Product Not Recommended for New Designs

XAPP691 (v1.0.1) May 10, 2007

Parameterizable LocalLink FIFO

Author: Wen Ying Wei, Dai Huang

Summary

This application note describes the implementation of a parameterizable LocalLink FIFO, which is a First-In-First-Out memory queue with LocalLink interfaces on both sides. The LocalLink interface defines a set of protocol-agnostic signals that allows transmission of packet-oriented data, and enables a set of features such as flow control and transfer data of arbitrary length. The LocalLink FIFO consists of two LocalLink interfaces, one on the write port to interface with an upstream user application, the other on the read port to interface with a downstream user application. Its control logic interprets and generates LocalLink signaling, performs all read and write pointer management, and generates FIFO status signals for flow control purposes. The LocalLink FIFO uses fully synchronous and independent clock domains for the read and write ports. The LocalLink FIFO handles data width conversion between read and write ports. Its memory can be constructed in block SelectRAM™ memory or distributed RAM with parameterizable depth. The optional outputs of frame length and length ready provide visibility into byte numbers of each received frames, which allows downstream user application to acquire the length of a frame prior to reading the data.

Introduction

This application note describes the implementation of a LocalLink FIFO, which integrates features such as LocalLink interface, parameterizable FIFO data width and depth, data width conversion, and optional frame length count into an asynchronous FIFO. The block diagram in Figure 1 shows the two LocalLink interfaces, one on the write port to interface with an upstream user application, the other on the read port to interface with a downstream user application.

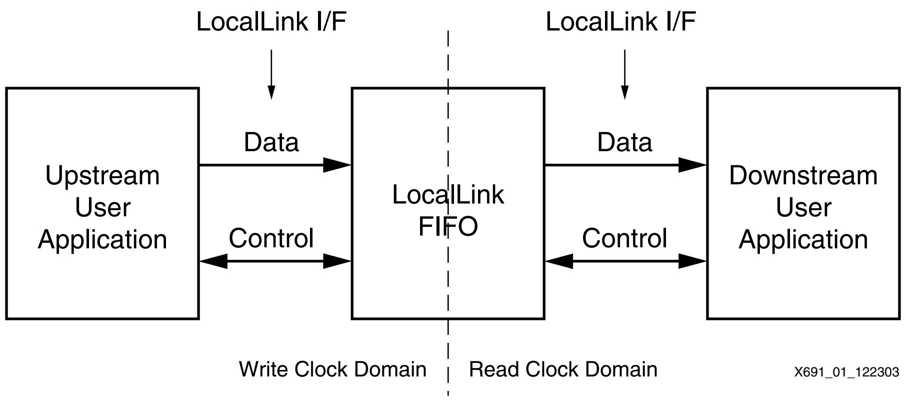

Figure 1: **LocalLink FIFO**

The LocalLink interface defines a set of protocol-agnostic signals that allows transmission of packet-oriented data and enables a set of features such as flow control and transfer data of arbitrary length. The source and destination ready signals allow for explicit flow control on the LocalLink interface. The LocalLink FIFO is designed based on XAPP258 *FIFOs using Virtex-II Block RAM* and XAPP261 *Data-Width Conversion FIFOs Using the Virtex-II Block RAM Memory*. The reader is strongly encouraged to read these two application notes and the *LocalLink Interface Specification* before proceeding.

The LocalLink FIFO can be implemented in either block SelectRAM memory or distributed RAM in Virtex-II and Virtex-II Pro series FPGAs. FIFOs implemented in block SelectRAM provide a larger memory space than the ones implemented in distributed RAM.

The LocalLink FIFO features the following:

- Implements memory in either block SelectRAM or LUT-based distributed RAM
- Provides a LocalLink interface on both read and write ports
  - Inherits LocalLink features such as flow control and packet-oriented data transfer
- Supports data widths from 8 bits to 128 bits
- Supports memory depths of up to 512 locations for distributed RAM and up to 32,767 locations for block SelectRAM
- Provides fully synchronous and independent clock domains for the read and write ports
- Maintains low latency with only 2 to 3 pipeline stages implemented
- Includes 4-bit FIFO status signals for indicating the percentage of the FIFO occupied
- Offers optional frame length count, length ready, and length error outputs
- Operates at speeds around 200 MHz

Figure 2 is a high-level block diagram of the LocalLink FIFO and its interfaces. It contains three major blocks, the Data FIFO, the Control FIFO, and the optional Length FIFO. The Data FIFO stores the frame data. The Control FIFO stores the LocalLink control data such as the frame delimiters (SOF, EOF) and the remainder (REM). Both the Data FIFO and the Control FIFO share the same read and write address counters, hence they operate synchronously. Every data word stored into the Data FIFO associates with a control word stored into the Control FIFO. The optional Length FIFO uses its own read and write address counters. Each data entry in the Length FIFO indicates the length of a full frame, which may contain one or more data words stored in the Data FIFO. When the Length FIFO is implemented, the frame length (**len_out**), length ready (**len_rdy_out**), and length error (**len_err_out**) outputs on the LocalLink FIFO interface provide visibility into the byte length of each received frame. These signals allow a downstream user application to acquire the length of a frame prior to reading the data.

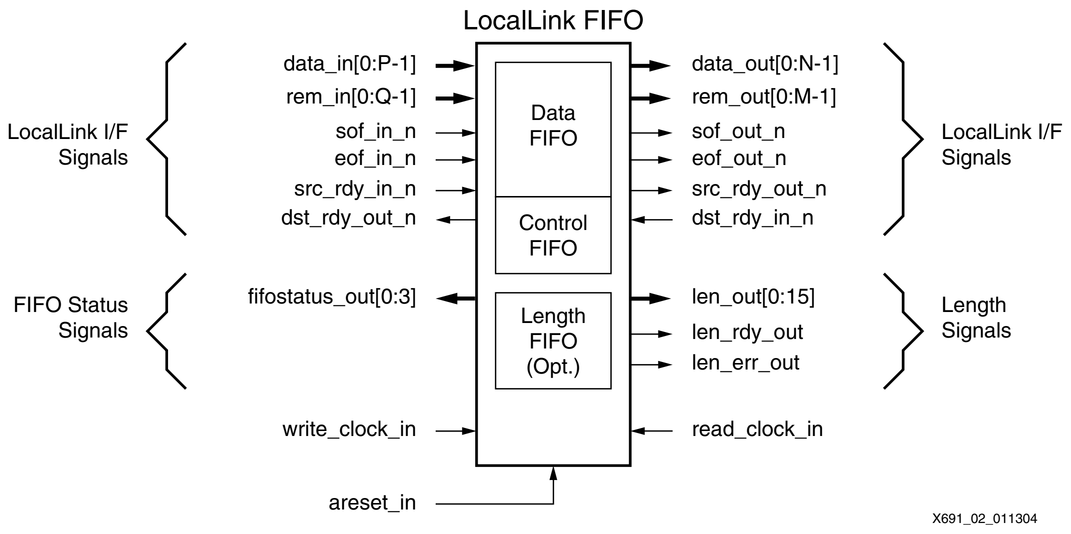

Figure 2: **LocalLink FIFO Interface**

Interface

The LocalLink FIFO interface signals are shown in Figure 2, page 2. Table 1 lists the clock and reset signals on the LocalLink FIFO interface. Table 2 provides the description of the LocalLink data and control signals to the upstream and downstream applications. Table 3, page 4 and Table 4, page 4 list the FIFO status signals. The optional Length signals listed in Table 5, page 5 provide frame length count, length ready, and length error outputs.

The LocalLink FIFO interfaces use big-endian data ordering of vector signals such as **data_in**, **data_out**, **rem_in**, and **rem_out**. The most significant byte (MSB) on **data_in** is transmitted first and appears on the MSB on **data_out**.

Table 1: **Clock and Reset Signals on LocalLink FIFO**

Signal Name	Scope	Direction	Description
areset_in	Upstream/Downstream	Input	Asynchronous reset input. (Active high)
write_clock_in	Upstream	Input	Clock input for the write port from upstream user application.
read_clock_in	Downstream	Input	Clock input for the read port from the downstream user application.

Table 2: **LocalLink Interface Signals on LocalLink FIFO**

Signal Name	Scope	Direction	Description	Notes
data_in[0:P-1]	Upstream	Input	Data input from upstream user application. Data is *P* bits wide.	P = 8, 16, 32, 64, 128
rem_in[0:Q-1]		Input	Remainder input from the upstream user application. Indicates number of valid bytes on the **data_in** when **eof_in_n** is asserted. Remainder is *Q* bits wide. Remainder is binary encoded. If **data_in** is 32 bits wide, then the remainder signal is **rem_in[0:1]**. Remainder value of 0 indicates byte **rem_in[0:7]** is valid, while a value of 3 indicates all bytes are valid.	If P = 8, Q should be 1; otherwise, Q = $\log_2(P/8)$
sof_in_n		Input	Indicates the beginning of a frame transfer on the **data_in** bus.	Active low
eof_in_n		Input	Indicates the end of a frame transfer on the **data_in** bus.	Active low
src_rdy_in_n		Input	Indicates **data_in** is valid during the current cycle.	Active low
dst_rdy_out_n		Output	Indicates that the LocalLink FIFO is ready to accept data presented to it on the **data_in** bus in the current cycle.	Active low

Table 2: **LocalLink Interface Signals on LocalLink FIFO *(Continued)***

Signal Name	Scope	Direction	Description	Notes
data_out[0:N-1]	Down-stream	Output	Data output to downstream user application. Data is *N* bits wide.	N = 8, 16, 32, 64, 128
rem_out[0:M-1]		Output	Remainder output to the downstream user application. Indicates number of valid bytes on the **data_out** when **eof_out_n** is asserted. Remainder is *M* bits wide.Remainder is binary encoded. If **data_out** is 32 bits wide, then the remainder signal is **rem_out[0:1]**. Remainder value of 0 indicates byte **rem_out[0:7]** is valid, while a value of 3 indicates all bytes are valid.	If N = 8, M should be 1; otherwise, M = $\log_2(N/8)$
sof_out_n		Output	Indicates the beginning of a frame transfer on the **data_out** bus.	Active low
eof_out_n		Output	Indicates the end of a frame transfer on the **data_out** bus.	Active low
src_rdy_out_n		Output	Indicates **data_out** is valid during the current cycle.	Active low
dst_rdy_in_n		Input	Indicates that the downstream user application is ready to accept data presented to it on the **data_out** bus in the current cycle.	Active low

Table 3: **FIFO Status Signals on LocalLink FIFO**

Signal Name	Direction	Description	Notes
fifostatus_out[0:3]	Output	To indicate when the FIFO is half full, three-quarters full, and so on. See Table 4 for bit definition of this signal.	Synchronous to **write_clock_in**.

Table 4: **Bit Definition of FIFO Status Signal**

Bit 0	Bit 1	Bit 2	Bit 3	Description
1	1	1	1	LocalLink FIFO is 15/16 full
1	1	1	0	LocalLink FIFO is 7/8 full
1	1	0	1	LocalLink FIFO is 13/16 full
1	1	0	0	LocalLink FIFO is 3/4 full
1	0	1	1	LocalLink FIFO is 11/16 full
1	0	1	0	LocalLink FIFO is 5/8 full
1	0	0	1	LocalLink FIFO is 9/16 full
1	0	0	0	LocalLink FIFO is 1/2 full

Table 4: **Bit Definition of FIFO Status Signal *(Continued)***

Bit 0	Bit 1	Bit 2	Bit 3	Description
0	1	1	1	LocalLink FIFO is 7/16 full
0	1	1	0	LocalLink FIFO is 3/8 full
0	1	0	1	LocalLink FIFO is 5/16 full
0	1	0	0	LocalLink FIFO is 1/4 full
0	0	1	1	LocalLink FIFO is 3/16 full
0	0	1	0	LocalLink FIFO is 1/8 full
0	0	0	1	LocalLink FIFO is 1/16 full
0	0	0	0	LocalLink FIFO is < 1/16 full

Table 5: **LocalLink FIFO Length Signals**

Signal Name	Direction	Description
len_out[0:15]	Output	Frame length in bytes. If the length of a received frame exceeds the maximum frame length supported by the LocalLink FIFO, this output bus will remain at 65,535. See "(c) Maximum Frame Length," page 12 for details.
len_rdy_out	Output	An active high pulse signal to indicate a new frame length output is valid on **len_out** bus.
len_err_out	Output	An active high signal to indicate an overflow of the Length FIFO in a block SelectRAM implementation. Always deasserted in a distributed RAM implementation.

Notes:

1. Outputs are synchronous to **read_clock_in**
2. If Length FIFO is not implemented, all these outputs are tied to zero.
3. Outputs are not valid if users choose distributed RAM implementation and the read clock frequency is faster than the write clock frequency. See details in section Length FIFO.

Hardware Implementation

Memory Types

The LocalLink FIFO uses one of two memory types in Virtex-II or Virtex-II Pro series FPGAs. Block SelectRAM or distributed RAM can be used to implement Data, Control, and Length FIFOs. Users can choose one or the other but should not mix the two types of memories in the implementation. For example, if users choose block SelectRAM memory for the Data FIFO, they must also choose block SelectRAM memory for the Control FIFO. Since block SelectRAM memory is capable of providing larger memory space for the Data FIFO, the alternative of using the distributed RAM to implement the Control FIFO is not practical. Making a distributed RAM Control FIFO coupled to a block SelectRAM Data FIFO typically requires multiple distributed RAMs connected in parallel. This results in a large multiplexer at the output of these distributed RAMs, which yields low performance. For example, the depth of block SelectRAM to store 8-bit data is 2,048. This requires 32X8 RAM64X1D cascaded together. Using this type of implementation, the speed performance will significantly decrease as the memory depth increases. Therefore, implementation in such mixed types of memory is not a good design practice.

Timing Considerations

The read and write ports on the LocalLink FIFO can be operated on independent asynchronous clock domains. However, the user application logic must address synchronization issues. As shown in Figure 2, page 2, interface signals are segmented according to their applicable clock domains, write on the left side and read on the right side. All interface signals must be synchronous to one of the two clocks (**read_clock_in** or **write_clock_in**) and sampled at the rising edge of the clock, with exception of **areset_in**, which is an asynchronous reset of the entire LocalLink FIFO.

Block SelectRAM Implementation

The Data FIFO can be constructed from dual-port block SelectRAM primitives in Virtex-II or Virtex-II Pro series FPGAs. A single primitive (RAMB16_Sm_Sn) provides up to 32-bit wide (not including parity bits) read and write ports and 18 Kb of memory. When connected in parallel, these primitives can form macros with different address widths and memory depths, with up to 128-bit data inputs and outputs. By using these macros, the LocalLink FIFO design can support data widths of 8, 16, 32, 64, and 128 bits on either read or write ports. Data width conversion is handled by the block SelectRAM macros internally.

The BRAM_MACRO_NUM parameter is used to specify the number of macros implemented in the FIFO, thus affecting the depth of the FIFO. Table 6 lists the number of block SelectRAM memories required to implement the Data FIFO with various data widths and BRAM_MACRO_NUM settings. See Table 8, page 13 for a complete description of the BRAM_MACRO_NUM parameter.

Table 6: **Data FIFO Capacity and Block SelectRAM Usage in LocalLink FIFO**

Data Width*	Number of Block SelectRAM	Data FIFO Capacity
8 bits	1 x BRAM_MACRO_NUM	2 x BRAM_MACRO_NUM KB
16 bits		
32 bits		
64 bits	2 x BRAM_MACRO_NUM	4 x BRAM_MACRO_NUM KB
128 bits	4 x BRAM_MACRO_NUM	8 x BRAM_MACRO_NUM KB

* Indicates the largest data width between read and write ports

The Control FIFO can be constructed from the parity memory available in each block SelectRAM memory. In the Type I implementation shown in Figure 3, page 7, the LocalLink control signals fit within the parity ports (DIPA and DOPB) and thus the Control FIFO and Data FIFO can share the same block SelectRAM memory. If the LocalLink control signals exceed the width of the parity ports, a separate block SelectRAM memory is implemented to store the rest of the control data, as shown in the Type II implementation in Figure 4, page 7. The Type III implementation shown in Figure 5, page 8, uses two dedicated block SelectRAM memories to serve as the Data FIFO and the Control FIFO. The LocalLink FIFO chooses one of these three implementations according to the read and write data width combinations selected by users.

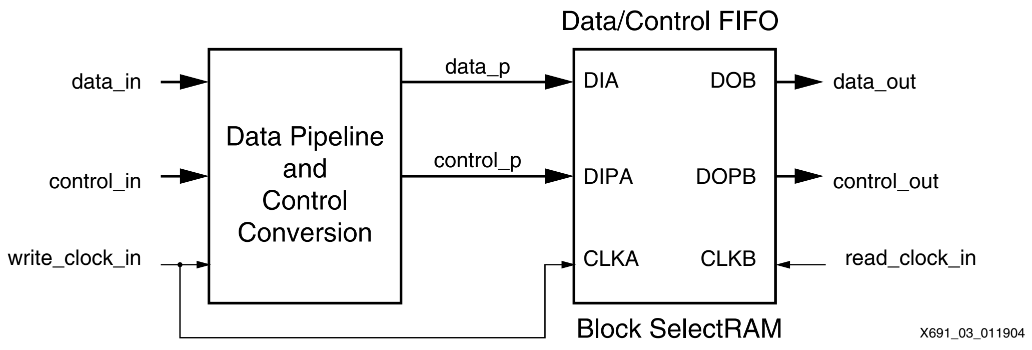

Figure 3: **Type I Block SelectRAM Implementation for Data and Control FIFO**

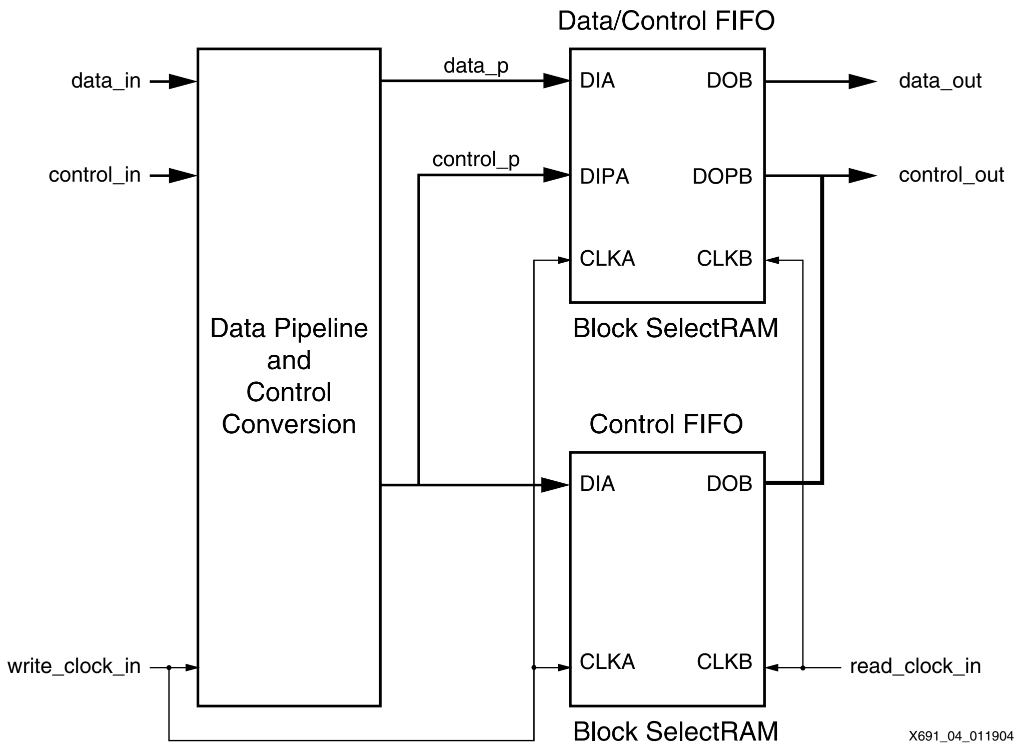

Figure 4: **Type II Block SelectRAM Implementation for Data and Control FIFO**

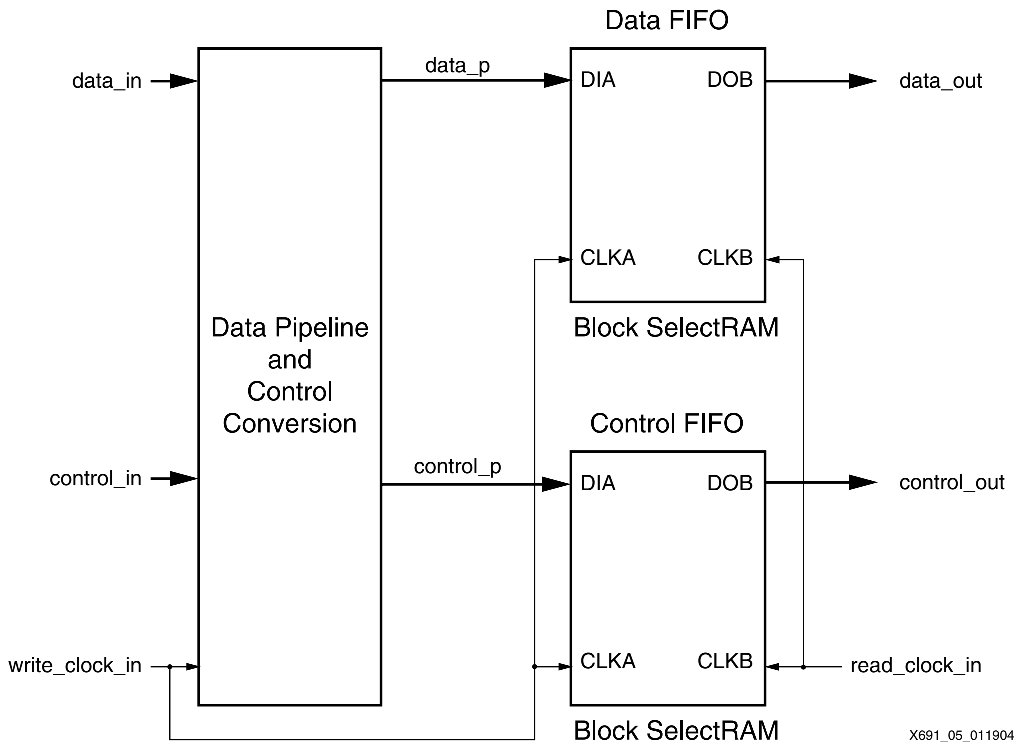

Figure 5: **Type III Block SelectRAM Implementation for Data and Control FIFO**

Distributed RAM Implementation

As an alternative to a block SelectRAM implementation, the Data FIFO can use dual-port, LUT-based distributed RAM. These primitives are cascaded to form macros using multiple RAM16X1D, RAM32X1D, and RAM64X1D primitives that accomplish the selected depth and width. Unlike block SelectRAM memory, dual-port distributed RAM cannot perform data width conversion, thus the data width conversion logic is attached to either the read port or the write port of the distributed RAM.

The Type I implementation in Figure 6, page 9 shows that when the data width on the write port (WR_DWIDTH parameter) is larger than the width on the read port (RD_DWIDTH parameter), the data width conversion logic is placed at the read port of the distributed RAM. The Type II implementation in Figure 7, page 9 shows that when the data width on the write port is smaller than the width on the read port, the data width conversion logic is placed at the write port of the distributed RAM. If the data widths of the read and write ports are the same, data width conversion logic is unnecessary and not implemented, as shown in the Type III implementation in Figure 8, page 10. See Table 8, page 13 for a complete description of the parameters for a distributed RAM implementation.

The Control FIFO stores the REM, SOF, and EOF control data. The Control FIFO is always implemented in a dedicated distributed RAM which shares the same address pointers as the Data FIFO. If data width conversion is required for the Data FIFO, similar mapping logic is implemented for the Control FIFO.

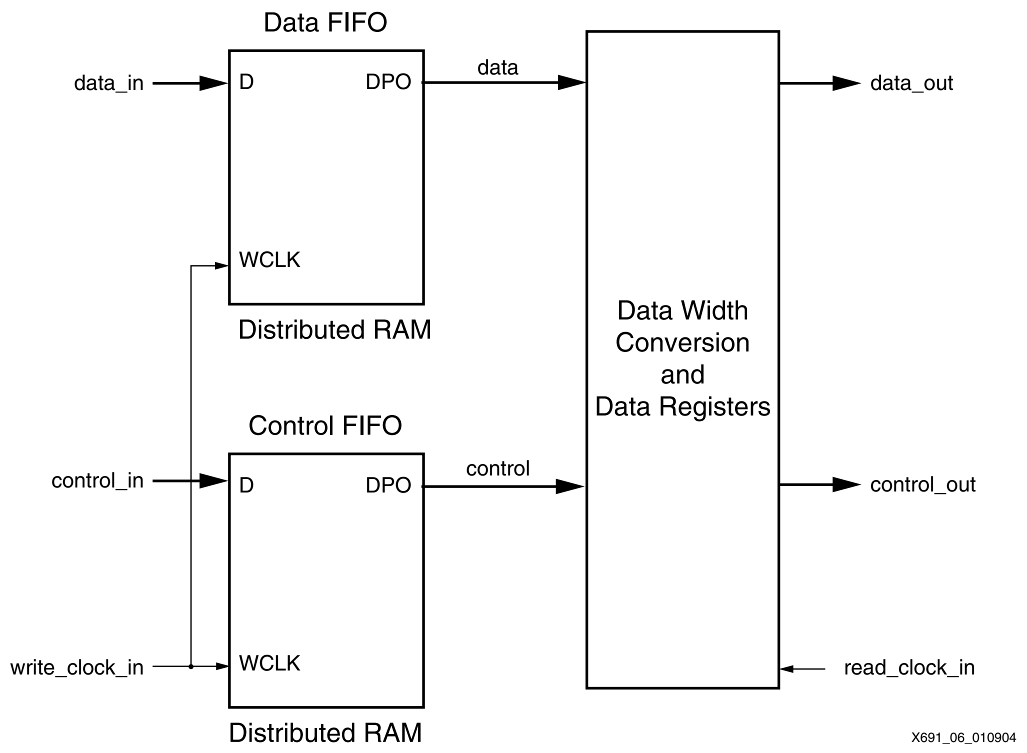

Figure 6: **Type I Distributed RAM Implementation for Data and Control FIFO (WR_DWIDTH > RD_DWIDTH)**

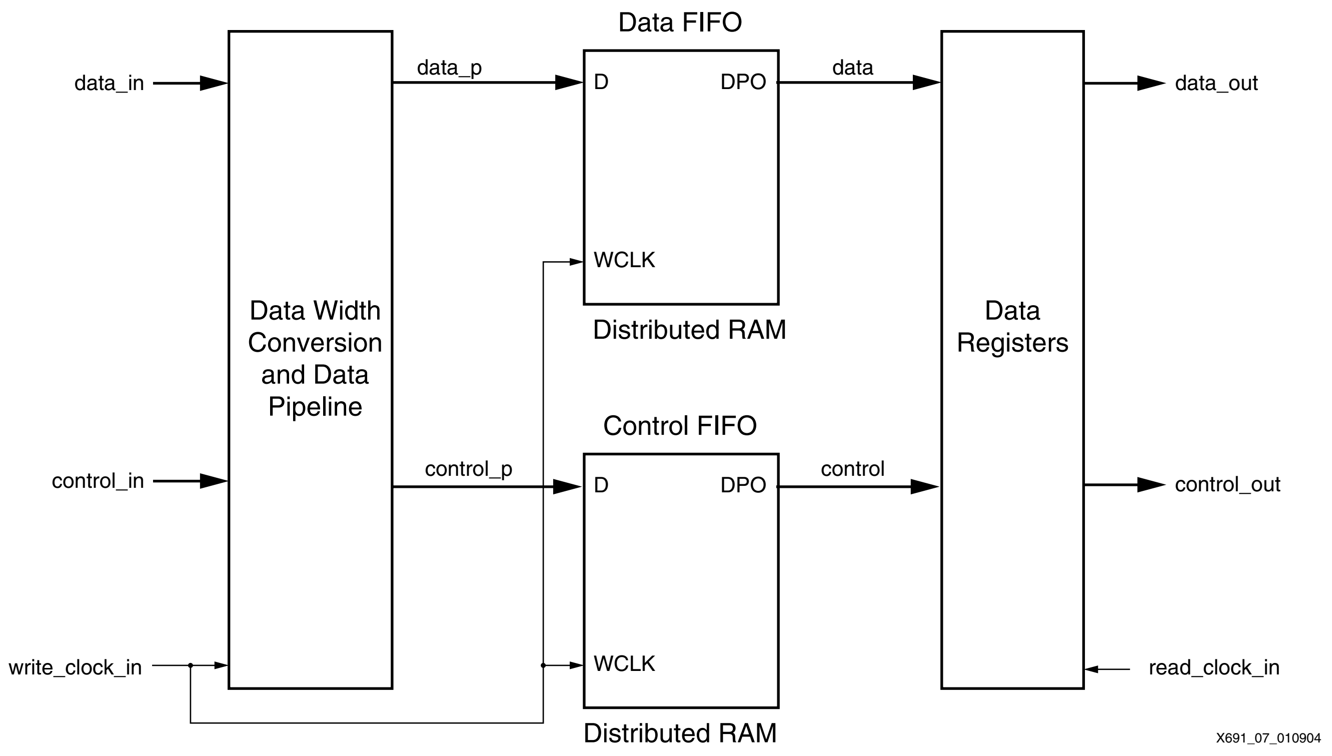

Figure 7: **Type II Distributed RAM Implementation for Data and Control FIFO (WR_DWIDTH < RD_DWIDTH)**

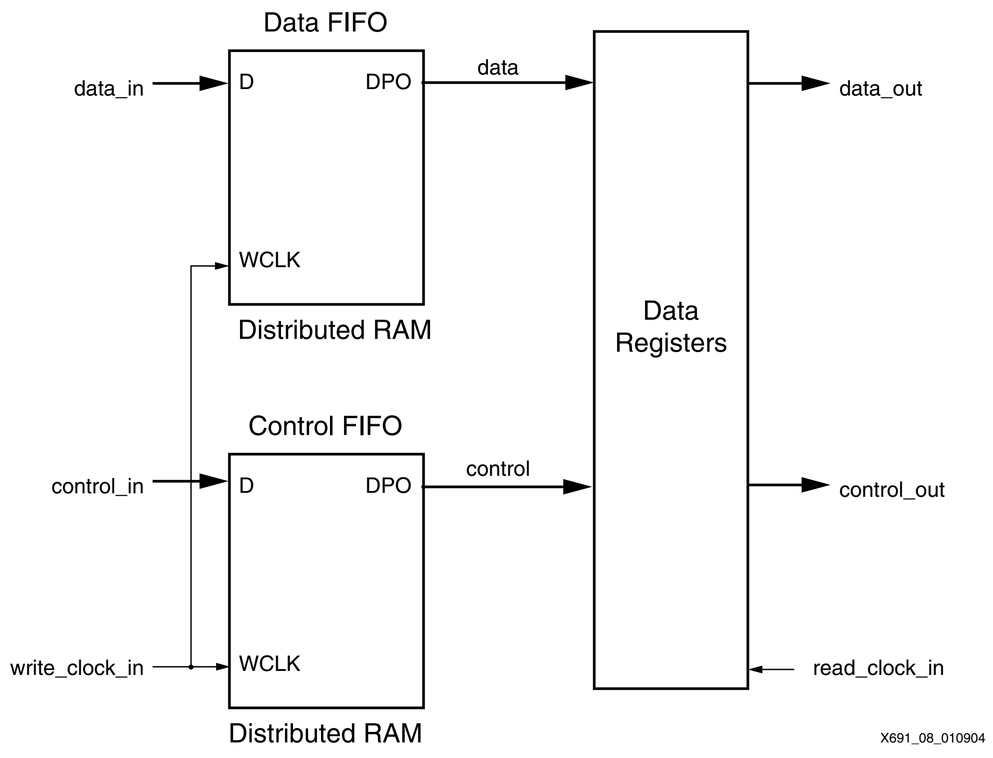

Figure 8: **Type III Distributed RAM Implementation for Data and Control FIFO (WR_DWIDTH = RD_DWIDTH)**

Length FIFO

The Length FIFO is an optional block used to store frame lengths in bytes for each received frame. A downstream user application that needs to acquire the frame length before buffering the entire frame can choose to sample the length ready signal (**len_rdy_out**) on the LocalLink FIFO. When **len_rdy_out** is asserted high, it indicates an entire frame has been stored into the LocalLink FIFO, and the frame length count is preset on the **len_out** bus. The **len_rdy_out** signal is a single-cycle pulse for each frame.

The Length FIFO can be implemented in block SelectRAM memory or distributed RAM depending on the memory type used by the Data FIFO and Control FIFO. A fixed 14-bit counter adds up the number of data words written into the Data FIFO within the frame. The frame length is calculated by multiplying the counter value with the byte number in each word. After the entire frame is received and the computed frame length is stored into the Length FIFO, a length ready (**len_rdy_out**) signal is asserted at the LocalLink FIFO interface, which is a single pulse for each received frame. The downstream user application can sample this signal and latch in the frame length output (**len_out**) at the interface.

The Length FIFO uses dedicated memory, address pointers, and associated logic. The pointer management on the Length FIFO is independent to the Data and Control FIFOs. Each address location in the Length FIFO stores a frame length, which corresponds to multiple locations in the Data and Control FIFOs. Because of this “one-to-many” nature, the Length FIFO cannot share the pointer management logic (especially the FIFO status generation) with the Data and Control FIFOs. To simplify and reduce the size of the implementation, the Length FIFO does not implement FIFO pointer management logic, i.e., using the full and empty status flags to guard the write and read operations. Therefore, some special considerations and limitations are applied in the Length FIFO as described in “(a) Length FIFO Self-Clear Mechanism,” “(b) Length FIFO Overflow Issue,” and “(c) Maximum Frame Length.”

(a) Length FIFO Self-Clear Mechanism

The Length FIFO implemented in block SelectRAM memory uses the parity bits to store the length ready flags. As described before, the Length FIFO does not use full or empty status flags to guard the read or write pointers. If the FIFO read pointer wraps around, a residual data left in the memory from a previous write could accidentally trigger an incorrect length output. To avoid this, the Length FIFO requires a self-clear mechanism to clear any residual data in the FIFO. After the computed frame length is presented at the interface, the Length FIFO writes zeros to the current read location, then advances to the next read location. Such a function requires a full dual-port memory, which has both read and write access to the same memory location on each port. Block SelectRAM memory is a good match for this purpose. As shown in Figure 9, port B on the block SelectRAM is used to read out the length value from the FIFO and write in zeros to the same memory location at the next cycle.

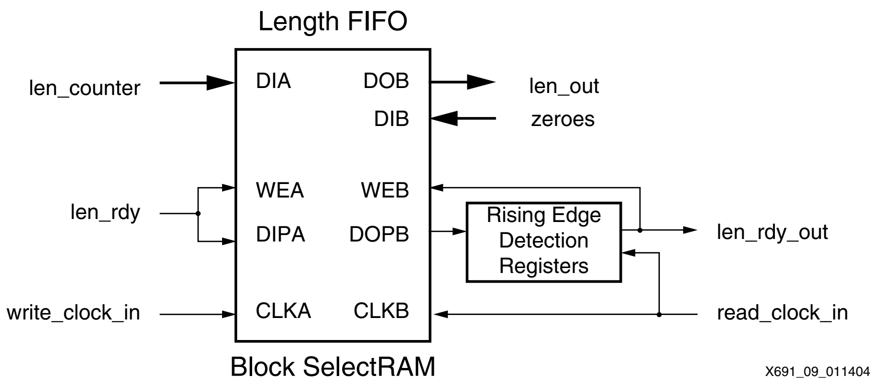

Figure 9: **Block SelectRAM Implementation for Length FIFO**

Unlike the block SelectRAM, the distributed RAM has only one write port; such self-clear logic cannot be implemented on the distributed RAM. Instead, a compromise solution for distributed RAM is implemented in the Length FIFO to simplify the overall design. Each time the Length FIFO finishes writing a length value into the FIFO, it advances its write pointer and immediately writes zeros into the memory location before writing the next valid length. However, this solution is only valid when the write clock frequency is faster than or equal to the read clock frequency.

The LocalLink FIFO does not provide valid length outputs if users choose a distributed RAM implementation with the read clock frequency set faster than the write clock frequency.

(b) Length FIFO Overflow Issue

In a distributed RAM implementation, the depth of the Length FIFO is parameterizable and always equal to the depth of the Data FIFO. The Data FIFO never overflows because its pointer management guards against excessive writes to the Data FIFO when it becomes full. As described before, the Length FIFO and Data FIFO have a “one-to-many” relation. Hence, the Length FIFO will never overflow. In a distributed RAM implementation, the **len_err_out** signal on the LocalLink FIFO interface is always deasserted.

In a block SelectRAM implementation, the Length FIFO uses only one block SelectRAM for simplicity, which provides up to 1,024 memory locations to store the length count values. Since the memory depth of the Length FIFO is not parameterizable, the Length FIFO may not have enough depth to store the length count values for the maximum number of frames stored in the Data FIFO. This will likely cause the Length FIFO to overflow if the Data FIFO depth is much greater and the average length of received frames is relatively small. For example, if the received frame is 4 bytes long on average, an 8-bit wide Data FIFO with 4,096 write address locations will likely cause the Length FIFO to overflow. If the Length FIFO overflows, the **len_err_out** signal on the LocalLink FIFO interface will be asserted to indicate that the length output on the interface is no longer valid. It requires a reset on the LocalLink FIFO to leave this error state.

(c) Maximum Frame Length

The Length FIFO uses a fixed 14-bit counter, which counts up to 16,383 data words. The width of a data word is set by the WR_DWIDTH parameter. The frame length output of the LocalLink FIFO is fixed at 16-bits wide, which provides a frame length count in bytes up to 65,535. Table 7 lists the maximum frame length supported by the Length FIFO. If WR_DWIDTH is equal to 8, the Length FIFO can count a frame up to 16,385 bytes to accommodate jumbo frame lengths in Ethernet applications. If a received frame has a length that exceeds the maximum frame length supported on the LocalLink FIFO, the frame length output will remain at 65,535.

Table 7: **Maximum Frame Length Supported on the Length FIFO**

WR_DWIDTH (Bits)	Maximum Frame Length (Bytes)
8	16,385
16	32,770
32	65,535
64	65,535
128	65,535

LocalLink FIFO Status Signal

The LocalLink FIFO provides the FIFO status signal (**fifostatus_out**) to indicate half full, quarter full, and so on, to the upstream user application. This signal is synchronous to the write clock (**write_clock_in**). This signal inherits this FIFO status signal from the original FIFO designs that accompany XAPP258 and XAPP261. Please consult these application notes for details.

Compilation Parameters

Table 8 lists the LocalLink FIFO parameterization values and their descriptions.

Table 8: **LocalLink FIFO Parameters**

Name	Values	Default Values	Description
MEM_TYPE	Integer: 0 or 1	0	Selects the type of memory. 0 = block SelectRAM 1 = distributed RAM
DRAM_DEPTH	Integer: 16, 32, 64, 128, 256, or 512	16	Sets the memory depth in a distributed RAM implementation. If the data width on the read port is larger than the write port, this memory depth number will regulate the maximum address locations on the read port. If the data width on the read port is smaller than the write port, this memory depth number will regulate the maximum address locations on the write port. For example, a LocalLink FIFO with 32-bit read data port and 64-bit write data port, and having a depth of 64 will be implemented using a 64X64 bit memory block.
BRAM_MACRO_NUM	Integer: 1, 2, 4, 8, or 16	1	Sets the memory depth in a block SelectRAM implementation by setting the number of block SelectRAM macros. As shown in Table 6, data widths and this parameter determine the actual depth of the FIFO.
WR_DWIDTH	Integer: 8, 16, 32, 64, or 128	32	Selects the data width on the write port.
RD_DWIDTH	Integer: 8, 16, 32, 64, or 128	32	Selects the data width on the read port.
WR_REM_WIDTH	Integer: 1, 2, 3, or 4	2	Sets the remainder width on the write port. If WR_DWIDTH = 8, this parameter must be set to 1. In all other cases this parameter can be calculated using the equation $\log_2$(WR_DWDITH/8).
RD_REM_WIDTH	Integer: 1, 2, 3, or 4	2	Sets the remainder width on the read port. If RD_DWIDTH = 8, this parameter must be set to 1. In all other cases this parameter can be calculated using the equation $\log_2$(RD_DWDITH/8).
USE_LENGTH	Boolean: TRUE or FALSE	TRUE	Implements the optional Length FIFO if set to TRUE. If set to FALSE, the Length FIFO is removed from the implementation and the length signals on the LocalLink FIFO are tied to zero.

Waveforms

Figure 10 shows basic frame transfers on the LocalLink FIFO with the write data width of 32 (WR_DWIDTH = 32) and the read data width of 16 (RD_DWIDTH = 16). A transfer starts with the upstream user application by asserting **sof_in_n** and **src_rdy_in_n**. The LocalLink FIFO has free buffer space to accept these frames, so it asserts **dst_rdy_out_n**. In this scenario, the downstream application permanently asserts **dst_rdy_in_n** to always accept data outputs from the LocalLink FIFO. Note that the data width conversion and remainder conversion are handled inside the LocalLink FIFO.

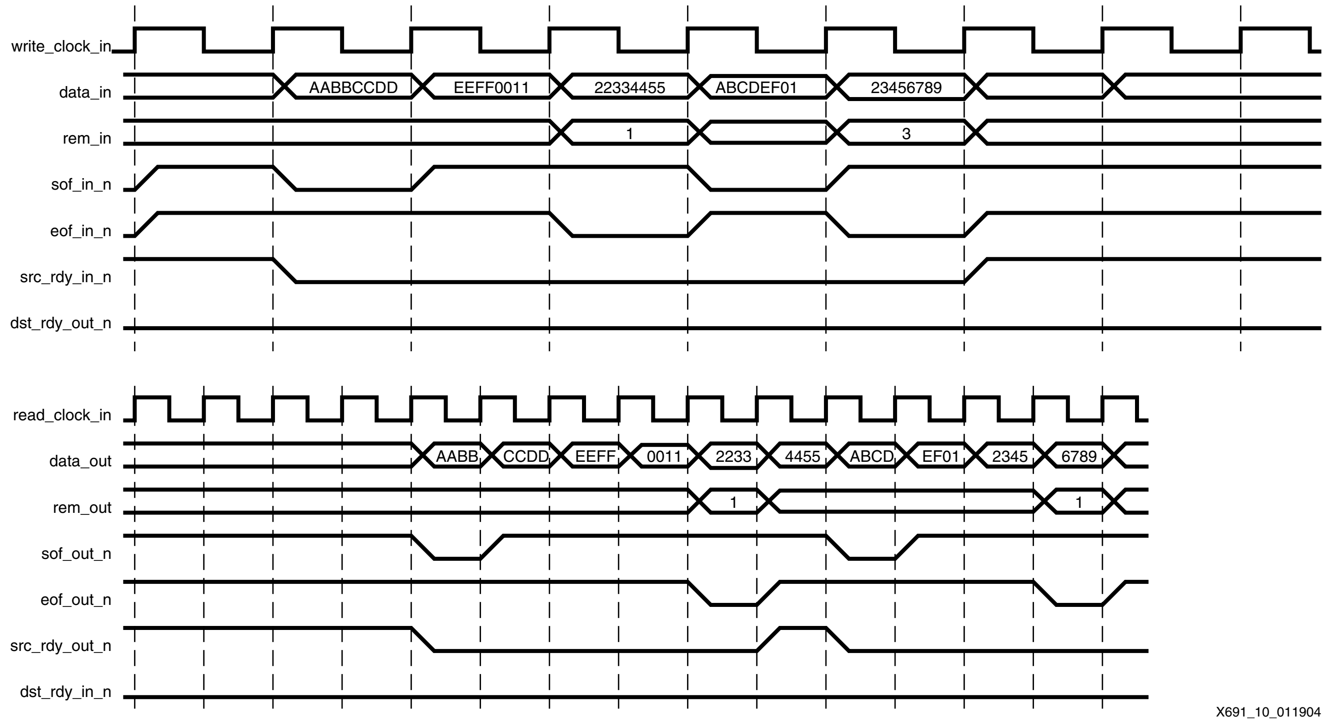

Figure 10: **Basic Frame Transfers on the LocalLink FIFO**

Figure 11 shows frame transfers with flow control between the upstream user application and the LocalLink FIFO. Whenever the LocalLink FIFO is full and not ready to accept new data from the upstream user application, it will hold the **dst_rdy_out_n** deasserted. The upstream user application presents the next set of data bytes after the LocalLink FIFO asserts **dst_rdy_out_n**. This assures that no data is lost when the LocalLink FIFO is not ready to receive data. Note that the **rem_out** signal is not shown in Figure 11 because the **data_out** bus is only one byte wide and does not require a reminder value.

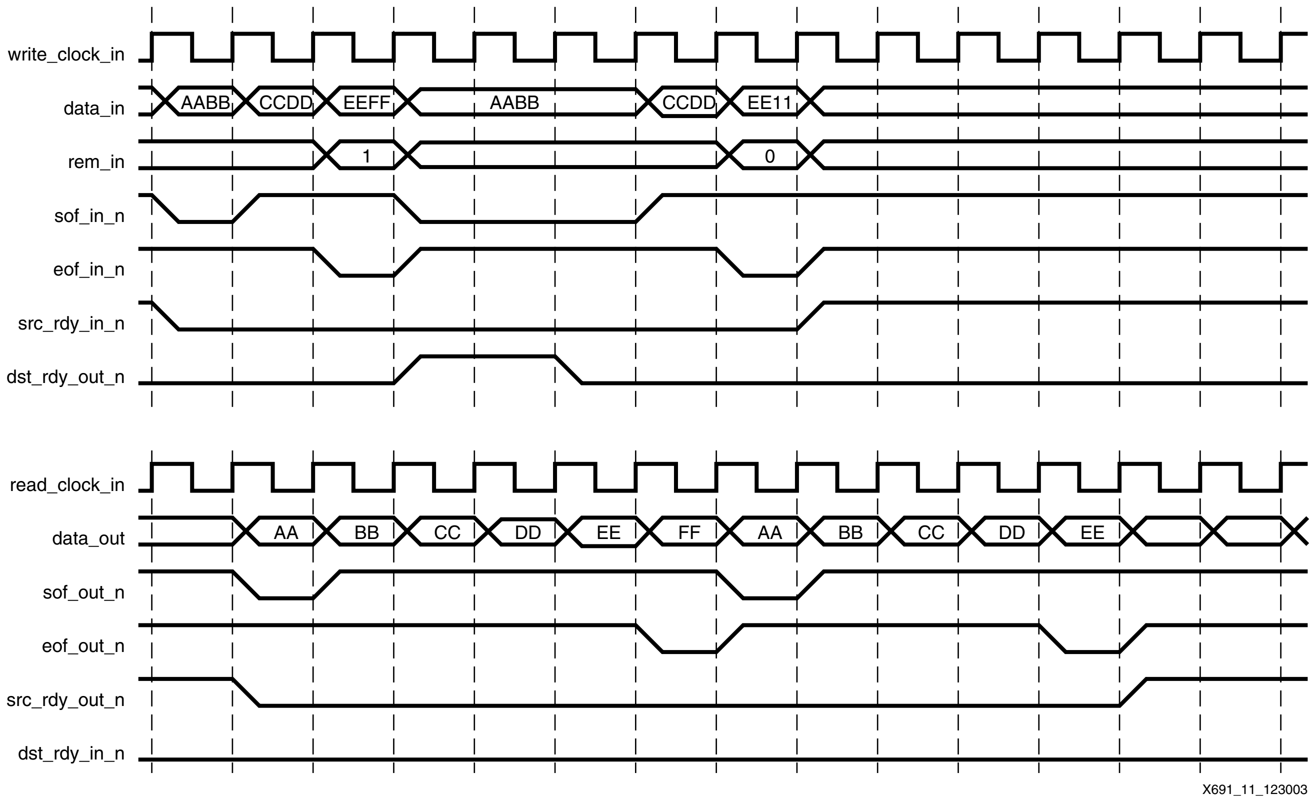

Figure 11: **Frame Transfers on the LocalLink FIFO with Flow Control on the Upstream Interface**

Figure 12 shows frame transfers with flow control between the LocalLink FIFO and the downstream user application. If the downstream user application is not ready to accept new data from the LocalLink FIFO, it should hold **dst_rdy_in_n** deasserted. The LocalLink FIFO will present the next set of data bytes after the downstream application asserts **dst_rdy_in_n**. This assures that no data is lost when the downstream application is not ready to receive data.

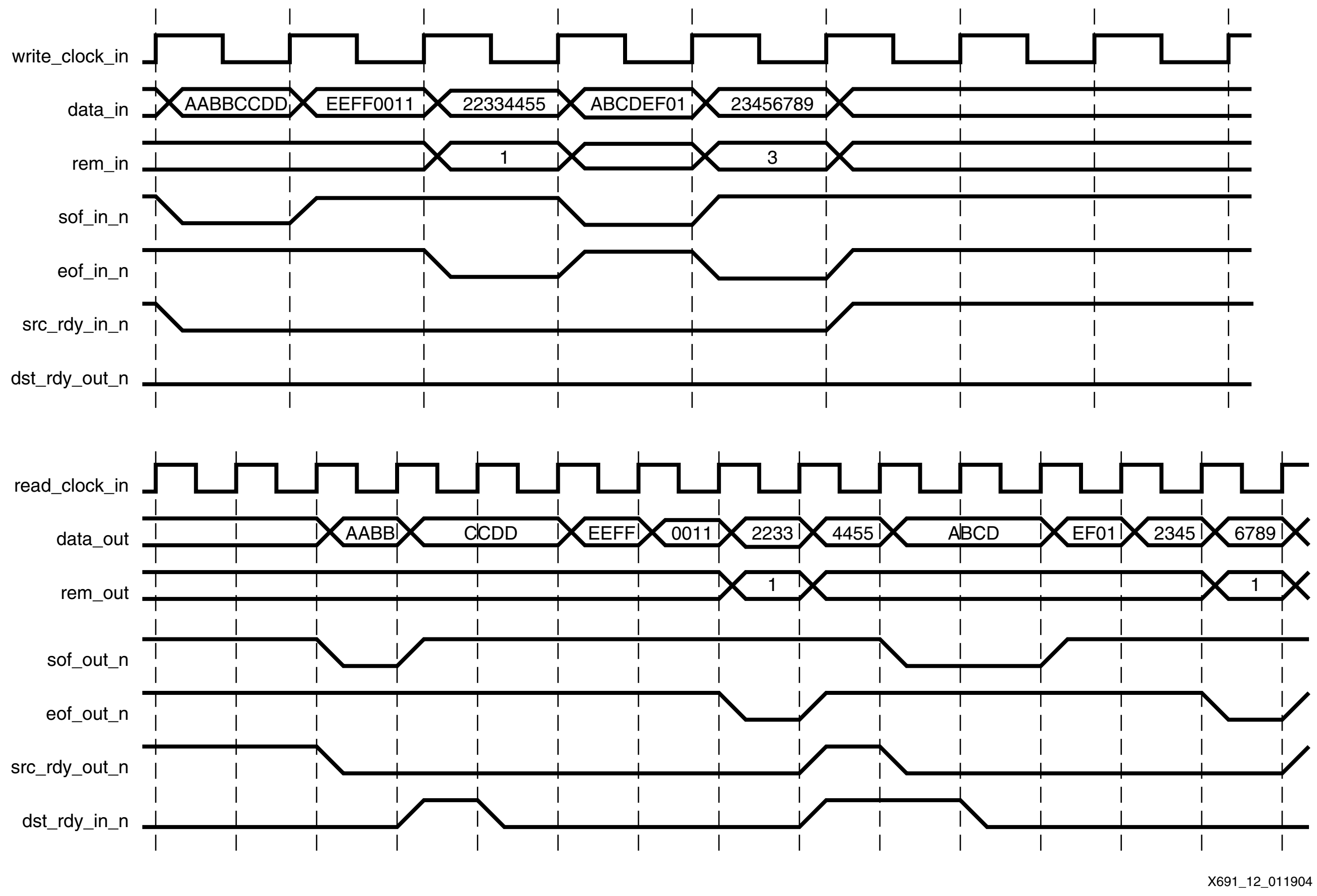

Figure 12: **Frame Transfers on the LocalLink FIFO with Flow Control on the Downstream Interface**

Figure 13 shows frame transfers on the LocalLink FIFO based on the length outputs. The downstream user application can sample on the **len_rdy_out** output and latch the **len_out** bus value from the LocalLink FIFO to acquire the byte length of each frame stored in the LocalLink FIFO prior to reading the frame data. Note that the downstream user application must be able to store all the length outputs if it is still processing a previous frame. The outputs of the length values cannot be held by the downstream user application.

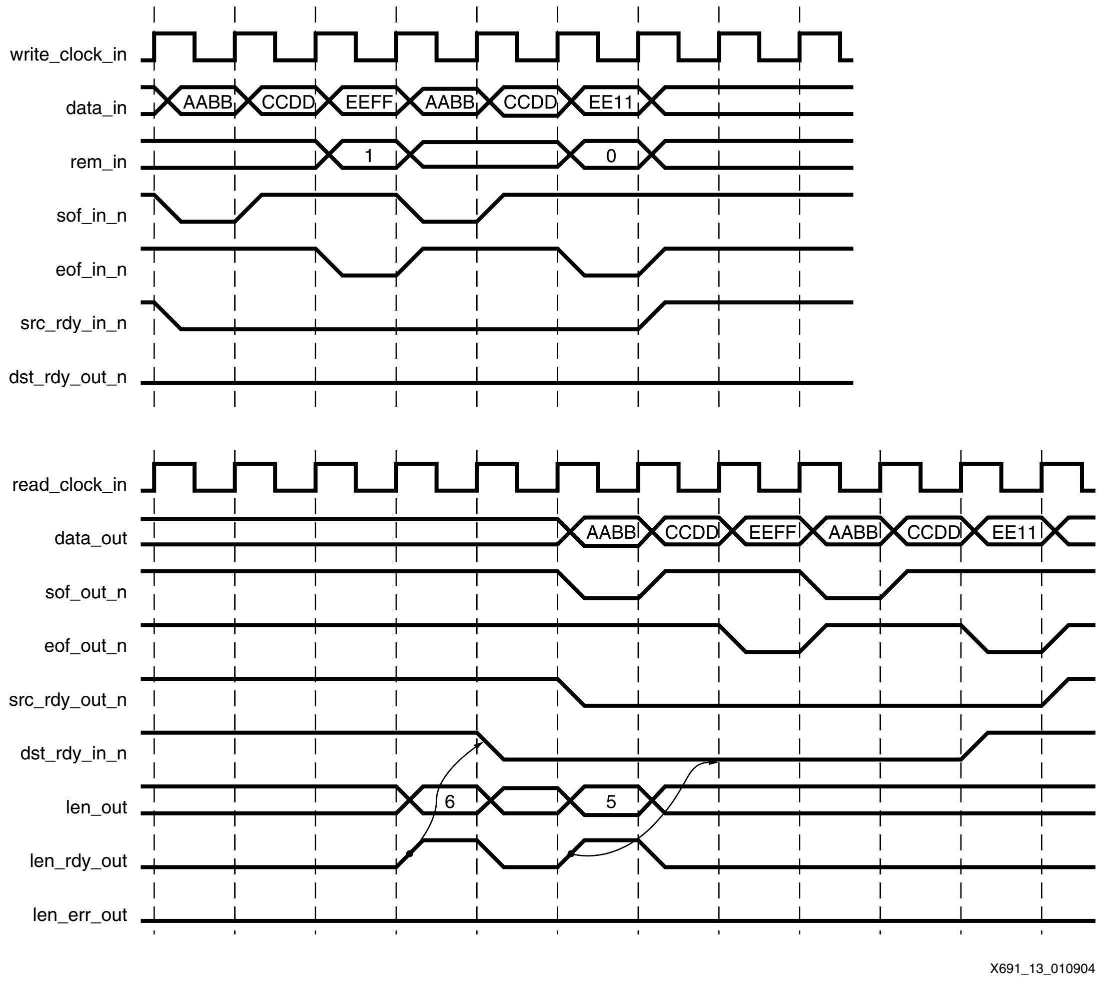

Figure 13: **Frame Transfers on the LocalLink FIFO Based on the Length Outputs**

Verification

Simulation

Figure 14 shows the simulation testbench used to verify the LocalLink FIFO. There are two LocalLink FIFOs instantiated in the testbench, the egress LocalLink FIFO and the ingress LocalLink FIFO. A tester module is connected to one side of both egress and ingress FIFOs in Clock Domain A. The egress and ingress FIFOs are directly connected to each other on the other side in Clock Domain B. The tester module reads a test vector file, which provides vectors for both frame data and control data, and generates signals to the LocalLink interface on the egress FIFO. The tester module also reads the data from the ingress FIFO, compares it to the expected data and generates the test result. The LocalLink FIFO was simulated using Mentor Graphics ModelSim simulator.

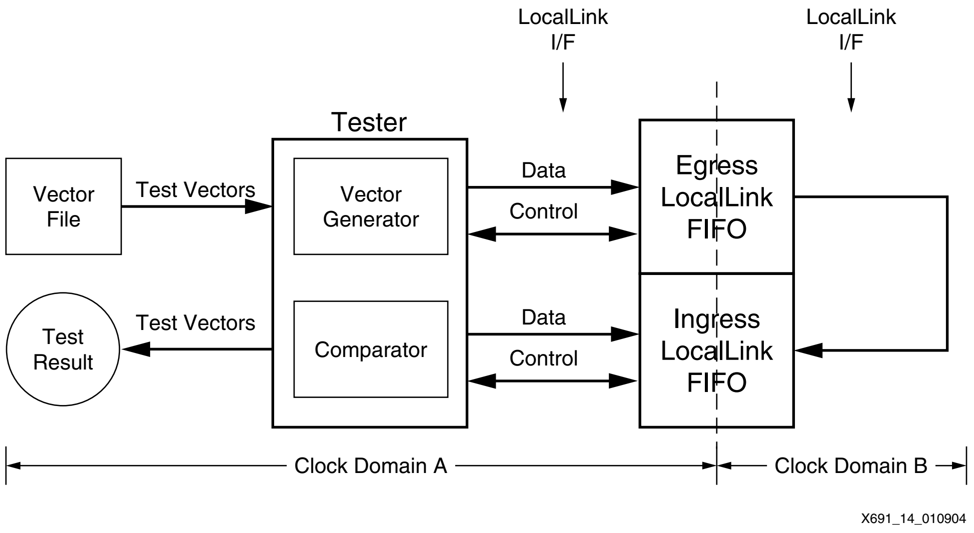

Figure 14: **LocalLink FIFO Simulation Testbench**

Hardware Testing

The LocalLink FIFO was tested in hardware using the ChipScope Integrated Logic Analyzer (ILA) core and its software on the ML320 platform. A pattern generator connected to the upstream interface of the LocalLink FIFO generated framed data with incremental frame lengths. The downstream interface signals on the LocalLink FIFO were brought up to the ChipScope ILA core and verified using ChipScope software. Since the LocalLink FIFO is parameterizable, hardware testing included only typical settings on the data widths and memory types. Tests for both block SelectRAM and distributed RAM implementations included 8-bit, 32-bit, and 128-bit writes with any read width. Read clock and write clock frequencies varied from 50 MHz to 125 MHz in these tests.

Resource Utilization and Performance

Resource utilization of the LocalLink FIFO varies depending on different combinations of data widths on read and write ports, which memory type is used, and whether the optional Length FIFO is implemented. The following tables show the resource utilization and performance for these combinations.

Table 9: **Resources Used (FF, LUT, BRAM) for Block SelectRAM Implementation without Length FIFO**

Write Data Width	Read Data Width	BRAM_MACRO_NUM				
		1	2	4	8	16
8	8	(95, 59, 2)	(102, 74, 3)	(108, 88, 5)	(116, 114, 9)	(136, 178, 20)
8	16	(99, 63, 2)	(105, 86, 3)	(112, 111, 5)	(121, 155, 9)	(133, 263, 18)
8	32	(113, 70, 2)	(119, 112, 3)	(128, 154, 5)	(133, 233, 9)	(160, 406, 17)
8	64	(139, 82, 3)	(145, 148, 5)	(154, 214, 9)	(171, 356, 20)	(250, 644, 40)
8	128	(205, 108, 5)	(211, 238, 9)	(227, 367, 17)	(250, 634, 34)	(330, 1164, 68)
16	8	(89, 59, 2)	(95, 73, 3)	(101, 87, 5)	(109. 115. 10)	(130. 182. 20)
16	16	(100, 57, 2)	(106, 82, 3)	(113, 106, 5)	(121, 152, 9)	(135, 249, 17)
16	32	(112, 67, 2)	(118, 107, 3)	(125, 149, 5)	(133, 227, 9)	(156, 396, 17)
16	64	(140, 80, 3)	(146, 150, 5)	(155, 222, 9)	(158, 360, 17)	(226, 659, 34)
16	128	(204, 105, 5)	(210, 235, 9)	(225, 366, 17)	(247, 632, 34)	(326, 1162, 68)
32	8	(83, 56, 2)	(89, 71, 3)	(95, 85, 5)	(103, 113, 10)	(120, 180, 20)
32	16	(94, 55, 2)	(100, 79, 3)	(107, 104, 5)	(114, 149, 9)	(128, 249, 17)
32	32	(103, 59, 1)	(111, 103, 2)	(123, 145, 4)	(137, 226, 8)	(157, 396, 16)
32	64	(139, 77, 3)	(145, 145, 5)	(155, 215, 9)	(166, 349, 17)	(229, 641, 34)
32	128	(203, 119, 5)	(210, 276, 9)	(218, 410, 17)	(230, 706, 34)	(247, 1300, 68)
64	8	(82, 57, 3)	(88, 72, 5)	(96, 91, 10)	(125, 123, 20)	(149, 192, 40)
64	16	(93, 57, 3)	(99, 82, 5)	(106, 106, 9)	(113, 151, 17)	(147, 258, 34)
64	32	(104, 61, 2)	(112, 102, 4)	(124, 146, 8)	(130, 227, 16)	(171, 400, 32)
64	64	(132, 69, 2)	(140, 142, 4)	(155, 212, 8)	(167, 353, 16)	(223, 640, 32)
64	128	(198, 103, 4)	(207, 243, 8)	(233, 386, 16)	(247, 665, 32)	(357, 1232, 64)
128	8	(81, 59, 5)	(89, 75, 10)	(119, 95, 20)	(124, 124, 40)	(157, 193, 80)
128	16	(88, 58, 5)	(96, 83, 10)	(122, 111, 20)	(154, 158, 40)	(190, 261, 80)
128	32	(109, 63, 5)	(115, 104, 9)	(122, 145, 17)	(152, 228, 34)	(208, 400, 68)
128	64	(133, 71, 4)	(142, 141, 8)	(158, 215, 16)	(178, 354, 32)	(246, 646, 64)
128	128	(197, 91, 4)	(205, 228, 8)	(223, 365, 16)	(261, 635, 32)	(369, 1186, 64)

Table 10: **Resources Used (FF, LUT, BRAM) for Block SelectRAM Implementation with Length FIFO**

Write Data Width	Read Data Width	BRAM_MACRO_NUM				
		1	2	4	8	16
8	8	(145, 111, 3)	(151, 125, 4)	(157, 139, 6)	(165, 163, 10)	(173, 231, 21)
8	16	(149, 114, 3)	(155, 137, 4)	(161, 162, 6)	(167, 207, 10)	(175, 315, 19)
8	32	(163, 120, 3)	(169, 162, 4)	(175, 204, 6)	(181, 286, 10)	(187, 458, 18)
8	64	(189, 133, 4)	(195, 199, 6)	(201, 265, 10)	(209, 408, 21)	(217, 691, 41)
8	128	(255, 159, 6)	(261, 289, 10)	(267, 418, 18)	(275, 684, 35)	(284, 1216, 69)
16	8	(137, 108, 3)	(143, 122, 4)	(149, 136, 6)	(157, 163, 11)	(165, 228, 21)
16	16	(148, 106, 3)	(154, 130, 4)	(160, 154, 6)	(166, 198, 10)	(172, 298, 18)
16	32	(160, 116, 3)	(166, 156, 4)	(172, 198, 6)	(178, 277, 10)	(184, 445, 18)
16	64	(188, 128, 4)	(194, 198, 6)	(200, 270, 10)	(206, 410, 18)	(215, 705, 35)
16	128	(252, 154, 6)	(258, 284, 10)	(264, 414, 18)	(272, 678, 35)	(281, 1211, 69)
32	8	(129, 105, 3)	(135, 120, 4)	(141, 134, 6)	(149, 162, 11)	(166, 230, 21)
32	16	(140, 104, 3)	(146, 128, 4)	(152, 153, 6)	(161, 197, 10)	(174, 299, 18)
32	32	(149, 107, 2)	(157, 149, 3)	(165, 191, 5)	(173, 271, 9)	(193, 444, 17)
32	64	(185, 125, 4)	(191, 193, 6)	(197, 264, 10)	(203, 398, 18)	(252, 689, 35)
32	128	(249, 169, 6)	(256, 324, 10)	(264, 460, 18)	(276, 755, 35)	(293, 1347, 69)
64	8	(126, 104, 4)	(132, 118, 6)	(140, 137, 11)	(150, 167, 21)	(187, 237, 41)
64	16	(137, 103, 4)	(143, 128, 6)	(149, 152, 10)	(157, 198, 18)	(191, 305, 35)
64	32	(148, 107, 3)	(156, 148, 5)	(165, 192, 9)	(175, 273, 17)	(213, 445, 33)
64	64	(176, 114, 3)	(184, 185, 5)	(192, 256, 9)	(200, 394, 17)	(240, 682, 33)
64	128	(242, 148, 5)	(250, 288, 9)	(268, 430, 17)	(285, 710, 33)	(401, 1278, 65)
128	8	(123, 102, 6)	(131, 118, 11)	(140, 138, 21)	(166, 167, 41)	(199, 235, 81)
128	16	(130, 101, 6)	(138, 125, 11)	(146, 152, 21)	(178, 198, 41)	(233, 304, 81)
128	32	(151, 105, 6)	(157, 146, 10)	(163, 187, 18)	(183, 267, 35)	(224, 440, 69)
128	64	(175, 113, 5)	(183, 183, 9)	(195, 257, 17)	(209, 395, 33)	(276, 686, 65)
128	128	(239, 132, 5)	(247, 269, 9)	(255, 405, 17)	(270, 672, 33)	(342, 1221, 65)

Table 11: **Performance Benchmarking on XC2VP7 -6 for Block SelectRAM Implementation without the Length FIFO**

Write Data Width	Read Data Width	BRAM_MACRO_NUM				
		1	2	4	8	16
8	8	240 MHz	236 MHz	197 MHz	202 MHz	191 MHz
8	16	179 MHz	190 MHz	176 MHz	168 MHz	142 MHz
8	32	262 MHz	231 MHz	223 MHz	185 MHz	183 MHz
8	64	253 MHz	251 MHz	219 MHz	198 MHz	168 MHz
8	128	247 MHz	210 MHz	203 MHz	169 MHz	169 MHz
16	8	251 MHz	214 MHz	170 MHz	177 MHz	174 MHz
16	16	205 MHz	204 MHz	204 MHz	210 MHz	200 MHz
16	32	256 MHz	254 MHz	198 MHz	182 MHz	194 MHz
16	64	272 MHz	231 MHz	207 MHz	179 MHz	169 MHz
16	128	247 MHz	209 MHz	181 MHz	168 MHz	168 MHz
32	8	242 MHz	211 MHz	251 MHz	191 MHz	183 MHz
32	16	175 MHz	171 MHz	173 MHz	173 MHz	151 MHz
32	32	255 MHz	272 MHz	245 MHz	209 MHz	182 MHz
32	64	226 MHz	208 MHz	188 MHz	170 MHz	170 MHz
32	128	215 MHz	179 MHz	172 MHz	164 MHz	164 MHz
64	8	272 MHz	238 MHz	176 MHz	187 MHz	169 MHz
64	16	187 MHz	188 MHz	181 MHz	176 MHz	153 MHz
64	32	192 MHz	238 MHz	173 MHz	188 MHz	173 MHz
64	64	262 MHz	286 MHz	228 MHz	188 MHz	170 MHz
64	128	242 MHz	214 MHz	192 MHz	170 MHz	170 MHz
128	8	245 MHz	180 MHz	175 MHz	168 MHz	168 MHz
128	16	207 MHz	172 MHz	171 MHz	150 MHz	150 MHz
128	32	246 MHz	203 MHz	200 MHz	172 MHz	172 MHz
128	64	190 MHz	250 MHz	210 MHz	168 MHz	168 MHz
128	128	256 MHz	240 MHz	193 MHz	170 MHz	170 MHz

Table 12: **Performance Benchmarking on XC2VP7 -6 for Block SelectRAM Implementation with Length FIFO**

Write Data Width	Read Data Width	BRAM_MACRO_NUM				
		1	2	4	8	16
8	8	172 MHz	170 MHz	171 MHz	169 MHz	169 MHz
8	16	168 MHz	170 MHz	171 MHz	167 MHz	143 MHz
8	32	173 MHz	169 MHz	170 MHz	169 MHz	169 MHz
8	64	169 MHz	174 MHz	170 MHz	167 MHz	168 MHz
8	128	168 MHz	173 MHz	174 MHz	170 MHz	170 MHz
16	8	168 MHz	169 MHz	168 MHz	169 MHz	168 MHz
16	16	170 MHz	169 MHz	177 MHz	169 MHz	173 MHz
16	32	169 MHz	173 MHz	169 MHz	169 MHz	167 MHz
16	64	168 MHz	171 MHz	171 MHz	169 MHz	168 MHz
16	128	171 MHz	175 MHz	170 MHz	172 MHz	172 MHz
32	8	215 MHz	179 MHz	181 MHz	187 MHz	178 MHz
32	16	181 MHz	190 MHz	173 MHz	168 MHz	152 MHz
32	32	182 MHz	196 MHz	188 MHz	187 MHz	177 MHz
32	64	225 MHz	184 MHz	205 MHz	175 MHz	167 MHz
32	128	192 MHz	182 MHz	169 MHz	161 MHz	161 MHz
64	8	194 MHz	184 MHz	205 MHz	190 MHz	168 MHz
64	16	195 MHz	174 MHz	177 MHz	170 MHz	151 MHz
64	32	183 MHz	180 MHz	171 MHz	173 MHz	168 MHz
64	64	226 MHz	205 MHz	195 MHz	177 MHz	169 MHz
64	128	245 MHz	215 MHz	170 MHz	168 MHz	168 MHz
128	8	189 MHz	221 MHz	175 MHz	169 MHz	169 MHz
128	16	217 MHz	169 MHz	174 MHz	166 MHz	166 MHz
128	32	243 MHz	235 MHz	168 MHz	177 MHz	177 MHz
128	64	169 MHz	176 MHz	182 MHz	172 MHz	172 MHz
128	128	200 MHz	185 MHz	177 MHz	169 MHz	169 MHz

Table 13: **Resources Used (FF, LUT) for Distributed RAM Implementation without Length FIFO**

Write Data Width	Read Data Width	Depth					
		16	32	64	128	256	512
8	8	49, 52	70, 80	111, 132	175, 230	267, 452	514, 897
8	16	72, 81	106, 129	181, 216	293, 399	456, 809	920, 1646
8	32	112, 127	170, 210	311, 368	513, 709	824, 1483	1697, 3064
8	64	177, 202	281, 350	542, 630	902, 1236	1450, 2631	3028, 5487
8	128	42, 46	51, 60	63, 71	84, 97	113, 148	89, 242
16	8	51, 0	78, 128	161, 217	230, 395	191, 804	379, 1650
16	16	74, 76	101, 122	176, 209	287, 392	451, 808	915, 1650
16	32	110, 124	166, 207	307, 365	509, 706	820, 1479	1693, 3065
16	64	174, 197	275, 345	534, 625	896, 1230	1444, 2625	3025, 5486
16	128	319, 359	511, 647	1021, 1195	1739, 2404	2836, 5194	5988, 10939
32	8	62, 126	80, 207	175, 360	157, 690	257, 1467	477, 3069
32	16	74, 128	117, 211	267, 371	315, 694	272, 1475	792, 3094
32	32	119, 126	161, 200	301, 358	503, 698	815, 1487	1689, 3091
32	64	170, 194	271, 343	530, 622	892, 1227	1440, 2622	3021, 5487
32	128	313, 353	505, 641	1013, 1190	1732, 2401	2830, 5188	5982, 10932
64	8	75, 215	90, 359	150, 644	245, 1275	427, 2761	815, 5829
64	16	93, 217	117, 360	303, 645	261, 1273	441, 2757	859, 5834
64	32	126, 222	193, 373	483, 671	545, 1282	475, 2778	891, 5873
64	64	194, 207	266, 335	525, 614	887, 1222	1436, 2640	3019, 5538
64	128	309, 350	500, 638	1009, 1187	1728, 2398	2826, 5185	5978, 10929
128	8	81, 387	130, 661	233, 1206	411, 2439	770, 5322	1545, 11289
128	16	112, 392	145, 662	249, 1209	423, 2432	780, 5309	1555, 11284
128	32	153, 394	178, 661	482, 1209	457, 2424	814, 5304	1625, 11281
128	64	209, 382	332, 649	858, 1189	972, 2298	841, 5027	1607, 10692
128	128	366, 372	493, 632	1005, 1180	1720, 2389	2823, 5227	5979, 11035

Table 14: **Resources Used (FF, LUT) for Distributed RAM Implementation with Length FIFO**

Write Data Width	Read Data Width	Depth					
		16	32	64	128	256	512
8	8	56, 61	65, 93	76, 147	90, 256	130, 496	170, 1003
8	16	80, 87	89, 139	100, 229	114, 419	172, 847	228, 1762
8	32	119, 131	128, 217	139, 375	157, 718	251, 1503	351, 3184
8	64	183, 205	192, 349	207, 623	233, 1227	391, 2621	571, 5605
8	128	323, 370	336, 648	359, 1183	391, 2371	698, 5131	1032, 11043
16	8	58, 91	67, 141	78, 231	110, 411	156, 849	303, 1765
16	16	73, 81	82, 131	93, 221	107, 411	169, 840	220, 1752
16	32	115, 128	124, 214	135, 372	153, 715	247, 1500	347, 3180
16	64	177, 199	186, 343	201, 617	227, 1221	385, 2615	565, 5601
16	128	313, 358	326, 632	349, 1165	381, 2354	688, 5114	1022, 11026
32	8	67, 135	90, 224	187, 385	177, 724	281, 1526	507, 3178
32	16	76, 136	106, 225	276, 390	320, 729	296, 1531	526, 3201
32	32	109, 121	132, 207	275, 371	352, 711	326, 1531	535, 3174
32	64	174, 196	211, 340	534, 618	597, 1220	244, 2679	907, 5598
32	128	311, 351	377, 625	1009, 1167	1108, 2350	1007, 5241	1677, 11023
64	8	83, 226	100, 379	162, 670	265, 1308	451, 2820	843, 5929
64	16	93, 226	127, 379	315, 671	281, 1307	465, 2816	887, 5973
64	32	112, 226	161, 384	420, 683	551, 1318	796, 2831	917, 5960
64	64	168, 189	204, 33	462, 617	891, 1213	543, 2673	900, 5594
64	128	307, 351	399, 631	1012, 1181	1104, 2348	1003, 5238	1673, 11021
128	8	88, 404	138, 681	245, 1231	425, 2468	789, 5376	1573, 11397
128	16	118, 402	155, 684	261, 1237	443, 2466	803, 5363	1586, 11391
128	32	145, 400	209, 685	494, 1238	477, 2458	838, 5363	1655, 11390
128	64	182, 380	266, 657	665, 1180	964, 2333	864, 5074	1611, 10768
128	128	300, 345	394, 623	800, 1156	1100, 2341	1001, 5232	1661, 11013

Table 15: **Performance Benchmarking on XC2VP7 -6 for Distributed RAM Implementation without Length FIFO**

Write Data Width	Read Data Width	Depth					
		16	32	64	128	256	512
8	8	231 MHz	259 MHz	218 MHz	177 MHz	171 MHz	167 MHz
8	16	224 MHz	236 MHz	187 MHz	173 MHz	167 MHz	168 MHz
8	32	206 MHz	206 MHz	194 MHz	175 MHz	167 MHz	167 MHz
8	64	228 MHz	184 MHz	172 MHz	168 MHz	167 MHz	167 MHz
8	128	244 MHz	249 MHz	243 MHz	194 MHz	178 MHz	172 MHz
16	8	270 MHz	233 MHz	191 MHz	173 MHz	167 MHz	168 MHz
16	16	212 MHz	219 MHz	187 MHz	174 MHz	169 MHz	168 MHz
16	32	219 MHz	210 MHz	190 MHz	170 MHz	168 MHz	168 MHz
16	64	200 MHz	172 MHz	176 MHz	170 MHz	168 MHz	161 MHz
16	128	200 MHz	169 MHz	169 MHz	168 MHz	167 MHz	167 MHz
32	8	280 MHz	215 MHz	173 MHz	167 MHz	167 MHz	149 MHz
32	16	233 MHz	207 MHz	176 MHz	172 MHz	168 MHz	164 MHz
32	32	217 MHz	221 MHz	171 MHz	169 MHz	168 MHz	168 MHz
32	64	212 MHz	185 MHz	168 MHz	167 MHz	167 MHz	146 MHz
32	128	198 MHz	171 MHz	168 MHz	168 MHz	167 MHz	167 MHz
64	8	235 MHz	179 MHz	170 MHz	167 MHz	146 MHz	139 MHz
64	16	220 MHz	196 MHz	162 MHz	167 MHz	149 MHz	138 MHz
64	32	189 MHz	202 MHz	173 MHz	169 MHz	167 MHz	131 MHz
64	64	194 MHz	178 MHz	173 MHz	168 MHz	168 MHz	167 MHz
64	128	205 MHz	171 MHz	168 MHz	167 MHz	167 MHz	167 MHz
128	8	174 MHz	173 MHz	168 MHz	152 MHz	132 MHz	132 MHz
128	16	194 MHz	169 MHz	169 MHz	144 MHz	140 MHz	140 MHz
128	32	192 MHz	168 MHz	170 MHz	167 MHz	136 MHz	136 MHz
128	64	233 MHz	180 MHz	167 MHz	168 MHz	152 MHz	152 MHz
128	128	197 MHz	197 MHz	168 MHz	169 MHz	167 MHz	167 MHz

Table 16: **Performance Benchmarking on XC2VP7 -6 for Distributed RAM Implementation with Length FIFO**

Write Data Width	Read Data Width	Depth					
		16	32	64	128	256	512
8	8	268 MHz	259 MHz	190 MHz	184 MHz	171 MHz	171 MHz
8	16	238 MHz	221 MHz	198 MHz	186 MHz	172 MHz	167 MHz
8	32	194 MHz	183 MHz	177 MHz	169 MHz	169 MHz	168 MHz
8	64	179 MHz	206 MHz	172 MHz	170 MHz	167 MHz	167 MHz
8	128	188 MHz	173 MHz	170 MHz	168 MHz	168 MHz	168 MHz
16	8	228 MHz	242 MHz	191 MHz	171 MHz	167 MHz	163 MHz
16	16	202 MHz	198 MHz	166 MHz	179 MHz	166 MHz	168 MHz
16	32	187 MHz	203 MHz	170 MHz	169 MHz	169 MHz	167 MHz
16	64	232 MHz	192 MHz	171 MHz	168 MHz	168 MHz	167 MHz
16	128	196 MHz	175 MHz	171 MHz	168 MHz	168 MHz	168 MHz
32	8	229 MHz	197 MHz	188 MHz	168 MHz	168 MHz	157 MHz
32	16	199 MHz	202 MHz	177 MHz	169 MHz	167 MHz	131 MHz
32	32	181 MHz	224 MHz	172 MHz	168 MHz	168 MHz	168 MHz
32	64	184 MHz	178 MHz	170 MHz	168 MHz	166 MHz	167 MHz
32	128	205 MHz	182 MHz	169 MHz	168 MHz	167 MHz	167 MHz
64	8	212 MHz	177 MHz	169 MHz	167 MHz	138 MHz	128 MHz
64	16	233 MHz	178 MHz	169 MHz	167 MHz	147 MHz	123 MHz
64	32	199 MHz	191 MHz	172 MHz	167 MHz	168 MHz	167 MHz
64	64	178 MHz	203 MHz	170 MHz	168 MHz	168 MHz	167 MHz
64	128	171 MHz	172 MHz	169 MHz	168 MHz	166 MHz	166 MHz
128	8	173 MHz	168 MHz	168 MHz	140 MHz	129 MHz	129 MHz
128	16	174 MHz	152 MHz	143 MHz	143 MHz	110 MHz	110 MHz
128	32	193 MHz	178 MHz	146 MHz	144 MHz	144 MHz	144 MHz
128	64	211 MHz	144 MHz	149 MHz	144 MHz	143 MHz	143 MHz
128	128	145 MHz	148 MHz	144 MHz	145 MHz	143 MHz	143 MHz

Applications

The LocalLink FIFO provides parameterizable data widths on read and write ports, and performs at up to 200 MHz. Hence, it is capable of a wide range of applications, such as 10/100/1000 Mbps Ethernet, 1G and 2G Fibre Channel, Aurora, 10 Gigabit Ethernet, and 10 Gigabit Fibre Channel. Table 17 and Table 18 show the application of the LocalLink FIFO in 1 Gigabit Ethernet and 10 Gigabit Ethernet.

Table 17: **1 Gigabit Ethernet Application Using LocalLink FIFO (Block SelectRAM Implementation)**

Data Width(1) (Bits)	Maximum FIFO Capacity (Bytes)	Clock Frequency (MHz)		Maximum Number of Packets(3)	Minimum Number of Packets(4)	Maximum Number of Jumbo Frames(5)
		Required	Delivered(2)			
8	32,767	125	191	511	21	3.6
16	32,767	62.5	200	511	21	3.6
32	32,767	31.25	182	511	21	3.6
64	65,535	15.625	170	1023	43	7.3
128	131,071	7.8125	170	1023	86	14.6

Notes:
1. Assumes read and write data widths are the same.
2. Data acquired from Table 11.
3. Measured by using minimum Ethernet frame size - 64 bytes, including the CRC bytes.
4. Measured by using maximum Ethernet frame size - 1518 bytes, including the CRC bytes.
5. Measured by using typical jumbo frame size - 9000 bytes.

Table 18: **10-Gigabit Ethernet Application Using LocalLink FIFO (Block SelectRAM Implementation)**

Data Width(1) (Bits)	Maximum FIFO Capacity (Bytes)	Clock Frequency (MHz)		Maximum Number of Packets(3)	Minimum Number of Packets(4)	Maximum Number of Jumbo Frames(5)
		Required	Delivered(2)			
64	65,535	156.25	170	1023	43	7.3
128	131,071	78.125	170	1023	86	14.6

Notes:
1. Assumes read and write data widths are the same.
2. Data acquired from Table 11.
3. Measured by using minimum Ethernet frame size - 64 bytes, including the CRC bytes.
4. Measured by using maximum Ethernet frame size - 1518 bytes, including the CRC bytes.
5. Measured by using typical jumbo frame size - 9000 bytes.

Reference Designs

The LocalLink FIFO reference design is available on the Xilinx website as VHDL files (http://www.xilinx.com/bvdocs/appnotes/xapp691.zip). Simulation testbenches and scripts are provided, as well as PERL scripts to generate the netlist and run the simulation.

Table 19 lists the VHDL source file names and the descriptions for the reference design. Table 20, page 29 lists the file names and descriptions for the simulation testbench. Table 21, page 29 lists the scripts and projects files.

Table 19: **VHDL Source Files**

Name	Description
ll_fifo.vhd	Top-level module. Instantiates ll_fifo_BRAM or ll_fifo_DRAM according to parameter settings.
ll_fifo_BRAM.vhd	Second top-level module that handles LocalLink signals and instantiates BRAM_fifo. It contains one LL_FIFO module.
ll_fifo_DRAM.vhd	Second top-level module that handles LocalLink signals and instantiates DRAM_fifo. It contains one LL_FIFO module.
BRAM_fifo.vhd	Low-level source code for block SelectRAM.
DRAM_fifo.vhd	Low-level source code for distributed RAM.
BRAM_fifo_pkg.vhd	Package for BRAM_fifo.
DRAM_fifo_pkg.vhd	Package for DRAM_fifo.
BRAM_macro.vhd	Source code to construct various block SelectRAM macros.
DRAM_macro.vhd	Source code to construct various distributed RAM macros.
Virtex2p.vhd	Package for Virtex-II Pro FPGA.
fifo_utils.vhd	FIFO utility file that contains functions for calculating constants for both BRAM_fifo and DRAM_fifo modules.
RAM_64nX1.vhd	A file that allows users to instantiate DRAM that has a data width more than 64 bits wide.
BRAM_S8_S72.vhd	Component for BRAM that instantiates a block that uses two BRAMs to have data conversion of 8 to 72 or vice versa.
BRAM_S8_S144.vhd	Component for BRAM that instantiates a block that uses four BRAMs to have data conversion of 8 to 144 or vice versa.
BRAM_S18_S72.vhd	Component for BRAM that instantiates a block that uses two BRAMs to have data conversion of 18 to 72 or vice versa.
BRAM_S16_S144.vhd	Component for BRAM that instantiates a block that uses two BRAM_S8_S72 to have data conversion of 16 to 144 or vice versa.
BRAM_S36_S144.vhd	Component for BRAM that instantiates a block that uses two BRAM_S18_S72 to have data conversion of 36 to 144 or vice versa.
BRAM_S36_S72.vhd	Component for BRAM that instantiates a block that uses two BRAMs to have data conversion of 36 to 72 or vice versa.
BRAM_S72_S72.vhd	Component for BRAM that instantiates a block that uses two BRAMs to have data conversion of 72 to 72 or vice versa.

Table 19: **VHDL Source Files *(Continued)***

Name	Description
BRAM_S72_S144.vhd	Component for BRAM that instantiates a block that uses two BRAM_S36_S72 to have data conversion of 72 to 144 or vice versa.
BRAM_S144_S144.vhd	Component for BRAM that instantiates a block that uses two BRAM_S72_S72 to have data conversion of 144 to 144 or vice versa.

Table 20: **Simulation Testbench Files**

Name	Description
ll_fifo.pl	PERL script for running LocalLink FIFO simulation.
ll_fifo_tb_wave.do	Waveform format file for LocalLink FIFO simulation.
FILEREAD_TESTER.v	Verilog code for testing the module that reads the vector file.
OUTPUT_TESTER.v	Verilog code for testing the module that outputs the testing vectors for all data widths except 8 bits.
OUTPUT_TESTER_8_BIT.v	Verilog code for testing the module that outputs the testing vectors for 8-bit data width.
TESTER_pkg.vhd	VHDL packages for the TESTER modules.
UFC_CONVERTER.v	Verilog code for the TESTER, dealing with flow control for all data widths except 8 bits.
UFC_CONVERTER_8_bit.v	Verilog code for the TESTER, dealing with flow control for only 8-bit data width.
user_data_packets8.vec	Test vector file for 8-bit data width.
user_data_packets16.vec	Test vector file for 16-bit data width.
user_data_packets32.vec	Test vector file for 32-bit data width.
user_data_packets64.vec	Test vector file for 64-bit data width.
user_data_packets128.vec	Test vector file for 128-bit data width.
ll_fifo_tb.vhd	Source code of the LocalLink FIFO testbench.

Table 21: **File Name and Description of Scripts and Project Files**

Name	Description
ll_fifo.xst, ll_fifo.prj	Project file and command option file for XST synthesis tool.
ll_fifo_run.pl	PERL script for generating the netlist.
config.csh	Shell script file for setting environment variables for the design.
modelsim_unix.ini	ModelSim initialization file for UNIX environment.
readme.txt	Readme file with instructions for simulating and implementing the design.

Design Tools

- Xilinx ISE 6.1.03i
- ModelSim SE 5.6e

Conclusion

The parameterizable LocalLink FIFO provides seamless connections to user applications that support LocalLink interfaces. The LocalLink FIFO implements data width conversion and supports asynchronous operation between the read and write ports. Virtex-II and Virtex-II Pro block SelectRAM memory or distributed RAM can be used to generate FIFO memory with parameterizable depths. The LocalLink FIFO can operate at speeds at around 200 MHz.

Reference

1. Xilinx, Inc., XAPP258: FIFOs Using Virtex-II Block RAM
2. Xilinx, Inc., XAPP261: Data-Width Conversion FIFOs Using the Virtex-II Block RAM Memory
3. Xilinx, Inc., SP006: LocalLink Interface. Click the **Access Lounge** button and follow the registration instructions to download the specification.
4. Xilinx, Inc., LogiCORE Data Sheet, DS232: Asynchronous FIFO v5.1

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
02/02/04	1.0	Initial Xilinx release.
05/10/07	1.0.1	Fixed link to Xilinx LocalLink lounge.