Soft Direct Memory Access (SDMA) (v1.00b)

DS607 December 20, 2007 **Product Specification**

Introduction

The Soft Direct Memory Access (SDMA), is designed to provide high-performance DMA for streaming data. The SDMA provides two channels, one for receiving data and one for transmitting data. Transmit and Receive is accomplished through two LocalLink interfaces. Also provided as a Native Port Interface (NPI) for interfacing directly to MPMC.

Features

- Direct plug in to the MPMC.
- Supports simultaneous independent Transmit and Receive DMA operations.
- Interrupt event reporting for each channel.
- Supports Interrupt Coalescing.
- Interfaces to Xilinx PLB v4.6 for register access.
- Supports user definable LocalLink Headers and Footers for use with functions such as checksum off loading.
- Supports Dynamic Scatter Gather Buffer Descriptor modification.

LogiCORE™ Facts		
Core Specifics		
Supported Device Family	Virtex™-4, Virtex-5, Spartan™, Spartan-3a, Spartan-3e,Spartan	
Version of core	sdma	v1.00b
Resources Used		
	Min	Max
Slices	738	844
LUTs	1535	1627
FFs	944	1092
Block RAMs	None	
Special Features	None	
Provided with Core		
Documentation	Product Specification	
Design File Formats	VHDL	
Constraints File	None	
Verification	None	
Instantiation Template	None	
Reference Designs & application notes	None	
Additional Items	None	
Design Tool Requirements		
Xilinx Implementation Tools	ISE 9.2 or later	
Verification	ModelSim PE 6.1d	
Simulation	ModelSim PE 6.1d	
Synthesis	XST	
Support		
Provided by Xilinx, Inc.		

Functional Description

Figure 1 illustrates a high-level block diagram of the SDMA. The SDMA utilizes an NPI Port Interface, two LocalLink interfaces, and a PLB Interface. The NPI Port Interface connects the SDMA into the MPMC's personality module interface. The two LocalLink interfaces, transmit and receive, providing full duplex LocalLink device access to the SDMA. The PLB Interface allows the CPU to interact with the SDMA for initiating DMA processes or for status gathering.

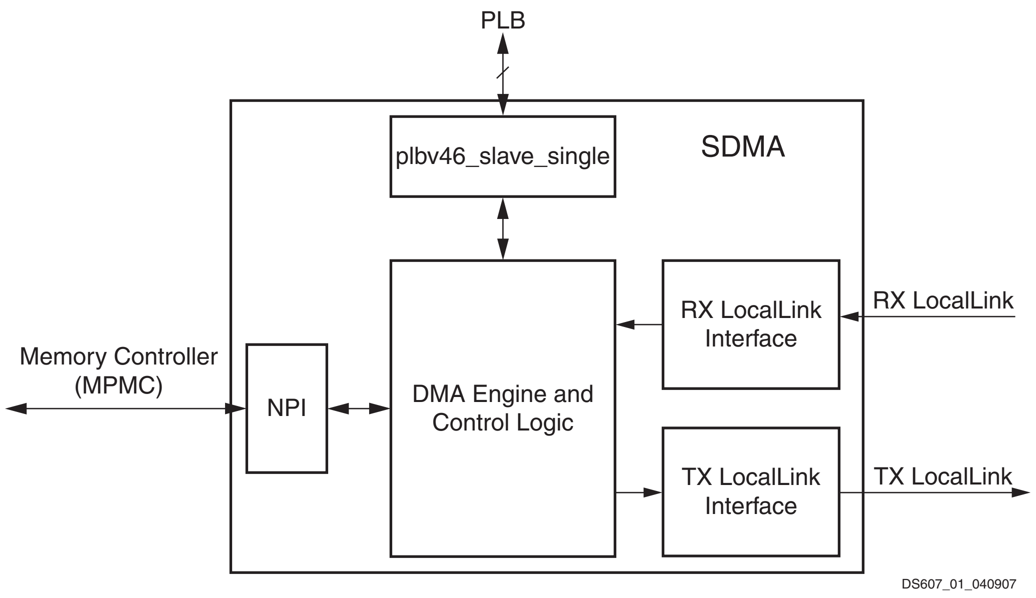

Figure 1: **SDMA Block Diagram**

I/O Signals

The SDMA signals are listed and described in Table 1.

Table 1: **SDMA I/O Signal Description**

Signal Name	Interface	Signal Type	Init Status	Description
PLB Interface				
SPLB_Clk	PLB Bus	I		PLB main bus clock. See table note 1.
SPLB_Rst	PLB Bus	I		PLB main bus reset. See table note 1.
PLB_ABus(0:**31**)	PLB Bus	I		See table note 1.
PLB_PAValid	PLB Bus	I		See table note 1.
PLB_masterID(0:**C_SPLB_MID_WIDTH**-1)	PLB Bus	I		See table note 1.
PLB_RNW	PLB Bus	I		See table note 1.
PLB_BE(0:[**C_SPLB_DWIDTH**/8]-1)	PLB Bus	I		See table note 1.
PLB_size(0:3)	PLB Bus	I		See table note 1.

Table 1: **SDMA I/O Signal Description**

Signal Name	Interface	Signal Type	Init Status	Description
PLB Interface				
PLB_type(0:2)	PLB Bus	I		See table note 1.
PLB_wrDBus(0:**C_SPLB_DWIDTH**-1)	PLB Bus	I		See table note 1.
Sl_addrAck	PLB Bus	O	0	See table note 1.
Sl_SSize(0:1)	PLB Bus	O	0	See table note 1.
Sl_wait	PLB Bus	O	0	See table note 1.
Sl_rearbitrate	PLB Bus	O	0	See table note 1.
Sl_wrDack	PLB Bus	O	0	See table note 1.
Sl_wrComp	PLB Bus	O	0	See table note 1.
Sl_rdBus(0:**C_SPLB_DWIDTH**-1)	PLB Bus	O	0	See table note 1.
Sl_rdDAck	PLB Bus	O	0	See table note 1.
Sl_rdComp	PLB Bus	O	0	See table note 1.
Sl_MBusy(0:**C_SPLB_NUM_MASTERS**-1)	PLB Bus	O	0	See table note 1.
Sl_MWrErr(0:**C_SPLB_NUM_MASTERS**-1)	PLB Bus	O	0	See table note 1.
Sl_MRdErr(0:**C_SPLB_NUM_MASTERS**-1)	PLB Bus	O	0	See table note 1.
Unused PLB Signals				
PLB_UABus(0:**31)**)	PLB Bus	I		Unused
PLB_SAValid	PLB Bus	I		Unused
PLB_rdPrim	PLB Bus	I		Unused
PLB_wrPrim	PLB Bus	I		Unused
PLB_abort	PLB Bus	I		Unused
PLB_busLock	PLB Bus	I		Unused
PLB_MSize(0:1)	PLB Bus	I		Unused
PLB_TAttribute(0 to 15)	PLB Bus	I		Unused
PLB_lockerr	PLB Bus	I		Unused
PLB_wrBurst	PLB Bus	I		Unused
PLB_rdBurst	PLB Bus	I		Unused
PLB_wrPendReq	PLB Bus	I		Unused
PLB_rdPendReq	PLB Bus	I		Unused
PLB_rdPendPri(0:1)	PLB Bus	I		Unused
PLB_wrPendPri(0,1)	PLB Bus	I		Unused

Table 1: **SDMA I/O Signal Description**

Signal Name	Interface	Signal Type	Init Status	Description
PLB Interface				
PLB_reqPri(0:1)	PLB Bus	I		Unused
Sl_wrBTerm	PLB Bus	O	0	Unused
Sl_rdWdAddr(0:3)	PLB Bus	O	0	Unused
Sl_rdBTerm	PLB Bus	O	0	Unused
Sl_MIRQ(0:**C_SPLB_NUM_MASTERS**-1)	PLB Bus	O	0	Unused
NPI Port Interface Signals				
PI_CLK	NPI	I		NPI Clock
PI_Addr(**C_PI_ADDR_WIDTH** - 1:0)	NPI	O	0	NPI Address Bus
PI_AddrReq	NPI	O	0	NPI Address Transfer Request 1=Request transfer 0=Normal state
PI_AddrAck	NPI	I		NPI Request Acknowledge 1=Request acknowledged 0=Normal state
PI_RNW	NPI	O	0	NPI Read/Not Write Control Signal 1= Read transfer 0= Write transfer
PI_RdModWr	NPI	O	0	NPI Read/Modify/Write Control 1=Unaligned transfer 0=Aligned transfer
PI_Size(3:0)	NPI	O	0	NPI Transfer Size 0010=8 Word Cache 0100=32 Word Burst (All others not used by SDMA)
PI_RdFIFO_Data(**C_PI_DATA_WIDTH** - 1:0)	NPI	I	0	NPI Read FIFO Data
PI_RdFIFO_Pop	NPI	O	0	NPI Read FIFO Pop Control
PI_RdFIFO_RdWdAddr(**C_PI_RDWDADDR_WIDTH** - 1:0)	NPI	I	0	NPI Read word address
PI_RdFIFO_DataAvailable	NPI	I	0	NPI Read FIFO Data Available flag
PI_RdFIFO_Empty	NPI	I		NPI Read FIFO Empty flag
PI_RdFIFO_Flush	NPI	O	0	NPI Read FIFO Flush control

Table 1: **SDMA I/O Signal Description**

Signal Name	Interface	Signal Type	Init Status	Description
PLB Interface				
PI_WrFIFO_Data(**C_PI_DATA_WIDTH** - 1:0)	NPI	O	0	NPI Write FIFO Data
PI_WrFIFO_BE(**C_PI_BE_WIDTH**-1:0)	NPI	O	0	NPI Write FIFO Byte Enables
PI_WrFIFO_Push	NPI	O	0	NPI Write FIFO Push control
PI_WrFIFO_AlmostFull	NPI	I		NPI Write FIFO Almost Full flag. Asserts when when PI_WrFIFO_Push=1 and the write fifo is almost full.
PI_WrFIFO_Empty	NPI	I		NPI Write FIFO Empty flag
PI_WrFIFO_Flush	NPI	O	0	Currently unused by SDMA. Tied to ground.
LocalLink System Interface				
LLink_Clk	LLINK	I		LocalLink Clock
Transmit LocalLink Interface				
TX_D(31:0)	LLINK	O	0	Transmit LocalLink Data Bus
TX_Rem(3:0)	LLINK	O	1	Transmit LocalLink Remainder Bus
TX_SOF	LLINK	O	1	Transmit LocalLink Start of Frame
TX_EOF	LLINK	O	1	Transmit LocalLink End of Frame
TX_SOP	LLINK	O	1	Transmit LocalLink Start of Payload
TX_EOP	LLINK	O	1	Transmit LocalLink End of Payload
TX_Src_Rdy	LLINK	O	1	Transmit LocalLink Source Ready
TX_Dst_Rdy	LLINK	I		Transmit LocalLink Destination Ready
Receive LocalLink Interface				
RX_D(31:0)	LLINK	I		Receive Local Link Data Bus
RX_Rem(3:0)	LLINK	I		Receive LocalLink Remainder Bus
RX_SOF	LLINK	I		Receive LocalLink Start of Frame
RX_EOF	LLINK	I		Receive LocalLink End of Frame

Table 1: **SDMA I/O Signal Description**

Signal Name	Interface	Signal Type	Init Status	Description
PLB Interface				
RX_SOP	LLINK	I		Receive LocalLink Start of Payload
RX_EOP	LLINK	I		Receive LocalLink End of Payload
RX_Src_Rdy	LLINK	I		Receive LocalLink Source Ready
RX_Dst_Rdy	LLINK	O	1	Receive LocalLink Destination Ready
SDMA System Interface				
SDMA_RxIntOut	SDMA	O	0	Receive interrupt output
SDMA_TxIntOut	SDMA	O	0	Transmit interrupt output
SDMA_RstOut	SDMA	O	0	Soft Reset Acknowledge

Note:

1. This function and timing of the signal is defined in the IBM® **128-Bit Processor Local Bus Architecture Specification Version 4.6.**

Design Parameters

The SDMA provides for User interface tailoring via VHDL Generic parameters. These parameters are detailed in Table 2.

Table 2: **SDMA Design Parameters**

Feature/Description	Parameter Name	Allowable Values	Default Values	VHDL Type
MPMC2 Specification				
MPMC Port Interface Base Address	C_PI_BASEADDR		0x00000000	std_logic_vector
MPMC Port Interface High Address	C_PI_HIGHADDR		0xFFFFFFFF	std_logic_vector
MPMC Port Interface Address Bus Width	C_PI_ADDR_WIDTH	32	32	integer
MPMC Port Interface Data Bus Width	C_PI_DATA_WIDTH	64	64	integer
MPMC Port Interface BE Bus Width	C_PI_BE_WIDTH	C_PI_DATA_WIDTH/8	8	integer
MPMC Port Interface Read Word Address Width	C_PI_RDWDADDR_WIDTH	4	4	integer
DMA Specification				

Table 2: **SDMA Design Parameters**

Feature/Description	Parameter Name	Allowable Values	Default Values	VHDL Type
Enable transmit complete with error checking	C_COMPLETED_ERR_TX	0 = Disable complete bit error checking 1 = Enable complete bit error checking	1	integer
Enable receive complete with error checking	C_COMPLETED_ERR_RX	0 = Disable complete bit error checking 1 = Enable complete bit error checking	1	integer
Interrupt Delay Timer Scale Factor	C_PRESCALAR	0 to 1023	1023	integer
Read Data PipeLine Delay	C_PI_RDDATA_DELAY	0 = No Read Data Pipeline stage 1= 1 Read Data Pipeline stage 2=2 Read Data Pipeline stages	0	integer
NPI To LocalLink clock Ratio	C_PI2LL_CLK_RATIO	1 = 1:1 Clock Ratio 2 = 2:1 Clock Ratio	1	integer
PLB Specification				
PLB Master ID Bus Width	C_SPLB_MID_WIDTH	$\log_2$(**C_SPLB_NUM_ MASTERS**) with a minimum value of 1[1]	1	integer
Number of PLB Masters	C_SPLB_NUM_MASTERS	1 to 16	1	integer
Width of the PLB Least Significant Address Bus	C_SPLB_AWIDTH	32	32	integer
Width of the PLB Data Bus	C_SPLB_DWIDTH	32, 64, 128	32	integer
Selects point-to-point or shared PLB topology for the PLB slave port.	C_SPLB_P2P	0 = PLB Shared Bus Topology 1 = PLB Point-to-Point Bus Topology	0	integer
Width of the Slave Data Bus	C_SPLB_NATIVE_DWIDTH	32	32	integer
Slave Attachment I/O Specification				
PLB Base Address	C_SDMA_BASEADDR	Must aligned to a word boundary	0x00000000	std_logic_vector
PLB High Address	C_SDMA_HIGHADDR	C_SDMA_BASEADDR + 0x43	0xFFFFFFFF	std_logic_vector

Table 2: **SDMA Design Parameters**

Feature/Description	Parameter Name	Allowable Values	Default Values	VHDL Type
FPGA Family Type				
Xilinx FPGA Family	C_FAMILY	spartan, spartan3a, spartan3e, virtex4, virtex5	virtex5	string

Note:

1. .$\log_2$ represents a logarithm function of base 2. For example, $\log_2(1)=0$, $\log_2(2)=1$, $\log_2(4)=2$, $\log_2(8)=3$, $\log_2(16)=4$, etc.

Allowable Parameter Combinations

Parameter - Port Parameter

Table 3: **SDMA Parameter-Port Dependencies**

Generic or Port	Name	Affects	Depends	Relationship Description
Design Parameters				
G1	C_SPLB_MID_WIDTH	P1	G3	The width of the Master ID Bus is set by the C_SPLB_MID_WIDTH parameter. C_SPLB_MID_WIDTH = log2(C_SPLB_NUM_MASTERS) with a minimum value of 1.
G2	C_SPLB_DWIDTH	P2, P3, P4		The width PLB Data buses as well as the BE bus are set by the C_SPLB_DWIDTH parameter.
G3	C_SPLB_NUM_MASTER S	P5, P6, P7,P8		The width of several PLB slave reply signals is set by the C_SPLB_NUM_MASTER parameter.
G4	C_PI_ADDR_WIDTH	P9		The width of the Native Port Interface Address Bus is set by the C_PI_ADDR_WIDTH parameter
G5	C_PI_DATA_WIDTH	P10,P11		The width of the Native Port Interface Read and Write Data Buses are set by the C_PI_DATA_WIDTH parameter
G6	C_PI_BE_WIDTH	P12		The width of the Native Port Interface BE Bus is set by the C_PI_BE_WIDTH parameter
G7	C_PI_RDWDADDR_WID TH	P13		The width of the Native Port Interface Read Word Address Bus is set by the C_PI_RDWDADDR_WIDTH parameter
G8	C_PI2LL_CLK_RATIO		P14, P15	The clock ratio between LLink_Clk and PI_Clk determines the setting C_PI2LL_CLK_RATIO. A setting of 1 indicates LLink_Clk = PI_Clk, a setting of 2 indicates PI_Clk = 2 x LLink_Clk, etc.
I/O Signals				

Table 3: **SDMA Parameter-Port Dependencies**

Generic or Port	Name	Affects	Depends	Relationship Description
P1	PLB_masterID(0:**C_SPLB_MID_WIDTH**-1)		G1	The width of the PLB Master ID bus is set by C_PLB_MID_WIDTH parameter.
P2	PLB_BE(0:[**C_SPLB_DWIDTH**/8]-1)		G2	The width PLB Data buses as well as the BE bus are set by the C_SPLB_DWIDTH parameter.
P3	PLB_wrDBus(0:**C_SPLB_DWIDTH**-1)		G2	The width PLB Data buses as well as the BE bus are set by the C_SPLB_DWIDTH parameter.
P4	PLB_rdDBus(0:**C_SPLB_DWIDTH**-1)		G2	The width PLB Data buses as well as the BE bus are set by the C_SPLB_DWIDTH parameter.
P5	Sl_mBusy(0:**C_SPLB_NUM_MASTERS**-1)		G3	The width of several PLB slave reply signals is set by the C_SPLB_NUM_MASTER parameter.
P6	Sl_MWrErr(0:**C_SPLB_NUM_MASTERS**-1)		G3	The width of several PLB slave reply signals is set by the C_SPLB_NUM_MASTER parameter.
P7	Sl_MRdErr(0:**C_SPLB_NUM_MASTERS**-1)		G3	The width of several PLB slave reply signals is set by the C_SPLB_NUM_MASTER parameter.
P8	Sl_MIRQ(0:**C_SPLB_NUM_MASTERS**-1)		G3	The width of several PLB slave reply signals is set by the C_SPLB_NUM_MASTER parameter.
P9	PI_Addr(**C_PI_ADDR_WIDTH** - 1:0)		G4	The width of the Native Port Interface Address Bus is set by the C_PI_ADDR_WIDTH parameter
P10	PI_WrFIFO_Data(**C_PI_DATA_WIDTH** - 1:0)		G5	The width of the Native Port Interface Read and Write Data Buses are set by the C_PI_DATA_WIDTH parameter
P11	PI_RdFIFO_Data(**C_PI_DATA_WIDTH** - 1:0)		G5	The width of the Native Port Interface BE Bus is set by the C_PI_BE_WIDTH parameter
P12	PI_WrFIFO_BE(**C_PI_BE_WIDTH** - 1:0)		G6	The width of the Native Port Interface BE Bus is set by the C_PI_BE_WIDTH parameter
P13	PI_RdFIFO_RdWdAddr(**C_PI_RDWDADDR_WIDTH** - 1:0)		G7	The width of the Native Port Interface Read Word Address Bus is set by the C_PI_RDWDADDR_WIDTH parameter
P14	LLink_Clk	G8		The ratio between LLink_Clk and PI_Clk determines the setting of C_PI2LL_CLK_RATIO.
P15	PI_Clk	G8		The ratio between LLink_Clk and PI_Clk determines the setting of C_PI2LL_CLK_RATIO.

Parameter Detailed Descriptions

C_PI_BASEADDR

This value is the NPI base address. This value along with C_PI_HIGHADDR is used for Current Descriptor Pointer, Next Descriptor Pointer, and Current Buffer Address error detection. This value must be set to the NPI base address of the port where this module connects.

C_PI_HIGHADDR

This value is the NPI high address. This value along with C_PI_HIGHADDR is used for Current Descriptor Pointer, Next Descriptor Pointer, and Current Buffer Address error detection. This value must be set to the NPI high address of the port where this module connects.

C_PI_ADDR_WIDTH

This value sets the width of the NPI address bus. This value must be set to 32.

C_PI_DATA_WIDTH

This value sets the width of the NPI data bus. This value must be set to 64.

C_PI_BE_WIDTH

This value sets the width of the NPI BE bus. This value must be set to C_PI_DATA_Width/8.

C_PI_RDWDADDR_WIDTH

This value sets the width of the NPI Read Word Address Bus. This value must be set to 4.

C_SDMA_BASEADDR

This value sets the base address for the PLB Slave Registers of the SDMA. The base address must be aligned to word (i.e. 32-bit) boundaries.

C_SDMA_HIGHADDR

This value sets the high address for the PLB Slave Registers of the SDMA. The range between C_SDMA_BASEADDR and C_SDMA_HIGHADDR must cover a minimum of 68 bytes, i.e. 0x00 to 0x43.

C_COMPLETED_ERR_TX

This parameter enables or disables error checking on transmit DMA transfer completion. Setting this value to a 1 will enable error checking and setting this value to a 0 will disable checking.

C_COMPLETED_ERR_RX

This parameter enables or disables error checking on receive DMA transfer completion. Setting this value to a 1 will enable error checking and setting this value to a 0 will disable checking.

C_PRESCALAR

This parameter is used to set the scale factor of the interrupt time-out delay timers. This value is only valid if one or both delays timers are instantiated.

C_PI_RDDATA_DELAY

This parameter is used to set the pipeline relationship between PI_RdFIFO_Pop and read data being available on the NPI. For example, the MPMC can be configured to work in a 0, 1, or 2 pipeline configuration. C_PI_RDDATA_DELAY must be set to match the configuration of the MPMC or whichever native port interface SDMA is attached to.

C_PI2LL_CLK_RATIO

This parameter specifies the ratio between the NPI clock domain and the LocalLink clock domain. The LocalLink clock domain is the native clock domain of the SDMA and this parameter is used to allow for a proper crossing at the NPI. Valid values for C_PI2LL_CLK_RATIO are 1 or 2 where a 1 says that the PI_Clk = LLink_Clk, and a 2 indicates that PI_Clk = 2 * LLink_Clk.

Note that the SDMA assumes that the NPI clock domain will be equal to or faster than the LocalLink clock domain.

C_SPLB_MID_WIDTH

This parameter is defined as an integer and has a minimum value of 1. It is equal to $\log_2$ of the number of PLB Masters connected to the PLB bus or 1, whichever is greater. It is used to size the PLB_masterID bus input from the PLB Bus to the Slave Attachment. For example, if eight PLB Masters are connected to the PLB Bus, then this parameter must be set to $\log_2(8)$ which is equal to 3. The PLB Bus PLB_masterID bus will be sized to 3 bits wide. If only one master exists, then the parameter needs to be set to 1.

C_SPLB_NUM_MASTERS

This parameter is defined as an integer and is equal to the number of Masters connected to the PLB bus. This parameter is used to size the Sl_MBusy and Sl_MErr slave reply buses to the PLB. For example, if eight PLB Masters are connected to the PLB Bus, then this parameter must be set to 8. The Sl_MBusy bus and Sl_MErr bus will be sized to 8 bits wide each.

C_SPLB_SMALLEST_MASTER

This parameter is defined as an integer and is equal to the native data width of the smallest Master connected to the PLB bus that will be accessing the plbv46_slave attachment. This generic is used to generate and optimize steering logic in the slave attachment.

C_SPLB_AWIDTH

This integer parameter is used by the PLB Slave to size the PLB address related components within the Slave Attachment. This value should be set 32.

C_SPLB_DWIDTH

This integer parameter is used by the PLB Slave to size PLB data bus related components within the Slave Attachment. This value should be set to match the actual width of the PLB bus, 32, 64 or 128-Bits.

C_SPLB_NATIVE_DWIDTH

This integer parameter is used to specify the native data width of the slave attachment. This parameter is used to interface various width Slave devices with various width PLB Buses. This parameter should be set to 32.

C_FAMILY

This parameter is defined as a string. It specifies the target FPGA technology for implementation of the PLB Slave. This parameter is required for proper selection of FPGA primitives. The configuration of these primitives can vary from one FPGA technology family to another.

Register Descriptions

SDMA Register Summary

Table 4: **SDMA Registers**

Register Name	PLB Address Offset from Service's Base Address Assignment	Allowed Access
Transmit Registers		
TX Next Descriptor Pointer (TX_NXTDESC_PTR)	0x00	Read
TX Current Buffer Address (TX_CURBUF_ADDR)	0x04	Read
TX Current Buffer Length (TX_CURBUF_LENGTH)	0x08	Read
TX Current Descriptor Pointer (TX_CURDESC_PTR)	0x0C	Read/Write
TX Tail Descriptor Pointer (TX_TAILDESC_PTR)	0x10	Read/Write
TX Channel Control (TX_CHNL_CTRL)	0x14	Read/Write
TX Interrupt Register (TX_IRQ_REG)	0x18	Read/Write
TX Status Register (TX_CHNL_STS)	0x1C	Read
Receive Registers		
RX Next Descriptor Pointer (RX_NXTDESC_PTR)	0x20	Read
RX Current Buffer Address (RX_CURBUF_ADDR)	0x24	Read
RX Current Buffer Length (RX_CURBUF_LENGTH)	0x28	Read
RX Current Descriptor Pointer (RX_CURDESC_PTR)	0x2C	Read/Write
RX Tail Descriptor Pointer (RX_TAILDESC_PTR)	0x30	Read/Write
RX Channel Control (RX_CHNL_CTRL)	0x34	Read/Write
RX Interrupt Register (RX_IRQ_REG)	0x38	Read/Write
RX Status Register (RX_CHNL_STS)	0x3C	Read
Control Register		
DMA Control Register (DMA_CONTROL_REG)	0x40	Read/Write

Register Details

Next Descriptor Pointer (TX_NXTDESC_PTR and RX_NXT_DESC_PTR)

(Offsets: 0x00 and 0x20)

The Next Descriptor Pointer, one for transmit and one for receive, is loaded from the value contained in the Next Descriptor Pointer field in the currently pointed to Descriptor for the respective channel. This value is kept in the respective SDMA register until the SDMA has completed all DMA transactions within the DMA transfer (reference Figure 3-19). After all DMA transactions are complete, the current Descriptor is complete, and the SDMA_COMPLETED bit is set in the respective STATUS_REGISTER, the current Descriptor is written to update the status of the STS_CTRL_APP0 field within the Descriptor.

After this, the SDMA evaluates whether there is a halt condition due to the Current Descriptor Pointer equaling the Tail Descriptor Pointer. If there is no halt condition then the address contained in the Next Descriptor Pointer register is evaluated.

- If a Null (0x00000000) is contained in the Next Descriptor Pointer register then SDMA engine stops processing Buffer Descriptors
- If the address contained in the Next Descriptor Pointer register is not 8-word aligned, or reaches beyond the range of available memory, the SDMA halts processing and sets the SDMA_ERROR bit in the respective status register (TX_CHNL_STS or RX_CHNL_STS)
- If the Next Descriptor Pointer register contains a valid address, then the contents are moved to the respective Current Descriptor Register (TX_CURDESC_PTR or RX_CUR_DESC_PTR). This movement causes the SDMA to begin another DMA transaction.

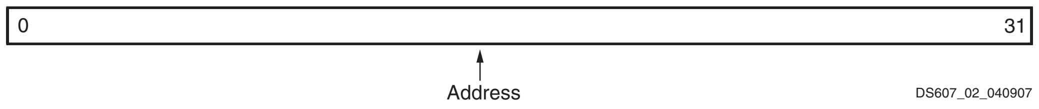

Figure 2: **Next Descriptor Pointer**

Table 5: **Next Descriptor Pointer Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-31	**TX_NXTDESC_PTR** and **RX_NXT_DESC_PTR**	Read	zeros	8 word aligned pointer to the next Buffer Descriptor in the chain. If NULL (0x0), DMA engine stops processing Buffer Descriptors.

Current Buffer Address (TX_CURBUF_ADDR and RX_CURBUF_ADDR)

(Offsets: 0x04 and 0x24)

The Current Buffer Address register, one for transmit and one for receive, maintains the contents of the address in memory where the DMA operation is conducted next. This value is originally loaded into the SDMA when the Descriptor is read by the SDMA. Once set by the current Buffer Descriptor, the SDMA then occasionally transfers this value to an internal Address Counter that then updates the value for each DMA transaction completed. Upon termination of the transaction, the SDMA overwrites the value of the Current Buffer Address register with the last value of the Address Counter. This process continues repeatedly until the SDMA has completed the current Descriptor. The reason for this mechanism is so the SDMA can maintain multiple temporal channels of DMA at a substantially reduced hardware cost. It is not recommended that software use the Current Buffer Address register to determine SDMA progress as it dynamically changes.

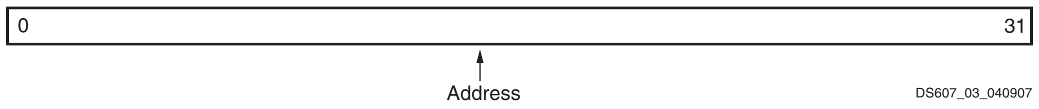

Figure 3: **Current Buffer Address**

Table 6: **Current Buffer Address Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-31	**TX_CURBUF_ADDR** and **RX_CUR_BUF_ADDR**	Read	zeros	Address to the current buffer being processed by SDMA.

Current Buffer Length (TX_CURBUF_LENGTH and RX_CURBUF_LENGTH)

(Offsets: 0x08 and 0x28)

The Current Buffer Length register, one for transmit and one for receive, maintains the contents of the remaining length of the data to be transferred by the SDMA. The value is originally loaded into the SDMA when the Descriptor is read by the SDMA. Once set by the current Descriptor, the SDMA then occasionally transfers this value to an internal Length Counter, which then updates the value for each DMA transaction completed. Upon termination of the transaction, the DMAC overwrites the value of the Current Buffer Length register with the last value of the internal Length Counter. This process continues repeatedly until the SDMA has completed the current Descriptor. The reason for this mechanism is so the SDMA can maintain multiple temporal channels of DMA at a substantially reduced hardware cost. However, software can find this mechanism useful for identifying where in a DMA operation the SDMA is. It is not recommended that software avail itself of this.

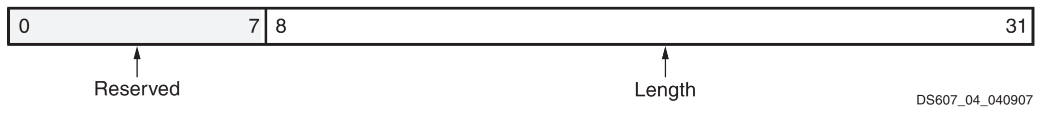

Figure 4: **Current Buffer Length**

Table 7: **Current Buffer Length Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-31	**TX_CURBUF_LENGTH** and **RX_CURBUF_LENGTH**	Read	zeros	Length in bytes of the current buffer being processed by SDMA.

Current Descriptor Pointer (TX_CURDESC_PTR and RX_CURDESC_PTR)

(Offsets: 0x0C and 0x2C)

The Current Descriptor Pointer register, one for transmit and one for receive, maintains the pointer to the Buffer Descriptor that is currently being processed. The value was set either by the CPU when it first initiated a DMA operation, or is copied from the Next Descriptor Pointer register upon completion of the prior Descriptor. This value is maintained by the SDMA as a pointer so that the SDMA can update the Descriptor's Status and Application Dependent fields once the Buffer Descriptor has been fully processed.

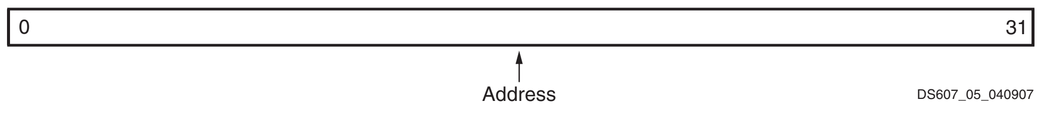

Figure 5: **Current Descriptor Pointer**

Table 8: **Current Descriptor Pointer Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-31	**TX_CURDESC_PTR** and **RX_CURDESC_PTR**	Read/Write	zeros	8 word aligned pointer to the current Buffer Descriptor being processed by the SDMA

Tail Descriptor Pointer (TX_TAILDESC_PTR and RX_TAILDESC_PTR)

(Offsets: 0x10 and 0x30)

The Tail Descriptor Pointer register, one for transmit and one for receive, maintains the pointer to the Buffer Descriptor chain tail. For this version of SDMA Tail Pointer Mode is always enabled thus DMA operations will halt when processing of the Buffer Descriptor pointed to by TAILDESC_PTR is completed. Writing to this register will start DMA operations.

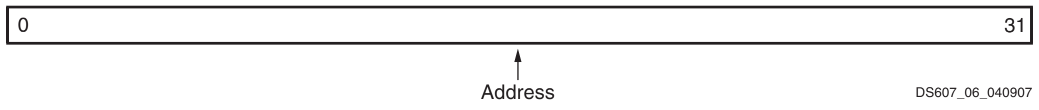

Figure 6: **Tail Descriptor Pointer**

Table 9: **Tail Descriptor Pointer Register Description**

Bit(s)	Name	Core Access	Reset Value	Description
0-31	**TX_TAILDESC_PTR** and **RX_TAILDESC_PTR**	Read/Write	zeros	8 word aligned pointer to the tail Descriptor. The Software Application writes to this field to kick off a DMA transfer.

Channel Control Register (TX_CHNL_CTRL and RX_CHNL_CTRL)

(Offsets: 0x14 and 0x34)

The Channel Control register, one for transmit and one for receive, controls interrupt processing for the particular channel.

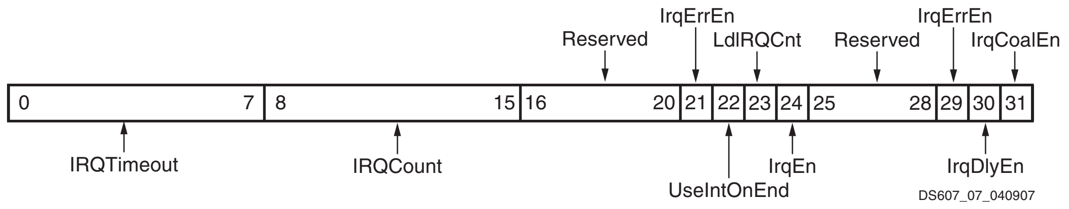

Figure 7: **Channel Control Register**

Table 10: **Channel Control Register**

Bit(s)	Name	Core Access	Reset Value	Description
0 to 7	IRQTimeout	Read/Write	0	**Interrupt Delay Time-out Value**. The maximum amount of time that an unreported packet is required to wait until generating a Delay Interrupt (DlyIRQ) event. (Must remain unchanged for the duration of an DMA operation.)
8 to 15	IRQCount	Read/Write	0	**Interrupt Coalescing Threshold Count Value**. The number of packets that must be received to generate a Coalescing Interrupt (ClscIrq) event. This value is loaded into the packet threshold counter when LdIrqCnt = ’1’ and subsequently re-loaded whenever the threshold count is reached.
16 to 20	Reserved			Reserved - read as zero
21	Use1BitCnt	Read/Write	0	**Use 1 Bit Wide Counters.** Currently Not Used
22	UseIntOnEnd	Read/Write	0	**Use Interrupt On End.** Selects between using the interrupt-on-end mechanism or using the EOP mechanism for interrupt coalescing. 1 - Selects the interrupt-on-end mechanism 0 - Selects the EOP mechanism
23	LdIRQCnt	Write	0	**Load IRQ Count.** Writing a 1 to this field forces the loading of the Interrupt Coalescing counters from the CHANNEL_CTRL.IrqCount[0:7] field. Self-clearing field. read as zero
24	IrqEn	Read/Write	0	**Master Interrupt Enable.** When set, indicates that the DMA channel is enabled to generate interrupts. This is the "master" enable for the channel. Individual sources can be enabled/disabled separately.
25 to 28	Reserved			Reserved - read as zero
29	IrqErrEn	Read/Write	0	**Interrupt on Error Enable**. When set, indicates that an interrupt will be generated if an error occurs
30	IrqDlyEn	Read/Write	0	**Interrupt on Delay Enable**. When set, indicates that an interrupt will be generated when the time-out value is reached.
31	IrqCoalEn	Read/Write	0	**Interrupt on Count Enable**. When set, indicates that an interrupt will be generated when the interrupt coalescing threshold value is reached.

Interrupt Status Register (TX_IRQ_REG and RX_IRQ_REG)

(Offsets: 0x18 and 0x38)

The Interrupt Status register, one for transmit and one for receive, indicates interrupt pending and interrupt coalescing count values. This register also is used by the Software application to acknowledge pending interrupts by writing a 1 to clear the pending interrupts.

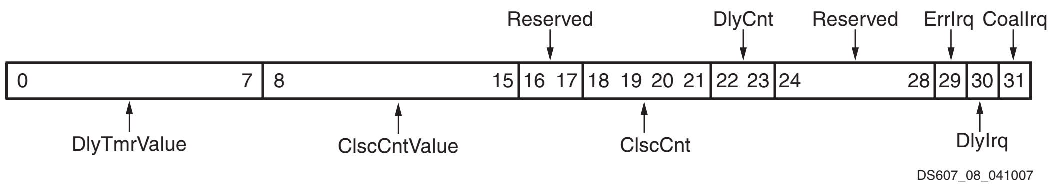

Figure 8: **Interrupt Status Register**

Table 11: **Interrupt Status Register**

Bit(s)	Name	Core Access	Reset Value	Description
0 to 7	DlyTmrValue	Read	0	Delay Timer Value. This field contains the real time delay timer value.
8 to 15	ClscCntrValue	Read	FF	Coalesce Counter Value. This field contains the real time coalesce counter value.
16 to 17	Reserved			Reserved - read as zero.
18 to 21	ClscCnt	Read	0	Coalesce Interrupt Count. Indicates the number of events due to reaching the interrupt coalesce threshold.
22 to 23	DlyCnt	Read	0	Delay Interrupt Count. Indicates the number of events due to reaching the wait bound delay time.
24 to 28	Reserved			Reserved - read as zero
29	ErrIrq	Read/Write	0	Error Interrupt Event. Indicates that an error has occurred. Writing a ’1’ to this bit will clear the interrupt.
30	DlyIrq	Read/Write	0	Delay Interrupt Event. Indicates that delay time-out event has occurred. Writing a ’1’ to this bit will clear the interrupt.
31	CoaIrq	Read/Write	0	Coalesce Interrupt Event. Indicates that an interrupt event threshold count has been reached. Writing a ’1’ to this bit will clear the interrupt.

Channel Status Register (TX_CHNL_STS and RX_CHNL_STS)

(Offsets: 0x1C and 0x3C)

The Channel Status register, one for transmit and one for receive, contains status for a particular channel.

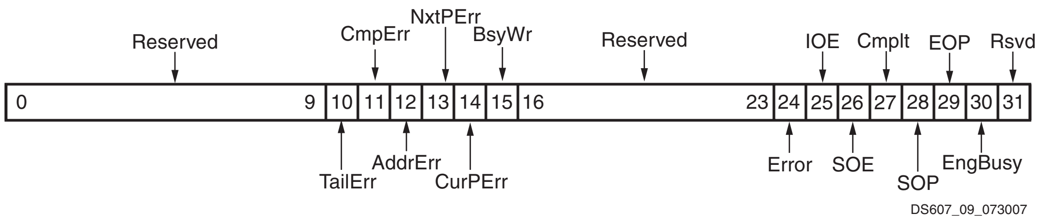

Figure 9: **Channel Status Register**

Table 12: **Channel Status Register**

Bit(s)	**Name**	**Core Access**	**Reset Value**	**Description**
0 to 9	Reserved			Reserved - read as zero.
10	TailPErr	Read	0	Tail Pointer Error This bit indicates that Tail Pointer is NOT a valid address. Valid addresses are between C_PI_BASEADDR and C_PI_HIGHADDR.
11	CmpErr	Read	0	Complete Error This bit indicates a Descriptor was fetched with the Cmplt=1 in the STS_CNTRL_APP0 field of the Descriptor. This error check is enabled by setting C_COMPLETED_ERR_RX and/or C_COMPLETED_ERR_TX to 1 for the respective channel.
12	AddrErr	Read	0	Address Error This bit indicates the Current Buffer Address is NOT a valid address. Valid addresses are between C_PI_BASEADDR and C_PI_HIGHADDR.
13	NxtPErr	Read	0	Next Descriptor Pointer Error This bit indicates the Next Descriptor Pointer is NOT a valid address. Valid addresses are between C_PI_BASEADDR and C_PI_HIGHADDR.
14	CurPErr	Read	0	Current Descriptor Pointer Error This bit indicates the Current Descriptor Pointer is NOT a valid address.Valid addresses are between C_PI_BASEADDR and C_PI_HIGHADDR.
15	BsyWr	Read	0	Busy Write Error This bit indicates the Current Descriptor Pointer register was written to while the DMA Engine was busy.
16 to 23	Reserved			Reserved - read as zero
24	Error	Read	0	DMA Error This bit indicates that an error occurred during DMA operations. This bit is an OR'ing of error bits 10 to 15.
25	IOE	Read	0	Interrupt On End This bit is a copy of the corresponding bit in the STS_CTRL_APP0 field of the Descriptor.

Table 12: **Channel Status Register**

Bit(s)	Name	Core Access	Reset Value	Description
26	SOE	Read	0	Stop On End This bit is a copy of the corresponding bit in the STS_CTRL_APP0 field of the Descriptor.
27	Cmplt	Read	0	Complete When set indicates that the DMA has transferred all data defined by the current Descriptor
28	SOP	Read	0	Start of Packet When set indicates that the current Descriptor is the start of a packet. For transmit the CPU sets this bit in the Descriptor to indicate that this is the first Descriptor of a packet to be transmitted. For receive, when an SOP is received by the Local Link interface, the DMA sets this bit in the Descriptor. This informs the CPU that this Descriptor is the first Descriptor of the packet.
29	EOP	Read	0	End of Packet When set, indicates that the current Descriptor is the final one of a packet. For transmit the CPU sets this bit in the Descriptor to indicate that this is the last Descriptor of a packet to be transmitted. For receive, when a EOP is received by Local Link interface, the DMA sets this bit in the Descriptor. This informs the CPU that the current Descriptor is the first of a received packet.
30	EngBusy	Read	0	Engine Busy When set, indicates that the respective channel is busy with a DMA operation. In general, software should not write any DMA registers while this bit is set. Reading of registers is allowed.
31	Reserved	Read		Reserved - read as zero.

DMA Control Register

(Offset: 0x40)

The DMA Control Register controls over all DMA operation.

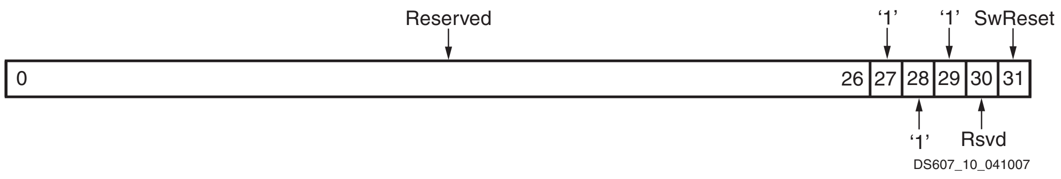

Figure 10: **DMA Control Register**

Table 13: **DMA Control Register**

Bit(s)	Name	Core Access	Reset Value	Description
0 to 26	Reserved		0	Reserved - read as zero.
27	Reserved		1	Reserved - read as one
28	Reserved		1	Reserved - read as one
29	Reserved		1	Reserved - read as one
30	Reserved		0	Reserved - read as zero.
31	SwReset	Read/Write	0	Software Reset Writing a 1 to this field forces the DMA engine to shutdown and reset itself. After setting this bit, software must poll it until the bit is cleared by the DMA. This indicates that the reset process is done and the pipeline has been flushed 1=Reset DMA - Resets both Rx and Tx Channels 0=Normal Operation (default)

SDMA Operation

Descriptors

SDMA operation requires a common memory-resident data structure that holds the list of DMA operations to be performed. This list of instructions are organized into what is referred to as a Descriptor chain. The Descriptor shown in Table 14 is the basis for organizing the DMA operations as a linked list. Descriptors are fetched through the NPI. A similar mechanism is used for performing Descriptor updates.

Table 14: **Descriptor**

NXTDESC_PTR	Next Descriptor Pointer	Indicates where in memory the next descriptor should be fetched.
CURBUF_ADDR	Buffer Address	Indicates where in memory the buffer for receiving or transmitting data is.
CURBUF_LENGTH	Buffer Length	For transmit, indicates the amount of data in bytes that are to be transmitted. For receive, indicates the amount of space in bytes that is available to receive data.
STS_CTRL_APP0	Status/Control and App Data 0	Status/Control for controlling and status'ing the DMA transfer. For transmit, App0 is used for application specific data. For receive, App0 is not used.
APP1	Application Data 1	Application specific data
APP2	Application Data 2	Application specific data
APP3	Application Data 3	Application specific data
APP4	Application Data 4	Application specific data

Table 15: **STS_CTRL_APP0**

Bit(s)	Name	Type	Description
0	Error	Status	This bit will be set by the DMA when a error is encountered.
1	IrqOnEnd	Control	When set, causes the DMA to generate an interrupt event when the current descriptor has been completed
2	Reserved	N/A	Undefined
3	Completed	Status	When set, indicates that the current descriptor has been completed.
4	SOP	Status/Control	Transmit Channel: (Control) Set by software in the descriptor indicating this Buffer Descriptor is the first Descriptor of a packet. Receive Channel: (Status) Set by DMA in the descriptor indicating that a start of packet was received on Local Link
5	EOP	Status/Control	Transmit Channel: (Control) Set by software in the descriptor indicating this Buffer Descriptor is the last Descriptor of a packet. Receive Channel: (Status) Set by DMA in the descriptor indicating that an end of packet was received on Local Link
6	EngBusy	Status	When set, indicates that the DMA is processing buffer descriptors. This bit is simply a copy of the corresponding bit in the CHANNEL_STS register.
7	Reserved	N/A	Undefined
8 to 31	Application Data 0	N/A	Application specific data

Each field of the Descriptor is four bytes in length and corresponds to either one of the DMA channel registers or User Application Fields.

For transmit channels, the Application Data Fields (App0 to App1) of the first Descriptor are transmitted as part of the Header of the Local Link Transmit Data stream. For receive channels, the Application Data Fields of the last buffer descriptor will be updated with a portion of the Footer of the Local Link Receive Data stream. (See "Local Link Headers and Footers" on page 29 for more information about Local Link Headers and Footers).

Figure 11 shows Descriptors organized into a linked list. SDMA will successively perform the DMA operations specified in the Descriptors up to and including the Descriptor with the CURDESC_PTR = TAILDESC_PTR.

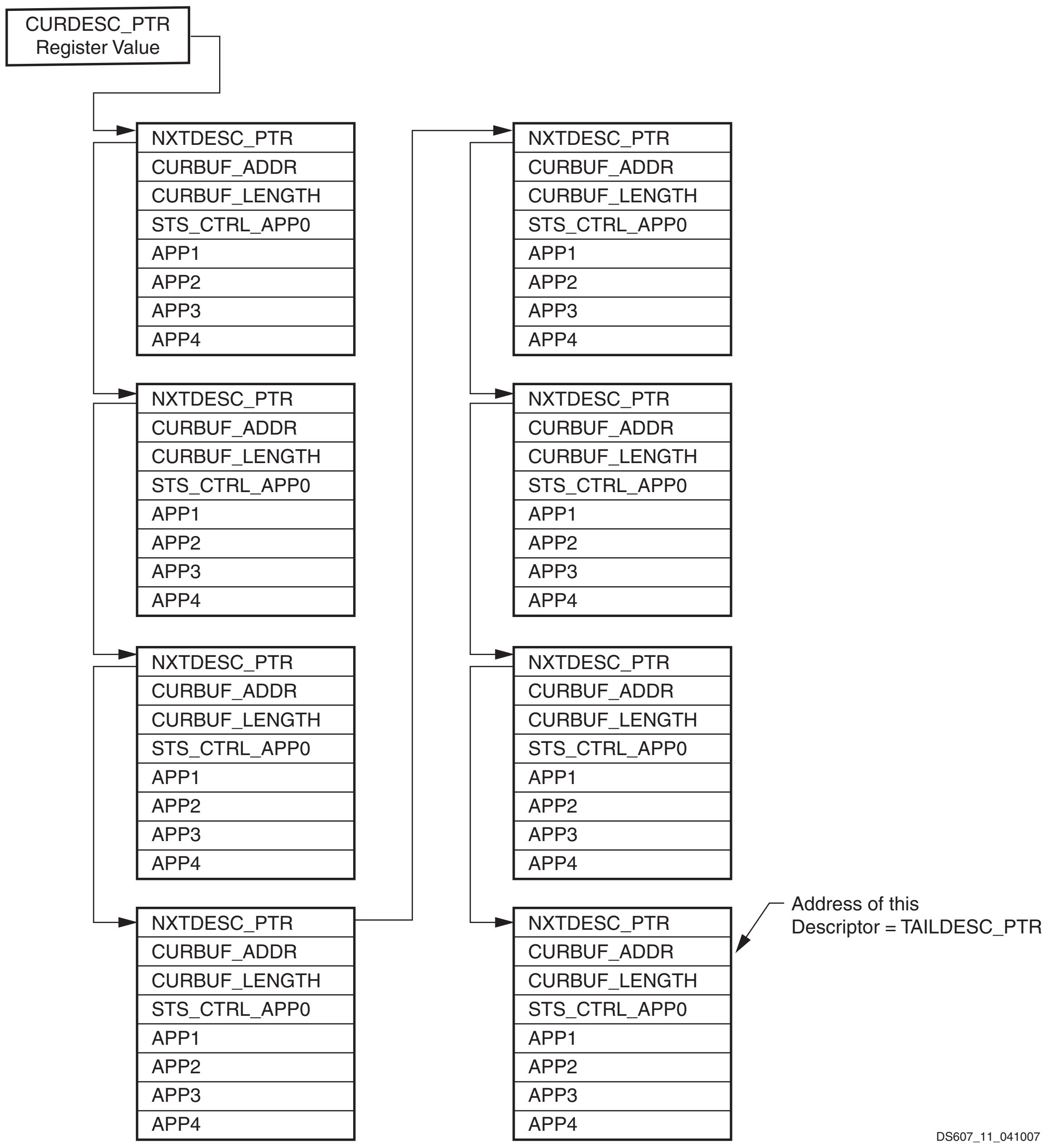

Figure 11: **Linked List of Descriptors**

Figure 12 shows Descriptors organized into a buffer ring. The buffer ring is for a Transmit channel as evidenced by STS_CTRL_APP0.SOP=1 and STS_CTRL_APP0.EOP=1 tags. Note that packet 4 is specified by a single Descriptor and others by more than one consecutive Descriptor. The last ready packet's address is equal to the TAILDESC_PTR, giving a sentinel position in the ring. Also note that even if Descriptors are contiguously allocated, they are required to be linked through the NXTDESC_PTR field. It should be noted that for receive channels STS_CTRL_APP0.SOP and STS_CTRL_APP0.EOP are set by SDMA and updated to memory for use by the Software application.

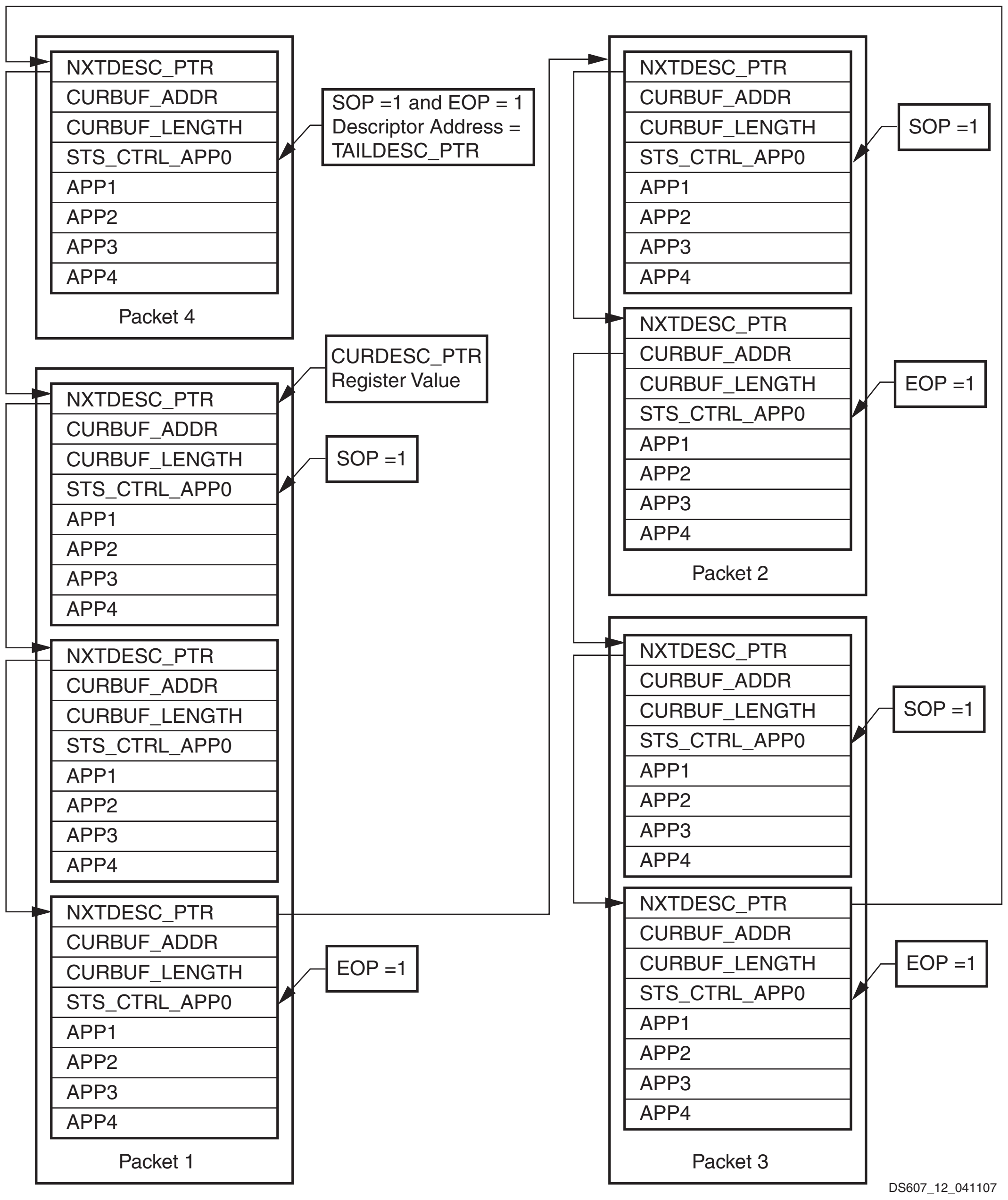

Figure 12: **Descriptors Organized Into a Buffer Ring**

Scatter Gather Operation

Scatter Gather operation has the concept of Descriptor Chaining which allows a packet to be described by more than one Descriptor. Typical use for this feature is to allow storing or fetching of Ethernet Headers from one location in memory and payload data from another location. Software applications that can take advantage of this can improve though put. To delineate packets in a buffer descriptor chain the Start of Packet bit (SOP) and End of Packet bit (EOP) are utilized. When the DMA fetches a Descriptor with the STS_CTRL_APP0.SOP bit set this will trigger the start of a packet. The packet will continue with the fetching of subsequent Descriptors until a Descriptor with the STS_CTRL_APP0.EOP bit is set.

For the receive channel, when a packet has been completely received the SDMA acquires the footer fields of the Local Link stream and writes these values to APP0 through APP5 fields of the last Descriptor. SDMA also sets STS_CTRL_APP0.EOP=1 indicating to the software that the current receive buffer as described by the Descriptor contains the last of the packet data.

For the transmit channel, SDMA uses the STS_CTRL_APP0.EOP bit as discussed above. When SDMA determines that the STS_CTRL_APP0.EOP bit is set in STS_CTRL_APP0 the SDMA will complete the currently requested transfer and then terminate the Local Link transfer with a LocalLink End of Frame, EOF.

Transmit Channel Operation

The following is a list of the steps required/performed to execute a packet transmit operation.

1. Software creates a chain of Descriptors
   a. Specify in the Descriptor the packet boundaries using the STS_CTRL_APP0.SOP and STS_CTRL_APP0.EOP bits.
   b. Specify the address to the data buffer to transmit in the CURBUF_ADDR Field.
   c. Specify the amount of data to transfer for each Descriptor in the CURBUF_LENGTH Field
   d. Specify a pointer to the next Descriptor in NXTDESC_PTR
2. Software prepares the DMA Channel Registers (Order of steps ’a’ and ’b’ are not critical),
   e. Set up Interrupts if so desired by writing to the TX_CHNL_CTRL register, specifying interrupt coalescing information if enabled.
   f. Set a pointer to the first Descriptor in the TX_CURDESC_PTR register
3. Software starts SG Automation by writing the pointer to the last Descriptor to fetch into the TAILDESC_PTR register.
4. SDMA will request the first Descriptor pointed to by the TX_CURDESC_PTR register.
5. Upon completion of the Descriptor fetch the DMA cycle will begin.
6. If the currently fetched Descriptor has STS_CTRL_APP0.EOP set then the data of that descriptor will be transmitted and the Master will complete the packet on the LocalLink with and End of Payload, EOP, and End of Frame, EOF. If the currently fetched Descriptor does NOT have STS_CTRL_APP0.EOP set then the packet will continue.
7. At the completion of each Descriptor the channel register information will be updated to the corresponding Descriptor memory location.
8. This process will continue until the Descriptor TX_CURDESC_PTR = TX_TAILDESC_PTR is completed processing.

Receive Channel Operation

The following is a list of the steps required/performed to execute a receive operation.

1. Software creates a chain of Descriptors. Note SDMA supports multiple Descriptors being used to describe a single packet. The STS_CTRL_APP0.EOP bit will be set by SDMA in the Descriptor associated to the buffer containing the last byte of the received packet.
   a. Specify in the CURBUF_ADDR field of each Descriptor the address to the start of the associated buffer for receiving data.
   b. Specify in the CURBUF_LENGTH field of each Descriptor the available size of the associated buffer for receiving data. The sum total of the Length field/s in the Descriptors must specify a byte count that is large enough to hold an entire packet.
   c. Specify a pointer to the next Descriptor in NXTDESC_PTR Field
9. Software prepares the DMA Channel Registers (Order of steps 'a' and 'b' are not critical),
   d. Set the pointer to the first Descriptor in RX_CURDESC_PTR
   e. Set up Interrupts if so desired by writing to the RX_CHNL_CTRL register, specifying interrupt coalescing information if enabled
10. Software starts SG Automation by writing the pointer to the last Descriptor to fetch into the RX_TAILDESC_PTR register.
11. CMDAC will request the first Descriptor pointed to by the RX_NXTDESC_PTR register. For receive channels the User Application fields will be updated in the Descriptor during the Descriptor update phase of processing.
12. Upon completion of the Descriptor fetch, the DMA cycle will begin.
13. This process will continue until the Descriptor RX_CURDESC_PTR = RX_TAILDESC_PTR is completed processing.

Stopping and Starting DMA Operation

Starting DMA Operation

DMA operations can be started writing an address to the respective TAILDESC_PTR register. When the start condition is met, CHNL_STS.EngBusy of the respective channel, will be set and the SDMA will fetch the first Descriptor pointed to by the address in respective CURDESC_PTR register.

Stopping Operation

DMA processing of Descriptors will continue until finished processing a descriptor that has the TAILDESC_PTR = CURDESC_PTR for the respective channel.

Error Conditions

SDMA performs several error checking functions to ensure proper operation of the DMA engine. If an error occurs then the channel on which the error is detected is halted and the channel status register Error bit for the channel is set to 1. If possible the Error bit for the current descriptor will also be set to 1 though depending on the error condition this may not get updated to remote memory.

To recover from an error condition the SDMA must be reset either by driving a hard reset or by issuing a soft reset (i.e. Set SwReset = 1 in the DMA Control Register, Offset 0x0x40).

The following lists the possible errors that can be flagged and their causes.

- Current Descriptor Pointer Error (TX_CHNL_STS.CurPErr and RX_CHNL_STS.CurPErr) - This error occurs if the Current Descriptor Pointer does not fall within the C_PI_BASEADDR to C_PI_HIGHADDR range. Descriptors must reside within memory as mapped by the base address and high address of the NPI. Addresses outside this range are detected and flagged as errors.
- Tail Descriptor Pointer Error (TX_CHNL_STS.TailPErr and RX_CHNL_STS.TailPErr) - This error occurs if the Tail Descriptor Pointer does not fall within the C_PI_BASEADDR to C_PI_HIGHADDR range. Descriptors must reside within memory as mapped by the base address and high address of the NPI. Addresses outside this range are detected and flagged as errors.
- Next Descriptor Pointer Error (TX_CHNL_STS.NxtPErr and RX_CHNL_STS.NxtPErr) - This error occurs if the Next Descriptor Pointer does not fall within the C_PI_BASEADDR to C_PI_HIGHADDR range. Descriptors must reside within memory as mapped by the base address and high address of the NPI Addresses outside this range are detected and flagged as errors.
- Buffer Address Error (TX_CHNL_STS.AddrErr and RX_CHNL_STS.AddrErr) - This error occurs if the Buffer Address does not fall within the C_PI_BASEADDR to C_PI_HIGHADDR range. All transmit and receive data buffers must reside within memory as mapped by the base address and high address of the NPI. Addresses outside this range are detected and flagged as errors.
- Complete Bit Error (TX_CHNL_STS.CmpErr and RX_CHNL_STS.CmpErr) - This error occurs if a Descriptor is fetched with the Complete bit set to 1 (i.e. STS_CTRL_APP0.Cmplt=1). This error is an indication that a Descriptor, which had been already used by SDMA, is being processed again, before the software application has had a chance to process the descriptor and associated data buffer. This error checking can be disabled or enabled by setting/clearing C_COMPLETED_ERR_TX and C_COMPLETED_ERR_RX for the associated channel. Setting the parameters to 1 enables checking and setting the parameters to 0 disables checking.
- Busy Write Error (TX_CHNL_STS.BsyWr and RX_CHNL_STS.BsyWr) - This error occurs if the Current Descriptor Pointer register is written to via the PLB v4.6 slave port while the SDMA engine is busy. Examining TX_CHNL_STS.EngBusy and RX_CHNL_STS.EngBusy for the respective channel will indicate to the software application whether or not the channel is busy. If EngBusy = 1 then the channel is busy and The Current Descriptor Pointer should not be written to by the software application.

Management of Descriptors

Prior to starting DMA operations, the software application must set up a Descriptor or chain of Descriptors. Once the SDMA begins processing the Descriptors, it will fetch, process, and then update the Descriptors. By analyzing the Descriptors, the software application can read status on the associated DMA transfer, fetch user information on receive channels and determine completion of the transfer. With this information the software application can manage the Descriptors and data buffers.

Once a Descriptor has been processed by the SDMA the STS_CTRL_APP0 field will be updated with status concerning the transfer. When the software application sets up the Descriptor chain the Status bits in the Control/Status field of each Descriptor must be set to zero. The status bits are, Error, Complete, and EngBusy and for receive channels SOP and EOP are status bits and must be set to zero. For transmit channels SOP and EOP are control bits set by the Software Application. This will allow the software application to easily determine when a Descriptor has been processed and whether or not there was an errors during processing.

As each Descriptor is updated into remote memory, STS_CTRL_APP0.Completed bit will be set to 1. By looking at the STS_CTRL_APP0.Completed bit and walking through the Descriptor chain, the software application can determine which Descriptors have been completed and which ones have not. Figure 13 shows remote memory where software has constructed a Descriptor chain. The SDMA, shown on the right, fetches Descriptors, processes them, and then updates the Descriptor in remote memory providing status. As can be seen STS_CTRL_APP0.Completed=1 for all of the Descriptors that have been processed.

For further clarity, on receive channels, by monitoring the STS_CTRL_APP0.Completed bit in the Descriptor, the software application can determine which data buffers, as described by the Descriptor, have received data and need to be processed. By looking for STS_CTRL_APP0.SOP and STS_CTRL_APP0.EOP the Software application can determine the start and end buffers containing a packet.

For transmit channels, STS_CTRL_APP0.Completed=1 indicates to the software application that the data in the associated buffer has been transmitted and is free to be modified.

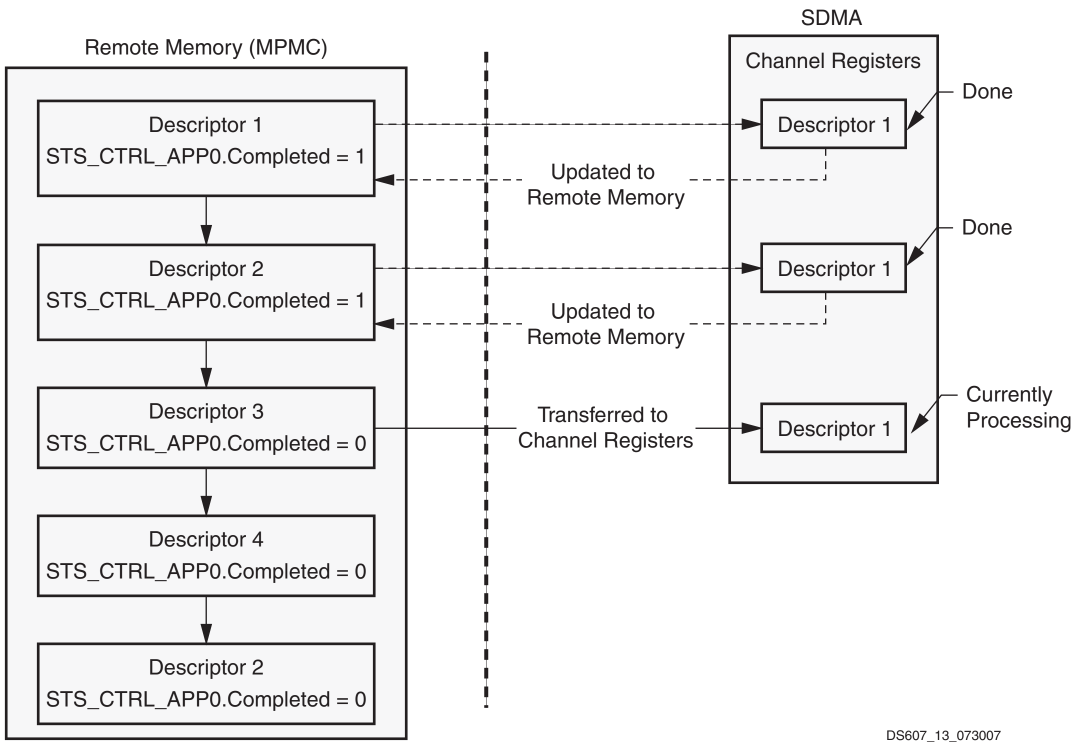

Figure 13: **Software - Hardware Descriptor Processing**

If an error occurs during the DMA Transfer, either during the Descriptor fetch/update or during the actual requested DMA transfer, the STS_CTRL_APP0.Error bit will be set. If an error occurs during a transfer, an IRQ_REG.ErrIrq interrupt will be generated, the respective CHNL_STS.EngBusy will be cleared to 0, and DMA operations will be halted. At this point the Software application must issue a reset to the SDMA to reset and resume DMA operations. This is done by writing a 1 to DMA_CONTROL.SwReset bit.

Local Link Headers and Footers

SDMA utilizes Local Link Headers and Footers to pass data in and out of the Buffer Descriptor User Application Fields. This allows the software application to pass user defined data to and from UserIP via the Local Link data stream.

For the transmit channel the first Descriptor, which includes APP0 to APP4, describing a packet is transferred in the header of the LocalLink data stream. (See Figure 14).

For the receive channel the last Descriptor is populated with the LocalLink footer user App Fields, App1 to App4 (See Figure 15). Note that for receive App0in the Descriptor is not updated.

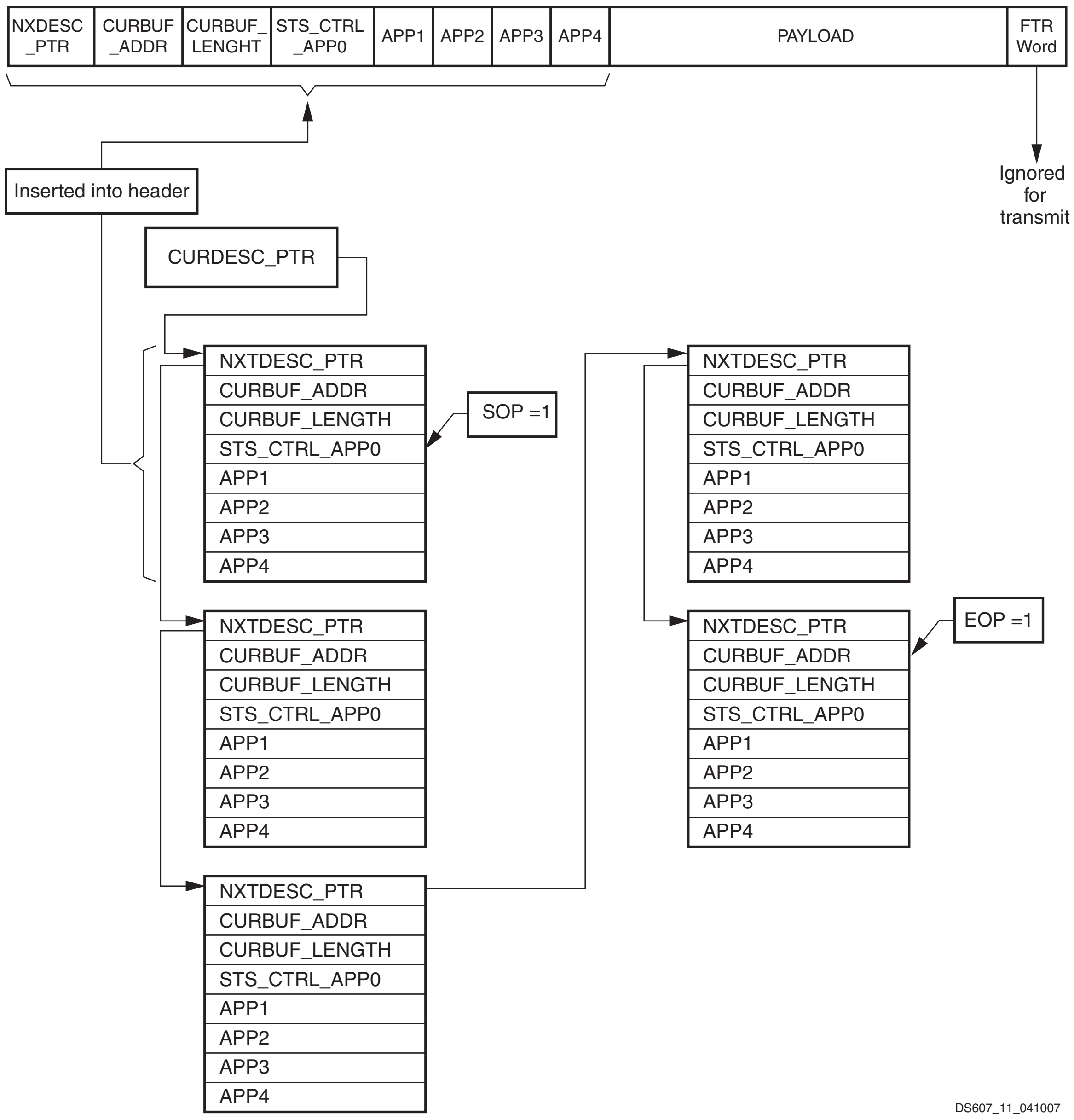

Figure 14: **LocalLink Transmit Data Stream Header Assignment**

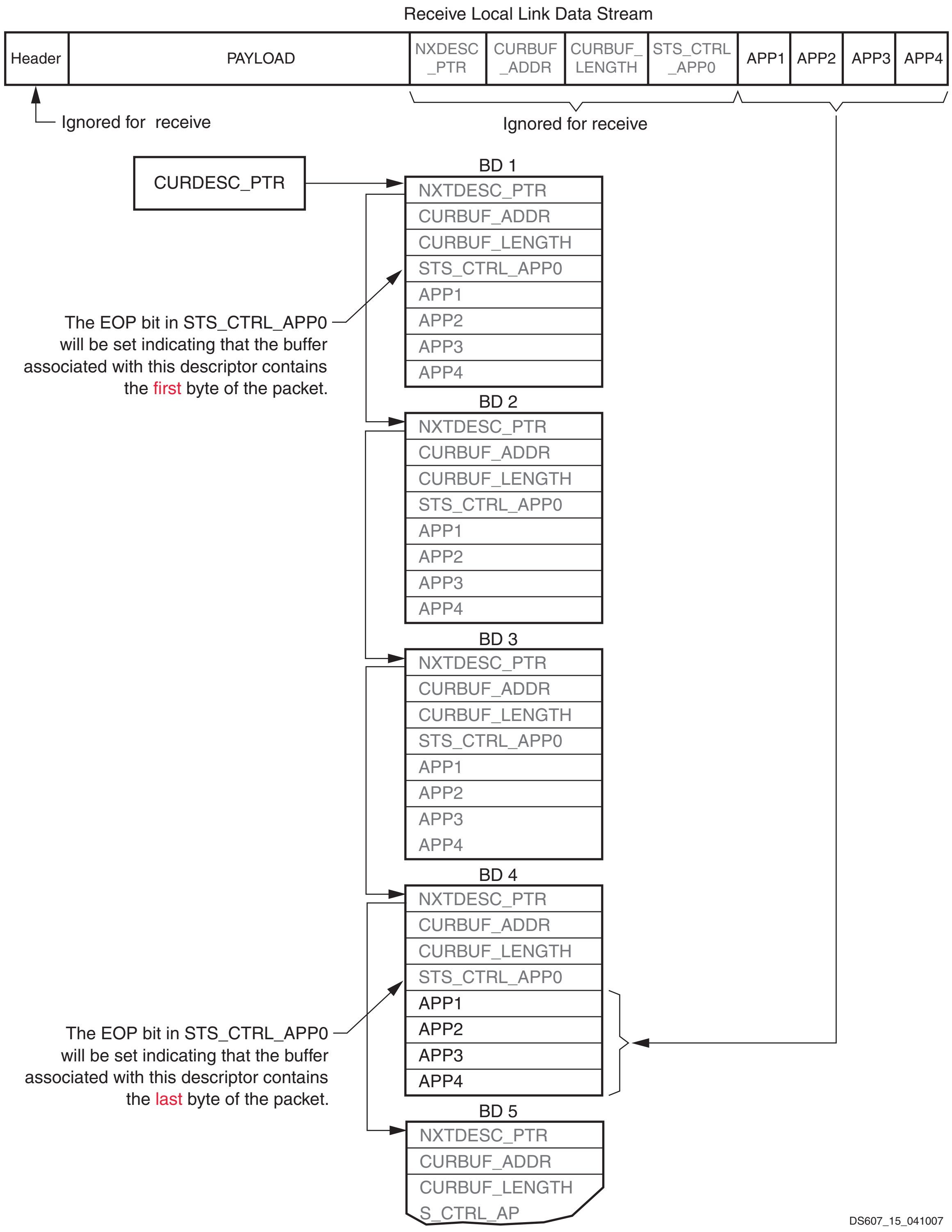
Receive Local Link Data Stream
Header
PAYLOAD
NXDESC_PTR
CURBUF_ADDR
CURBUF_LENGTH
STS_CTRL_APP0
APP1
APP2
APP3
APP4
Ignored for receive
Ignored for receive
CURDESC_PTR
BD 1
NXTDESC_PTR
CURBUF_ADDR
CURBUF_LENGTH
STS_CTRL_APP0
APP1
APP2
APP3
APP4
The EOP bit in STS_CTRL_APP0 will be set indicating that the buffer associated with this descriptor contains the first byte of the packet.
BD 2
BD 3
BD 4
The EOP bit in STS_CTRL_APP0 will be set indicating that the buffer associated with this descriptor contains the last byte of the packet.
BD 5
DS607_15_041007

Figure 15: **LocalLink Receive Data Stream Footer Assignment**

Transmit Local Link Byteshifter

The Transmit LocalLink and Byteshifter Logic take data from the appropriate place in memory and move the data across the LocalLink interface. This concept is shown in Figure 16. In this example, the SDMA reads the descriptor at address p to p+1C and sends it to the LocalLink as the header. The payload is 136 bytes and starts at address m+79. The Transmit Byteshifter sends data acknowledges to the memory controller while keeping the Src_Rdy signal to the LocalLink de-asserted, because address m+79 is not 32-word aligned. Data from address m to m+78 are discarded. Data is offset by 78 bytes, so the first byte of data occurs on the second byte location on the posedge of the DDR SDRAM. The Transmit Byteshifter takes the posedge (x 0 1 2) and negedge (3 4 5 6) data from DDR SDRAM, which are both present at the time, recombines them to form a new, correctly shifted, word (0 1 2 3), and sends it over the LocalLink as the payload. At the end of the first 32-word burst read (B16R), 3 bytes are left over and kept in the Byteshifter. When the second burst occurs, those 3 bytes are combined with the first byte of the second burst and sent over LocalLink. This happens again between the second burst and third burst. The fourth burst is generated due to a second descriptor. It too describes a buffer that begins at an odd boundary, i.e. offset 0x7E. Byte r0 and r1 are combined with the left over bytes from the previous burst, n and n+1. On the last word of the payload the Rem signal is set to indicate which bytes of the word are valid. Rem is 0x3 in this example to indicate only the first 2 bytes are valid. After byte n+1 is sent, the FIFOs in MPMC, which hold all 32 words of the burst, are reset to avoid extra data acknowledge. For Tx transfer, the footer is not used. The status bits are written back to the descriptor's status field.

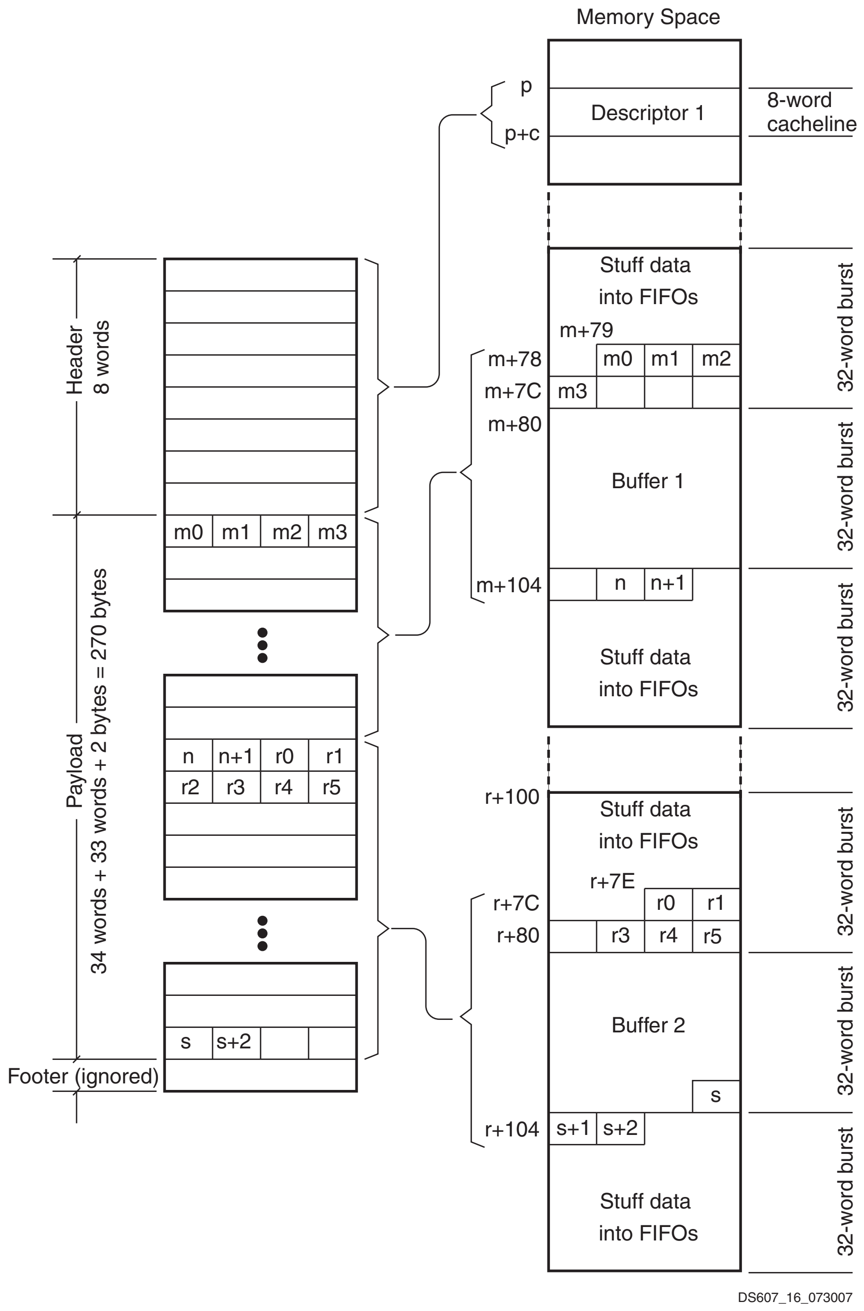

Figure 16: **Transmit Byte Shift Example**

Receive LocalLink Byte Shifter

The LocalLink and Byteshifter Rx Logic receive data from the LocalLink interface and move the data to the appropriate place in memory. This concept is shown in Figure 17. The SDMA always ignores the Rx LocalLink Header. The Payload is processed by the Rx Byteshifter and pushed into the MPMC's Write FIFOs. In this example, the Payload is 270 bytes. The data is pushed into the FIFOs in bursts of 32-words (B16W). Data is stuffed into the FIFOs from address m through address m+0x7C because the Payload is written to address m+0x79, which is a not a 32-word aligned address. When these bytes are written to memory, the byte enables are turned off. In the second B16W, all of the data is valid. In the third B16W, only 3 bytes need to be written to memory. This means that the remaining 125 bytes need to be stuffed into the FIFOs at the end of the burst. In this example, the length of buffer 1 was specified to be 138 bytes long in the receive Descriptor. After the first 3 B16W's buffer 1 is full. The remainder of the payload is transferred to buffer 2. The fourth, fifth, and sixth transfers are similar to the first three transfers in that the first valid byte is not at an even boundary. SDMA will begin the fourth B16W at r+00 setting the byte enables to off for all the bytes up to the first valid byte at r+7E. The fifth B16W has all bytes valid and the fifth and final B16W only 2 bytes are valid. The remaining 126 bytes are pushed to the FIFO with the byte enables set to off. After the Payload has been processed, the Footer is processed and written to memory at address p. The SDMA changes the first three words' byte enables to prevent the Next Descriptor Pointer, the Buffer Address, and the Buffer Length from being overwritten. Thus only the status field of STS_CTRL_APP0 as well as APP1, APP2, APP3, and APP4 are updated in the memory space. Note that for receive the App0 field is not updated in the Descriptor.

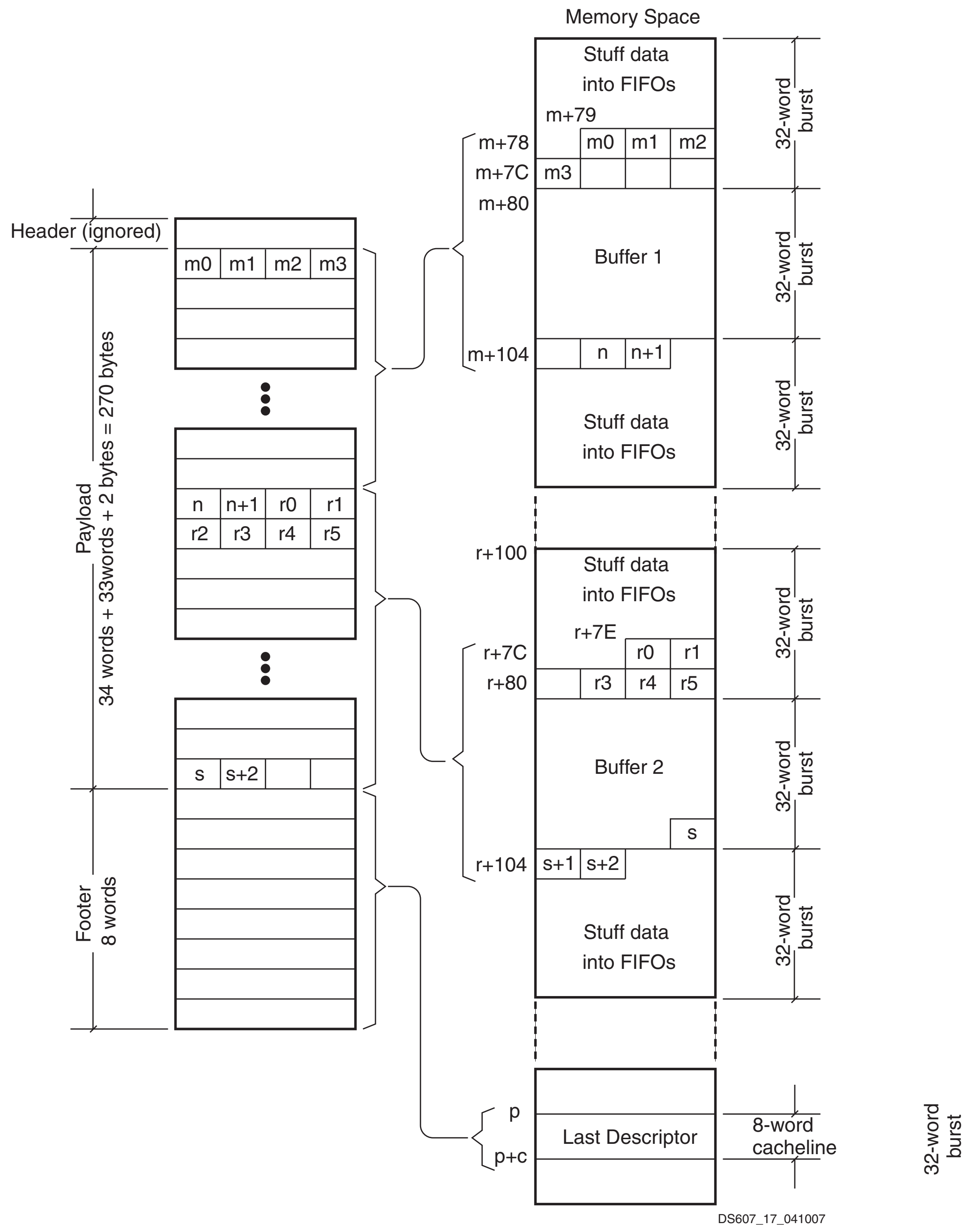

Figure 17: **Receive Byte Shift Example**

DMA Controller Interrupt Description

Each channel may generate one or more events that are of interest to the user. Events are reported via the Interrupt Registers (IRQ_REG) for the individual channel, Figure 18. The IRQ_REG generates a system interrupt when ever ERRIrq, DlyIrq, or ClscIrq is set and the corresponding enable bit is set in the CHANNEL_CTRL Register. Reading of the individual IRQ's allows the software to determine which event occurred and for which channel the event was logged. Writing a 1 to the bit position of the event that was logged clears that event, in the case of the

IRQ_REG.ErrIrq interrupt, or decrements the ClscCnt or DlyCnt depending on which bit is written to. When the IRQ_REG.ClscCnt is zero then the IRQ_REG.ClscIrq will be cleared to zero. Likewise when the IRQ_REG.DlyCnt is zero then the IRQ_REG.DlyIrq will be cleared to zero.

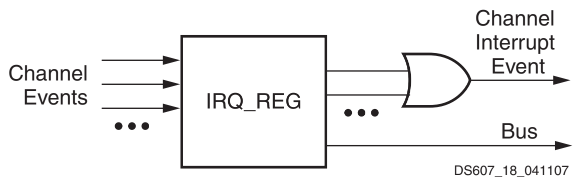

Figure 18: **Interrupt Status Register**

Error Event

This event occurs when an error is detected during DMA operations. If an error is detected then IRQ_REG.ErrIrq will be set and if CHANNEL_CTRL.IrqEn = 1 and CHANNEL_CTRL.IrqErrEn = 1 then an interrupt will also be generated or the coalescing event counter will be incremented. This event can be cleared by writing a 1 to IRQ_REG.ErrIrq. When a DMA transfer error occurs for a particular channel that channel is shut down and no more DMA processing occurs for that channel.

If counter overflow errors are enabled (i.e. DMA_CONTROL.RxOFD=0 and/or DMA_CONTROL.TxOFD=0) then and error event will be generated when the IRQ_REG.ClscCnt or IRQ_REG.DlyCnt is overflowed. The error events will not affect DMA transfers and by default are disabled.

Interrupt On End Event

This event occurs when SDMA has completed processing of a Buffer Descriptor with the IOE bit set in the STS_CTRL_APP0 field or an end of packet, EOP has been received or transmitted. If the DMA has completed operations then IRQ_REG.CoalIrq will be set and if enabled then an interrupt will also be generated or the coalescing event counter will be incremented.

Interrupt Coalescing

Delay Timer

The delay timer is needed because we're using interrupt coalescing in the DMA. An example is, if the RX coalescing counter is set to 10, every 10 packets received will generate an interrupt. But let's say we receive 5 packets on the ethernet and then the channel goes idle (no traffic). The CPU will never process the 5 packets because no interrupt was generated and this interrupt will happen only when (or if) 5 more packets arrive. To avoid this latency, we want a timer which will fire when a packet has been received AND some (software settable) time has elapsed AND there are no more packets received during this time. The only purpose of this timer is to avoid large latencies in the received packet (which is sitting in main memory by this time) from being processed by the CPU when there's non-continuous traffic on the wire

As shown in Figure 19, the Clock Divider module uses a 10-bit value, C_PRESCALAR, to determine how many LocalLink clock cycles to count before generating a single "Timer_ce" pulse. For a typical LocalLink clock speed of 200MHz and C_PRESCALAR=1023, this translates to a 5.12 usec "Timer_ce" period. The 8-bit timer will therefore be able to count up to a maximum of 256*5.12 = 1.3 msecs, before generating an interrupt.

Note that when the Coalescing counter fires, the delay timer is automatically cleared.

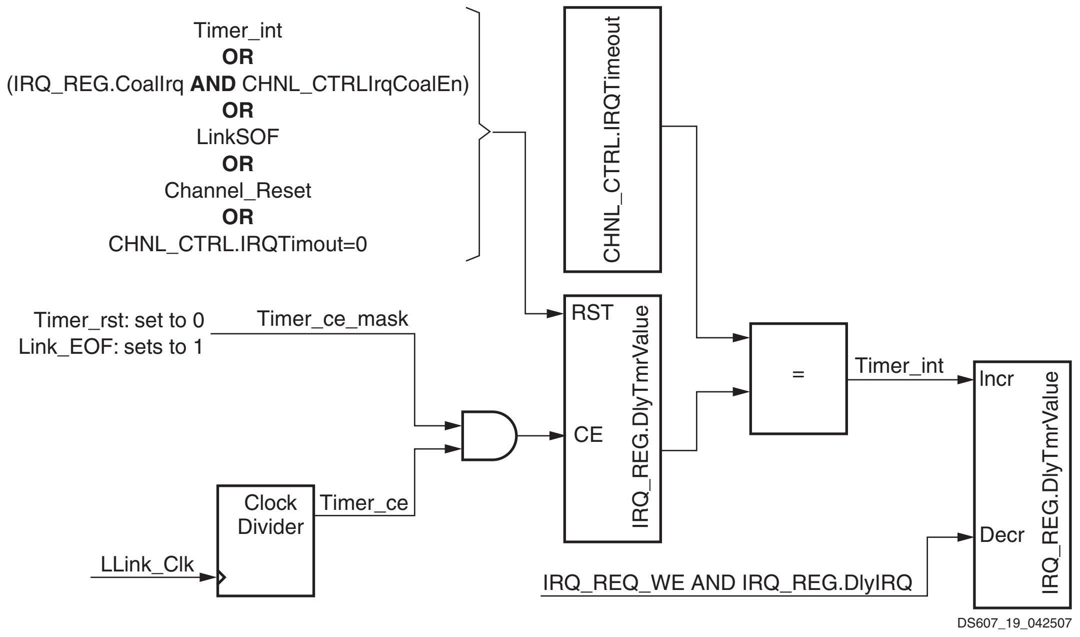

Figure 19: **Delay Timer Interrupt Scheme**

The Interrupt Coalescing counter is an additional mechanism used for interrupt handing. Its purpose is to relieve the CPU of having to service an interrupt at the end of every packet. Instead, a pre-loadable number of interrupt events (up to 256) will generate a single interrupt to the CPU. Figure 20 shows the mechanism used for the Tx coalescing counter interrupt generation.

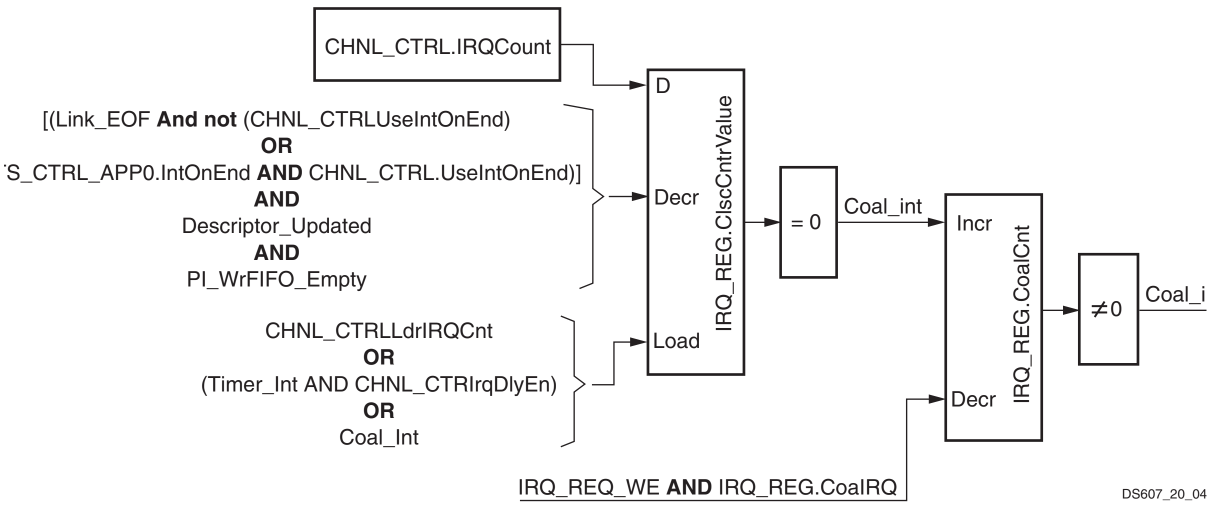

Figure 20: **Coalescing Counter Interrupt Scheme**

On reset, the "CHNL_CTRL.IRQCount" value is used to load the coalescing counter. The register field, "TxIrqCountReg[0:7]" can be subsequently programmed with any 8-bit value. On every eop or "irq-on-end" (selected by CHNL_CTRL_UseIntOnEnd") the counter will decrement. When the coalescing counter hits 0, the dma increments the 4-bit int counter. Whenever the 4-bit int counter is non-zero, it generates an interrupt to the

CPU (if the respective channels irq enable bit is set). Whenever the interrupt is acknowledged (DCR write of 1), the 4-bit int counter is decremented. The contents of the "TxIrqCountReg[0:7]" register is re-loaded when the 4-bit int counter is incremented.

There is also a means provided for the CPU to force the counter to load the contents of the "TxIrqCountReg[0:7]" register. This is achieved when the CPU writes the "ldIrqCnt" field.

Note that when the delay timer fires, the coalescing counter is automatically re-loaded.

DMA Engine Reset

A capability is included to reset a particular DMA engine (both RX and TX channels simultaneously) whenever a "lockup" situation arises or an error is detected.

A software reset bit, DMA_CONTROL_REG.SwReset allows the software application to reset SDMA. When you write a "1" to DMA_CONTROL_REG.SwReset, it will initiate the reset sequence for that SDMA. At the same time, the SDMA_RstOut output will be asserted, synchronous to the LocalLink clock. This output can be used as an external logic reset. Once a soft reset is initiated, software needs to poll the DMA_CONTROL_REG.SwReset bit until it is sampled de-asserted. This indicates that the reset sequence has completed and the pipeline is flushed. Simultaneously with the DMA_CONTROL_REG.SwReset bit being cleared, the SDMA_RstOut will be automatically de-asserted.

Note that whenever the DMA engine reset function is used, there is no guarantee that the current descriptor completed correctly. The assumption should be that the descriptor did not complete and it should be restarted again using the normal CPU technique for kicking off a new DMA operation.

Transaction Timing

Descriptor Fetch

Figure 21 shows NPI port interface timing for a Descriptor fetch. The SDMA uses 8-Word Cache line reads to perform a Descriptor Fetch. The timing from when the SDMA makes a request of the NPI will depend on arbitration in MPMC. The SDMA will monitor PI_RdFIFO_Empty to determine when to begin 'Pop'ing data out of the MPMC's read FIFO. Figure 21 pi_rdfifo_pop and pi_rdfifo_data timing for C_PI_RDDATA_DELAY=0 and C_PI2LL_CLK_RATIO=1.

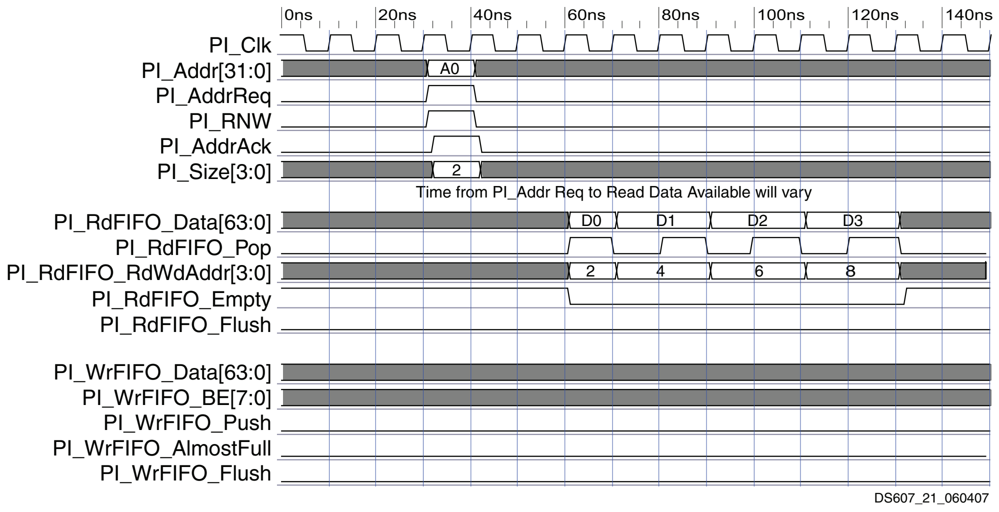

Figure 21: **Descriptor Fetch Timing**

Descriptor Update

Figure 22 shows MPMC2 port interface timing for a Descriptor fetch. The SDMA uses 8-Word Cache line writes to perform a Descriptor Update. The SDMA will Push the data into the MPMC's write FIFO if there is space available. Once all of the data has been Pushed to the FIFO the SDMA will make a write request of the MPMC. Figure 22 pi_wrfifo_push and pi_wrfifo_data timing for C_PI2LL_CLK_RATIO=1.

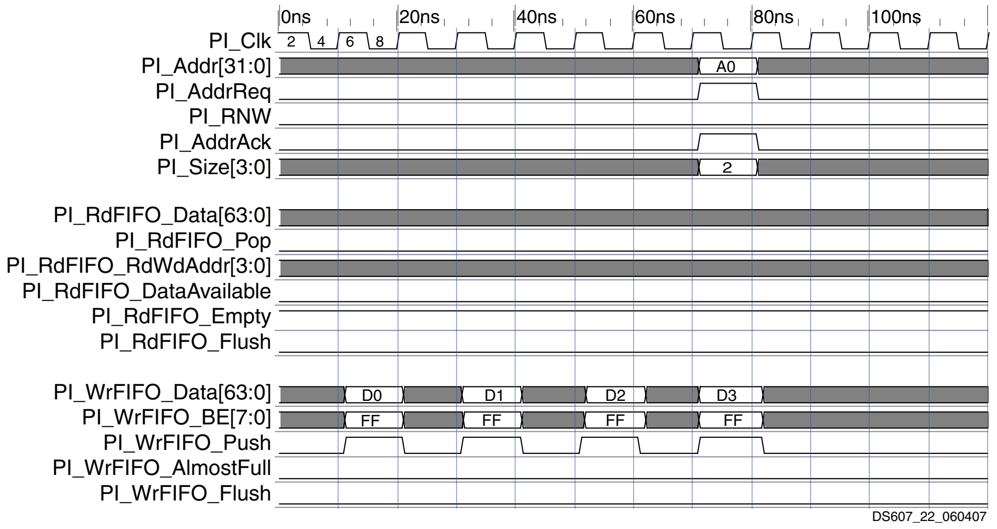

Figure 22: **Descriptor Update Timing**

Transmit Data Read

Figure 23 shows a transmit data fetch from the MPMC. The SDMA uses 32-word (i.e. 16-Double Words) reads to fetch data for transmitting across LocalLink. Fetches always begin at 32-word boundaries. Only the data starting with the Current Buffer Address is used by SDMA and additional invalid data fetched due to the 32-word boundary restriction is ignored by SDMA. Figure 23 pi_rdfifo_pop and pi_rdfifo_data timing for C_PI_RDDATA_DELAY=0 and C_PI2LL_CLK_RATIO=1

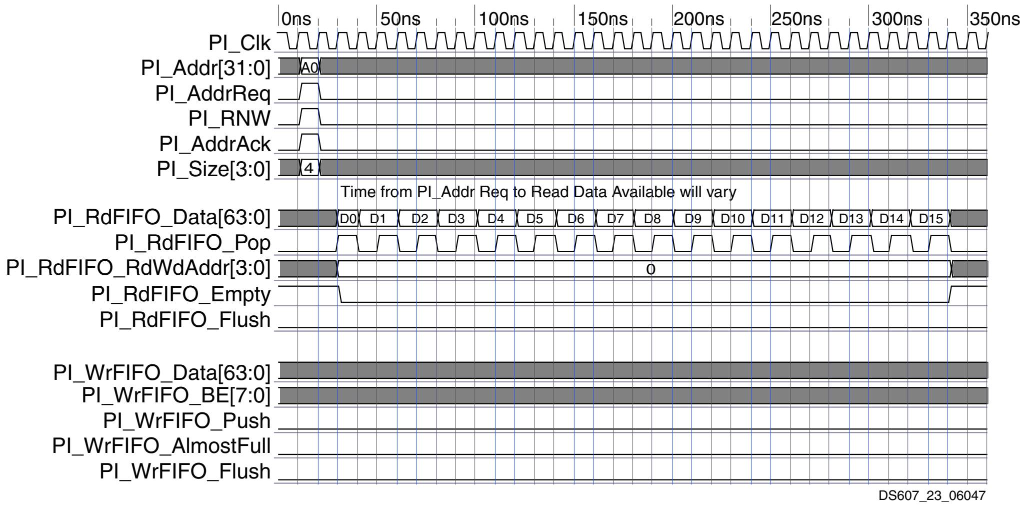

Figure 23: **Transmit Data Read**

Receive Data Write

Figure 24 shows received data being written to the MPMC. The SDMA uses 32-word burst writes to write data to the MPMC2. The transfers are always 32-word aligned. The PI_WrFIFO_BE bus is used to indicate which bytes of the 32 words are valid. The first two bytes are shown as being invalid and the last byte being invalid. Figure 24 pi_wrfifo_push and pi_wrfifo_data timing for C_PI2LL_CLK_RATIO=1

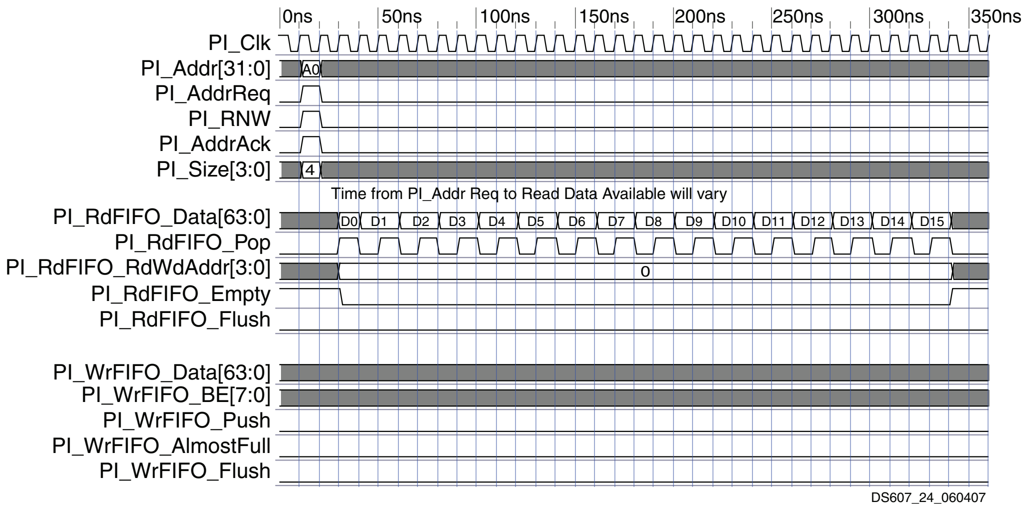

Figure 24: **Receive Data Write**

Transmit LocalLink

Figure 25 shows an example transmit LocalLink transfer of 8 words. Note that during a transmit the first buffer descriptor is transferred in the header of the LocalLink data stream.

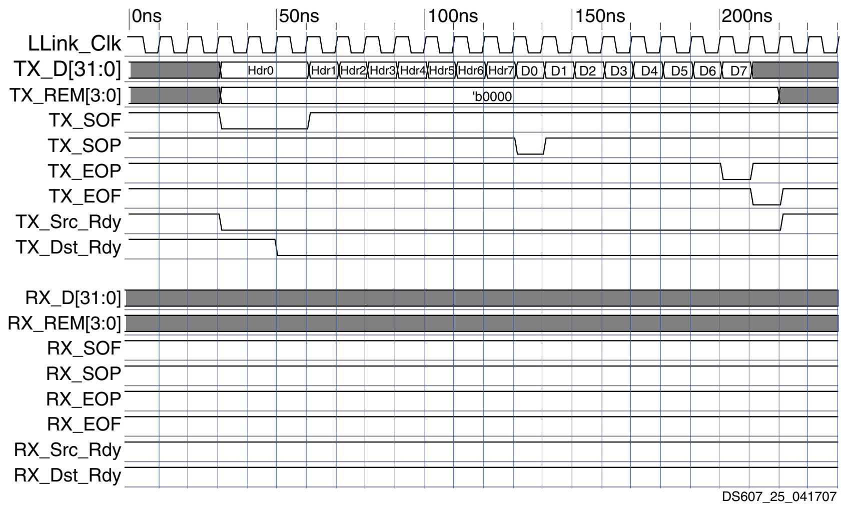

Figure 25: **Transmit LocalLink Timing**

Receive LocalLink

An example receive LocalLink transfer of 8 words is shown in Figure 26. Note the during a receive the last buffer descriptor of a packet is populated with the APP fields of the LocalLink footer.

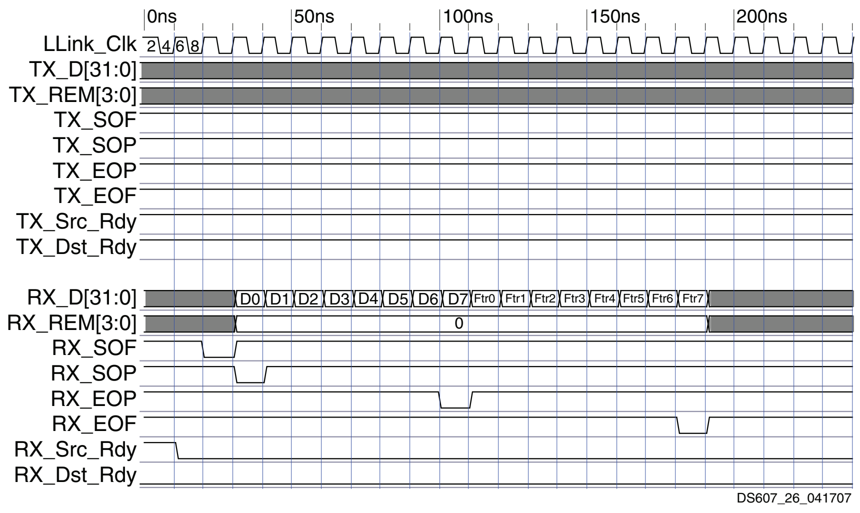

Figure 26: **Receive LocalLink Timing**

User Application Topics

NPI Interface Byte Swap

Per MPMC specification data is written and read in byte based Little Endian format. The SDMA convert all data writte to the NPI's write fifo port from Big Endian to Little Endian format. Similarly, SDMA converts all data read from NPI's read fifo port from Little Endian to Big Endian format. See Figure 27 and Figure 28

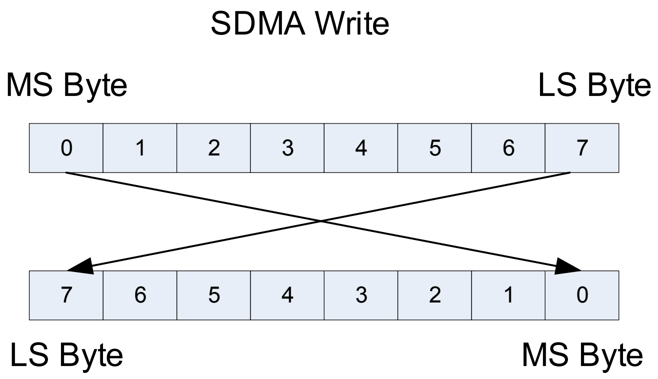

Figure 27: **SDMA Write To NPI**

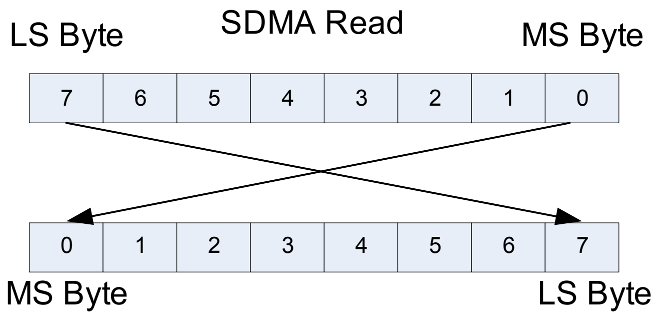

Figure 28: **SDMA Read From NPI**

Dynamic Descriptor Update

At times it may be necessary to update a Descriptor chain while the SDMA is actively processing the Buffer Descriptors. This can be achieved when a channel is configured to operate in TailPointer Mode.

Appending A Descriptor Chain

The SDMA has been designed such that the software application can append new Buffer Descriptor chains into an already active chain with minimal effort on the part of the software application. This can be accomplished simply by updating the TAILDESC_PTR if the Descriptors are arranged in a ring, or by the use of a dummy descriptor where NXTDESC_PTR and TAILDESC_PTR can be updated to point to the start and end of the appended Descriptor chain. The following sections describe each method for modifying an active Descriptor Chain.

Descriptor Modification Using Dummy Descriptor

To append a Descriptor or set of Descriptors to a descriptor chain that is not arranged in a ring, two updates need to occur, one to the NXTDESC_PTR and one to the TAILDESC_PTR. To avoid a race condition in which the SDMA is already fetched the NXTDESC_PTR of the last Descriptor of the chain before the software application can update the last descriptor to point to the newly appended Descriptors, a Dummy Descriptor is used. As an example, assume the software application has set up the following Descriptor Chain, Descriptor 0 to Descriptor 9 with a Dummy Descriptor, DMY, at the end of the chain. Also assume that the TAILDESC_PTR register contains the address of Descriptor 9. Figure 27 illustrates the Descriptor Chain. If no updates where made to the Descriptor chain then the DMA controller would process Descriptor 0 through Descriptor 9 and stop after Descriptor 9 because the original TAILDESC_PTR would equal the CURDESC_PTR of Descriptor 9. The NXTDESC_PTR that SDMA has stored would be pointing to the Dummy Descriptor.

For this example, assume that the software application has started DMA operation by setting the TAILDESC_PTR address to Descriptor 9. The SDMA will fetch Descriptor 0 and begin processing Descriptor 0. At this point software determines that it needs to append Descriptor 10, 11, and 12 to the already active chain.

Three steps are required by software to append the new Descriptors:

1. Setup Descriptor 10, 11, and 12 in remote memory
14. Update the NXTDESC_PTR of the Dummy Descriptor in memory space to point to Descriptor 10
15. Update the TAILDESC_PTR register with the address of Descriptor 12.

A race condition would have occurred if SDMA was in the middle of processing Descriptor 9 and there was no Dummy Descriptor. The NXTDESC_PTR that SDMA would have would be pointing to a potentially invalid location. If the software application where to update NXTDESC_PTR in memory space and then update the TAILDESC_PTR in the SDMA, the SDMA would not update its NXTDESC_PTR register from memory space and thus would fetch the next descriptor from an invalid place.

With NXTDESC_PTR pointing to a Dummy Descriptor the SDMA will fetch the Dummy Descriptor once the TAILDESC_PTR is updated which has a correctly updated NXTDESC_PTR pointing to the newly appended

Descriptors. Because the Dummy Descriptor has a SOP, EOP and Length set to zero, the Descriptor is dropped and the Next Descriptor is fetched.

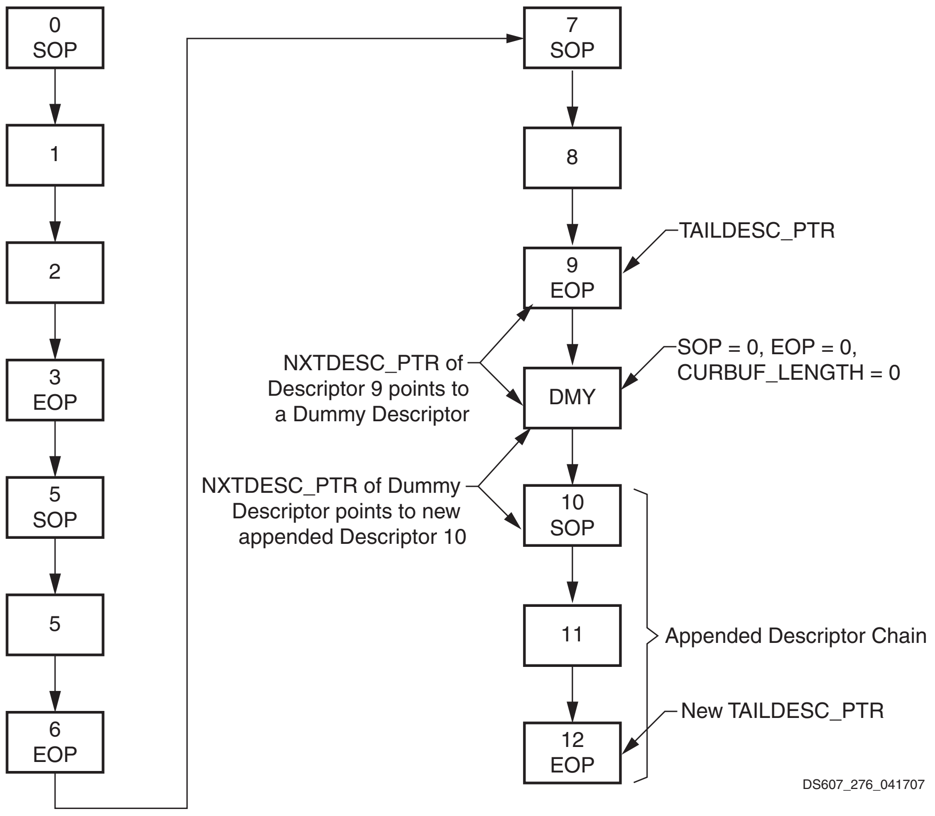
0
SOP
1
2
3
EOP
5
SOP
5
6
EOP
7
SOP
8
TAILDESC_PTR
9
EOP
NXTDESC_PTR of Descriptor 9 points to a Dummy Descriptor
DMY
SOP = 0, EOP = 0, CURBUF_LENGTH = 0
NXTDESC_PTR of Dummy Descriptor points to new appended Descriptor 10
10
SOP
11
Appended Descriptor Chain
New TAILDESC_PTR
12
EOP
DS607_276_041707

Figure 29: **Descriptor Chain Append**

Descriptor Modification Using A Descriptor Ring

With this method no new Descriptors are added. This method works for Descriptors configured into a ring as shown in Figure 28. Consider a situation where 14 Descriptors arranged in a loop and the stop point from Descriptor 13 is to be moved to Descriptor 5 without stopping the chain. Follow the steps below to perform that operation. It is assumed that the new Descriptor chain has been created in remote memory and that the SDMA is actively processing the original Descriptor chain.

1. Modify already processed Descriptors, i.e. ones with STS_CTRL_APP0.Completed = 1

16. Move TAILDESC_PTR to point at Descriptor 5.

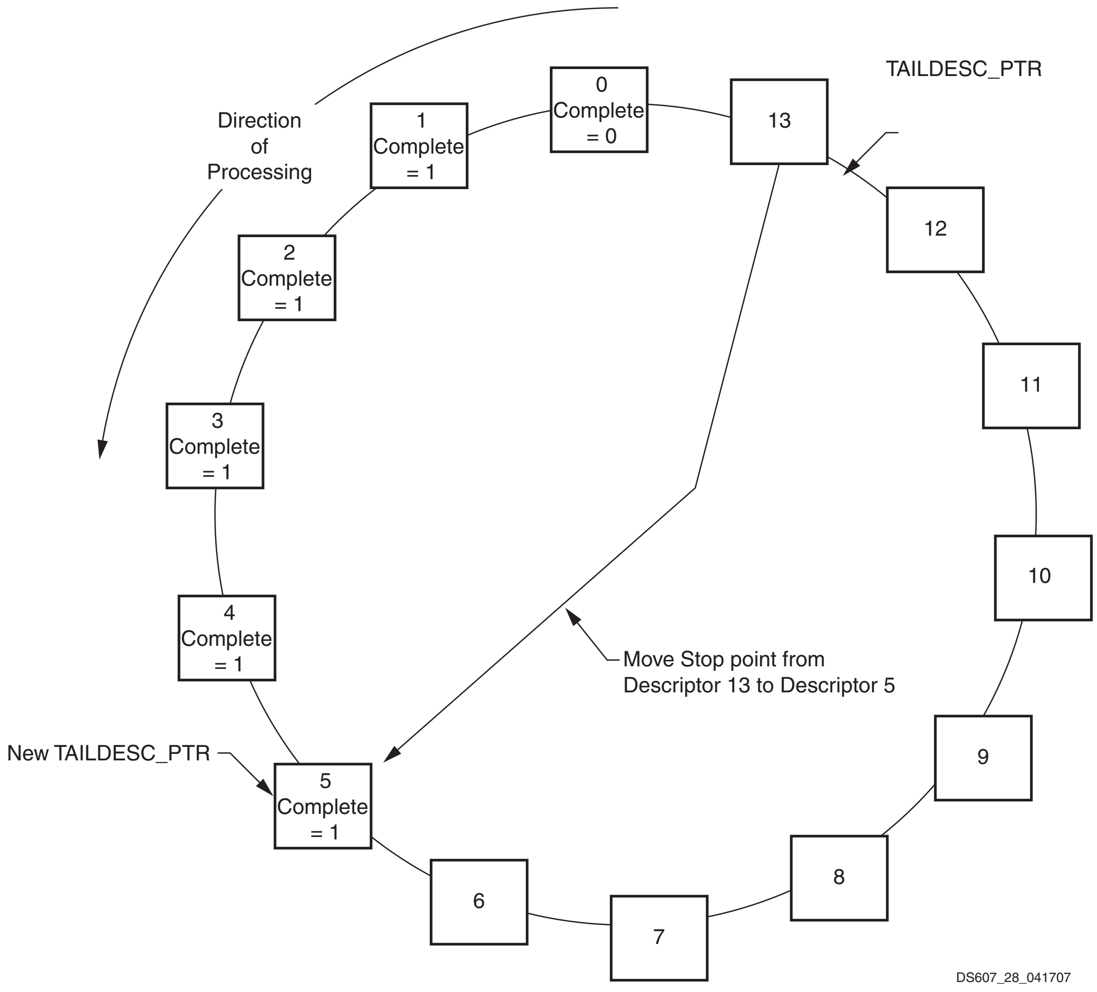

Figure 30: **Descriptor Chain - Loop Configuration**

Design Implementation

Target Technology

The intended target technology is a Spartan, Spartan-3e, Spartan-3a, Virtex-4 and Virtex-5 FPGA.

Device Utilization and Performance Benchmarks

Because the SDMA is a module that will be used with other design modules in the FPGA, the utilization and timing numbers reported in this section are just estimates. As the SDMA is combined with other pieces of the FPGA design, the utilization of FPGA resources and timing will vary from the results reported here

The resource utilization of this version of the SDMA is shown in Table 16 for some example configurations. The SDMA was synthesized using the Xilinx XST tool. The XST resource utilization report was then used as the source data for the table.

The SDMA benchmarks are shown in Table 16 for xc5vlx220-2-ff1760 FPGA..

Table 16: **SDMA FPGA Performance and Resource Utilization Benchmarks**

Parameter Values					Device Resources			f_{MAX}(1)		
C_SPLB_DWIDTH	C_SPLB_P2P	C_PI_RDDATA_DELAY	C_COMPLETED_ERR_TX	C_COMPLETED_ERR_RX	Slices	Flip-Flops	LUTs	SPLB_Clk (Mhz)	LLink_Clk (Mhz)	PI_Clk (Mhz)
32	1	0	0	0	738	944	1535	250.8	200.1	440.9
32	0	0	0	0	869	1018	1532	230.0	200.4	431.8
32	0	1	0	0	869	1018	1532	230.0	200.4	431.8
32	0	2	0	0	792	1091	1625	204.6	200.1	400.3
32	0	2	1	0	844	1092	1627	218.0	200.1	407.5
32	0	2	1	1	813	1093	1623	209.5	200.3	411.0
64	0	2	1	1	773	1093	1623	223.8	200.7	411.0
128	0	2	1	1	795	1093	1620	223.3	200.3	416.1

Notes:

1. Fmax represents the maximum frequency of the SDMA in a standalone configuration. The actual maximum frequency will depend on the entire system and may be greater or less than what is recorded in this table.

Specification Exceptions

None

Reference Documents

The following documents contain reference information important to understanding the Channelized DMA Controller design.

- SP006, LocalLink Interface Specification
- DS643, Multi-Port Memory Controller
- *IBM CoreConnect™128-Bit Processor Local Bus, Architectural Specification* (v4.6).

Revision History

Date	Version	Revision
12/20/07	1.0	Initial Xilinx release.