XAPP1126 (v1.0) December 10, 2008

Reference System: Designing an EDK Custom Peripheral with a LocalLink Interface

Author: James Lucero

Abstract

This application note discusses the designing of an EDK core with a LocalLink interface. The Create IP Wizard application is used to create the PLB v4.6 slave interface, then the LocalLink interface is added to the core. To create simple user logic with the LocalLink interface, the payload data transmitted from the Hard DMA (HDMA) is looped back to the payload on the receive channel.

The XPS LL EXAMPLE core is connected to the Master PLB (MPLB) for the slave registers and the LocalLink interface is connected to the first HDMA block on the PowerPC® 440 processor block. ChipScope™ Analyzer is used to verify the LocalLink loopback functionality of the core.

A stand-alone software application is included to initiate a single DMA transaction. The receive channel is set up to accept and verify the Rx payload data is the same as the Tx payload data.

This application note describes adding the loopback user logic for the LocalLink interface, adding the XPS LL EXAMPLE core to a PowerPC 440 processor system, and running the software application and analyzing the results within the ChipScope tool. The Xilinx ML507 Rev A board is used for this reference system.

Included Systems

Included with this application note is one reference system, V5_PPC440_ll_example, built for the Xilinx ML507 Rev A board. The reference system is available in the following ZIP file available at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=114065

Introduction

With PLB v4.6 IPIFs, DMA functionality is not included for the user logic. The user has a choice of connecting XPS Central DMA to the system for simple DMA or adding a LocalLink interface to the user logic for Scatter-Gather DMA (SGDMA). For more information about DMA solutions, see the Migration of DMA Solutions chapter in the UG443 PLB v3.4 and OPB to PLB v4.6 System and Core Migration User Guide.

The LocalLink interface is connected to the DMA engine in the HDMA or Soft DMA (SDMA). In this application note the LocalLink interface is connected to the HDMA on the PowerPCC 440 processor block.

In this application note, adding a LocalLink interface with loopback functionality to the user logic generated by the Create IP Wizard is discussed.

Hardware Requirements

The hardware requirements for this reference system are:

- Xilinx ML507 Rev A board
- Xilinx Platform USB or Parallel IV programming cable
- RS232 serial cable and serial communication utility (HyperTerminal)
- Xilinx Platform Studio 10.1.03
- Xilinx Integrated Software Environment (ISE®) 10.1.03

Reference System Specifics

This system uses the PowerPC 440 processor block with a processor frequency of 400 MHz and the Memory Interface Block (MIB) frequency of 266 MHz. The processor block crossbar is set to 266 MHz. In addition, the MPLB and the first HDMA port of the crossbar is set to 133 MHz.

The reference system includes PPC440MC DDR2, XPS LL EXAMPLE, XPS BRAM, XPS UART16550, XPS GPIO, and XPS INTC.

PPC440MC DDR2 is connected to the MIB of the processor block with a frequency of 266 MHz.

The XPS BRAM, XPS GPIO, and XPS UART16550 cores are connected as slaves to the PLB v4.6 instance connected to the MPLB.

The XPS LL EXAMPLE core which contains a LocalLink interface is connected to the first HDMA port. The core's PLB v4.6 slave interface is connected to MPLB.

A ChipScope ILA core is added for the LocalLink interface signals. Adding this core to this system is not discussed in this application note.

See Figure 1 for the block diagram and Table 1 for the address map of the system.

Block Diagram

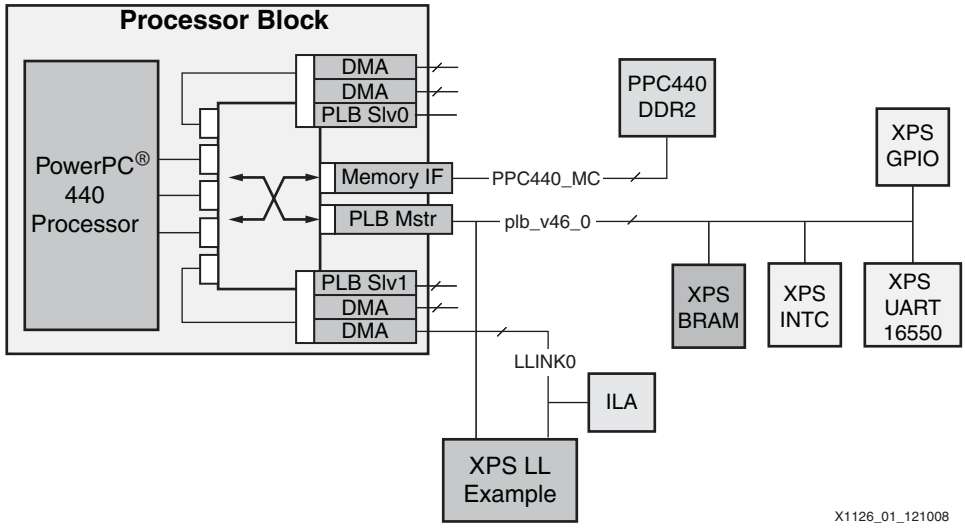


Figure 1: **Reference System Block Diagram**

Address Map

Table 1: **Reference System Address Map**

Peripheral	Instance	Base Address	High Address
ppc440mc_ddr2	ppc440_mc_ddr2_0	0x00000000	0x0FFFFFFF
xps_uart16550	xps_uart16550_0	0x40400000	0x4040FFFF
xps_gpio	LEDs_8Bit	0x40000000	0x4000FFFF
xps_bram_if_cntlr	xps_bram_if_cntlr_1	0xFFFF0000	0xFFFFFFFF
xps_intc	xps_intc_0	0x41200000	0x4120FFFF
xps_ll_example	xps_ll_example_0	0x60000000	0x6000FFFF

Overview of XPS LL EXAMPLE core

The user logic of the XPS LL EXAMPLE core receives the Tx payload on the LocalLink Tx interface from the HDMA and the user logic transmits the Tx payload data on the LocalLink Rx interface to the HDMA. For the loopback functionality, the user logic includes a FIFO. With the XPS LL EXAMPLE core, the Tx and Rx channels transmit and receive in serial. The purpose of the XPS LL EXAMPLE is to show how to create logic with the LocalLink interface and not to illustrate logic for performing flow control. This section covers the logistics of the core.

The block diagram of the user logic is shown in Figure 2.

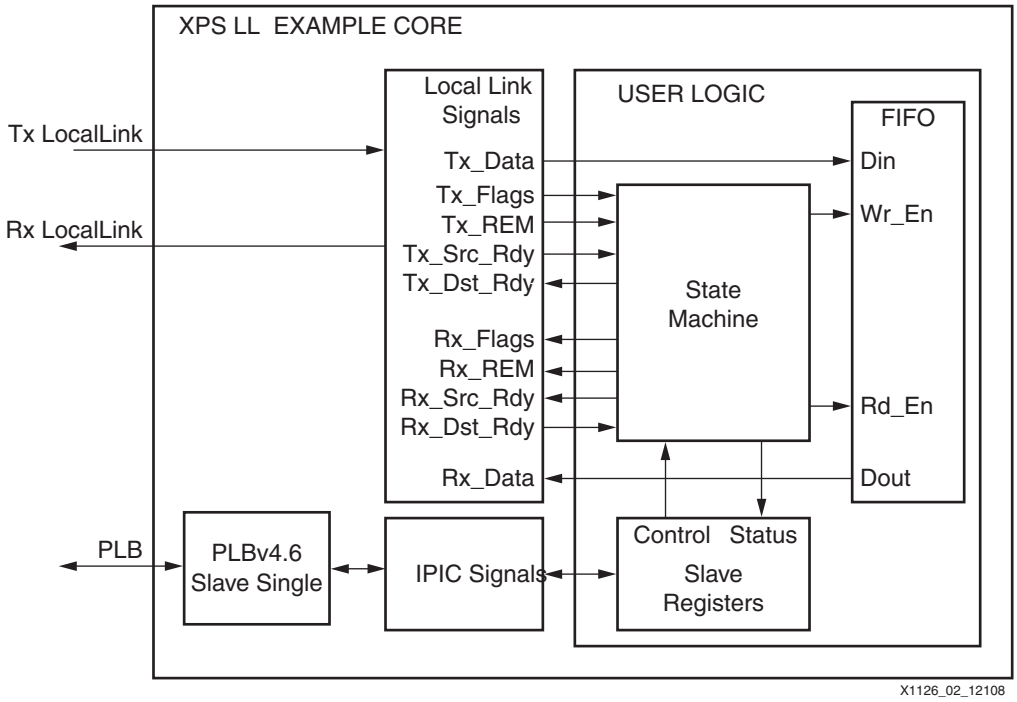


Figure 2: **XPS LL EXAMPLE User Logic Block Diagram**

LocalLink Interface

The HDMA block contains both a Tx channel and Rx channel for full-duplex mode of operation. The HDMA block is controlled by Device Control Registers (DCRs) in the processor block. In the user logic, both the Tx channel and Rx channel are used for the LocalLink loopback operation on the first HDMA block.

Tx LocalLink Interface

The Tx channel LocalLink interface is a subset of the Xilinx LocalLink Specification. The Tx channel is a point to point interface with a 32-bit data width. The following signals are used for the Tx channel. For more information, see the UG200 Embedded Processor Block Reference Guide.

Note: For the Tx channel, the HDMA is the source and the user logic is the destination.

Table 2: **LocalLink Tx Signals**

Signals	I/O	Bits	Description
tx_dst_rdy_n	I	1	Asserted Low where user logic is ready to accept data.
tx_src_rdy_n	O	1	Asserted Low indicating source is ready to transmit.
tx_data	O	32	Transmit data.

Table 2: **LocalLink Tx Signals *(Cont'd)***

Signals	I/O	Bits	Description
tx_rem	O	4	Asserted Low where each bit represents 8-bits of the 32-bits of data that is valid.
tx_sof_n	O	1	Asserted Low indicating start of frame.
tx_eof_n	O	1	Asserted Low indicating end of frame.
tx_sop_n	O	1	Asserted Low indicating start of packet.
tx_eop_n	O	1	Asserted Low indicating end of packet.

A Tx frame consists of a header, payload, and footer.

For the header, payload, or footer to be transferred from the Tx channel to the user logic, both tx_dst_rdy_n and tx_src_rdy_n are asserted Low (ready to transmit and accept data).

The header is eight 32-bits of data. The header starts transmitting when tx_sof_n is asserted Low from the DMA engine. The header consists of Buffer Descriptor (BD) fields. This includes the next destination pointer address (32-bits), current buffer address (32-bits), current buffer length (32-bits), control/status flags (32-bits) and four 32-bits of application data which can be used based upon the design. The XPS LL EXAMPLE does not use the four 32-bits of application data.

The payload starts transmitting to the user logic when tx_sop_n is asserted Low from the DMA engine. The payload continues to transmit until tx_eop_n is asserted Low. At this time, the number of transmits should be equal to the current buffer length in the BD. With every tx_data data beat before tx_eop_n, the tx_rem signal is all zeros (0000) which indicates 4 bytes (32-bits) is transferred. However, if the last data beat is an incomplete 32-bits (8/16/24-bits) when tx_eop_n is asserted Low, the appropriate tx_rem bit(s) is asserted Low depending on the bytes transferred. Table 3 shows the tx_rem bit(s) associated with the amount of bits transferred.

Table 3: **Tx REM Relation to Bits and Bytes Transferred**

tx_rem	Number of bits	Number of Bytes
0000	32-bits	4-bytes
0001	24-bits	3-bytes
0011	16-bits	2-bytes
0111	8-bits	1-byte

Because the footer is not relevant for Tx, The footer is ignored which occurs when tx_eof_n is asserted Low.

In the user logic for the XPS LL EXAMPLE core, the tx_dst_rdy_n signal is asserted high (source can not accept Tx data) when the FIFO is full from the payload data or when the user logic transmits the header, payload or footer on the Rx interface to the HDMA. The tx_dsy_rdy_n signal is asserted Low (source can accept Tx data) when the header, payload (FIFO is getting filled) or footer is transmitted from the Tx interface to the user logic.

Rx LocalLink Channel

The Rx channel LocalLink interface is a subset of the Xilinx LocalLink Specification. The Rx channel is a point to point interface with a 32-bit data width. The signals litsed in Table 4 are used for the Rx channel. For more information, see the UG200 Embedded Processor Block Reference Guide.

Note: For the Rx channel, the user logic is the source and the HDMA is the destination.

Table 4: **LocalLink Rx Signals**

Signals	I/O	Bits	Description
rx_dst_rdy_n	O	1	Asserted Low where HDMA is ready to accept data.
rx_src_rdy_n	I	1	Asserted Low indicating user logic is ready to transmit.
rx_data	I	32	Receive data.
rx_rem	I	4	Asserted Low where each bit represents 8-bits of the 32-bits of data that is valid.
rx_sof_n	I	1	Asserted Low where user logic indicates start of frame.
rx_eof_n	I	1	Asserted Low where user logic indicates end of frame.
rx_sop_n	I	1	Asserted Low where user logic indicates start of packet.
rx_eop_n	I	1	Asserted Low where user logic indicates end of packet.

A Rx frame consists of a header, payload and footer.

For the header, payload or footer to be transferred from the user logic to the Rx channel, both rx_dst_rdy_n and rx_src_rdy_n must be asserted Low.

Because the header is not relevant for Rx, the header is ignored which occurs when rx_sof_n is asserted Low.

The payload starts transmitting to the HDMA when rx_sop_n is asserted Low by the user logic. The payload continues to transmit until rx_eop_n is asserted Low which is controlled by the user logic. With every data beat of the rx_data payload before rx_eop_n is asserted Low, the rx_rem signal is all zeros (0000) which indicates that 4 bytes is transferred (32-bits). However, if the last data beat is not a complete 32-bits (8/16/24-bits) during rx_eop_n, the user logic asserts Low the appropriate rx_rem bit(s) based upon which bytes need to be transferred. See Table 4.

In the user logic for the XPS LL EXAMPLE core the rx_src_rdy_n signal is asserted high (source is not ready to transmit Rx data to the HDMA) when the Tx interface transmits the header, payload (FIFO is getting filled) or the footer to the user logic. The rx_dsy_rdy_n signal is asserted Low (source is ready to transmit Rx data to the HDMA) when the header, payload (FIFO is emptied), nad footer are being transmitted on the Rx interface from the user logic to the HDMA.

Creating XPS LL EXAMPLE Core and HDL

Create IP Wizard

Before adding the LocalLink interface and logic, the XPS LL EXAMPLE core is created with Create IP Wizard. The initial core is set up for a slave PLB v4.6 interface and 8 slave registers and a soft reset. Table 5 shows the registers in the XPS LL EXAMPLE core.

Table 5: **XPS LL EXAMPLE Slave Registers**

Register Name	(offset from C_BASEADDR)	Access	Size in Bits	Description
Control Register	0x0	Read/Write	32	Bit 31 - Invert Payload Data Bit
Status Register	0x4	Read	32	Bit 31 - Done Bit Bit 30 - Busy Bit
Total Tx Frames Sent	0x8	Read	32	32-bit counter value.
Total Rx Packets Received	0xC	Read	32	32-bit counter value.

Table 5: **XPS LL EXAMPLE Slave Registers *(Cont'd)***

Register Name	(offset from C_BASEADDR)	Access	Size in Bits	Description
Total Tx Bytes Sent	0x10	Read	32	32-bit counter value.
Total Rx Bytes Received	0x14	Read	32	32-bit counter value.
Software Reset	0x100	Write	32	Reset core by writing 0xA.

FIFO Generator

For the loopback functionality for the payload, a FIFO is created using the Xilinx CORE Generator™. The FIFO is set up for Virtex®-5 FPGA block RAM, the Write Width and Read Width is 32 because the LocalLink interface is 32-bits and the Write Depth and Read Depth is 32. There is no specific reason why the FIFO depth is set to 32.

FIFO Generator creates several files that need to be included with the EDK core. With the XPS LL EXAMPLE, the `payload_fifo.vhd` file is necessary which contains the FIFO entity. The `payload_fifo.ngc` file is necessary for the implemented Netlist file.

The FIFO is instantiated in the user logic as the payload_fifo instance. The following ports are used in the user logic design. The user logic with these ports are described in the next section.

Table 6: **Payload_fifo Ports**

Signal	I/O	Description
CLK	I	Connects to LocalLink clock.
SRST	I	Connects to core's reset.
DIN	I	Tx_data from the HDMA to the user logic.
WR_EN	I	Asserted high, write tx_data (payload data) into FIFO when Tx source and destination are ready to accept.
RD_EN	I	Asserted high, read from FIFO.
DOUT	O	Rx_data (payload data) is valid one clock cycle after RD_EN is asserted. User logic transmits to HDMA when Rx source and destination are ready to accept.
FULL	O	Asserted high, user logic empties the FIFO until empty.
EMPTY	O	Asserted high, user logic fills the FIFO until full or Tx EOP.

Designing User Logic with a LocalLink Interface

This section covers the general design of the XPS LL EXAMPLE core. Refer to the HDL for the core for more details in the design.

Adding LocalLink Interface into the HDL

The entity of `xps_ll_example.vhd` file is updated with the signals in Table 2 and Table 4 which are added with the direction of the signal and the bit length(s). An addition signal for the LocalLink reset (ll_rst) is added to the entity as well. In addition, in the user logic instance, the signals in Table 2 and Table 4 are to be mapped to each other for the user logic component.

The `user_logic.vhd` file's entity is updated with the signals in Table 2 and Table 4 are added with the direction of the signal and the bit length(s). An additional signal for the LocalLink reset (ll_rst) is added to the entity. This is the same procedure as the `xps_ll_example.vhd` file.

State Machine

The state machine contains eight states. The eight states are IDLE, HEADER, WRITE_FIFO, TX_EOF, RX_SOF, RX_SOP, READ_FIFO and FOOTER.

Figure 3 shows the state machine used for the XPS LL EXAMPLE core. The description of these states is described in the subsequent sections.

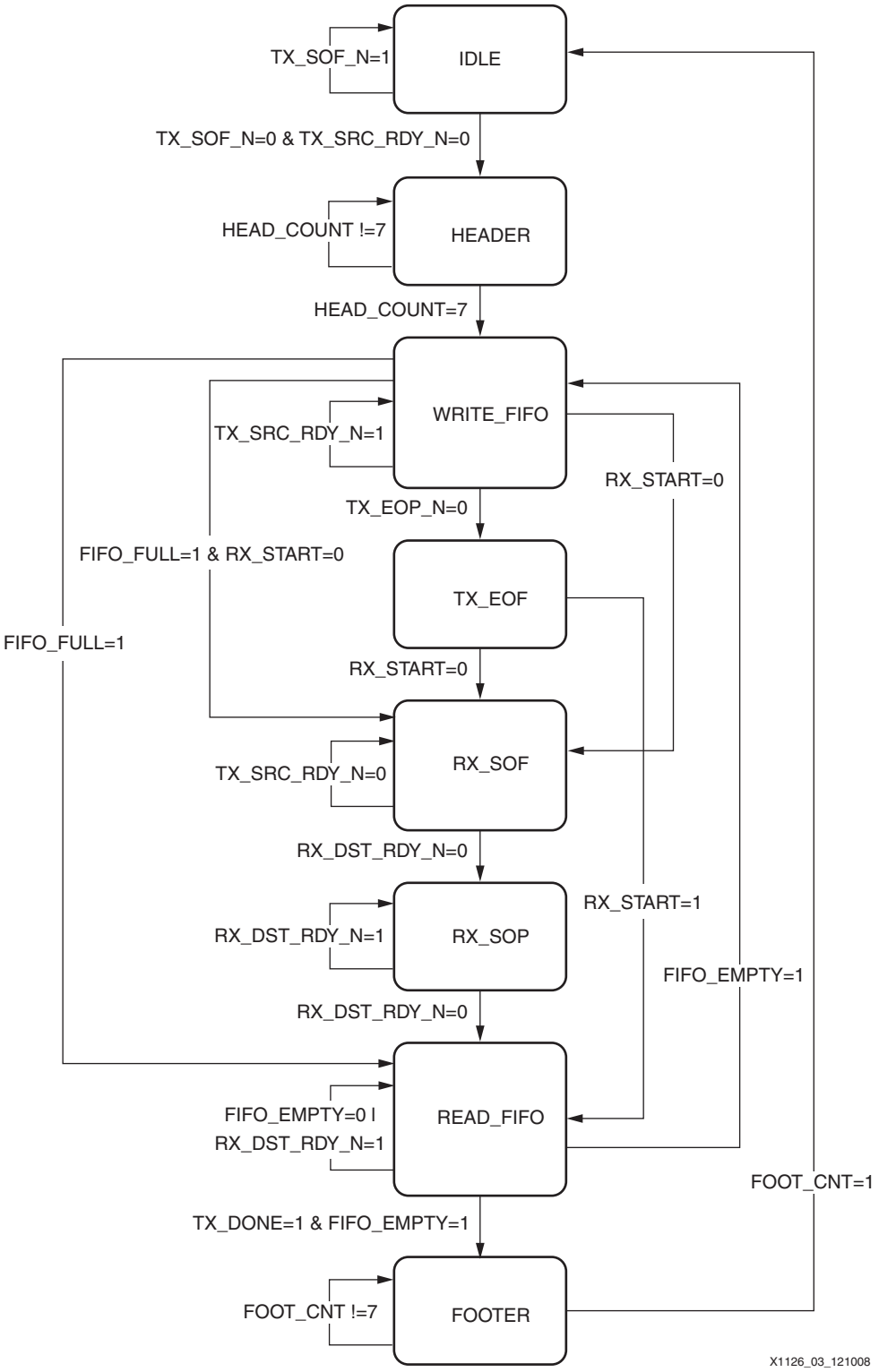


Figure 3: **XPS LL EXAMPLE State Machine**

Idle State

The IDLE state resets most registers in the design. See the Processes section on page 9. In this state, the tx_dst_rdy_n is asserted Low (ready to accept Tx data) and the rx_src_rdy_n is asserted high (source is not ready to send Rx data). The state machine transitions to the header state when the tx_sof_n and tx_src_rdy_n are both asserted Low from the HDMA. If this condition is not met, the state machine transitions back to the IDLE state.

HEADER State

The header state waits for the eight 32-bits of the header to be transmitted to the user logic. Once this condition is met through the header counter, the state machine transitions to the WRITE_FIFO state. If the header is not completely transmitted, the state machine goes back to the header state.

WRITE_FIFO State

The WRITE_FIFO state writes the Tx payload data with the assertion of the write enable signal to the payload FIFO until the FIFO is filled or tx_eop_n is asserted Low (Tx is done transmitting the payload).

Once the FIFO is full or tx_eop_n is asserted Low, the state machine transitions to the RX_SOF state if the Rx channel has not transmitted the Rx header or to the READ_FIFO state for the FIFO to be emptied (user logic transmits payload FIFO data to the HDMA by the Rx channel).

Before transitioning to the READ_FIFO state, the read enable signal is asserted to the payload FIFO for the payload data to be available in the READ_FIFO state.

If the source is not ready to transmit to the user logic (tx_src_rdy_n is asserted high), the state machine transitions back to the WRITE_FIFO state waiting for the source to be ready to transmit.

TX_EOF State

The TX_EOF state alLows for the user logic to accept the footer of the Tx frame. The state machine transitions to the RX_SOF state if the header of the Rx channel has not been transmitted.

If the Rx header has been transmitted, the state machine transitions to the READ_FIFO state to empty the payload FIFO on the Rx channel and a read from the FIFO occurs because it takes one clock cycle for the data to be available on DOUT of the FIFO.

RX_SOF State

The RX_SOF state transmits the header from the user logic to the HDMA by means of the Rx channel. Because the header is ignored, a constant value is used for the rx_data (`0xDEADBEEF`) and the rx_sof_n signal is asserted Low during this state. If the destination is not ready to accept the transaction, the state machine transitions back to the RX_SOF state.

If the destination is ready to accept the transaction, the state machine transitions to the RX_SOP state. Before the state machine goes to the RX_SOP state, a read from the FIFO occurs because it takes one clock cycle for the data to be available on DOUT of the FIFO (read FIFO data is available for the RX_SOP state).

RX_SOP State

The RX_SOP state transmits the first 32-bits of the payload from the user logic to the HDMA by means of the Rx channel. Because this is the first 32-bits of the payload, the rx_sop_n signal is asserted Low during this state.

If the destination is ready to accept the first 32-bits of the payload, the read enable signal is asserted to the payload FIFO for payload data to be available in the READ_FIFO state. If the

destination is not ready to accept the first 32-bits of data, the state machine transitions back to the RX_SOP waiting for the destination to accept the transaction.

READ_FIFO State

The READ_FIFO state transmits payload data after rx_sop_n is asserted Low.

If the FIFO is not empty, the read enable for the payload FIFO is asserted and the state machine transitions back to the READ_FIFO state.

If the Tx channel has transmitted tx_eof_n and the FIFO is empty, the state machine transitions to the FOOTER state.

FOOTER State

The FOOTER state transmits the footer from the user logic to the HDMA. A constant value (0xDEADBEEF) is set for the footer except for the last 32-bits of the footer (the eighth word).

After the eighth word which is determined by the footer counter, the state machine transitions back to the IDLE state waiting for the next Tx frame.

Processes

Most of the processes are for counters and registering events in the state machine.

The TX_EOP register is set after tx_eop_n is asserted Low. This register is used in the state machine to signal the READ_FIFO state to assert Low the rx_eop_n signal after emptying the FIFO (transition to the FOOTER state). The reset is enabled in the IDLE state.

The rx_start_bit register is set after the RX_SOF and RX_SOP states indicating that Rx SOF and Rx SOP flags have been sent from the user logic for the current frame to the HDMA. The reset is enabled in the IDLE state.

The rx_rem register is set with the tx_rem value when tx_eop_n is asserted Low. When the state machine asserts Low the rx_eop_n signal, the data in rx_rem register is used for the rx_rem data to the HDMA.

The header counter counts from 0-7. The counter counts when both the tx_src_rdy_n and tx_dst_rdy_n signals are both asserted Low when the state machine is in the header state. The header counter is reset in the IDLE state.

The footer counter counts from 0-7. The counter counts when both the rx_src_rdy_n and rx_dst_rdy_n signals are both asserted Low in the footer state. The footer counter's reset is enabled in the IDLE state.

The Tx and Rx packet counters count the total amount of frames after the reset. The counter counts when the tx_eof_n or rx_eof_n signals are asserted Low.

The Tx byte counter counts the total amount of bytes sent from the HDMA to the user logic after the reset. The Rx byte counter counts the total amount of bytes sent from the user logic to the HDMA after the reset.

The single Rx byte counter is used to count the total amount of bytes sent between rx_sop_n and rx_eop_n in the current frame. The counter's value is used for the rx_data on the last 32-bits of the footer. The data is stored in APP3 field of the BD. This allows the software or a software driver to read the number of bytes sent to the HDMA from the user logic. The reset is enabled in the IDLE state.

Flags

For the Rx channel, the user logic needs to assert Low the rx_sof_n, rx_sop_n, rx_eop_n and rx_eof_n signals at the different times. Combinatorial logic sets these flags dependent on states and conditions in the state machine.

The rx_sof_n is asserted Low in the RX_SOF state and high in any other state.

The rx_sop_n is asserted Low in the RX_SOF state and high in any other state.

The rx_eop_n is asserted Low in the READ_FIFO state when the FIFO is empty and Tx channel is done transmitting to the user logic. The signal is high in any other condition or in any other state.

The rx_eof_n is asserted Low in the FOOTER state when the eighth 32-bits of the footer is transmitted from the user logic to the HDMA. The signal is high in any other condition or in any other state.

Importing the XPS LL EXAMPLE Core using Create IP Wizard

Because a FIFO is instantiated and a LocalLink interface is added to the core, the modified core must be imported using Create IP Wizard.

1. Open Create IP Wizard and select **Import existing peripheral**. Click on **Next**.
2. Select a valid EDK user repository. Click on Next.
3. The name of the core is the same as before **xps_ll_example**. Select **Use version:** and select **Major revision:** to **1** and **Minor revision:** to **00** and **Hardware/Software compatibility revision** to **a**. Click on Next.
4. In the Source File Types window, under **Indicate the types of files that make up your peripheral.** select **HDL source files** and **Netlist files**. Click on Next.
5. In the HDL Source Files window, the **HDL language used to implement your peripheral** is **VHDL** and select **Use existing Peripheral Analysis Order file (*.pao)**. Click on the **Browse** button and select the `xps_ll_example.pao` file in the data directory of core. Click on Next.
6. In the HDL Analysis Information window, select the **Add Files** button. Add `payload_fifo.vhd` (file created by FIFO generator). Click on Next.
7. For the Bus interface window, select **PLBV46 Slave (SPLB)** under Processor Local Bus (version 4.6) interface. Click on Next.
8. For the SPLB: Port window, select Next for bus interface which are the defaults.
9. For the SPLB: Parameter window, select **Register Space**. Click on Next.
10. For the Identify Interrupt Signals window, deselect **Select and configure interrupt(s)**. Click on Next.
11. In the Parameter Attributes window, select **Next** for parameter defaults.
12. Click Next in the Port Attributes window. Because Create IP Wizard does not support the LocalLink interface, the user logic ports in the MPD will be modified to connect to a LocalLink target like the HDMA.
13. In the Netlist Files window, click on **Select Files...** and browse to the `payload_fifo.ngc` (file created by Fifo Generator). Click on Next.
14. Click on Finish to import the core.

Open the `xps_ll_example_v2_1_0.mpd` file location in the data directory of the imported core.

Under the Bus Interfaces section, add the following bus interface for LocalLink as shown in Figure 4.

```
## Bus Interfaces
BUS_INTERFACE BUS = SPLB, BUS_TYPE = SLAVE, BUS_STD = PLBV46
BUS_INTERFACE BUS = LLINK0, BUS_STD = XIL_LL_DMA, BUS_TYPE = INITIATOR
```

X1126_04_121008

Figure 4: **Adding Bus Interface to MPD**

For each LocalLink port, add the bus switch attribute which is equal to the LocalLink bus interface as shown in Figure 5.

```
PORT Ll_Clk = "", DIR = I, SIGIS = CLK
PORT Ll_Rst = "", DIR = I, SIGIS = RST
PORT tx_data = "", DIR = I, VEC = [0:31], BUS = LLINK0
PORT tx_rem = "", DIR = I, VEC = [0:3], BUS = LLINK0
PORT tx_sof_n = "", DIR = I, BUS = LLINK0
PORT tx_eof_n = "", DIR = I, BUS = LLINK0
PORT tx_sop_n = "", DIR = I, BUS = LLINK0
PORT tx_eop_n = "", DIR = I, BUS = LLINK0
PORT tx_src_rdy_n = "", DIR = I, BUS = LLINK0
PORT tx_dst_rdy_n = "", DIR = O, BUS = LLINK0
PORT rx_data = "", DIR = O, VEC = [0:31], BUS = LLINK0
PORT rx_rem = "", DIR = O, VEC = [0:3], BUS = LLINK0
PORT rx_sof_n = "", DIR = O, BUS = LLINK0
PORT rx_eof_n = "", DIR = O, BUS = LLINK0
PORT rx_sop_n = "", DIR = O, BUS = LLINK0
PORT rx_eop_n = "", DIR = O, BUS = LLINK0
PORT rx_src_rdy_n = "", DIR = O, BUS = LLINK0
PORT rx_dst_rdy_n = "", DIR = I, BUS = LLINK0
```

X1126_05_121008

Figure 5: **Adding Bus Interface Attribute to LocalLink Ports**

Software Application

LL Example

This software application demonstrates a LocalLink transfer in which the transmit data is looped back to the receive channel by means of the XPS LL EXAMPLE core connected to the LocalLink interface.

The packet size is 1 Kb. With this software application, the number of BDs (Buffer Descriptor) is set to 128 for both channels based upon the base and high address of the Tx and Rx BDs. Each BD takes 0x40 of space which contains the next pointer information, current buffer address, current buffer length, status and control, and application specific data. Figure 6 shows a single BD.

NXTDESC_PTR
CURBUF_ADDR
CURBUF_LENGTH
STS_CRTL_APP0
APP1
APP2
APP3
APP4

X1126_06_121008

Figure 6: **Buffer Descriptor Diagram**

In this software application, the TX BD base and high addresses are `0x01000000` and `0x01001FFF`. The total address range is `0x1FFF`. Dividing the total address range by `0x40` leaves 128 BDs for the Tx side. The same procedure is for the Rx channel which also contains 128 BDs. In addition, regions of memory is allocated for the Tx and Rx buffers.

Currently the software application is not inverting the Rx payload data. By uncommenting the #define INVERT, the Rx payload data is inverted and is verified against the Tx payload after the DMA operation is complete.

In the main function, the DMA instance is started based upon the base address for the DMA engine (DCR address for HDMA).

In the TxSetup function calculates the number of BDs depending on the base and high address set for the Tx BDs and creates a ring. The fields for the first BD are cleared then every BD in the ring is copied with the same data for the fields as the first BD. Then the interrupt for the Tx channel is started and the TX ring is started.

In the RxSetup function, it calculates the number of BDs depending on the base and high address set for the Rx BDs and creates a ring. The fields for the first BD are cleared and then every BD in the ring is copied with the same data for the fields as the first BD. The Rx available BDs in the Rx ring are reserved. Every BD in the ring is assigned with the buffer address, SOP and EOP flags, length, and the buffer is assigned to every BD. The BDs are written to memory. The interrupts for the Rx channel is started and the Rx ring is started. After this process, the Rx channel is ready to receive packets.

In the SetupIntrSystem, the interrupt system is set up for the Rx and Tx channels. Once an interrupt occurs on the Rx channel, the interrupt routine ensures that no error occurred, verifies that the data is transferred correctly (compare Tx and Rx buffers), and releases the BD back into memory. Once an interrupt occurs on the Tx channel, the interrupt routine ensures that no error occurred during transmit and then releases the BDs back into memory.

If INVERT is defined in the code, the software driver is called to set the invert bit in the Control Register before the DMA operation.

In the SendPacket function, the packet is created with a simple pattern. In the Tx ring, one Tx BD is reserved in memory. The BD is assigned with the Tx buffer address, SOP and EOP flags, length, and the packet. The BD is written to memory which starts the DMA transfer.

After the DMA operation is complete, the frames sent, the frames received, the bytes sent, and the bytes received, are reported on the Hyperterminal screen.

Included with the XPS LL EXAMPLE is a software driver. The driver contains all the functions that are generated with Create IP Wizard. In addition, functions are added for the LocalLink user logic, such as setting the control register, polling the status register until DMA operations are complete, and displaying register values using print functions for Tx bytes and frames sent and Rx bytes/frames received. The driver is included in the reference system.

Reference System Specifics

Adding the XPS LL EXAMPLE core to the system is discussed (which is already done in the reference system). This includes setting the PPC440 processor block for a single HDMA block with interrupts.

In addition, analyzing the behavior of the LocalLink transaction based upon the software application in the ChipScope analyzer is discussed.

Adding HDMA to the PPC440 Processor Block

PPC440 Processor Block Parameters

In the System Assembly View, right click on the **ppc440_0** instance, then select **Configure IP**. Select the DMA tab. For the **Number of DMA Channel** select **1** as shown in Figure 7.

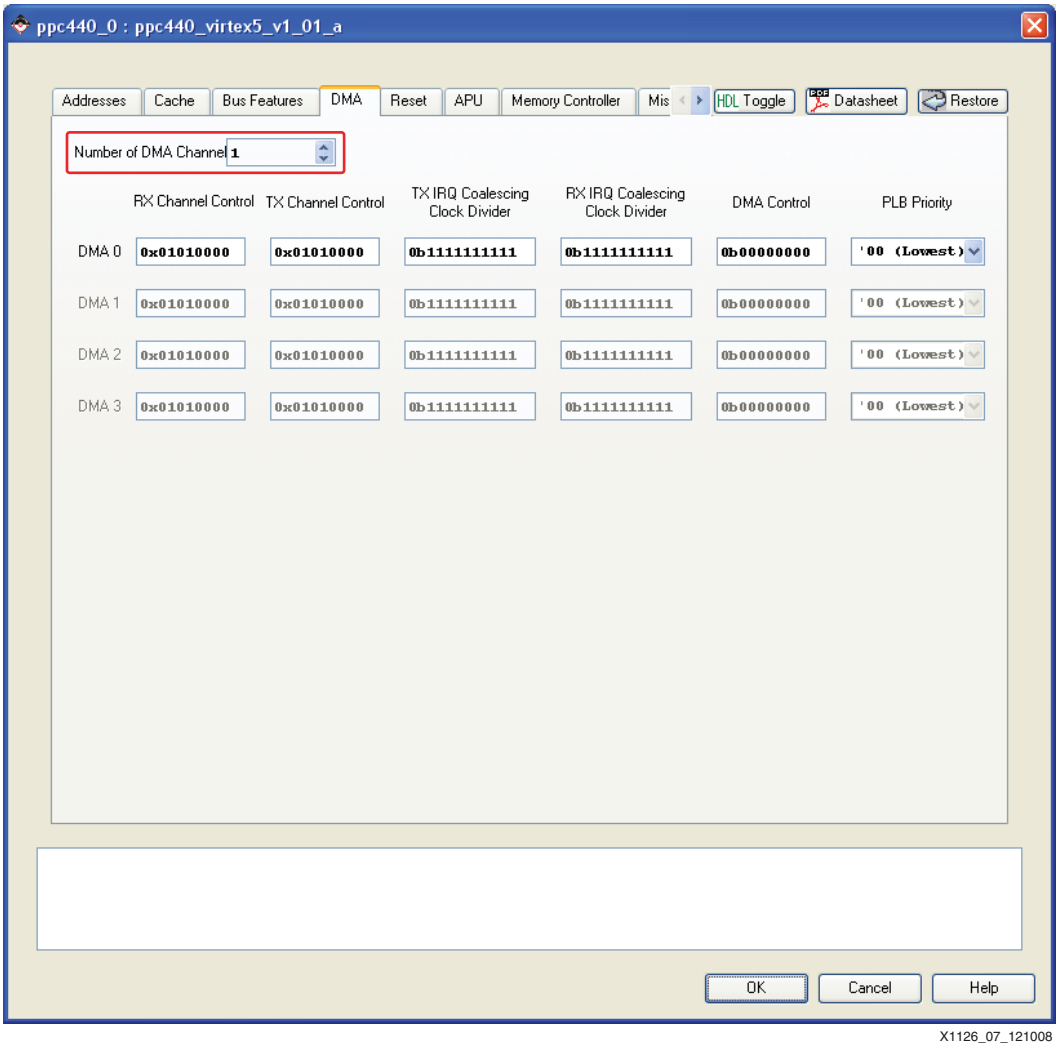
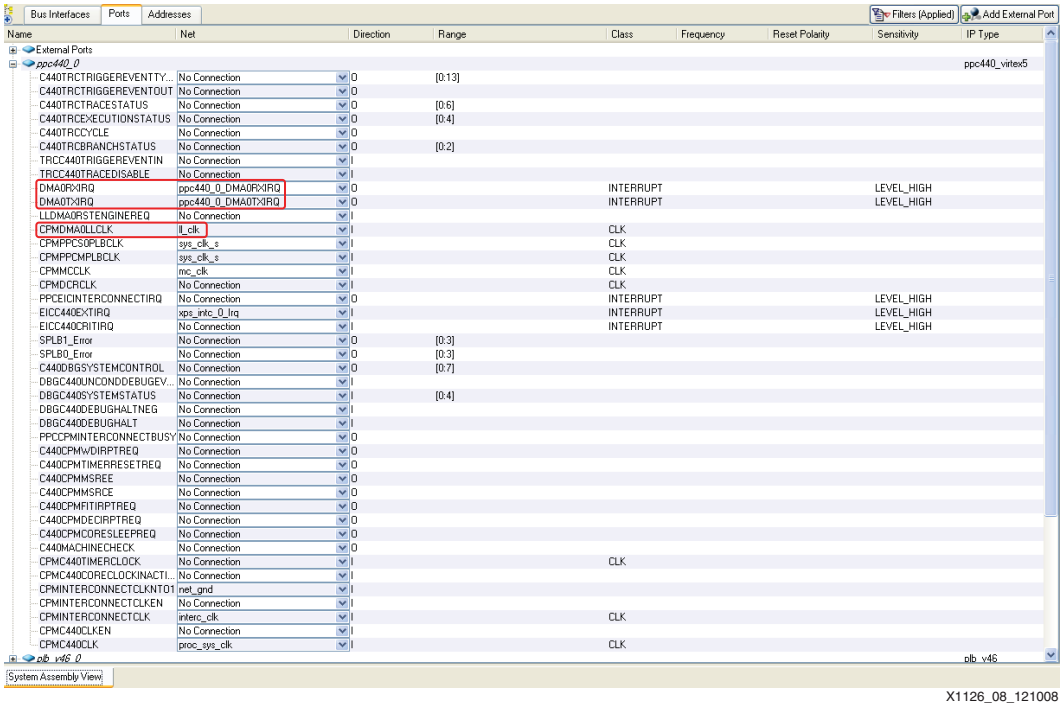


Figure 7: **PPC440 Processor Block Parameters**

PPC440 Processor Block Ports

In the System Assembly View/Ports tab, expand the tree node for ppc440_0.

For **DMA0RXIRQ** and **DMA0TXIRQ** select **New Connection**. In addition, for the LocalLink clock **CPMDMA0LLCLK** select **ll_clk** as shown in Figure 8.



X1126_08_121008

Figure 8: **PPC440 Processor Block Ports**

In the System Assembly View/Ports tab, expand the tree node for the **xps_intc_0** instance and double click on the **Net** column for the **Intr** port. This will bring up the Interrupt Connection Dialog Box. Select **ppc440_0_DMA0RXIRQ** and select the **Plus** button. Select **ppc440_0_DMA0TXIRQ** and select the **Plus** button as shown in Figure 9. Select **OK**.

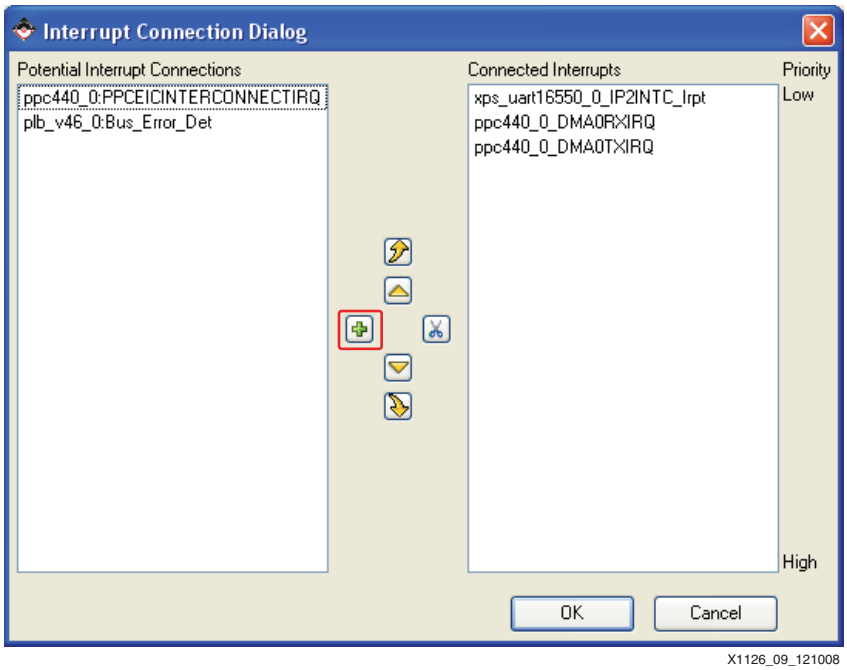


Figure 9: **Interrupt Connection Dialog**

Adding The XPS LL EXAMPLE core

The XPS LL EXAMPLE core is located in the pcores/ area of the project.

Add the XPS LL EXAMPLE core by expanding the USER tree node in the IP Catalog tab in XPS. Right select the **XPS_LL_EXAMPLE** core, then select **Add IP**.

Connecting LocalLink and PLB v4.6 Slave Bus Interfaces

In the Bus Interface tabl of the System Assembly View/Bus Interface tab, expand the ppc440_0 and xps_ll_example_0 tree nodes.

In the ppc440_0 tree node, for the **LLDMA0** bus interface, select **xps_ll_example_0_LLINK0**.

In the xps_ll_example_0 tree node, for the **SPLB** bus interface, select **plb_v46_0**.

The LocalLink and PLB v4.6 connections are shown in Figure 10.

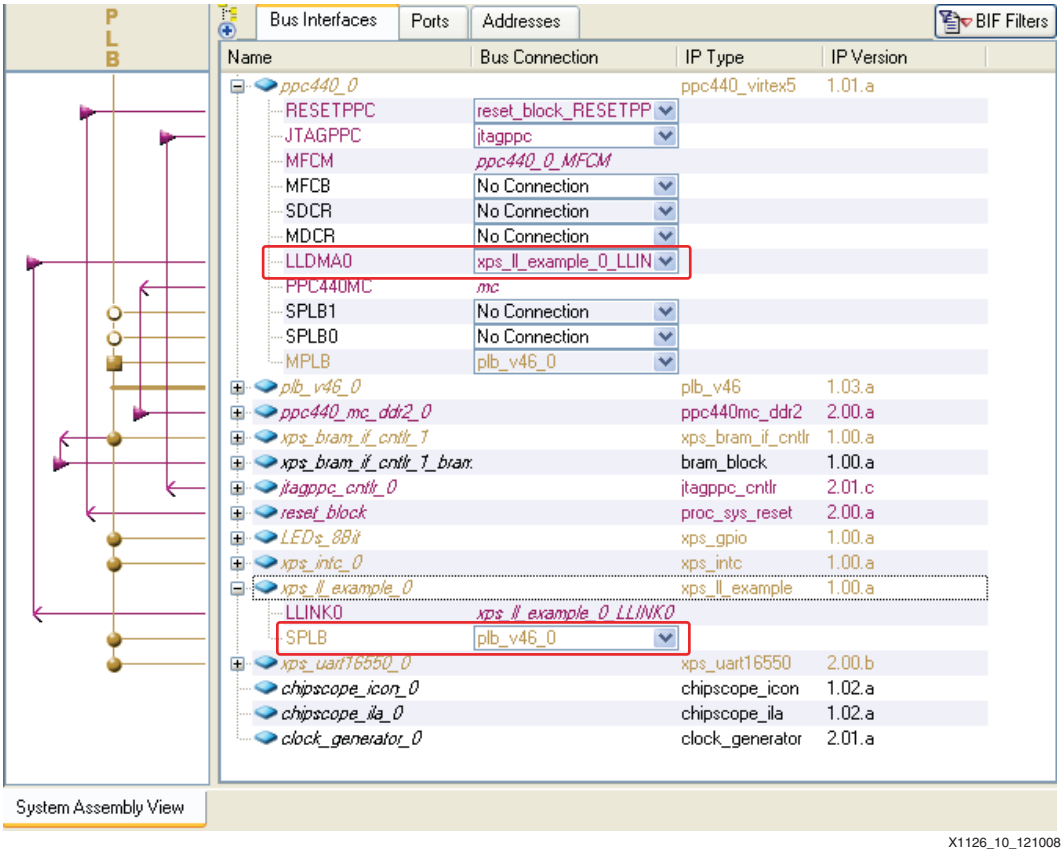


Figure 10: **LocalLink and PLB v4.6 Slave Bus Interfaces**

Setting Addresses

Right click on the xps_ll_example_0 instance and select **Configure IP**. Set the **C_BASEADDR** and **C_HIGHADDR** to **0x60000000** and **0x6000FFFF** respectively. The address range is for the slave PLB v4.6 registers.

Connecting Ports

In the System Assembly View/Ports tab, expand the tree node for **xps_ll_example_0**. Connect **LI_Rst** to **ll_rst** and **LI_Clk** to **ll_clk** as shown in Figure 11.

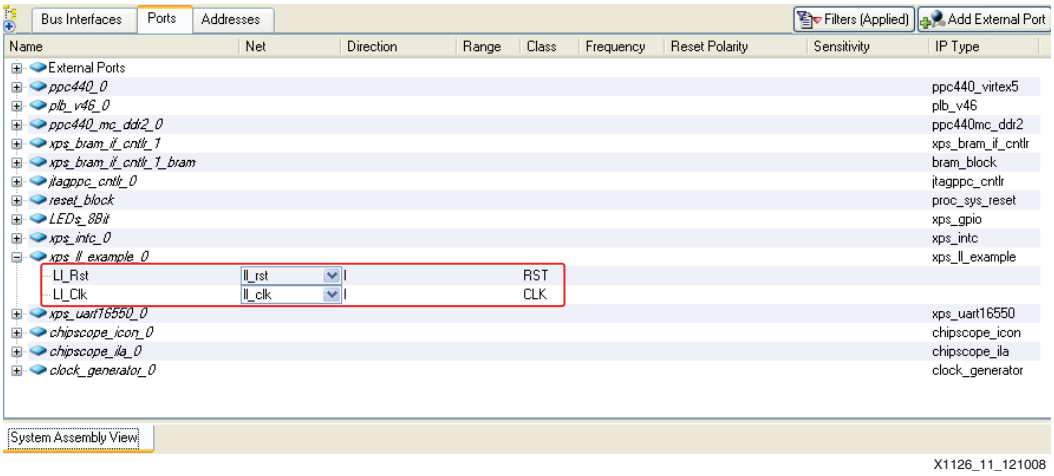


Figure 11: **XPS LL EXAMPLE Ports**

Executing the Reference System in Hardware

Using HyperTerminal or a similar serial communications utility, map the operation of the utility to the physical COM port to be used. Then connect the UART of the board to this COM port. Set the HyperTerminal to the Bits per second to **9600**, Data Bits to **8**, Parity to **None,** and Flow Control to **None**. The settings are shown in Figure 12. This is necessary to see the results from the software application.

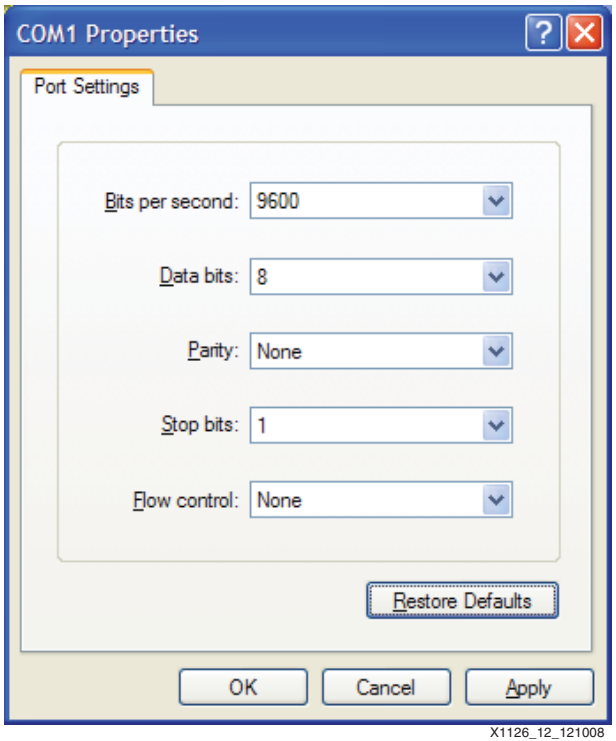


Figure 12: **HyperTerminal Settings**

Executing the Reference System using the Pre-Built Bitstream and the Compiled Software Applications

To execute the system using files in the `ready_for_download/` directory in the project root directory, follow these steps:

1. Change directories to the `ready_for_download` directory.
2. Use iMPACT to download the bitstream by using the following command:

   ```
   impact -batch xapp1126.cmd
   ```

3. Invoke XMD and connect to the PowerPC 440 processor by using the following command:

   ```
   xmd -opt xapp1126.opt
   ```

4. Download the executable by using the following command depending on the software application:
   - **`dow ll_example.elf`**
5. Enter in the `run` command to run the software application.

Executing the Reference System from XPS for Hardware

To execute the system for hardware using XPS, follow these steps:

1. Open `system.xmp` in XPS.
2. Select **Hardware→Generate Bitstream** to generate a bitstream for the system.
3. Select **Software→Build All User Application** to compile the software applications**.**
4. Select **Device Configuration→Download Bitstream** to download the bitstream.
5. Select **Debug→Launch XMD to i**nvoke XMD.
6. Download the executable file by using the following command depending on the software application:
   - **`dow ll_example/ll_example.elf`**
7. Enter in the `run` command to run the software application.

Running the Software Applications

Running the LL EXAMPLE Software Application

After running the software application, the following output should be seen.

```
Starting DMA Transfer:

Tx Packets Sent:1
Rx Packets Received:1
Tx Bytes Sent:1024
Rx Bytes Received:1024

Test Completed!
```

Analyzing Single Tx and Rx Frame During Loopback

In this section, the header, payload and footer of both the Tx and Rx interface are examined. In addition, the loopback functionality is verified with the ChipScope tool results.

Starting the ChipScope Tool

1. Open the ChipScope Analyzer.
2. Click on the Open Cable / Search JTAG Chain button and then click **OK** on the ChipScope Pro Analyzer dialog box.

3. Select **File→Open Project**, select **No** to save the new project, then select the `ppc440_ll_example.cpj` in the cpj directory of the project.

   In the ChipScope tool project, UNIT:0 MyILA0 ChipScope core is associated with the LocalLink interface signals.

4. For UNIT:0 MyILA0, set the trigger by **Trigger Setup→Run**. The trigger is set up when the LLINK0_LL_Tx_SOF_n signal is asserted Low.

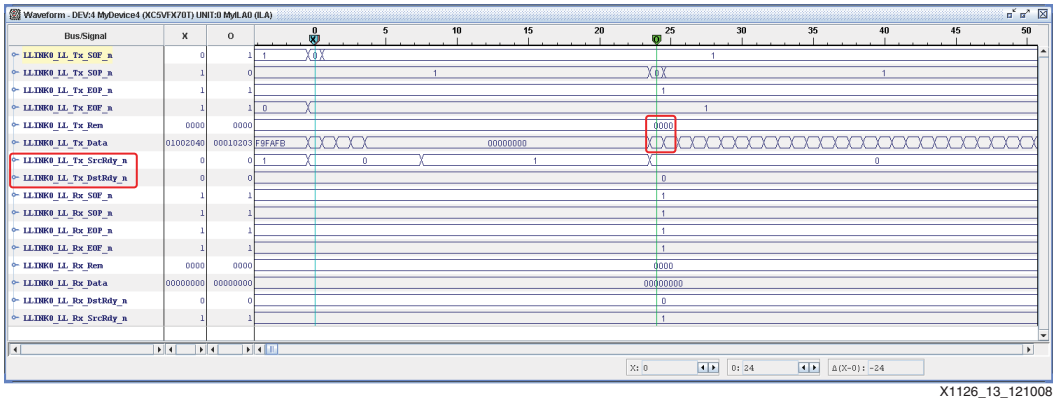
5. Because the software application has been compiled, in XMD, download ll_example by the following command and run the executable:

   - **`dow ll_example/ll_example.elf`**

6. Enter in the `run` command to run the software application.

Analyzing Tx and Rx SOF SOP Flags

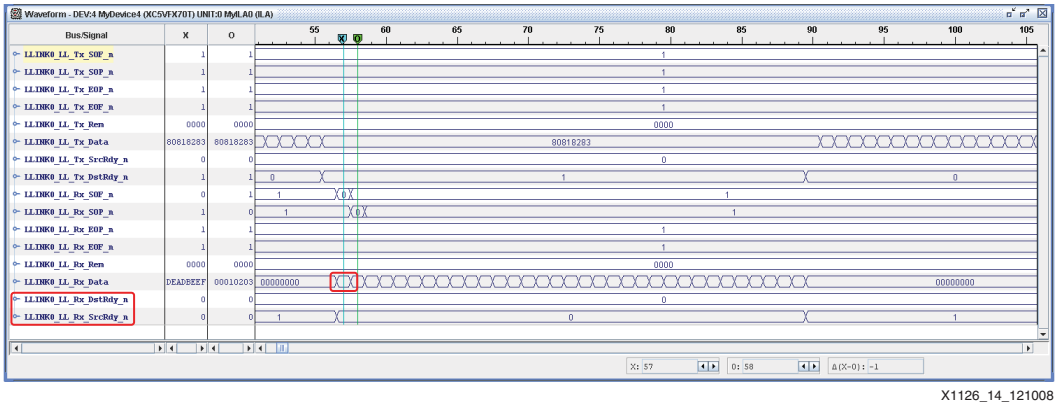
1. For Unit:0 MyILA0/Waveform window, zoom in where LLINK0_LL_Tx_SOF_n and LLINK0_LL_Tx_SOP_n are asserted Low. When the flags are both asserted Low, notice that the LLINK0_LL_Tx_SrcRdy_n and LLINK0_LL_Tx_DstRdy_n signals are asserted Low (Tx transfer is occurring). In addition, notice the LLINK0_LL_Tx_data when LLINK0_LL_Tx_SOP_n is asserted Low which is the value `0x00010203`.



X1126_13_121008

Figure 13: **Tx SOF and Tx SOP Signals**

2. For Unit:0 MyILA0/Waveform window, zoom in where the LLINK0_LL_Rx_SOF_n and LLINK0_LL_Rx_SOP_n signals are asserted Low. When the flags are both asserted Low, notice that the LLINK0_LL_Rx_SrcRdy_n and LLINK0_LL_Rx_DstRdy_n signals are both asserted Low. In addition, notice the LLINK0_LL_Rx_Data when the LLINK0_LL_Rx_SOP_n signal is asserted Low which is the value `0x00010203`. The Rx data is the same as the start of payload for Tx data.



X1126_14_121008

Figure 14: **Rx SOF and Rx SOP Signals**

Analyzing Tx and Rx SOF SOP Flags

1. For Unit:0 MyILA0/Waveform window, zoom in where the LLINK0_LL_Tx_EOP_n and LLINK0_LL_Tx_EOF_n signals are asserted Low. Notice the LLINK0_LL_Tx_Data when LLINK0_LL_Tx_EOP_n is asserted Low which is the value `0xFCFDFEFF`.

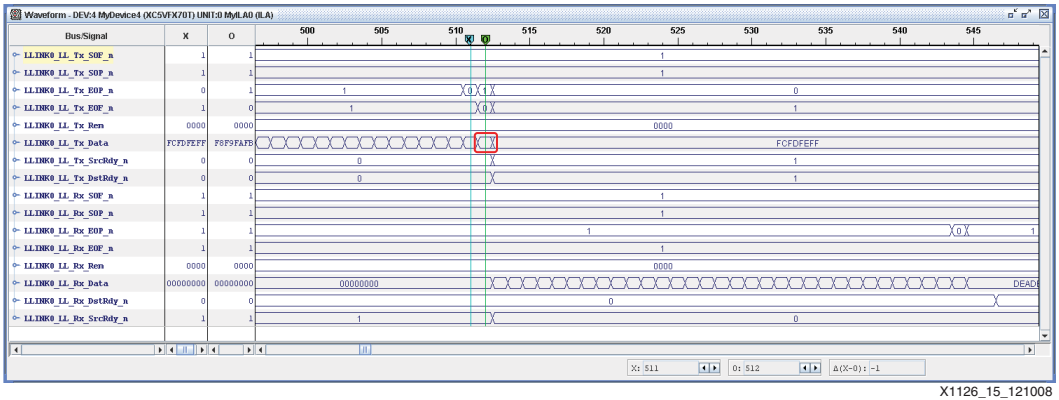


Figure 15: **Tx EOP and Tx EOF Signals**

2. For Unit:0 MyILA0/Waveform window, zoom in where the LLINK0_LL_Rx_EOP_n and LLINK0_LL_Rx_EOF_n signals are asserted Low. The LLINK0_LL_Rx_Data is `0xFCFDFEFF` when the LLINK0_LL_Rx_EOP_n signal is asserted Low. Note when LLINK0_LL_Rx_EOF_n is asserted Low, that the LLINK0_LL_Rx_Data is `0x00000400` which is the value `1024` in decimal which represents the bytes sent from the user logic to the HDMA on the Rx channel. The value `1024` matches the amount of bytes transferred in the software application.

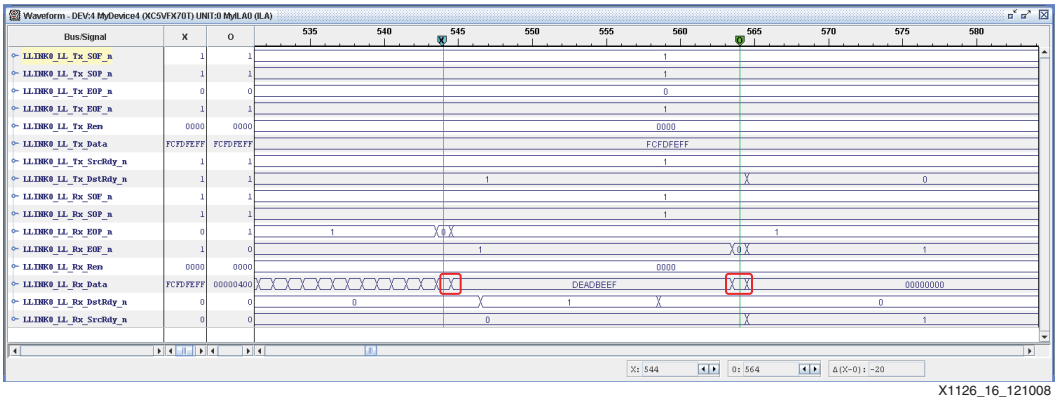


Figure 16: **Rx EOP and Rx EOF Signals**

References

1. UG200 *Embedded Processor Block in Virtex-5 FPGAs Reference Guide*
2. UG443 *PLB v3.4 and OPB to PLB v4.6 System and Core Migration User Guide*

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/10/08	1.0	Initial Xilinx release.

Notice of Disclaimer

Xilinx is disclosing this Application Note to you “AS-IS” with no warranty of any kind. This Application Note is one possible implementation of this feature, application, or standard, and is subject to change without further notice from Xilinx. You are responsible for obtaining any rights you may require in connection with your use or implementation of this Application Note. XILINX MAKES NO REPRESENTATIONS OR WARRANTIES, WHETHER EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL XILINX BE LIABLE FOR ANY LOSS OF DATA, LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES ARISING FROM YOUR USE OF THIS APPLICATION NOTE.