Using DDR for Fusion Devices

Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Instantiating DDR Registers</td>
<td>3</td>
</tr>
<tr>
<td>Conclusion</td>
<td>12</td>
</tr>
<tr>
<td>Related Documents</td>
<td>12</td>
</tr>
<tr>
<td>List of Changes</td>
<td>13</td>
</tr>
</tbody>
</table>

Introduction

The I/Os on the Fusion device families support Double Data Rate (DDR) mode. In this mode, new data is present on every transition (or clock edge) of the clock signal. This mode doubles the data transfer rate when compared with Single Data Rate (SDR) mode where new data is present on one transition (or clock edge) of the clock signal. The Fusion families have DDR circuitry built into the I/O tiles. I/Os are configured to be DDR receivers or transmitters by instantiating the appropriate special macros and buffers (DDR_OUT or DDR_REG) in the RTL design. This application note discusses the options you can choose to configure the I/Os in this mode and how to instantiate them in the design.

I/O Cell Architecture

The Fusion families support DDR in the I/O cells in four different modes: Inputs, Outputs, Tristate, and Bidirectional pins. For each mode, different I/O standards are supported, with most I/O standards having special sub-options. Refer to Table 1 on page 2 for a sample of the available I/O options. Additional I/O options can be found in the Fusion Family of Mixed Signal FPGAs datasheet.
### Table 1 • DDR I/O Options

<table>
<thead>
<tr>
<th>DDR Register Type</th>
<th>I/O Type</th>
<th>I/O Standard</th>
<th>Sub-options</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receive Register</td>
<td>Input</td>
<td>Normal</td>
<td>None</td>
<td>3.3 V TTL (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVCMOS</td>
<td>Voltage 1.5 V, 1.8 V, 2.5 V, 5 V (1.5 V default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PCI/PCIX</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GTL/GTLP</td>
<td>Voltage 2.5 V, 3.3 V (3.3 V default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>HSTL</td>
<td>Class I / II (I default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SSTL2/SSTL3</td>
<td>Class I / II (I default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVPECL</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVDS</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Transmit Register</td>
<td>Output</td>
<td>Normal</td>
<td>None</td>
<td>3.3 V TTL (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVTTTL</td>
<td>Output drive 2, 4, 6, 8, 12, 16, 24, 36 (8 mA default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Slew rate Low/High (High default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVCMOS</td>
<td>Voltage 1.5 V, 1.8 V, 2.5 V, 5 V (1.5 V default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PCI/PCIX</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GTL/GTLP</td>
<td>Voltage 1.8 V, 2.5 V, 3.3 V (3.3 V default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>HSTL</td>
<td>Class I / II (I default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SSTL2/SSTL3</td>
<td>Class I / II (I default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVPECL</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVDS</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Tristate Buffer</td>
<td></td>
<td>Normal</td>
<td>Enable polarity Low/high (low default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVTTTL</td>
<td>Output Drive 2, 4, 6, 8, 12, 16, 24, 36 (8 mA default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Slew rate Low/High (high default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity Low/high (low default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pull-up/down None (default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVCMOS</td>
<td>Voltage 1.5 V, 1.8 V, 2.5 V, 5 V (1.5 V default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Output drive 2, 4, 6, 8, 12, 16, 24, 36 (8 mA default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Slew rate Low/High (high default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity Low/high (low default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pull-up/down None (default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PCI/PCI-X</td>
<td>Enable polarity Low/high (low default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GTL/GTLP</td>
<td>Voltage 1.8 V, 2.5 V, 3.3 V (3.3 V default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity Low/high (low default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>HSTL</td>
<td>Class I / II (I default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity Low/high (low default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SSTL2/SSTL3</td>
<td>Class I / II (I default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity Low/high (low default)</td>
<td></td>
</tr>
</tbody>
</table>
Instantiating DDR Registers

**Instantiations**

Using SmartGen is the simplest way to generate the appropriate RTL files for use in the design. SmartGen provides the capability to generate all of the DDR I/O cells as described. Through the graphical user interface (GUI), you can select from among the many supported I/O standards. The output formats supported are Verilog, VHDL, or EDIF files.

*Figure 2 on page 4, Figure 3 on page 5, Figure 4 on page 7, and Figure 5 on page 8 show the I/O cell configured for DDR using SSTL2 Class I technology. For each I/O standard, the I/O pad is buffered by a special primitive that indicates the I/O standard type.*

<table>
<thead>
<tr>
<th>DDR Register Type (continued)</th>
<th>I/O Type</th>
<th>I/O Standard</th>
<th>Sub-options</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit Register</td>
<td>Bidirectional</td>
<td>Normal</td>
<td>Enable polarity</td>
<td>Low/high (low default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVTTTL</td>
<td>Output drive</td>
<td>2, 4, 6, 8, 12, 16, 24, 36 (8 mA default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Slew rate</td>
<td>Low/high (high default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity</td>
<td>Low/high (low default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pull-up/down</td>
<td>None (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVCMOS</td>
<td>Voltage</td>
<td>1.5 V, 1.8 V, 2.5 V, 5 V (1.5 V default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity</td>
<td>Low/high (low default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pull-up</td>
<td>None (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PCI/PCIX</td>
<td>None</td>
<td>None (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity</td>
<td>Low/high (low default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GTL/GTLP</td>
<td>Voltage</td>
<td>1.8 V, 2.5 V, 3.3 V (3.3 V default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity</td>
<td>Low/high (low default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HSTL</td>
<td>Class</td>
<td>I / II (I default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity</td>
<td>Low/high (low default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SSTL2/SSTL3</td>
<td>Class</td>
<td>I / II (I default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Enable polarity</td>
<td>Low/high (low default)</td>
</tr>
</tbody>
</table>

*Table 1 • DDR I/O Options (continued)*
Figure 1 • Example of Using SmartGen to Generate a DDR SSTL2 Class I Input Register

Figure 2 • DDR Input Register (SSTL2 Class I)
The corresponding structural representations as generated by SmartGen are shown below:

**Verilog**

```verilog
module DDR_InBuf_SSTL2_I(PAD, CLR, CLK, QR, QF);

input   PAD, CLR, CLK;
output  QR, QF;

wire Y;

INBUF_SSTL2_I INBUF_SSTL2_I_0_inst(.PAD(PAD),.Y(Y));
DDR_REG DDR_REG_0_inst(.D(Y),.CLK(CLK),.CLR(CLR),.QR(QR),.QF(QF));

endmodule
```

**VHDL**

```vhdl
library ieee;
use ieee.std_logic_1164.all;
library fusion;

entity DDR_InBuf_SSTL2_I is
    port(PAD, CLR, CLK : in std_logic;  QR, QF : out std_logic);
end DDR_InBuf_SSTL2_I;

architecture DEF_ARCH of  DDR_InBuf_SSTL2_I is
    component INBUF_SSTL2_I
        port(PAD : in std_logic := 'U'; Y : out std_logic);
    end component;

    component DDR_REG
        port(D, CLK, CLR : in std_logic := 'U'; QR, QF : out std_logic);
    end component;

    signal Y : std_logic;

begin

    INBUF_SSTL2_I_0_inst : INBUF_SSTL2_I
        port map(PAD => PAD, Y => Y);
    DDR_REG_0_inst : DDR_REG
        port map(D => Y, CLK => CLK, CLR => CLR, QR => QR, QF => QF);

end DEF_ARCH;
```

![Figure 3 • DDR Output Register (SSTL3 Class I)](image-url)
**Verilog**

module DDR_OutBuf_SSTL3_I(DataR, DataF, CLR, CLK, PAD);

input   DataR, DataF, CLR, CLK;
output  PAD;
wire Q, VCC;

VCC VCC_1_net(.Y(VCC));
DDR_OUT DDR.OUT_0_inst(.DR(DataR),.DF(DataF),.CLK(CLK),.CLR(CLR),.Q(Q));
OUTBUF_SSTL3_I OUTBUF_SSTL3_I_0_inst(.D(Q),.PAD(PAD));

endmodule

**VHDL**

library ieee;
use ieee.std_logic_1164.all;
library fusion;

entity DDR_OutBuf_SSTL3_I is
  port(DataR, DataF, CLR, CLK : in std_logic;  PAD : out std_logic) ;
end DDR_OutBuf_SSTL3_I;

architecture DEF_ARCH of DDR_OutBuf_SSTL3_I is

  component DDR_OUT
    port(DR, DF, CLK, CLR : in std_logic := 'U'; Q : out std_logic) ;
  end component;

  component OUTBUF_SSTL3_I
    port(D : in std_logic := 'U'; PAD : out std_logic) ;
  end component;

  component VCC
    port(Y : out std_logic);
  end component;

  signal Q, VCC_1_net : std_logic ;

begin

  VCC_2_net : VCC port map(Y => VCC_1_net);
  DDR.OUT_0_inst : DDR_OUT
  port map(DR => DataR, DF => DataF, CLK => CLK, CLR => CLR, Q => Q);
  OUTBUF_SSTL3_I_0_inst : OUTBUF_SSTL3_I
  port map(D => Q, PAD => PAD);
end DEF_ARCH;
Figure 4 • DDR Tristate Output Register, Low Enable, 8 mA, Pull-Up (LVTTL)

Verilog

module DDR_TriStateBuf_LVTTL_8mA_HighSlew_LowEnb_PullUp(DataR, DataF, CLR, CLK, Trien, PAD);
input DataR, DataF, CLR, CLK, Trien;
output PAD;
wire TrienAux, Q;
INV Inv_Tri (.A(Trien),.Y(TrienAux));
DDR_OUT DDR_OUT_0_inst (.DR(DataR),.DF(DataF),.CLK(CLK),.CLR(CLR),.Q(Q));
TRIBUFF_F_8U TRIBUFF_F_8U_0_inst (.D(Q),.E(TrienAux),.PAD(PAD));
endmodule

VHDL

library ieee;
use ieee.std_logic_1164.all;
library fusion;

entity DDR_TriStateBuf_LVTTL_8mA_HighSlew_LowEnb_PullUp is
port(DataR, DataF, CLR, CLK, Trien : in std_logic;  PAD : out std_logic) ;
end DDR_TriStateBuf_LVTTL_8mA_HighSlew_LowEnb_PullUp;

architecture DEF_ARCH of DDR_TriStateBuf_LVTTL_8mA_HighSlew_LowEnb_PullUp is

component INV
  port(A : in std_logic := 'U'; Y : out std_logic) ;
end component;

component DDR_OUT
  port(DR, DF, CLK, CLR : in std_logic := 'U'; Q : out std_logic) ;
end component;

component TRIBUFF_F_8U
  port(D, E : in std_logic := 'U'; PAD : out std_logic) ;
end component;

signal TrienAux, Q : std_logic ;
begin
Inv_Tri : INV
port map(A => Trien, Y => TrienAux);
Verilog

module DDR_BiDir_HSTL_I_LowEnb(DataR, DataF, CLR, CLK, Trien, QR, QF, PAD);

input   DataR, DataF, CLR, CLK, Trien;
output  QR, QF;
inout   PAD;

wire TrienAux, D, Q;

INV Inv_Tri (.A(Trien), .Y(TrienAux));

DDR_OUT DDR_OUT_0_inst (.DR(DataR), .DF(DataF), .CLK(CLK), .CLR(CLR), .Q(Q));

DDR_REG DDR_REG_0_inst (.D(D), .CLK(CLK), .CLR(CLR), .QR(QR), .QF(QF));

BIBUF_HSTL_I BIBUF_HSTL_I_0_inst (.PAD(PAD), .D(Q), .E(TrienAux), .Y(D));

endmodule

Figure 5 • DDR Bidirectional Buffer, Low Output Enable (HSTL Class 2)
VHDL

library ieee;
use ieee.std_logic_1164.all;
library fusion;

entity DDR_BiDir_HSTL_I_LowEnb is
  port(DataR, DataF, CLR, CLK, Trien : in std_logic;
       QR, QF : out std_logic;  PAD : inout std_logic) ;
end DDR_BiDir_HSTL_I_LowEnb;

architecture DEF_ARCH of DDR_BiDir_HSTL_I_LowEnb is

  component INV
    port(A : in std_logic := 'U'; Y : out std_logic) ;
  end component;

  component DDR_OUT
    port(DR, DF, CLK, CLR : in std_logic := 'U'; Q : out std_logic) ;
  end component;

  component DDR_REG
    port(D, CLK, CLR : in std_logic := 'U'; QR, QF : out std_logic) ;
  end component;

  component BIBUF_HSTL_I
    port(PAD : inout std_logic := 'U'; D, E : in std_logic := 'U';
         Y : out std_logic) ;
  end component;

  signal TrienAux, D, Q : std_logic ;

begin

  Inv_Tri : INV
    port map(A => Trien, Y => TrienAux);
  DDR_OUT_0_inst : DDR_OUT
    port map(DR => DataR, DF => DataF, CLK => CLK, CLR => CLR, Q => Q);
  DDR_REG_0_inst : DDR_REG
    port map(D => D, CLK => CLK, CLR => CLR, QR => QR, QF => QF);
  BIBUF_HSTL_I_0_inst : BIBUF_HSTL_I
    port map(PAD => PAD, D => Q, E => TrienAux, Y => D);

end DEF_ARCH;
Design Example

Figure 6 shows a simple example of a design using both DDR input and DDR output registers. You can copy the HDL code in Libero SoC software and go through the design flow. Figure 7 and Figure 8 on page 11 show the netlist and ChipPlanner view of the ddr_test design.

Figure 6 • Design Example

Figure 7 • DDR Test Design as Seen by NetlistViewer
Instantiating DDR Registers

Verilog

module Inbuf_ddr(PAD, CLR, CLK, QR, QF);
input PAD, CLR, CLK;
output QR, QF;
wire Y;
    DDR_REG DDR_REG_0_inst (.D(Y), .CLK(CLK), .CLR(CLR), .QR(QR), .QF(QF));
    INBUF INBUF_0_inst (.PAD(PAD), .Y(Y));
endmodule

module Outbuf_ddr(DataR, DataF, CLR, CLK, PAD);
input DataR, DataF, CLR, CLK;
output PAD;
wire Q, VCC;
    VCC VCC_1_net (.Y(VCC));
    DDR_OUT DDR_OUT_0_inst (.DR(DataR), .DF(DataF), .CLK(CLK), .CLR(CLR), .Q(Q));
    OUTBUF OUTBUF_0_inst (.D(Q), .PAD(PAD));
endmodule

module ddr_test(DIN, CLK, CLR, DOUT);
input DIN, CLK, CLR;
output DOUT;
Inbuf_ddr Inbuf_ddr (.PAD(DIN), .CLR(clr), .CLK(clk), .QR(qr), .QF(qf));
Outbuf_ddr Outbuf_ddr (.DataR(qr), .DataF(qf), .CLR(clr), .CLK(clk), .PAD(DOUT));
INBUF INBUF_CLR (.PAD(CLR), .Y(clr));
INBUF INBUF_CLK (.PAD(CLK), .Y(clk));
endmodule

Figure 8 • DDR Input/Output Cells as Seen by ChipPlanner
Simulation Consideration

Microsemi DDR simulation models use inertial delay modeling by default (versus transport delay modeling). As such, pulses that are shorter than the actual gate delays should be avoided as they are not seen by the simulator and may be an issue in post-routed simulations. You should be aware of the default delay modeling and must set the correct delay model in the simulator as needed.

Conclusion

The Fusion devices support a wide range of DDR applications with the different I/O standards and include built-in DDR macros. The powerful capabilities provided by SmartGen macro builder simplify the process of including DDR macros in the designs and minimize the design errors. The designer should take the additional considerations into account in design floorplaning and placement of I/O flip-flops to minimize the datapath skew and to help improve the system timing margins. Other system-related issues to consider include PLL and clock partitioning.

Related Documents

Datasheets

_Fusion Family of Mixed Signal FPGAs_

List of Changes

The following table lists critical changes that were made in each revision of the document.

<table>
<thead>
<tr>
<th>Revision*</th>
<th>Changes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision 1 (February 2012)</td>
<td>The &quot;Design Example&quot; section was revised. (SAR - 32657)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>The &quot;Related Documents&quot; section was revised. (SAR - 32657)</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>The Figure 1 was updated.</td>
<td>4</td>
</tr>
</tbody>
</table>

**Note:** The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.