Interfacing Linear Technology's DDR LVDS ADCs to an Altera Stratix IV FPGA

In Collaboration with Fidus Systems, Inc.

1 DESCRIPTION

The purpose of this project is to provide reference code for receiving data from an LTC2158-14, dual channel 14-bit 310Msps parallel DDR LVDS interface ADC, with the Altera Stratix IV FPGA development board EP4SGX230KF40C2N.

The Linear Technology design circuit DC1564A-G features the LTC2158-14 ADC. This board provides two ADC channels (DATA_A_IN and DATA_B_IN). Each channel has a 7-bit data bus output in a double data rate format. A reference data clock (CLK_OUT) is also sent from the ADC into the FMC HPC connector present on the board. Using the DC1933A adapter board, a connection between the FMC connector-based Linear Technology board and the HSMC connector-based Altera FPGA board is made possible. The digitized signals coming from the LTC2158-14 will travel through the adapter board and then enter the FPGA fabric where they will be collected and packetized and then transmitted to the host computer over a USB connection for analysis and verification.

Note, in this design the 14-bit data coming from the ADC has been left justified to 16 bits inside the FPGA to ease future integration of 16-bit ADCs.

2 HARDWARE SETUP

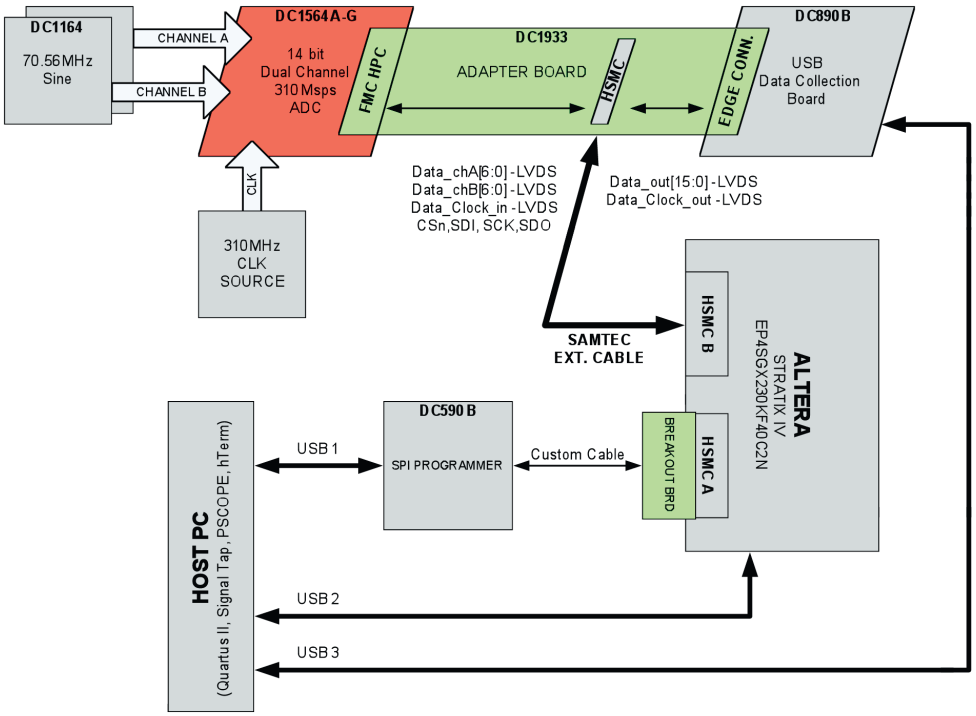

Figure 1. Hardware Setup Schematic

Table of Contents

2.2 Hardware Table

Item	Vendor	Description
DC1164A	Linear Technology	A 70.56MHz signal source
DC1564A-G	Linear Technology	Demo board for the LTC2158-14 ADC. (Device Under Test, DUT)
DC1933	Linear Technology	FMC HPC to HSMC adapter board and HSMC to edge connector
DC890B	Linear Technology	High speed (up to 250Mbps, CMOS/LVDS) USB data collection board
DC590B	Linear Technology	Generic SPI and I²C programmer
Breakout Board	Altera	Breakout board for the HSMC connector
SPI Cable	Custom	Custom cable from the DC590B pins to the user-selected breakout board pins
HSMC Ext. Cable	Samtec	14 in. HSMC to HSMC extension cable
Stratix IV	Altera	Altera Stratix IV development board EP4SGX230KF40C2N
Host PC		PC running Quartus II, SignalTap II, PScope™ and a terminal
8640B	Hewlett Packard	Clock generator source

2.1 Description of Hardware Setup

Two DC1164As feed two 70.56MHz sinusoidal waveforms into the DC1564A-G inputs. The magnitude switches are set to different values on the DC1164A boards in order to easily distinguish the signals from each other once plotted.

The adapter board provides a bridge between the HSMC connector and the FMC HPC connector. A very important point to notice is that the adapter board DC1933A, which connects the Linear Technology ADC board with the Stratix IV board, inverts all the differential pairs. This means that on a given differential pair connection between the DC1564A-G board and the Stratix IV board, the P channel from the Linear Technology board connects to the N channel on the Stratix IV board. Consequently, both the data buses and the clock will be inverted. Thus, besides the data bits being inverted, a 180 degree phase shift has been introduced between the clock signal and the data (refer to the Timing Analysis section for further details). The adapter board also inverts the differential pairs in the path from the Stratix IV HSMC to the DC890B edge connector.

The DC590B and the DC890B are optional for the overall system to work and are merely used for debugging and verification purposes.

The DC590B provides a simple and easy way to program the DC1564A-G through an SPI interface for debugging purposes. Through the SPI, the ADC can be programmed to output fixed digital known patterns, turn on the clock duty cycle stabilizer, clock phase delay and more (refer to the LTC2158-14 data sheet for more information on how to program the ADC).

The DC890B provides a real-time verification of our captured signal by displaying, in the time and frequency domain, the data forward by the FPGA using PScope software (see Appendix A for more info on PScope).

3 FPGA BLOCK DIAGRAM

In our test design, a deserialization factor of 4 is implemented in the FPGA receiver block. This will produce two samples, odd and even, for each channel at a frequency of 155MHz. Each sample is 16 bits wide (14-bit left-adjusted to 16-bit with padding 0's in the two LSBs). Using a deserialization factor of 4, not only allows us to run the FPGA logic at a lower speed but it also makes it possible to output either even (or odd) sample data at 155Msps to the DC890B for real-time data verification. Remember that the DC890B maximum speed is 250Msps so sending the data at full ADC capacity, 310Msps, would not work and a downsampling of 2 would be necessary before passing the data to the DC890B.

A factor of 4 deserialization scheme is shown in the block diagram in Figure 2.

Altera primitives, such as differential input buffers and DDR buffers, could be used to bring the data buses coming from the ADC into the FPGA and then route them into the SignalTap II block. Nonetheless, a more efficient, compact, and scalable approach is to use the Altera LVDS block. This megafunction provides a full LVDS receiver with dynamic phase alignment (DPA), PLL (internal or external), Bit Slip and deserialization functionality. Note that the DPA, Bit Slip, and deserialization path are bypassed if the deserialization factor is 2 (DDR case). In our design, the DPA and Bit Slip functionalities are not implemented.

A deserialization of 2 could also be used to achieve this data capture at a faster speed rate. Nonetheless, it would require a different time analysis than the one shown in the Timing Analysis section. A factor of 2 deserialization design is outside the scope of this document.

The Altera LVDS Megafunction receiver block diagram is shown in Figure 3.

Lastly, after the incoming data has been received, it needs to be collected and transmitted to the host computer to perform a data integrity check and make sure that the signal is captured properly.

The SignalTap II block provides an easy and fast way to record and output data to a host computer for data analysis and plotting. In our case, we are using a SignalTap II to record 8192 word samples for each channel and then transmit them to the host through USB.

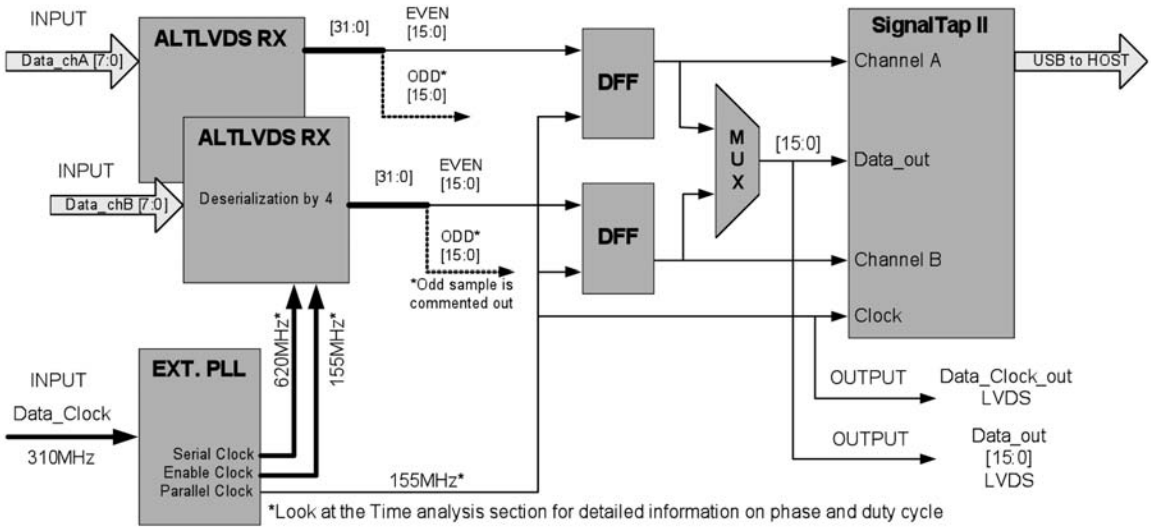

Figure 2. FPGA Block Diagram Depicting a Factor of 4 Deserialization Scheme

3.1 Altera LVDS Receiver Block Diagram

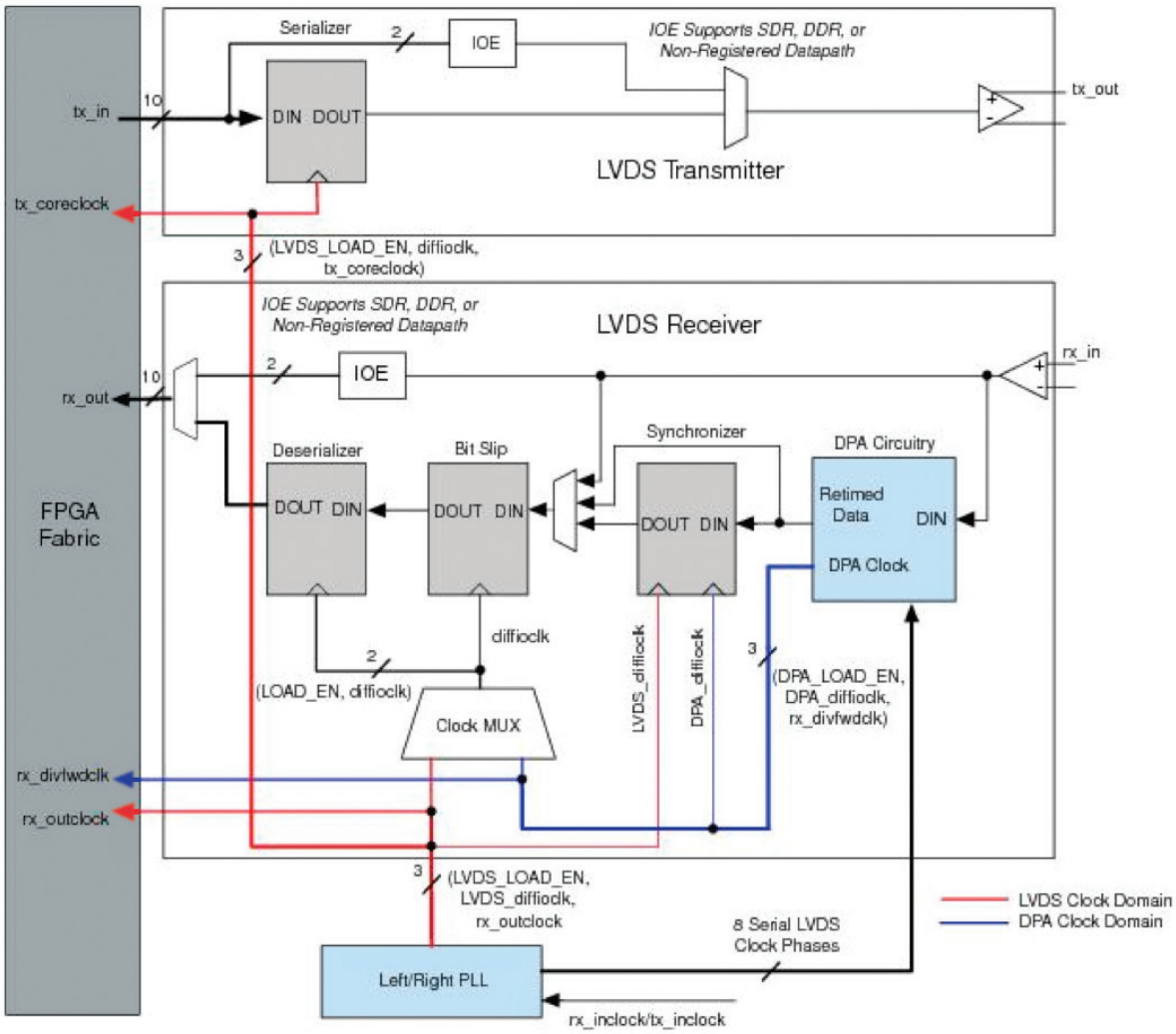

Figure 3. LVDS Megafunction Block Diagram

4 TIMING ANALYSIS

The Altera LVDS Megafunction receiver requires three tightly related clocks in order to execute properly. An external FAST PLL is used in our design to achieve aforementioned clocks. In Stratix IV devices, the Altera PLL Megafunction must be defined as a "Left-Right PLL" in order to be implemented as a FAST PLL.

The three needed clocks are:

- Serial Clock: used to clock the data into the receiver
- Enable Clock: used to enable the receiver
- Parallel Clock: used to register the output of the receiver and for the core logic

The following settings are used in the external PLL:

Serial Clock

- Frequency = 620MHz
  (2 • Input Clock frequency, since incoming data is in DDR format)
- Phase Shift = –180 degrees
- Duty Cycle = 50%

Enable Clock

- Frequency = 620/4 = 155MHz
  (Serial clock divided by deserialization factor)
- Phase Shift = (4 – 2) • 360/4 = 180 degrees
  [(deserialization factor – 2) • 360/deserialization factor]
- Duty Cycle = 100/4 = 25% (100/deserialization factor)

Parallel Clock

- Frequency = 620/4 = 155MHz
  (Serial clock divided by deserialization factor)
- Phase Shift = –180/4 = –45 degrees
  (serial clock phase shift divided by deserialization factor)
- Duty Cycle = 50%

IMPORTANT: Due to the 180 degrees phase shift introduced by the adapter board DC1933A into the input clock to the FPGA, the Altera ALTLVDS_RX Megafunction needs to be configured to trigger on the FALLING EDGE of the Serial Clock instead of the rising edge in order to compensate for this shift. If another adapter board is used in place of the DC1933A which does not introduce a 180 degree phase shift, then the ALTLVDS_RX should be configured to trigger on the RISING EDGE of the Serial Clock.

The output data is sent out of the FPGA along with the Parallel clock to the DC890B. Due to the 180 degree phase shift introduced again by the adapter board DC1933A, a output delay constraint is necessary in order to capture the data correctly and meet the setup and hold time requirements. In our design, a delay value of 2ns (about 1/3 of the Parallel clock frequency) is used.

For more information about timing and PLL settings for high speed LVDS receivers, refer to the Altera application notes listed in Appendix A.

5 DEMO

5.1 Demo Using DC590B and DC890B Boards

Before you can start running the demo and collecting data, the following software and drivers need to be installed:

- Quartus II with SignalTap II and USB-Blaster driver from Altera's website
- PScope from Linear Technology's website
- Any terminal such as HTERM to open a serial port communication

After installing the drivers and programs, set up the hardware as show in the Hardware Setup section of this document. Use the dedicated power supply when available and/or an external power supply to power up the various boards. **CAUTION**! Refer to each board data sheet for input power requirements. Power up all the boards and turn on the input clock to the ADC board.

In out setup, the DC1564A-G jumper JP1 is set to serial mode and the SPI interface is looped through the FPGA to the DC590B. Parallel mode is also supported but the FPGA needs to drive the SPI pins for proper ADC operation. Refer to the next section for more information on parallel mode.

Some setups may benefit from programming the DC1564A-G to turn on the clock duty cycle stabilizer. In our setup, this step is not necessary. Nevertheless, if you want to do this, set the jumper on the DC1564A-G board to serial programming. On the host machine, open a serial communication with the DC590B using any terminal application. Send the command (in ASCII) xS02S01XZ. A "/n" character will be returned if the transmission was successful. If a DC590B board is not available to you please refer to the next section in this document for more information on how to set the duty cycle stabilizer while in parallel mode.

On the host machine, open the "stp1.stp" under the project folder. The Quartus II suite and SignalTap II tool will start. Make sure the Stratix IV board is connected to the host and the board is powered on. In the SignalTap II window setup the JTAG chain by clicking on the SETUP button in the upper right corner of the screen. Once the JTAG chain has been detected, attach the "linear_1564a.sof" file found in the project folder and program the device.

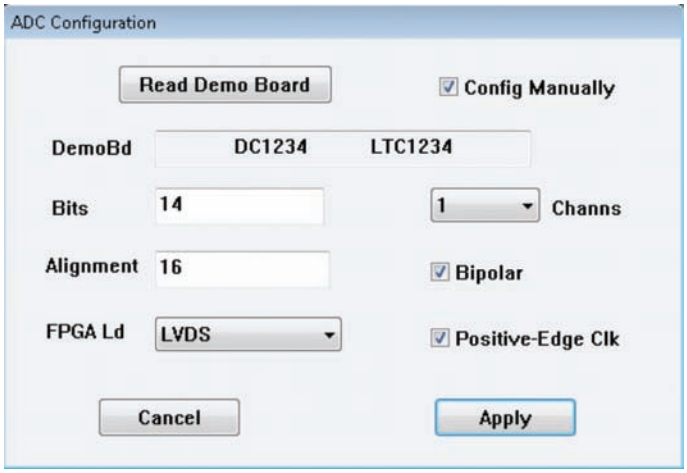

Figure 4. Network Architecture for Refinery Process Control

Once programmed, click the “Run” button in SignalTap II to start capturing and displaying data. SignalTap II will continue capturing data as long as there is a valid incoming signal.

Leaving SignalTap II running, open PScope. Make sure the DC890B board is on and that the software detects it properly. Click “Configure→ADC Configuration…” on the top right corner. Set the different parameters as shown in the image bellow and click “APPLY."

Return to the main screen. Choose 8192 as “Size” and a “Blkmn-Harris 92dB” windowing. Click “COLLECT." Data in the time and frequency domain will start displaying.

You can choose which channel to display by switching the dip switch SW3.1 on and off on the Stratix IV development board.

5.2 Demo without Using DC590B and DC890B Boards

If a DC590B board is not available to you, the clock duty cycle stabilizer can be turned on by setting the jumper JP1 on the DC1564A-G to parallel mode and by driving the CSn line HIGH from the FPGA. For SDI and SCK, follow SPI pin configuration on Table 2 of the LTC2158-14 data sheet for parallel programming and drive those pins accordingly from the FPGA.

If a DC890B board is not available to you, data can be exported from SignalTap II and saved as a “.csv” file. From SignalTap II click File→Export. Data can be exported in different formats. To select a different format right click any of the data buses in SignalTap II. A menu opens up. Select Bus Display Format and chose the format that more suits your needs. After the format is selected, save and export the file to a .csv.

The data can now be imported into other tools such as Matlab or PScope for post analysis.

In order to import the SignalTap II data into PScope, follow these next steps. Stop collecting data on PScope. Go to File→save data as… This will create an .adc file. Open the .adc file with a text editor such as a Notepad++. Replace the data shown in the .adc file with the data from the .csv file (the data format in the .csv file needs to be Sign Decimal). Remember that the data in .csv is four times larger since we left-adjusted the data to 16 bits and therefore it needs to be adjusted. After replacing the values, save the .adc file. In PScope, click File→Load Data…and select the modified .adc file.

6 APPENDIX A: REFERENCE MATERIAL

Software Tools and Components Data Sheets

Tools/Component	Data Sheet / Source
Quartus II Software	http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html
SignalTap II Tool	http://www.altera.com/literature/hb/qts/qts_qii53009.pdf
PScope Software	http://www.linear.com/designtools/software/
DC1164A Board	http://www.linear.com/demo/DC1164A
DC1564A-G Board	http://www.linear.com/demo/DC1564A-G
DC1933A Board	http://www.linear.com/demo
DC890B Board	http://www.linear.com/demo/DC890B
DC590B Board	http://www.linear.com/demo/DC590B
Samtec Extension Cable	http://www.samtec.com/signal_integrity/altera.aspx
HSMC Connector Specification	http://www.altera.com/literature/ds/hsmc_spec.pdf?GSA_pos=1&WT.oss_r=1&WT.oss=hsmc%20breakout%20board
Altera Stratix IV Development Board EP4SGX230KF40C2N	http://www.altera.com/products/devkits/altera/kit-siv-gx.html
Altera Breakout Board	For more information, refer to the breakout board schematic included in the Design Folder.

7 APPENDIX A: REFERENCE MATERIAL

Recommended Reading

Publication	Source
"High-Speed Differential I/O Interfaces and DPA in Stratix IV Devices"	http://www.altera.com/literature/hb/stratix-iv/stx4_siv51008.pdf
"Design Example Using ALTLVDS Megafunction & the External PLL Option in Stratix II Devices"	http://www.altera.com/literature/hb/qts/qts_qii53009.pdf
"LVDS SERDES Transmitter / Receiver (ALTLVDS_TX and ALTLVDS_RX) Megafunction User Guide"	http://www.altera.com/literature/ug/ug_altlvds.pdf
"Phase-Locked Loop (ALTPLL) Megafunction User Guide"	http://www.altera.com/literature/ug/ug_altpll.pdf

8 APPENDIX B: SOURCE CODE AND TIME CONSTRAINT FILE

```
//-----------------------------------------------------------------------------
// Company      : Fidus Systems
// Date         : 07/25/2012
// Designer     : Javier Rodriguez Molina
// Contact      : j.rodriguez.molina@fidus.com
// Design Name  : DC1564A 14 bit ADC
// File Name    : linear_1564a.v
// Hardware     : Stratix IV development board DK-DEV-4SGX230N
// Description  :
// This design captures incoming data from the Linear Technology DC1654A-G board.
// The dual channel 14 bit 310Msps DDR LVDS ADC feeds data to the FPGA. The FPGA
// captures the 7 bit bus (configurable to 8) and deserializes the stream by a
// factor of 4 producing 2 samples (even and odd) at half the speed of the
// incoming data (in this case 155MHz). The captured data from either the odd
// or even sample is outputed to the HSMC connector on the Stratix development
// board for testing and it is also captured by Signal Tap. Refer to the
// DC1564A Design Document for more information
//-----------------------------------------------------------------------------
module linear_1564a
(
 //Input Clock
  input                 CLK_IN          ,
  //Input Data
  input      [7:0]      DATA_A_IN       ,
  input      [7:0]      DATA_B_IN       ,
  //User
  input                 CH_SEL          ,
  //SPI loopback
  output                SCK             ,
  output                SDI             ,
  input                 SDO             ,
  output                CSn             ,
  input                 CSn_in          ,
  input                 SCK_in          ,
  input                 SDI_in          ,
  output                SDO_out         ,
  //Output lock
  output                CLK_OUT         ,
  //Output data
  output    [15:0]      DATA_OUT
);
////////////////////////////////////////////////////////////////////////////////
```

8 APPENDIX B: SOURCE CODE AND TIME CONSTRAINT FILE

```
//*********************************************************************************
//*********************************************************************************
//*********************************************************************************
//*********************************************************************************
////////////////////////////////////////////////////////////////////////////////
// Signal Declaration
wire [31:0]       data_samples_chA;
wire [31:0]       data_samples_chB;
wire [15:0]       data_align_chA;
wire [15:0]       data_align_chB;
wire [15:0]       data_out_chA;
wire [15:0]       data_out_chB;
wire              clk_620mhz;
wire              clk_en;
wire              pll_locked;
wire              clk_155mhz;
//SPI assignment. Modify based on your SPI interface needs
//Refer to the DC1564A Design Document for more information
assign    SCK      = SCK_in;
assign    SDI      = SDI_in;
assign    CSn      = CSn_in;
assign    SDO_out  = SDO;
//Output Data Assignment -- User Dip Switch SW3.1 -- OFF => chA ON => chB
assign DATA_OUT = (CH_SEL)? data_out_chA : data_out_chB;
////////////////////////////////////////////////////////////////////////////////
//Choose sample to output.
//Uncomment EITHER the even or odd sample to output from channel A and B.
//With a deserialization factor of 4 we obtain two 16 bit samples @ 155MHz
//(the 2 LSB are for future used and left are connected to ground) but only
//one of the samples is transmitted. This creates an effective down sample of 2.
//The MSB its inverted to convert from the data format of the DC1564A-G to the
//data format expected by the DC890B. (Unsigned decimal -> Signed decimal)
//The capture data from the ADC is inverted before entering the FPGA and it is
//inverted when exciting the FPGA by the adapter board. Keep in mind the data
//inside the FPGA it is inverted from its original value.
////////////////////////////////////////////////////////////////////////////////
//Even Sample Channel A
assign data_align_chA = {~data_samples_chA[30],data_samples_chA[31],
                  data_samples_chA[26],data_samples_chA[27],
                  data_samples_chA[22],data_samples_chA[23],
                  data_samples_chA[18],data_samples_chA[19],
                  data_samples_chA[14],data_samples_chA[15],
                  data_samples_chA[10],data_samples_chA[11],
                  data_samples_chA[6],data_samples_chA[7],
                  1'b0,1'b0};                              //Delete for 16 bit ADC
                  //data_samples_chA[2],data_samples_chA[3]}; //Uncomment for 16 bit ADC
//Odd Sample Channel A
//assign data_align_chA = {~data_samples_chA[28],data_samples_chA[29],
//                  data_samples_chA[24],data_samples_chA[27],
//                  data_samples_chA[20],data_samples_chA[23],
//                  data_samples_chA[16],data_samples_chA[19],
//                  data_samples_chA[12],data_samples_chA[15],
//                  data_samples_chA[8],data_samples_chA[11],
//                  data_samples_chA[4],data_samples_chA[7],
//                  1'b0,1'b0};                                //Delete for 16 bit ADC
//                  //data_samples_chA[0],data_samples_chA[1]}; //Uncomment for 16 bit ADC
//Even Sample Channel B
assign data_align_chB = {~data_samples_chB[30],data_samples_chB[31],
                  data_samples_chB[26],data_samples_chB[27],
```

8 APPENDIX B: SOURCE CODE AND TIME CONSTRAINT FILE

```
                    data_samples_chB[22],data_samples_chB[23],
                    data_samples_chB[18],data_samples_chB[19],
                    data_samples_chB[14],data_samples_chB[15],
                    data_samples_chB[10],data_samples_chB[11],
                    data_samples_chB[6],data_samples_chB[7],
                    1'b0,1'b0};                                  //Delete for 16 bit ADC
                    //data_samples_chB[2],data_samples_chB[3]}; //Uncomment for 16 bit ADC
//Odd Sample Channel B
//assign data_align_chA = {~data_samples_chB[28],data_samples_chB[29],
//                    data_samples_chB[24],data_samples_chB[27],
//                    data_samples_chB[20],data_samples_chB[23],
//                    data_samples_chB[16],data_samples_chB[19],
//                    data_samples_chB[12],data_samples_chB[15],
//                    data_samples_chB[8],data_samples_chB[11],
//                    data_samples_chB[4],data_samples_chB[7],
//                    1'b0,1'b0};                                   //Delete for 16 bit ADC
//                    //data_samples_chB[0],data_samples_chB[1]}; //Uncomment for 16 bit ADC
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
// Instantiate pll
ext_pll ext_pll_inst (
  .areset   (                     ),
  .inclk0   ( CLK_IN              ),  // Input clock    -  310MHz
  .c0       ( clk_620mhz          ),  // Serial clock   -  620MHz
  .c1       ( clk_en              ),  // Enable clock   -  155Mhz
  .c2       ( clk_155mhz          ),  // Parallel clock -  155MHz
  .locked   ( pll_locked          )   //
  );
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//The 180 degrees phase shift introduced into the input clock
//by the adapter board DC1933A is removed by triggering the
//LVDS receiver on the FALLING EDGE of the clock instead of
//the rising edge.
//This fix applies only in the case of using the DC1933A
//adapter board and needs to be modified if not.
////////////////////////////////////////////////////////////////////////////////
//LVDS Receiver Channel A
lvds_rx_ext_pll lvds_receiver_chA (
  .rx_enable            ( clk_en                  ),
  .rx_in                ( DATA_A_IN               ),
  .rx_inclock           ( clk_620mhz              ),
  .rx_out               ( data_samples_chA        )
  );
//LVDS Receiver Channel B
lvds_rx_ext_pll lvds_receiver_chB (
  .rx_enable            ( clk_en                  ),
  .rx_in                ( DATA_B_IN               ),
  .rx_inclock           ( clk_620mhz              ),
  .rx_out               ( data_samples_chB        )
  );
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//The 180 degrees phase shift introduced into the output clock
//by the adapter board DC1933A is compensated by introducing an
//an output delay to the Data_out paths with respect to the
//the output clock.
//This fix applies only in the case of using the DC1933A
//adapter board and needs to be modified if not.
```

8 APPENDIX B: SOURCE CODE AND TIME CONSTRAINT FILE

```
////////////////////////////////////////////////////////////////////////////////
// LVDS Transmitter: Output Clock
lvds_tx_pass_thru clk_out_lvds (
  .tx_in          ( clk_155mhz        ),
  .tx_out         ( CLK_OUT           )
  );
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
// Deserialized data coming out from the LVDS receiver needs to be registered
genvar i;
generate
  for(i = 0; i < 16; i = i + 1) begin: GEN_LOOP
  DFF data_chA_reg (
    .d              (data_align_chA[i]              ),
    .clk            (clk_155mhz                     ),
    .clrn           (1'b1                           ),
    .prn            (1'b1                           ),
    .q              (data_out_chA[i]                )
    );
  DFF data_chB_reg (
    .d              (data_align_chB[i]              ),
    .clk            (clk_155mhz                     ),
    .clrn           (1'b1                           ),
    .prn            (1'b1                           ),
    .q              (data_out_chB[i]                )
    );
  end
endgenerate
////////////////////////////////////////////////////////////////////////////////
endmodule



********************************************************************************
** Company      : Fidus Systems
** Date         : 07/25/2012
** Designer     : Javier Rodriguez Molina
** Contact      : j.rodriguez.molina@fidus.com
** Design Name  : DC1564A 14 bit ADC
** File Name    : linear_1564a.sdc
** Hardware     : Stratix IV development board DK-DEV-4SGX230N
** Description  :
** Time constraints for DC1564A-G 14 bit ADC design
********************************************************************************
#*******************************************************************************
# Time Information
#*******************************************************************************
set_time_format -unit ns -decimal_places 3
#*******************************************************************************
# Create Clock
#*******************************************************************************
#SignalTap II clock
create_clock -name {altera_reserved_tck} -period 100.000 -waveform { 0.000 50.000 } [get_ports {altera_reserved_tck}]
#Main Input clock
create_clock -name {clk1} -period 3.225 -waveform { 0.000 1.667 } [get_ports { CLK_IN }]
#*******************************************************************************
# Create Generated Clock
#*******************************************************************************
#derive pll clocks
create_generated_clock  -name  {ext_pll_inst|altpll_component|auto_generated|pll1|clk[0]}  -source  [get_pins  {ext_pll_inst|altpll_component|auto_
```

8 APPENDIX B: SOURCE CODE AND TIME CONSTRAINT FILE

```
generated|pll1|inclk[0]}] -duty_cycle 50.000 -multiply_by 2 -phase -180.000 -master_clock {clk1} [get_pins {ext_pll_inst|altpll_component|auto_
generated|pll1|clk[0]}]
create_generated_clock -name {ext_pll_inst|altpll_component|auto_generated|pll1|clk[1]} -source [get_pins {ext_pll_inst|altpll_component|auto_
generated|pll1|inclk[0]}] -duty_cycle 25.000 -multiply_by 1 -divide_by 2 -phase 180.000 -master_clock {clk1} [get_pins {ext_pll_inst|altpll_component|auto_
generated|pll1|clk[1]}]
create_generated_clock -name {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]} -source [get_pins {ext_pll_inst|altpll_component|auto_
generated|pll1|inclk[0]}] -duty_cycle 50.000 -multiply_by 1 -divide_by 2 -phase -45.000 -master_clock {clk1} [get_pins {ext_pll_inst|altpll_component|auto_
generated|pll1|clk[2]}]
#create output clock
create_generated_clock -name {clk_out} -source [get_pins {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]}] -duty_cycle 50.000 -multiply_by 1
-master_clock {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]} [get_ports { CLK_OUT }]
#**************************************************************************
# Set Clock Latency
#**************************************************************************
#**************************************************************************
# Set Clock Uncertainty
#**************************************************************************
set_clock_uncertainty -rise_from [get_clocks {altera_reserved_tck}] -rise_to [get_clocks {altera_reserved_tck}] 0.020
set_clock_uncertainty -rise_from [get_clocks {altera_reserved_tck}] -fall_to [get_clocks {altera_reserved_tck}] 0.020
set_clock_uncertainty -fall_from [get_clocks {altera_reserved_tck}] -rise_to [get_clocks {altera_reserved_tck}] 0.020
set_clock_uncertainty -fall_from [get_clocks {altera_reserved_tck}] -fall_to [get_clocks {altera_reserved_tck}] 0.020
set_clock_uncertainty -rise_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[0]}] -rise_to [get_clocks {ext_pll_inst|altpll_
component|auto_generated|pll1|clk[2]}] 0.050
set_clock_uncertainty -rise_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[0]}] -fall_to [get_clocks {ext_pll_inst|altpll_
component|auto_generated|pll1|clk[2]}] 0.050
set_clock_uncertainty -fall_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[0]}] -rise_to [get_clocks {ext_pll_inst|altpll_
component|auto_generated|pll1|clk[2]}] 0.050
set_clock_uncertainty -fall_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[0]}] -fall_to [get_clocks {ext_pll_inst|altpll_
component|auto_generated|pll1|clk[2]}] 0.050
set_clock_uncertainty -rise_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]}] -rise_to [get_clocks {ext_pll_inst|altpll_
component|auto_generated|pll1|clk[2]}] 0.020
set_clock_uncertainty -rise_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]}] -fall_to [get_clocks {ext_pll_inst|altpll_
component|auto_generated|pll1|clk[2]}] 0.020
set_clock_uncertainty -fall_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]}] -rise_to [get_clocks {ext_pll_inst|altpll_
component|auto_generated|pll1|clk[2]}] 0.020
set_clock_uncertainty -fall_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]}] -fall_to [get_clocks {ext_pll_inst|altpll_
component|auto_generated|pll1|clk[2]}] 0.020
set_clock_uncertainty -rise_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]}] -rise_to [get_clocks {clk_out}] 0.050
set_clock_uncertainty -rise_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]}] -fall_to [get_clocks {clk_out}] 0.050
set_clock_uncertainty -fall_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]}] -rise_to [get_clocks {clk_out}] 0.050
set_clock_uncertainty -fall_from [get_clocks {ext_pll_inst|altpll_component|auto_generated|pll1|clk[2]}] -fall_to [get_clocks {clk_out}] 0.050
#**************************************************************************
# Set Input Delay
#**************************************************************************
#This delay is NOT needed since alignment between the rising
#edge of the input clock and the incoming data is performed in
#the external PLL
#**************************************************************************
# Set Output Delay
#**************************************************************************
#This delay is needed to properly align the rising edge of the output
#clock to the output data
set_output_delay -add_delay -clock [get_clocks {clk_out}] 2.000 [get_ports {DATA_OUT[0]}]
set_output_delay -add_delay -clock [get_clocks {clk_out}] 2.000 [get_ports {DATA_OUT[0](n)}]
set_output_delay -add_delay -clock [get_clocks {clk_out}] 2.000 [get_ports {DATA_OUT[1]}]
set_output_delay -add_delay -clock [get_clocks {clk_out}] 2.000 [get_ports {DATA_OUT[1](n)}]
set_output_delay -add_delay -clock [get_clocks {clk_out}] 2.000 [get_ports {DATA_OUT[2]}]
set_output_delay -add_delay -clock [get_clocks {clk_out}] 2.000 [get_ports {DATA_OUT[2](n)}]
set_output_delay -add_delay -clock [get_clocks {clk_out}] 2.000 [get_ports {DATA_OUT[3]}]
```

8 APPENDIX B: SOURCE CODE AND TIME CONSTRAINT FILE

```
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[3](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[4]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[4](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[5]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[5](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[6]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[6](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[7]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[7](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[8]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[8](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[9]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[9](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[10]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[10](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[11]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[11](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[12]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[12](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[13]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[13](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[14]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[14](n)}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[15]}]
set_output_delay -add_delay  -clock [get_clocks {clk_out}]  2.000 [get_ports {DATA_OUT[15](n)}]
#**************************************************************************
# Set Clock Groups
#**************************************************************************
set_clock_groups -asynchronous -group [get_clocks {altera_reserved_tck}]
#**************************************************************************
# Set False Path
#**************************************************************************
#set_false_path -to [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_0|datain}]
#set_false_path -to [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_1|datain}]
#set_false_path -to [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_2|datain}]
#set_false_path -to [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_3|datain}]
#set_false_path -to [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_4|datain}]
#set_false_path -to [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_5|datain}]
#set_false_path -to [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_6|datain}]
#set_false_path -to [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_7|datain}]
#set_false_path -to [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_0|datain}]
#set_false_path -to [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_1|datain}]
#set_false_path -to [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_2|datain}]
#set_false_path -to [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_3|datain}]
#set_false_path -to [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_4|datain}]
#set_false_path -to [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_5|datain}]
#set_false_path -to [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_6|datain}]
#set_false_path -to [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_7|datain}]
#**************************************************************************
# Set Multicycle Path
#**************************************************************************
#These multicycle assignment are necessary to properly hand off
#data from the serial clock (fast clock) to the parallel clock
#(slow clock)
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_0|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_0|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_1|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_1|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_2|clk0}] 4
```

8 APPENDIX B: SOURCE CODE AND TIME CONSTRAINT FILE

```
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_2|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_3|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_3|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_4|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_4|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_5|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_5|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_6|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_6|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_7|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chA|ALTLVDS_RX_component|auto_generated|rx_7|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_0|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_0|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_1|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_1|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_2|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_2|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_3|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_3|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_4|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_4|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_5|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_5|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_6|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_6|clk0}] 3
set_multicycle_path -setup -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_7|clk0}] 4
set_multicycle_path -hold  -start -from [get_pins { lvds_receiver_chB|ALTLVDS_RX_component|auto_generated|rx_7|clk0}] 3
#**************************************************************************
# Set Maximum Delay
#**************************************************************************
#**************************************************************************
# Set Minimum Delay
#**************************************************************************
#**************************************************************************
# Set Input Transition
#**************************************************************************
```

9 REVISION HISTORY

Date	Revised By	Reason
7/26/2012	J. Rodriquez Molina	File Created

Linear Technology Corporation
1630 McCarthy Blvd. Milpitas, CA 95035-7417
Tel: 408 432-1900 • FAX: (408) 434-0507 • www.linear.com

LT 0912 • PRINTED IN USA