# Using LVDS in the Quartus Software

## Introduction

Low-voltage differential signaling (LVDS) in APEX™ 20KE devices is Altera's solution for the continuously increasing demand for high-speed data-transfer at low power consumption rates. APEX 20KE devices are designed with dedicated LVDS circuitry that supports transfer rates of up to 840 megabits per second (Mbps). A total of 16 transmitter and 16 receiver channels provide up to 27 gigabits per second (Gbps) of total LVDS bandwidth in a single device. Designs that take advantage of dedicated APEX 20KE LVDS circuitry are implemented using the Quartus™ software, Altera's next-generation development system for programmable logic. LVDS can be easily implemented in APEX devices using the Quartus software and the `altlvds` megafunction, saving design time and reducing board space.

This white paper describes how to use the Quartus development tool with designs that enable LVDS and its various features in APEX devices. For more information about LVDS in APEX 20KE devices, see the *Using LVDS in APEX 20KE Devices White Paper*.

## LVDS in APEX 20KE Devices

LVDS is a low-voltage swing I/O standard that meets performance requirements for high-speed, low-power, and low-noise applications. LVDS transfers data via differential signaling instead of less efficient single-ended techniques.

LVDS is characterized by two IEEE standards: IEEE std. 1596.3 SCI-LVDS and ANSI/TIA/EIA-644. The ANSI/TIA/EIA-644 standard defines driver output and receiver input characteristics at a maximum data rate of 655 Mbps and a theoretical maximum of 1.923 gigabits per second (Gbps).

A differential scheme is used in LVDS instead of a single-ended scheme because of its immunity to electromagnetic interference (EMI). Also, because the noise margin is significantly greater for differential signals, the voltage swing can be minimized to approximately 350 mV, thereby reducing power consumption.

APEX 20KE devices feature phase-locked loops (PLLs) with enhanced ClockLock™, ClockBoost™, and ClockShift™ circuitry. The ClockLock circuitry uses a synchronizing PLL with an extended frequency range that reduces the clock delay and skew within the device. The ClockBoost circuitry, which provides a clock multiplier, allows the designer to enhance device area and efficiency by sharing resources within the device. The ClockShift feature allows the clock phase and delay to be adjusted. The PLL is the key to enabling the transmission of data at such high rates. More information about PLLs can be found in the *Using APEX 20K & APEX 20KE PLLs in the Quartus Software White Paper*.

The device's PLLs generate the high frequency clock signals that are required for serial-to-parallel and parallel-to-serial conversion. The example in Figure 1 shows a block diagram of the LVDS circuitry and how it interfaces with user logic and the LVDS PLLs. In this case, the LVDS transmitter converts a maximum of 128 CMOS on-chip data bits into 16 LVDS data streams, using an 8-to-1 parallel-to-serial converter. Similarly, the LVDS receiver converts a maximum of 16 LVDS signals into 128 CMOS data bits that feed internal logic elements (LEs) within the device.

Operation can occur in one of four modes: 1×, 4×, 7×, or 8× mode. The mode of conversion is specified by the deserialization factor. When operation occurs in 1× mode, the dedicated LVDS circuitry is bypassed and the data is fed directly in and out of the internal LEs. When operation occurs in 4×, 7×, or 8× mode, data passes through either the dedicated serial-to-parallel or parallel-to-serial converters, which have clocks generated by the device's PLLs.

The transmitter PLL's input clock can be driven by an off-chip clock source or internally by the output clock of the receiver PLL via an internal global net. The output LVDS clock of the transmitter PLL can only be driven off-chip in

A-WP-LVDSQUARTUS-01

1× mode, in-phase with the LVDS data being driven out, and cannot feed internal logic. Similarly, the receiver PLL's input clock can be driven from an off-chip source, and its output clock can feed internal logic in 1× mode. However, the output clock cannot be driven off-chip.

For EP20K200E devices and smaller in ball-grid array (BGA) packages, all devices support LVDS clock inputs. Devices equipped with PLLs (denoted by a "-X" suffix in the ordering code) can drive out an LVDS clock and accept LVDS feedback. LVDS inputs and outputs are not supported.

All EP20K300E devices in BGA packages support LVDS clock inputs. All devices support 16 LVDS input and output channels in 1× mode. Devices with PLLs can drive out an LVDS clock and accept LVDS feedback.

Devices larger than a EP20K300E device support LVDS clock inputs, outputs, and feedback. These devices also support 16 LVDS input and output channels in 1× mode. For devices with PLLs, full LVDS support is available, including all operating modes. LVDS support is summarized in Table 1.

*Table 1. LVDS Support in APEX 20KE Devices*

| Device Density | Feature | Devices with PLLs | Devices without PLLs |
|---|---|---|---|
| EP20K200E and smaller | LVDS Clock | Input, output, and feedback | Input |
| | LVDS I/O pins | Not supported | Not supported |
| EP20K300E | LVDS Clock | Input, output, and feedback | Input |
| | LVDS I/O pins | 1× mode | 1× mode |
| EP20K400E and larger | LVDS Clock | Input, output, and feedback | Input |
| | LVDS I/O pins | All modes | 1× mode |

*Figure 1. LVDS Receiver and Transmitter Interface*
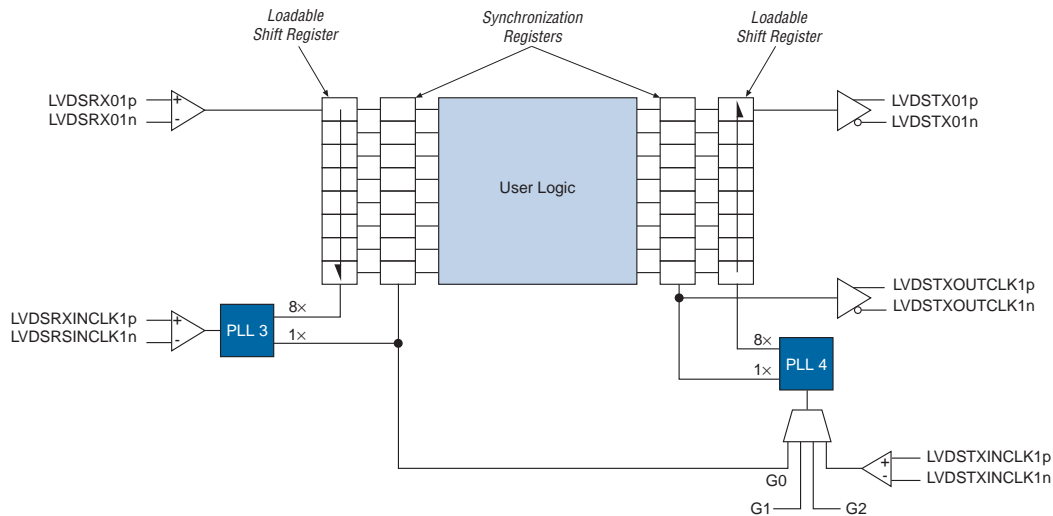


EP20K300E devices and larger contain 4 PLLs. Two of the PLLs are available for LVDS applications. PLL 4 is used for the LVDS transmitter, and PLL 3 is used for the LVDS receiver. Figure 2 displays a block diagram of the LVDS PLLs.

*Figure 2. LVDS PLL Block Diagram*
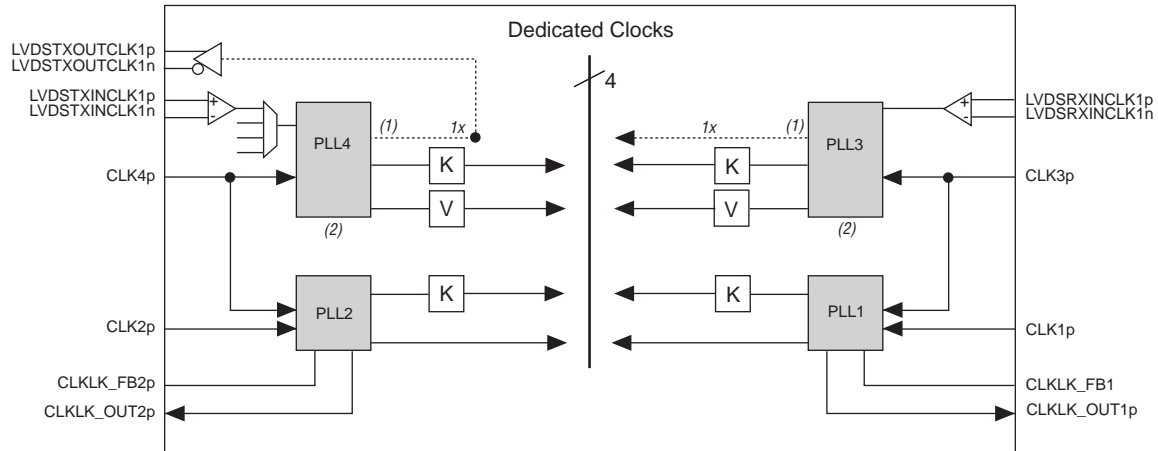


The operation frequencies are specified by the limitations of the PLLs in the device at a given deserialization factor. The input frequency range is limited to 50-80 MHz in the Quartus software version 2000.05 and lower when the deserialization factor is 4, 30-80 MHz when the deserialization factor is 7, and 30-78 MHz when the deserialization factor is 8. Although the maximum input frequency in the Quartus software is currently limited to 78 MHz in 8× mode, APEX 20KE devices support transfer rates of 840 Mbps and input frequencies of up to 105 MHz. 840 Mbps performance is achieved in APEX 20KE devices by selecting 8× mode for the deserialization factor and 78 MHz for the input frequency in the Quartus software; the 78 MHz setting only limits simulation support to 622 Mbps. Future versions of the Quartus software will support the 105 MHz input clock setting for full simulation capabilities.

LVDS in APEX 20KE devices can be configured in several ways. The device can be used only as an LVDS transmitter or receiver that accepts parallel CMOS or serialized LVDS data from an off-chip source and outputs serialized LVDS or parallel CMOS data to an off-chip destination. In both cases, the number of channels is limited to 16 or less. Alternatively, the device can accommodate both a single transmitter and a single receiver. Both modules can receive parallel CMOS or serial LVDS data from an off-chip source and output to an off-chip destination. The device can also be configured so that the receiver receives inputs from an external source, and outputs to an internal signal processing logic block that subsequently outputs to the transmitter. The transmitter can then output to an off-chip destination. This configuration is also limited to 16 channels or less.

The placement of LVDS pin/pads are restricted according to predefined banking rules. LVDS blocks cannot be located near non-LVDS output pins that share the same $V_{CCIO}$ bus because any switching that occurs on them may degrade performance. For more information on pin/pad placement, refer to the *Using I/O standards in the Quartus Software White Paper*.

Deskew circuitry is also available in APEX 20KE devices, which reduces the effects of channel-to-channel skew and clock-to-channel skew within the device itself, as seen in Figure 3.

*Figure 3. Channel-to-Channel and Clock-to-Channel Skew*
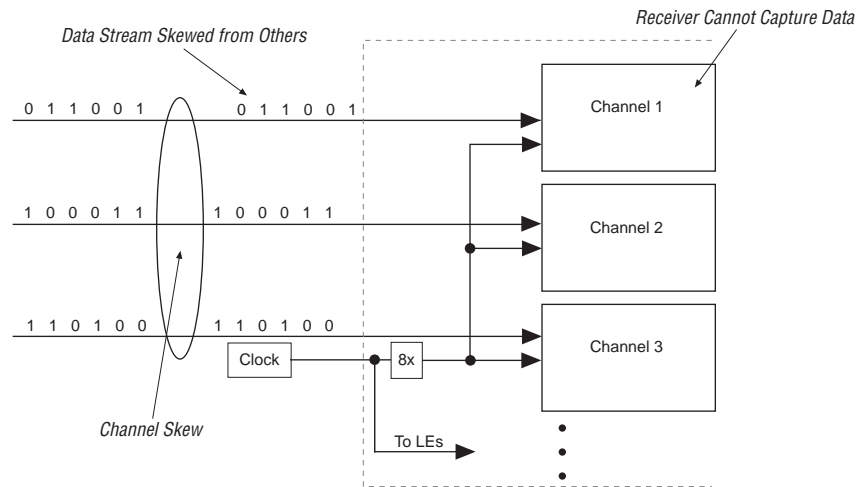


Because of the high bandwidth of the LVDS inputs in 8× and 7× modes, LVDS data is captured with an over-sampling circuit. These inputs are captured by four separate clocks and subsequently compared to determine which clock successfully captured the data. The deskew circuitry can compensate up to ±25% of the bit time period. For 4× mode, calibration is optional.

To ensure that the data is captured accurately, the LVDS receiver must be calibrated properly. When the `deskew` pin is asserted, the receiver is put into calibration mode. A calibration pattern must be applied to every input channel for at least three clock cycles to phase-align the clock with the incoming LVDS data, as seen in the example for 8× mode in Figure 4.

*Figure 4. Deskew Circuitry Calibration Waveform for 8× Mode*
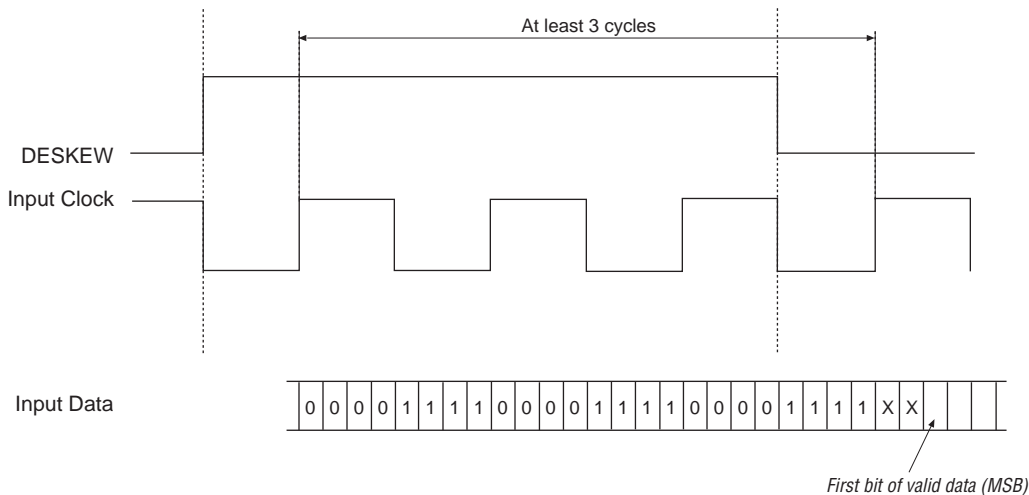
The calibration pattern depends on the deserialization factor as seen in Table 2.

*Table 2. Calibration Data Pattern for Deskew Circuitry*

| Deserialization Factor | Calibration Pattern |
|:---:|:---:|
| 4 | 0011 |
| 7 | 0000111 |
| 8 | 00001111 |

Each LVDS input channel is calibrated separately because of the differences in routing; therefore, the calibration pattern must be present on each of the LVDS channels. The first bit of the calibration data is the first bit after the input clock. For more information on APEX 20KE deskew circuitry, see the *Using LVDS in APEX 20KE Devices White Paper*.

Data synchronization is necessary for successful data transmission at high frequencies. Figure 5 shows the data bit orientation for a receiver channel operating in 8× mode. Unlike in calibration mode, the first bit of data for the current clock cycle is the third bit because the first two bits belong to the previous cycle. Similar positioning exists for the most significant bits (MSBs) and LSBs after deserialization, as seen in Table 3.

*Figure 5. Bit Order for One Channel of LVDS Data*
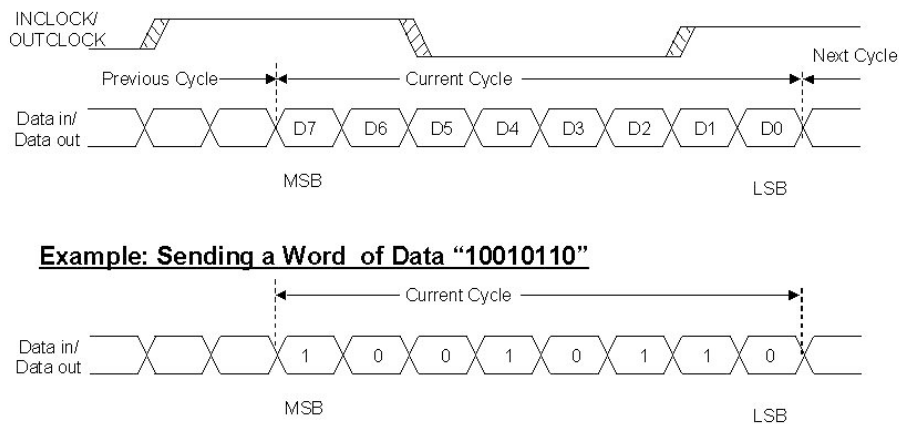


Table 3 shows the conventions for LVDS bit naming.

*Table 3. LVDS Bit Naming  (Sheet 1 of 2)*

| Rx Data Channel Number | Internal CMOS 8-bit Parallel Data | |
|:---:|:---:|:---:|
|  | **MSB Position** | **LSB Position** |
| 1 | 7 | 0 |
| 2 | 15 | 8 |
| 3 | 23 | 16 |
| 4 | 31 | 24 |
| 5 | 39 | 32 |
| 6 | 47 | 40 |
| 7 | 55 | 48 |
| 8 | 63 | 56 |
| 9 | 71 | 64 |
| 10 | 79 | 72 |
| 11 | 87 | 80 |

*Table 3. LVDS Bit Naming  (Sheet 2 of 2)*

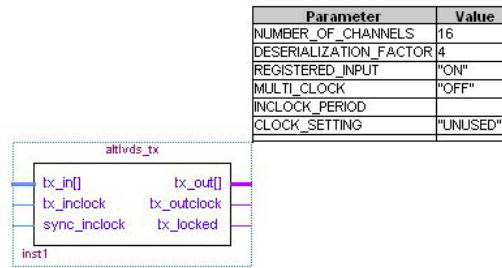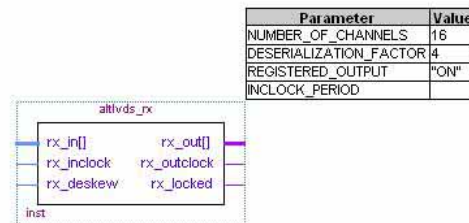| Rx Data Channel Number | Internal CMOS 8-bit Parallel Data | |
|---|---|---|
| | **MSB Position** | **LSB Position** |
| 12 | 95 | 88 |
| 13 | 103 | 96 |
| 14 | 111 | 104 |
| 15 | 119 | 112 |
| 16 | 127 | 120 |

Table 4 shows the pin naming convention used with all APEX 20KE devices.

*Table 4. LVDS Pin Naming*

| Pin Name | Function |
|---|---|
| LVDSRX*<number>*p | Receiver positive data pin |
| LVDSRX*<number>*n | Receiver negative data pin |
| LVDSTX*<number>*p | Transmitter positive data pin |
| LVDSTX*<number>*n | Transmitter negative data pin |
| LVDSRXINCLK1p | Receiver input clock positive pin |
| LVDSRXINCLK1n | Receiver input clock negative pin |
| LVDSTXINCLK1n | Transmitter input clock negative pin |
| LVDSTXINCLK1n | Transmitter input clock positive pin |
| LVDSTXOUTCLK1p | Transmitter output clock positive pin |
| LVDSTXOUTCLK1n | Transmitter output clock negative pin |
| CLK1p | Dedicated clock 1 positive pin (PLL 1) |
| CLK1n | Dedicated clock 1 negative pin (PLL 1) |
| CLK2p | Dedicated clock 2 positive pin (PLL 2) |
| CLK2n | Dedicated clock 2 negative pin (PLL 2) |
| CLK3p | Dedicated clock 3 positive pin (PLL 3) |
| CLK3n | Dedicated clock 3 negative pin (PLL 3) |
| CLK4p | Dedicated clock 4 positive pin (PLL 4) |
| CLK4n | Dedicated clock 4 negative pin (PLL 4) |
| CLKLK_FB1p | Dual-purpose ClockLock feedback positive pin (PLL 1) |
| CLKLK_FB1n | Dual-purpose ClockLock feedback negative pin (PLL 1) |
| CLKLK_FB2p | Dual-purpose ClockLock feedback positive pin (PLL 2) |
| CLKLK_FB2n | Dual-purpose ClockLock feedback negative pin (PLL 2) |
| CLKLK_OUT1p | Dual-purpose ClockLock output positive pin (PLL 1) |
| CLKLK_OUT1n | Dual-purpose ClockLock output negative pin (PLL 1) |
| CLKLK_OUT2p | Dual-purpose ClockLock output positive pin (PLL 2) |
| CLKLK_OUT2n | Dual-purpose ClockLock output negative pin (PLL 2) |

## The `altlvds` Megafunction

Figures 6 and (7) show the symbols for the `altlvds` Megafunction transmitter and receiver, respectively. Each module represents the dedicated LVDS silicon present in APEX 20KE devices as well as the dedicated LVDS PLLs that are present for clock generation. A single module represents either one or multiple LVDS channels.

*Figure 6. The* `altlvds` *Megafunction Transmitter Module Symbol*

| Parameter | Value |
|---|---|
| NUMBER_OF_CHANNELS | 16 |
| DESERIALIZATION_FACTOR | 4 |
| REGISTERED_INPUT | "ON" |
| MULTI_CLOCK | "OFF" |
| INCLOCK_PERIOD | |
| CLOCK_SETTING | "UNUSED" |

```
altlvds_tx

tx_in[]              tx_out[]
tx_inclock           tx_outclock
sync_inclock         tx_locked

inst1
```

*Figure 7. The* `altlvds` *Megafunction Receiver Module Symbol*

| Parameter | Value |
|---|---|
| NUMBER_OF_CHANNELS | 16 |
| DESERIALIZATION_FACTOR | 4 |
| REGISTERED_OUTPUT | "ON" |
| INCLOCK_PERIOD | |

```
altlvds_rx

rx_in[]              rx_out[]
rx_inclock           rx_outclock
rx_deskew            rx_locked

inst
```

The following sample scripts show the AHDL Function Prototype (port name and order also apply to Verilog HDL) and VHDL Component Declaration for both the LVDS transmitter and receiver.

AHDL Function Prototype (transmitter):

```
FUNCTION altlvds_tx (tx_in[DESERIALIZATION_FACTOR*NUMBER_OF_CHANNELS-1..0],
tx_inclock, sync_inclock)
    WITH (NUMBER_OF_CHANNELS, DESERIALIZATION_FACTOR, REGISTERED_INPUT,
MULTI_CLOCK, INCLOCK_PERIOD)
    RETURNS (tx_out[NUMBER_OF_CHANNELS-1..0], tx_outclock, tx_locked);
```

VHDL Component Declaration (transmitter):

```
COMPONENT altlvds_tx
    GENERIC (NUMBER_OF_CHANNELS: NATURAL;
        DESERIALIZATION_FACTOR: NATURAL;
        REGISTERED_INPUT: STRING := "ON";
        MULTI_CLOCK: STRING := "OFF";
        INCLOCK_PERIOD: NATURAL);
        CLOCK_SETTING: STRING := "UNUSED");
    PORT (tx_in: IN
STD_LOGIC_VECTOR(DESERIALIZATION_FACTOR*NUMBER_OF_CHANNELS-1 DOWNTO 0);
        tx_inclock: IN STD_LOGIC;
        sync_inclock: IN STD_LOGIC := '0';
        tx_out: OUT STD_LOGIC_VECTOR(NUMBER_OF_CHANNELS-1 DOWNTO 0);
        tx_outclock, tx_locked: OUT STD_LOGIC);
END COMPONENT;
```

AHDL Function Prototype (receiver):

```
FUNCTION altlvds_rx (rx_in[NUMBER_OF_CHANNELS-1..0], rx_inclock, rx_deskew)
   WITH (NUMBER_OF_CHANNELS, DESERIALIZATION_FACTOR, REGISTERED_OUTPUT,
INCLOCK_PERIOD)
   RETURNS (rx_out[DESERIALIZATION_FACTOR*NUMBER_OF_CHANNELS-1..0],
rx_outclock, rx_locked);
```

VHDL Component Declaration (receiver):

```
COMPONENT altlvds_rx
   GENERIC(NUMBER_OF_CHANNELS: NATURAL;
      DESERIALIZATION_FACTOR: NATURAL;
      REGISTERED_OUTPUT: STRING := "ON";
      INCLOCK_PERIOD: NATURAL;
      CLOCK_SETTING: STRING := "UNUSED");
   PORT (rx_in: IN STD_LOGIC_VECTOR(NUMBER_OF_CHANNELS-1 DOWNTO 0);
      rx_inclock: IN STD_LOGIC;
      rx_deskew: IN STD_LOGIC := '0';
      rx_out: OUT STD_LOGIC_VECTOR(DESERIALIZATION_FACTOR*NUMBER_OF_CHANNELS-1
DOWNTO 0);
      rx_outclock, rx_locked: OUT STD_LOGIC);
END COMPONENT;
```

The `altlvds` megafunction input and output ports are described in Tables 5 and (6), respectively. Table 7 lists the parameters that are used to configure the `altlvds` megafunction.

*Table 5. Input Ports of the `altlvds` Megafunction*

| Port Name | Required | Description | Notes |
|---|---|---|---|
| **LVDS TRANSMITTER INPUT PORTS** | | | |
| tx_in[] | Yes | Input data | Input port [DESERIALIZATION_FACTOR * NUMBER_OF_CHANNELS-1..0] wide. |
| tx_inclock | Yes | LVDS reference input clock | |
| sync_inclock | No | Optional clock for the input registers | If the MULTI_CLOCK parameter is turned on, you must use this port. |
| **LVDS RECEIVER INPUT PORTS** | | | |
| rx_in[] | Yes | LVDS input data channel | Input port [NUMBER_OF_CHANNELS-1..0] wide. |
| rx_inclock | Yes | LVDS reference input clock | |
| rx_deskew | No | Specifies whether to activate calibration mode | For more information on the rx_deskew port, contact Altera Applications. |

*Table 6. Output Ports of the `altlvds` Megafunction*

| Port Name | Required | Description | Notes |
|---|---|---|---|
| **LVDS TRANSMITTER OUTPUT PORTS** |
| `tx_out[]` | Yes | Serialized LCDS data signal | Output port [NUMBER_OF_CHANNELS-1..0] wide. |
| `tx_outclock` | No | External reference clock | |
| `tx_locked` | No | Gives the status of the LVDS PLL | When the PLL is locked, this signal is VCC. When the PLL fails to lock, this signal is GND. |
| **LVDS RECEIVER OUTPUT PORTS** |
| `rx_out[]` | Yes | Deserialized data signal | Output port [DESERIALIZATION_FACTOR * NUMBER_OF_CHANNELS-1..0] wide. |
| `rx_outclock` | No | Internal reference clock | |
| `rx_locked` | No | Gives the status of the LVDS PLL | When the PLL is locked, this signal is VCC. When the PLL fails to lock, this signal is GND. |

*Table 7. The `altlvds` Megafunction Parameters*

| Parameter | Type | Required | Description |
|---|---|---|---|
| **LVDS TRANSMITTER PARAMETERS** |
| NUMBER_OF_CHANNELS | Integer | Yes | Specifies the number of LVDS channels. |
| DESERIALIZATION_FACTOR | Integer | Yes | Specifies the number of bits per channel. Values are 8, 7, or 4. When you specify 4 bits per channel, the value of the INCLOCK_PERIOD parameter can be 50-80 MHz. When you specify 7 bits per channel, the value of the INCLOCK_PERIOD parameter can be 30-80 MHz. For 8 bits per channel, the INCLOCK_PERIOD can be 30-78 MHz. *(1)* |
| REGISTERED_INPUT | String | No | Indicates whether the `tx_out[]` and `tx_outclock` ports should be registered. Values are "ON" and "OFF". If omitted the default is "ON". |
| MULTI_CLOCK | String | No | Indicates whether the `sync_inclock` port is used for input registering. Values are "ON" and "OFF". If omitted the default is "OFF." |
| INCLOCK_PERIOD | String | Yes | Specifies the period or frequency of the input clock. The default time unit is ps. |
| **LVDS RECEIVER PARAMETERS** |
| NUMBER_OF_CHANNELS | Integer | Yes | Specifies the number of LVDS channels. |
| DESERIALIZATION_FACTOR | Integer | Yes | Specifies the number of bits per channel. Values are 8, 7, or 4. When you specify 4 bits per channel, the value of the INCLOCK_PERIOD parameter can be 50-80 MHz. When you specify 7 bits per channel, the value of the INCLOCK_PERIOD parameter can be 30-80 MHz. For 8 bits per channel, the INCLOCK_PERIOD can be 30-78 MHz. *(1)* |
| REGISTERED_OUTPUT | String | No | Indicates whether the `rx_out[]` port should be registered. Values are "ON" and "OFF". If omitted, the default is "ON". |
| INCLOCK_PERIOD | String | Yes | Specifies the period or frequency of the `rx_inclock` port. The default time unit is ps. |

*Note:*

(1)   For 840 Mbps transfer rates in the Quartus software version 2000.05 and lower, the deserialization factor must be set to 8✕ mode and the input frequency to 78 MHz. Currently, the `altlvds` megafunction and the Quartus Simulator do not accept input frequencies greater than 78 MHz in 8✕ mode; however, APEX 20KE devices are capable of supporting input frequencies of up to 105 MHz. Future versions of the Quartus software will reflect the higher transfer rate capabilities of APEX 20KE devices.

## The MegaWizard Interface

The MegaWizard® interface allows users to customize the LVDS megafunction. The MegaWizard Plug-In Manager automatically generates the following files:

■ Component Declaration File (**.cmp**) that can be used in VHDL Design Files (**.vhd**)
■ Include File (**.inc**) that can be used in Text Design Files (**.tdf**) and Verilog Design Files (**.v**)
■ Quartus Block Symbol File (**.bsf**) that can be used in Quartus Block Design Files (**.bdf**)
■ Custom Megafunction variation file (TDF, VHD, or V file)

The MegaWizard Plug-In Manager can be invoked in two ways:

■ Choosing the MegaWizard Plug-In Manager command from the Tools menu, as seen in Figure 8.
■ Selecting MegaWizard Plug-In Manager from the symbol dialog box in the Block Editor, as seen in Figure 9.

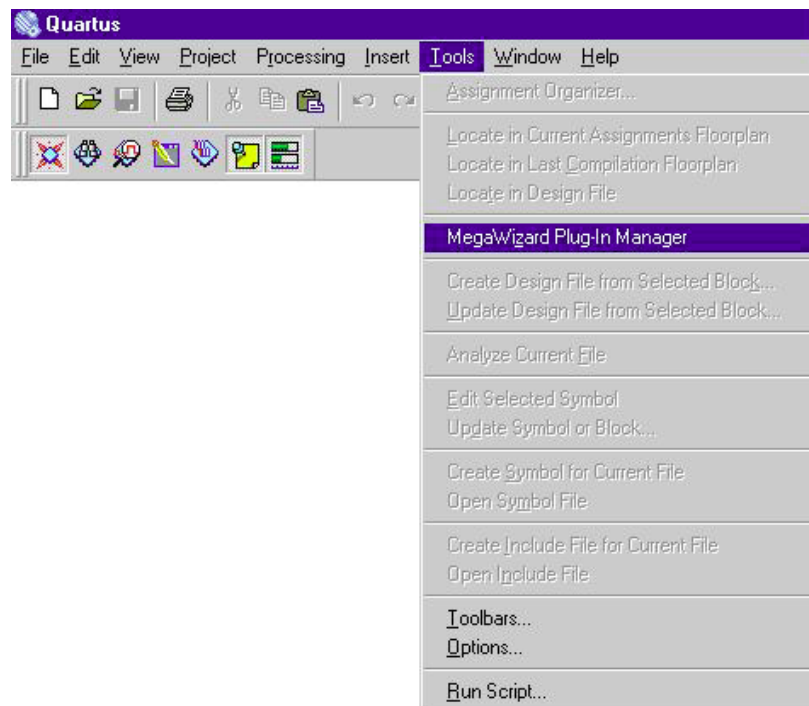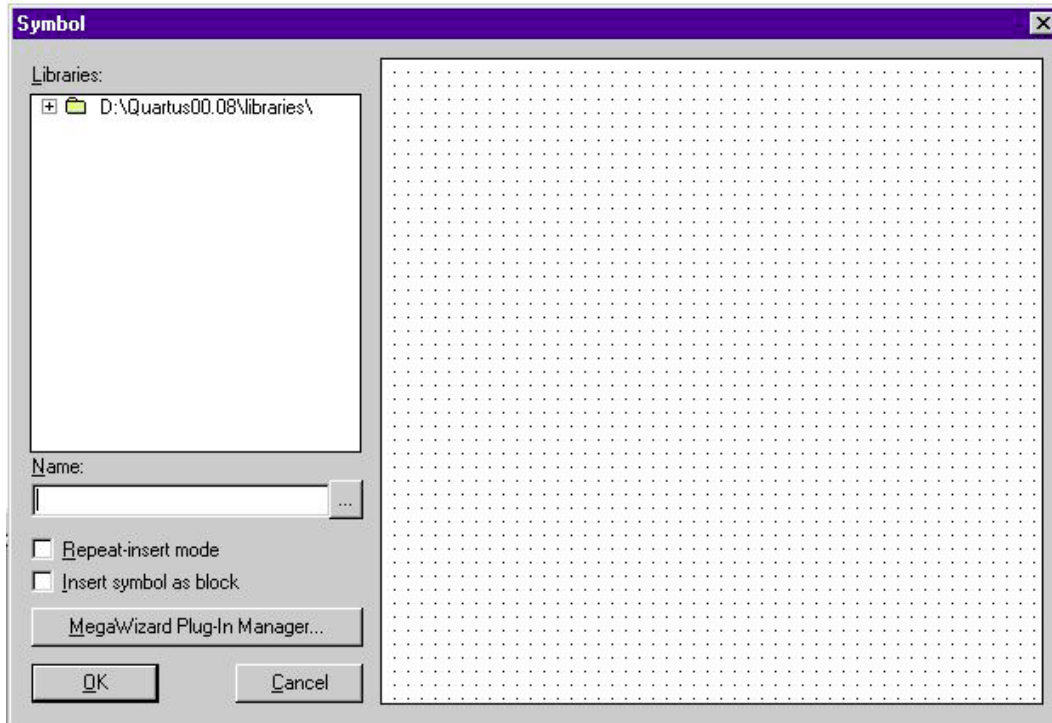*Figure 8. Invoking the MegaWizard Plug-In Manager from the Tools Menu*

*Figure 9. Invoking the MegaWizard Plug-In Manager from the Block Editor*



The MegaWizard Plug-In Manager takes a step-by-step approach to generating customized LVDS transmitter and receiver modules. Each page of the MegaWizard Plug-In Manager allows the user to select from a set of customizable features that tailors the modules to the needs of the design.

Figure 10 displays the third page of the altlvds Megafunction in the MegaWizard Plug-In Manager when instantiating an LVDS transmitter. Figure 11 shows the third page for a receiver instantiation. These pages allow the user to customize the LVDS transmitter and receiver modules.

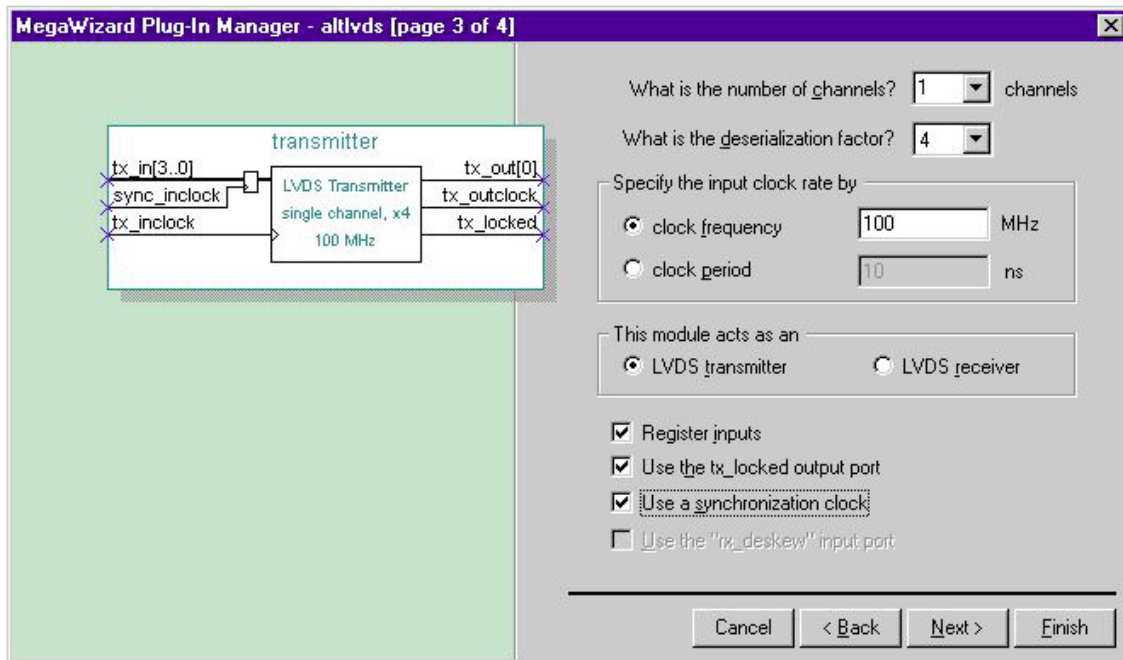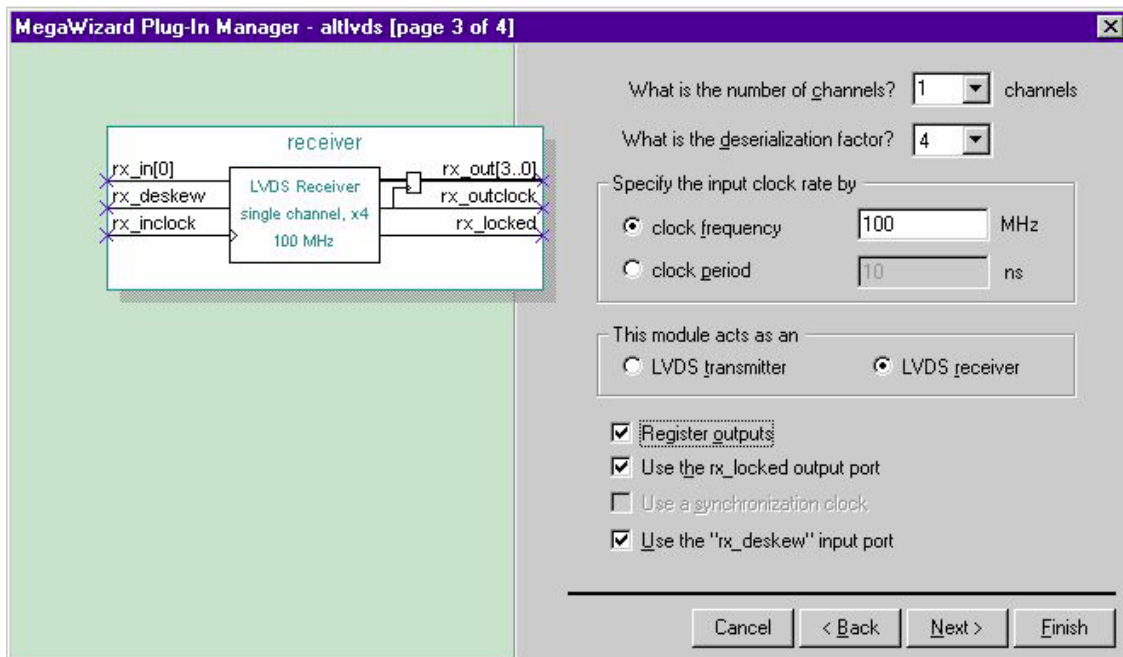*Figure 10. Page 3 of the* `altlvds` *Transmitter MegaWizard Plug-In Manager*



*Figure 11. Page 3 of the* `altlvds` *Receiver MegaWizard Plug-In Manager*

Described below are the various customizable features that are available in the MegaWizard interface:

■   Number of channels – this option allows the user to select the number of LVDS channels to be used in the design. The desired value can be either typed or selected from the pop-up menu, up to a maximum of 16 channels. This simplifies the complexity of the design in that only one transmitter or receiver module needs to be instantiated to represent multiple LVDS channels.

■   Deserialization factor – this option specifies the number of bits per channel. The user can either type or select 4, 7 or 8 from the pop-up menu.

■   Clock frequency / period – this option specifies the clock frequency or period of the LVDS input clock. If the deserialization factor is 4 bits per channel, the clock frequency can be 50-80 MHz. Otherwise, the clock frequency can be 30-80 for 7 bits per channel and 30-78 MHz for 8 bits per channel. (See Note 1 on page 9.)

■   LVDS transmitter / receiver – this option specifies the function of the LVDS module.

■   Register inputs / outputs – specifies whether or not to register the inputs for the transmitter and outputs for the receiver.

■   Use the `tx_locked` / `rx_locked` port – this option enables the use of the locked pin for the transmitter and receiver. When the phase-locked loop (PLL) locks onto the incoming clock and generates an internal clock, the locked signal is driven high. It remains high as long as the input clock remains within specification.

■   Use a synchronization clock – this option activates the synchronization clock for the transmitter. If this option is activated, the synchronization clock must have the same frequency and phase as the transmitter clock in order to avoid hold time violations.

■   Use the `rx_deskew` input port – this option activates the deskew input port for the receiver which is used to calibrate the module.

## MegaWizard Examples

Figure 12 shows an `altlvds` transmitter module generated by the MegaWizard Plug-In Manager with an input frequency of 50 MHz. 16 channels are used with a deserialization factor of 4. In this example, the `sync_inclock` input and the `tx_locked` output are both used as well as the input registers.

*Figure 12. 50 MHz 16-Channel 4× LVDS Transmitter*
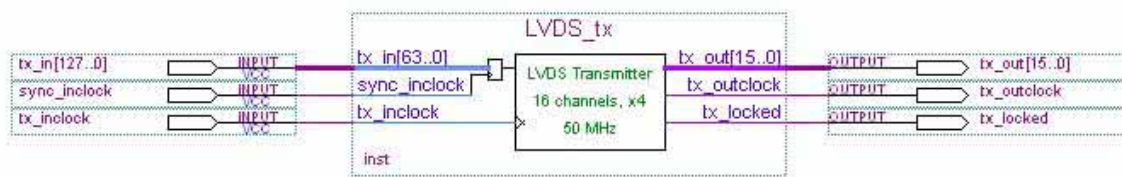


Figure 13 shows the instantiation of an LVDS receiver with 16 channels and an input frequency of 50 MHz. The deserialization factor is set to 4. The `rx_deskew`, `rx_outclock` and output registers are all used in this example.

*Figure 13. 50 MHz 16-Channel 4× LVDS Receiver*
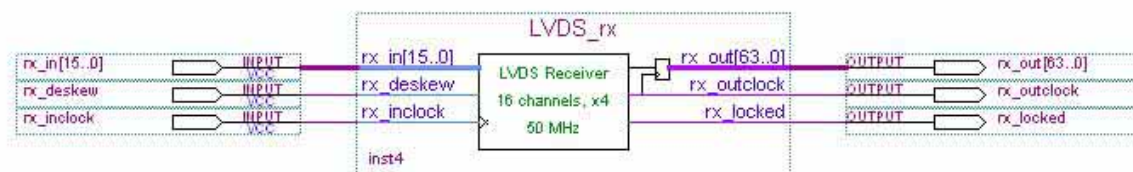
## HDL Examples

The following examples show the `altlvds` megafunction in transmitter mode and receiver mode, respectively, in both VHDL and Verilog. These examples instantiate the LVDS modules and connect them to input and output pins.

### VHDL HDL

The LVDS transmitter is a 16-channel module operating in 8× mode with an input clock of 70 MHz. The transmitter's output clock is fed out of the module in 1× mode through the `tx_outclock` pin. The status of the PLL can be monitored from the `tx_locked` pin. The input registers are also used.

### lvds_tx.vhd

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

entity lvds_tx is
   port
       ( tx_in:          in       std_logic_vector(127 downto 0);
         tx_inclock:     in       std_logic;
         sync_inclock:   in       std_logic;
         tx_out:         out      std_logic_vector(15 downto 0);
         tx_outclock:    out      std_logic;
         tx_locked:      out      std_logic
       );
end lvds_tx;

architecture apex of lvds_tx is
   component altlvds_tx
   generic
         (number_of_channels:     positive;
          deserialization_factor: positive;
          registered_input:       string      := "ON";
          multi_clock:            string      := "OFF";
          inclock_period:         positive;
          clock_setting:          string      := "UNUSED"
       );
   port
       ( tx_in:          in       std_logic_vector
         (deserialization_factor*number_of_channels-1 downto 0);
         tx_inclock:     in       std_logic;
         sync_inclock:   in       std_logic := '0';
         tx_out:         out      std_logic_vector
         (number_of_channels-1 downto 0);
         tx_outclock:    out      std_logic;
         tx_locked:      out      std_logic
       );
end component;
begin
U0:altlvds_tx
   generic map (  number_of_channels => 16,
                  deserialization_factor => 8,
                  inclock_period => 14285
```

```
                )
   port map      (  tx_in => tx_in,
                    tx_inclock => tx_inclock,
                    sync_inclock => sync_inclock,
                    tx_out => tx_out,
                    tx_outclock => tx_outclock,
                    tx_locked => tx_locked
                );
end apex;
```

## lvds_rx.vhd

The LVDS receiver is a 16-channel module operating in 8× mode with an input clock of 70 MHz. The receiver's output clock cannot be fed out directly to an output pin; therefore, it feeds the clock port of a dff register.

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

entity lvds_rx is
   port
      (  rx_in:instd_logic_vector(15 downto 0);
         rx_inclock:instd_logic;
         rx_deskew:instd_logic;
         rx_out:outstd_logic_vector(127 downto 0);
         rx_outclock:outstd_logic;
         rx_locked:outstd_logic;
         inpin:instd_logic;
         outpin:outstd_logic;
         clear:instd_logic;
         preset:instd_logic
      );
end lvds_rx;

architecture apex of lvds_rx is
   component altlvds_rx
      generic
            (     number_of_channels:positive;
                  deserialization_factor :positive;
                  registered_output:string = "ON";
                  inclock_period:positive;
                  clock_setting:string := "UNUSED"
            );
      port
            (     rx_in:instd_logic_vector(number_of_channels-1 downto 0);
                  rx_inclock:instd_logic;
                  rx_deskew:instd_logic := '0';
                  rx_out:out std_logic_vector(deserialization_factor *
                  number_of_channels-1 downto 0);
                  rx_outclock:outstd_logic;
                  rx_locked:outstd_logic
            );
   end component;
```

```
   component dff
   port
      (
      d   : in std_logic;
      clk : in std_logic;
      clrn: in std_logic;
      prn : in std_logic;
      q   : out std_logic
      );
   end component;

signal clock: std_logic;
begin
U0: altlvds_rx
   generic map
      (      number_of_channels => 16,
             deserialization_factor => 8,
             inclock_period => 14285
      )
   port map
      (      rx_in => rx_in,
             rx_inclock => rx_inclock,
             rx_deskew => rx_deskew,
             rx_out => rx_out,
             rx_outclock => clock,
             rx_locked => rx_locked
      );

U1: dff
   port map (d => inpin,
        clk => clock,
        clrn => clear,
        prn => preset,
        q => outpin);
end apex;
```

## Verilog HDL

The following examples show the `altlvds` megafunction in Verilog HDL for both the transmitter and receiver, respectively. The LVDS transmitter is a 16-channel module, operating with an input clock of 70 MHz and a deserialization factor of 8×. The output clock of the PLL is fed directly to the `tx_outclock` output pin.

### lvds_tx.v

```
module lvds_tx (tx_in, tx_inclock, tx_out, tx_outclock, tx_locked);
     input[127:0] tx_in;
     input tx_inclock;
     output[15:0] tx_out;
     output tx_outclock;
     output tx_locked;
altlvds_tx U0 (.tx_in (tx_in), .tx_inclock (tx_inclock), .tx_out (tx_out),
.tx_outclock (tx_outclock), .tx_locked (tx_locked));
```

```
defparam
      U0.number_of_channels      =    16,
      U0.deserialization_factor =    8,
      U0.registered_input        =    "On",
      U0.multi_clock             =    "Off",
      U0.inclock_period          =    14285;
endmodule
```

### lvds_rx.v

The receiver is also a 16-channel module with an input clock frequency of 70 MHz and a deserialization factor of 8×. The output clock of the PLL is fed to the clock port of register D1 since it cannot be fed directly to an output pin.

```
module lvds_rx ( rx_in, rx_inclock, rx_deskew, rx_out, rx_locked, inpin,
outpin);
      input[15:0] rx_in;
      input rx_inclock;
      input rx_deskew;
      output[127:0] rx_out;
      output rx_locked;
      input inpin;
      output outpin;
reg pll_out;
altlvds_rx U0 (.rx_in (rx_in), .rx_inclock (rx_inclock), .rx_deskew
(rx_deskew), .rx_out (rx_out), .rx_outclock (pll_out), .rx_locked
(rx_locked));
defparam
      U0.number_of_channels      =    16,
      U0.deserialization_factor =    8,
      U0.registered_output       =    "ON",
      U0.inclock_period          =    14285;
dff D1 (.d(inpin), .q(outpin), .clk(pll_out));
endmodule
```

## Synthesis with Third Party Tools

To synthesize the design successfully in third party tools such as Synplify, FPGA Compiler/II, FPGA Express, and LeonardoSpectrum, the LVDS design component must be treated as a black box. By declaring the module a black box, synthesis tools will refrain from synthesizing the module. However, the correct port connections will be made in the output EDIF netlist file (**.edf**) or verilog Quartus mapping file (**.vqm**). When the netlist file is brought into the Quartus software, native synthesis on the black-boxed module is automatically performed.

The LVDS module must first be generated by the MegaWizard Plug-In Manager which involves specifying the name of the module and the ports that are used. Below are examples of an LVDS transmitter design in VHDL and Verilog for several 3rd party tools. The file named mylvds_tx is the MegaWizard-generated file.

### VHDL: lvds_tx.vhd

Synplicity Synplify and Synopsys FPGA Compiler II/FPGA Express/FPGA Compiler II Altera Edition

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

```
entity lvds_tx is
   port
       (  tx_in:           in    std_logic_vector(127 downto 0);
          tx_inclock:    in    std_logic;
          sync_inclock:  in    std_logic;
          tx_out:        out   std_logic_vector(15 downto 0);
          tx_outclock:   out   std_logic;
          tx_locked:     out   std_logic
       );
end lvds_tx;

architecture apex of lvds_tx is
component mylvds_tx
       port
           (      tx_in:         in    std_logic_vector(127 downto 0);
                  tx_inclock:    in    std_logic;
                  sync_inclock:  in    std_logic;
                  tx_out:        out   std_logic_vector(15 downto 0);
                  tx_outclock:   out   std_logic;
                  tx_locked:     out   std_logic
           );
end component;

attribute black_box: boolean;
attribute black_box of mylvds_tx: component is true;

begin

U0:mylvds_tx
       port map (tx_in => tx_in, tx_inclock => tx_inclock, sync_inclock =>
       sync_inclock, tx_out => tx_out, tx_outclock => tx_outclock, tx_locked =>
       tx_locked);
end apex;
```

**Exemplar LeonardoSpectrum: lvds_tx.vhd**

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

entity lvds_tx is
       port
           (      tx_in:         in    std_logic_vector(127 downto 0);
                  tx_inclock:    in    std_logic;
                  sync_inclock:  in    std_logic;
                  tx_out:        out   std_logic_vector(15 downto 0);
                  tx_outclock:   out   std_logic;
                  tx_locked:     out   std_logic
           );
end lvds_tx;

architecture apex of lvds_tx is
       component mylvds_tx
```

```
        port
            (       tx_in:          in      std_logic_vector(127 downto 0);
                    tx_inclock:     in      std_logic;
                    sync_inclock:   in      std_logic;
                    tx_out:         out     std_logic_vector(15 downto 0);
                    tx_outclock:    out     std_logic;
                    tx_locked:      out     std_logic
            );
end component;

attribute noopt: boolean;
attribute noopt of mylvds_tx: component is true;

begin
U0:mylvds_tx
        port map (tx_in => tx_in, tx_inclock => tx_inclock, sync_inclock =>
sync_inclock, tx_out => tx_out, tx_outclock => tx_outclock, tx_locked =>
tx_locked);
end apex;
```

## Verilog: lvds_tx.v

The code below demonstrates black-boxing in Verilog using the same transmitter module generated by the MegaWizard Plug-In Manager.

Synplicity Synplify and Synopsys FPGA Compiler II/FPGA Express/FPGA Compiler II Altera Edition

```
module mylvds_tx (tx_in, tx_inclock, tx_out, tx_outclock, tx_locked);
/*synthesis black_box*/
        input[127:0] tx_in;
        input tx_inclock;
        output[15:0] tx_out;
        output tx_outclock;
        output tx_locked;
endmodule

module lvds_tx (tx_in, tx_inclock, tx_out, tx_outclock, tx_locked);
        input[127:0] tx_in;
        input tx_inclock;
        output[15:0] tx_out;
        output tx_outclock;
        output tx_locked;

mylvds_tx U0 (.tx_in (tx_in), .tx_inclock (tx_inclock), .tx_out (tx_out),
.tx_outclock (tx_outclock), .tx_locked (tx_locked));

endmodule
```

## Exemplar LeonardoSpectrum: lvds_tx.v

```
module mylvds_tx (tx_in, tx_inclock, tx_out, tx_outclock, tx_locked);
      input[127:0] tx_in;
      input tx_inclock;
      output[15:0] tx_out;
      output tx_outclock;
      output tx_locked;
endmodule
// higher level file
module lvds_tx (tx_in, tx_inclock, tx_out, tx_outclock, tx_locked);
      input[127:0] tx_in;
      input tx_inclock;
      output[15:0] tx_out;
      output tx_outclock;
      output tx_locked;

mylvds_tx U0 (.tx_in (tx_in), .tx_inclock (tx_inclock), .tx_out (tx_out),
.tx_outclock (tx_outclock), .tx_locked (tx_locked));
//exemplar attribute U0 NOOPT TRUE
endmodule
```

The Quartus software must be configured so that it recognizes the EDF or VQM netlist file from the third party synthesis tool. The synthesis tool can be selected from the **EDA Tool Settings** dialog box (Project menu) in the Quartus software, as seen in Figure 14. More information regarding the Quartus software's integration with third party tools can be found on the Altera web site at **http://www.altera.com/html/nativelink/nativelink.html**.

*Figure 14. EDA Tool Settings in the Quartus Software*
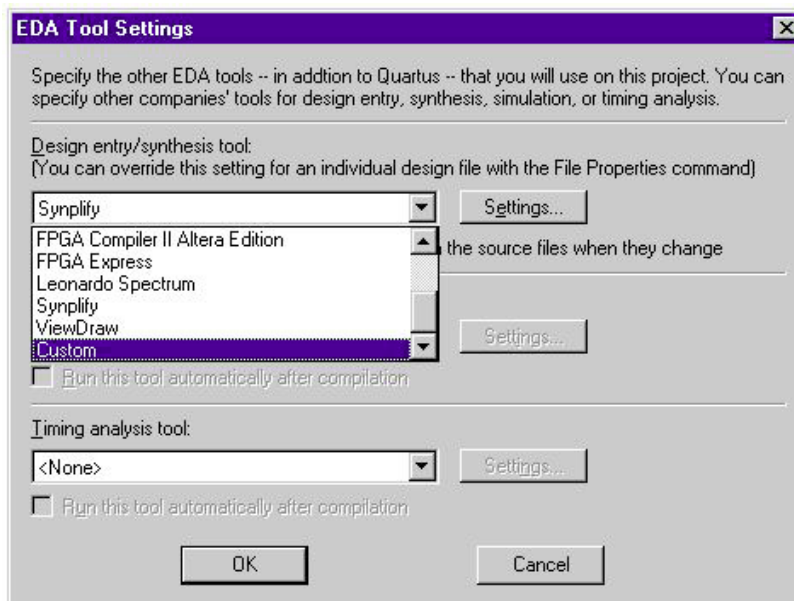
## Testbenches

The following testbench examples, which can be used in third party simulators such as ModelSim, show the functionality of the LVDS behavioral model. The receiver's output is connected directly to the transmitter's input. The deskew pin is asserted, and the calibration pattern is applied first. If this is not done correctly, the model will not allow data to exit the transmitter. The test pattern inputs to the receiver's input port and subsequently leaves the transmitter's output port after two clock cycles of latency. When running this test bench, ensure that the time resolution of the simulator is set to picoseconds. More information on receiver calibration can be found in the following VHDL HDL and Verilog HDL testbench examples.

### VHDL HDL: lvds_test.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;
USE work.apex20ke_mf_components.ALL;
USE std.textio.ALL;

ENTITY lvds_test IS
END lvds_test;

ARCHITECTURE testbench OF lvds_test IS

        SIGNAL rx_in: std_logic_vector(3 downto 0) := "0000";
        SIGNAL rx_inclock: std_logic := '1';
        SIGNAL rx_deskew: std_logic := '0';
        SIGNAL synch_inclock: std_logic := '0';
        SIGNAL tx_out: std_logic_vector(3 downto 0);
        SIGNAL tx_outclock: std_logic;
        SIGNAL rx_out: std_logic_vector(31 downto 0);
        SIGNAL rx_outclock: std_logic;
        SIGNAL lvds_data_clk: std_logic := '1';

        TYPE rx_buffer IS ARRAY(0 to 17, 0 to 3) OF std_logic;

BEGIN

-- Instantiate the LDVS Receiver
L0: altlvds_rx
        GENERIC MAP (number_of_channels => 4,
                    deserialization_factor => 8,
                    inclock_period => 128000,
                    registered_output => "ON")
        PORT MAP (rx_in => rx_in,
                    rx_inclock => rx_inclock,
                    rx_deskew => rx_deskew,
                    rx_out => rx_out,
                    rx_outclock => rx_outclock);

-- Instantiate the LVDX Transmitter
L1: altlvds_tx
```

```
        GENERIC MAP (number_of_channels => 4,
                     deserialization_factor => 8,
                     inclock_period => 128000,
                     registered_input => "ON")
        PORT MAP (tx_in => rx_out,
                     tx_inclock => rx_inclock,
                     sync_inclock => synch_inclock,
                     tx_out => tx_out,
                     tx_outclock => tx_outclock);

-- Create a 7,812,500 Hz clock
PROCESS(rx_inclock)
BEGIN
        rx_inclock <= NOT rx_inclock AFTER 64 ns;
END PROCESS;


-- Create a 62,500,000 lvds data clock to synch data inputs
PROCESS(lvds_data_clk)
BEGIN
        lvds_data_clk <= NOT lvds_data_clk AFTER 8 ns;
END PROCESS;


PROCESS
        VARIABLE deskew_pattern : std_logic_vector(23 downto 0) :=
"000011110000111100001111";
        VARIABLE test_pattern : std_logic_vector(15 downto 0) :=
"0000011000000001";
        VARIABLE cnt : integer range 0 to 15 := 0;
        VARIABLE deskew_cnt: integer range 0 to 48 := 0;
BEGIN
        IF deskew_cnt < 48 THEN
                IF deskew_cnt > 23 THEN
                        rx_deskew <= '1';
                        rx_in(0) <= deskew_pattern(deskew_cnt -24);
                        rx_in(1) <= deskew_pattern(deskew_cnt -24);
                        rx_in(2) <= deskew_pattern(deskew_cnt -24);
                        rx_in(3) <= deskew_pattern(deskew_cnt -24);
                END IF;
                deskew_cnt := deskew_cnt + 1;
                wait until ((lvds_data_clk'event) and (lvds_data_clk = '1'));
        ELSIF cnt < 15 THEN
                rx_deskew <= '0';
                rx_in(0) <= test_pattern(cnt);
                rx_in(1) <= test_pattern(cnt);
                rx_in(2) <= test_pattern(cnt);
                rx_in(3) <= test_pattern(cnt);
                cnt := cnt + 1;
                wait until ((lvds_data_clk'event) and (lvds_data_clk = '1'));
        ELSE cnt := 0;
        END IF;
END PROCESS;
```

```
END testbench;
```

## Verilog HDL: lvds_test.v

```verilog
`timescale 1ps/1ps

module lvds_test();
reg [3:0] rx_in;
reg rx_inclock;
reg rx_deskew;
reg synch_inclock;
wire [3:0] tx_out;
wire tx_outclock;
wire [31:0] rx_out;
wire rx_outclock;
reg lvds_data_clk;

reg [23:0] deskew_pattern;
reg [15:0] test_pattern;

reg [3:0] cnt;
reg [5:0] deskew_cnt;

altlvds_rx L0(.rx_in(rx_in), .rx_inclock(rx_inclock), .rx_deskew(rx_deskew),
.rx_out(rx_out), .rx_outclock(rx_outclock));
defparam L0.number_of_channels = 4;
defparam L0.deserialization_factor = 8;
defparam L0.registered_output = "ON";
defparam L0.inclock_period = 128000;

altlvds_tx L1(.tx_in(rx_out), .tx_inclock(rx_inclock),
.sync_inclock(synch_inclock), .tx_out(tx_out), .tx_outclock(tx_outclock));
defparam L1.number_of_channels = 4;
defparam L1.deserialization_factor = 8;
defparam L1.registered_input = "ON";
defparam L1.inclock_period = 128000;

initial
begin
      rx_in = 4'b0000;
      rx_deskew = 1'b0;
      synch_inclock = 1'b0;
      deskew_cnt = 6'b000001;
      cnt = 4'b0000;
      deskew_pattern = 24'b000011110000111100001111;
      test_pattern = 16'b1000011110000001;
end

initial
begin
      rx_inclock = 1'b1;
      forever #64000 rx_inclock = ~rx_inclock;
```

```
end

initial
begin
      lvds_data_clk = 1'b1;
      forever #8000 lvds_data_clk = ~lvds_data_clk;
end

always@(posedge lvds_data_clk)
begin
      if (deskew_cnt < 48)
            begin
            if (deskew_cnt > 23)
                  begin
                  rx_deskew = 1'b1;
                  rx_in[0] = deskew_pattern[deskew_cnt-24];
                  rx_in[1] = deskew_pattern[deskew_cnt-24];
                  rx_in[2] = deskew_pattern[deskew_cnt-24];
                  rx_in[3] = deskew_pattern[deskew_cnt-24];
                  end
            deskew_cnt = deskew_cnt + 1;
            end
      else if (cnt <= 15)
            begin
            #1 rx_deskew = 1'b0;
            rx_in[0] = test_pattern[cnt];
            rx_in[1] = test_pattern[cnt];
            rx_in[2] = test_pattern[cnt];
            rx_in[3] = test_pattern[cnt];
            cnt = cnt + 1;
            end
      else cnt = 0;
end
endmodule
```

## Quartus LVDS Reporting

The Quartus software reports LVDS usage in the compilation report file. The report file documents all information pertaining to LVDS resource usage and placement in the APEX device under the following categories:

■    All Package Pins
■    Control Signals
■    Global and Other Fast Signals
■    LVDS
■    ClockLock

This section briefly describes each category.

## All Package Pins

This category of the report file indicates the function and location of all package pins. LVDS pins are displayed with their names and pin numbers, as seen in the Figure 15 example.

*Figure 15. All Package Pins Section of the Report File*

| Pin # | Usage |
|---|---|
| D2 | GND |
| D3 | GND |
| D4 | GND |
| D5 | VCCIO |
| D6 | rx_out[0] |
| D7 | rx_out[6] |
| D8 | GND* |
| D9 | GND* |
| D10 | GND* |
| D11 | GND* |
| D12 | VCCINT |
| D13 | GND* |
| D14 | GND* |

The Quartus software adheres to the previously-discussed banking rules and will not place non-LVDS outputs in LVDS-enabled banks. In such configurations, the design yields a no-fit, indicating that these non-LVDS outputs are illegally placed.

For more information on using I/O standards in the Quartus software, refer to the *Using I/O Standards in the Quartus Software White Paper*.

## Control Signals

The Control Signals category reports the control signals that are present in the design. LVDS control signals, such as input clocks and PLL output clocks, are reported as seen in Figure 16. PLL output clocks are denoted as either pll_clk0 or pll_clk1. pll_clk1 can be fed directly out of the transmitter in 1× mode.

*Figure 16. Control Signals Section of the Report File*

| Name | Pin # | Fan-Out | Usage |
|---|---|---|---|
| \|rec:inst1\|altlvds_rx:altlvds_rx_component\|pll_clk0 | LVDS_PLLRX_1 | 0 | Clock |
| \|rec:inst1\|altlvds_rx:altlvds_rx_component\|pll_clk1 | LVDS_PLLRX_1 | 0 | Clock |
| \|sync_inclock | N8 | 16 | Clock |
| \|tra:inst\|altlvds_tx:altlvds_tx_component\|pll_clk0 | LVDS_PLLTX_1 | 0 | Clock |
| \|tra:inst\|altlvds_tx:altlvds_tx_component\|pll_clk1 | LVDS_PLLTX_1 | 0 | Clock |
| \|rx_inclock | AB25 | 16 | Clock |
| \|tx_inclock | D1 | 16 | Clock |

## Global and Other Fast Signals

The Global and Other Fast Signals section displays the globally routed signals in the design. When LVDS is used, only the PLL-generated clocks and the synchronization clocks are routed globally as seen in Figure 17. The number of fan-out nodes for the global signal is also displayed.

*Figure 17. Global and Other Fast Signals Section of the Report File*

| Global & Other Fast Signals | | | |
|---|---|---|---|
| **Name** | **Pin #** | **Fan-Out** | **Global** |
| \|rec:inst1\|altlvds_rx:altlvds_rx_component\|pll_clk0 | LVDS_PLLRX_1 | 1 | yes |
| \|rec:inst1\|altlvds_rx:altlvds_rx_component\|pll_clk1 | LVDS_PLLRX_1 | 10 | yes |
| \|sync_inclock | N8 | 16 | yes |
| \|tra:inst\|altlvds_tx:altlvds_tx_component\|pll_clk0 | LVDS_PLLTX_1 | 1 | yes |
| \|tra:inst\|altlvds_tx:altlvds_tx_component\|pll_clk1 | LVDS_PLLTX_1 | 2 | yes |

## LVDS

This category reports LVDS usage in the design, as seen in Figure 18. The instance name is displayed along with its function and deserialization factor. Both PLL output clocks for the LVDS modules are also shown. The LVDS category is omitted when LVDS is not used in the APEX device.

*Figure 18. LVDS Section of the Report File*

| LVDS | | | | | |
|---|---|---|---|---|---|
| **Name** | **Function** | **Clock0** | **Clock1** | **Data Width** |
| \|lvds:inst5\|altlvds_rx:altlvds_rx_component\|rx[0] | Transmitter | \|lvds:inst5\|altlvds_rx:altlvds_rx_component\|pll_clk0 | \|lvds:inst5\|altlvds_rx:altlvds_rx_component\|pll_clk1 | 4 |
| \|tx:inst14\|altlvds_tx:altlvds_tx_component\|tx_out[0] | Receiver | \|tx:inst14\|altlvds_tx:altlvds_tx_component\|pll_clk0 | \|tx:inst14\|altlvds_tx:altlvds_tx_component\|pll_clk1 | 4 |

## ClockLock

The ClockLock category of the report file, as seen in Figure 19, gives the specifications of each PLL that was used. The input frequency is indicated as well as the various resulting clock frequencies after multiplication by the deserialization factor.

*Figure 19. ClockLock Section of the Report File*

| ClockLock | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Name** | **Mode** | **Input Frequency** | **Multiply Clock0** | **Divide Clock0** | **Ouput Frequency** | **Multiply Clock1** | **Divide Clock1** | **Ouput Frequency** | **Phase Shift** | **External Feedback** |
| \|rec:inst1\|altlvds_rx:altlvds_rx_component | LVDS | 50.0 MHz | 8 | 1 | 400.0 MHz | 1 | 1 | 50.0 MHz | 0 ps | no |
| \|tra:inst\|altlvds_tx:altlvds_tx_component | LVDS | 50.0 MHz | 8 | 1 | 400.0 MHz | 1 | 1 | 50.0 MHz | 0 ps | no |

## Floorplanner

The Floorplanner gives a visual representation of the internal routing and placement of logic within the device.

Figure 20 shows the Floorplan view for an LVDS transmitter. The transmitter is divided between two or more colored blocks: the LVDS PLL is located adjacent to the transmitter output clock pins (`LVDSTXOUTCLK1p` and `LVDSTXOUTCLK1n`), and the individual parallel-to-serial converters are located adjacent to each pair of LVDS data-out pins (e.g. `LVDSTX01p` and `LVDSTX01n`).

*Figure 20. Floorplanner View of LVDS Transmitter*
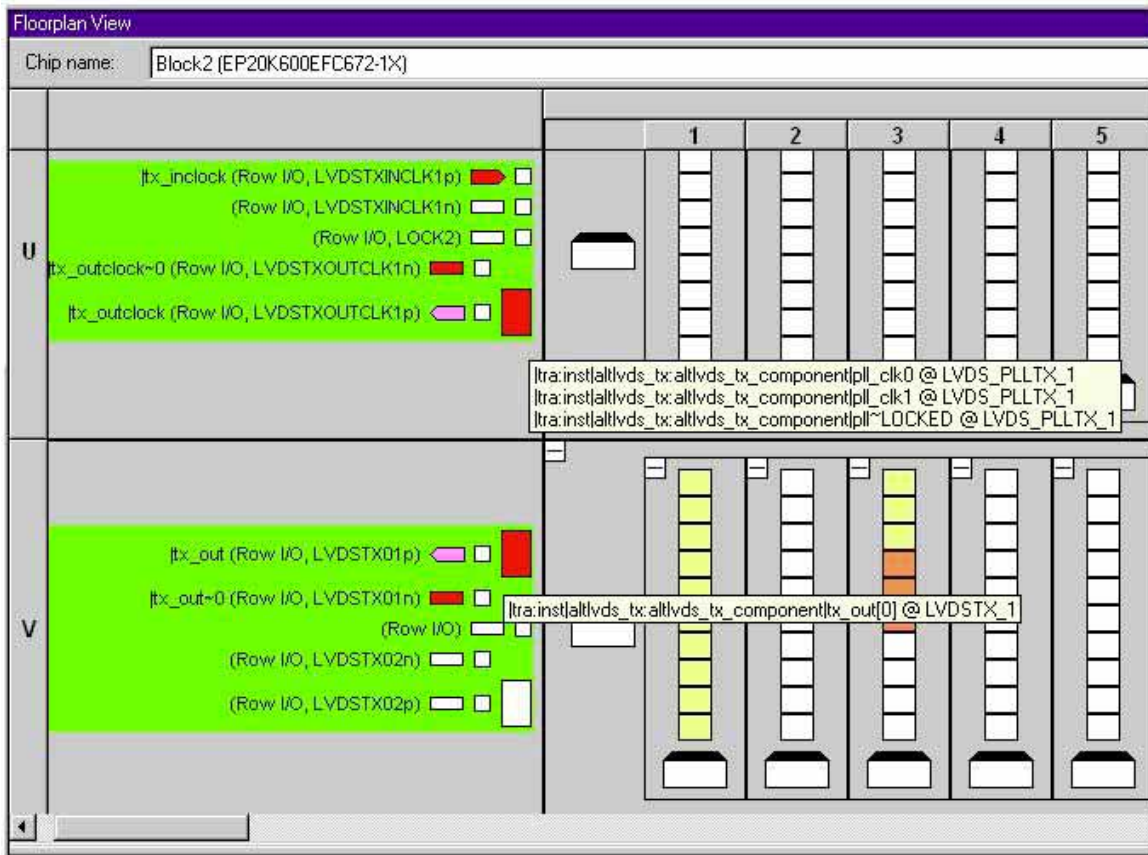


In Figure 20, the PLL appears in the equations as `LVDS_PLLTX_1`, and the single parallel-to-serial converter appears as `LVDSTX_1`. The logic cells that appear to the right of the LVDS modules represent additional logic that is consumed during implementation. The transmitter's locked PLL clock (`pll_clk1`) can be driven off-chip in 1× mode through the transmitter output clock pins.

The Quartus software displays the LVDS receiver module in a similar fashion, as seen in Figure 21. The receiver is divided between two or more colored blocks: the LVDS PLL is located adjacent to the receiver input clock pins (`LVDSRXINCLK1p` and `LVDSRXINCLK1n`), and the individual serial-to-parallel converters are located adjacent to the LVDS data-in pins (`LVDSRX01p` and `LVDSRX01n`). The PLL appears as `LVDS_PLLRX_1`, and the serial-to-parallel converter appears as `LVDSRX_1` in this example. The logic cells that appear to the left of the LVDS modules represent additional logic that is consumed during implementation. Because the PLL receiver output clock cannot be fed externally, it does not fan-out to any I/O pins in the Floorplan view.

*Figure 21.  Floorplanner View of LVDS Receiver*



## Simulation in Quartus

The Quartus development tool provides users with the capability to conveniently and efficiently simulate the LVDS design. Vector waveform files (**.vwf**), which are used as inputs to the native simulation tool, can be created within the Quartus software. The simulation model for the LVDS receiver is essentially a serialization shift-register that is driven by an LVDS data channel and clocked by an LVDS PLL multiplied by the serialization value. The shift-register drives a bank of data registers clocked by the original clock. The LVDS transmitter module is the inverse of the receiver. A data register is driven by internal parallel data signals and clocked by the original LVDS clock. It then loads a shift-register that drives the LVDS output pin and is clocked by the multiplied output of the LVDS PLL. For more information on simulation in the Quartus software, see Quartus Help.

Figure 22 shows the results of an example functional simulation of an LVDS transmitter. The 16-channel transmitter is operating at 60 MHz with the synchronization clock activated and a deserialization factor of 8.

*Figure 22. Example Functional Simulation Waveform of LVDS Transmitter*



The locked pin `tx_locked` remains high as long as the input frequency is valid. The input clock `tx_inclock` and the synchronization clock `syn_inclock` must have the exact same phase and frequency for the module to function correctly. The clocks must also have the same frequency specified in the design files. If the frequency differs, the Simulator will warn that the PLL was unable to lock onto the incoming clocks.

The incoming data `tx_in` is synchronized with the input clocks `tx_inclock` and `syn_inclock`. The output data `tx_out` is synchronized with the output clock `tx_outclock` that has the same frequency as the input clock. The output clock has the same frequency as the input clock, (not the internally multiplied clock), because only the 1× version of the PLL-generated clock can be fed out. The output data transitions 8 times within one period of the 1× clock, indicating that the deserialization factor is 8.

Figure 23 shows the results of an example functional simulation of the deskew circuitry in the LVDS receiver. The deskew pin `rx_deskew` is asserted for at least three clock cycles after the PLL locks onto the incoming clock and the deskew calibration pattern is applied to all channels. If the deskew pin is prematurely de-asserted or the deskew calibration pattern is incorrect, the Quartus software will warn that the deskew pin was de-asserted at an invalid time.

*Figure 23. Example Functional Simulation Waveform of Receiver Calibration*



Figure 24 shows the results of an example functional simulation of an LVDS receiver. The locked pin `rx_locked` is asserted as long as the incoming clock signal is valid. The valid frequency is determined by the value that is set in the MegaWizard. If this frequency does not correspond, the simulation will indicate that the PLL could not lock onto the incoming clock signal.

*Figure 24. Example Functional Simulation Waveform of LVDS Receiver*
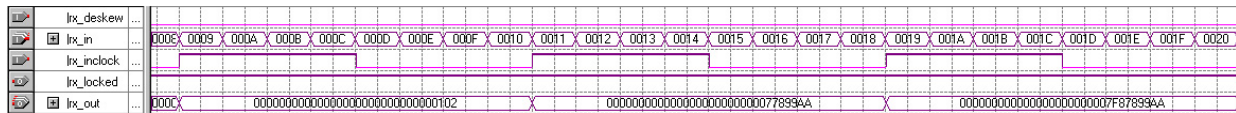


The incoming data is synchronized with the incoming clock signal `rx_inclock`. The input data transitions 8 times within one incoming clock period, indicating that the deserialization factor is 8. The output data is synchronized with the output clock of the receiver module, which is not displayed in the figure.

Figure 25 indicates the bit mapping performed by the Quartus software for a single receiver channel operating in 8× mode at 60 MHz. The functional simulation waveforms show that the data for the current clock cycle of `rx_inclock` (beginning at 41.66ns) map to the appropriate positions, as indicated in Table 3 on page 5. The first bit of data for the current clock cycle is not accepted until 45.83ns, which is immediately after the last two bits of the previous cycle are accepted.

*Figure 25. Bit Mapping Sample Waveform*



## Summary

The dedicated LVDS circuitry in APEX 20KE devices can be easily controlled by using the Quartus software. Internal LVDS PLLs work in conjunction with this circuitry to provide adequate clock speeds for serial-to-parallel and parallel-to-serial conversion while minimizing the required input clock frequency. The Quartus MegaWizard Plug-In Manager interface greatly simplifies the potentially complex task of LVDS module instantiation in designs with its easy-to-use graphical interface. Simulation can be easily performed in standard third party EDA tools as well as within the Quartus software, permitting the user to calibrate the LVDS module and to verify the accuracy of the design.

Following the guidelines provided in this document eases the difficult task of designing circuits for high-speed data transmission.