XAPP491 (v1.0) October 4, 2006

Inverting LVDS Signals for Efficient PCB Layout in Spartan-3 Generation FPGAs

Author: Nick Sawyer and Gary Lawman

Summary

Differential signals, such as LVDS or LVPECL, can be difficult to route on simple, four-layer or six-layer PCBs without excessive use of vias. The reason has been because the positive pins on the driver must drive the corresponding positive pins on the receiver, and the negative pins must drive the receiver's negative pins. Sometimes the traces end up with the wrong orientation, in effect, adding an inverter to the circuit. This application note shows how Spartan™-3 Generation FPGAs, with just the inclusion of an inverter in the receiver datapath, can avoid excessive use of vias and fix accidental PCB trace swapping without requiring a PCB respin. The technique is equally applicable to the case where the FPGA is the driver, and swapping traces allows easier PCB routing to another device or to a connector.

Introduction

Figure 1 shows a PCB example with the positive pins driving the receiver's positive pins and the negative pins driving the receiver's negative pins. If the pins are accidentally swapped, the PCB traces effectively become an inverter, possibly requiring a board respin.

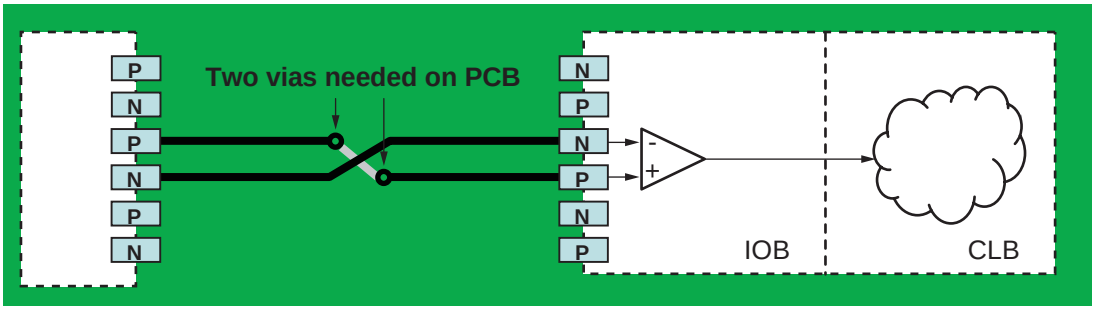

Figure 1: **PCB Layout Requires Use of Vias to Swap Traces**

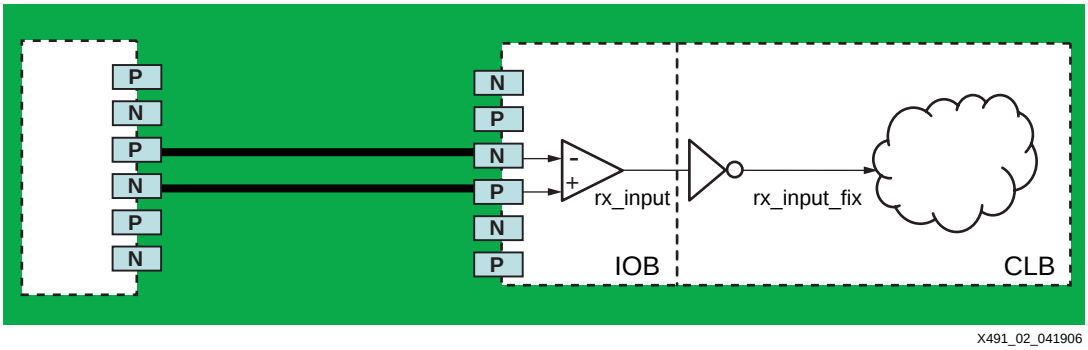

Figure 2: **PCB Layout Requires No Vias to Swap Traces**

Figure 2 shows how the Spartan-3 Generation FPGA resolves this problem by including the necessary inverter in the receiver datapath. With this feature, the designer can choose to deliberately swap the traces for simplified routing. Thus the PCB designer is free to lay out the

differential pairs for the maximum signal integrity; any swapping that occurs can be corrected inside the FPGA. In cases where a DCM is being used (see "Asynchronous Inputs"), this routing freedom applies only to data lines but *not* to clock lines. Swapping the lines can never damage the device.

Absorbed Inverter Examples

There are two cases where inverters can be absorbed forward:

1. When driving flip-flop inputs directly
2. When driving into a logic function

In Case #1, the Spartan-3 Generation FPGAs have a multiplexer on the direct (D) inputs of the CLB flip-flops, as shown in Figure 3. This multiplexer selects between the true and the complement of the input signal. The multiplexer is configured from a configuration cell, which is initialized by the bitstream loaded into the device. The user cannot access the multiplexer during operation.

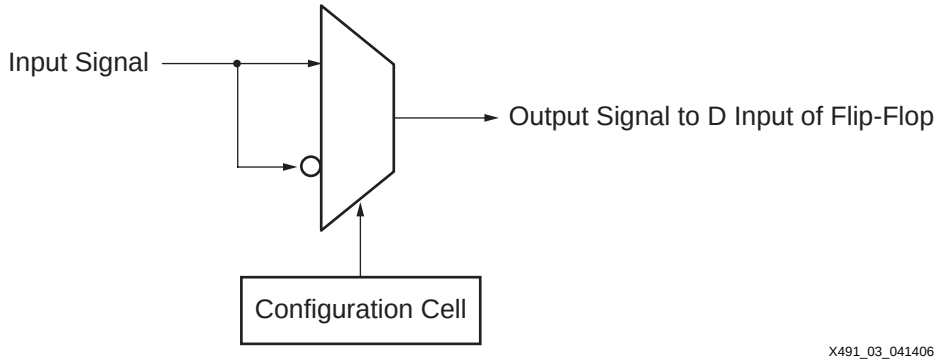

Figure 3: **Programmable Inverter in Front of CLB Flip-Flops**

In Case #2, the inverter is simply absorbed. For example, if an inverter performing B = ~A is followed by an AND gate performing D = B AND C, a simple substitution has an AND gate performing D = ~A AND C with no inverter; in other words, the inverter absorption is always "free" in terms of logic utilization and delay.

This mechanism for inverter absorption also applies to the IOB output flip-flops. Again the absorption is "free" to include an inverter in the output path of the FPGA, if needed to make PCB layout easier. This mechanism can help where the FPGA is driving a connector with predefined pins that directly match to the N and P LVDS outputs of the FPGA.

Asynchronous Inputs

Figure 2 is the simplest example to consider. The received, swapped LVDS signal will be used in combinational logic inside the FPGA. In this example, only a simple inverter needs to be added to the code. The code for this inverter is shown below in Verilog and VHDL:

```
Verilog:    assign rx_input_fix = ~rx_input;
VHDL:       rx_input_fix <= not rx_input;
```

This inverter can be absorbed either into the combinational logic driven by the input signal or into the D input of a flip-flop in the interior of the FPGA. It cannot be absorbed into a flip-flop in the IOB of the FPGA, into a DCM, or into a BUFGMUX clock buffer. For this reason, the flexibility of pin swapping cannot be applied to clock signals where that clock signal is going to be used to clock in data. If the clock in question is just an oscillator for a system, then the lines can be swapped and not re-inverted without any negative effects.

Figure 4 shows an example where the input is actually a bus of *n* signal pairs. Some pairs are correct whereas others are swapped for convenience. In this example, it makes sense to define

a mask in the design corresponding to the *n* inputs. The mask is used to selectively invert (in fact, exclusive OR) those bits that need correction and not invert the correctly received bits. In Figure 4, bits 0 and 2 are correct, and bit 1 needs inversion. The best way to handle the correction in the code is to use generate loops that instantiate the input buffer and perform the optional inversion on a bit-by-bit basis.

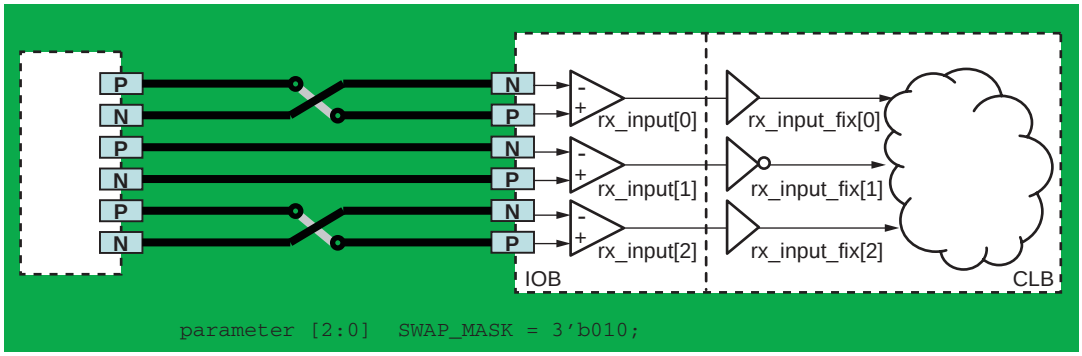

Figure 4: **Some Bus Traces are Polarity Swapped on the PCB to Simplify Routing**

The following Verilog code performs the receive inversion using generate loops:

```
.
parameter [2:0] SWAP_MASK = 3'b010;
.
.
genvar i;
generate
for (i = 0; i <= 2; i = i + 1)
begin: loop0
IBUFDS
#(.IOSTANDARD("LVDS_25"), .IBUF_DELAY_VALUE("0"), .DIFF_TERM("FALSE"))
ibuf_d   (.I(datain_p[i]), .IB(datain_n[i]), .O(rx_input[i]));
assign rx_input_fix[i] = rx_input[i] ^ SWAP_MASK[i];
end
endgenerate
```

The following VHDL code performs the receive inversion using generate loops:

```
.
constant SWAP_MASK : std_logic_vector(2 downto 0):= "010";
.
.
loop0: for i in 0 to 2 generate
ibuf_d: ibufds generic map
(IOSTANDARD => "LVDS_25", IBUF_DELAY_VALUE => "0", DIFF_TERM => FALSE)
port map
(i => datain_p(i), iB => datain_n(i), o => rx_input(i));
rx_input_fix(i) <= rx_input(i) xor SWAP_MASK(i);
end generate;
```

This mechanism is easily extendable to various bit widths by changing the characters marked in red, boldface text.

Synchronous Use of IOB Input Flip-Flops

Typically, input signals are registered in the IOB flip-flops because high-speed data transmission is the most common reason for using LVDS. Data can be registered using either:

- a single data rate (SDR) technique, which uses just one (usually positive edge triggered) flip-flop in the IOB

  or

- a double data rate (DDR) technique, where the input data line is sampled using both positive and negative edge triggered flip-flops

In both cases, it is not possible to invert the input signal between the input amplifier and the flip-flop because the flip-flops within the IOB blocks do not have invertible inputs. The inverters need to be added after the IOB input flip-flop(s), and they can be absorbed into the following registered or combinational logic.

SDR Example

Figure 5 shows the SDR scenario with one flip-flop in the IOB.

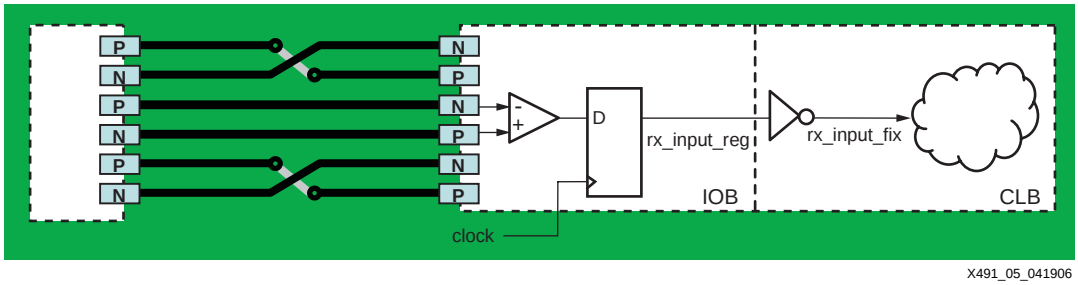

Figure 5: **SDR Registered Receiver**

The following code illustrates the SDR registered case for the same generate loop example. The only change is the addition of the flip-flop instantiation.

In Verilog:

```
.
parameter [2:0] SWAP_MASK = 3'b010;
.
.
genvar i;
generate
for (i = 0; i <= 2; i = i + 1)
begin: loop0
IBUFDS#(.IOSTANDARD("LVDS_25"), .IFD_DELAY_VALUE("0"), .DIFF_TERM("FALSE"))
ibuf_d   (.I(datain_p[i]), .IB(datain_n[i]), .O(rx_input[i]));
FD fd_d (.C(clkin), .D(rx_input[i]), .Q(rx_input_reg[i]));
assign rx_input_fix[i] = rx_input_reg[i] ^ SWAP_MASK[i];
end
endgenerate
```

In VHDL:

```
.
constant SWAP_MASK : std_logic_vector(2 downto 0):= "010";
.
.
loop0: for i in 0 to 2 generate
ibuf_d: ibufds
generic map (IOSTANDARD => "LVDS_25", IFD_DELAY_VALUE => "0", DIFF_TERM => FALSE)
port map    (i => datain_p(i), iB => datain_n(i), o => rx_input(i));
fd_d: fd     port map    (c => clkin, d => rx_input(i), q => rx_input_reg(i));
rx_input_fix(i) <= rx_input_reg(i) xor SWAP_MASK(i);
end generate;
```

This mechanism is easily extendable to various bit widths by changing the characters marked in red, boldface text.

Input DDR Example

Figure 6 shows the receive DDR scenario, where each input line generates two internal data lines that might need inverting. For DDR inputs with the Spartan-3E FPGA, the new IDDR2 structure of input flip-flops is recommended for use. This structure can make the internal logic easier to design by removing any paths from a falling edge to the next rising edge. For more information on IDDR2, see DS312, *Spartan-3E FPGA Family Data Sheet*.

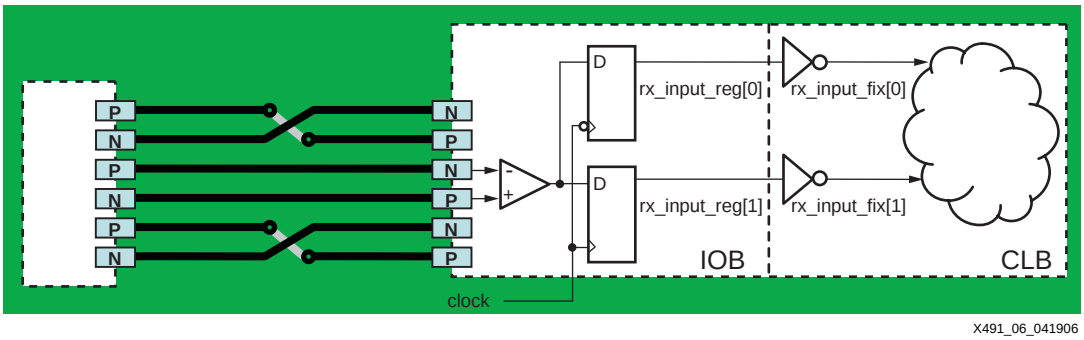

Figure 6: **DDR Registered Receiver**

The following code illustrates the DDR registered receiver case for the same generate loop example. The only change is the addition of the IDDR2 instantiation for the Spartan-3E FPGA. The original Spartan-3 device requires slightly different coding because it does not include the IDDR2 structure. Complete details are in the accompanying ZIP file (see "Design Files").

In Verilog:

```
.
parameter [2:0] SWAP_MASK = 3'b010;
.
.
genvar i;
generate
for (i = 0; i <= 2; i = i + 1)
begin: loop0
IBUFDS#(.IOSTANDARD("LVDS_25"), .IFD_DELAY_VALUE("0"), .DIFF_TERM("FALSE"))
ibuf_d   (.I(datain_p[i]), .IB(datain_n[i]), .O(rx_input[i]));
IDDR2 #(.DDR_ALIGNMENT("C0")) fd_ioc(.C0(clkin), .C1(notclk), .D(rx_input[i]),
        .CE(1'b1), .R(1'b0), .S(1'b0), .Q0(rx_input_reg[i+3]),
        .Q1(rx_input_reg[i]));
assign rx_input_fix[i] = rx_input_reg[i] ^ SWAP_MASK[i];
assign rx_input_fix[i+3] = rx_input_reg[i+3] ^ SWAP_MASK[i];
end
endgenerate
```

In VHDL:

```
.
constant SWAP_MASK : std_logic_vector(2 downto 0):= "010";
.
.
loop0: for i in 0 to 2 generate
ibuf_d : ibufds
generic map (IOSTANDARD => "LVDS_25", IFD_DELAY_VALUE => "0", DIFF_TERM => FALSE)
port map    (i => datain_p(i), iB => datain_n(i), o => rx_input(i));
fd_d : iddr2
generic map (DDR_ALIGNMENT => "C0")
port map (c0 => clkin, c1 => notclock, d => rx_input(i), ce => '1', r => '0',
          s => '0', q0 => rx_input_reg(i+3), q1 => rx_input_reg(i));
rx_input_fix(i)   <= rx_input_reg(i) xor SWAP_MASK(i);
rx_input_fix(i+3) <= rx_input_reg(i+3) xor SWAP_MASK(i);
```

This mechanism is easily extendable to various bit widths by changing the characters marked in red, boldface text.

Bit manipulation can be important when using DDR techniques. The DDR generate loop example generates a bus whose low-order bits are collected on the falling edge of the clock and whose high-order bits are collected on the following rising edge. Figure 7 is a screen shot of a simulation run of the DDR design, showing this bit collection. This simulation assumes all traces are correct (that is, no pin swapping has been performed), to clearly show which bit ends up where.

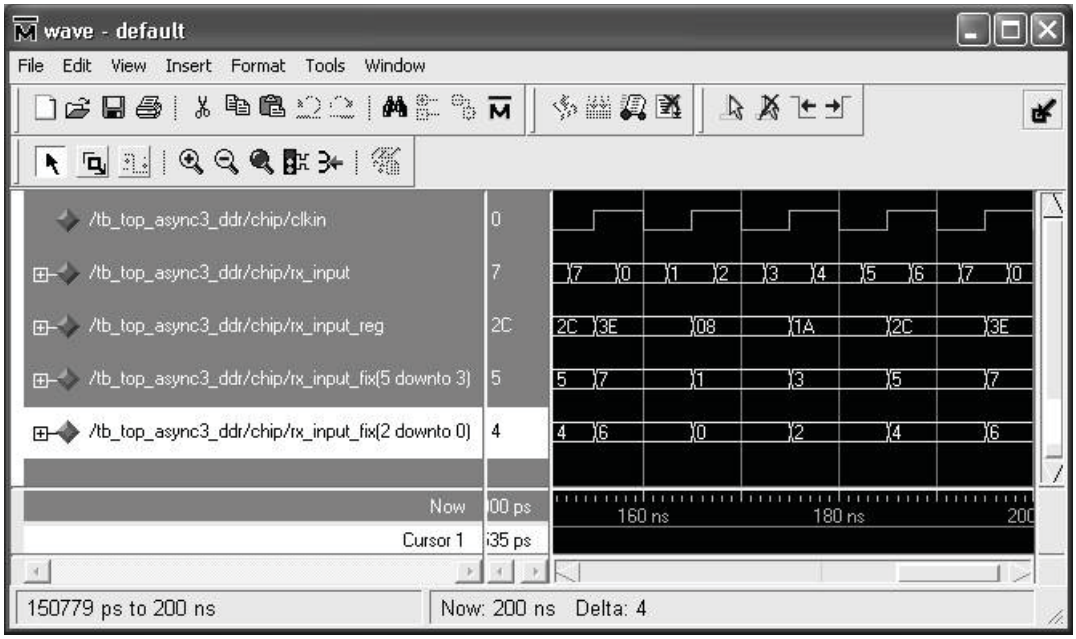

Figure 7: **Simulation of the Input DDR Case**

Output DDR Example

Figure 8 shows the transmitter DDR scenario where each pair of transmit data lines is multiplexed by the ODDR2 mechanism in the Spartan-3E FPGA (FDDRRSE in the Spartan-3 FPGA). In this case, inverters are added to each line associated with the LVDS output whose polarity is required to be inverted. Because these inverters are absorbed into the output flip-flops during implementation as described above, they do not change the timing of the circuit.

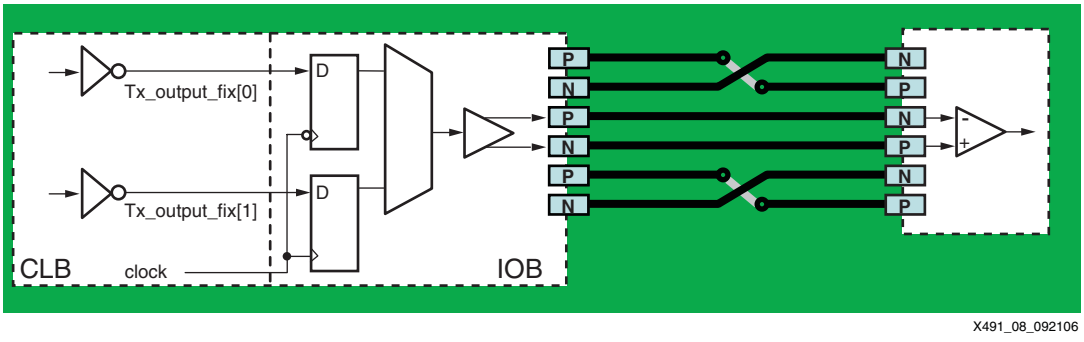

Figure 8: **Double Data Rate Registered Transmitter**

The following code illustrates the transmitter DDR registered case for the generate loop example. The original Spartan-3 device requires slightly different coding because it does not include the ODDR2 structure. Complete details are in the accompanying ZIP file (see "Design Files").

In Verilog:

```
parameter [2:0] SWAP_MASK = 3'b010 ;

genvar i ;
generate
for (i = 0 ; i <= 2 ; i = i + 1)
begin : loop0
OBUFDS #(.IOSTANDARD("LVDS_25"))
obuf_d   (.I(tx_output_reg[i]), .O(dataout_p[i]), .OB(dataout_n[i]));
ODDR2  #(.DDR_ALIGNMENT("NONE")) fd_ioc  (.C0(clkin), .C1(notclk),
.D0(tx_output_fix[i+3]), .D1(tx_output_fix[i]), .CE(1'b1), .R(1'b0),
.S(1'b0), .Q(tx_output_reg[i])) ;
assign tx_output_fix[i]   = tx_output[i]   ^ SWAP_MASK[i] ;
assign tx_output_fix[i+3] = tx_output[i+3] ^ SWAP_MASK[i] ;
end
endgenerate
```

In VHDL:

```
constant SWAP_MASK  : std_logic_vector(2 downto 0) := "010" ;

loop0 : for i in 0 to 2 generate
ibuf_d : obufds generic map (IOSTANDARD => "LVDS_25")
           port map    (i => tx_output_reg(i), o => dataout_p(i), oB =>
dataout_n(i));
fd_d  : oddr2  generic map (DDR_ALIGNMENT => "NONE")
          port map (c0 => clkin, c1 => notclock, d0 => tx_output_fix(i),
d1 => tx_output_fix(i+3), ce => '1', r => '0', s => '0', q =>
tx_output_reg(i));
tx_output_fix(i)   <= tx_output(i)   xor SWAP_MASK(i) ;
tx_output_fix(i+3) <= tx_output(i+3) xor SWAP_MASK(i) ;
end generate ;
```

This mechanism is easily extendable to various bit widths by changing the characters marked in red, boldface text.

As mentioned, bit manipulation can be important when using DDR techniques. The DDR generate loop example generates a bus whose low-order bits are transmitted on the falling edge of the clock and whose high-order bits are transmitted on the following rising edge.

Design Files

The design files for the various receiver and transmitter cases presented in this application note have been written for all Spartan-3 and Spartan-3E family devices. Both Verilog and VHDL design files are available from the Xilinx website (xapp491.zip). The enclosed `readme.txt` file provides the latest details.

Conclusion

With some planning and careful use of Spartan-3 Generation FPGA resources when designing with LVDS, the complexity of PCB layout can be substantially reduced, while improving the overall board signal integrity. This is true for the LVDS receivers and the LVDS transmitters incorporated within the device with the exception of the input clock pin, which has to have the correct polarity.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/04/06	1.0	Initial Xilinx release.