもしれない



IDELAYを図で書くと

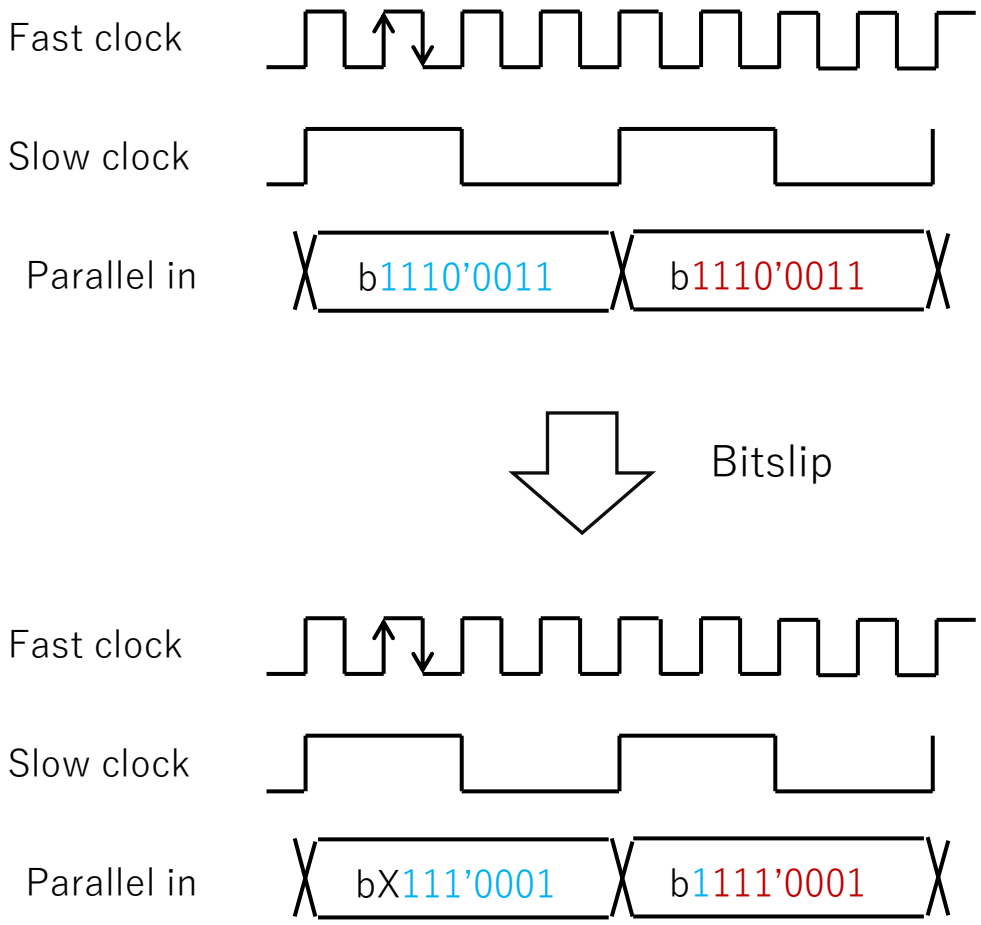
IDELAY

クロックエッジに対して
相対タイミングを変更するのが
IDELAY



IDELAYではクロックに対して遅延を与える事も可能

Bit slipを図で書くと



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

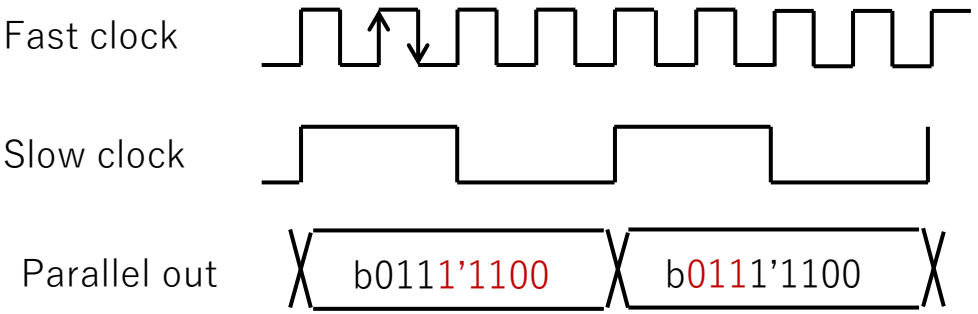
ug471_c3_11_012211

図 3-11 : Bitslip の処理例

SDRとDDRモードとでbitslipの動作が異なり、
DDRモードの場合は少し複雑。

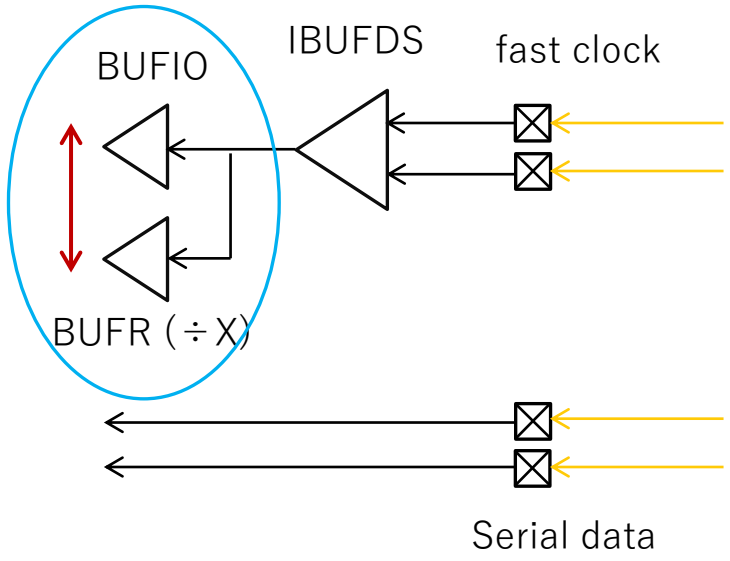
なぜbitslipが必要か？

例えば
ADCは8-bit data (b1110’0011)を送信しているが…



データが変な場所で切れてしまった
正しい位相を得るまでbitslipを繰り返す

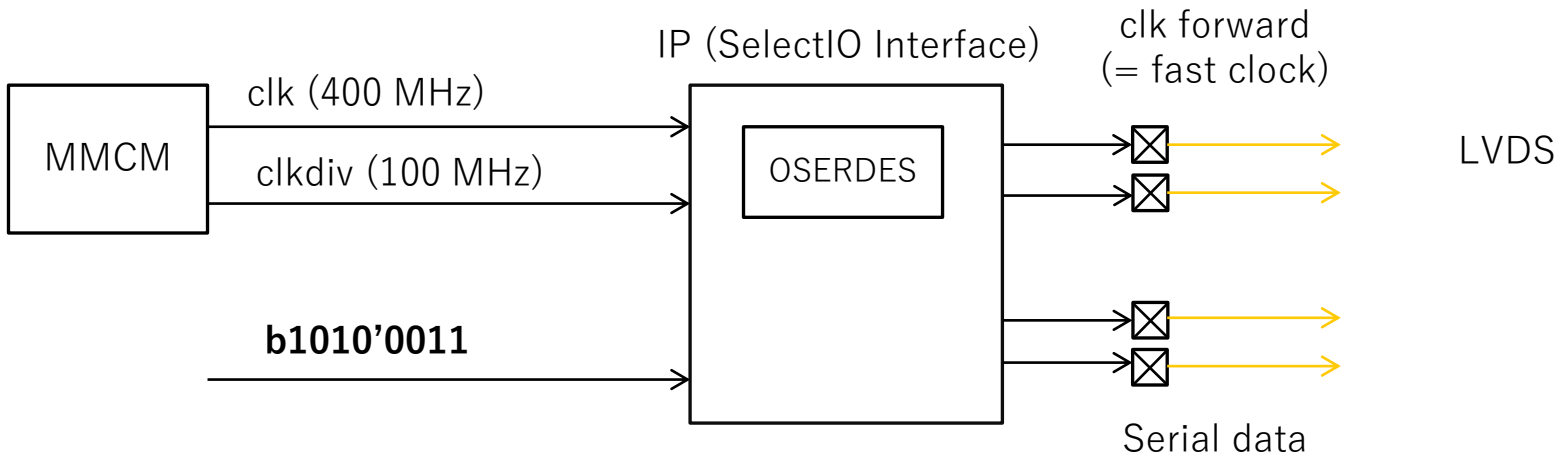
分周比に応じた位相が出る
slow clockの位相がデータの切れ目と
一致していないかもしれない

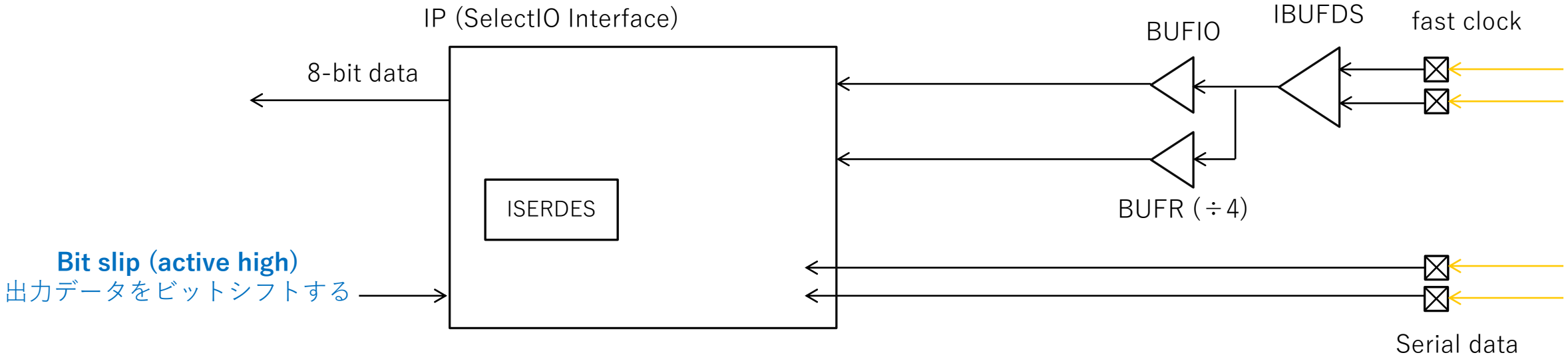


例題 EX2

1. OSERDESを使って8-bitの固定パターンデータを800 Mbpsで出力。
2. 基板外でケーブルを使ってループバックを作る。
3. ISERDESで受信。受け取ったビットパターンが正しいか確認する。

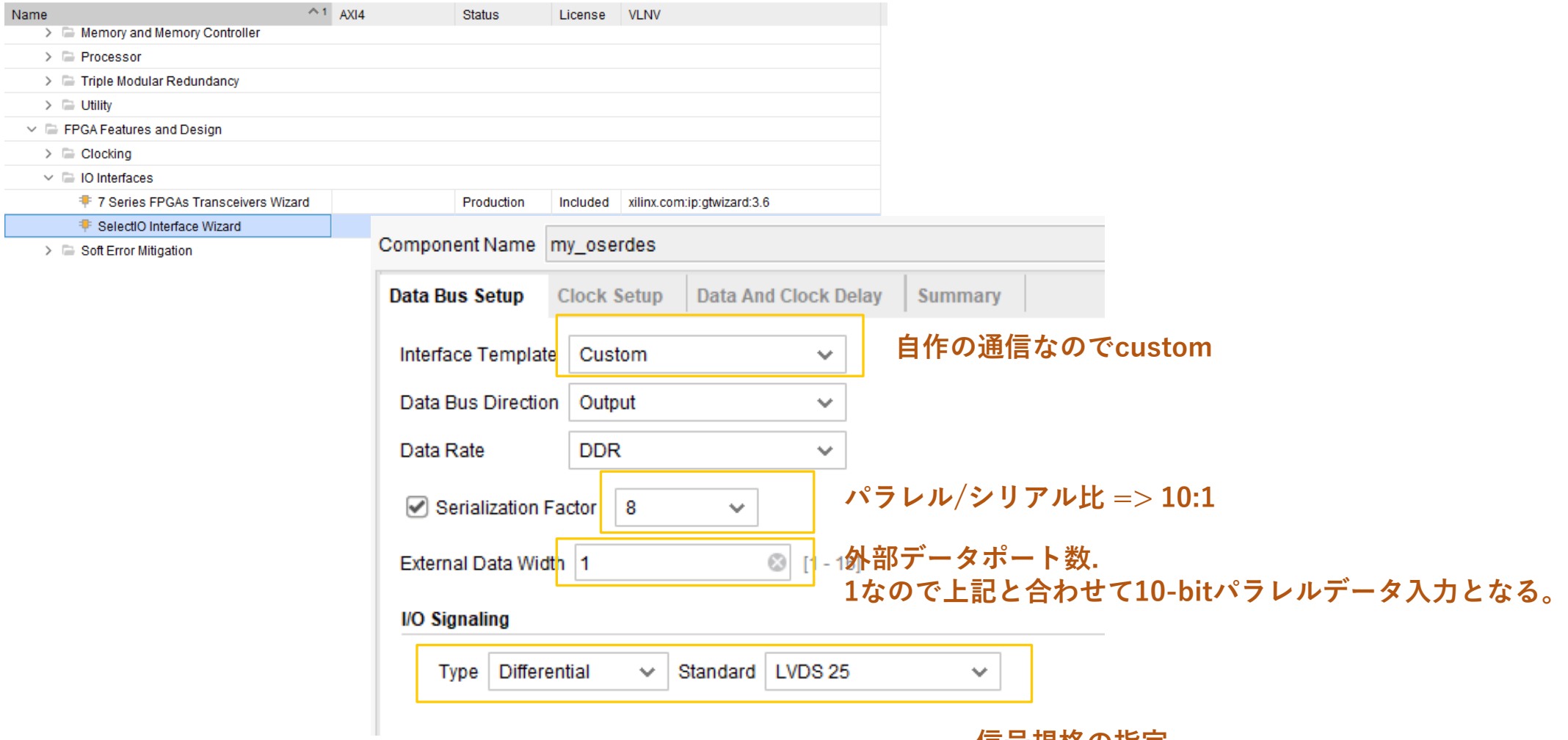
演習手順書
EX2





まずOSERDESを生成する

IP catalog



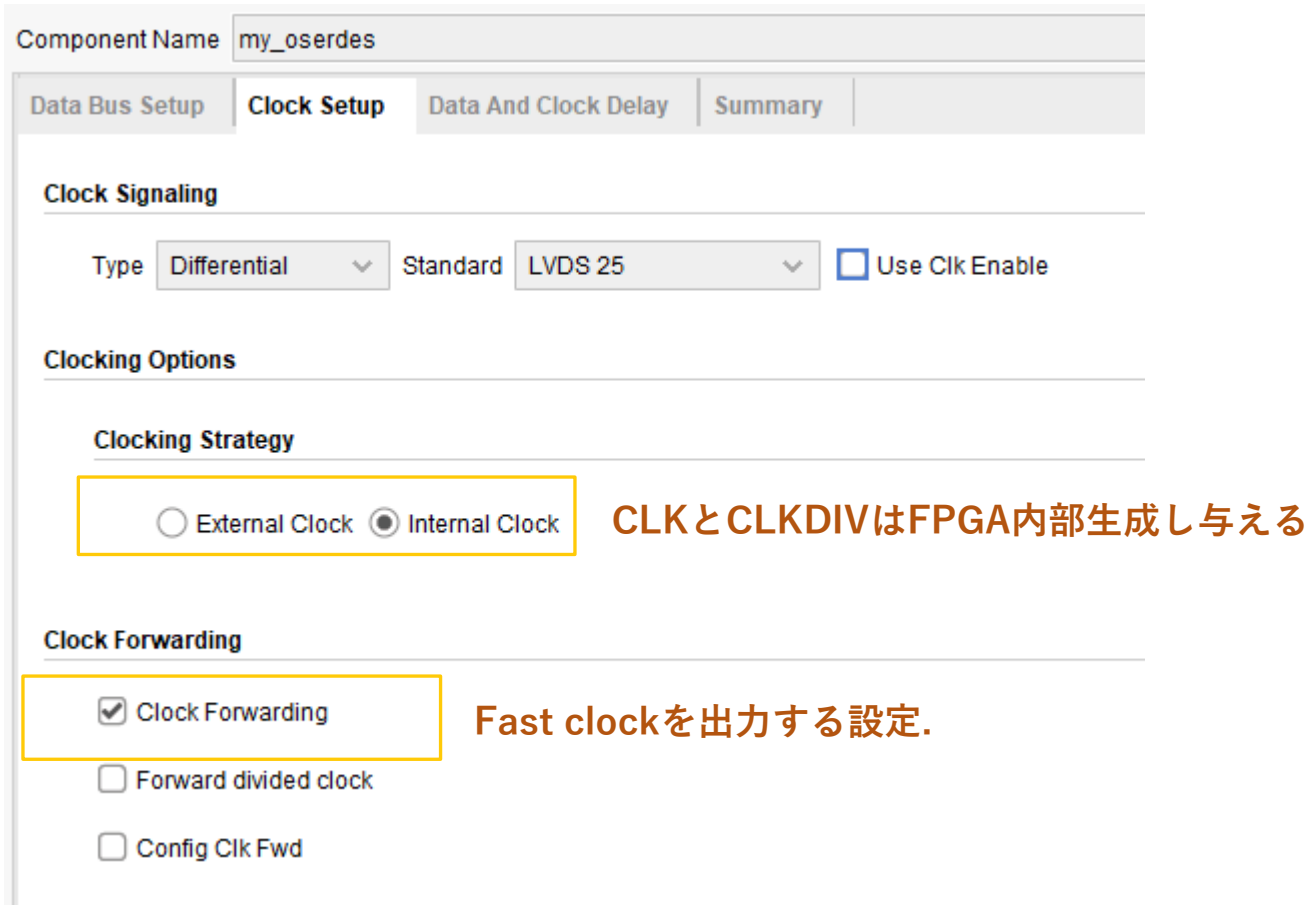
自作の通信なのでcustom

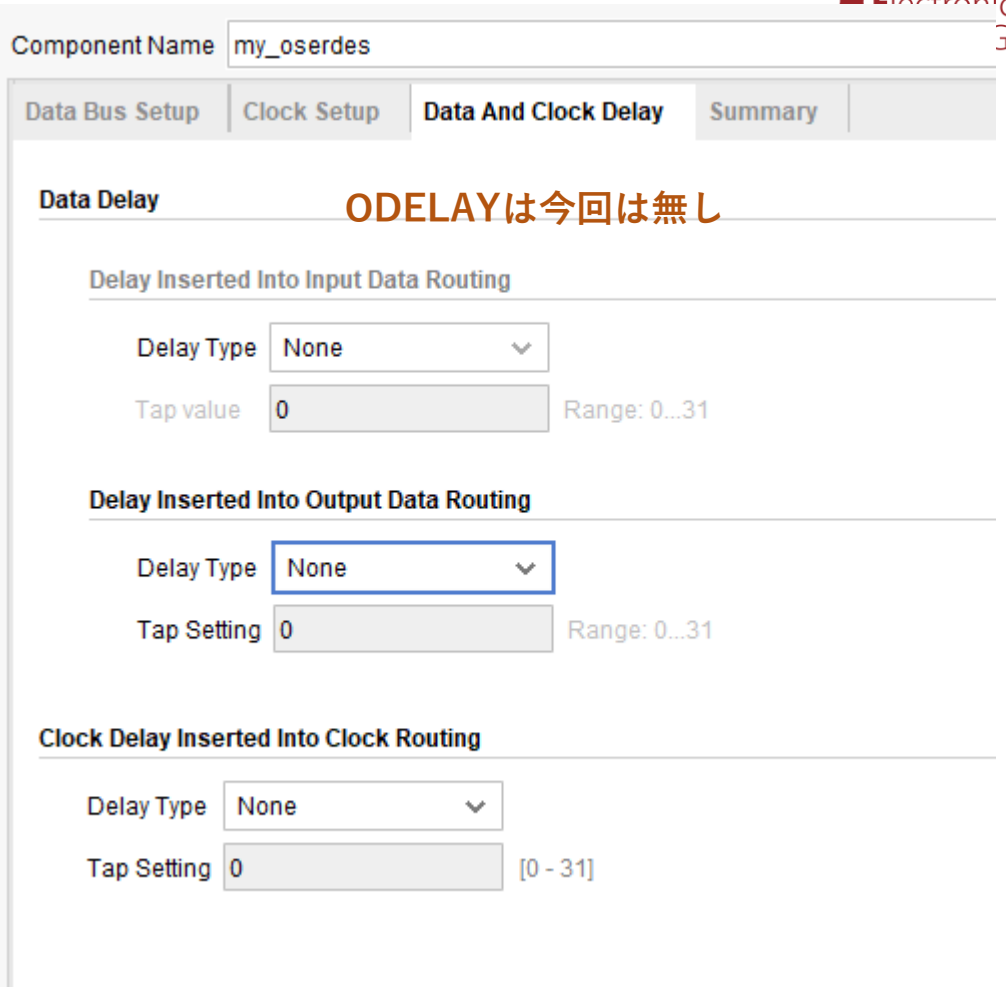
パラレル/シリアル比 => 10:1

外部データポート数.
1なので上記と合わせて10-bitパラレルデータ入力となる。

信号規格の指定.

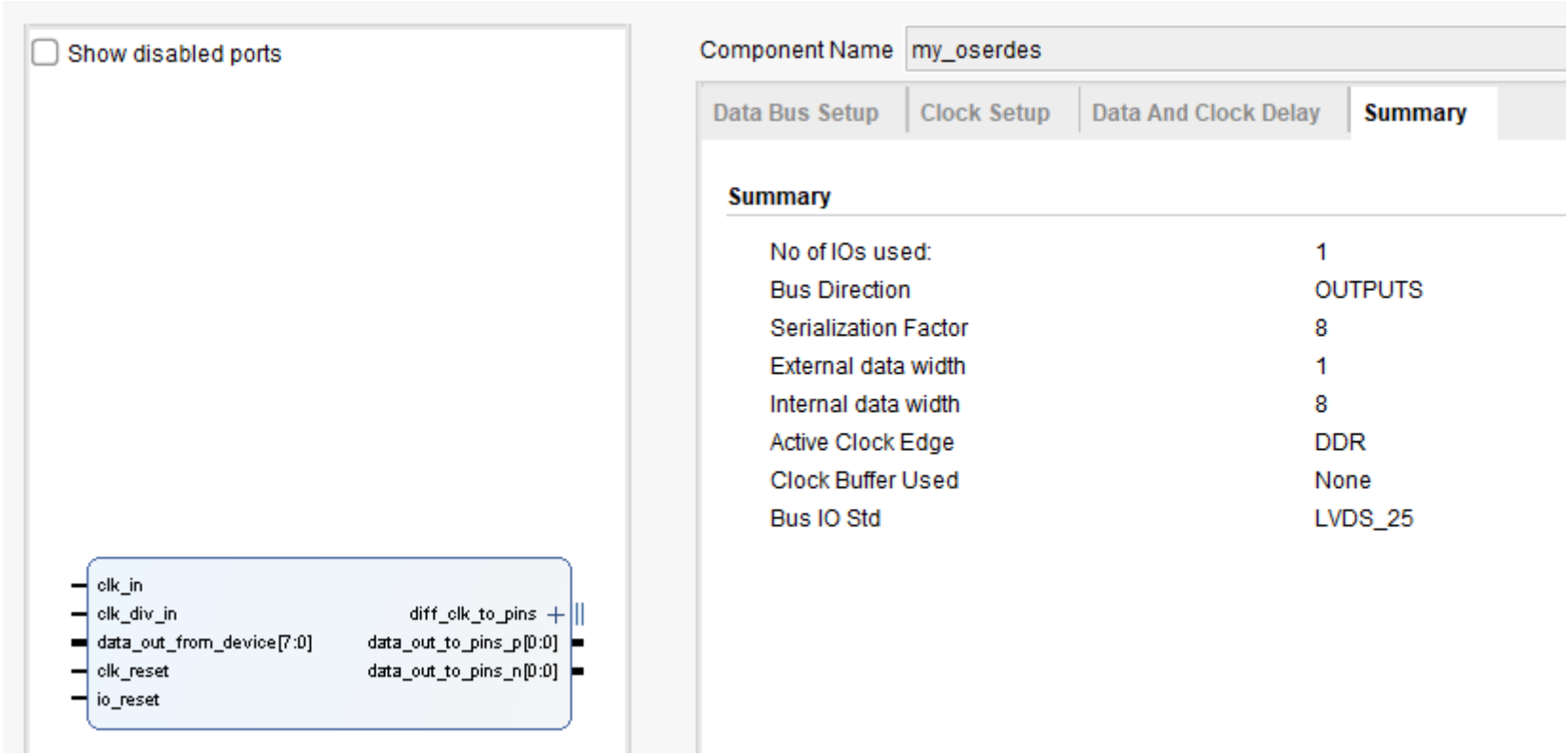
まずOSERDESを生成する



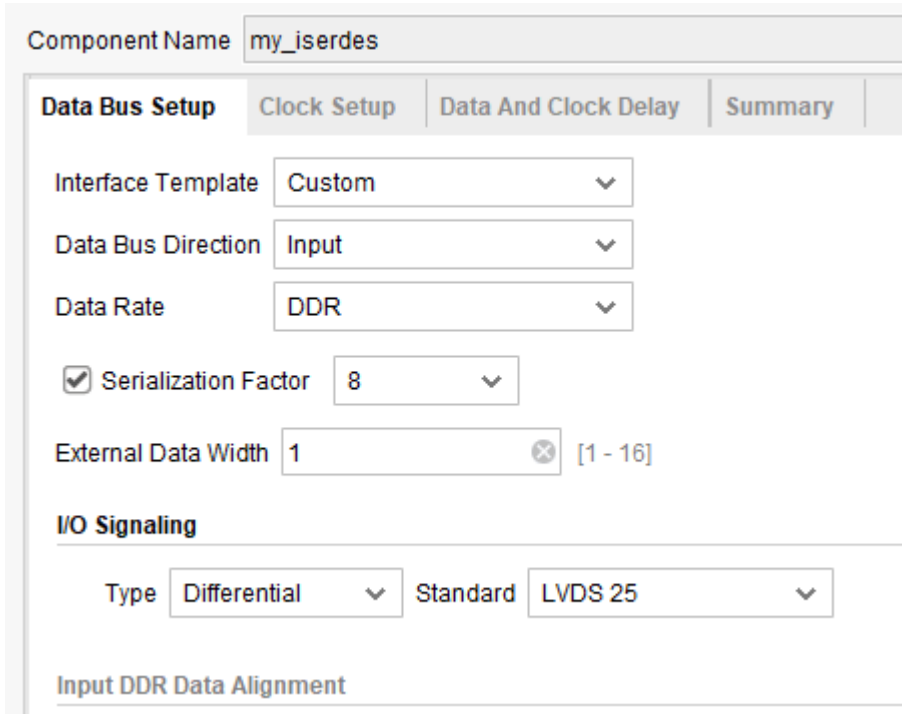


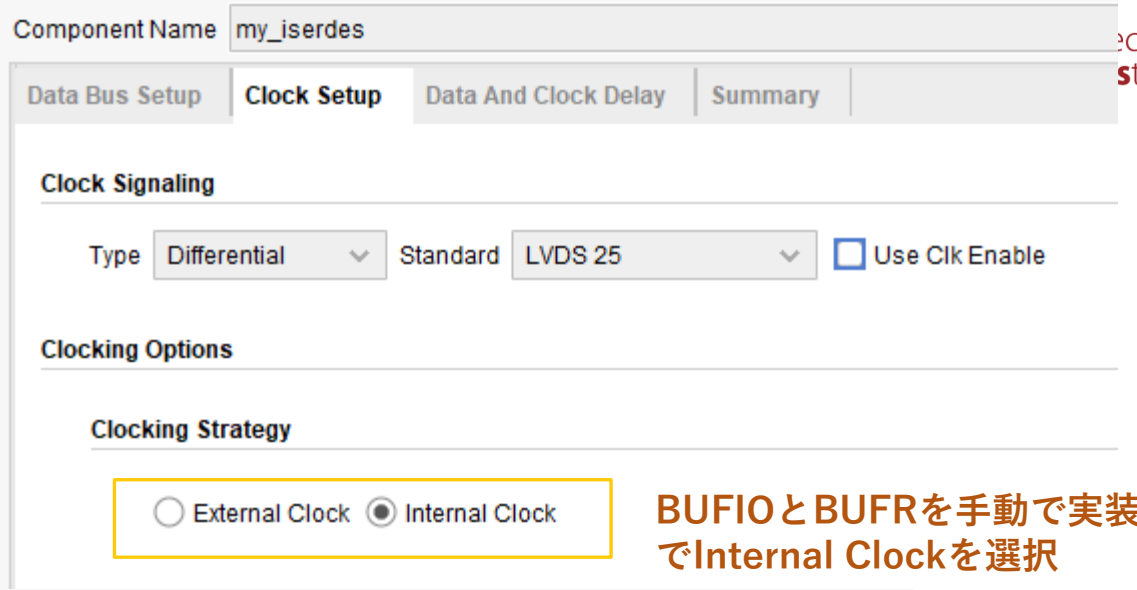
まずOSERDESを生成する

このようになっていますか？

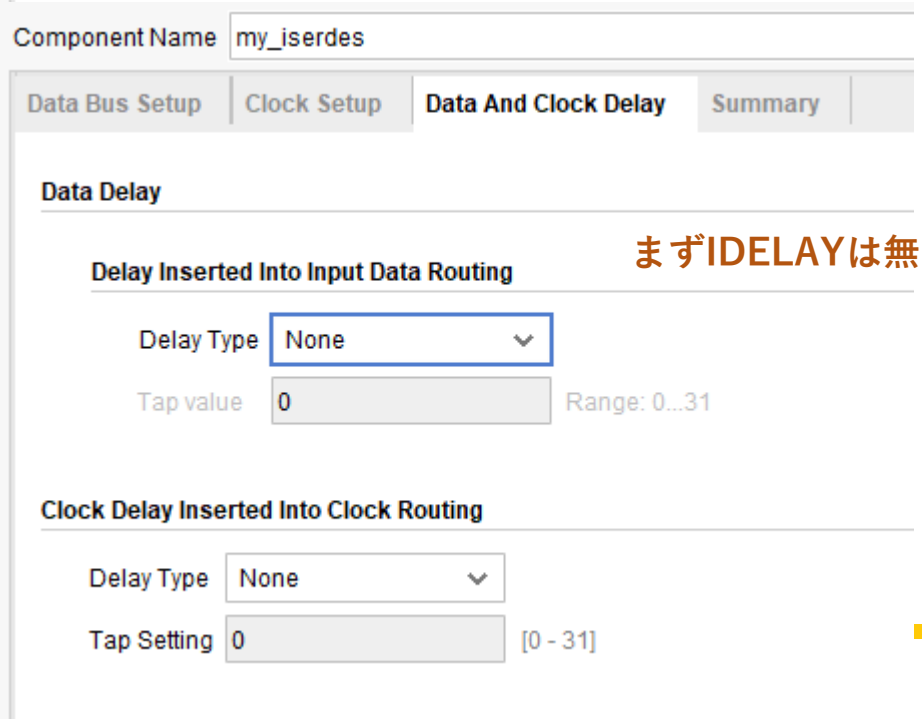


次にISERDESをIDEALY無しで生成してみる





BUFIOとBUFRを手動で実装するのでInternal Clockを選択

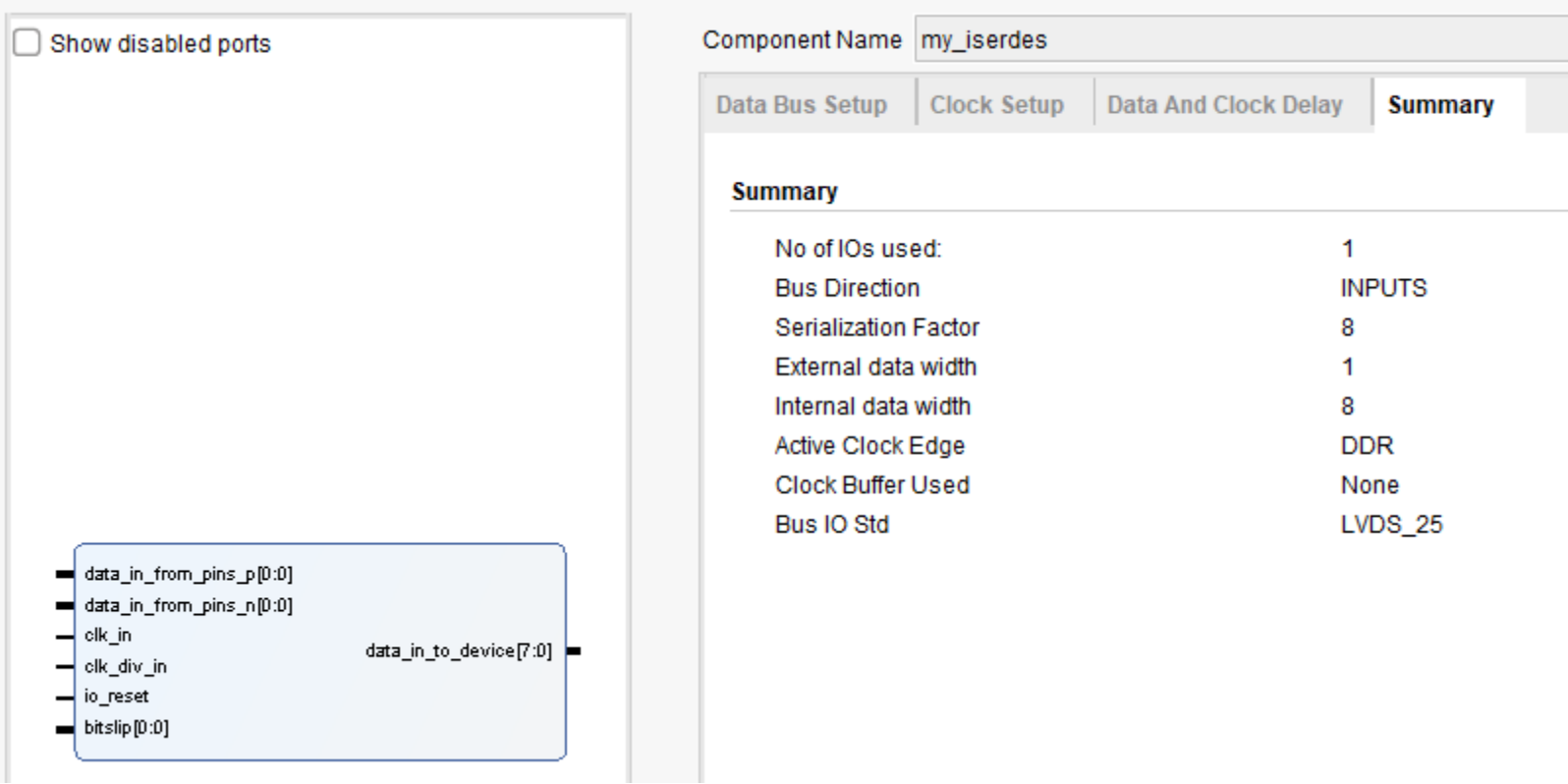


まずIDELAYは無しでやってみる

次にISERDESをIDEALY無しで生成してみる

Electronics System Group

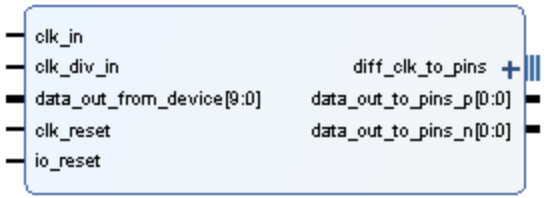
このようになっていますか？



openit
Open source consortium of Instrumentation

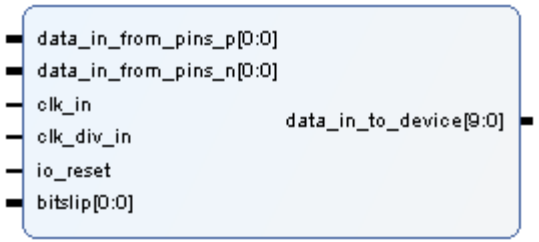
HDL上で配線する

- 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. 100 MHz clockに同期。
  - **定数:b1010'0011 を与える**
- clk_reset
  - Active high. プッシュボタンを配線.
- io_reset
  - Active high. プッシュボタンを配線.



- diff_clk_to_pins
  - トップレベルポートに直結。
- data_out_to_pins_p/n
  - トップレベルポートに直結。

- diff_clk_in
  - トップレベルポートに直結。
- data_in_from_pins_p/n
  - トップレベルポートに直結。
- clk_reset
  - open
- io_reset
  - Active high. プッシュボタンを配線.
- bitslip
  - **VIOのprobe_outで制御する**
  - **VIOはリージョナルクロックで駆動**



- data_in_to_device
  - 8-bit data出力
  - **ILAへ接続。**
  - **リージョナルクロックに同期**

ISERDESはbitslipがHIGHの間毎クロックビットシフトを行う。
VIOのボタンを押したときにone-shot pulseが出るようにしてください。

できましたか？

VIOのボタンを押すたびに1回ビットスリップが
発生していればOK。

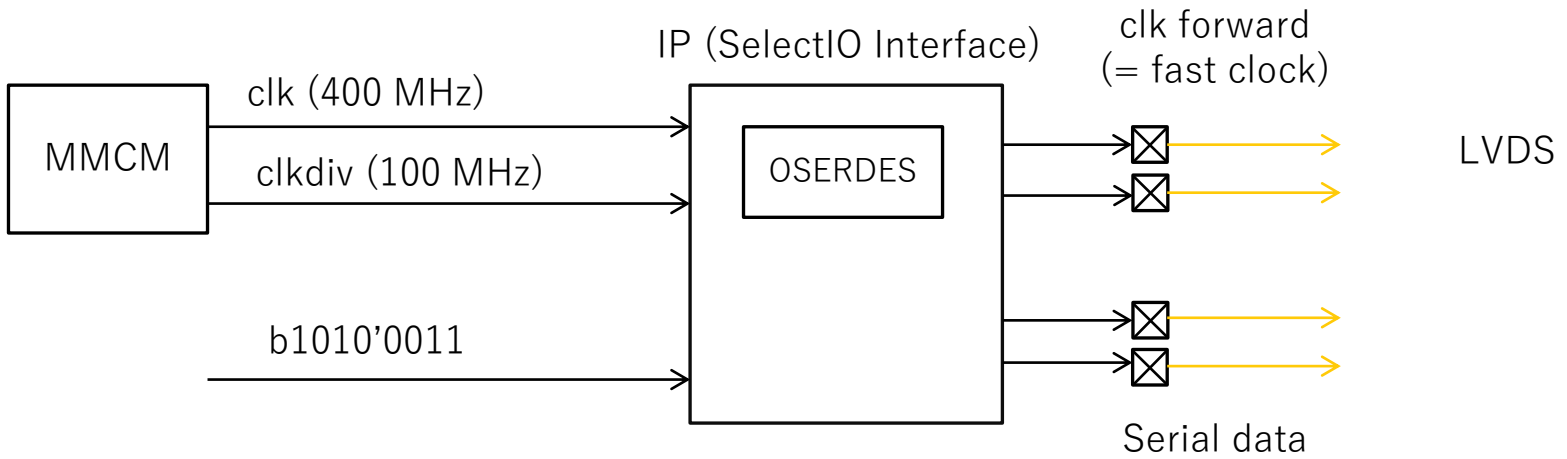
受信データのビットパターンが送信パターンに
一致するまでビットシフトしてみましょう。

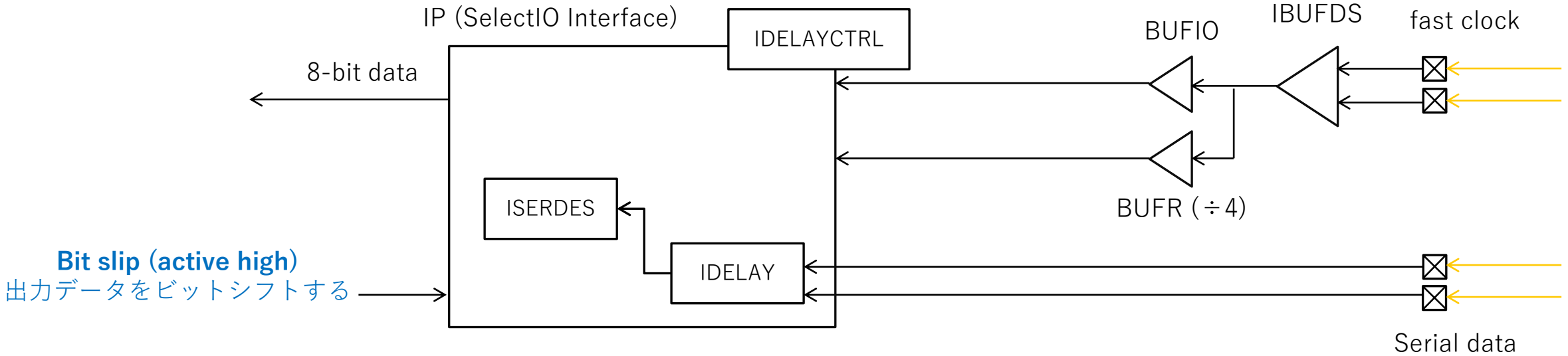
ここから発展演習用のスライド
初日は一旦スキップ

発展課題 H1

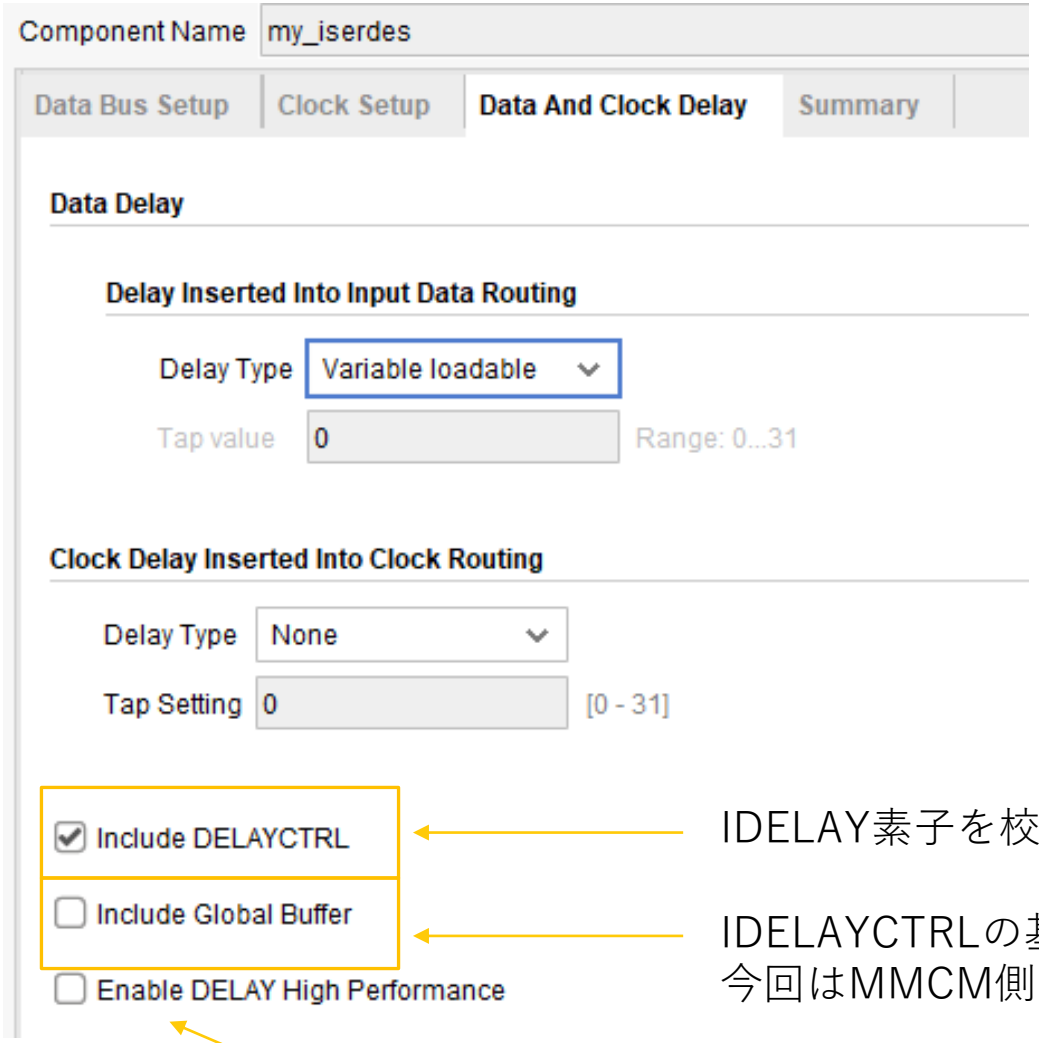
1. ISERDESの前にIDELAYを実装してみましょう

演習手順書
H1





IDEALYを実装する



IDELAYは32段階 (tap)でディレイ調整が可能

- Fixed: 固定値の遅延を与える。
- Variable: CEとINC信号を使って1つずつ上げ下げする。
- Variable loadable: Tap値を動的に指定する。

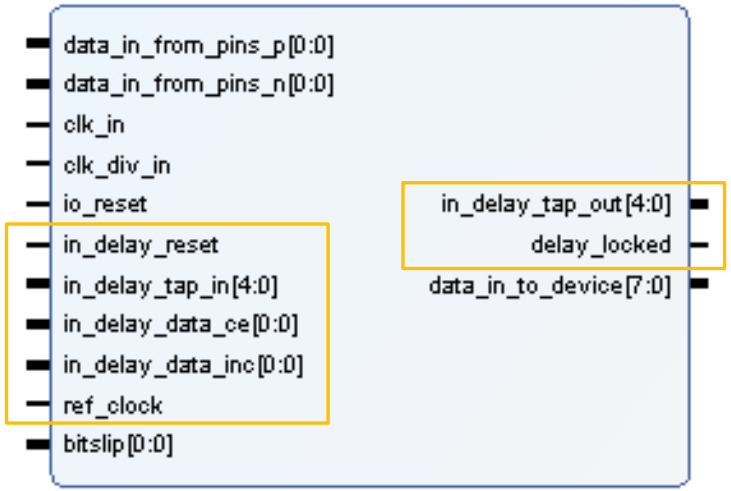
今回はvariable loadableでやってみます。

IDELAY素子を校正するためのIDELAYCTRLをIP内に含める

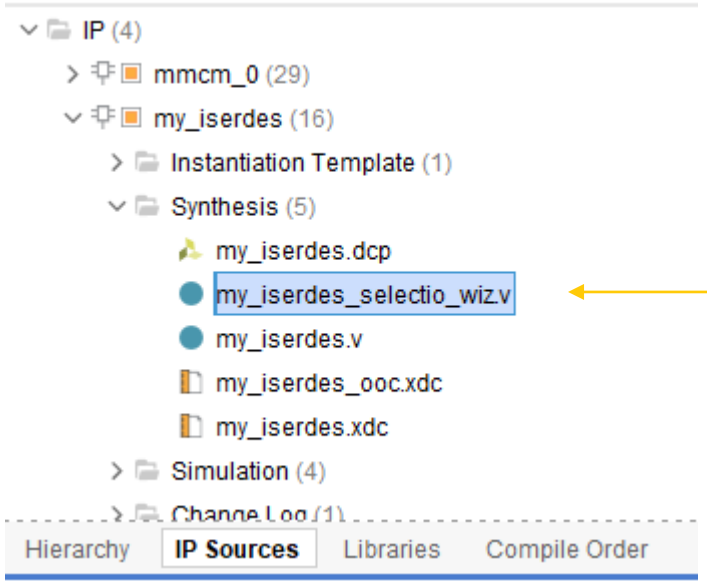
IDELAYCTRLの基準クロックはBUFGかBUFHで駆動されている必要がある。
今回はMMCM側でBUFG指定するのでチェックを外す。

指定するとIDELAY通過時のジッタが低減されるらしい。
限界速度付近で通信させるときはONした方が無難そう。

IDEALYを実装する

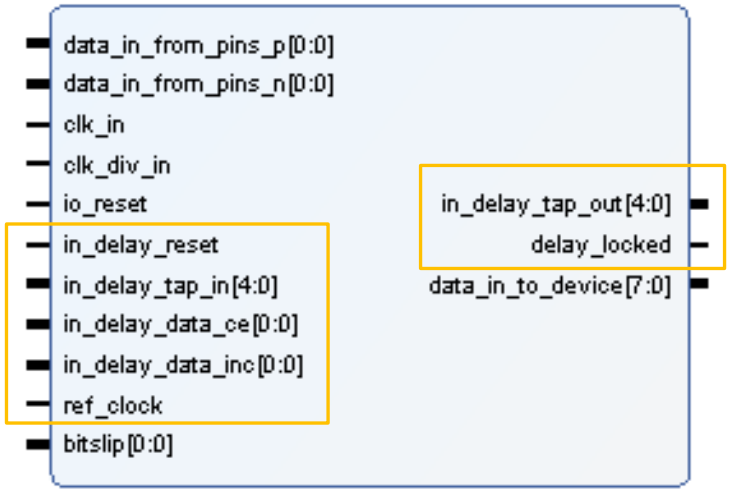


新しいポートが追加されたが機能があまり想像できない物がある…
IPあるあるなので生成されたHDLコードを読み解いて何に配線されているかを調べる。



ウィザードで生成した実装用のコード
IPカタログは様々な専用ブロックの実装例を与えるものだと思えばよい。

IDEALYを実装する




```
(* 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));
```

tronics
em Group

openit
Open source consortium of Instrumentation

IDEALYを実装する

```
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とINCはクロック (C)の同期入力。

- CEがHIGHの時INCがHIGHであればタップが1つ進む。
- CEがHIGHの時INCがLOWであればタップが1つ戻る。

モジュール ロード - LD

VARIABLE モードの場合、IDELAY のロード ポート LD は、IDELAY_VALUE 属性で設定した値をロードします。IDELAY_VALUE 属性のデフォルト値は 0 です。デフォルト値を使用する場合、LD ポートは ILDELAY 用の非同期リセットとして機能します。LD は入力クロック信号 (C) に同期したアクティブ High の信号です。

VAR_LOAD モードの場合、IDELAY のロード ポート LD は、CNTVALUEIN 属性で設定した値をロードします。CNTVALUEIN[4:0] に現れる値が新しいタップ値となります。VAR_LOAD_PIPE モードの場合、IDELAY のロード ポート LD は、パイプライン レジスタにある値をロードします。パイプライン レジスタに現れる値が新しいタップ値となります。

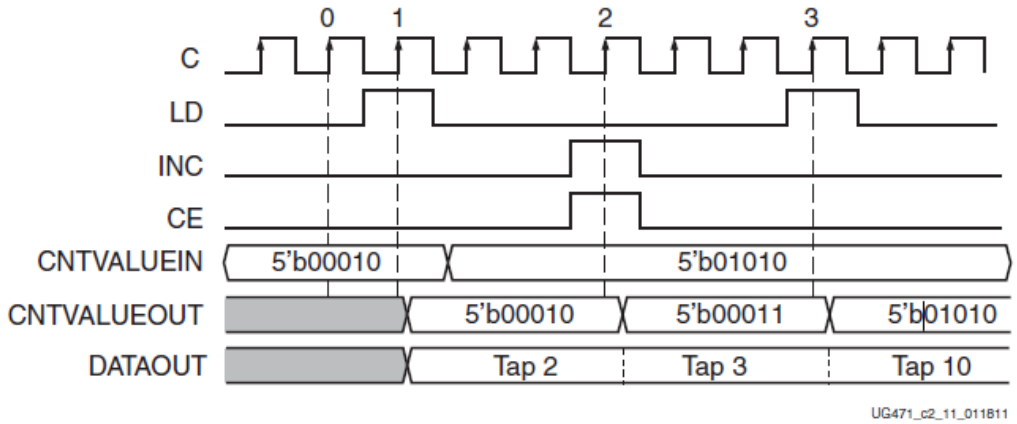


図 2-13 : VAR_LOAD モードの IDELAY のタイミング図

UG471

IDEALYを実装する

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

- RDY: 校正が出来ている事を示す。
- REFCLK: 校正用の基準クロック。
  - ⇒入れるべき周波数が分からない

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

Notes:

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

動作可能な周波数が
スピードグレードによって異なる！

ソースコードを移植するときは注意
しましょう。

余談

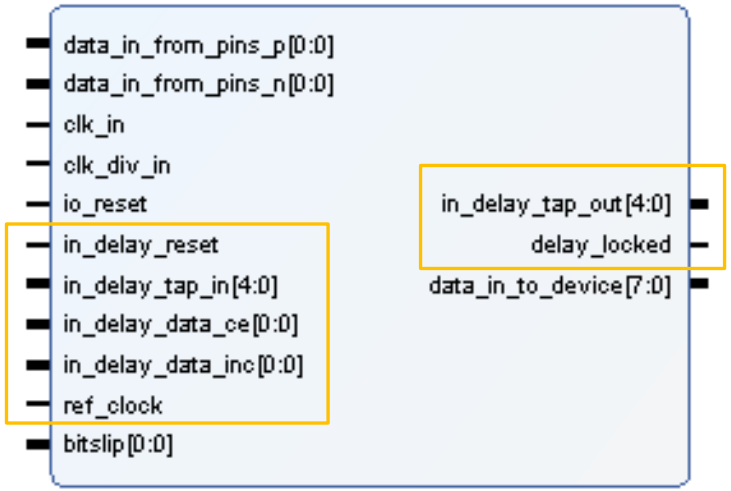
```
(* IODELAY_GROUP = "my_iserdes_group" *)
IDELAYE2
  # (
    .CINVCTRL_SEL           ("FALSE"),                          // TRUE, FALSE
    .DELAY_SRC              ("IDATAIN"),                        // IDATAIN, DATAIN
    .HIGH_PERFORMANCE_MODE  ("FALSE"),                          // TRUE, FALSE
    .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

書いてあることが矛盾していて良く分からない。
想像ではこの属性値は静的タイミング解析に利用するようなので、300や400を書いても解析結果には反映されません、という意味であると思われる。

IDEALYを実装する



まとめるとこの例題では次のようにするのが良さそうだ。

- in_delay_reset
  - VIOのprobe_outを使ってone-shot pulseを入れる。
  - bitslipの機能は残しておこう。
- in_delay_tap_in
  - VIOから与える
- in_delay_data_ce/inc
  - 利用しない。
- ref_clock
  - BUFG経由の200 MHzクロック。
- in_delay_tap_out
  - ILAへ接続
- delay_locked
  - Open

先ほどのソースコードを改造して実装してみましょう。

できましたか？

800 Mbpsなので1-bitあたり1.25 ns.
IDELAYは78 ps/tapで32 tapsなので2.5 ns.
すなわち32 tapsの中で1度くらいは正しくないタイミングになる
（実際には正しくなさそうなtap範囲）

実際にやってみて、どうなりました？

IDELAYCTRL

IDELAYCTRLは全てバンクのIOカラム**1つ**存在し
IDELAYとODELAYを校正します。

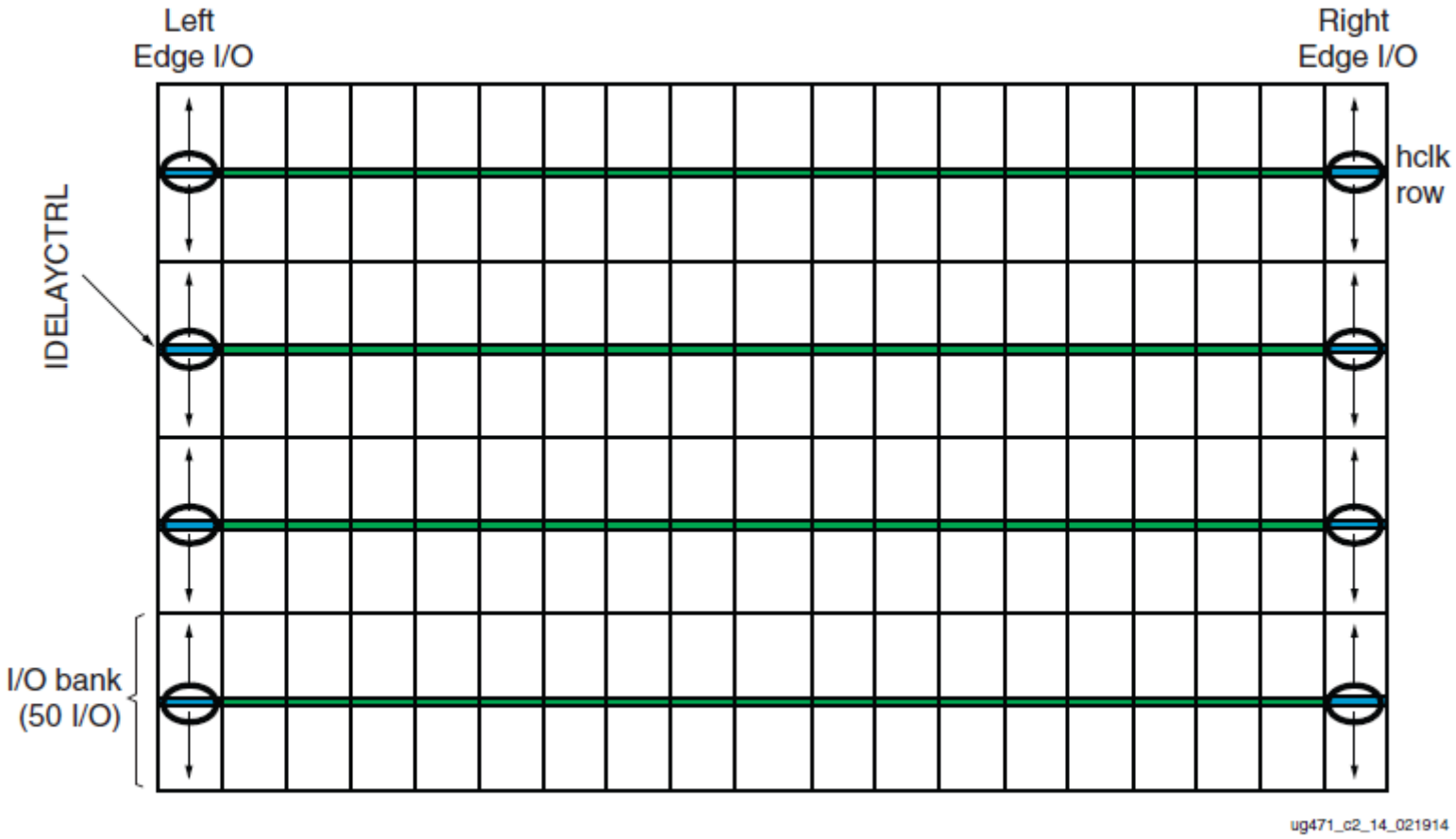


図 2-16 : IDELAYCTRL モジュールの位置関係

IDELAYCTRL

先ほど作ったIPを使ってこれから意図的にエラーを出します。
- my_iserdesをもう1つ実装し同一バンクのIOピンへ配線する。

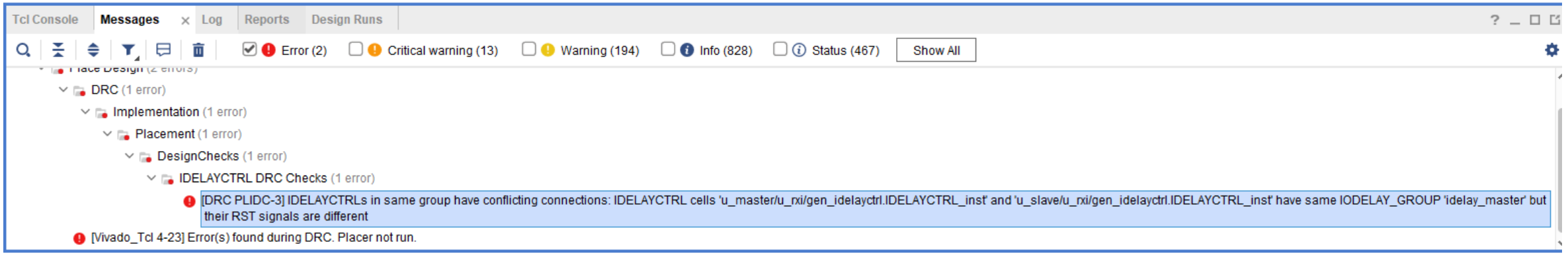
ポイント
- このIPは各バンクに1つしかないIDELAYCTRLを内蔵している。

これは私の方でやってみます。

IDELAYCTRL

未使用のIOカラムの中に用途不明なIDELAYCTRLが生成されたがエラーにはならなかった。
（未使用領域が減ってくるとエラーになる可能性がある）

今度は確実にエラーを出してみます。
- my_iserdes1のio_resetをmy_iserdes0とは別の信号にします。



IPからIDELAYCTRLを抜けばよいのでは？
答: それだとトラブルが潜在しています。

IODELAY_GROUPという制約がこのトラブルの鍵を握っています

IODELAY_GROUP

IODELAY_GROUPはIDELAYCTRLとIDELAY, ODELAY素子を関連付け、
校正の対応関係を与える制約。

Verilog-HDL

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

VHDL

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

```
-- 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;
```

対応関係を与える制約なので正しくグループ名を付けないといけないが、
IPで生成すると後から変更することが出来ない。

余談
私はコード内制約が可能な物はXDCではなくHDLで制約を与えてしまいます。
デバイスとコードが束縛されれてしまうので、再利用の点では悪い設計です。

IODELAY_GROUP

- 同一グループ名で機能が同じであると思われるIDELAYCTRL-IODELAYの組み合わせが複数ある場合、IDELAYCTRLの複製を行いVivadoが開発者が意図したであろう形に実装してくれる。
  - 先ほど見たように同一バンク内に無理やり2つIDELAYCTRLを実装しようとしても、なんとかしてくれる。
  - ただし、潜在的エラーのリスクを抱える事になる。

- 意図的に出したエラーではRST信号を変えることで、同一機能IDELAYCTRLで無いよう見せてVivadoの対処を阻害した。

完全な対処法

- 正しくIODELAY_GROUPをプログラマが記述する事。
- IPを利用すると名前を変えて別のIPにしない限り同一IODELAY_GROUPになってしまう。
  - ADCのデータをSERDESで受けるだけのようなファームウェアでは、VivadoによるIDELAYCTRLの複製に期待してもよい。
  - 複数の外部機器とIOSERDESを利用して通信を行う複雑なファームウェアでは、IPのHDLコードを参考にしつつ手動実装の方が柔軟性が高い。