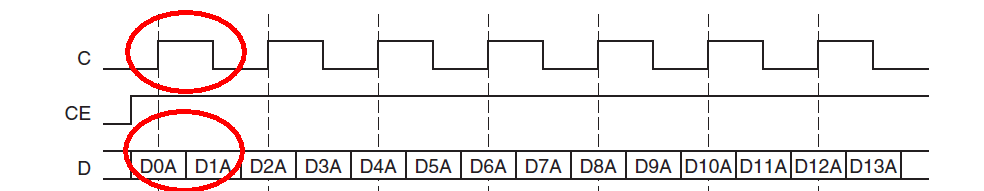
# Chip Sync, Source synchronous serialization & deserialization

## IDDR

FPGA 와 DDR와 DDR2 메모리를 인터페이스 할 때는 단순히 신호를 연결하는 것 이상의 주의가 필요 합니다.

왜냐하면 DDR이나 DDR2와 데이터를 주고 받기 위해서는 데이터 라인뿐만 아니라 클럭도 같이 고려해야 하기 때문 입니다.

다음 그림을 보시면 하나의 클럭의 상승 시점과 하강 시점에 데이터가 변하고 있는 것을 확인할 수 있습니다.



이러한 신호를 FPGA에서 읽으려 할 때는 다음과 같은 로직을 생각할 수 있습니다.

???? 🡺

이 그림을 보면 데이터 입력 핀으로부터 2개의 FF을 연결 하도록 합니다.

그리고 어디선가 입력된 클럭 신호를 그대로 하나의 FF에 연결하고 나머지 하나의 FF에는 클럭의 인버팅 (not gate)를 붙여서 연결하면 됩니다.

음~~ 이렇게 생각하는 것은 일단 불가피해 보입니다.

하지만 FPGA 내부 구조를 좀더 자세히 이해하신다면 하나의 데이터 입력 핀으로부터 2개의 FF까지의 선의 길이를 일정하게 하는 것이 쉬운 일이 아님을 알 수 있습니다.

요즘처럼 DDR이나 DDR2의 데이터 인터페이스 속도가 높은 경우에는 하나의 데이터 입력 핀으로부터 2개의 FF까지의 데이터 패스 길이를 정확하게 맞추는 것은 결코 쉬운 일이 아닙니다.

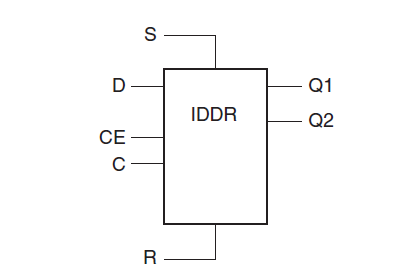
또하나의 문제점은 DDR이나 DDR2 메모리들의 데이터 핀들은 보통 16비트, 32비트, 64비트라는 버스 형태로 이루어져 있기 때문 입니다.

예를 들어 16비트 버스로 이루어진 DDR2 메모리로부터 데이터를 읽어들이는 핀으로부터 FF까지를 일정하게 하는 것은 일반적인 FPGA 설계 방법으로는 불가능 합니다.

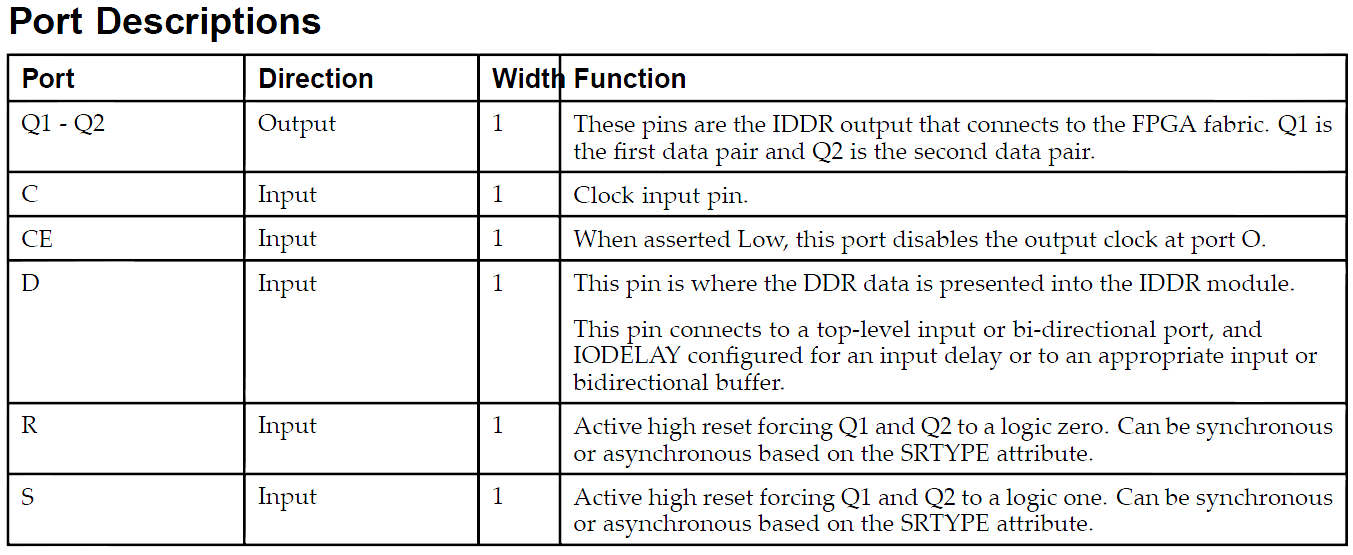
이런 문제를 해결하기 위해서는 기존의 설계방식 (전통적인 HDL 코딩 방식)에서는 벗어나야 합니다.

### IDDR

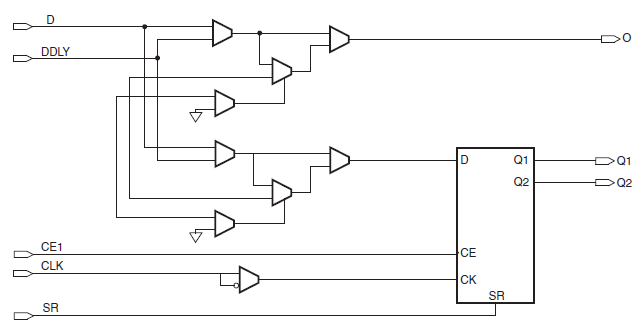
다음 그림은 IDDR에 대한 블록도 입니다. 먼저 이 그림에서 나타난 IDDR의 여러 입출력 포트를 살펴보고 이러한 포트가 어떻게 앞에서 언급한 문제를 해결할 수 있는 살펴 보도록 하겠습니다.



다음 그림은 IDDR의 포트에 대한 설명 입니다.



이 블록도에 대한 내부 회로도는 다음과 같습니다.

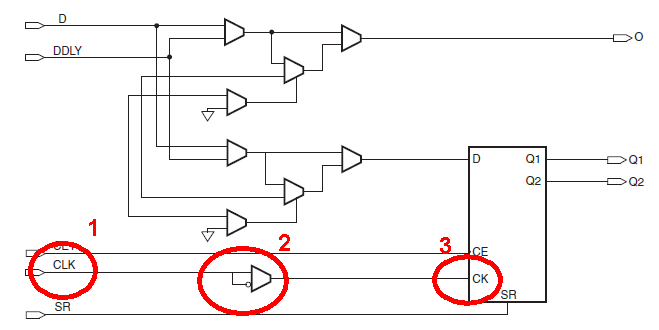


#### CLK 포트

다음 그림을 보면 외부에서 입력되는 클럭(1)의 상승 시점과 하강 시점에 데이터를 읽기 위해서는 인버터(2)가 필요하다고 설명 했습니다.

일반적으로 인버터는 FPGA 로직으로 구현되지만 여기에 있는 인버터는 IDDR 내부에 있기 때문에 추가로 로직을 사용하지 않습니다. 따라서 IDDR 내부에 있는 2개의 FF까지의 딜레이를 일정하게 할 수 있습니다.

요약하면 IDDR을 사용하면 클럭 입력 핀으로부터 2개의 FF까지의 클럭 신호 딜레이를 최소화 할 수 있습니다. (거의 0nsec)



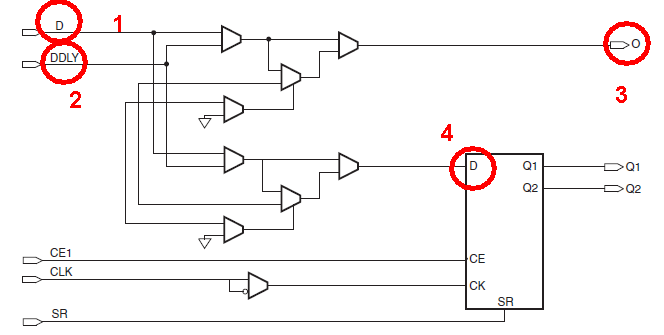
#### 데이터 패스

다음 그림은 IDDR의 데이터 패스에 대해서 설명하고 있습니다.

먼저 4번 포트로 입력되는 소스를 2가지 (1, 2번)을 선택할 수 있습니다. 1번 소스는 데이터 입력 포트로부터 직접 연결되는 경우에 사용되는 패스 입니다.

나중에 설명 하겠지만 DDLY(2번)은 Virtex6의 idelay 블록의 출력 포트와 연결할 때 사용하는 패스 입니다.

3번은 IDDR블록을 거치지 않고 바로 FPGA 내부에 연결할 때 사용 합니다.



### IDDR 타이밍도

IDDR을 이용할 경우 IDDR/Q1, Q2로부터 데이터를 읽는 방법은 다음과 같이 3가지가 있습니다.

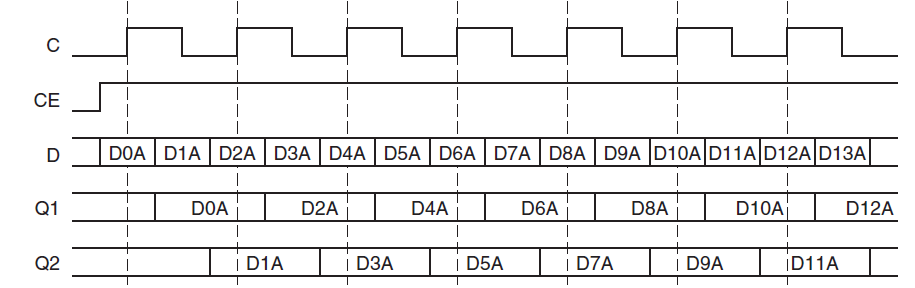
• OPPOSITE\_EDGE mode

• SAME\_EDGE mode

• SAME\_EDGE\_PIPELINED mode

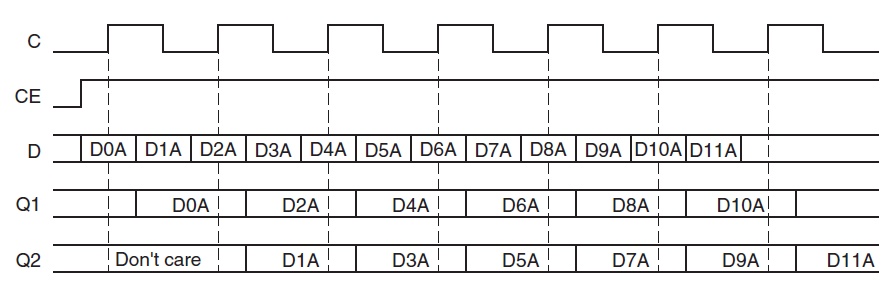
#### OPPOSITE\_EDGE mode

다음 그림은 IDDR을 OPPOSITE\_EDGE mode로 사용할 경우 나타나는 타이밍도 입니다.



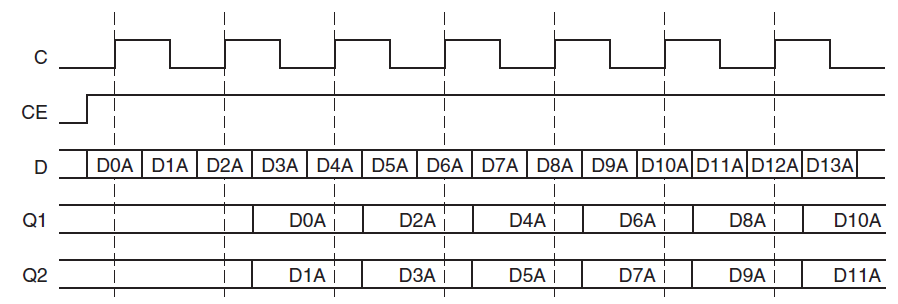
#### SAME\_EDGE mode

다음 그림은 IDDR을 SAME\_EDGE mode 로 사용할 경우 나타나는 타이밍도 입니다.



#### SAME\_EDGE\_PIPELINED Mode

다음 그림은 IDDR을 SAME\_EDGE\_PIPELINED Mode 로 사용할 경우 나타나는 타이밍도 입니다.



IDDR\_TEST.VHD

다음 리스트는 FPGA에 있는 IDDR을 사용하는 방법에 대해서 보여주고 있습니다.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

library UNISIM;

use UNISIM.VComponents.all;

entity IDDR\_test is

Port (

clkin\_p : in std\_logic;

clkin\_n : in std\_logic;

din\_p : in std\_logic\_vector(7 downto 0);

din\_n : in std\_logic\_vector(7 downto 0);

douta : out std\_logic\_vector(7 downto 0);

doutb : out std\_logic\_vector(7 downto 0) );

end IDDR\_test;

architecture Behavioral of IDDR\_test is

signal clkin : std\_logic;

signal din : std\_logic\_vector(7 downto 0);

begin

IBUFGDS\_U0 : IBUFGDS

generic map (

DIFF\_TERM => FALSE,

IBUF\_LOW\_PWR => TRUE,

IOSTANDARD => "LVDS\_25" )

port map (

I => clkin\_p,

IB => clkin\_n,

O => clkin );

loop0 : for i in 7 downto 0 generate

IBUFDS\_U0 : IBUFDS

generic map (

DIFF\_TERM => FALSE,

IBUF\_LOW\_PWR => TRUE,

IOSTANDARD => "LVDS\_25" )

port map (

I => din\_p(i),

IB => din\_n(i),

O => din(i) );

IDDR\_inst : IDDR

generic map (

DDR\_CLK\_EDGE => "OPPOSITE\_EDGE", -- "OPPOSITE\_EDGE", "SAME\_EDGE" or "SAME\_EDGE\_PIPELINED"

INIT\_Q1 => '0', -- Initial value of Q1: '0' or '1'

INIT\_Q2 => '0', -- Initial value of Q2: '0' or '1'

SRTYPE => "SYNC") -- Set/Reset type: "SYNC" or "ASYNC"

port map (

S => '0', -- 1-bit set

R => '0', -- 1-bit reset

CE => '1', -- 1-bit clock enable input

C => clkin, -- 1-bit clock input

D => din(i), -- 1-bit DDR data input

Q1 => douta(i), -- 1-bit output for positive edge of clock

Q2 => doutb(i) ); -- 1-bit output for negative edge of clock

end generate;

end Behavioral;

TB001.VHD

다음 리스트는 IDDR을 위한 테스트 벤치 입니다.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_unsigned.ALL;

USE ieee.numeric\_std.ALL;

ENTITY tb001 IS

END tb001;

ARCHITECTURE behavior OF tb001 IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT IDDR\_test

PORT(

clkin\_p : IN std\_logic;

clkin\_n : IN std\_logic;

din\_p : IN std\_logic\_vector(7 downto 0);

din\_n : IN std\_logic\_vector(7 downto 0);

douta : OUT std\_logic\_vector(7 downto 0);

doutb : OUT std\_logic\_vector(7 downto 0)

);

END COMPONENT;

--Inputs

signal clkin\_p : std\_logic := '0';

signal clkin\_n : std\_logic := '0';

signal din\_p : std\_logic\_vector(7 downto 0) := "00000000";

signal din\_n : std\_logic\_vector(7 downto 0) := "00000000";

--Outputs

signal douta : std\_logic\_vector(7 downto 0);

signal doutb : std\_logic\_vector(7 downto 0);

signal tclk : std\_logic := '0';

signal en : std\_logic := '0';

signal temp : std\_logic\_vector(7 downto 0) := "00000000";

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: IDDR\_test PORT MAP (

clkin\_p => clkin\_p,

clkin\_n => clkin\_n,

din\_p => din\_p,

din\_n => din\_n,

douta => douta,

doutb => doutb

);

-- Clock process definitions

clkin\_p\_process :process

begin

clkin\_p <= '0';

wait for 5 ns;

clkin\_p <= '1';

wait for 5 ns;

end process;

clkin\_n <= not clkin\_p;

en <= '0', '1' after 102.5 ns;

tclk <= not ( tclk and en ) after 2.5 ns;

datain\_gen :process(tclk)

begin

if rising\_edge(tclk) then

temp <= temp + 1;

end if;

end process;

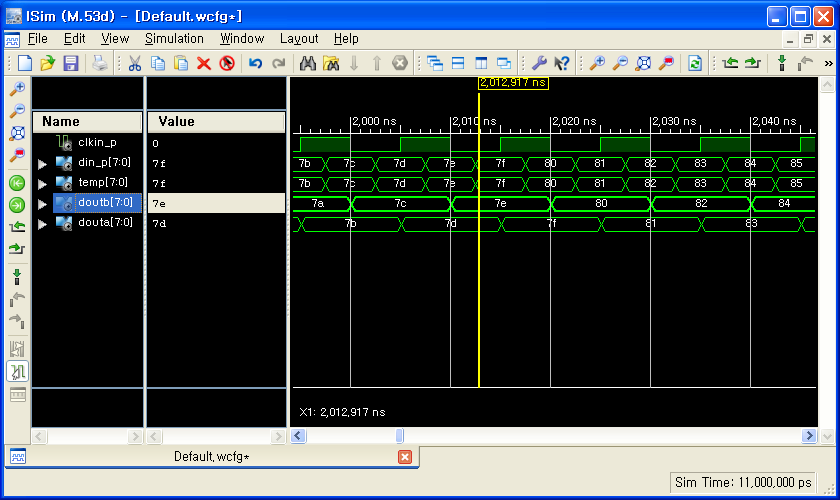
din\_p <= temp;

din\_n <= not temp;

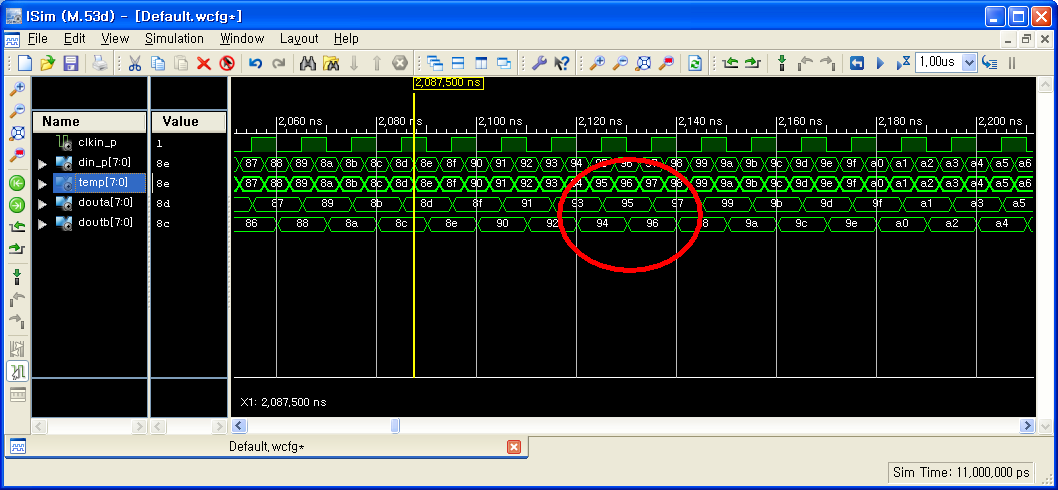
END;

시뮬레이션 결과

다음 그림은 IDDR을 시뮬레이션한 결과 입니다.



시뮬레이션을 하게 되면 clkin\_p의 rising과 falling에 데이터가 동시에 증가하는 것을 보여 줍니다. 이 값이 temp[7:0] 에 저장된 후 각각 douta[7:0]과 doubt[7:0]으로 출력 됩니다.



IDDR\_inst : IDDR

generic map (

-- DDR\_CLK\_EDGE => "OPPOSITE\_EDGE",

DDR\_CLK\_EDGE => "SAME\_EDGE",

-- DDR\_CLK\_EDGE => "SAME\_EDGE\_PIPELINED",

INIT\_Q1 => '0', -- Initial value of Q1: '0' or '1'

INIT\_Q2 => '0', -- Initial value of Q2: '0' or '1'

SRTYPE => "SYNC") -- Set/Reset type: "SYNC" or "ASYNC"

port map (

S => '0', -- 1-bit set

R => '0', -- 1-bit reset

CE => '1', -- 1-bit clock enable input

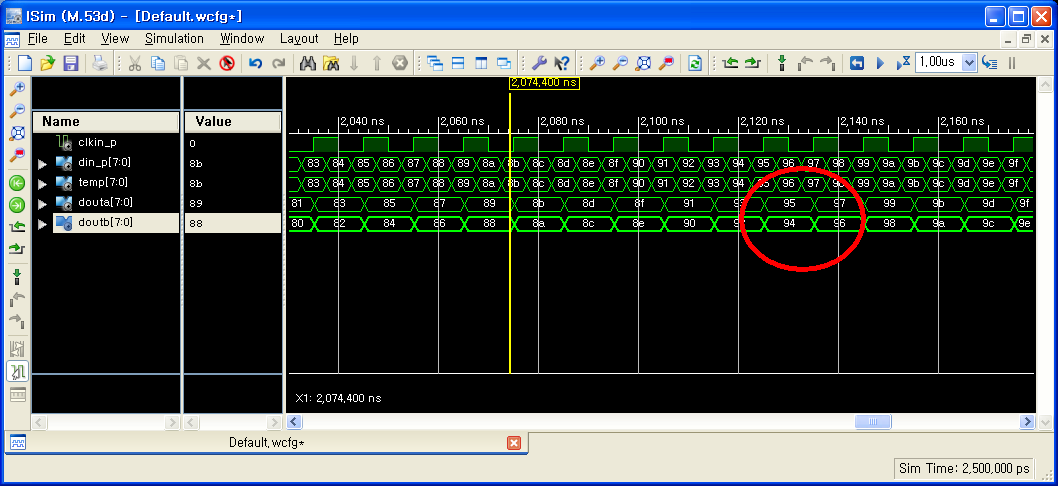
C => clkin, -- 1-bit clock input

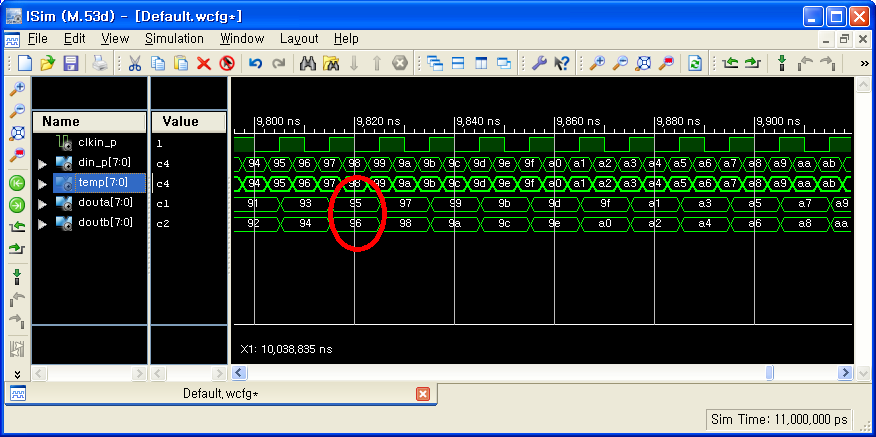
D => din(i), -- 1-bit DDR data input

Q1 => douta(i), -- 1-bit output for positive edge of clock

Q2 => doutb(i) ); -- 1-bit output for negative edge of clock

end generate;





IDDR\_inst : IDDR

generic map (

-- DDR\_CLK\_EDGE => "OPPOSITE\_EDGE",

DDR\_CLK\_EDGE => "SAME\_EDGE",

-- DDR\_CLK\_EDGE => "SAME\_EDGE\_PIPELINED",

INIT\_Q1 => '0', -- Initial value of Q1: '0' or '1'

INIT\_Q2 => '0', -- Initial value of Q2: '0' or '1'

SRTYPE => "SYNC") -- Set/Reset type: "SYNC" or "ASYNC"

port map (

S => '0', -- 1-bit set

R => '0', -- 1-bit reset

CE => '1', -- 1-bit clock enable input

C => clkin, -- 1-bit clock input

D => din(i), -- 1-bit DDR data input

Q1 => douta(i), -- 1-bit output for positive edge of clock

Q2 => doutb(i) ); -- 1-bit output for negative edge of clock

end generate;

## ODDR

앞에서 설명해 드린 IDDR과 마찬가지로 FPGA와 DDR/DDR2 같은 메모리에 데이터를 전달하기 위해서는 클럭의 상승시점과 하강시점에 데이터를 전달해야 합니다.

이렇게 하기 위해서는 데이터와 마찬가지로 클럭도 하나의 데이터로 취급하여 데이터와 함께 전달해야 합니다.

따라서 일반적으로 아래와 같은 회로를 구성하는 것이 자연스럽습니다.

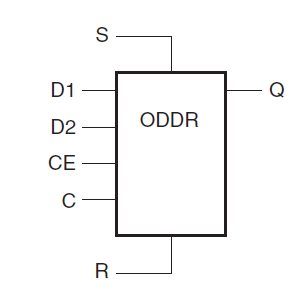
=🡺 ????

하지만 메모리의 처리 속도가 높아짐에 따라 클럭의 상승시점과 하강시점에 데이터를 출력 포트로 전달할 때 각각의 전달 지연시간이 최소가 되도록 하는데는 많은 노력이 필요 합니다.

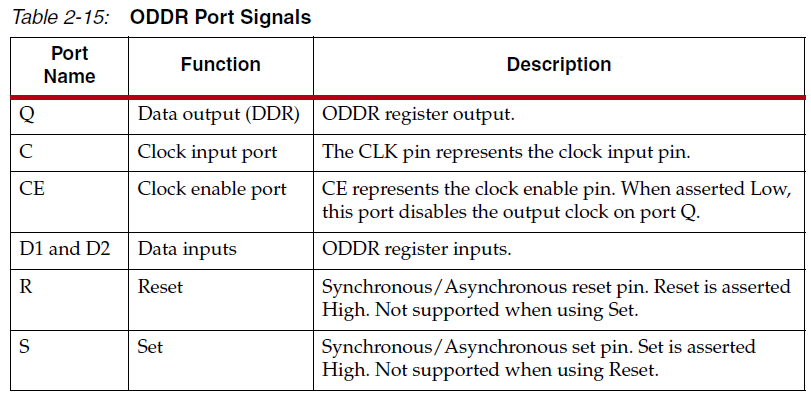
ODDR은 FPGA 외부에 있는 디바이스가 클럭의 상승시점과 하강시점에서 데이터를 읽는 디바이스와 효과적으로 인터페이스 하기 위해서 개발되었습니다.

다음 그림은 ODDR에 대한 블록도 입니다.

2개의 입력 (D1, D2)이 클럭(C)의 상승시점 하강시점에 출력 (Q) 됩니다. CE는 clock enable 이며 각각 S (set)와 R (reset) 포트가 추가되었습니다.



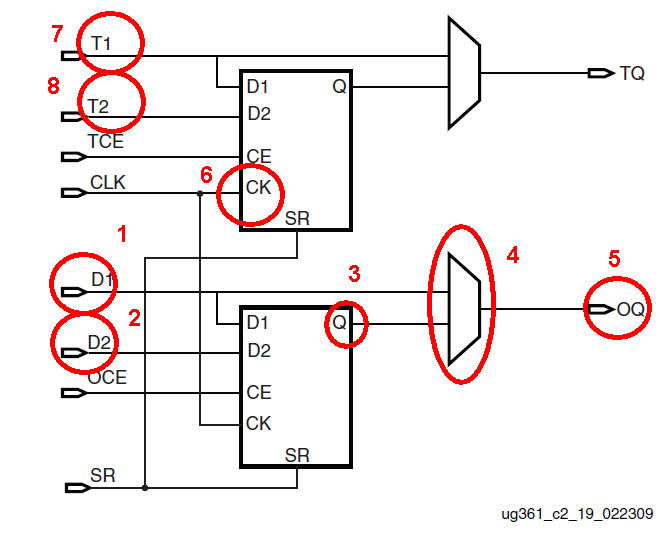
다음 그림은 ODDR의 포트에 대한 설명 입니다.



T1, T2 (7, 8번)을 같이 그린 것은 데이터를 출력할 때 때때로 하이 임피던스 신호를 출력해야 할 경우가 있기 때문 입니다

ODDR은 IDDR과는 다르게 2개의 FF의 입력단(1번, 2번) 에 있는 데이터를 클럭(6번)의 상승 시점과 하강 시점에 맞추어서 출력 핀(5번)으로 보내는 역할을 합니다.

ODDR을 쓰지 않는 경우에는 D1 (1번)이 MUX를 거쳐서 출력핀으로 전달 됩니다.



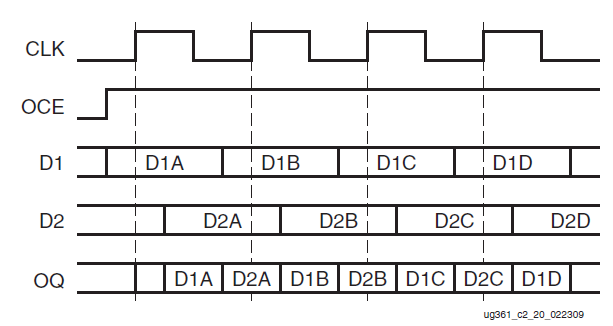
.

### ODDR 타이밍도

ODDR을 이용할 경우 ODDR/D1, D2로부터 데이터를 출력하는 방법은 다음과 같이 2가지가 있습니다.

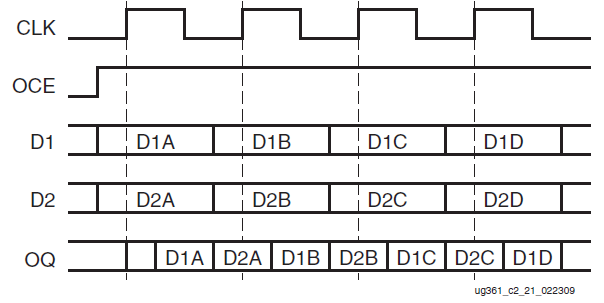
#### OPPOSITE\_EDGE Mode

다음 그림은 출력할 데이터 D1, D2이 ODDR에 입력되는 시점이 각각 상승시점와 하강시점에 전달되었을 때 출력 (OQ)으로 전달되는 모습 입니다.



#### SAME\_EDGE Mode

다음 그림은 클럭의 상승시점에만 데이터 D1, D2를 전달하면 ODDR의 OQ로 데이터가 전달되는 것을 보여 줍니다.



#### TEST BENCH

다음 리스트는 ODDR의 기능을 점검하기 위한 테스트 벤치 입니다.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_unsigned.ALL;

USE ieee.numeric\_std.ALL;

ENTITY tb001 IS

END tb001;

ARCHITECTURE behavior OF tb001 IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT ODDR\_test

PORT(

clkin\_p : IN std\_logic;

clkin\_n : IN std\_logic;

dina : IN std\_logic\_vector(7 downto 0);

dinb : IN std\_logic\_vector(7 downto 0);

dout\_p : OUT std\_logic\_vector(7 downto 0);

dout\_n : OUT std\_logic\_vector(7 downto 0)

);

END COMPONENT;

--Inputs

signal clkin\_p : std\_logic := '0';

signal clkin\_n : std\_logic := '0';

signal dina : std\_logic\_vector(7 downto 0) := (others => '0');

signal dinb : std\_logic\_vector(7 downto 0) := (others => '0');

--Outputs

signal dout\_p : std\_logic\_vector(7 downto 0);

signal dout\_n : std\_logic\_vector(7 downto 0);

signal en : std\_logic := '0';

signal tclk : std\_logic := '0';

signal tempa : std\_logic\_vector(7 downto 0) := "00000000";

signal tempb : std\_logic\_vector(7 downto 0) := "00000001";

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: ODDR\_test PORT MAP (

clkin\_p => clkin\_p,

clkin\_n => clkin\_n,

dina => dina,

dinb => dinb,

dout\_p => dout\_p,

dout\_n => dout\_n

);

-- Clock process definitions

clkin\_p\_process :process

begin

clkin\_p <= '0';

wait for 5 ns;

clkin\_p <= '1';

wait for 5 ns;

end process;

clkin\_n <= clkin\_p;

en <= '1';--, '1' after 102.5 ns;

tclk <= not ( tclk and en ) after 5 ns;

dina\_gen :process(tclk)

begin

if falling\_edge(tclk) then

tempa <= tempa + 2;

end if;

end process;

dina <= tempa;

---OPPOSITE\_EDGE

dinb\_gen :process(tclk)

begin

if rising\_edge(tclk) then

tempb <= tempb + 2;

end if;

end process;

---SAME\_EDGE

-- dinb\_gen :process(tclk)

-- begin

-- if falling\_edge(tclk) then

-- tempb <= tempb + 2;

-- end if;

-- end process;

dinb <= tempb;

END;

#### ODDR\_TEST.VHD

다음 리스트는 ODDR를 사용한 로직을 구성한 코드 입니다.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.NUMERIC\_STD.ALL;

library UNISIM;

use UNISIM.VComponents.all;

entity ODDR\_test is

port(

clkin\_p : in std\_logic;

clkin\_n : in std\_logic;

dina : in std\_logic\_vector(7 downto 0);

dinb : in std\_logic\_vector(7 downto 0);

dout\_p : out std\_logic\_vector(7 downto 0);

dout\_n : out std\_logic\_vector(7 downto 0)

);

end ODDR\_test;

architecture Behavioral of ODDR\_test is

signal clkin : std\_logic;

signal dout : std\_logic\_vector(7 downto 0);

begin

IBUFGDS\_U0 : IBUFGDS

generic map (

DIFF\_TERM => FALSE,

IBUF\_LOW\_PWR => TRUE,

IOSTANDARD => "LVDS\_25" )

port map (

I => clkin\_p,

IB => clkin\_n,

O => clkin );

loop0 : for i in 7 downto 0 generate

ODDR\_inst : ODDR

generic map(

DDR\_CLK\_EDGE => "OPPOSITE\_EDGE", -- "OPPOSITE\_EDGE" or "SAME\_EDGE"

INIT => '0', -- Initial value for Q port ('1' or '0')

SRTYPE => "SYNC") -- Reset Type ("ASYNC" or "SYNC")

port map (

S => '0', -- 1-bit set input

R => '0', -- 1-bit reset input

CE => '1', -- 1-bit clock enable input

C => clkin, -- 1-bit clock input

D1 => dina(i), -- 1-bit data input (positive edge)

D2 => dinb(i), -- 1-bit data input (negative edge)

Q => dout(i) -- 1-bit DDR output

);

OBUFDS\_inst : OBUFDS

generic map (

IOSTANDARD => "DEFAULT")

port map (

I => dout(i), -- Buffer input

O => dout\_p(i), -- Diff\_p output (connect directly to top-level port)

OB => dout\_n(i) -- Diff\_n output (connect directly to top-level port)

);

end generate;

end Behavioral;

#### IBUFGDS

먼저 IBUFGDS에 대해서 살펴 보겠습니다. IBUFGDS는 FPGA 외부에서 differential clock으로 클럭을 공급할 때 FPGA 내부에서 single endede clock으로 바꿔주는 역할을 합니다.

I와 IB에 각각 differential clock의 positive, negative clock 신호를 연결 합니다. 포트 O에서는 single endede clock 이 출력 됩니다.

일반적으로 single endede clock의 파형은 differential clock의 differential clock의 positive clock 신호와 같습니다.

IBUFGDS\_U0 : IBUFGDS

generic map (

DIFF\_TERM => FALSE,

IBUF\_LOW\_PWR => TRUE,

IOSTANDARD => "LVDS\_25" )

port map (

I => clkin\_p,

IB => clkin\_n,

O => clkin );

#### OBUFDS

다음 리스트는 ODDR에서 나오는 데이터를 differential data 로 출력하는 기능을 하는 OBUFDS 에 대한 설명 입니다.

ODDR의 출력 dout(i)이 OBUFDS의 입력 I에 연결되고 OBUFDS의 출력 O와 OB를 통해 FPGA 외부로 출력 됩니다.

OBUFDS\_inst : OBUFDS

generic map (

IOSTANDARD => "DEFAULT")

port map (

I => dout(i), -- Buffer input

O => dout\_p(i), -- Diff\_p output (connect directly to top-level port)

OB => dout\_n(i) -- Diff\_n output (connect directly to top-level port)

);

#### ODDR

다음 리스트는 두 개의 데이터 D1, D2이 클럭의 상승시점과 하강시점에 출력Q 로 연결되도록 하는 ODDR에 대한 설명 입니다.

여기에는 generic 문이 있어서 ODDR의 2가지 사용모드중 하나를 선택할 수 있도록 해 줍니다.

ODDR\_inst : ODDR

generic map(

DDR\_CLK\_EDGE => "OPPOSITE\_EDGE", -- "OPPOSITE\_EDGE" or "SAME\_EDGE"

INIT => '0', -- Initial value for Q port ('1' or '0')

SRTYPE => "SYNC") -- Reset Type ("ASYNC" or "SYNC")

port map (

S => '0', -- 1-bit set input

R => '0', -- 1-bit reset input

CE => '1', -- 1-bit clock enable input

C => clkin, -- 1-bit clock input

D1 => dina(i), -- 1-bit data input (positive edge)

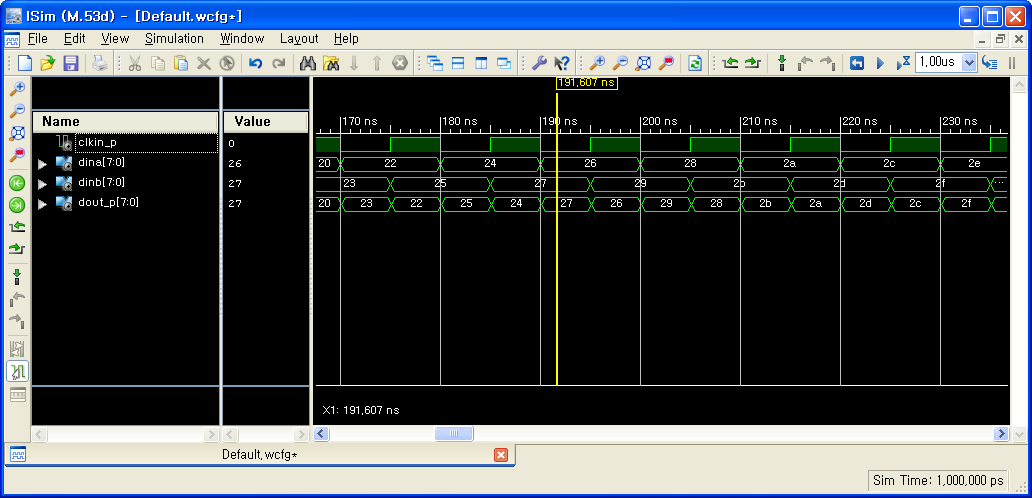
D2 => dinb(i), -- 1-bit data input (negative edge)

Q => dout(i) -- 1-bit DDR output

);

#### OPPOSITE\_EDGE 시뮬레이션

다음 그림은 클럭의 상승시점과 하강시점에 데이터를 각각 전달 했을 때 dout\_p로 데이터가 출력되는 것을 확인할 수 있습니다.



#### SAME\_EDGE 시뮬레이션

SAME\_EDGE 를 사용하기 위해서는 데이터를 전달하는 방식을 바꿔야 합니다. 다음 리스트는 수정된 테스트 벤치 입니다.

---OPPOSITE\_EDGE

-- dinb\_gen :process(tclk)

-- begin

-- if rising\_edge(tclk) then

-- tempb <= tempb + 2;

-- end if;

-- end process;

----SAME\_EDGE

dinb\_gen :process(tclk)

begin

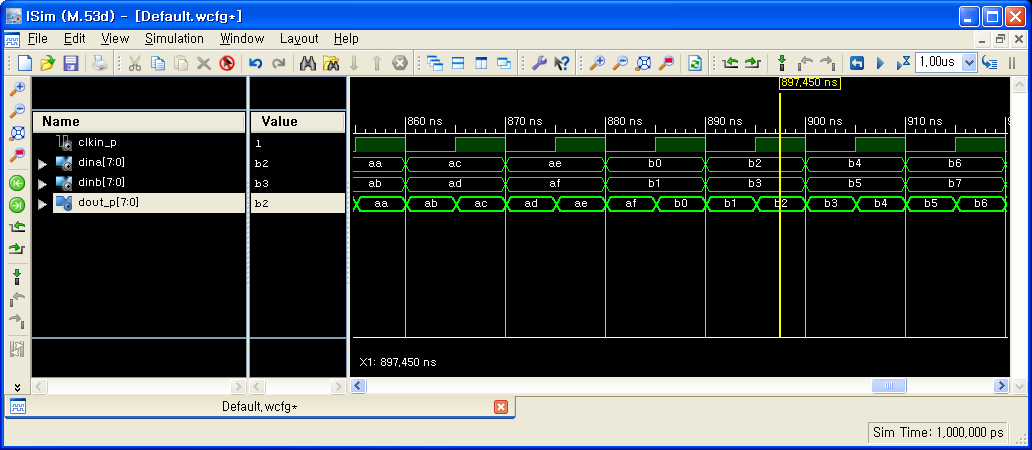
if falling\_edge(tclk) then

tempb <= tempb + 2;

end if;

end process;

다음 그림은ODDR의 SAME\_EDGE Mode를 시뮬레이션한 파형 입니다.



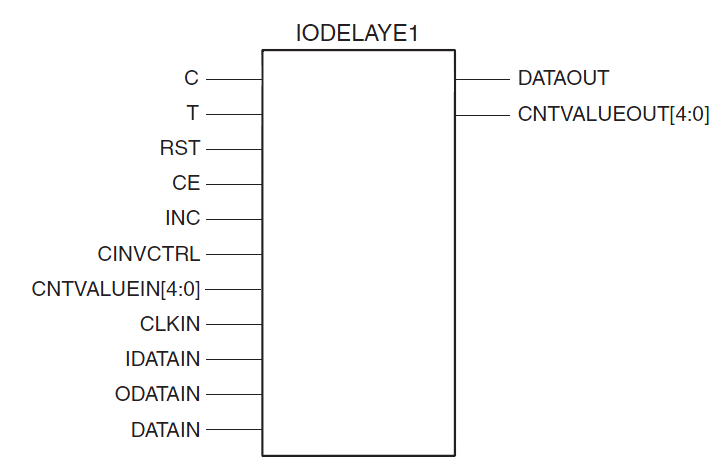
## IDELAY

고속 데이터 인터페이스를 하게 되면 많은 개발자들은 충분하지 못한 셋업 타임과 홀드 타임 때문에 고생을 하게 됩니다.

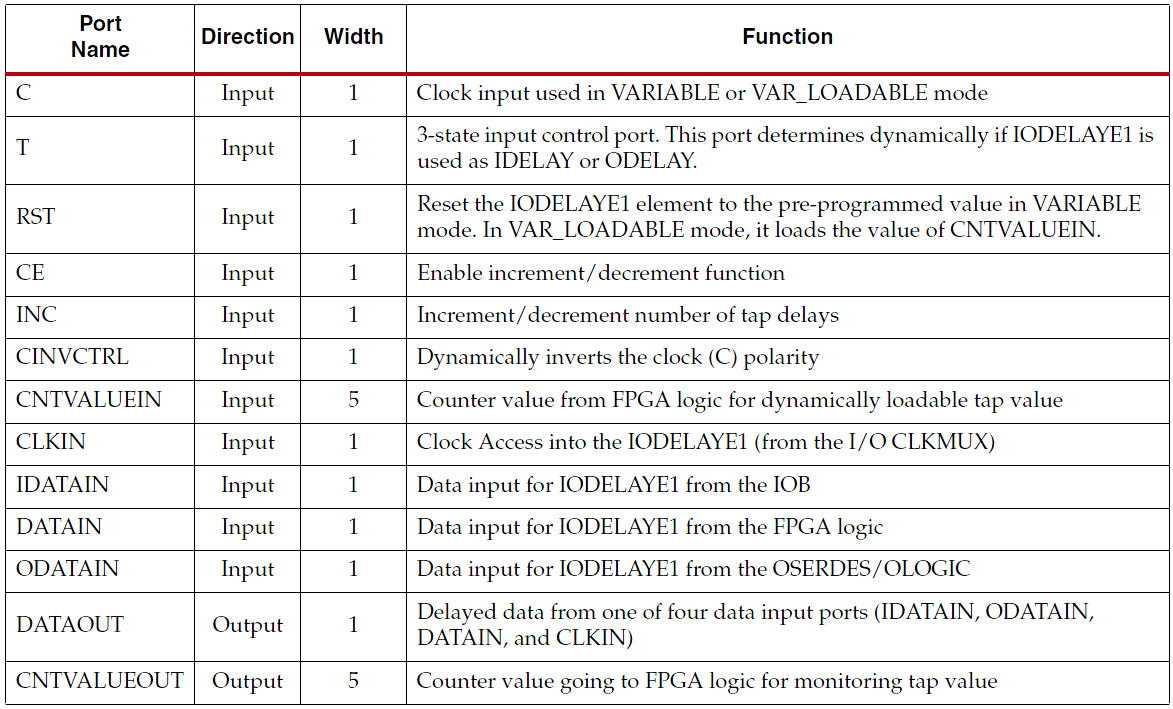
사실 셋업 타임과 홀드 타임이 모자란다고 하기 보다는 클럭의 상승시점이 data valid 구간의 중간에 있어야 하는데 너무 앞쪽에 있거나 너무 뒤쪽에 있으면 처음에는 좀 동작하다가도 나중에 데이터가 깨지는 경우가 많이 발생 합니다.

이럴 경우에는 클럭의 상승시점을 조정할 수 있어서 클럭의 상승시점 시점을 우리가 원하는 곳을 밀거나 당길 수 있으면 FPGA 외부 환경 변화에도 데이터의 신뢰성을 높이는데 많은 도움을 줄 수 있습니다.

특정한 데이터의 딜레이를 조정할 수 있는 것이 바로 IDELAY 입니다.

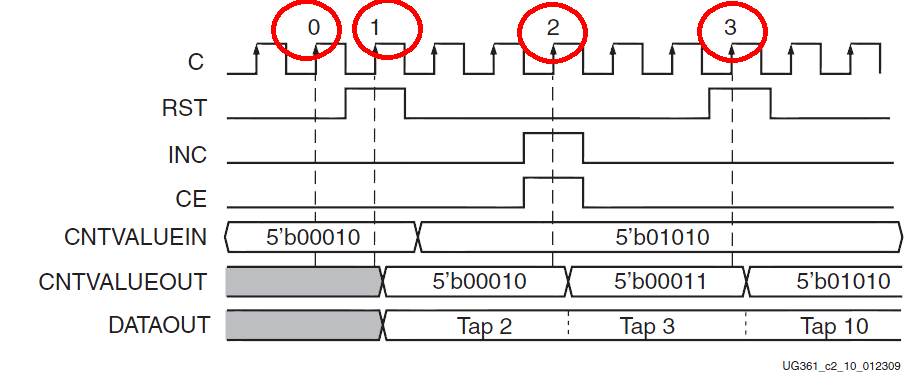


다음 그림은 IDELAY의 포트에 대한 설명 입니다.



음... 보기에도 좀 숨막혀 보이는 알 수 없는 여러 포트 이름 같이 보이지만 하나 하나 살펴 보면 다 의미가 있습니다.

다음 그림을 먼저 살펴봐 주시기 바랍니다.



이 그림에서 예상할 수 있는 것은 RST가 assert 되면 (1번, 3번) CNTVALUEIN 에 지정된 값으로 TAP의 개수가 지정된다는 것 입니다.

여기서 TAP에 대해서 잠시 설명하도록 하겠습니다.

TAP은 FPGA 핀으로부터 입력신호나 출력 신호 (V6의 경우에만 적용)에 대해 원하는 만큼 딜레이를 주기 위한 기본 단위 입니다.

TAP의 개수는 FPGA 마다 다르며 하나의 TAP에서 만들수 있는 딜레이는 데이터 쉬트를 참조해야 합니다.

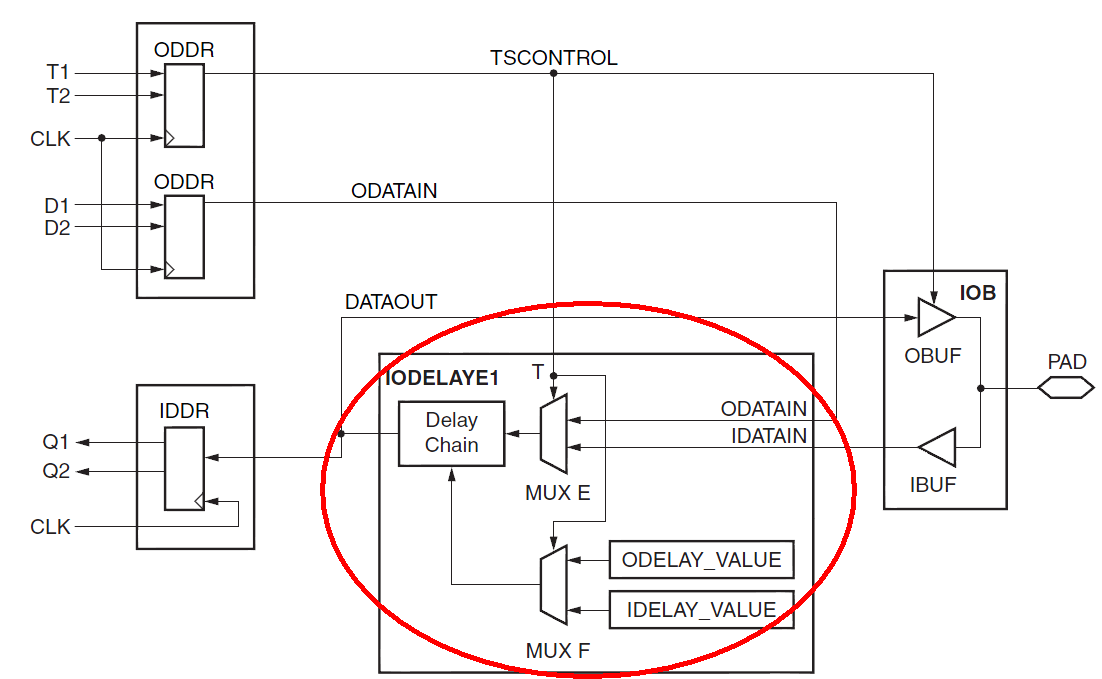
하여간 자일링스 VIRTEX6의 모든 IO에는 IODELAY1 이라는 딜레이 소자가 포함되어 있습니다.

중요한 것은 IODELAY가 VIRTEX5 에서는 입력쪽에 대해서만 사용할 수 있었는데 VIRTEX6에서는 입출력 모두 사용가능 합니다.

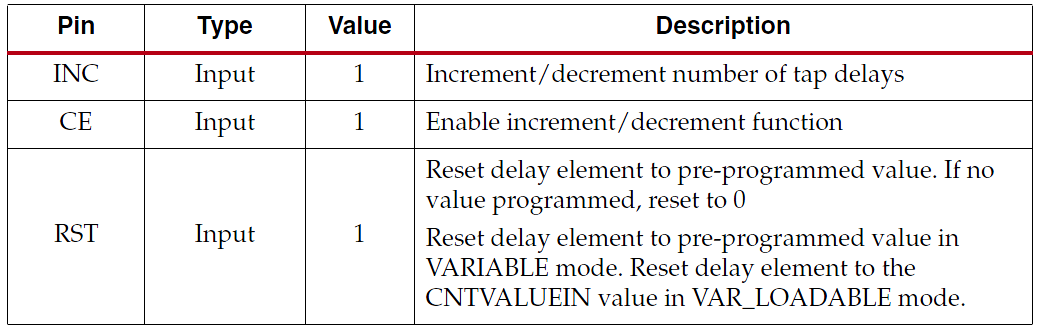
좀더 구체적으로 보면 IODELAY1은 ILOGIC이나 ISERDES 및 OLOGIC 및 OSERDES에 연결될 수 있습니다.

IODELAY1에는 모두 32개의 딜레이 TAP이 있으며 TAP의 정밀도는 데이터 쉬트를 참조해야 합니다.

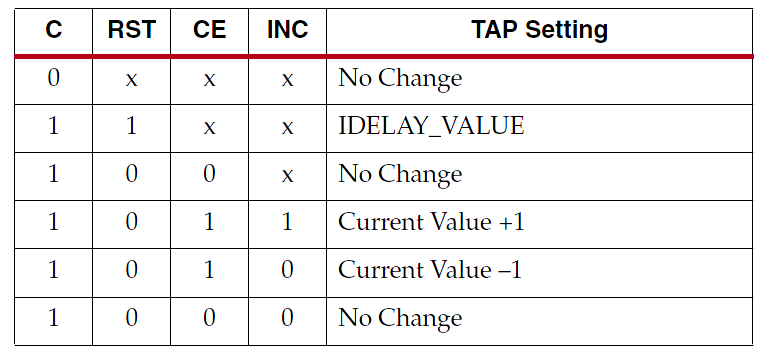
다음 그림을 보면 IDELAY1의 구체적인 입출력 포트의 연결관계를 확인 할 수 있습니다.



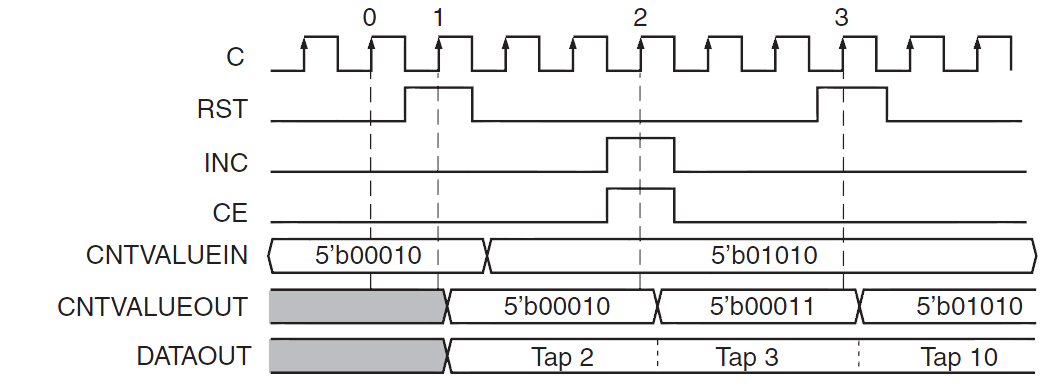
다음 그림은IODELAY1의 딜레이 값을 정하기 위해 사용하는 포트에 대한 설명 입니다.



다음 그림은 IODELAY1의 딜레이 값을 바꾸기 위한 포트들의 동작상태에 대한 설명 입니다.



이렇게 IODELAY1 의 딜레이 값을 바꾸기위한 포트를 이해하면 다음 그림을 이해할 수 있습니다.



#### IDELAYCTRL

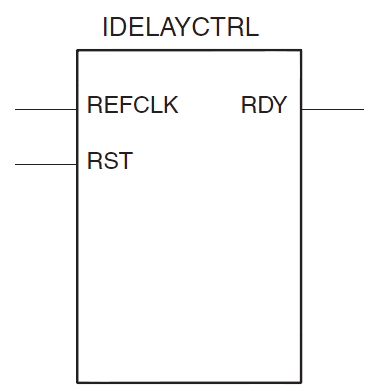
일반적으로IDELAYCTRL의 사용법을 정확히 이해하고 있는 개발자는 많지 않습니다. 하지만 IODELAY1이나 앞으로 설명할 ISERDES의 속성이 FIXED, VARIABLE, VAR\_LOADABLE인 경우에는 반드시 IDELAYCTR을 같이 사용해야 합니다.

비록 개발자가 IODELAY1의 최적의 TAP의 숫자를 찾아서 결정했다고 하더라도 실제 딜레이 값은 process, voltage 및 온도의 변화에 따라 계속 변하게 됩니다.

이런 변화를 추적해서 계속 보상해줘야 하기 위해서는 무엇인가가 필요한데 이것이 바로 IDELAYCTRL 입니다.

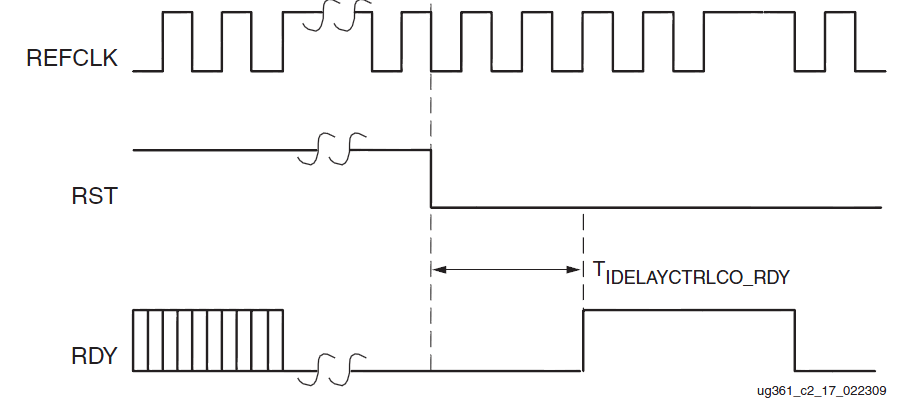
다음 그림은 IDELAYCTRL의 블록도 입니다.

사실 IDELAYCTRL의 내부 블록에 대해서는 잘 알려진 부분은 없습니다. 다만 IDELAYCTRL이 제대로 동작하기 위해서 자일링스는 REFCLK으로 200Mhz를 공급하도록 되어 있습니다.

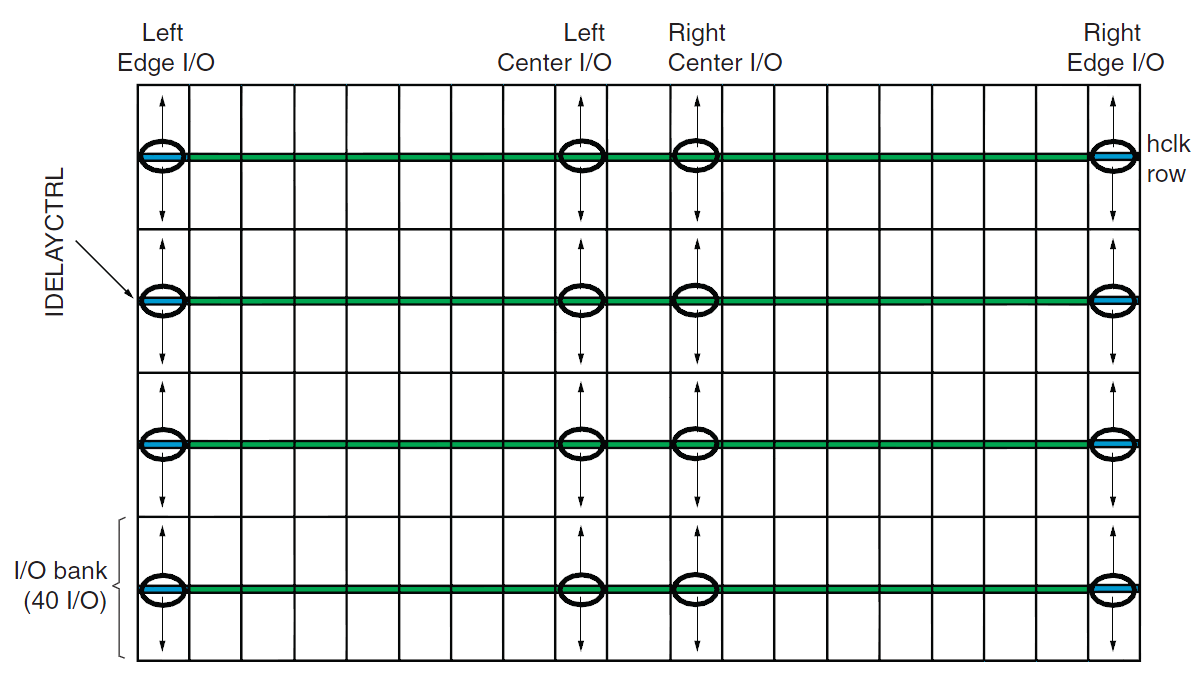


IDELAYCTRL은 리셋이 deassert 된 후부터 자신이 관리하는 IO 그룹 의 IODELAY1의 딜레이 값을 조정한 후 RDY 신호를 assert 합니다.

다음 그림은 IDELAYCTRL의 동작과정을 보여주고 있습니다.



다음 그림은 이러한 IDELAYCTRLD 이 FPGA 어디에 있으며 하나의 IDELAYCTRL이 적용되는 범위는 어떻게 되는지 보여 줍니다.



#### 테스트 벤치 코드

다음 리스트는 IODELAY의 기능을 점검하기 위한 테스트 벤치 입니다.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

ENTITY tb002 IS

END tb002;

ARCHITECTURE behavior OF tb002 IS

COMPONENT IODELAY\_test

PORT(

CE : IN std\_logic;

INC : IN std\_logic;

C : IN std\_logic;

REFCLK : IN std\_logic;

RST : IN std\_logic;

din : IN std\_logic;

dout : OUT std\_logic;

CNTVALUEOUT : OUT std\_logic\_vector(4 downto 0)

);

END COMPONENT;

--Inputs

signal CE : std\_logic := '0';

signal INC : std\_logic := '0';

signal C : std\_logic := '0';

signal REFCLK : std\_logic := '0';

signal RST : std\_logic := '0';

signal din : std\_logic := '0';

--Outputs

signal dout : std\_logic;

signal CNTVALUEOUT : std\_logic\_vector(4 downto 0);

BEGIN

uut: IODELAY\_test PORT MAP (

CE => CE,

INC => INC,

C => C,

REFCLK => REFCLK,

RST => RST,

din => din,

dout => dout,

CNTVALUEOUT => CNTVALUEOUT

);

REFCLK\_process :process

begin

REFCLK <= '0';

wait for 2.5 ns;

REFCLK <= '1';

wait for 2.5 ns;

end process;

C\_process :process

begin

C <= '0';

wait for 10 ns;

C <= '1';

wait for 10 ns;

end process;

ce <= '0', '1' after 300 ns, '0' after 320 ns, '1' after 500 ns, '0' after 520 ns;

inc <= '0', '1' after 300 ns, '0' after 320 ns;

din\_process :process

begin

din <= '0';

wait for 10 ns;

din <= '1';

wait for 200 ns;

end process;

END;

#### 200Mhz 클럭

다음 리스트는 IDELAYCTRL에 200Mhz를 공급하기 위한 테스트 벤치 입니다.

REFCLK\_process :process

begin

REFCLK <= '0';

wait for 2.5 ns;

REFCLK <= '1';

wait for 2.5 ns;

end process;

#### IODELAY1 제어 신호

다음 리스트는 IODELAY1의 CLK, CE, INC 및 초기 카운터 값을 정하기 위한 테스트 벤치 입니다.

C\_process :process

begin

C <= '0';

wait for 10 ns;

C <= '1';

wait for 10 ns;

end process;

ce <= '0', '1' after 300 ns, '0' after 320 ns, '1' after 500 ns, '0' after 520 ns;

inc <= '0', '1' after 300 ns, '0' after 320 ns;

din\_process :process

begin

din <= '0';

wait for 10 ns;

din <= '1';

wait for 200 ns;

end process;

### IODELAY1

#### IODELAY1 전체 코드

다음 리스트는 IODELAY1를 사용하는 방법에 대해서 보여 주고 있습니다.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

library UNISIM;

use UNISIM.VComponents.all;

entity IODELAY\_test is

Port (

CE : in STD\_LOGIC;

INC : in STD\_LOGIC;

C : in STD\_LOGIC;

REFCLK : in STD\_LOGIC;--200MHz

RST : in STD\_LOGIC;

din : in STD\_LOGIC;

dout : out STD\_LOGIC;

CNTVALUEOUT : out std\_logic\_vector(4 downto 0)

);

end IODELAY\_test;

architecture Behavioral of IODELAY\_test is

signal temp : std\_logic;

attribute IODELAY\_GROUP : string;

attribute IODELAY\_GROUP of IDELAYCTRL\_U0: label is "GRP0";

attribute IODELAY\_GROUP of IODELAYE00\_U0: label is "GRP0";

begin

IDELAYCTRL\_U0 : IDELAYCTRL

port map (

RDY => open, -- 1-bit output indicates validity of the REFCLK

REFCLK => REFCLK, -- 1-bit reference clock input

RST => RST -- 1-bit reset input

);

IODELAYE00\_U0: IODELAYE1

GENERIC MAP(

IDELAY\_TYPE => "VARIABLE",

IDELAY\_VALUE => 10,

ODELAY\_VALUE => 0,

REFCLK\_FREQUENCY => 200.00,

DELAY\_SRC => "I",

HIGH\_PERFORMANCE\_MODE => TRUE,

SIGNAL\_PATTERN => "DATA")

PORT MAP (

DATAOUT => dout,

IDATAIN => din,

ODATAIN => '0',

DATAIN => '0',

T => '0',

CINVCTRL => '0',

CLKIN => '0',

CNTVALUEIN => "00000",

CNTVALUEOUT => CNTVALUEOUT,

CE => CE,

INC => INC,

C => C,

RST => RST);

end Behavioral;

#### Attribute

현재 사용하는 IODELAY1과 IDELAYCTRL이 하나의 클럭 범위 내에 존재시키기 위해서는 2개의 블록 (IODELAY1, IDELAYCTRL) 이 하나의 그룹을 형성한다는 것을 알려 줘야 합니다.

이렇게 그룹을 형성시키기 위해서는 다음 리스트와 같이 attrbiute를 통해 지정 합니다.

architecture Behavioral of IODELAY\_test is

attribute IODELAY\_GROUP : string;

attribute IODELAY\_GROUP of IDELAYCTRL\_U0: label is "GRP0";

attribute IODELAY\_GROUP of IODELAYE00\_U0: label is "GRP0";

begin

#### IDELAYCTRL

다음 리스트는 IDELAYCTRL에 200Mhz 과 리셋을 연결한 것을 보여주고 있습니다.

IDELAYCTRL\_U0 : IDELAYCTRL

port map (

RDY => open, -- 1-bit output indicates validity of the REFCLK

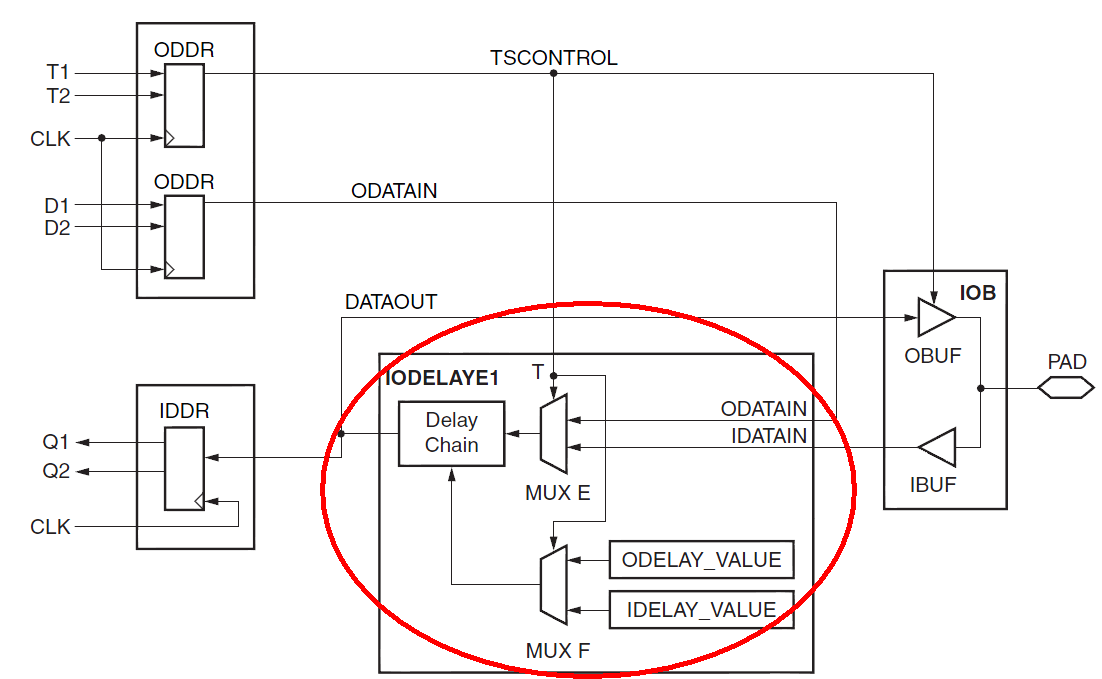
REFCLK => REFCLK, -- 1-bit reference clock input

RST => RST -- 1-bit reset input

);

#### IODELAY1

앞에서 살펴 보았던 그림을 다시 한번 보도록 하겠습니다.



IODELAY1 블록 내부에는 여러가지 MUX가 존재한다는 것을 알 수 있습니다. 그중 대표적인 것이

딜레이 소스를 OUTPUT과 INPUT 중 하나를 선택해야 합니다. 두번째는 ODELAY\_VALUE와 IDELAY\_VALUE를 결정해야 합니다.

이러한 값을 결정하기 위해서 IODELAY1는 GENERIC\_MAP을 사용합니다.

GENERIC MAP(

IDELAY\_TYPE => "VARIABLE",

IDELAY\_VALUE => 10,

ODELAY\_VALUE => 0,

REFCLK\_FREQUENCY => 200.00,

DELAY\_SRC => "I",

HIGH\_PERFORMANCE\_MODE => TRUE,

SIGNAL\_PATTERN => "DATA")

이제 IODELAY1의 포트에 대해서 살펴보겠습니다. 먼저 dout과 din은 FPGA의 입출력 포트 입니다. 나머지 CNTVALUEIN, CNTVALUEOUT, CE, INC, C, RST는 외부 테스트 벤치에서 공급 합니다.

PORT MAP (

DATAOUT => dout,

IDATAIN => din,

ODATAIN => '0',

DATAIN => '0',

T => '0',

CINVCTRL => '0',

CLKIN => '0',

CNTVALUEIN => "00000",

CNTVALUEOUT => CNTVALUEOUT,

CE => CE,

INC => INC,

C => C,

RST => RST);

다음 리스트는 전체 IODELAY1에 대한 GENERIC문과 포트맵을 보여주고 있습니다.

IODELAYE00\_U0: IODELAYE1

GENERIC MAP(

IDELAY\_TYPE => "VARIABLE",

IDELAY\_VALUE => 10,

ODELAY\_VALUE => 0,

REFCLK\_FREQUENCY => 200.00,

DELAY\_SRC => "I",

HIGH\_PERFORMANCE\_MODE => TRUE,

SIGNAL\_PATTERN => "DATA")

PORT MAP (

DATAOUT => dout,

IDATAIN => din,

ODATAIN => '0',

DATAIN => '0',

T => '0',

CINVCTRL => '0',

CLKIN => '0',

CNTVALUEIN => "00000",

CNTVALUEOUT => CNTVALUEOUT,

CE => CE,

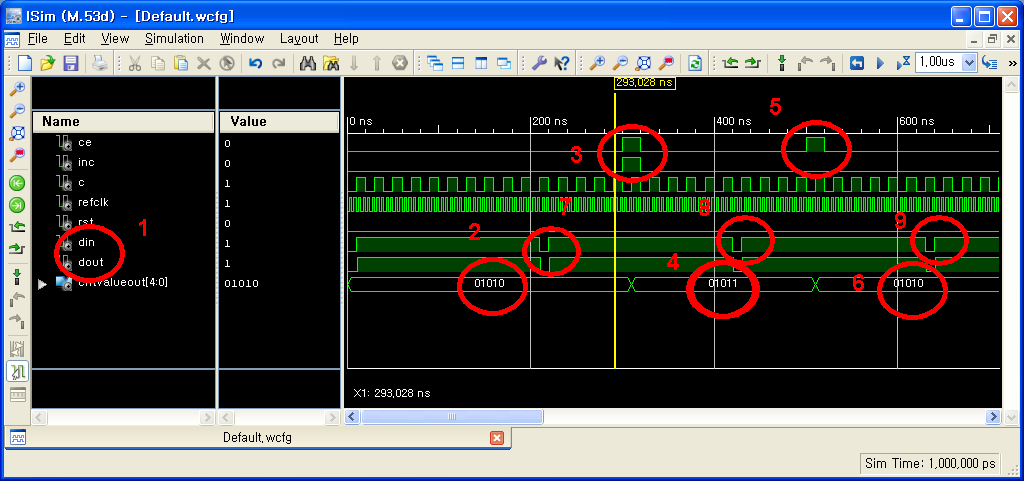
INC => INC,

C => C,

RST => RST);

#### 시뮬레이션 결과

다음 그림은 IODELAY1의 동작상태를 시뮬레이션한 결과 입니다.



단순히 이 그림만 보면 뭐가 어떤게 달라진다는 것을 확실히 파악하기 쉽지 않습니다.

일단 그림을 보면 DIN이 입력되면 IODELAY1을 통해 DOUT으로 출력 (1번) 된다는 것을 볼 수 있>

일단 CNTVALUE의 초기값 (2번)은 2이지만 C, CE, INC에 의해서 (3 번) CNTVALUE가 3으로 (4번)으로 변한 것을 확인할 수 있습니다.

다시 C, CE, INC(5번)에 의해서 CNTVALUE는 3에서 2로 (6번)로 바뀌었습니다.

### IODELAY1의 OUTPUT DELAY

지금까지 IODELAY1을 사용하여 Input 쪽에 Delay를 주는 방법에 대하여 설명하였습니다.

다음은 IODELAY1을 사용하여 Output 쪽에 Delay를 주는 방법에 대하여 설명하겠습니다.

전체 code는 다음과 같습니다.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

library UNISIM;

use UNISIM.VComponents.all;

entity IODELAY\_test2 is

Port (

CE : in STD\_LOGIC;

INC : in STD\_LOGIC;

C : in STD\_LOGIC;

REFCLK : in STD\_LOGIC;--200MHz

RST : in STD\_LOGIC;

din : in STD\_LOGIC;

dout : out STD\_LOGIC;

CNTVALUEOUT : out std\_logic\_vector(4 downto 0)

);

end IODELAY\_test2;

architecture Behavioral of IODELAY\_test2 is

attribute IODELAY\_GROUP : string;

attribute IODELAY\_GROUP of IDELAYCTRL\_U0: label is "GRP0";

attribute IODELAY\_GROUP of IODELAYE00\_U0: label is "GRP0";

begin

IDELAYCTRL\_U0 : IDELAYCTRL

port map (

RDY => open, -- 1-bit output indicates validity of the REFCLK

REFCLK => REFCLK, -- 1-bit reference clock input

RST => RST -- 1-bit reset input

);

IODELAYE00\_U0: IODELAYE1

GENERIC MAP(

ODELAY\_TYPE => "VARIABLE",

IDELAY\_VALUE => 0,

ODELAY\_VALUE => 10,

REFCLK\_FREQUENCY => 200.00,

DELAY\_SRC => "O",

HIGH\_PERFORMANCE\_MODE => TRUE,

SIGNAL\_PATTERN => "DATA")

PORT MAP (

DATAOUT => dout,

IDATAIN => '0',

ODATAIN => din,

DATAIN => '0',

T => '0',

CINVCTRL => '0',

CLKIN => '0',

CNTVALUEIN => "00000",

CNTVALUEOUT => CNTVALUEOUT,

CE => CE,

INC => INC,

C => C,

RST => RST);

end Behavioral;

앞서의 Code와 동일하게 IDELAYCTRL, IODELAY1이 하나의 클럭 범위 내에 존재시키기 위해서는 2개의 블록 (IODELAY1, IDELAYCTRL) 이 하나의 그룹을 형성한다는 것을 알려 줘야 합니다.

이렇게 그룹을 형성시키기 위해서는 다음 리스트와 같이 attrbiute를 통해 지정 합니다.

architecture Behavioral of IODELAY\_test2 is

attribute IODELAY\_GROUP : string;

attribute IODELAY\_GROUP of IDELAYCTRL\_U0: label is "GRP0";

attribute IODELAY\_GROUP of IODELAYE00\_U0: label is "GRP0";

begin

앞서의 Code와 동일하게 다음리스트는 IDELAYCTRL에 200Mhz 과 리셋을 연결한 것을 보여주고 있습니다.

IDELAYCTRL\_U0 : IDELAYCTRL

port map (

RDY => open, -- 1-bit output indicates validity of the REFCLK

REFCLK => REFCLK, -- 1-bit reference clock input

RST => RST -- 1-bit reset input

);

다음 리스트는 전체 IODELAY1에 대한 GENERIC문과 포트맵을 보여주고 있습니다.

앞서의 Code와 비교하였을 때 아래의 빨간 색 글자 부분에 차이가 있습니다.

어느 Port에 연결되어지는지에 따라 Input 혹은 Output에 각 delay를 적용할 수 있습니다.

Input 혹은 Output 모드의 결정은 Generic 선언 부의 DELAY\_SRC의 Value가 I 혹은 O에 따라 결정지어집니다.

IODELAYE00\_U0: IODELAYE1

GENERIC MAP(

ODELAY\_TYPE => "VARIABLE",

IDELAY\_VALUE => 0,

ODELAY\_VALUE => 10,

REFCLK\_FREQUENCY => 200.00,

**DELAY\_SRC => "O",**

HIGH\_PERFORMANCE\_MODE => TRUE,

SIGNAL\_PATTERN => "DATA")

PORT MAP (

**DATAOUT => dout,**

**IDATAIN => '0',**

**ODATAIN => din,**

**DATAIN => '0',**

T => '0',

CINVCTRL => '0',

CLKIN => '0',

CNTVALUEIN => "00000",

CNTVALUEOUT => CNTVALUEOUT,

CE => CE,

INC => INC,

C => C,

RST => RST);

다음 리스트는 위 Code를 simulation하기 위한 testbench file입니다.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

ENTITY tb001 IS

END tb001;

ARCHITECTURE behavior OF tb001 IS

COMPONENT IODELAY\_test2

PORT(

CE : IN std\_logic;

INC : IN std\_logic;

C : IN std\_logic;

REFCLK : IN std\_logic;

RST : IN std\_logic;

din : IN std\_logic;

dout : OUT std\_logic;

CNTVALUEOUT : OUT std\_logic\_vector(4 downto 0)

);

END COMPONENT;

--Inputs

signal CE : std\_logic := '0';

signal INC : std\_logic := '0';

signal C : std\_logic := '0';

signal REFCLK : std\_logic := '0';

signal RST : std\_logic := '0';

signal din : std\_logic := '0';

--Outputs

signal dout : std\_logic;

signal CNTVALUEOUT : std\_logic\_vector(4 downto 0);

BEGIN

uut: IODELAY\_test2 PORT MAP (

CE => CE,

INC => INC,

C => C,

REFCLK => REFCLK,

RST => RST,

din => din,

dout => dout,

CNTVALUEOUT => CNTVALUEOUT

);

REFCLK\_process :process

begin

REFCLK <= '0';

wait for 2.5 ns;

REFCLK <= '1';

wait for 2.5 ns;

end process;

C\_process :process

begin

C <= '0';

wait for 10 ns;

C <= '1';

wait for 10 ns;

end process;

ce <= '0', '1' after 300 ns, '0' after 320 ns, '1' after 500 ns, '0' after 520 ns;

inc <= '0', '1' after 300 ns, '0' after 320 ns;

din\_process :process

begin

din <= '0';

wait for 10 ns;

din <= '1';

wait for 200 ns;

end process;

END;

200Mhz 클럭

다음 리스트는 IDELAYCTRL에 200Mhz를 공급하기 위한 테스트 벤치 입니다.

REFCLK\_process :process

begin

REFCLK <= '0';

wait for 2.5 ns;

REFCLK <= '1';

wait for 2.5 ns;

end process;

IODELAY1 제어 신호

다음 리스트는 IODELAY1의 CLK, CE, INC 및 초기 카운터 값을 정하기 위한 테스트 벤치 입니다.

C\_process :process

begin

C <= '0';

wait for 10 ns;

C <= '1';

wait for 10 ns;

end process;

ce <= '0', '1' after 300 ns, '0' after 320 ns, '1' after 500 ns, '0' after 520 ns;

inc <= '0', '1' after 300 ns, '0' after 320 ns;

din\_process :process

begin

din <= '0';

wait for 10 ns;

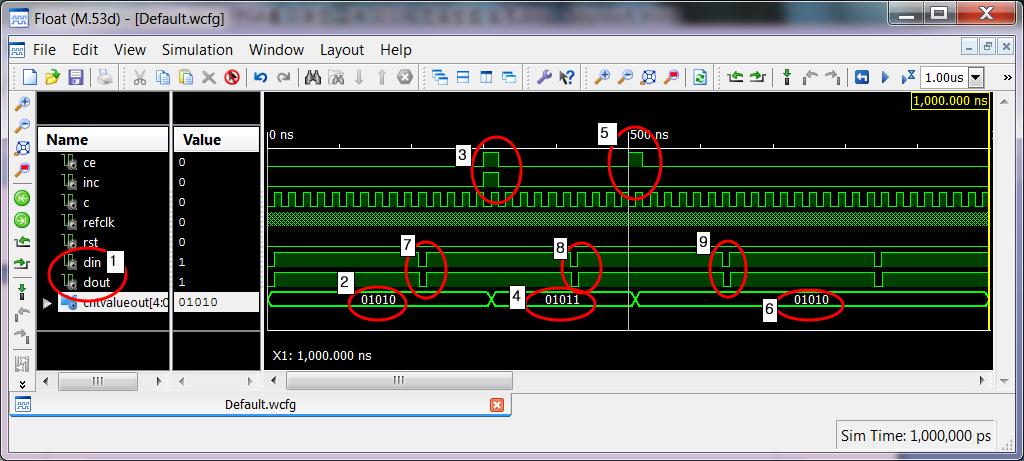
din <= '1';

wait for 200 ns;

end process;

시뮬레이션 결과

다음 그림은 IODELAY1의 동작상태를 시뮬레이션한 결과 입니다.



단순히 이 그림만 보면 뭐가 어떤게 달라진다는 것을 확실히 파악하기 쉽지 않습니다.

일단 그림을 보면 DIN이 입력되면 IODELAY1을 통해 DOUT으로 출력 (1번) 된다는 것을 볼 수 있습니다.

일단 CNTVALUE의 초기값 (2번)은 “01010”이지만 C, CE, INC에 의해서 (3 번) CNTVALUE가 “01011”으로 (4번)으로 변한 것을 확인할 수 있습니다.

다시 C, CE, INC(5번)에 의해서 CNTVALUE는 “01011”에서 “01010”로 (6번)로 바뀌었습니다.

이상 IODELAY를 사용하여 Input에 Delay를 적용하는 방법과 Output에 Delay를 적용하는 방법에 대하여 설명드렸습니다.

## ISERDES1

일반적으로 Serial-to-Parallel Logic을 일반 RTL code로도 구현은 가능합니다.

그러나, Serial Data에 동기되는 clock으로 200MHz 이상이 되는 Design의 경우에는 Timing 문제가 발생할 수 밖에 없습니다. 또한, ISE Compile(P&R) 할 때마다 Timing이 틀어지는 문제에 부딪치게 됩니다.( 각 Register들의 location 과 Routing path delay에 따라 Timing이 재각각일 수 밖에 없습니다. )

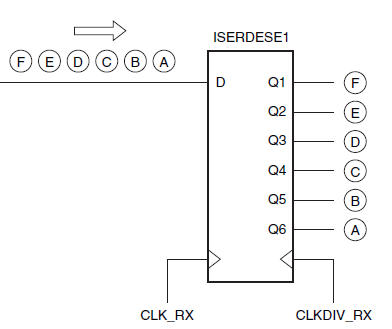
Manual location constraints와 manual Routing constraints를 사용하여 Timing 문제를 해결할 수도 있으나, 모든 Pin에 대하여 각각의 이러한 Constraints를 적용한다는 것은 매우 많은 시간과 번거로운 작업을 진행하여야 합니다.

이러한 문제를 해결하기 위하여 ISERDES1을 사용함으로써 여러가지 고민을 할 필요가 없게 됩니다.

다음은 Input Serial-to-Parallel Logic을 위하여 ISERDES1의 사용법에 대하여 설명 드리겠습니다.

### 기본 이론

일반적으로 Input Serial-to-Parallel Logic를 구현하기 위하여아래의 그림과 같이 ISERDESE1을 사용합니다.



또한 Clock 동기를 맞추기 위하여 CLK\_RX(1x clock)과 CLKDIV\_RX(1/6x clock)이 필요합니다.

위 그림의 ISERDESE1의 Serial-to-Parallel parameter는 6입니다.

실제 설명하게 될 ISERDESE1의 Serial-to-Parallel parameter는 4 입니다.

따라서 clock은 1x clock과 1/4x clock이 필요함을 알 수 있습니다.

### ISERDES1

아래의 Code는 ISERDESE1을 사용하여 Input Serial-to-Parallel Logic을 구현한 예입니다.

(Serial-to-Parallel parameter는 4 입니다.)

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

library UNISIM;

use UNISIM.VComponents.all;

entity ISERDES\_1to4\_test is

Port (

RESET : in std\_logic;

BITSLIP : in std\_logic;

inclk\_p : in std\_logic;

inclk\_n : in std\_logic;

sdatain\_p : in std\_logic;

sdatain\_n : in std\_logic;

pdataout : out std\_logic\_vector(3 downto 0)

);

end ISERDES\_1to4\_test;

architecture Behavioral of ISERDES\_1to4\_test is

signal Low : std\_logic;

signal High : std\_logic;

signal IBUFGDS\_OCLK : std\_logic;

signal BUFIO\_ICLK : std\_logic;

signal BUFIO\_OCLK : std\_logic;

signal BUFR\_ICLK : std\_logic;

signal BUFR\_OCLK : std\_logic;

signal Srds0sIn : std\_logic;

signal Srds0pOut : std\_logic\_vector(3 downto 0);

signal temp\_reg0 : std\_logic\_vector(3 downto 0);

signal temp\_reg1 : std\_logic\_vector(3 downto 0);

-- signal temp\_reg2 : std\_logic\_vector(3 downto 0);

begin

Low <= '0';

High <= '1';

IBUFGDS\_clk0 : IBUFGDS

generic map ( IOSTANDARD => "LVDS\_25" )

port map (

I => inclk\_p,

IB => inclk\_n,

O => IBUFGDS\_OCLK

);

BUFIO\_ICLK <= IBUFGDS\_OCLK;

BUFIO\_clk0 : BUFIO

port map (

I => BUFIO\_ICLK,

O => BUFIO\_OCLK

);

BUFR\_ICLK <= IBUFGDS\_OCLK;

BUFR\_clk0 : BUFR

generic map (

BUFR\_DIVIDE => "4", -- "BYPASS", "1", "2", "3", "4", "5", "6", "7", "8"

SIM\_DEVICE => "VIRTEX6") -- Specify target device, "VIRTEX4", "VIRTEX5", "VIRTEX6"

port map (

I => BUFR\_ICLK, -- Clock buffer input

O => BUFR\_OCLK, -- Clock buffer output

CE => High, -- Clock enable input

CLR => Low -- Clock buffer reset input

);

IBUFDS\_data0 : IBUFDS

generic map ( IOSTANDARD => "LVDS\_25" )

port map (

I => sdatain\_p,

IB => sdatain\_n,

O => Srds0sIn

);

I\_Isrds\_D0\_p : ISERDESE1

generic map (

SERDES\_MODE => "MASTER",

INTERFACE\_TYPE => "NETWORKING",

IOBDELAY => "NONE",

DATA\_RATE => "SDR",

DATA\_WIDTH => 4,

DYN\_CLKDIV\_INV\_EN => FALSE,

DYN\_CLK\_INV\_EN => FALSE,

NUM\_CE => 1,

OFB\_USED => FALSE

)

port map (

D => Srds0sIn, -- in

DDLY => Low, -- in

DYNCLKDIVSEL => Low, -- in

DYNCLKSEL => Low, -- in

OFB => Low, -- in

BITSLIP => BITSLIP, -- in

CE1 => High, -- in

CE2 => Low, -- in

RST => RESET, -- in

CLK => BUFIO\_OCLK, -- in

CLKB => Low, -- in

CLKDIV => BUFR\_OCLK, -- in

OCLK => Low, -- in

SHIFTIN1 => Low, -- in

SHIFTIN2 => Low, -- in

SHIFTOUT1 => open, -- out

SHIFTOUT2 => open, -- out

O => open, -- out

Q1 => Srds0pOut(3), -- out

Q2 => Srds0pOut(2), -- out

Q3 => Srds0pOut(1), -- out

Q4 => Srds0pOut(0), -- out

Q5 => open, -- out

Q6 => open -- out

);

process(BUFR\_OCLK)

begin

if rising\_edge(BUFR\_OCLK) then

temp\_reg0 <= Srds0pOut;

temp\_reg1 <= temp\_reg0;

pdataout <= temp\_reg1;

end if;

end process;

end Behavioral;

#### IBUFGDS

Input으로 들어는 Defferential clock을 signle net로 만들기 위하여 아래와 같이 IBUFGDS를 사용합니다.

IBUFGDS\_clk0 : IBUFGDS

generic map ( IOSTANDARD => "LVDS\_25" )

port map (

I => inclk\_p,

IB => inclk\_n,

O => IBUFGDS\_OCLK

);

높은 주파수의 clock 입력의 경우 LVTTL 혹은 LVCMOS와 같은 IO stanard보다는 LVDS, LVPEC, SSTL와 같은 Defferential IO standrad를 많이 사용합니다.

#### IBUFDS

Input으로 들어는 Defferential data을 signle net로 만들기 위하여 아래와 같이 IBUFDS를 사용합니다.

IBUFDS\_data0 : IBUFDS

generic map ( IOSTANDARD => "LVDS\_25" )

port map (

I => sdatain\_p,

IB => sdatain\_n,

O => Srds0sIn

);

#### BUFIO

BUFIO는 일반적으로 사용되는 BUFG(Global clock buffer)와는 다르게 IDDR, ODDR, IODELAY, ISERDES, OSERDES에만 clock을 공급할 수 있는 수 있는 clock buffer입니다.

BUFIO는 max frequency 가 800MHz(Virtex-6, speed grade -2이상기준)입니다.

아래는 BUFIO을 사용한 예입니다.

BUFIO\_clk0 : BUFIO

port map (

I => BUFIO\_ICLK,

O => BUFIO\_OCLK

);

#### BUFR

분주클럭(1/?x clock)을 만들기 위하여 아래와 같이 BUFR을 사용합니다.

최대 1/8x까지 가능하며, 아래와 같이 Generic의 BUFR\_DIVIDE의 value를 선언하여 분주클럭을 지정합니다.

BUFR\_clk0 : BUFR

generic map (

BUFR\_DIVIDE => "4", -- "BYPASS", "1", "2", "3", "4", "5", "6", "7", "8"

SIM\_DEVICE => "VIRTEX6") -- Specify target device, "VIRTEX4", "VIRTEX5", "VIRTEX6"

port map (

I => BUFR\_ICLK, -- Clock buffer input

O => BUFR\_OCLK, -- Clock buffer output

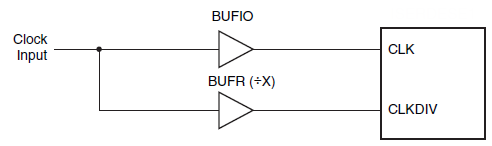
CE => High, -- Clock enable input

CLR => Low -- Clock buffer reset input

);

#### Clock scheme

아래와 같은 clock scheme으로 ISERDES1에 clock이 공급됩니다.



#### ISERDES1

Serial-to-Parallel Logic을 구현하기 위하여 아래와 같이 ISERDES1을 사용합니다.

I\_Isrds\_D0\_p : ISERDESE1

generic map (

SERDES\_MODE => "MASTER",

INTERFACE\_TYPE => "NETWORKING",

IOBDELAY => "NONE",

DATA\_RATE => "SDR",

DATA\_WIDTH => 4,

DYN\_CLKDIV\_INV\_EN => FALSE,

DYN\_CLK\_INV\_EN => FALSE,

NUM\_CE => 1,

OFB\_USED => FALSE

)

port map (

D => Srds0sIn, -- in

DDLY => Low, -- in

DYNCLKDIVSEL => Low, -- in

DYNCLKSEL => Low, -- in

OFB => Low, -- in

BITSLIP => BITSLIP, -- in

CE1 => High, -- in

CE2 => Low, -- in

RST => RESET, -- in

CLK => BUFIO\_OCLK, -- in

CLKB => Low, -- in

CLKDIV => BUFR\_OCLK, -- in

OCLK => Low, -- in

SHIFTIN1 => Low, -- in

SHIFTIN2 => Low, -- in

SHIFTOUT1 => open, -- out

SHIFTOUT2 => open, -- out

O => open, -- out

Q1 => Srds0pOut(3), -- out

Q2 => Srds0pOut(2), -- out

Q3 => Srds0pOut(1), -- out

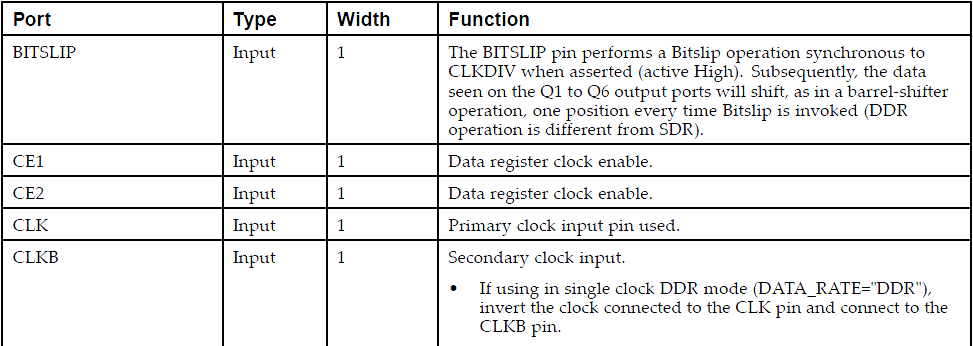
Q4 => Srds0pOut(0), -- out

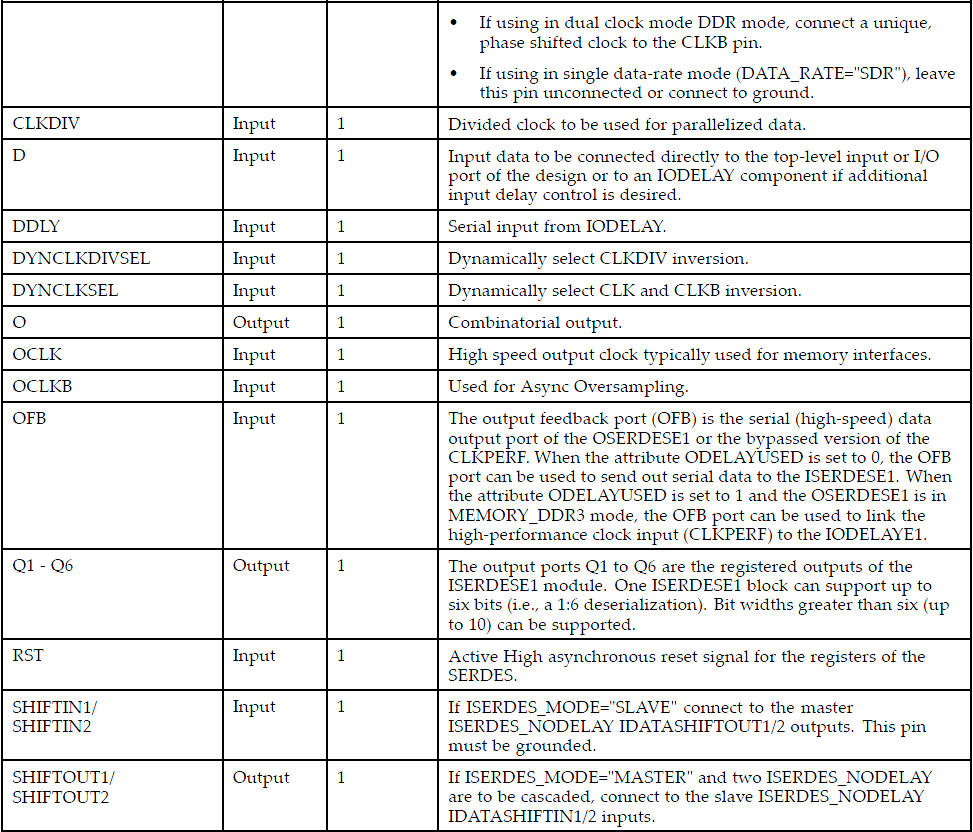
Q5 => open, -- out

Q6 => open -- out

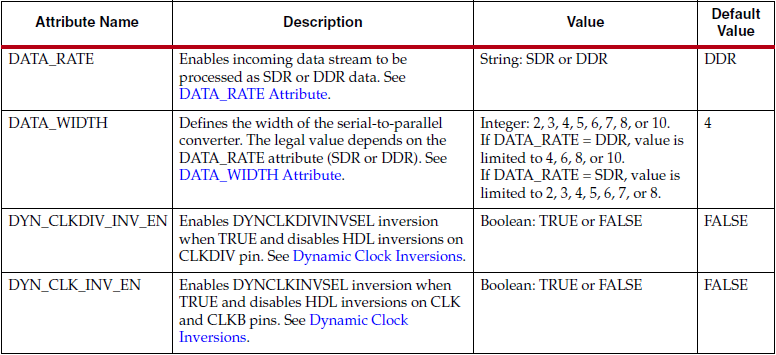
);

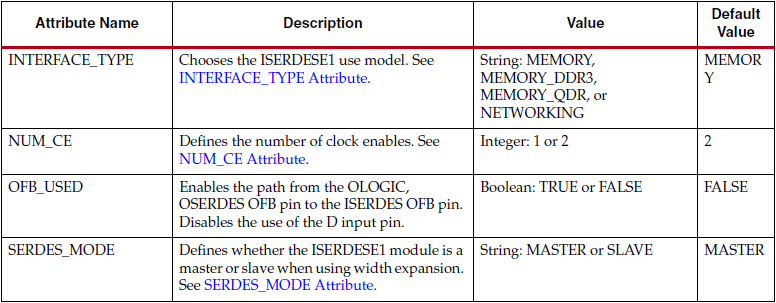
아래의 리스트는 ISERDES1의 각 Pin에 대한 설명 입니다.





아래의 리스트는 ISERDES1의 Generic 선언부에 있는 각 attribute에 대한 설명 입니다.





### Testbench file

아래의 Code는 ISERDESE1을 사용하여 Input Serial-to-Parallel Logic의 testbench file을 구현한 예입니다.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

ENTITY tb01 IS

END tb01;

ARCHITECTURE behavior OF tb01 IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT ISERDES\_1to4\_test

PORT(

RESET : IN std\_logic;

BITSLIP : IN std\_logic;

inclk\_p : IN std\_logic;

inclk\_n : IN std\_logic;

sdatain\_p : IN std\_logic;

sdatain\_n : IN std\_logic;

pdataout : OUT std\_logic\_vector(3 downto 0)

);

END COMPONENT;

--Inputs

signal RESET : std\_logic := '0';

signal BITSLIP : std\_logic := '0';

signal inclk\_p : std\_logic := '0';

signal inclk\_n : std\_logic := '0';

signal sdatain\_p : std\_logic := '0';

signal sdatain\_n : std\_logic := '0';

--Outputs

signal pdataout : std\_logic\_vector(3 downto 0);

BEGIN

uut: ISERDES\_1to4\_test PORT MAP (

RESET => RESET,

BITSLIP => BITSLIP,

inclk\_p => inclk\_p,

inclk\_n => inclk\_n,

sdatain\_p => sdatain\_p,

sdatain\_n => sdatain\_n,

pdataout => pdataout

);

inclk\_p\_process :process

begin

inclk\_p <= '1';

wait for 5 ns;

inclk\_p <= '0';

wait for 5 ns;

end process;

inclk\_n <= not inclk\_p;

RESET <= '0';

sdatain\_p <= '0',

'1' after 500 ns,

'1' after 510 ns,

'0' after 520 ns,

'1' after 530 ns,

'0' after 540 ns,

'1' after 1020 ns,

'1' after 1030 ns,

'0' after 1040 ns,

'1' after 1050 ns,

'0' after 1060 ns;

sdatain\_n <= not sdatain\_p;

BITSLIP <= '0',

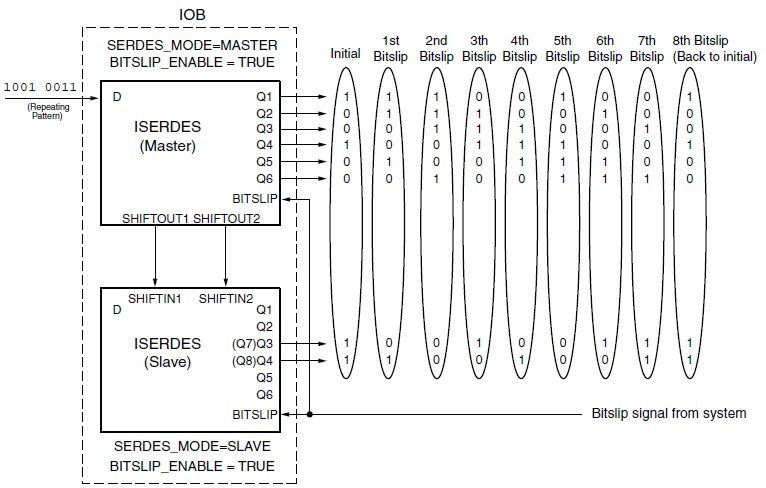
'1' after 860 ns,

'0' after 900 ns;

END;

#### BITSLIP

BITSLIP을 적용할 때마다 아래의 그림과 같이 Parallel data 값이 변하게 됩을 알 수 있습니다.

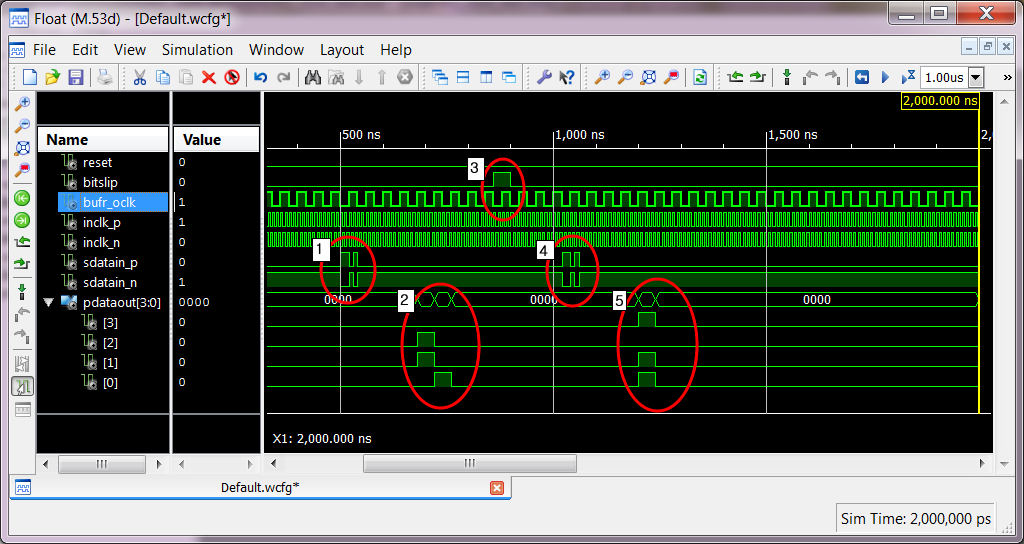


위 그림과 같이 ISERDES1 2개를 각각 master, slave로 연결하여 사용할 수 있습니다.

Simulation 결과를 통하여 BITSLIP의 동작원리에 대하여 알 수 있습니다.

#### Simulation 결과

아래는 위 testbench file을 이용한 simulaton 결과 입니다.



1번과 같이 serial data를 입력으로 사용하였을 때 2번과 같이 parallel data가 틀어짐을 알 수 있습니다.

3번과 같이 분주클럭에 동기되는 BITSLIP을 한 클럭 하이를 유지하면, 4번의 serial data를 입력으로 사용하였을 때 5번과 같이 parallel data가 원하는 파형이 나오는 것을 볼 수 있습니다.

BITSLIP은 분주클럭 기준으로 몇 클럭을 하이로 유지하느냐에 따라 parallel data가 shift되어집니다.

그러므로 BITSLIP을 통하여 serial data 대비 parallel data의 동기를 맞출 수 있습니다.

지금까지 Input Serial-to-Parallel Logic의 구현에 대하여 설명하였습니다.

## OSERDES1

일반적으로 Parallel-to-Serial Logic을 일반 RTL code로도 구현은 가능합니다.

그러나, Serial Data에 동기되는 clock으로 200MHz 이상이 되는 Design의 경우에는 Timing 문제가 발생할 수 밖에 없습니다. 또한, ISE Compile(P&R) 할 때마다 Timing이 틀어지는 문제에 부딪치게 됩니다.( 각 Register들의 location 과 Routing path delay에 따라 Timing이 재각각일 수 밖에 없습니다. )

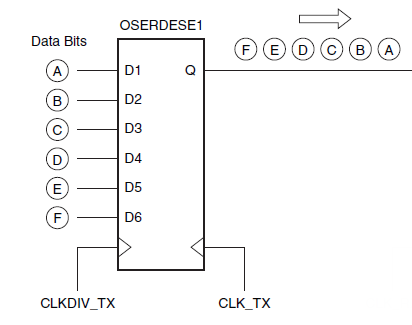
Manual location constraints와 manual Routing constraints를 사용하여 Timing 문제를 해결할 수도 있으나, 모든 Pin에 대하여 각각의 이러한 Constraints를 적용한다는 것은 매우 많은 시간과 번거로운 작업을 진행하여야 합니다.

이러한 문제를 해결하기 위하여 OSERDES1을 사용함으로써 여러가지 고민을 할 필요가 없게 됩니다.

다음은 Output Parallel-to-Serial Logic을 위하여 OSERDES1의 사용법에 대하여 설명 드리겠습니다.

### 기본 이론

일반적으로 Output Parallel-to-Serial Logic를 구현하기 위하여아래의 그림과 같이 OSERDESE1을 사용합니다.



또한 Clock 동기를 맞추기 위하여 CLK\_RX(1x clock)과 CLKDIV\_RX(1/6x clock)이 필요합니다.

위 그림의 OSERDESE1의 Parallel-to-Serial parameter는 6입니다.

실제 설명하게 될 OSERDESE1의 Parallel-to-Serial parameter는 4 입니다.

따라서 clock은 1x clock과 1/4x clock이 필요함을 알 수 있습니다.

### OSERDES1

아래의 Code는 OSERDESE1을 사용하여 Output Parallel-to-Serial Logic을 구현한 예입니다.

(Parallel-to-Serial parameter는 4 입니다.)

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

library UNISIM;

use UNISIM.VComponents.all;

entity OSERDES\_1to4\_test is

Port (

RESET : in std\_logic;

inclk\_p : in std\_logic;

inclk\_n : in std\_logic;

pdatain : in std\_logic\_vector(3 downto 0);

sdataout\_p : out std\_logic;

sdataout\_n : out std\_logic

);

end OSERDES\_1to4\_test;

architecture Behavioral of OSERDES\_1to4\_test is

signal Low : std\_logic;

signal High : std\_logic;

signal IBUFGDS\_OCLK : std\_logic;

signal BUFIO\_ICLK : std\_logic;

signal BUFIO\_OCLK : std\_logic;

signal BUFR\_ICLK : std\_logic;

signal BUFR\_OCLK : std\_logic;

signal temp\_reg0 : std\_logic\_vector(3 downto 0);

signal temp\_reg1 : std\_logic\_vector(3 downto 0);

signal temp\_reg2 : std\_logic\_vector(3 downto 0);

signal Srds0pIn : std\_logic\_vector(3 downto 0);

signal Srds0sOut : std\_logic;

begin

Low <= '0';

High <= '1';

IBUFGDS\_clk0 : IBUFGDS

generic map ( IOSTANDARD => "LVDS\_25" )

port map (

I => inclk\_p,

IB => inclk\_n,

O => IBUFGDS\_OCLK

);

BUFIO\_ICLK <= IBUFGDS\_OCLK;

BUFIO\_clk0 : BUFIO

port map (

I => BUFIO\_ICLK,

O => BUFIO\_OCLK

);

BUFR\_ICLK <= IBUFGDS\_OCLK;

BUFR\_clk0 : BUFR

generic map (

BUFR\_DIVIDE => "4", -- "BYPASS", "1", "2", "3", "4", "5", "6", "7", "8"

SIM\_DEVICE => "VIRTEX6") -- Specify target device, "VIRTEX4", "VIRTEX5", "VIRTEX6"

port map (

I => BUFR\_ICLK, -- Clock buffer input

O => BUFR\_OCLK, -- Clock buffer output

CE => High, -- Clock enable input

CLR => Low -- Clock buffer reset input

);

process(BUFR\_OCLK)

begin

if rising\_edge(BUFR\_OCLK) then

temp\_reg0 <= pdatain;

temp\_reg1 <= temp\_reg0;

Srds0pIn <= temp\_reg1;

end if;

end process;

O\_Srds0pIn\_m : OSERDESE1

generic map (

INTERFACE\_TYPE => "DEFAULT", -- string

SERDES\_MODE => "MASTER", -- string

DATA\_RATE\_OQ => "SDR", -- string

DATA\_RATE\_TQ => "BUF", -- string

DATA\_WIDTH => 4, -- integer

DDR3\_DATA => 0, -- integer

INIT\_OQ => '0', -- bit

INIT\_TQ => '0', -- bit

ODELAY\_USED => 0, -- integer

SRVAL\_OQ => '0', -- bit

SRVAL\_TQ => '0', -- bit

TRISTATE\_WIDTH => 1 -- integer

)

port map (

T1 => Low, -- in

T2 => Low, -- in

T3 => Low, -- in

T4 => Low, -- in

TCE => Low, -- in

ODV => Low, -- in

WC => Low, -- in

TQ => open, -- out

TFB => open, -- out

OFB => open, -- out

OQ => Srds0sOut, -- out

OCBEXTEND => open, -- out

D1 => Srds0pIn(0), -- in

D2 => Srds0pIn(1), -- in

D3 => Srds0pIn(2), -- in

D4 => Srds0pIn(3), -- in

D5 => Low, -- in

D6 => Low, -- in

SHIFTIN1 => Low, -- in

SHIFTIN2 => Low, -- in

SHIFTOUT1 => open, -- out

SHIFTOUT2 => open, -- out

OCE => High, -- in

RST => RESET, -- in

CLK => BUFIO\_OCLK, -- in

CLKDIV => BUFR\_OCLK, -- in

CLKPERF => Low, -- in

CLKPERFDELAY => Low -- in

);

OBUFDS\_data0 : OBUFDS

generic map ( IOSTANDARD => "LVDS\_25" )

port map (

I => Srds0sOut,

O => sdataout\_p,

OB => sdataout\_n

);

end Behavioral;

#### IBUFGDS

Input으로 들어는 Defferential clock을 signle net로 만들기 위하여 아래와 같이 IBUFGDS를 사용합니다.

IBUFGDS\_clk0 : IBUFGDS

generic map ( IOSTANDARD => "LVDS\_25" )

port map (

I => inclk\_p,

IB => inclk\_n,

O => IBUFGDS\_OCLK

);

높은 주파수의 clock 입력의 경우 LVTTL 혹은 LVCMOS와 같은 IO stanard보다는 LVDS, LVPEC, SSTL와 같은 Defferential IO standrad를 많이 사용합니다.

#### BUFIO

BUFIO는 일반적으로 사용되는 BUFG(Global clock buffer)와는 다르게 IDDR, ODDR, IODELAY, ISERDES, OSERDES에만 clock을 공급할 수 있는 수 있는 clock buffer입니다.

BUFIO는 max frequency 가 800MHz(Virtex-6, speed grade -2이상기준)입니다.

아래는 BUFIO을 사용한 예입니다.

BUFIO\_clk0 : BUFIO

port map (

I => BUFIO\_ICLK,

O => BUFIO\_OCLK

);

#### BUFR

분주클럭(1/?x clock)을 만들기 위하여 아래와 같이 BUFR을 사용합니다.

최대 1/8x까지 가능하며, 아래와 같이 Generic의 BUFR\_DIVIDE의 value를 선언하여 분주클럭을 지정합니다.

BUFR\_clk0 : BUFR

generic map (

BUFR\_DIVIDE => "4", -- "BYPASS", "1", "2", "3", "4", "5", "6", "7", "8"

SIM\_DEVICE => "VIRTEX6") -- Specify target device, "VIRTEX4", "VIRTEX5", "VIRTEX6"

port map (

I => BUFR\_ICLK, -- Clock buffer input

O => BUFR\_OCLK, -- Clock buffer output

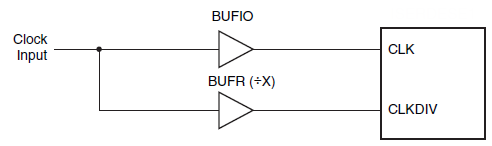
CE => High, -- Clock enable input

CLR => Low -- Clock buffer reset input

);

#### Clock scheme

아래와 같은 clock scheme으로 OSERDES1에 clock이 공급됩니다.



#### OSERDES1

Parallel-to- Serial Logic을 구현하기 위하여 아래와 같이 OSERDES1을 사용합니다.

O\_Srds0pIn\_m : OSERDESE1

generic map (

INTERFACE\_TYPE => "DEFAULT", -- string

SERDES\_MODE => "MASTER", -- string

DATA\_RATE\_OQ => "SDR", -- string

DATA\_RATE\_TQ => "BUF", -- string

DATA\_WIDTH => 4, -- integer

DDR3\_DATA => 0, -- integer

INIT\_OQ => '0', -- bit

INIT\_TQ => '0', -- bit

ODELAY\_USED => 0, -- integer

SRVAL\_OQ => '0', -- bit

SRVAL\_TQ => '0', -- bit

TRISTATE\_WIDTH => 1 -- integer

)

port map (

T1 => Low, -- in

T2 => Low, -- in

T3 => Low, -- in

T4 => Low, -- in

TCE => Low, -- in

ODV => Low, -- in

WC => Low, -- in

TQ => open, -- out

TFB => open, -- out

OFB => open, -- out

OQ => Srds0sOut, -- out

OCBEXTEND => open, -- out

D1 => Srds0pIn(0), -- in

D2 => Srds0pIn(1), -- in

D3 => Srds0pIn(2), -- in

D4 => Srds0pIn(3), -- in

D5 => Low, -- in

D6 => Low, -- in

SHIFTIN1 => Low, -- in

SHIFTIN2 => Low, -- in

SHIFTOUT1 => open, -- out

SHIFTOUT2 => open, -- out

OCE => High, -- in

RST => RESET, -- in

CLK => BUFIO\_OCLK, -- in

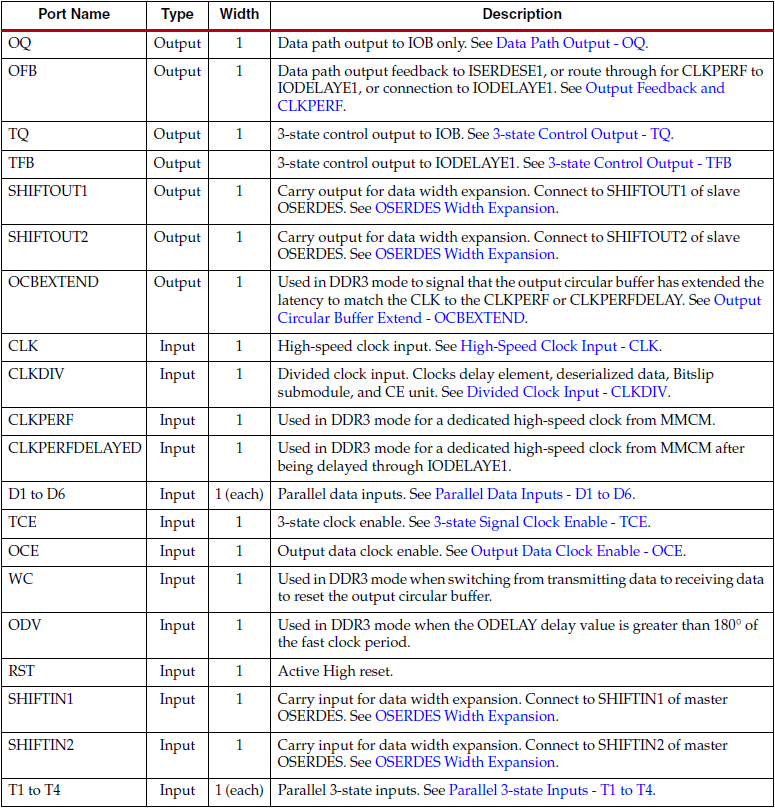
CLKDIV => BUFR\_OCLK, -- in

CLKPERF => Low, -- in

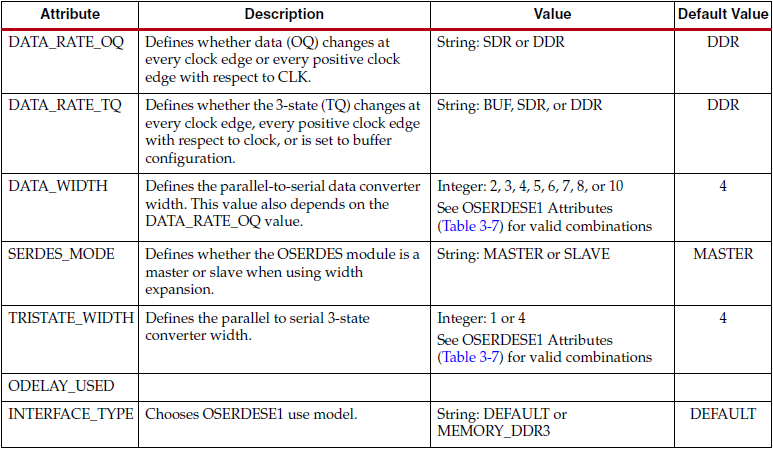
CLKPERFDELAY => Low -- in

);

아래의 리스트는 OSERDES1의 각Pin에 대한 설명 입니다.



아래의 리스트는 OSERDES1의 Generic 선언부에 있는 각 attribute에 대한 설명 입니다.



### Testbench file

아래의 Code는 OSERDESE1을 사용하여 Output Parallel-to-Serial Logic의 testbench file을 구현한 예입니다.

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.numeric\_std.ALL;

ENTITY tb01 IS

END tb01;

ARCHITECTURE behavior OF tb01 IS

COMPONENT OSERDES\_1to4\_test

PORT(

RESET : IN std\_logic;

inclk\_p : IN std\_logic;

inclk\_n : IN std\_logic;

pdatain : IN std\_logic\_vector(3 downto 0);

sdataout\_p : OUT std\_logic;

sdataout\_n : OUT std\_logic

);

END COMPONENT;

--Inputs

signal RESET : std\_logic := '0';

signal inclk\_p : std\_logic := '0';

signal inclk\_n : std\_logic := '0';

signal pdatain : std\_logic\_vector(3 downto 0) := (others => '0');

--Outputs

signal sdataout\_p : std\_logic;

signal sdataout\_n : std\_logic;

-- No clocks detected in port list. Replace <clock> below with

-- appropriate port name

BEGIN

uut: OSERDES\_1to4\_test PORT MAP (

RESET => RESET,

inclk\_p => inclk\_p,

inclk\_n => inclk\_n,

pdatain => pdatain,

sdataout\_p => sdataout\_p,

sdataout\_n => sdataout\_n

);

inclk\_p\_process :process

begin

inclk\_p <= '1';

wait for 5 ns;

inclk\_p <= '0';

wait for 5 ns;

end process;

inclk\_n <= not inclk\_p;

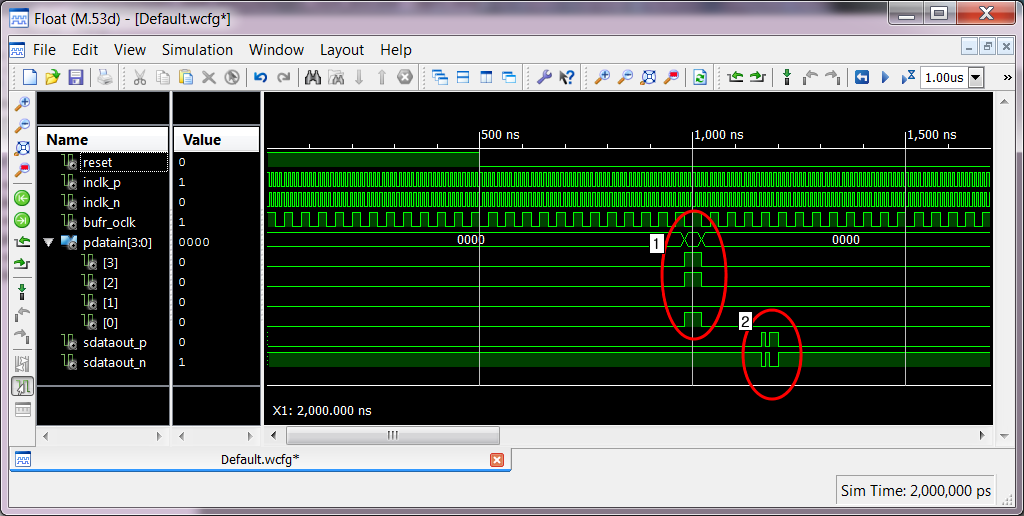
RESET <= '1', '0' after 500 ns;

pdatain <= "0000", "1101" after 980 ns, "0000" after 1020 ns;

END;

#### Simulation 결과

아래는 위 testbench file을 이용한 simulaton 결과 입니다.



1번과 같이 parallel data를 입력으로 사용하였을 때 2번과 같이 serial data가 원하는 파형이 나오는 것을 볼 수 있습니다.

지금까지 Output Parallel-to-Serial Logic의 구현에 대하여 설명하였습니다.

## ISERDES & OSERDES1