Using DDR I/O

Introduction

Virtex-II devices have dedicated registers in a single IOB to implement input, output, and output with 3-state control Double Data Rate (DDR) registers. Input and output DDR is accomplished with the use of two registers in the IOB. A single clock triggers one register on a Low to High transition and a second register on a High to Low transition. Output DDR with 3-state requires the use of four registers in the IOB clocked in a similar fashion. Since the introduction of DLLs, Xilinx devices can generate low-skew clock signals that are 180 degrees out of phase, with a 50/50 duty cycle. These clocks reach the DDR registers in the IOB via dedicated routing resources.

Data Flow

Input DDR

Input DDR is accomplished via a single input signal driving two registers in the IOB. Both registers are clocked on the rising edge of their respective clocks. With proper clock forwarding, alternating bits from the input signal are clocked in on the rising edge of the two clocks, which are 180 degrees out of phase. Figure 2-100 depicts the input DDR registers and the signals involved.

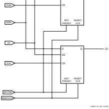

Figure 2-100: **Input DDR**

CLK0 and CLK1 are 180 degrees out of phase. Both registers share the SET/PRE and RESET/CLR lines. As shown in Figure 2-101, alternating bits on the DATA line are clocked in via Q0 and Q1 while CE is High. The clocks are shifted out of phase by the DCM and there is a single clock cycle latency between CE transitioning to a logic High and output at Q0 and Q1.

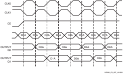

Figure 2-101: **Input DDR Timing Diagram**

Output DDR

Output DDR registers are used to clock output from the chip at twice the throughput of a single rising-edge clocking scheme. Clocking for output DDR is the same as input DDR. The clocks driving both registers are 180 degrees out of phase. The DDR MUX selects the register outputs. The output consists of alternating bits from DATA_1 and DATA_2. Figure 2-102 depicts the output DDR registers and the signals involved.

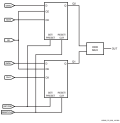

Figure 2-102: **Output DDR**

Both registers share the SET/PRE and RESET/CLR line. Both registers share the CE line which must be High for outputs to be seen on Q1 and Q2. Figure 2-103 shows the data flow for the output DDR registers.

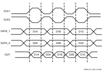

Figure 2-103: **Output DDR Timing Diagram**

Output DDR With 3-State Control

The 3-state control allows the output to have one of two values, either the output from the DDR MUX or high impedance.

The Enable signal is driven by a second DDR MUX (Figure 2-104). This application requires the instantiation of two output DDR primitives.

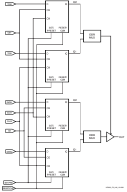

Figure 2-104: **Output DDR With 3-State Control**

All four registers share the SET/PRESET and RESET/CLEAR lines. Two registers are required to accomplish the DDR task and two registers are required for the 3-state control. There are two Clock Enable signals, one for output DDRs performing the DDR function and another for the output DDRs performing the 3-state control function. Two 180 degree out of phase clocks are used. CLK1 clocks one of the DDR registers and a 3-state register. CLK2 clocks the other DDR register and the other 3-state register.

The DDR registers and 3-state registers are associated by the clock that is driving them. Therefore, the DDR register that is clocked by CLK1 is associated to the 3-state register being clocked by CLK1. The remaining two registers are associated by CLK2. If both 3-state registers are driving a logic High, the output sees a high impedance. If both 3-state registers are driving a logic Low, the output sees the values from the DDR MUX see Figure 2-105).

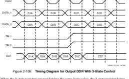

Figure 2-105: **Timing Diagram for Output DDR With 3-State Control**

When the 3-state registers are not driving the same logic value, the 3-state register being clocked by CLK1 is called TREG1. The other 3-state register TREG2 is clocked by CLK2. Similarly, the DDR register being clocked by CLK1 is called DREG1, and the other DDR register DREG2 is clocked by CLK2. If TREG1 is driving a logic High and TREG2 is driving a logic Low, the output sees a high impedance when CLK1 is High and the value out of DREG2 when CLK2 is High. If TREG2 is driving a logic High and TREG1 is driving a logic Low, the output sees a high impedance when CLK2 is High and the value out of DREG1 when CLK1 is High.

Characteristics

- All registers in an IOB share the same SET/PRE and RESET/CLR lines.
- The 3-State and Output DDR registers have common clocks (OTCLK1 & OTCLK2).
- All signals can be inverted (with no added delay) inside the IOB.
- DDR MUXing is handled automatically within the IOB. There is no manual control of the MUX-select. This control is generated from the clock.

Library Primitives

Input DDR registers are inferred, and dedicated output DDR registers have been provided as primitives for Virtex-II designs. Input DDR registers consist of two inferred registers that clock in a single data line on each edge. Generating 3-state output with DDR registers is as simple as instantiating a primitive.

S
D0
FDDRRSE
Q
D1
CE
C0
C1
R

Figure 2-106: **FDDRRSE Symbol: DDR Flip-Flop With Clock Enable and Synchronous Reset and Set**

PRE
D0
FDDRCPE
Q
D1
CE
C0
C1
CLR

Figure 2-107: **FDDRCPE Symbol: DDR Flip-Flop With Clock Enable and Asynchronous PRESET and CLR**

VHDL and Verilog Instantiation

Examples are available in "VHDL and Verilog Templates" on page 294.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template should be inserted within the VHDL design file. The port map of the architecture section should include the design signal names.

Constraints file syntax is provided where input registers need to be used. These settings force the input DDR registers into the IOB. The output registers should be instantiated and do not require any constraints file syntax to be pushed into the IOB.

Port Signals

FDDRRSE

Data inputs - D0 and D1

D0 and D1 are the data inputs into the DDR flip-flop. Data on the D0 input is loaded into the flip-flop when R and S are Low and CE is High during a Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when R and S are Low and CE is High during a Low-to-High C1 clock transition.

Clock Enable - CE

The enable pin affects the loading of data into the DDR flip-flop. When Low, new data is not loaded into the flip-flop. CE must be High to load new data into the flip-flop.

Clocks - C0 and C1

These two clocks are phase shifted 180 degrees (via the DLL) and allow selection of two separate data inputs (D0 and D1).

Synchronous Set - S and Synchronous Reset - R

The Reset (R) input, when High, overrides all other inputs and resets the output Low during any Low-to-High clock transition (C0 or C1). Reset has precedence over Set. When the Set (S) input is High and R is Low, the flip-flop is set, output High, during a Low-to-High clock transition (C0 or C1).

Data Output - Q

When power is applied, the flip-flop is asynchronously cleared and the output is Low.

During normal operation, The value of Q is either D0 or D1. The Data Inputs description above states how the value of Q is chosen.

FDDRCPE

Data inputs - D0 and D1

D0 and D1 are the data inputs into the DDR flip-flop. Data on the D0 input is loaded into the flip-flop when PRE and CLR are Low and CE is High during a Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when PRE and CLR are Low and CE is High during a Low-to-High C1 clock transition.

Clock Enable - CE

The enable pin affects the loading of data into the DDR flip-flop. When Low, clock transitions are ignored and new data is not loaded into the flip-flop. CE must be High to load new data into the flip-flop.

Clocks - C0 and C1

These two clocks are phase shifted 180 degrees (via the DLL) and allow selection of two separate data inputs (D0 and D1).

Asynchronous Preset - PRE and Asynchronous Clear - CLR

The Preset (PRE) input, when High, sets the Q output High. When the Clear (CLR) input is High, the output is reset to Low.

Data Output - Q

When power is applied, the flip-flop is asynchronously cleared and the output is Low. During normal operation, The value of Q is either D0 or D1. The Data Inputs description above states how the value of Q is chosen.

Initialization in VHDL or Verilog

Output DDR primitives can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the output DDR instantiation and are copied in the EDIF output file to be compiled by Xilinx tools. The VHDL code simulation uses a `generic` parameter to pass the attributes. The Verilog code simulation uses the `defparam` parameter to pass the attributes.

The DDR code examples (in VHDL ad Verilog) illustrate the following techniques.

Location Constraints

DDR instances can have LOC properties attached to them to constrain pin placement.

The LOC constraint uses the following form.

```
NET <net_name> LOC=A8;
```

Where "A8" is a valid I/O pin location.

Applications

DDR SDRAM

The DDR SDRAM is an enhancement to the Synchronous DRAM by effectively doubling the data throughput of the memory device. Commands are registered at every positive clock edge. Input data is registered on both edges of the data strobe, and output data is referenced to both edges of the data strobe, as well as both edges of the clock.

Clock Forwarding

DDR can be used to forward a copy of the clock on the output. This can be useful for propagating a clock along with double data rate data that has an identical delay. It is also useful for multiple clock generation, where there is a unique clock driver for every clock load.

VHDL and Verilog Templates

VHDL and Verilog templates are available for output, output with 3-state enable, and input DDR registers.

Input DDR

To implement an Input DDR application, paste the following template in your code.

DDR_input.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DDR_Input is


 Port(
  clk : in std_logic;
        d : in std_logic;
          rst : in std_logic;
   q1 : out std_logic;
   q2 : out std_logic
     );

end DDR_Input;
```

```
--Describe input DDR registers (behaviorally) to be inferred

architecture behavioral of DDR_Input is

begin

q1reg : process (clk, d, rst)

begin
   if rst='1' then --asynchronous reset, active high
      q1 <= '0';

   elsif clk'event and clk='1' then  --Clock event - posedge
         q1 <= d;

   end if;
end process;

q2reg : process (clk, d, rst)

begin
   if rst='1' then --asynchronous reset, active high
      q2 <= '0';
   elsif clk'event and clk='0' then  --Clock event - negedge
      q2 <= d;
   end if;
end process;

end behavioral;


-- NOTE: You must include the following constraints in the .ucf file
when running back-end tools,
--in order to ensure that IOB DDR registers are used:
--
--INST "q2_reg" IOB=TRUE;
--INST "q1_reg" IOB=TRUE;
--
--Depending on the synthesis tools you use, it may be required to check
the edif file for modifications to
--original net names...in this case, Synopsis changed the names: q1 and
q2 to q1_reg and q2_reg
```

DDR_input.v

```
module DDR_Input (data_in , data_out, clk, rst);

input data_in, clk, rst;
output data_out;
reg q1, q2;


//Describe input DDR registers (behaviorally) to be inferred

always @ (posedge clk or posedge rst) //rising-edge DDR reg. and
synchronous reset

  begin
  if (rst)
     q1 = 1'b0;
  else
     q1 = data_in;
  end

always @ (negedge clk or posedge rst) //falling-edge DDR reg. and 
synchronous reset

begin
  if (rst)
     q2 = 1'b0;
  else
     q2 = data_in;
  end

assign data_out = q1 & q2;



endmodule

/* NOTE: You must include the following constraints in the .ucf file 
when running back-end tools, \
  in order to ensure that IOB DDR registers are used:

INST "q2_reg" IOB=TRUE;
INST "q1_reg" IOB=TRUE;

Depending on the synthesis tools you use, it may be required to check 
the edif file for modifications to
original net names...in this case, Synopsis changed the names: q1 and q2 
to q1_reg and q2_reg

*/
```

Output DDR

To implement an Output DDR application, paste the following template in your code.

DDR_out.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- pragma translate_off
LIBRARY UNISIM;
use UNISIM.VCOMPONENTS.ALL;
--pragma translate_on

entity DDR_Output is
  Port(
             clk : in std_logic;  --clk and clk180 can be outputs from 
the DCM or clk180 can be the
    clk180 : in std_logic;  --logical inverse of clk (the inverter is 
located in the IOB and will be inferred.
         d0 : in std_logic;  --data in to fddr
         d1 : in std_logic;  --data in to fddr
        ce : in std_logic;  --clock enable
       rst : in std_logic;  --reset
             set : in std_logic;  --set
         q : out std_logic  --DDR output
       );

end DDR_Output;
```

```
architecture behavioral of DDR_Output is

component FDDRRSE
   port(
      Q  : out std_logic;
      D0 : in std_logic;
      D1 : in std_logic;
      C0  : in std_logic;
      C1  : in std_logic;
      CE : in std_logic;
      R  : in std_logic;
      S : in std_logic
      );
end component;

begin

U0: FDDRRSE
  port map (
    Q => q,
    D0 => d0,
    D1 => d1,
    C0 => clk,
    C1 => clk180,
    CE => ce,
    R => rst,
    S => set
    );


end behavioral;
```

DDR_out.v

```
module DDR_Output (d0 , d1, q, clk, clk180, rst, set, ce);

input d0, d1, clk, clk180, rst, set, ce;
output q;


//Synchronous Output DDR primitive instantiation

FDDRRSE U1  (.D0(d0),
     .D1(d1),
     .C0(clk),
     .C1(clk180),
     .CE(ce),
     .R(rst),
     .S(set),
     .Q(q)
         );
endmodule
```

Output DDR With 3-State Enable

To implement an Output DDR with 3-state Enable, paste the following template in your code:

DDR_3state.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- pragma translate_off
```

```
LIBRARY UNISIM;
use UNISIM.VCOMPONENTS.ALL;
--pragma translate_on

entity DDR_3state is
  Port(
    clk : in std_logic; --clk and clk180 can be outputs from the DCM or
clk180 can be the
       clk180 : in std_logic; --logical inverse of clk (the inverter is
located in the IOB and will be inferred.
             d0 : in std_logic; --data in to fddr
            d1 : in std_logic; --data in to fddr
     ce : in std_logic; --clock enable
    set : in std_logic; --set
    rst : in std_logic; --reset
    en0 : in std_logic; --enable signal
    en1 : in std_logic; --enable signal
     data_out : out std_logic --data seen at pad
  );

end DDR_3state;

architecture behavioral of DDR_3state is

signal ddr_out, tri;

component FDDRRSE
   port(
      Q  : out std_logic;
      D0 : in std_logic;
      D1 : in std_logic;
      C0  : in std_logic;
      C1  : in std_logic;
      CE : in std_logic;
      R  : in std_logic;
      S: in std_logic
      );
end component;


begin

--Instantiate Ouput DDR registers
U0: FDDRRSE port map(Q  => tri,
       D0 => en0,
       D1 => en1,
       C0 => clk,
       C1 => clk180,
       CE => ce,
       R  => rst,
       S  => set
          );

--Instantiate three-state DDR registers
U1: FDDRRSE port map(   Q  => ddr_out,
       D0 => d0,
       D1 => d1,
       C0 => clk,
       C1 => clk180,
       CE => ce,
       R  => rst,
       S  => set
```

```
        );

--inferr the 3-State buffer
process(tri)
begin
  if tri = '1' then
     data_out <= 'Z';
  elsif tri = '0' then
     data_out <= ddr_out;
  end if;
end process;

end behavioral;
```

DDR_3state.v

```
module DDR_3state (d0 , d1, data_out, en_0, en_1, clk, clk180, rst, set,
ce);

input d0, d1, clk, clk180, rst, set, ce, en_0, en_1;

output data_out;
reg data_out;

wire q, q_tri;

//Synchronous Output DDR primitive instantiation

FDDRRSE U1  (    .D0(d0),
    .D1(d1),
    .C0(clk),
    .C1(clk180),
    .CE(ce),
    .R(rst),
    .S(set),
    .Q(q)
      );

//Synchronous 3-State DDR primitive instantiation

FDDRRSE U2  (    .D0(en_0),
    .D1(en_1),
    .C0(clk),
    .C1(clk180),
    .CE(ce),
    .R(rst),
    .S(set),
    .Q(q_tri)
      );

//3-State buffer description

always @ (q_tri or q)
  begin
  if (q_tri)
    data_out = 1'bz;
  else
    data_out = q;
  end

endmodule
```