

Avalon-MM via LVDS Reference Design

Chris Esser
Staff Product FAE
October 7, 2015

1. Introduction

The Avalon Memory-Mapped (Avalon-MM) interface is a control-plane interface that can easily be mapped to most peripherals. It can range from an extremely simple localbus interface, to a complex pipelined interface, capable of handling variable latency burst transactions. This reference design of parameterizable IP blocks allows the Avalon-MM interface to be extended across a high-speed LVDS interface between a set of devices. This might be done for various reasons, such as to simply extend the control plane, or to expand the I/O interface of a device for lower-speed signals.

2. Overview of the Design Archive

Once extracted (you may double-click on the hyperlink below), the Avalon-MM via LVDS reference design archive (AVMM_via_LVDS.zip) contains the following directories:

[AVMM via LVDS Archive](#)



<path>/ AVMM_via_LVDS
Installation Directory.



quartus

Contains top-level Quartus project and constraint files.



CycloneV

Contains top-level Quartus project and constraint files for targeting a CycloneV master device.



Max10

Contains top-level Quartus project and constraint files for targeting a Max10 slave device.



source

Contains all design files for the Avalon-MM via LVDS design. Quartus IP Catalog-generated files are contained in the following subdirectories:



LVDS

Contains the low level LVDS interface blocks, as well as the block that formats and buffers the byte stream to/from LVDS.



qsys

Directory of various hierarchical Qsys systems that comprise the Avalon-MM via LVDS blocks, as well as the complete reference design and simulation testbench.



Inside the Quartus projects for the Cyclone V and Max 10 devices, there are two revisions. The default revision matches the project name, and the other revision starts with “Devkit_”. The “Devkit” revision has some additional constraints that are applicable **ONLY** for demonstration purposes, when connecting the Cyclone V SoC FPGA Development Kit to a Max 10 FPGA Development Kit, and should not be used as a starting point for creating your own project.

3. Architecture

On the master device (which contains the AVMM2LVDS block), Avalon-MM transactions are buffered and translated into Avalon-ST packets by the “Avalon Transactions to Packets” block. Next, the “Avalon-ST Packets to Bytes Converter” encodes the packet into a byte stream that is serialized by the “Bytes to LVDS” block. The 2-bit LVDS interface operates at 5x the rate of the packet clock. On the slave device (containing the LVDS2AVMM block), LVDS data is accepted by the “Bytes to LVDS” block, is converted to packets by the “Avalon-ST Bytes to Packets Converter”, and finally to Avalon-MM transactions by the “Avalon Packets to Transactions converter”. Transaction responses follow this same path in reverse, back to the “Avalon Transactions to Packets” block, for response back to the issuing master.

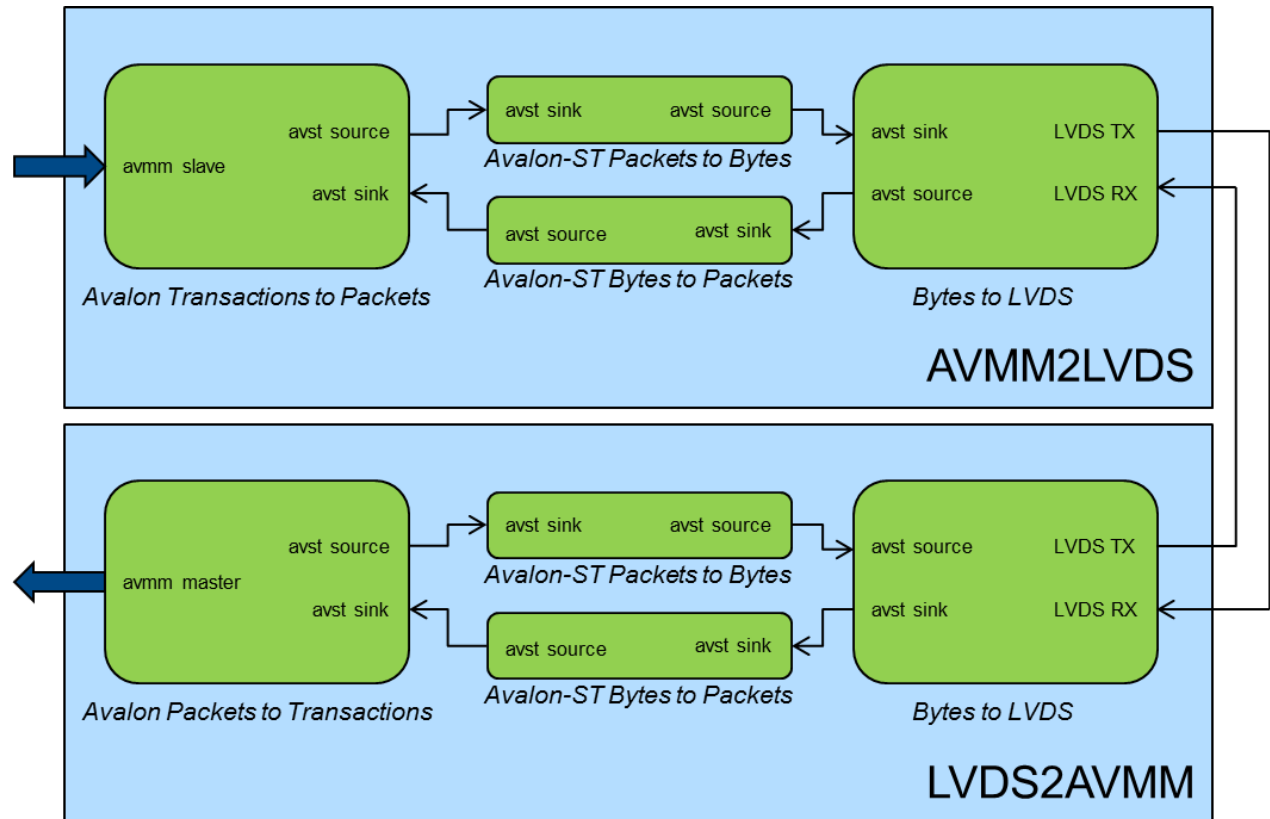


Figure 1: Top-level block diagram

3.1. AVMM2LVDS

The AVMM2LVDS block is the top-level hierarchical Qsys block for the master FPGA. It accepts transactions over an Avalon-MM slave interface, and translates those transactions to a serial LVDS interface. The LVDS interfaces are source-synchronous, meaning that the LVDS_TX interface in the “Bytes to LVDS” block provides a transmit clock synchronous to the transmit data being sent to the LVDS_RX block to which it communicates. The format of the output data stream is shown in Figure 2 below. It is important to note that the relationship between the LVDS_CLK and LVDS_DATA may be different from one device family to the next. However, it will always be consistent for that specific device family – for example, TX_DATA(6) will always be transmitted with the rising edge of LVDS_CLK in Cyclone V, whereas RX_READY is transmitted with the rising edge of LVDS_CLK in Max10. The data from different bytes is represented by a different color.

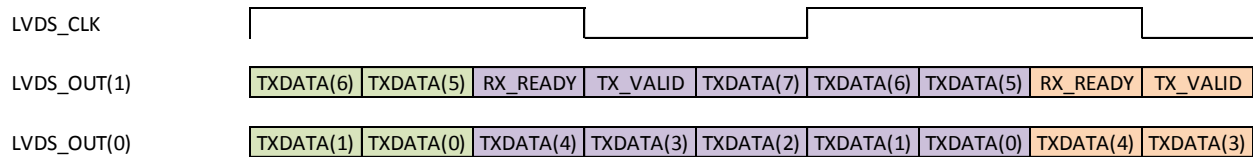


Figure 2: LVDS Streaming Packet Data Format (as observed from the TX perspective)

The RX_READY timeslot is a backpressure indicator, informing back in the upstream direction that it is capable of receiving more data. The TX_VALID timeslot indicates to the downstream direction that the data being sent in the timeslots for TXDATA(7:0) is valid data for packetization.

The LVDS_TX block receives a reference clock that it utilizes for its clock source; by default, its frequency is 1/5th the data rate and is equivalent to the parallel clock rate. However, in case of the AVMM2LVDS block, it simply uses the same reference clock received by the LVDS_RX block as its clock source. This allows us to merge the RX and TX PLLs in the AVMM2LVDS block. Nevertheless, there must be a primary source for the clock somewhere within the system, and due to architectural reasons for the development kit demo platform, it was necessary that it be located on the slave device (i.e. the LVDS_TX reference clock to the LVDS2AVMM block). If desired, this could just as easily be changed to the master device, when configuring your own system. Since all blocks within AVMM2LVDS are on the LVDS RX parallel clock (“rx_clkout”) clock domain, there are no clock domain crossings or clock domain crossing logic within this system. The rx_clkout clock signal is exported from this block, and can be used by any higher-level components (such as the Avalon-MM master to which it connects) to reduce clock domain crossings. However, it is not necessary to do so, and if clock domain crossings exist, Qsys will automatically insert the appropriate clock domain crossing bridges.

3.2. LVDS2AVMM

The LVDS2AVMM block is the top-level hierarchical Qsys block for the slave FPGA. It receives transactions over a serial LVDS interface, translates them back to an Avalon-MM master interface, and responds back, when applicable. The LVDS interfaces are source-synchronous, meaning that the LVDS_TX interface in the “Bytes to LVDS” block provides a transmit clock, synchronous to the TX data, that is used by the LVDS_RX block to which it communicates. The LVDS_TX block also receives a reference clock that it utilizes for its clock source. It is the primary source for the clock within the system, and it needs to operate at the internal parallel clock frequency, which is 1/5th the rate of the serial LVDS data interface. Except for the final stage of the TX serializer (which includes an optional phase compensation FIFO), all blocks within LVDS2AVMM are on the LVDS RX parallel clock (“rx_clkout”) clock domain, and there are no clock domain crossings or clock domain crossing logic within this system. The rx_clkout clock signal is exported from this block, and can be used by any higher-level components (such as the slave peripherals to which the Avalon-MM master connects) to reduce clock domain crossings. However, it is not necessary to do so, and if clock domain crossings exist, Qsys will automatically insert the appropriate clock domain crossing bridges.

3.3. Avalon Transactions to Packets

This block resides in the “master” device, and acts as the slave-side of the Avalon-MM via LVDS bridge design. It interfaces to the Avalon-MM master device, accepting the transaction, and reformats it into a streaming Avalon (Avalon-ST) packet-based transaction. It has a fixed 8-bit data interface (the existing protocol that is being used within the system does not allow for byte enables, thus the 8-bit interface seems to make the most sense), and offers a variable size for both address width and maximum burst size.

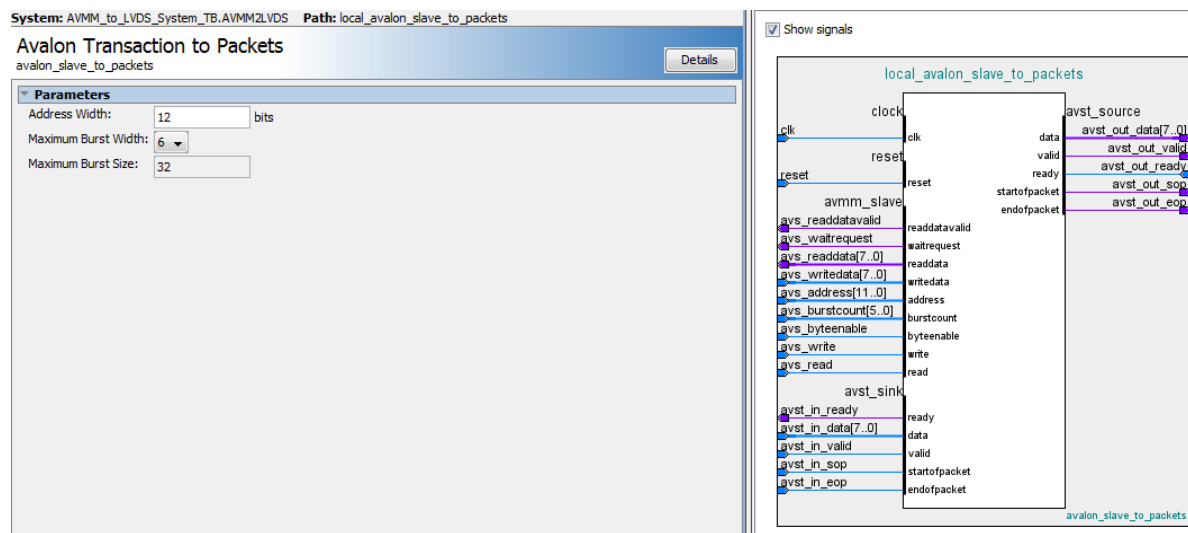


Figure 3: Symbol and Settings for "Avalon Transactions to Packets" block

Note – the address width needs to be set to a value that is large enough to accommodate the size of the address map on your slave device.



It is imperative that you either fully populate the address map on the slave device, such that there are no gaps, or have default slave components to respond to undefined address locations on the slave. Accessing an undefined address that does not have the aforementioned protection, will result in waitrequest being asserted with no slave available to provide the response that would deassert the pending access, locking up the Avalon-MM interface to the master.



If your master doesn't support bursting, set the maximum burst width to "1". This removes the burst capability, and will provide the greatest overall system performance. When burst-capable components are mixed with non-burstable components, adapters are inserted into the fabric, increasing latency and decreasing bandwidth.

The block contains a pending transaction FIFO that ranges from 4 to 32 transactions deep, depending upon the selected burst width.

3.4. *Avalon-ST Packets to Bytes Converter*

This block is part of the Qsys library of basic functions for Avalon-ST adapters. It encodes the Avalon-ST packet into a byte stream for transmission. Further detail about the interface can be found in the [Embedded Peripheral IP User Guide](#).

3.5. *Bytes to LVDS*

This block resides on both sides of the interface for both the master and slave devices. For the transmit path, this block serializes and transfers the byte stream, incorporating the sideband signals (data_ready and data_valid) into a serial bit stream. The resulting 10-bit interface (8 bits of data plus the two sideband signals) is serialized and transmitted across two LVDS channels to the slave device. Therefore, the serial data transmission rate will be at 5x the frequency of the system clock. On the receive path, the data is buffered into a 64 word deep elastic FIFO. This depth is designed to accommodate for multiple transactions in-flight, when backpressure is requested by either the master or slave. The transmit path has an optional phase compensation FIFO, which will be discussed in more detail below.

As shown in Figure 4: Symbol and Settings for "Bytes to LVDS" block, below, the Qsys parameters for this block define whether it is on the master or slave device, allow the user to enable the Phase Compensation FIFO before the TX output, define the length of time before the aligner block declares out-of-sync (OOS), adjust the initial bit-slip alignment for each of the byte deserializers, and specify the device family (since the LVDS SerDes implementations can be different, depending upon device).

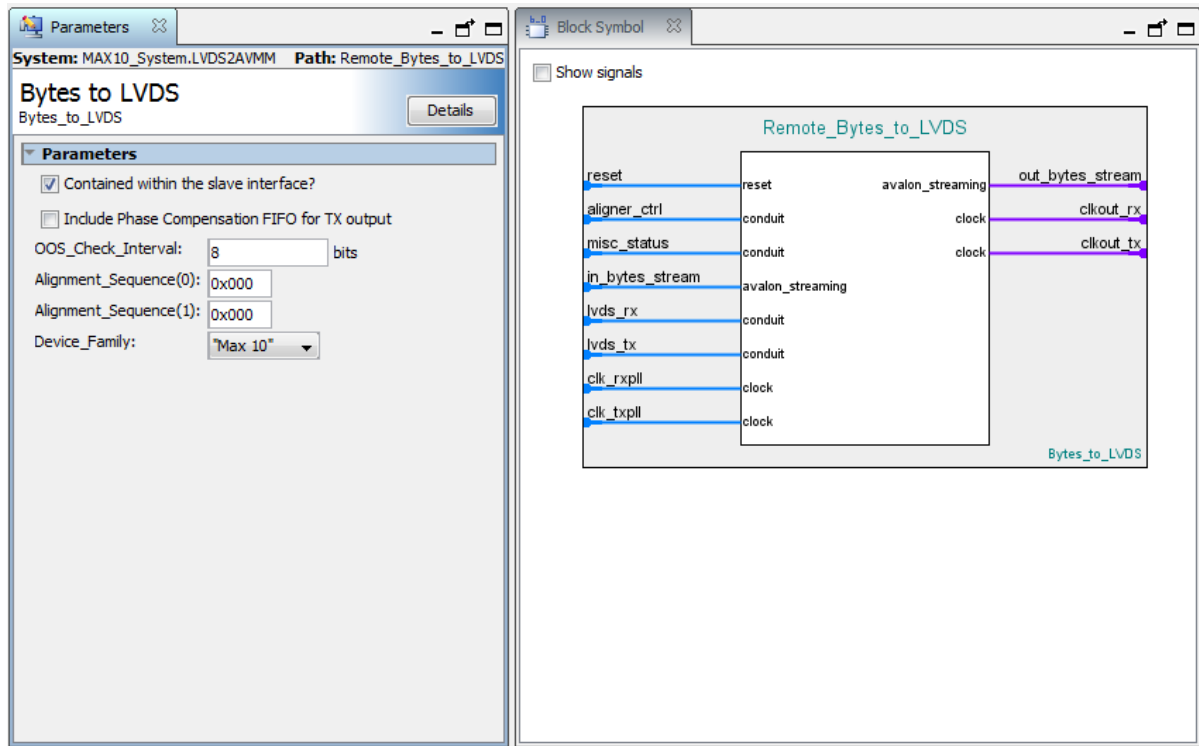


Figure 4: Symbol and Settings for "Bytes to LVDS" block

This block has both automatic as well as manual bit alignment. Alignment is edge-sensitive, and each rising edge will cause the deserializer to shift the data right by one bit position. This can be manually accomplished by sending a series of pulses on the "aligner_shift" input. The initial (power-on) position can also be set by adjusting the "Alignment Sequence" parameter. Just like the aligner_shift signal, the Alignment_Sequence parameter needs to be a stream of 1 / 0 pulses, where each '1' sent results in an equivalent number of "shift right" operations to the parallelized output byte. For example, sending "0x00a" transmits two pulses to the alignment block, resulting in a shift right of two bits. The appropriate number of shifts should ultimately be determined on your hardware system, and can most easily be determined by monitoring the parallel output data at both LVDS receivers. At startup with no traffic present, this value should be 0x200, and can easily be monitored from within SignalTap. Once identified, this will be a static value for your system, and that value should never change. In addition, there is an aligner_ena signal, which is used to turn on or off the automatic bit aligner. At startup, the master device transmits 0x200 across the interface to the slave device. The interface comprises {RX ready to accept data, TX data valid, and TX data(8:0)}, so 0x200 indicates that the master device is ready to accept data, and no data is currently being sent. The slave device will send all zeroes on its TX interface until the aligner has bit slipped the receiver to align to the 0x200 pattern. Once the RX slave is in sync, it will send 0x200 on its TX interface and allow the master to sync up. This is to ensure that the entire interface is aligned before data can be transmitted in either direction. There is a state machine within the block that detects what are considered "out of state" data conditions. These conditions consist of the databus either being all zeroes, or a data pattern with only a single bit set that isn't 0x200 (the in-sync pattern). If the interface is in this

state for 2⁸OOS_Check_Interval clock cycles (the default is 2⁸, or 256 bytes), the aligner state machine will declare out of sync, and attempt to re-align the RX data interface.



Note – it is possible for pathologic data patterns to exist that could result in causing the aligner to falsely declare “out of sync”. For example, it could be possible for the transmitting device to backpressure the other side (i.e. indicate that it is not ready to receive data), while the transmitting device is transmitting valid data that is all zeroes. This would result in a pattern of 0x100, which could cause the state machine to issue a series of four aligner pulses, in order to align it to 0x200. You need to determine if this potential condition could occur in your system for an extended duration, and either increase the OOS counter, or selectively enable and disable the auto aligner (aligner_ena).

The Phase Compensation FIFO crosses the clock domain boundaries from the LVDS RX output clock to the LVDS TX output clock. It is a self-centering FIFO, which will pause transmitting if the FIFO is in danger of underflow, and will clear itself when in danger of overflow. Thus, it is expected that the RX and TX clocks be frequency aligned, but not necessarily phase aligned. It is assumed that all blocks for the AVMM via LVDS function operate on the RX output clock domain. If the same PLL is being used for both the TX and RX LVDS interfaces, it is not necessary to enable the phase compensation FIFO. This will save 3 clock cycles of latency within the system.

3.6. Avalon-ST Bytes to Packets Converter

This block is part of the Qsys library of basic functions for Avalon-ST adapters. It decodes the byte stream into an Avalon-ST packet. Further detail about the interface can be found in the [Embedded Peripheral IP User Guide](#).

3.7. Avalon Packets to Transactions

This block is part of the Qsys library of basic functions for Avalon-ST adapters. It receives Avalon-ST packet transactions, and initiates Avalon-MM transactions. Avalon-MM responses are translated back into Avalon-ST packets, for response to the requesting master. Further detail about the interface can be found in the [Embedded Peripheral IP User Guide](#).

3.8. Latency

Latency is dependent on the devices being used for both the master as well as the slave component. In addition, throughput is highly dependent on transactions being pipelined throughout the control path. There are numerous FIFOs throughout the blocks which are used to queue up transactions, in order to support backpressure and clock domain crossings. In order to understand precise latencies for your specific interface, it is recommended that you perform functional simulation for the exact configuration, devices, and traffic pattern. The information below in Table 1 and Table 2 shows the latencies for Cyclone V and Max 10 devices. (Other device mixes can affect latency, primarily due to differences in the implementations for the LVDS SerDes blocks. The hard IP blocks in Cyclone V are 7 clock cycles faster than those in Max10, although throughput remains identical, due to pipelining.) For a non-bursted transaction, a write transaction takes approximately 16 clock cycles to complete, and a read transaction takes approximately 22 clock cycles (on average, assuming pipelined transactions). Bursted transactions can significantly increase throughput efficiency, assuming that both the master as well as the downstream slaves are capable of effectively performing bursts. For example, a single pipelined read might take ~22 clock cycles to complete, whereas a 32 byte burst read would only increase the access by an additional 31 cycles, for an average access of 1.66 clock cycles per byte read. For a Max10 interface running at 800Mbps in both directions, this equates to a write throughput of 871.5Mbps and a read throughput of

772.8Mbps. Again, you would really need to simulate your system, to understand how it would ultimately perform, and to identify potential bottlenecks.

Here are tables, listing the approximate latencies for the various design stages within the system. The units listed are given as parallel clock cycles.

Design Block	Latency (Cyclone V)	Latency (Max 10)
Avalon Transactions to Packets	2 cycles	2 cycles
Avalon-ST Packets to Bytes Converter	1 cycle	1 cycle
Bytes to LVDS (TX) Optional Phase Comp FIFO	(3 cycles)	(3 cycles)
Bytes to LVDS (TX)	1 cycle	2 cycles
Bytes to LVDS (RX deserialization)	2 cycles (appx)	8 cycles (appx)
Bytes to LVDS (RX FIFO)	2 cycles	2 cycles
Avalon-ST Bytes to Packets Converter	3 cycles	3 cycles
Avalon Packets to Transactions (SOP to Read or Write request)	10 cycles	10 cycles

Table 1: Transaction Latencies (Write or Read request): Master to Slave Device

Design Block	Latency (Cyclone V)	Latency (Max 10)
Avalon Packets to Transactions (readdatavalid to SOP)	7 cycles	7 cycles
Avalon-ST Packets to Bytes Converter	1 cycle	1 cycle
Bytes to LVDS (TX) Optional Phase Comp FIFO	(3 cycles)	(3 cycles)
Bytes to LVDS (TX)	1 cycle	2 cycles
Bytes to LVDS (RX deserialization)	2 cycles (appx)	8 cycles (appx)
Bytes to LVDS (RX FIFO)	2 cycles	2 cycles
Avalon-ST Bytes to Packets Converter	3 cycles	3 cycles
Avalon Transactions to Packets (SOP to readdatavalid)	1 cycle	1 cycle

Table 2: Transaction Latencies (Read response): Slave to Master Device

4. Simulating the Complete System

Contained within the source/qsys directory is a top-level testbench that incorporates the AVMM2LVDS and LVDS2AVMM blocks, as well as an Avalon-MM traffic generator (which generates the Avalon-MM transactions and validates the responses), and an on-chip RAM block (to which the Avalon-MM transactions are directed). The testbench provides an excellent framework for understanding and validating the functionality of the design blocks, especially once you customize them for your specific application.

In order to start the simulation, simply open the “AVMM_to_LVDS_System_TB.qsys” file, and generate the HDL – make sure that the Verilog simulation model is being generated:

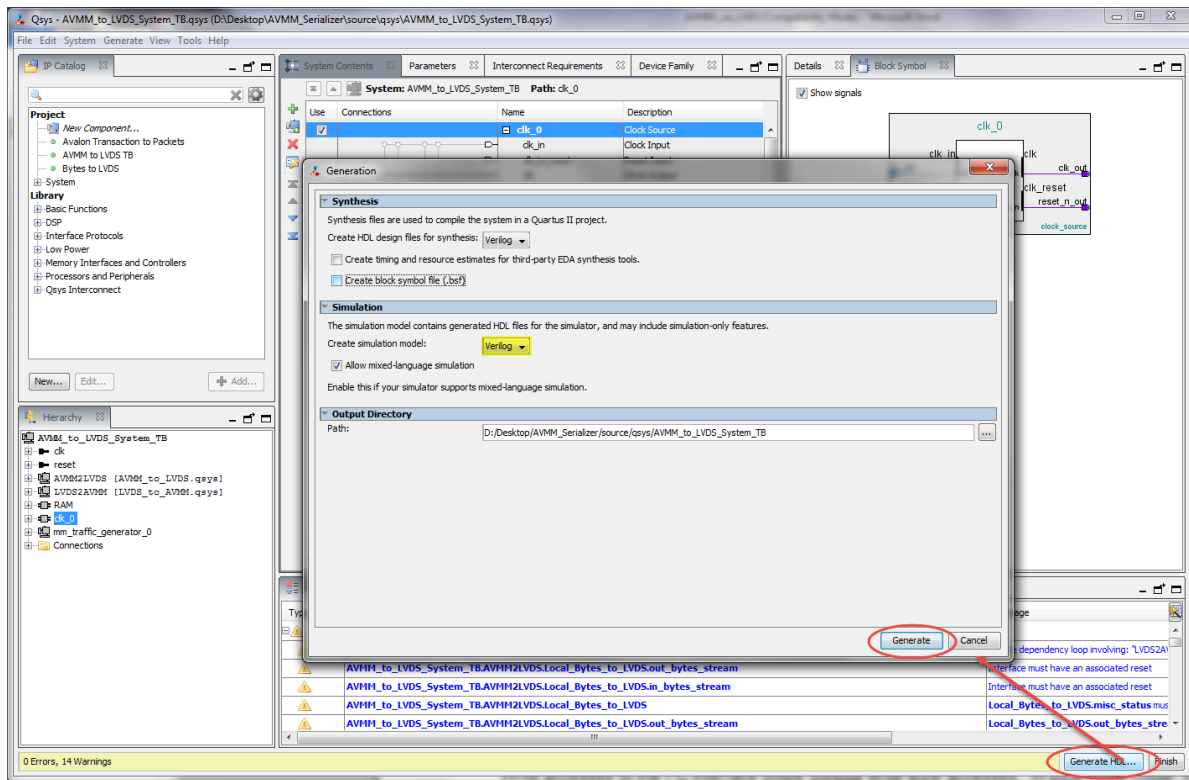


Figure 5: Opening the Example Testbench

Qsys will spend a minute or so, generating the HDL for the system, which will be found in `source\qsys\AVMM_to_LVDS_System_TB\simulation\submodules`.

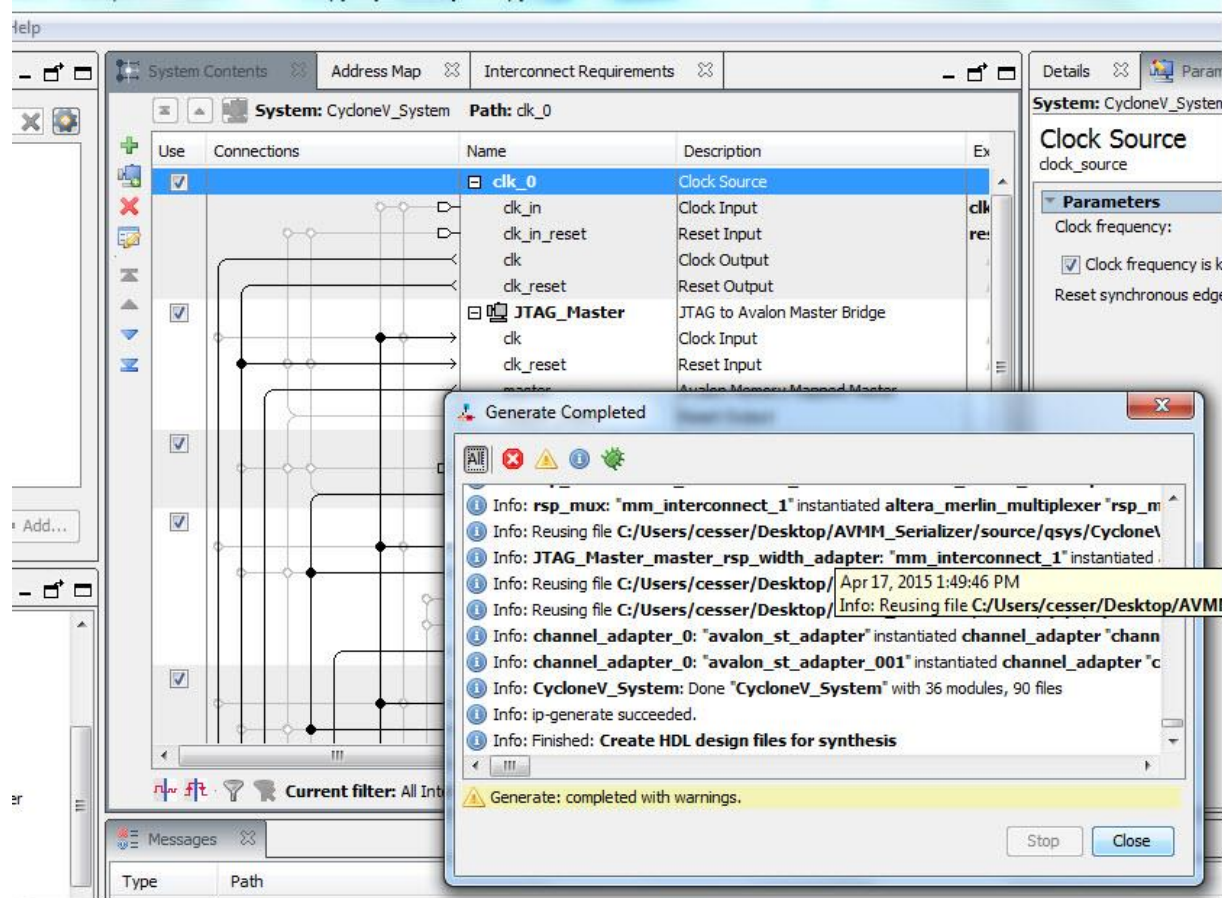


Figure 6: Generating the Example Testbench

Next, start QuestaSim (or your desired simulation tool). Contained within the `source\qsys\AVMM_to_LVDS_System_TB\simulation\mentor` directory are a couple files that are used to set up the simulation environment and run the simulation. In the console window for QuestaSim, type `"source msim_setup.tcl"`. This script contains a number of command aliases to compile and simulate the design, and a description on how to utilize them is echoed back to the console.

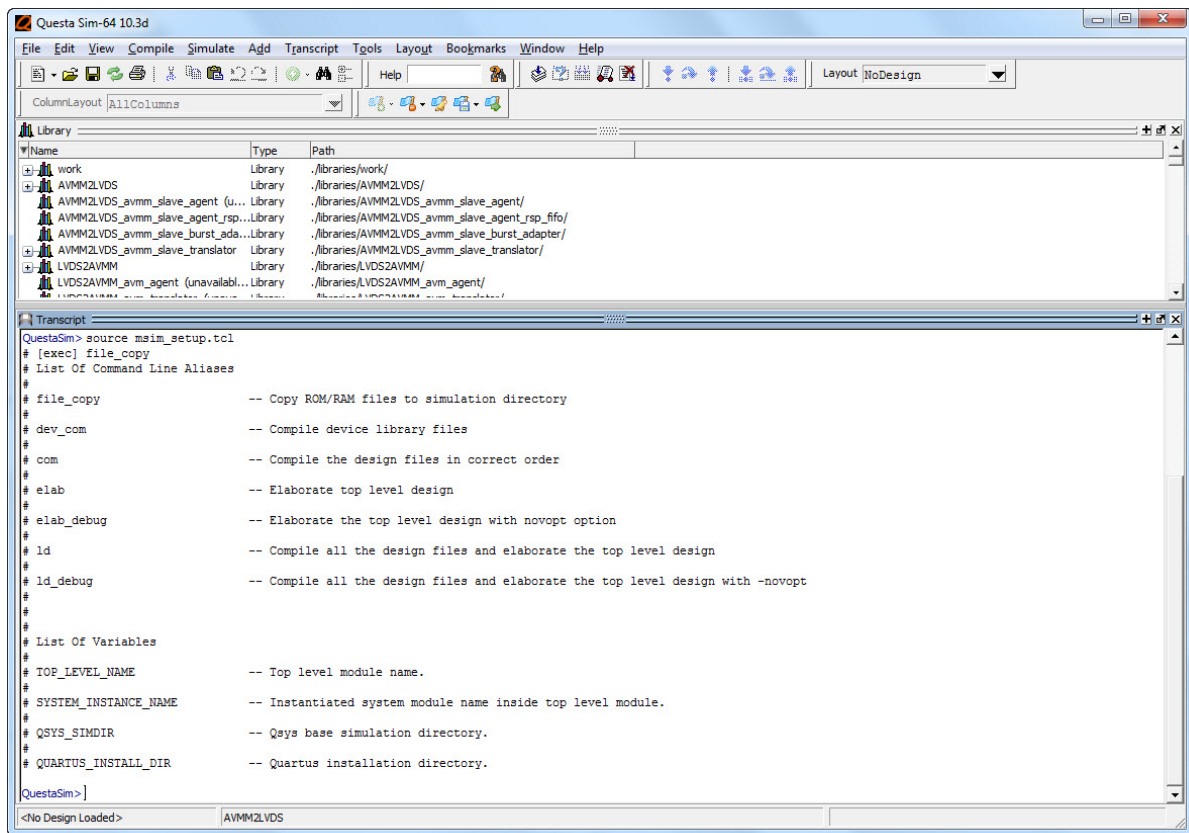


Figure 7: Opening QuestaSim on the Example Testbench

Type "ld_debug". This will compile the complete set of Quartus libraries, design libraries, and start the simulation.

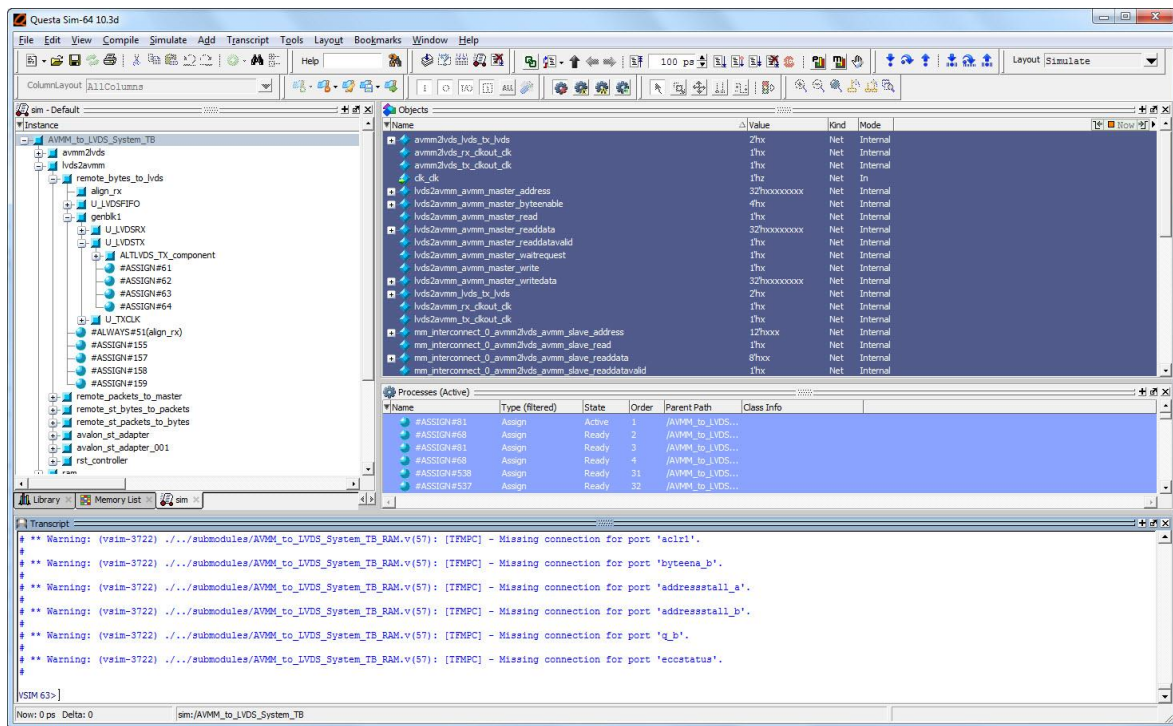


Figure 8: Using QuestaSim to Compile and Elaborate the Example Testbench

Next, type “source wave.do” to open up a representative listing of signals at various points within the design.

Lastly, type “source force.do” to initialize some signals, provide clock and reset, and run the simulation (“run 200 us”) until the activity on the Avalon-MM interface stops (~200us). You should see the top two signals in the wave window, *test_complete* and *pass*, asserted at the end of the simulation, as they are in Figure 9 below.

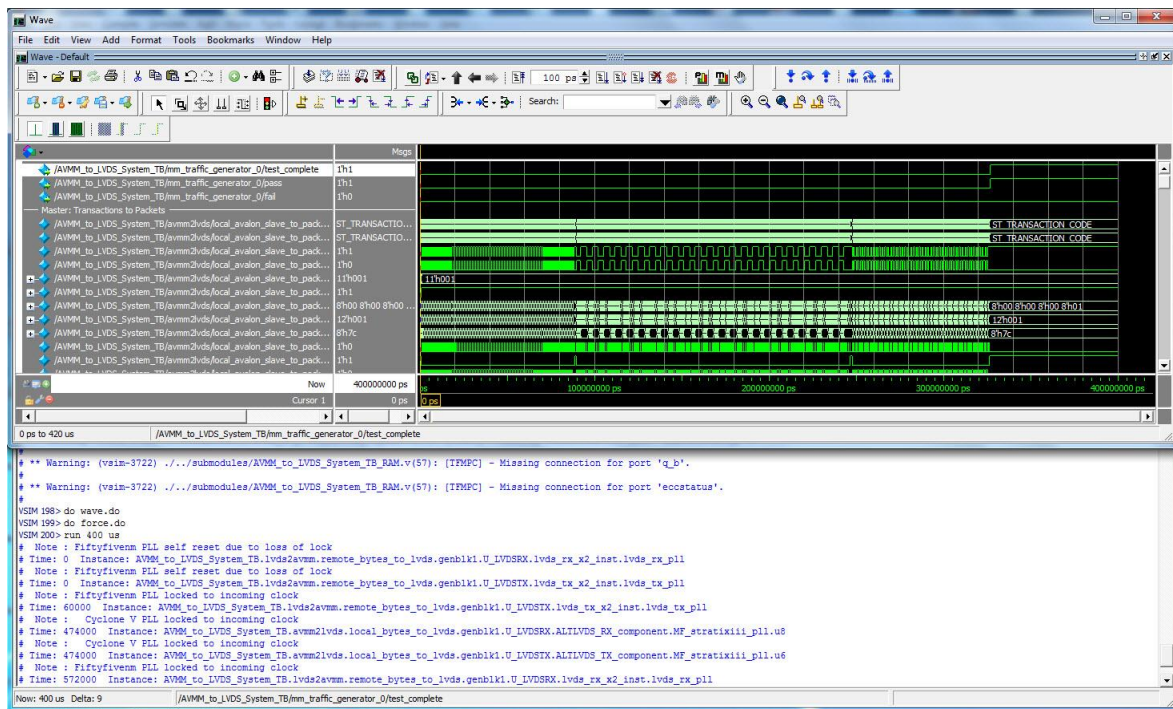


Figure 9: Using QuestaSim to Simulate the Example Testbench

5. Customizing the Design Example

The design example consists of a Cyclone V master device that utilizes the CycloneV_System.qsys as its top-level entity. The CycloneV_System block contains the AVMM2LVDS sub-hierarchy, an Avalon-MM traffic generator (for automated testing and validation of the Avalon-MM interface to the AVMM2LVDS block), and a JTAG to Avalon Master Bridge (for manual access to the Avalon-MM interface on the AVMM2LVDS block, via System Console). The slave device is implemented in a Max10 FPGA. It's top-level entity utilizes the Qsys system MAX10_System.qsys, which instantiates the LVDS2AVMM sub-hierarchy, which is connected to an on-chip RAM and eight bits of GPIO output, for which the five LSBs drive user LEDs on the board. Although this design example has been implemented in actual hardware on the Cyclone V SOC and Max10 FPGA Development Kits, certain modifications needed to be made to the LVDS interface, in order for it to be implemented within the physical HW limitations of the development kit interconnections that would not have been done in a custom application.

When modifying the example design to suit your own system requirements, typical customizations of the system might involve:

- Setting the width of the address bus in the Avalon Transactions to Packets interface, for the address space visible at the slave device.
- Setting the maximum burst size in the Avalon Transactions to Packets interface.
- Setting the target device family in the Bytes to LVDS interface, for FPGA family that will be used to implement both the master device (AVMM2LVDS), as well as the slave device (LVDS2AVMM).
- Adjusting the PLL settings, in order to tune the reference clock or data rate to your application. This can be done by opening and modifying the *lvds_rx_x2_m10.v*, *lvds_tx_x2_m10.v*, *lvds_rx_x2_c5.v*, and *lvds_tx_x2_c5.v* files in the IP Catalog. The interfaces in the design example are currently set to expect the following data rates and reference clock frequencies:

Design Block	Data Rate	Refclk Freq
Max10: TX LVDS (<i>lvds_tx_x2_m10.v</i>)	500 Mbps	50 MHz
Max10: RX LVDS (<i>lvds_rx_x2_m10.v</i>)	500 Mbps	100 MHz
Cyclone V: TX LVDS (<i>lvds_tx_x2_c5.v</i>)	500 Mbps	50 MHz ¹
Cyclone V: RX LVDS (<i>lvds_rx_x2_c5.v</i>)	500 Mbps	50 MHz ¹

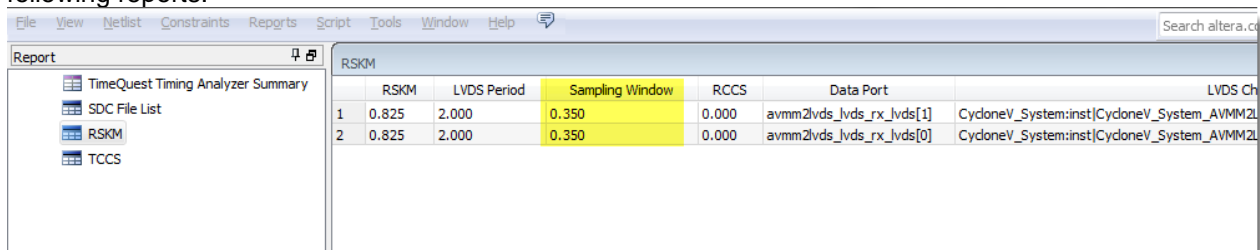
Table 3: Example Design Data Rates and Reference Clock Frequencies

¹ The normal reference clock frequency should be 1/5th the data rate, or 100MHz. This is due to a bug in the Max 10 Soft LVDS IP, discussed in Section 9: *Known Issues*.

6. Static Timing Analysis of the Design Example

When performing static timing analysis between a master and slave device that has hard LVDS SerDes capabilities (such as two Cyclone V devices), timing analysis is rather simple. You simply need to ensure that the value reported by TimeQuest for the receiver skew margin (RSKM) in each device is greater than the transmit channel to channel skew (TCCS) value reported by TimeQuest for the other device.

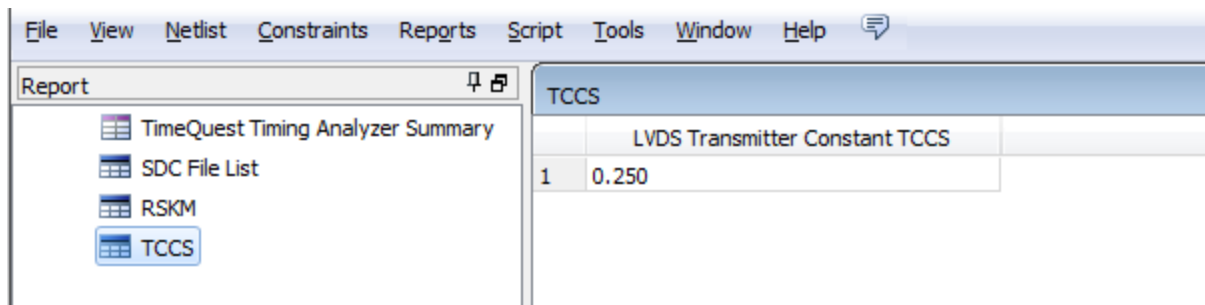
Different FPGA families have differing LVDS capabilities. The Max 10 and previous Cyclone families (Cyclone IV and earlier) utilize soft SerDes blocks, where data is serialized and deserialized through a sequence of double data rate flip-flops. Cyclone V, Arria, and Stratix families utilize hard SerDes blocks in the I/O ring, which are capable of faster data rates and offer lower latency. The design example illustrates the case where we are interfacing a device that has hard LVDS SerDes capabilities (a Cyclone V SoC device) to a device that has soft LVDS SerDes (the Max 10 device). TCCS and RSKM are only calculated for devices that have hard SerDes blocks, so timing analysis needs to be done a little differently for Max 10. In this instance, the static timing analysis needs to be completed for the device with hard SerDes first, and the calculated RSKM and TCCS values need to be applied to the device with soft SerDes blocks. So, in looking at the timing analysis for the Cyclone V master device, we see the following reports:



The screenshot shows the TimeQuest Reports window with the RSKM report selected. The report displays the following data:

	RSKM	LVDS Period	Sampling Window	RCCS	Data Port	LVDS Ch
1	0.825	2.000	0.350	0.000	avmm2lvds_lvds_rx_lvds[1]	CycloneV_System:inst CycloneV_System_AVMM2L
2	0.825	2.000	0.350	0.000	avmm2lvds_lvds_rx_lvds[0]	CycloneV_System:inst CycloneV_System_AVMM2L

Figure 10: Extracting the Sampling Window for TX LVDS Static Timing Analysis



The screenshot shows the TimeQuest Reports window with the TCCS report selected. The report displays the following data:

	LVDS Transmitter Constant TCCS
1	0.250

Figure 11: Extracting the TCCS for RX LVDS Static Timing Analysis

From the above reports, we can see that the sampling window is 350ps, and the RSKM is 250ps. Next, we simply apply these values to the timing constraints for the Max 10 device in the "Max10_LVDS2AVMM.sdc" file, and compile the Max 10 design:


```

1  # Define the parallel clock period for the LVDS interface (100MHz * 5 = 500Mbps)
2  set clk_period 10.0
3
4
5
6  set pcb_skew 0.030 ; # uncertainty between clock and data trace skew
7  set ext_tccs_skew 0.250 ; # This value should be copied from the TCCS report of the interfacing device
8  set ext_sampling_window 0.350 ; # This value should be copied from the RKSM analysis of the interfacing device
9
10 set lvds_input_delay_max [expr $ext_tccs_skew / 2 + $pcb_skew]
11 set lvds_input_delay_min [expr 0 - $ext_tccs_skew / 2 - $pcb_skew]
12
13 set lvds_output_delay_max [expr $ext_sampling_window / 2 + $pcb_skew]
14 set lvds_output_delay_min [expr 0 - $ext_sampling_window / 2 - $pcb_skew]
15
16 # The reference clock is 1/2 the frequency that it's supposed to be, due to
17 # an error in the Max10 ALTLVDS block
18 create_clock -name lvds2avmm_lvds_rx_clk -period $clk_period [get_ports {lvds2avmm_lvds_rx_clk}]
19 create_clock -name data_clk_virt -period [expr $clk_period / 5]
20
21 create_clock -name clk_50M -period 50.0MHz [get_ports {clk_50M}]
22
23 derive_pll_clocks
24 derive_clock_uncertainty
25
26 # Derive the LVDS TX clock at the output pin
27 create_generated_clock -name lvds2avmm_tx_clkout_clk -source [get_pins {inst|lvds2avmm|remote_bytes_to_lvds|U_LVDS2AVMM|lvds2avmm_tx_clkout_clk}]
28 create_generated_clock -name lvds2avmm_tx_clkout_dff -source [get_pins {inst|lvds2avmm|remote_bytes_to_lvds|U_LVDS2AVMM|lvds2avmm_tx_clkout_dff}]

```

Figure 12: Applying the Extracted Sampling Window and TCCS Values to Static Timing Constraints File

Analyzing the results in TimeQuest should yield positive slack for all I/O paths:

Slack	From Node	To Node
1 0.220	lvds2avmm_lvds_rx_lvds[0]	MAX10_System:inst MAX10_System_LVDS2AVMM:lvds...0_0002_lvds_ddio_in_jka:ddio_in[ddio_l_reg[0]
2 0.220	lvds2avmm_lvds_rx_lvds[1]	MAX10_System:inst MAX10_System_LVDS2AVMM:lvds...0_0002_lvds_ddio_in_jka:ddio_in[ddio_l_reg[1]
3 0.241	lvds2avmm_lvds_rx_lvds[0]	MAX10_System:inst MAX10_System_LVDS2AVMM:lvds...0_0002_lvds_ddio_in_jka:ddio_in[ddio_h_reg[0]
4 0.241	lvds2avmm_lvds_rx_lvds[1]	MAX10_System:inst MAX10_System_LVDS2AVMM:lvds...0_0002_lvds_ddio_in_jka:ddio_in[ddio_h_reg[1]

Figure 13: Analyzing the I/O Timing Constraint Results

7. Pin Description

NAME	DIRECTION	TYPE	DESCRIPTION
reset_reset_n	Input	<i>Any</i>	Global hardware reset. This signal flushes out the control path, including all FIFOs, as well as resynchronizes the LVDS SerDes. Active low input.
rx_clkin_clk	Input	<i>Any</i>	Reference clock for the RX LVDS.
rx_clkout_clk	Output	LVDS	Parallel output clock for the deserialized LVDS received data. This clock is currently used as the clock source for the entire design.
lvds_rx_lvds[1..0]	Input	LVDS	High speed LVDS RX data, which operates at 5x the data rate of the reference clock.
tx_clkin_clk	Input	<i>Any</i>	Reference clock for the TX LVDS interface. Note, this input is currently unused, since the RX reference clock is used to clock out the data.
tx_clkout_clk	Output	LVDS	Parallel output clock for the deserialized LVDS transmit data.
lvds_tx_lvds[1..0]	Input	LVDS	High speed LVDS TX data, which operates at 5x the data rate of the reference clock.
avmm_slave_address(n..0)	Input	<i>Any</i>	Avalon-MM address bus, which has a parameterized width.
avmm_slave_burstcount(n..0)	Input	<i>Any</i>	Avalon-MM burst count, which can be parameterized to match the size of the downstream slave peripherals.
avmm_slave_waitrequest	Input	<i>Any</i>	Avalon-MM wait request signal, which indicates whether the slave is capable of accepting a transaction.
avmm_slave_read	Input	<i>Any</i>	Avalon-MM read request.
avmm_slave_readdata	Input	<i>Any</i>	Avalon-MM read databus.
avmm_slave_readdatavalid	Input	<i>Any</i>	Avalon-MM read data valid indicator.
avmm_slave_write	Input	<i>Any</i>	Avalon-MM write request.
avmm_slave_writedata	Input	<i>Any</i>	Avalon-MM write databus.
aligner_ctrl_aligner_ena	Input	<i>Any</i>	Enables the automatic bit aligner for the RX LVDS interface
aligner_ctrl_aligner_shift	Input	<i>Any</i>	Manually slips the bit aligner by one bit to the right
aligner_ctrl_aligner_oos	Output	<i>Any</i>	Indicates that the aligner state machine has determined that the RX bit alignment is not in-sync

Table 4: AVMM2LVDS Signal Table

NAME	DIRECTION	TYPE	DESCRIPTION
reset_reset_n	Input	<i>Any</i>	Global hardware reset. This signal flushes out the control path, including all FIFOs, as well as resynchronizes the LVDS SerDes. Active low input.
rx_clkin_clk	Input	<i>Any</i>	Reference clock for the RX LVDS.
rx_clkout_clk	Output	LVDS	Parallel output clock for the deserialized LVDS received data. This clock is currently used as the clock source for the entire design.
lvds_rx_lvds[1..0]	Input	LVDS	High speed LVDS RX data, which operates at 5x the data rate of the reference clock.
tx_clkin_clk	Input	<i>Any</i>	Reference clock for the TX LVDS interface. This is the primary clock source for the system, and it should operate at the internal parallel clock frequency, which is 1/5 th the rate of the serial LVDS data interface.
tx_clkout_clk	Output	LVDS	Parallel output clock for the deserialized LVDS transmit data, used to retune the output data just prior to transmission.
lvds_tx_lvds[1..0]	Input	LVDS	High speed LVDS TX data, which operates at 5x the data rate of the reference clock.
avmm_master_address(31..0)	Output	<i>Any</i>	Avalon-MM address bus
avmm_master_byteenable(3..0)	Output	<i>Any</i>	Avalon-MM byte enable
avmm_master_waitrequest	Input	<i>Any</i>	Avalon-MM wait request signal, which indicates whether the slave is capable of accepting a transaction.
avmm_master_read	Output	<i>Any</i>	Avalon-MM read request.
avmm_master_readdata	Input	<i>Any</i>	Avalon-MM read databus.
avmm_master_readdatavalid	Input	<i>Any</i>	Avalon-MM read data valid indicator.
avmm_master_write	Output	<i>Any</i>	Avalon-MM write request.
avmm_master_writedata	Output	<i>Any</i>	Avalon-MM write databus.
aligner_ctrl_aligner_ena	Input	<i>Any</i>	Enables the automatic bit aligner for the RX LVDS interface
aligner_ctrl_aligner_shift	Input	<i>Any</i>	Manually slips the bit aligner by one bit to the right
aligner_ctrl_aligner_oos	Output	<i>Any</i>	Indicates that the aligner state machine has determined that the RX bit alignment is not in-sync

Table 5: LVDS2AVMM Signal Table

8. Resource Utilization

The following table estimates the approximate resources required to implement the Avalon-MM via LVDS blocks (estimates from Quartus 15.0.2).

Design Block	Cyclone V		Max 10	
	ALUTs	M10Ks	LEs	M9Ks
Avalon Transactions to Packets	158	2		
Avalon-ST Packets to Bytes Converter	37	0	30	0
Bytes to LVDS	109	1	296	2
Avalon-ST Bytes to Packets Converter	10	0	23	0
Avalon Packets to Transactions			691	0

Table 6: Resource Utilization Estimate

9. Known Issues

- The ALTLVDS_TX block for Max10 does not output the correct clock frequencies. It should transmit a clock that is $1/5^{\text{th}}$ the frequency of the data rate, but transmits a clock that is $1/10^{\text{th}}$ the data rate. In the example design, the Cyclone V LVDS RX and TX blocks (which both use the LVDS rx_refclk) assume that they are receiving their input from a Max10 device, and are thus set to accept a $1/10$ rate input clock frequency. If a different device family is used for the slave (such as Cyclone V), the input frequency settings must be changed back to the $1/5^{\text{th}}$ frequency values.
 - This errata is corrected in Quartus 15.1
- The ALTLVDS_TX block for Max10 generates a parallel clock that is $1/2$ the correct frequency. Therefore, it is not possible to implement a phase compensation FIFO in the “Bytes to LVDS” block for the Max 10 device family.
 - This is scheduled to be corrected in Quartus 16.0.

10. Future Enhancements

- Add clock and reset generation to testbench, such that simulation force files are not required (this assists when using simulation tools other than ModelSim/QuestaSim).
- Incorporate the ALTLVDS parameters into the settings for the “Bytes to LVDS” block, such that the ALTLVDS_RX and ALTLVDS_TX blocks do not need to be regenerated for changes to the reference clock or data rate.
- Add support for other device families, other than Cyclone V and Max 10 (this only applies to the “Bytes to LVDS” block).