Intel® 80331 I/O Processor

Application Accelerator Unit D-0 Addendum

January, 2005

Document Number: 304496001US

Contents

Figures

Tables

Revision History

Date	Revision	Description
January 2005	001	Initial Developer Web Site Release.

This Page Left Intentionally Blank

Application Accelerator Unit 1

This chapter describes the integrated Application Accelerator (AA) Unit. The operation modes, setup, external interface, and implementation of the AA unit are detailed in this chapter.

1.1 Overview

The Application Accelerator provides low-latency, high-throughput data transfer capability between the AA unit and Intel® 80331 I/O processor (80331) local memory. It executes data transfers to and from 80331 local memory, checks for all-zero result across local memory blocks, performs memory block fills, and provides the necessary programming interface. The Application Accelerator performs the following functions:

- Transfers data (read) from memory controller.
- Performs an optional boolean operation (XOR) on read data.
- Transfers data (write) to memory controller or PCI.
- Checks for All-zero result across local memory blocks.
- Performs memory block fills.
- Optional Dual-XOR for RAID-6 application single strip write.
- Optional Galois Field (GF) Multiply calculation for P+Q RAID-6 in conjunction with XOR operations.

The AA unit features:

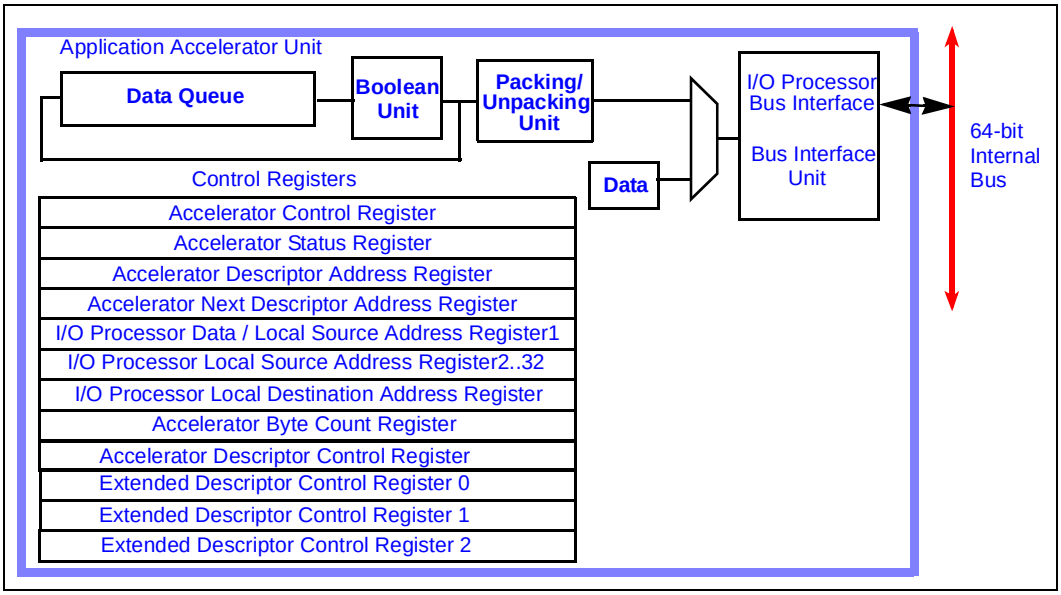
- 1Kbyte/512-byte store queue.
- Utilization of the 80331 memory controller Interface.
- 2^{32} addressing range on the 80331 local memory interface.
- Hardware support for unaligned data transfers for the internal bus.
- Fully programmable from the Intel XScale® core.
- Support for automatic data chaining for gathering and scattering of data blocks.
- Support for writing a constant value to a memory block (block fill).
- Support for writing descriptor status to local memory.
- Hardware to perform Galois Field (GF) Multiply function on the Source Data Streams during an XOR operation, when enabled.

1.2 Theory of Operation

The Application Accelerator is a master on the internal bus and performs data transfers to and from local memory. It does not interface to the PCI bus. AA uses direct addressing for memory controller.

The AA implements XOR algorithm in hardware. It performs XOR operation on multiple blocks of source (incoming) data and stores result back in 80331 local memory. The source and destination addresses are specified through chain descriptors resident in 80331 local memory. A Dual-XOR operation is also supported for optimized processing of two different, but related XOR operations in a single operation. The AA can also check for all-zero result across local memory blocks or fill a memory block with arbitrary data. Figure 1 shows a block diagram of the AA unit. The AA can also perform memory-to-memory transfers of data blocks controlled by 80331 memory controller unit.

Figure 1. Application Accelerator Block Diagram



AA programming interface is accessible from the internal bus through a memory-mapped register interface. Data for XOR operation is configured by writing source addresses, destination address, number of bytes to transfer, and various control information into a local memory chain descriptor. Chain descriptors are described in detail in Section 1.3.2, “Chain Descriptors” on page 13.

The AA unit contains a hardware data packing and unpacking unit. This unit enables data transfers from and to unaligned addresses in 80331 local memory. All combinations of unaligned data are supported with the packing and unpacking unit. Data is held internally in the AA until ready to be stored back to local memory. This is done using a 1KByte/512Byte holding queue. Data to be written back to 80331 local memory can either be aligned or unaligned.

Each chain descriptor contains necessary information for initiating an XOR operation on blocks of data specified by the source addresses. The AA unit supports chaining. Chain descriptors that specify the source data to be XORed can be linked together in 80331 local memory to form a linked list.

Similar to XOR operations, AA can be programed to compute parity across multiple memory blocks specified by chain descriptors. In addition, AA is also used for memory block fills. A Dual-XOR operation is available for use when calculating two parity blocks for a RAID-6 single strip write.

In conjunction with the XOR and Dual-XOR operations, a GF Multiply calculation can be applied to source data in support of P+Q RAID-6. The AA will perform a GF Multiply between source data and a control byte for each source before the XOR operation when enabled. P+Q RAID-6 is enabled through an enable bit in the Section 1.13.1, “Accelerator Control Register - ACR” (bit 3).

1.3 Hardware-Assist XOR Unit

The AAU implements the XOR algorithm in hardware. It performs the XOR operation on multiple blocks of source (incoming) data and stores the result back in 80331 local memory.

- The process of reading source data, executing the XOR algorithm, and storing the XOR data will hereafter is referred to as *XOR-transfer*.
- The operation of two *XOR-transfer*s defined in a single descriptor will hereafter be referred to as *Dual-XOR transfer.*
- The process of reading or writing data will hereafter is referred to as *data transfer*.

Source and destination addresses specified through chain descriptors resident in 80331 local memory.

1.3.1 Data Transfer

All transfers are configured and initiated through a set of memory-mapped registers and one or more chain descriptors located in local memory. A transfer is defined by the source address, destination address, number of bytes to transfer, and control values. These values are loaded in the chain descriptor before a transfer begins. Table 1 describes the registers that need to be configured for any operation.

When P+Q RAID-6 is enabled, the GF Multiplier bytes also act as control values in the data transfer.

Table 1. Register Description

Register	Abbreviation	Description
Accelerator Control Register	ACR	Application Accelerator Control Word
Accelerator Status Register	ASR	Application Accelerator Status Word
Accelerator Descriptor Address Register	ADAR	Address of Current Chain Descriptor
Accelerator Next Descriptor Address Register	ANDAR	Address of Next Chain Descriptor
Data / Source Address Register1	D/SAR1	Data to be written or Local memory addresses of source data
Source Address Register 2..32	SAR2.. SAR32	Local memory addresses of source data
P+Q RAID-6 Source Address Register 2..16	PQSAR2.. PQSAR16	Local memory addresses of source data when operating in P+Q RAID-6 mode
P+Q RAID-6 GF Multiplier Register 1..5,D	GFMR1.. GFMR5, GFMRD	P+Q RAID-6 GF Multiplier bytes when operating in P+Q RAID-6 mode
Destination Address Register	DAR	Address of result data
Accelerator Byte Count Register	ABCR	Number of Bytes to transfer
Data Register	DR	Data to be written to the destination
Accelerator Descriptor Control Register	ADCR	Chain Descriptor Control Word

1.3.2 Chain Descriptors

All transfers are controlled by chain descriptors located in local memory. A chain descriptor contains the necessary information to complete one transfer. A single transfer has only one chain descriptor in memory. Chain descriptors can be linked together to form more complex operations.

Warning: Chain descriptors can only be located in DDR SDRAM memory in order for the AAU to function properly. Location of the chain descriptors anywhere else (e.g., either on the Peripheral Bus or on PCI) is not supported and the results would be unpredictable.

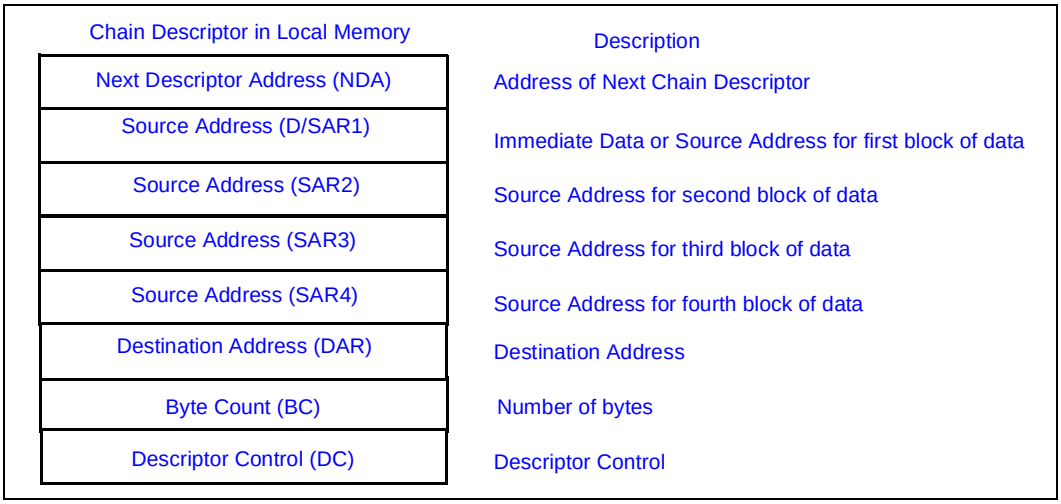
To perform a transfer, one or more chain descriptors must first be written to 80331 local memory. The words of a descriptor are contiguous in local memory. Descriptors can be different sizes, each dependent on the number of sources being addressed by the operation. The sizes supported by the Application Accelerator are four, eight, sixteen and thirty-two sources. The alignment of the descriptor in local memory is dependent on the descriptor size and is defined for each in the following sections. Not all sources must be used in a given descriptor.

Descriptor formats for P+Q RAID-6 enabled operation are defined separately. When P+Q RAID-6 is enabled, only the P+Q RAID-6 descriptor formats are valid.

1.3.2.1 Principle / Four-Source Descriptor Format

Figure 2 shows the format of an individual chain descriptor. This four-source descriptor is the smallest supported descriptor. The four-source descriptor requires eight contiguous words in 80331 local memory and is required to be aligned on an 8-word boundary. All eight words are required.

Figure 2. Principle / Four Source Descriptor Format



Each word in the chain descriptor is analogous to control register values. Bit definitions for the words in the chain descriptor are the same as for the control registers.

- First word is local memory address of next chain descriptor. A value of zero specifies the end of the chain. This value is loaded into the Accelerator Next Descriptor Address Register. Because chain descriptors must be aligned on a minimum 8-word boundary, the unit may ignore bits 04:00 of this address.
- Second word is address of the first block of data resident in local memory, or immediate data when performing a Memory Block Fill. This value is loaded into the Data / Source Address Register 1.
- Third word is the address of the second block of data resident in local memory. This address is driven on the internal bus. This value is loaded into the Source Address Register 2.
- Fourth word is the address of the third block of data resident in local memory. This address is driven on the internal bus. This value is loaded into the Source Address Register 3.
- Fifth word is the address of the fourth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into the Source Address Register 4.
- Sixth word is the destination address where data is stored in local memory or PCI. This address is driven on the internal bus. This value is loaded into the Destination Address Register.
- Seventh word is the Byte Count value. This value specifies the number of bytes of data in the current chain descriptor. This value is loaded into the Accelerator Byte Count Register.
- Eighth word is the Descriptor Control Word. This word configures the Application Accelerator for one operation. This value is loaded into the Accelerator Descriptor Control Register.

There are no data alignment requirements for any source addresses or destination address. However, maximum performance is obtained from aligned transfers, especially small transfers. See Section 1.4.

Refer to Section 1.13 for additional description on the control registers.

1.3.2.2 Eight-Source Descriptor Format

To perform an *XOR-transfer* with up to eight source blocks of data, a special chain descriptor needs to be configured:

- The first part is the four-source descriptor (referred to as the *principal-descriptor*) containing the address of the first 4 source data blocks along with other information.
- The second part (***mini-descriptor***) contains 4, DWORDs containing the address of the additional four (SAR5 - SAR8) source data blocks. The mini-descriptor is written to a contiguous address immediately following the principal descriptor.

To perform a transfer, both parts (principal and mini-descriptor) must be written to local memory. Figure 3 shows the format of this eight-source descriptor. The eight-source descriptor requires twelve contiguous words in local memory and is required to be aligned on an 16-word boundary. All twelve words are required.

Figure 3. Chain Descriptor Format for Eight Source Addresses (XOR Function)

Chain Descriptor in I/O Processor Memory	Description
Next Descriptor Address (NDA)	Address of Next Chain Descriptor
Source Address (D/SAR1)	Source Address for first block of data
Source Address (SAR2)	Source Address for second block of data
Source Address (SAR3)	Source Address for third block of data
Source Address (SAR4)	Source Address for fourth block of data
Destination Address (DAR)	Destination Address of XOR-ed data
Byte Count (BC)	Number of bytes to XOR
Descriptor Control (DC)	Descriptor Control
Source Address (SAR5)	Source Address for fifth data block
Source Address (SAR6)	Source Address for sixth data block
Source Address (SAR7)	Source Address for seventh data block
Source Address (SAR8)	Source Address for eighth data block

- The first eight words are defined in the four-source descriptor definition. See Section 1.3.2.1, “Principle / Four-Source Descriptor Format” for the definition of these words.
- The ninth word (1st word of mini-descriptor) is the address of the fifth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR5.
- The tenth word (2nd word of mini-descriptor) is the address of the sixth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR6.
- The eleventh word (3rd word of mini-descriptor) is the address of the seventh block of data resident in local memory. This address is driven on the internal bus. This value is loaded SAR7.
- The twelfth word (4th word of mini-descriptor) is the address of the eighth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 8.

1.3.2.3 Sixteen-Source Descriptor Format

To perform an *XOR-transfer* with up to sixteen source blocks of data, a special chain descriptor needs to be configured:

- The first part (***principal-descriptor***) contains the address of the first 4 source data blocks along with other information.
- The second part (***mini-descriptor***) contains four, DWORDs containing the address of the additional four (SAR5 - SAR8) source data blocks. The mini-descriptor is written to a contiguous address immediately following the principal descriptor.
- The third part (***extended-descriptor 0***) contains nine, DWORDs containing the address of the additional eight (SAR9 - SAR16) source data blocks and the command/control for these data blocks. The extended-descriptor 0 is written to a contiguous address immediately following the mini descriptor.

To perform a transfer, all three parts (principal descriptor, mini-descriptor and extended-descriptor 0) must be written to local memory. Figure 4 shows the format of this configuration. Every descriptor requires twenty one contiguous words in local memory and is required to be aligned on an 32-word boundary. All twenty one words are required.

Figure 4. Chain Descriptor Format for Sixteen Source Addresses (XOR Function)

Chain Descriptor in Intel XScale® Core Memory	Description
Next Descriptor Address (NDA)	Address of Next Chain Descriptor
Source Address (D/SAR1)	Source Address for first block of data
Source Address (SAR2)	Source Address for second block of data
Source Address (SAR3)	Source Address for third block of data
Source Address (SAR4)	Source Address for fourth block of data
Destination Address (DAR)	Destination Address of XOR-ed data
Byte Count (BC)	Number of bytes to XOR
Descriptor Control (DC)	Descriptor Control
Source Address (SAR5)	Source Address for fifth data block
Source Address (SAR6)	Source Address for sixth data block
Source Address (SAR7)	Source Address for seventh data block
Source Address (SAR8)	Source Address for eighth data block
Extended Descriptor Control 0 (EDC0)	Extended Descriptor 0 control
Source Address (SAR9)	Source Address for ninth block of data
Source Address (SAR10)	Source Address for tenth block of data
Source Address (SAR11)	Source Address for eleventh block of data
Source Address (SAR12)	Source Address for twelfth block of data
Source Address (SAR13)	Source Address for thirteenth block of data
Source Address (SAR14)	Source Address for fourteenth block of data
Source Address (SAR15)	Source Address for fifteenth block of data
Source Address (SAR16)	Source Address for sixteenth block of data

- The first eight words are defined in the four-source descriptor definition. See Section 1.3.2.1, “Principle / Four-Source Descriptor Format” for the definition of these words.
- Words nine through twelve are defined in the eight-source descriptor definition. See Section 1.3.2.2, “Eight-Source Descriptor Format” for the definition of these words.
- The thirteenth word (1st word of extended-descriptor 0) is the Extended Descriptor Control Word 0. This word configures the Application Accelerator for one operation. The value is loaded into the Extended Descriptor Control Register 0.
- The fourteenth word (2nd word of extended-descriptor 0) is the address of the ninth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 9.
- The fifteenth word (3rd word of extended-descriptor 0) is the address of the tenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 10.
- The sixteenth word (4th word of extended-descriptor 0) is the address of the eleventh block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 11.
- The seventeenth word (5th word of extended-descriptor 0) is the address of the twelfth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 12.
- The eighteenth word (6th word of extended-descriptor 0) is the address of the thirteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 13.
- The nineteenth word (7th word of extended-descriptor 0) is the address of the fourteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 14.
- The twentieth word (8th word of extended-descriptor 0) is the address of the fifteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 15.
- The twenty first word (9th word of extended-descriptor 0) is the address of the sixteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 16.

1.3.2.4 Thirty-two-Source Descriptor Format

To perform an *XOR-transfer* with up to thirty two source blocks of data, a special chain descriptor needs to be configured:

- The first part (***principal-descriptor***) contains the address of the first 4 source data blocks along with other information.
- The second part (***mini-descriptor***) contains four, DWORDs containing the address of the additional four (SAR5 - SAR8) source data blocks. The mini-descriptor is written to a contiguous address immediately following the principal descriptor.
- The third part (***extended-descriptor 0***) contains nine, DWORDs containing the address of the additional eight (SAR9 - SAR16) source data blocks and the command/control for these data blocks. The extended-descriptor 0 is written to a contiguous address immediately following the mini descriptor.
- The fourth part (***extended-descriptor 1***) contains nine, DWORDs containing the address of the additional eight (SAR17 - SAR24) source data blocks and the command/control for these data blocks. The extended-descriptor 1 is written to a contiguous address immediately following extended-descriptor 0.
- The fifth part (***extended-descriptor 2***) contains nine, DWORDs containing the address of the additional eight (SAR25 - SAR32) source data blocks and the command/control for these data blocks. The extended-descriptor 2 is written to a contiguous address immediately following extended-descriptor 1.

To perform a transfer, all five parts (principal descriptor, mini-descriptor, extended-descriptor 0, extended-descriptor 1, and extended-descriptor 2) must be written to local memory. Figure 5 shows the format of this configuration. The full descriptor requires thirty nine contiguous words in local memory and is required to be aligned on an 64-word boundary. All thirty nine words are required.

Figure 5. Chain Descriptor Format for Thirty Two Source Addresses (XOR Function)

Chain Descriptor in I/O Processor Memory	Description
Next Descriptor Address (NDA)	Address of Next Chain Descriptor
Source Address (D/SAR1)	Source Address for first block of data
Source Address (SAR2)	Source Address for second block of data
Source Address (SAR3)	Source Address for third block of data
Source Address (SAR4)	Source Address for fourth block of data
Destination Address (DAR)	Destination Address of XOR-ed data
Byte Count (BC)	Number of bytes to XOR
Descriptor Control (DC)	Descriptor Control
Source Address (SAR5)	Source Address for fifth data block
Source Address (SAR6)	Source Address for sixth data block
Source Address (SAR7)	Source Address for seventh data block
Source Address (SAR8)	Source Address for eighth data block
Extended Descriptor Control 0 (EDC0)	Extended Descriptor 0 control
Source Address (SAR9)	Source Address for ninth block of data
Source Address (SAR10)	Source Address for tenth block of data
Source Address (SAR11)	Source Address for eleventh block of data
Source Address (SAR12)	Source Address for twelfth block of data
Source Address (SAR13)	Source Address for thirteenth block of data
Source Address (SAR14)	Source Address for fourteenth block of data
Source Address (SAR15)	Source Address for fifteenth block of data
Source Address (SAR16)	Source Address for sixteenth block of data
Extended Descriptor Control 1 (EDC1)	Extended Descriptor 1 control
Source Address (SAR17)	Source Address for seventeenth block of data
Source Address (SAR18)	Source Address for eighteenth block of data
Source Address (SAR19)	Source Address for nineteenth block of data
Source Address (SAR20)	Source Address for twentieth block of data
Source Address (SAR21)	Source Address for twenty first block of data
Source Address (SAR22)	Source Address for twenty second block of data
Source Address (SAR23)	Source Address for twenty third block of data
Source Address (SAR24)	Source Address for twenty fourth block of data
Extended Descriptor Control 2 (EDC2)	Extended Descriptor 2 control
Source Address (SAR25)	Source Address for twenty fifth block of data
Source Address (SAR26)	Source Address for twenty sixth block of data
Source Address (SAR27)	Source Address for twenty seventh block of data
Source Address (SAR28)	Source Address for twenty eighth block of data
Source Address (SAR29)	Source Address for twenty ninth block of data
Source Address (SAR30)	Source Address for thirtieth block of data
Source Address (SAR31)	Source Address for thirty first block of data
Source Address (SAR32)	Source Address for thirty second block of data

- The first eight words are defined in the Four-source descriptor definition. See Section 1.3.2.1, “Principle / Four-Source Descriptor Format” for the definition of these words.
- Words nine through twelve are defined in the Eight-source descriptor definition. See Section 1.3.2.2, “Eight-Source Descriptor Format” for the definition of these words.
- Words thirteen through twenty-one are defined in the Sixteen-source descriptor definition. See Section 1.3.2.3, “Sixteen-Source Descriptor Format” for the definition of these words.
- The twenty second word (1st word of extended-descriptor 1) is the Extended Descriptor Control Word 1. This word configures the Application Accelerator for one operation. The value is loaded into the Extended Descriptor Control Register 1.
- The twenty third word (2nd word of extended-descriptor 1) is the address of the seventeenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR17.
- The twenty fourth word (3rd word of extended-descriptor 1) is the address of the eighteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 18.
- The twenty fifth word (4th word of extended-descriptor 1) is the address of the nineteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 19.
- The twenty sixth word (5th word of extended-descriptor 1) is the address of the twentieth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 20.
- The twenty seventh word (6th word of extended-descriptor 1) is the address of the twenty first block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 21.
- The twenty eighth word (7th word of extended-descriptor 1) is the address of the twenty second block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 22.
- The twenty ninth word (8th word of extended-descriptor 1) is the address of the twenty third block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 23.
- The thirtieth word (9th word of extended-descriptor 1) is the address of the twenty fourth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 24.
- The thirty first word (1st word of extended-descriptor 2) is the Extended Descriptor Control Word 2. This word configures the Application Accelerator for one operation. The value is loaded into the Extended Descriptor Control Register 2.
- The thirty second word (2nd word of extended-descriptor 2) is the address of the twenty fifth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 25.
- The thirty third word (3rd word of extended-descriptor 2) is the address of the twenty sixth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 26.
- The thirty fourth word (4th word of extended-descriptor 2) is the address of the twenty seventh block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 27.

- The thirty fifth word (5th word of extended-descriptor 2) is the address of the twenty eighth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 28.
- The thirty sixth word (6th word of extended-descriptor 2) is the address of the twenty ninth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 29.
- The thirty seventh word (7th word of extended-descriptor 2) is the address of the thirtieth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 30.
- The thirty eighth word (8th word of extended-descriptor 2) is the address of the thirty first block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 31.
- The thirty ninth word (9th word of extended-descriptor 2) is the address of the thirty second block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 32.

1.3.2.5 Dual-XOR-Transfer Descriptor Format

To perform a *Dual-XOR-transfer*, a special chain descriptor needs to be configured:

- The descriptor contains addresses for 4 source data blocks and 2 destination data buffers along with other information.
- This descriptor format is only valid when the *Dual-XOR-transfer* Enable bit (bit 27) in the Descriptor Control word is set.
- The format is based on the Eight Source Descriptor for *XOR-transfer*s, and the control registers of the corresponding words take on a different meaning when processing this descriptor.

Figure 6 shows the format of this *Dual-XOR-transfer* descriptor. The *Dual-XOR-transfer* descriptor requires nine contiguous words in local memory and is required to be aligned on an 16-word boundary. All nine words are required.

Figure 6. Chain Descriptor Format for *Dual-XOR-transfer*

Chain Descriptor in I/O Processor Memory	Description
Next Descriptor Address (NDA)	Address of Next Chain Descriptor
Source Address (SAR1)	Source Address for first block of data
Source Address (SAR2)	Source Address for second block of data
Horizontal Source Address (SAR_H)	Source Address for Horizontal data
Diagonal Source Address (SAR_D)	Source Address for Diagonal data
Horizontal Destination Address (DAR_H)	Destination Address of Horizontal XOR-ed data
Byte Count (BC)	Number of bytes to XOR
Descriptor Control (DC)	Descriptor Control
Diagonal Destination Address (DAR_D)	Destination Address for Diagonal XOR-ed data

Each word in the chain descriptor is analogous to control register values. Bit definitions for the words in the chain descriptor are the same as for the control registers.

- First word is local memory address of next chain descriptor. A value of zero specifies the end of the chain. This value is loaded into the Accelerator Next Descriptor Address Register. Because chain descriptors must be aligned on a minimum 8-word boundary, the unit may ignore bits 04:00 of this address.
- Second word is the address of the first block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into the Data / Source Address Register 1 (SAR1).
- Third word is the address of the second block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into the Source Address Register 2 (SAR2)
- Fourth word is the address of the third block of data resident in local memory. This address will be driven on the internal bus. This source is referred to as the Horizontal source, and is associated with the Horizontal Destination for the *Dual-XOR-transfer*. This value is loaded into the Horizontal Source Address Register (SAR3/SAR_H).
- Fifth word is the address of the fourth block of data resident in local memory. This address will be driven on the internal bus. This source is referred to as the Diagonal source, and is associated with the Diagonal Destination for the *Dual-XOR-transfer*. This value is loaded into the Diagonal Source Address Register 4 (SAR4/SAR_D).

- Sixth word is the destination address where the first XOR result will be stored in local memory. This address will be driven on the internal bus. This destination is referred to as the Horizontal Destination, and is associated with the Horizontal Source for the *Dual-XOR-transfer*. This value is loaded into the Destination Address Register (DAR/DAR_H).
- Seventh word is the Byte Count value. This value specifies the number of bytes of data in the current chain descriptor. This value is loaded into the Accelerator Byte Count Register.
- Eighth word is the Descriptor Control Word. This word configures the Application Accelerator for one operation. This value is loaded into the Accelerator Descriptor Control Register.
- The ninth word is the destination address where the second XOR result will be stored in local memory. This address will be driven on the internal bus. This destination is referred to as the Diagonal Destination, and is associated with the Diagonal Source for the *Dual-XOR-transfer*. This value is loaded into the Diagonal Destination Address Register (SAR5/DAR_D).

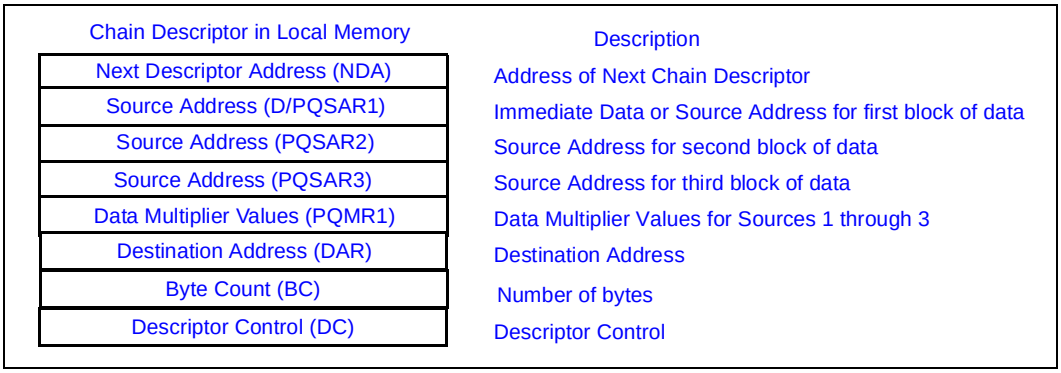
There are no data alignment requirements for any source addresses. While the destinations addresses (Horizontal and Diagonal) also have no data alignment requirements relative to memory, the alignment of the Horizontal and Diagonal destination addresses must match (relative to 16 Byte address).

Refer to Section 1.13 for additional description on the control registers.

1.3.2.6 P+Q Three-Source Descriptor Format

Figure 7 shows the format of an individual chain descriptor when P+Q RAID-6 is enabled. This three-source descriptor is the smallest supported descriptor for P+Q RAID-6 operations. The three-source descriptor requires eight contiguous words in 80331 local memory and is required to be aligned on an 8-word boundary. All eight words are required.

Figure 7. P+Q Base Chain Descriptor Format



Each word in the chain descriptor is analogous to control register values. Bit definitions for the words in the chain descriptor are the same as for the control registers.

- First word is local memory address of next chain descriptor. A value of zero specifies end of chain. Value is loaded into the Accelerator Next Descriptor Address Register. Because chain descriptors must be aligned on a minimum 8-word boundary, unit may ignore bits 04:00 of this address.
- Second word is address of the first block of data resident in local memory, or immediate data when performing a Memory Block Fill. This value is loaded into the Data / P+Q RAID-6 Source Address Register 1 (D/PQSAR1).
- Third word is address of second block of data resident in local memory. This address is driven on the internal bus. This value is loaded into the P+Q RAID-6 Source Address Register 2 (PQSAR2).
- Fourth word is address of third block of data resident in local memory and is driven on the internal bus. This value is loaded into P+Q RAID-6 Source Address Register 3 (PQSAR3).
- Fifth word contains Data Multiplier Values (DMLTx) for source addresses 1 through 3. These bytes are used as control input for GF Multiply of corresponding source. The respective byte will be driven to the GF Multiply when source data is being fetched. The lowest order byte of this word contains the data multiplier for source address 1, the second byte for source address 2, the third byte for source address 3, and the highest order byte is not used and is reserved. This value is loaded into the P+Q RAID-6 GF Multiply Multiplier Register 1 (GFMR1).
- Sixth word is the destination address where data will be stored in local memory. This address will be driven on the internal bus. This value is loaded into the Destination Address Register.
- Seventh word is the Byte Count value. This value specifies the number of bytes of data in the current chain descriptor. This value is loaded into the Accelerator Byte Count Register.
- Eighth word is the Descriptor Control Word. This word configures the Application Accelerator for one operation. This value is loaded into the Accelerator Descriptor Control Register.

There are no data alignment requirements for any source addresses or destination address. However, maximum performance is obtained from aligned transfers, especially small transfers. See Section 1.4.

Refer to Section 1.13 for additional description on the control registers.

1.3.2.7 P+Q Six-Source Descriptor Format

To perform an P+Q RAID-6 *XOR-transfer* with up to six source blocks of data, a special chain descriptor needs to be configured:

- First part: three-source descriptor (referred to as *principal-descriptor*) containing source address and data multiplier values of first 3 source data blocks along with other information.
- Second part: (***mini-descriptor***) contains 4 DWORDs containing the address of the additional three source data blocks and Data Multiplier values. The mini-descriptor is written to a contiguous address immediately following the principal descriptor.

To perform a transfer, both parts (principal and mini-descriptor) must be written to local memory. Figure 8 shows the format of this eight-source descriptor. The six-source descriptor requires twelve contiguous words in local memory and is required to be aligned on an 16-word boundary. All twelve words are required.

Figure 8. P+Q Chain Descriptor Format for Six Source Addresses (XOR Function)

Chain Descriptor in I/O Processor Memory	Description
Next Descriptor Address (NDA)	Address of Next Chain Descriptor
Source Address (PQSAR1)	Source Address for first block of data
Source Address (PQSAR2)	Source Address for second block of data
Source Address (PQSAR3)	Source Address for third block of data
Data Multiplier Values (GFMR1)	Data Multiplier Values for Sources 1 through 3
Destination Address (DAR)	Destination Address of XOR-ed data
Byte Count (BC)	Number of bytes to XOR
Descriptor Control (DC)	Descriptor Control
Source Address (PQSAR4)	Source Address for fourth data block
Source Address (PQSAR5)	Source Address for fifth data block
Source Address (PQSAR6)	Source Address for sixth data block
Data Multiplier Values (GFMR2)	Data Multiplier Values for Sources 4 through 6

- The first eight words are defined in the three-source descriptor definition. See Section 1.3.2.1 for the definition of these words.
- The ninth word (1st word of mini-descriptor) is the address of the fourth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR4.
- The tenth word (2nd word of mini-descriptor) is the address of the fifth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR5.
- The eleventh word (3rd word of mini-descriptor) is the address of the sixth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded PQSAR6
- The twelfth word contains the Data Multiplier Values (DMLT) for source addresses 4 through 6. These bytes are used as the control input for the GF Multiply of the corresponding source. The respective byte will be driven to the GF Multiply when source data is being fetched. The lowest order byte of this word contains the data multiplier for source address 4, the second byte for source address 5, the third byte for source address 6, and the highest order byte is not used and is reserved. This value is loaded into the P+Q RAID-6 GF Multiply Multiplier Register 2 (GFMR2).

1.3.2.8 P+Q Twelve-Source Descriptor Format

To perform an *XOR-transfer* with a GF Multiply data multiplier on up to twelve source blocks of data, a special chain descriptor needs to be configured:

- The first part (***principal-descriptor***) contains the address of the first 3 source data blocks and data multiplier values along with other information.
- The second part (***mini-descriptor***) contains four, DWORDs containing the address of the additional three source data blocks and data multiplier values. The mini-descriptor is written to a contiguous address immediately following the principal descriptor.
- The third part (***extended-descriptor 0***) contains nine, DWORDs containing the address of the additional six source data blocks and data multiplier values. The extended-descriptor 0 is written to a contiguous address immediately following the mini descriptor.

To perform a transfer, all three parts (principal descriptor, mini-descriptor and extended-descriptor 0) must be written to local memory. Figure 9 shows the format of this configuration. Every descriptor requires twenty one contiguous words in local memory and is required to be aligned on an 32-word boundary. All twenty one words are required.

Figure 9. P+Q Chain Descriptor Format for Twelve Source Addresses (XOR Function)

Chain Descriptor in Intel XScale® Core Memory	Description
Next Descriptor Address (NDA)	Address of Next Chain Descriptor
Source Address (D/PQSAR1)	Source Address for first block of data
Source Address (PQSAR2)	Source Address for second block of data
Source Address (PQSAR3)	Source Address for third block of data
Data Multiplier Values (GFMR1)	Data Multiplier Values for Sources 1 through 3
Destination Address (DAR)	Destination Address of XOR-ed data
Byte Count (BC)	Number of bytes to XOR
Descriptor Control (DC)	Descriptor Control
Source Address (PQSAR4)	Source Address for fourth data block
Source Address (PQSAR5)	Source Address for fifth data block
Source Address (PQSAR6)	Source Address for sixth data block
Data Multiplier Values (GFMR2	Data Multiplier Values for Sources 4 through 6
Extended Descriptor Control 0 (EDC0)	Extended Descriptor 0 control
Source Address (PQSAR7)	Source Address for seventh block of data
Source Address (PQSA8)R	Source Address for eighth block of data
Source Address (PQSAR9)	Source Address for ninth block of data
Data Multiplier Values (GFMR3)	Data Multiplier Values for Sources 7 through 9
Source Address (PQSAR10)	Source Address for tenth block of data
Source Address (PQSAR11)	Source Address for eleventh block of data
Source Address (PQSAR12)	Source Address for twelfth block of data
Data Multiplier Values (GFMR4)	Data Multiplier Values for Sources 10 through 12

- The first eight words are defined in the three-source descriptor definition. See Section 1.3.2.6, “P+Q Three-Source Descriptor Format” for the definition of these words.
- Words nine through twelve are defined in the six-source descriptor definition. See Section 1.3.2.7, “P+Q Six-Source Descriptor Format” for the definition of these words.
- The thirteenth word (1st word of extended-descriptor 0) is the Extended Descriptor Control Word 0. This word configures the Application Accelerator for one operation. The value is loaded into the Extended Descriptor Control Register 0.
- The fourteenth word (2nd word of extended-descriptor 0) is the address of the seventh block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR7.
- The fifteenth word (3rd word of extended-descriptor 0) is the address of the eighth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR8.
- The sixteenth word (4th word of extended-descriptor 0) is the address of the ninth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR9.
- The seventeenth word (5th word of extended-descriptor 0) contains the Data Multiplier Values (DMLTx) for source addresses 7 through 9. These bytes are used as the control input for the TDIfn of the corresponding source. The respective byte will be driven to the GF Multiply when source data is being fetched. The lowest order byte of this word contains the data multiplier for source address 7, the second byte for source address 8, the third byte for source address 9, and the highest order byte is not used and is reserved. This value is loaded into the P+Q RAID-6 GF Multiply Multiplier Register 3 (GFMR3).
- The eighteenth word (6th word of extended-descriptor 0) is the address of the tenth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR10.
- The nineteenth word (7th word of extended-descriptor 0) is the address of the eleventh block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR11.
- The twentieth word (8th word of extended-descriptor 0) is the address of the twelfth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR12.
- The twenty first word (9th word of extended-descriptor 0) contains the Data Multiplier Values (DMLTx) for source addresses 10 through 12. These bytes are used as the control input for the GF Multiply of the corresponding source. The respective byte will be driven to the GF Multiply when source data is being fetched. The lowest order byte of this word contains the data multiplier for source address 10, the second byte for source address 11, the third byte for source address 12, and the highest order byte is not used and is reserved. This value is loaded into the P+Q RAID-6 GF Multiply Multiplier Register 4 (GFMR4).

1.3.2.9 P+Q Sixteen-Source Descriptor Format

To perform an P+Q RAID-6 *XOR-transfer* with up to sixteen source blocks of data, a special chain descriptor needs to be configured:

- The first part (***principal-descriptor***) contains the address of the first 3 source data blocks (PQSAR1 - PQSAR3) and data multiplier values along with other information.
- The second part (***mini-descriptor***) contains four, DWORDs containing the address of the additional three (PQSAR4 - PQSAR6) source data blocks and data multiplier values. The mini-descriptor is written to a contiguous address immediately following the principal descriptor.
- The third part (***extended-descriptor 0***) contains nine, DWORDs containing the address of the additional six (PQSAR7 - PQSAR12) source data blocks and data multiplier values. The extended-descriptor 0 is written to a contiguous address immediately following the mini descriptor.
- The fourth part (***extended-descriptor 1***) contains nine, DWORDs containing the address of the additional four (PQSAR13 - PQSAR16) source data blocks and data multiplier values along with the command/control for these data blocks. The extended-descriptor 1 is written to a contiguous address immediately following extended-descriptor 0.

To perform a transfer, all four parts (principal descriptor, mini-descriptor, extended-descriptor 0, and extended-descriptor 1) must be written to local memory. Figure 10 shows the format of this configuration. The full descriptor requires twenty-seven contiguous words in local memory and is required to be aligned on an 32-word boundary. All twenty-seven words are required.

Figure 10. P+Q Chain Descriptor Format for Sixteen Source Addresses (XOR Function)

Chain Descriptor in I/O Processor Memory	Description
Next Descriptor Address (NDA)	Address of Next Chain Descriptor
Source Address (D/PQSAR1)	Source Address for first block of data
Source Address (PQSAR2)	Source Address for second block of data
Source Address (PQSAR3)	Source Address for third block of data
Data Multiplier Values (GFMR1)	Data Multiplier Values for Sources 1 through 3
Destination Address (DAR)	Destination Address of XOR-ed data
Byte Count (BC)	Number of bytes to XOR
Descriptor Control (DC)	Descriptor Control
Source Address (PQSAR4)	Source Address for fourth data block
Source Address (PQSAR5)	Source Address for fifth data block
Source Address (PQSAR6)	Source Address for sixth data block
Data Multiplier Values (GFMR2)	Data Multiplier Values for Sources 4 through 6
Extended Descriptor Control 0 (EDC0)	Extended Descriptor 0 control
Source Address (PQSAR7)	Source Address for seventh block of data
Source Address (PQSAR8)	Source Address for eighth block of data
Source Address (PQSAR9)	Source Address for ninth block of data
Data Multiplier Values (GFMR3)	Data Multiplier Values for Sources 4 through 9
Source Address (PQSAR10)	Source Address for tenth block of data
Source Address (PQSAR11)	Source Address for eleventh block of data
Source Address (PQSAR12)	Source Address for twelfth block of data
Data Multiplier Values (GFMR4)	Data Multiplier Values for Sources 10 through 12
Reserved	Reserved - not used
Source Address (PQSAR13)	Source Address for thirteenth block of data
Source Address (PQSAR14)	Source Address for fourteenth block of data
Source Address (PQSAR15)	Source Address for fifteenth block of data
Source Address (PQSAR16)	Source Address for sixteenth block of data
Data Multiplier Values (GFMR5)	Data Multiplier Values for Sources 13 through 16

- The first eight words are defined in the three-source descriptor definition. See Section 1.3.2.6, "P+Q Three-Source Descriptor Format" for the definition of these words.
- Words nine through twelve are defined in the six-source descriptor definition. See Section 1.3.2.7, "P+Q Six-Source Descriptor Format" for the definition of these words.
- Words thirteen through twenty-one are defined in the Twelve-source descriptor definition. See Section 1.3.2.8, "P+Q Twelve-Source Descriptor Format" for the definition of these words.
- The twenty second word (1st word of extended-descriptor 1) is reserved and not used by the AA in the processing of this descriptor.
- The twenty third word (2nd word of extended-descriptor 1) is the address of the thirteenth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR12.
- The twenty fourth word (3rd word of extended-descriptor 1) is the address of the fourteenth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR12.
- The twenty fifth word (4th word of extended-descriptor 1) is the address of the fifteenth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR12.
- The twenty sixth word (5th word of extended-descriptor 1) is the address of the sixteenth block of data resident in local memory. This address will be driven on the internal bus. This value is loaded into PQSAR12.
- The twenty seventh word (6th word of extended-descriptor 1) contains the Data Multiplier Values (DMLTx) for source addresses 13 through 16. These bytes are used as the control input for the GF Multiply of the corresponding source. The respective byte will be driven to the GF Multiply when source data is being fetched. The lowest order byte of this word contains the data multiplier for source address 13, the second byte for source address 14, the third byte for source address 15, and the highest order byte for source address 16. This value is loaded into the P+Q RAID-6 GF Multiply Multiplier Register 5 (GFMR5).

Note: The highest order byte (bits[31:24]) is unused in other Data Multiplier words GFMR[4:1], but is defined in GFMR5.

1.3.3 Descriptor Summary

Table 2 summarizes the content of the descriptors defined in previous sections.

Table 2. Descriptor Summary (Sheet 1 of 2)

Register Address	4-Source XOR	8-Source XOR	16-Source XOR	32-Source XOR	Dual XOR	P+Q 3-Source	P+Q 6-Source	P+Q 12-Source	P+Q 16-Source
FFFF E80Ch	NDA	NDA	NDA	NDA	NDA	NDA	NDA	NDA	NDA
FFFF E810h	SAR1	SAR1	SAR1	SAR1	SAR1	SAR1	SAR1	SAR1	SAR1
FFFF E814h	SAR2	SAR2	SAR2	SAR2	SAR2	SAR2	SAR2	SAR2	SAR2
FFFF E818h	SAR3	SAR3	SAR3	SAR3	SAR_H	SAR3	SAR3	SAR3	SAR3
FFFF E81Ch	SAR4	SAR4	SAR4	SAR4	SAR_D	GFMR1	GFMR1	GFMR1	GFMR1
FFFF E820h	DAR	DAR	DAR	DAR	DAR_H	DAR	DAR	DAR	DAR
FFFF E824h	BC	BC	BC	BC	BC	BC	BC	BC	BC
FFFF E828h	DC	DC	DC	DC	DC	DC	DC	DC	DC
FFFF E82Ch		SAR5	SAR5	SAR5	DAR_D		SAR4	SAR4	SAR4
FFFF E830h		SAR6	SAR6	SAR6			SAR5	SAR5	SAR5
FFFF E834h		SAR7	SAR7	SAR7			SAR6	SAR6	SAR6
FFFF E838h		SAR8	SAR8	SAR8			GFMR2	GFMR2	GFMR2
FFFF E83Ch			EDCR0	EDCR0				EDCR0	EDCR0
FFFF E840h			SAR9	SAR9				SAR7	SAR7
FFFF E844h			SAR10	SAR10				SAR8	SAR8
FFFF E848h			SAR11	SAR11				SAR9	SAR9
FFFF E84Ch			SAR12	SAR12				GFMR3	GFMR3
FFFF E850h			SAR13	SAR13				SAR10	SAR10
FFFF E854h			SAR14	SAR14				SAR11	SAR11
FFFF E858h			SAR15	SAR15				SAR12	SAR12
FFFF E85Ch			SAR16	SAR16				GFMR4	GFMR4

Table 2. Descriptor Summary (Sheet 2 of 2)

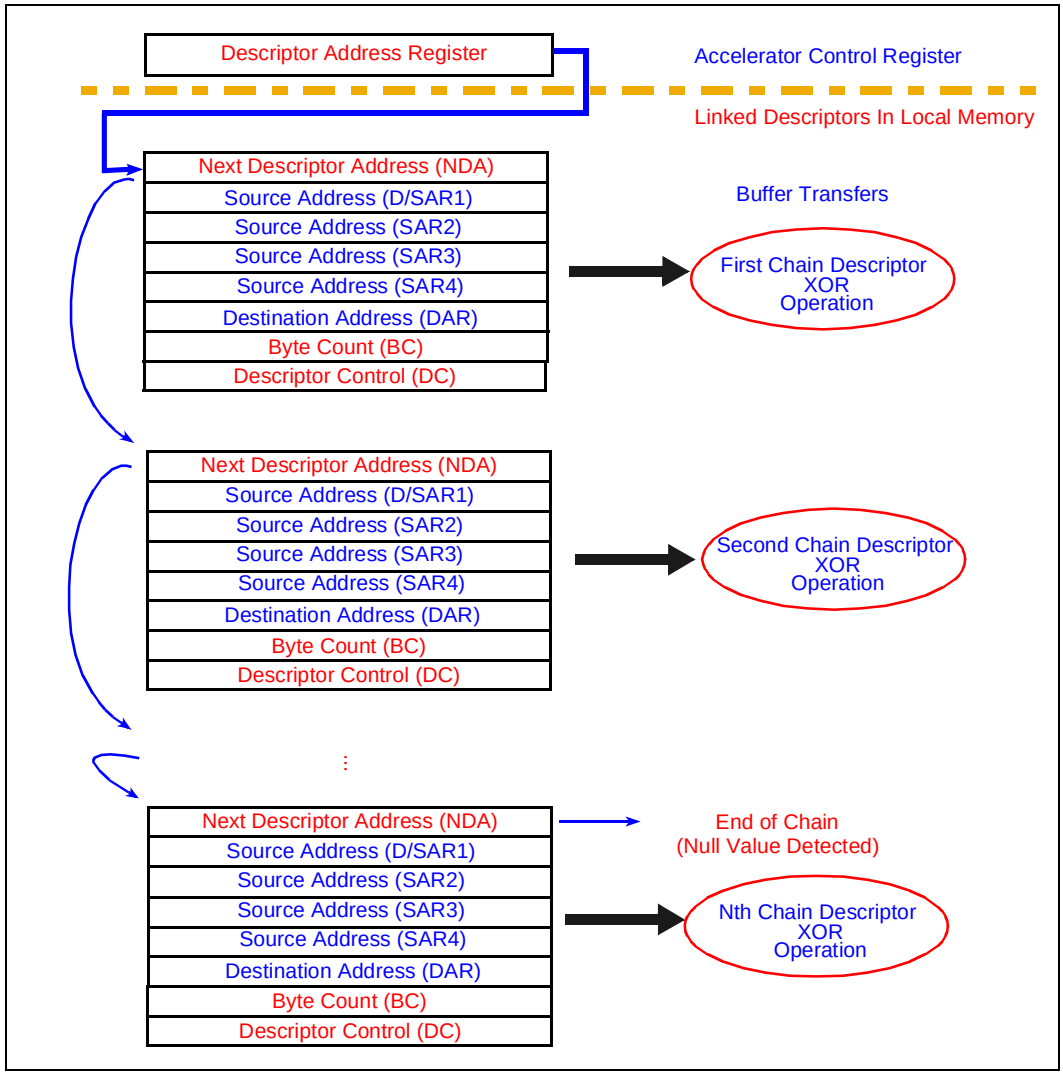
Register Address	4-Source XOR	8-Source XOR	16-Source XOR	32-Source XOR	Dual XOR	P+Q 3-Source	P+Q 6-Source	P+Q 12-Source	P+Q 16-Source
FFFF E860h				EDCR1					rsvd
FFFF E864h				SAR17					SAR13
FFFF E868h				SAR18					SAR14
FFFF E86Ch				SAR19					SAR15
FFFF E870h				SAR20					SAR16
FFFF E874h				SAR21					GFMR5
FFFF E878h				SAR22					
FFFF E87Ch				SAR23					
FFFF E880h				SAR24					
FFFF E884h				EDCR2					
FFFF E888h				SAR25					
FFFF E88Ch				SAR26					
FFFF E890h				SAR27					
FFFF E894h				SAR28					
FFFF E898h				SAR29					
FFFF E89Ch				SAR30					
FFFF E8A0h				SAR31					
FFFF E8A4h				SAR32					

1.3.4 Descriptor Chaining

To perform an AA operation, a series of chain descriptors can be built in local memory to operate on multiple blocks of source data resident in local memory. The result can then be stored back in local memory. An application can build multiple chain descriptors to operate on many blocks of data which have different source addresses within the local memory.

When multiple chain descriptors are built in local memory, the application can link each of these chain descriptors using the Next Descriptor Address in the chain descriptor. This address logically links the chain descriptors together. This allows the application to build a list of transfers which may not require the processor until all transfers are complete. Figure 11 shows an example of a linked-list of transfers using only four-source descriptors specified in external memory.

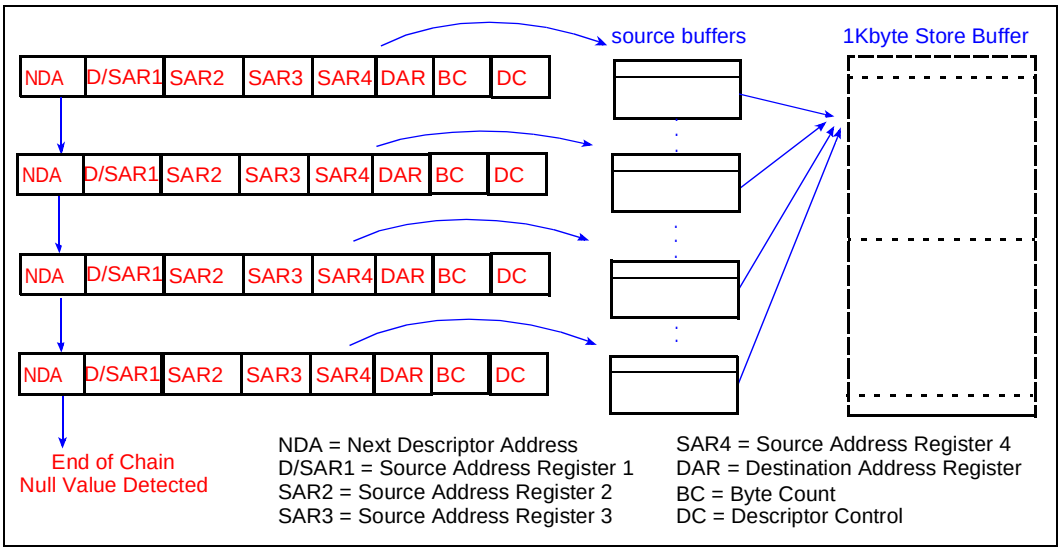
Figure 11. XOR Chaining Operation



1.4 AA Descriptor Processing

An AA operation is initiated by building one or more chain descriptors in Intel XScale® core local memory (ARM* architecture compliant). Figure 12 shows the format of a principal descriptor.

Figure 12. Example of Gather Chaining for Four Source Blocks



The following describes the steps for initiating a new AA operation:

1. The AA must be inactive prior to starting an AA operation. This can be checked by software by reading the *Accelerator Active* bit in the Accelerator Status Register. When this bit is clear, the unit is inactive. When this bit is set, the unit is currently active.
2. The ASR must be cleared of all error conditions.
3. The software writes the address of the first chain descriptor to the Accelerator Next Descriptor Address Register (ANDAR).
4. The software sets the *Accelerator Enable* bit in the Accelerator Control Register (ACR). Because this is the start of a new AA operation and not the resumption of a previous operation, the *Chain Resume* bit in the ACR should be clear.
5. The AA operation starts by reading the ANDAR chain descriptor address. The AA loads the chain descriptor values into the ADAR and begins data transfer. The Accelerator Descriptor Address Register (ADAR) contains the address of the chain descriptor just read and ANDAR now contains the Next Descriptor Address from the chain descriptor just read.

The last AA chain list descriptor has zero in the next descriptor address field specifying the last chain descriptor. A NULL value notifies AA not to read additional chain descriptors from memory.

Once an AA operation is active, it can be temporarily suspended by clearing the *Accelerator Enable* bit in the ACR. Note that this does not abort the AA operation. The unit resumes the process when the *Accelerator Enable* bit is set.

When descriptors are read from external memory, bus latency and memory speed affect chaining latency. Chaining latency is defined as the time required for the AA to access the next chain descriptor plus the time required to set up the next AA operation.

1.4.1 Scatter Gather Transfers

The Application Accelerator can be used to perform typical scatter gather transfers. This consists of programming the chain descriptors to gather data which may be located in non-contiguous blocks of memory. The chain descriptor specifies the destination location such that once all data has been processed, the data is contiguous in memory. Figure 12 shows how the destination pointers can gather data.

1.4.2 Synchronizing a Program to Chained Operation

Any operation involving the AA can be synchronized to a program executing on the Intel XScale® core through the use of processor interrupts. The AA generates an interrupt to the Intel XScale® core under certain conditions. They are:

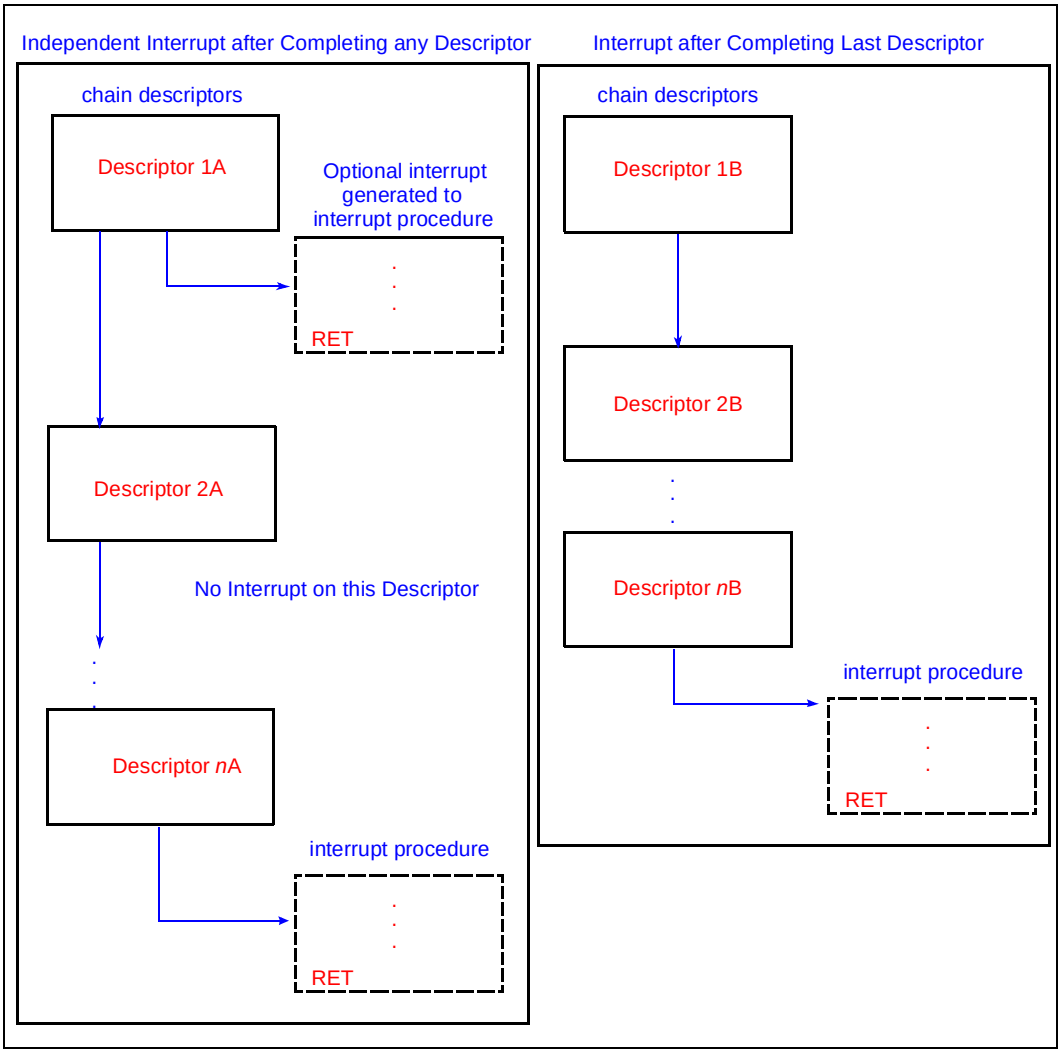
1. [Interrupt and Continue] The AA completes processing a chain descriptor and the Accelerator Next Descriptor Address Register (ANDAR) is non-zero. When the *Interrupt Enable* bit within the Accelerator Descriptor Control Register (ADCR) is set, an interrupt is generated to the Intel XScale® core. This interrupt is for synchronization purposes. The AA sets the *End Of Transfer Interrupt* flag in the Accelerator Status Register (ASR). Since it is not the last chain descriptor in the list, the AA starts to process the next chain descriptor without requiring any processor interaction.
2. [End of Chain] The AA completes processing a chain descriptor and the Accelerator Next Descriptor Address Register is zero specifying the end of the chain. When the *Interrupt Enable* bit within the ADCR is set, an interrupt is generated to the Intel XScale® core. The AA sets the *End Of Chain Interrupt* flag in the ASR.
3. [Error] An error condition occurs (refer to Section 1.11, “Error Conditions” on page 61 for Application Accelerator error conditions) during a transfer. The AA halts operation on the current chain descriptor and not proceed to the next chain descriptor.

Each chain descriptor can independently set the *Interrupt Enable* bit in the Descriptor Control word. This bit enables an independent interrupt once a chain descriptor is processed. This bit can be set or clear within each chain descriptor. Control of interrupt generation within each descriptor aids in synchronization of the executing software with AA operation.

Figure 13 shows two examples of program synchronization. The left column shows program synchronization based on individual chain descriptors. Descriptor 1A generated an interrupt to the processor, while descriptor 2A did not because the *Interrupt Enable* bit was clear. The last

descriptor *n*A, generated an interrupt to signify the end of the chain has been reached. The right column in Figure 13 shows an example where the interrupt was generated only on the last descriptor signifying the end of chain.

Figure 13. Synchronizing to Chained AA Operation



1.4.3 Appending to The End of a Chain

Once the AA has started processing a chain of descriptors, application software may need to append a chain descriptor to the current chain without interrupting the transfer in progress. The mechanism used for performing this action is controlled by the *Chain Resume* bit in the Accelerator Control Register (ACR).

The AA reads the subsequent chain descriptor each time it completes the current chain descriptor and the Accelerator Next Descriptor Address Register (ANDAR) is non-zero. ANDAR always contains the address of the next chain descriptor to be read and the Accelerator Descriptor Address Register (ADAR) always contains the address of the current chain descriptor.

The procedure for appending chains requires the software to find the last chain descriptor in the current chain and change the Next Descriptor Address in that descriptor to the address of the new chain to be appended. The software then sets the *Chain Resume* bit in the ACR. It does not matter when the unit is active or not.

The AA examines the *Chain Resume* bit of the ACR when the unit is idle or upon completion of a chain of transfers. When this bit is set, the AA re-reads the Next Descriptor Address of the current chain descriptor and loads it into ANDAR. The address of the current chain descriptor is contained in ADAR. The AA clears the *Chain Resume* bit and then examines ANDAR. When ANDAR is not zero, the AA reads the chain descriptor using this new address and begins a new operation. When ANDAR is zero, the AA remains or returns to idle.

There are three cases to consider:

1. The AA completes an AA operation and it is not the last descriptor in the chain. In this case, the AA clears the *Chain Resume* bit and reads the next chain descriptor. The appended descriptor is read when the AA reaches the end of the original chain.
2. The channel completes an AA transfer and it is the last descriptor in the chain. In this case, the AA examines the state of the *Chain Resume* bit. When the bit is set, the AA re-reads the current descriptor to get the address of the appended chain descriptor. When the bit is clear, the AA returns to idle.
3. The AA is idle. In this case, the AA examines the state of the *Chain Resume* bit when the ACR is written. When the bit is set, the AA re-reads the last descriptor from the most-recent chain to get the next descriptor address of the appended chain descriptor.

1.5 AA Operations

The AA can be configured on a per descriptor basis through the Descriptor Control Word to perform three distinct operations:

1. In an **XOR operation**, the AA generates a parity data stream in local memory that is comprised of the XOR of up to 32 distinct data streams (i.e., SAR1..32) on a per byte basis. All of the source data streams and the parity data stream can be up to 16 MB long.
2. In a Dual XOR operation, the AA will generate two parity data streams in local memory. The two parity data streams are the Horizontal and Diagonal parities for XOR based RAID-6 Each parity stream is comprised of the XOR of 2 common data streams (SAR1 and SAR2) and 1 distinct data streams (SAR_H, and SAR_D) respectively on a per byte basis. All of the source data streams and the parity data stream can be up to 16 MB long.
3. Perform a **Memory Block Fil**l of up to 16 MB of local memory with a 32-bit constant (DATA/SAR1).
4. With the **Zero Result Buffer Check**, the AA confirms that the XOR of all the bytes of source data results in 00H for the entire byte count. All the source data streams can be up to 16 MB long. The results of the check is written back to the Descriptor Control Word in local memory.

Note: P+Q RAID-6 operation is controlled for the entire AA, and is applicable to all descriptors processed, not on a per-descriptor basis.

Table 3 documents the combination of AA operations, modes and Descriptor Control features which are valid. The typical application usage of each combination is provided. Combinations of descriptor control features not listed are not valid.

Table 3. AA Operation and Command Combination Summary

Descriptor Control Feature			AA Operation (Source Command)	Application Usage Description
Zero-Result Check	Dual-XOR	Destination Write Enable		
0	0	1	XOR	Typical usage for RAID Applications, up to 32 sources for RAID-3, RAID-5 and 2D-XOR RAID-6. (up to16 sources for P+Q RAID-6)
			Block Fill[a]	Memory Block fill with constant data
0	1	1	XOR	Two parity calculation for single strip write I/O in 2D-XOR RAID-6
1	0	0	XOR	Parity Scrub for RAID array w/o saving check buffer (typical use). Can be used in conjunction with P+Q RAID-6 mode.
1	0	1	XOR	Parity Scrub for RAID array with check buffer saved to memory (not typical use). Can be used in conjunction with P+Q RAID-6 mode.
x	x	x	Direct Fill[a]	First Source moved into result buffer (not XORed with current contents) Normal use for RAID applications

a. Specified only in Block 1 Command of Accelerator Descriptor Control.

1.5.1 AA Addressing

All source address operated on by the AA must be local memory addresses. The destination address may be either a local memory address, or a PCI address mapped through the ATU outbound memory windows. Table 4 summarizes the application usage of the AA operations for the valid destination addressing options.

Table 4. Typical AA Operation and Addressing Summary

Descriptor Control Feature[a]		AA Operation (Source Command)	Destination[b] Address	Application Usage Description
Zero-Result Check	Dual-XOR			
0	0	XOR	PCI	RAID Application Degraded Read
0	0	XOR	Local	Typical usage for RAID Applications
		Block Fill[c]	Local	Memory Block fill with constant data
0	1	XOR	Local	Two parity calculation for single strip write I/O in 2D-XOR RAID-6
1	0	XOR	Local	Parity Scrub for RAID array with check buffer saved to memory. Can be used in conjunction with P+Q RAID-6 mode.

a. Destination Write Enable set for all cases listed. Cases with DWE clear are not listed.
b. All AA sources must be local memory addresses.
c. Specified only in Block 1 Command of Accelerator Descriptor Control.

The following sections describes the AA operations in detail.

1.5.2 XOR Operation

Figure 14 describes the XOR algorithm implementation. In this illustrative example, there are four blocks of source data to be XOR-ed. The intermediate result is kept by the store queue in the AA before being written back to local memory. The source data is located at addresses A000 0400H, A000 0800H, A000 0C00H and A000 1000H respectively.

All data transfers needed for this operation are controlled by chain descriptors located in local memory. The Application Accelerator as a master on the internal bus initiates data transfer. The algorithm is implemented such that as data is read from local memory, the boolean unit executes the XOR operation on incoming data.

Figure 14. The Bit-wise XOR Algorithm

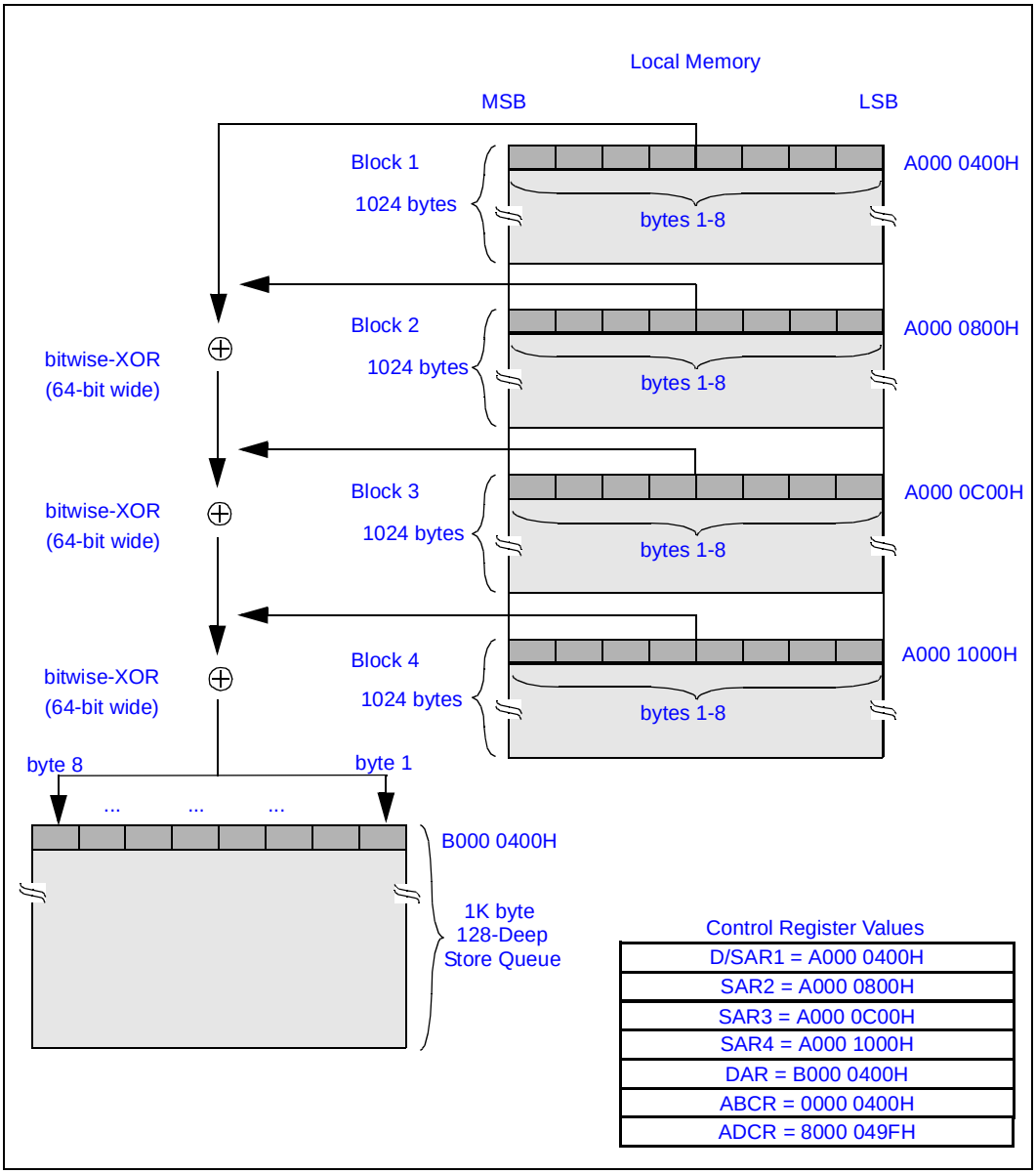
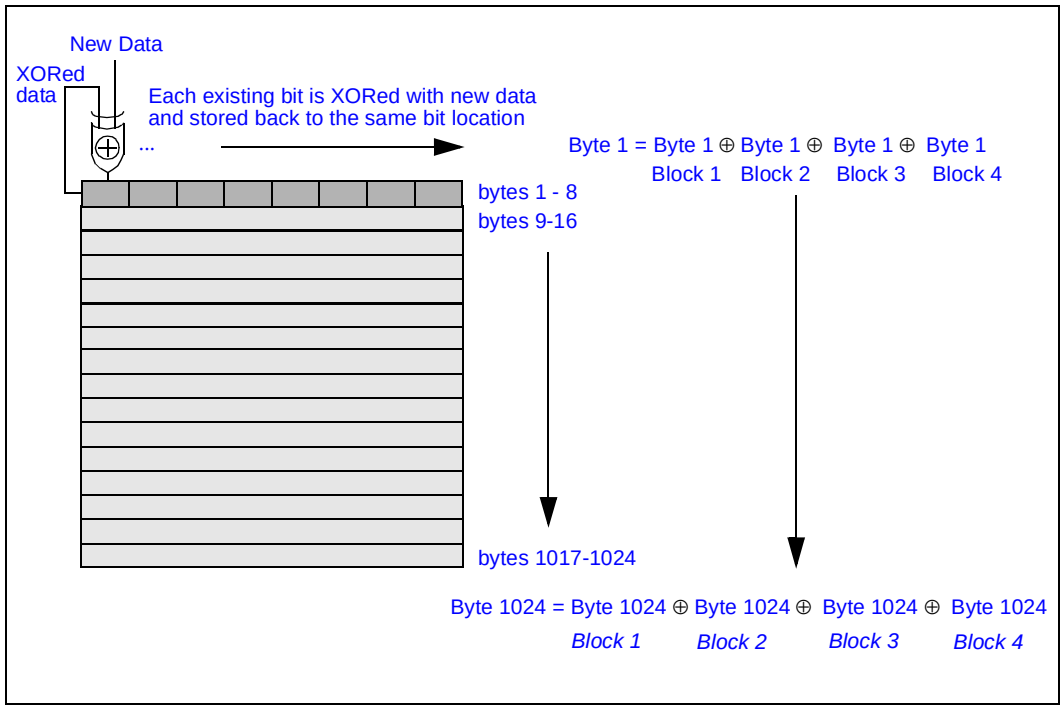


Figure 15. Hardware Assist XOR Unit



The XOR algorithm and methodology followed once a chain descriptor has been configured is detailed below:

1. The Application Accelerator as a master on the bus initiates data transfer from the address pointed at by the First Source Address Register (SAR1). The total number of bytes to *XOR-transfer* is specified by the Byte Count (BC) field in the chain descriptor.
   a. When the Direct Fill command is selected for SAR1, this is designated as the first block of data in the current XOR operation, and the data is transferred directly to the store queue. The number of bytes transferred to the store queue is 1KByte/512Bytes (based on bit 2 of the Accelerator Control Register).
   b. When the XOR command is selected for SAR1, the boolean unit performs the XOR operation on the data currently existing in the store queue with the data being transferred from memory (see steps 3-7 for SAR2). This may be done to XOR more than 32 blocks of data together with a byte count of 1KByte or less.

Note: When the Byte Count Register contains a value greater than the buffer size, the AA completes the *XOR-transfer* operation on the first buffer of data obtained from each Source Register (D/SAR1, SAR2- SAR4), then proceeds with the next buffer of data. This process is repeated until the BCR contains a zero value.

2. The Application Accelerator transfers the first eight bytes of data from the address pointed at by the Second Source Address Register (SAR2).
3. The boolean unit performs the bit-wise XOR algorithm on the input operands. The input operands are the first eight bytes of data read from D/SAR1 (bytes 1-8) which are stored in the queue and the first eight bytes of data just read from SAR2 (bytes 1-8).

4. The XOR-ed result is transferred to the store queue and stored in the first eight bytes (bytes 1-8) overwriting previously stored data.
5. The Application Accelerator transfers the next eight bytes of data (bytes 9-16) from address pointed at by the Second Source Address Register (SAR2).
6. The boolean unit performs the bit-wise XOR algorithm on the input operands. The input operands are the next eight bytes of data read from D/SAR1 (bytes 9-16 stored in the queue) and the eight bytes of data read from SAR2 in Step-5.
7. Step-5 and Step-6 (Data transfer and XOR) are repeated until all data pointed at by SAR1 is XOR-ed with the corresponding data pointed at by SAR2. The store queue now contains a buffer full of XOR-ed data, the source addresses for which were specified in SAR1 and SAR2.
8. Steps 1-7 are repeated once again. The first input to the XOR unit is the data held in the store queue and the second input is the data pointed at by SAR3.
9. The above steps are repeated once more. The first input to the XOR unit is the data held in the store queue and the second input is the data pointed at by SAR4.
10. Once Steps 1-9 are completed, the XOR operation is complete for the first full buffer of the current chain descriptor. When the Destination Write Enable Bit in the Accelerator Descriptor Control Register (ADCR) is set, the data in the store queue is written to local memory at the address pointed to by the Destination Address Register (DAR). When the Destination Write Enable Bit in the ADCR is not set, the data is not written to local memory and is held in the queue. Steps 1-9 are repeated until all the bytes of data have undergone the *XOR-transfer* operation.

Note: The Destination Write Enable bit should be SET when Descriptor Byte Count is larger than the AA buffer size. When the ABCR register contains a value greater than the buffer size and the ADCR.dwe bit is cleared, the AAU only reads the first buffer of data and performs the specified function. It does not read the remaining bytes specified in the ABCR. Furthermore, the AAU proceeds to process the next chain descriptor when it is specified.

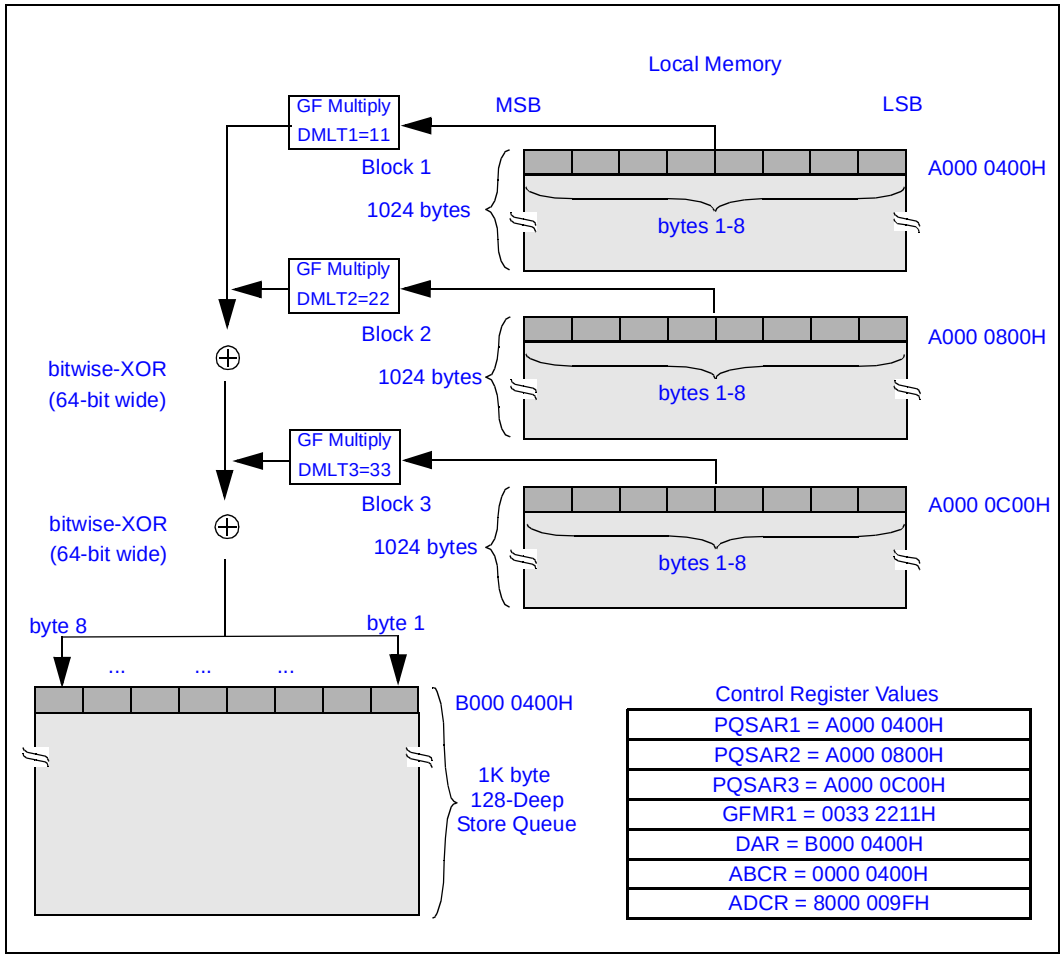
1.5.3 XOR Operation with P+Q RAID-6 Mode

Figure 16 describes the XOR with P+Q RAID-6 mode implementation. In this illustrative example, there are three blocks of source data to have a P+Q RAID-6 mode function performed on them followed by the an XOR function. The intermediate result is kept by the XOR store queue in the AA before being written back to local memory. The source data is located at addresses A000 0400H, A000 0800H, A000 0C00H and A000 1000H respectively.

All data transfers needed for this operation are controlled by chain descriptors located in local memory. The Application DMA as a master on the internal bus initiates a data transfer. The algorithm is implemented such that as data is read from local memory, the boolean unit executes the XOR operation on incoming data.

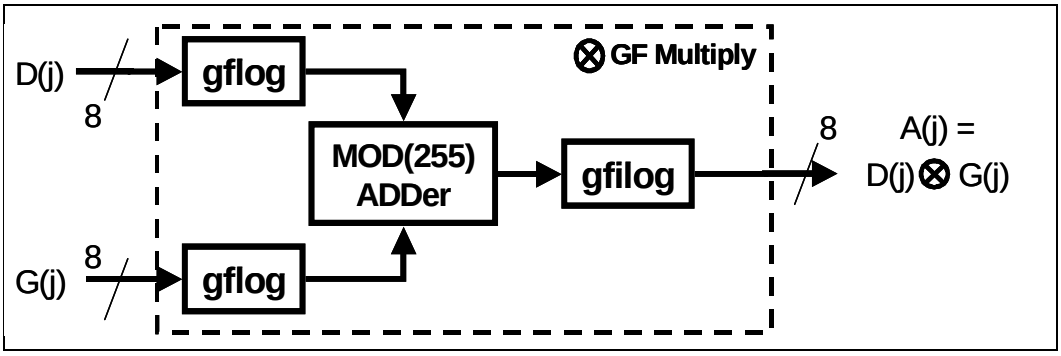
Note: Two descriptors are required for P+Q RAID-6 modes, one for each check value. Each descriptor would be processed as illustrated in Figure 16.

Figure 16. The Bit-wise XOR Algorithm including the P+Q RAID-6 Mode



The GF Multiply function is shown in Figure 17 for 8-bits of data. This function is replicated across each byte lane of the data path, where the G(j) input is the same for each byte lane for a given source.

Figure 17. GF Multiply Function



The blocks for gflog and gfilog are the Galois Field Logarithm and Inverse Logarithm transformations respectively. These transformations for 8-bit words are provided in Figure 18 and Figure 19, with the upper nibble used to index the rows, and the lower nibble used to index the columns. These are based on the primitive polynomial listed in Equation 1.

Figure 18. Galois Field Logarithm Transformation Table

Table of gflog(2^8): gflog(xy)																
gflog(xy)		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	--	00	01	19	02	32	1a	c6	03	df	33	ee	1b	68	c7	4b
	1	04	64	e0	0e	34	8d	ef	81	1c	c1	69	f8	c8	08	4c	71
	2	05	8a	65	2f	e1	24	0f	21	35	93	8e	da	f0	12	82	45
	3	1d	b5	c2	7d	6a	27	f9	b9	c9	9a	09	78	4d	e4	72	a6
	4	06	bf	8b	62	66	dd	30	fd	e2	98	25	b3	10	91	22	88
	5	36	d0	94	ce	8f	96	db	bd	f1	d2	13	5c	83	38	46	40
	6	1e	42	b6	a3	c3	48	7e	6e	6b	3a	28	54	fa	85	ba	3d
	7	ca	5e	9b	9f	0a	15	79	2b	4e	d4	e5	ac	73	f3	a7	57
	8	07	70	c0	f7	8c	80	63	0d	67	4a	de	ed	31	c5	fe	18
	9	e3	a5	99	77	26	b8	b4	7c	11	44	92	d9	23	20	89	2e
	a	37	3f	d1	5b	95	bc	cf	cd	90	87	97	b2	dc	fc	be	61
	b	f2	56	d3	ab	14	2a	5d	9e	84	3c	39	53	47	6d	41	a2
	c	1f	2d	43	d8	b7	7b	a4	76	c4	17	49	ec	7f	0c	6f	f6
	d	6c	a1	3b	52	29	9d	55	aa	fb	60	86	b1	bb	cc	3e	5a
	e	cb	59	5f	b0	9c	a9	a0	51	0b	f5	16	eb	7a	75	2c	d7
	f	4f	ae	d5	e9	e6	e7	ad	e8	74	d6	f4	ea	a8	50	58	af

Figure 19. Galois Field Inverse Logarithm Transformation table

Table of gfilog(2^8): gfilog(xy)																
gfilog(xy)		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	01	02	04	08	10	20	40	80	1d	3a	74	e8	cd	87	13	26
	1	4c	98	2d	5a	b4	75	ea	c9	8f	03	06	0c	18	30	60	c0
	2	9d	27	4e	9c	25	4a	94	35	6a	d4	b5	77	ee	c1	9f	23
	3	46	8c	05	0a	14	28	50	a0	5d	ba	69	d2	b9	6f	de	a1
	4	5f	be	61	c2	99	2f	5e	bc	65	ca	89	0f	1e	3c	78	f0
	5	fd	e7	d3	bb	6b	d6	b1	7f	fe	e1	df	a3	5b	b6	71	e2
	6	d9	af	43	86	11	22	44	88	0d	1a	34	68	d0	bd	67	ce
	7	81	1f	3e	7c	f8	ed	c7	93	3b	76	ec	c5	97	33	66	cc
	8	85	17	2e	5c	b8	6d	da	a9	4f	9e	21	42	84	15	2a	54
	9	a8	4d	9a	29	52	a4	55	aa	49	92	39	72	e4	d5	b7	73
	a	e6	d1	bf	63	c6	91	3f	7e	fc	e5	d7	b3	7b	f6	f1	ff
	b	e3	db	ab	4b	96	31	62	c4	95	37	6e	dc	a5	57	ae	41
	c	82	19	32	64	c8	8d	07	0e	1c	38	70	e0	dd	a7	53	a6
	d	51	a2	59	b2	79	f2	f9	ef	c3	9b	2b	56	ac	45	8a	09
	e	12	24	48	90	3d	7a	f4	f5	f7	f3	fb	eb	cb	8b	0b	16
	f	2c	58	b0	7d	fa	e9	cf	83	1b	36	6c	d8	ad	47	8e	--

Equation 1. Galois Field Primitive Polynomial (0x11D)

$$X^8 + X^4 + X^3 + X^2 + 1$$

1.5.4 Dual-XOR Operation

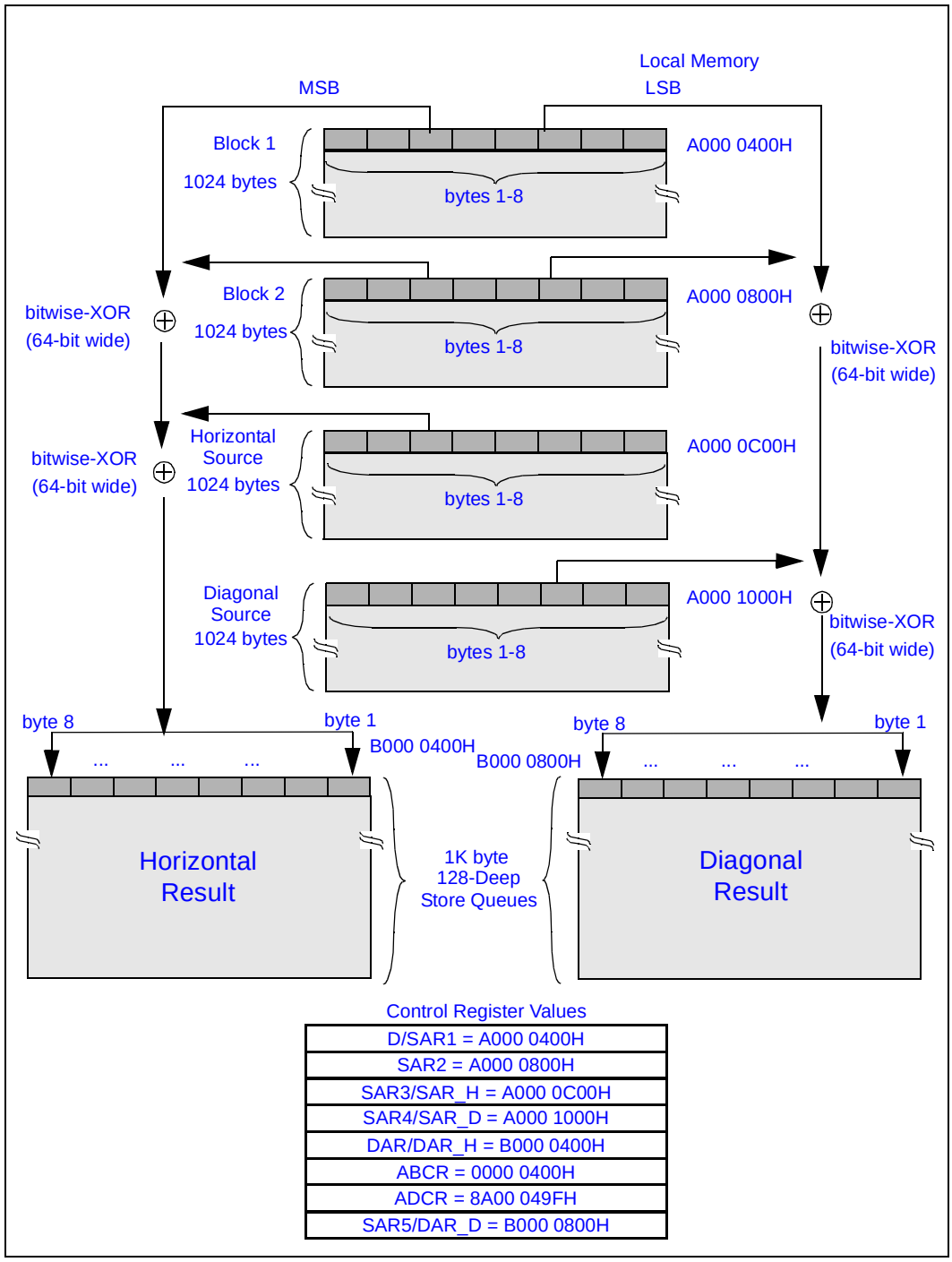
The Dual-XOR operation can be used in RAID-6 applications when a single strip write requires updating of two parity blocks. For this use, the two parity blocks are based on the same data being updated, and this operation performs the calculation of both updated parity blocks. In the illustrative Dual-XOR example in Figure 20, there are four blocks of source data to be XOR-ed. Two sources are used for both updated parity results, and there is one unique source for each parity result. The intermediate results are kept in store queues in the AA before being written back to local memory.

The two common source data blocks are located at addresses A000 0400H, A000 0800H. The Horizontal data source is located at address A000 0C00H and the Diagonal data source is located at address A000 1000H. The Horizontal and Diagonal destination addresses are located at addresses B000 0400H and B000 0800H respectively.

All data transfers needed for this operation are controlled by chain descriptors located in local memory. The Application Accelerator as a master on the internal bus initiates data transfer. The algorithm is implemented such that as data is read from local memory, the boolean unit executes the XOR operation on incoming data.

Note: Dual_XOR operation is intended for use with single strip write I/Os to RAID-6 arrays. To generate two check values for a full stripe of data in a RAID-6 array, XOR operations defined by separate descriptors for each check value must be used. See Section 1.5.2, “XOR Operation” on page 40 for details.

Figure 20. The Bit-wise Dual-XOR Algorithm



The XOR algorithm and methodology followed once a chain descriptor has been configured is similar to that described for the basic XOR transfer. The steps followed by the AA when processing a *Dual-XOR-transfer* are detailed below:

1. The Application Accelerator as a master on the bus initiates data transfer from the address pointed at by the First Source Address Register (SAR1). The total number of bytes to *XOR-transfer* is specified by the Byte Count (BC) field in the chain descriptor.

Note: The Direct Fill command must be selected for SAR1, designating it as the first block of data in the current Dual XOR operation, resulting in data being transferred directly to the horizontal and diagonal store queues. The number of bytes transferred to the store queues is 1KByte/512Bytes (based on bit 2 of the Accelerator Control Register).

Note: If the Byte Count Register contains a value greater than the buffer size, the AA completes the *XOR-transfer* operation on the first buffer (store queue size) of data obtained from each Source Register (SAR1, SAR2, SAR_H, SAR_D), then proceeds with the next buffer of data. This process is repeated until the BCR contains a zero value.

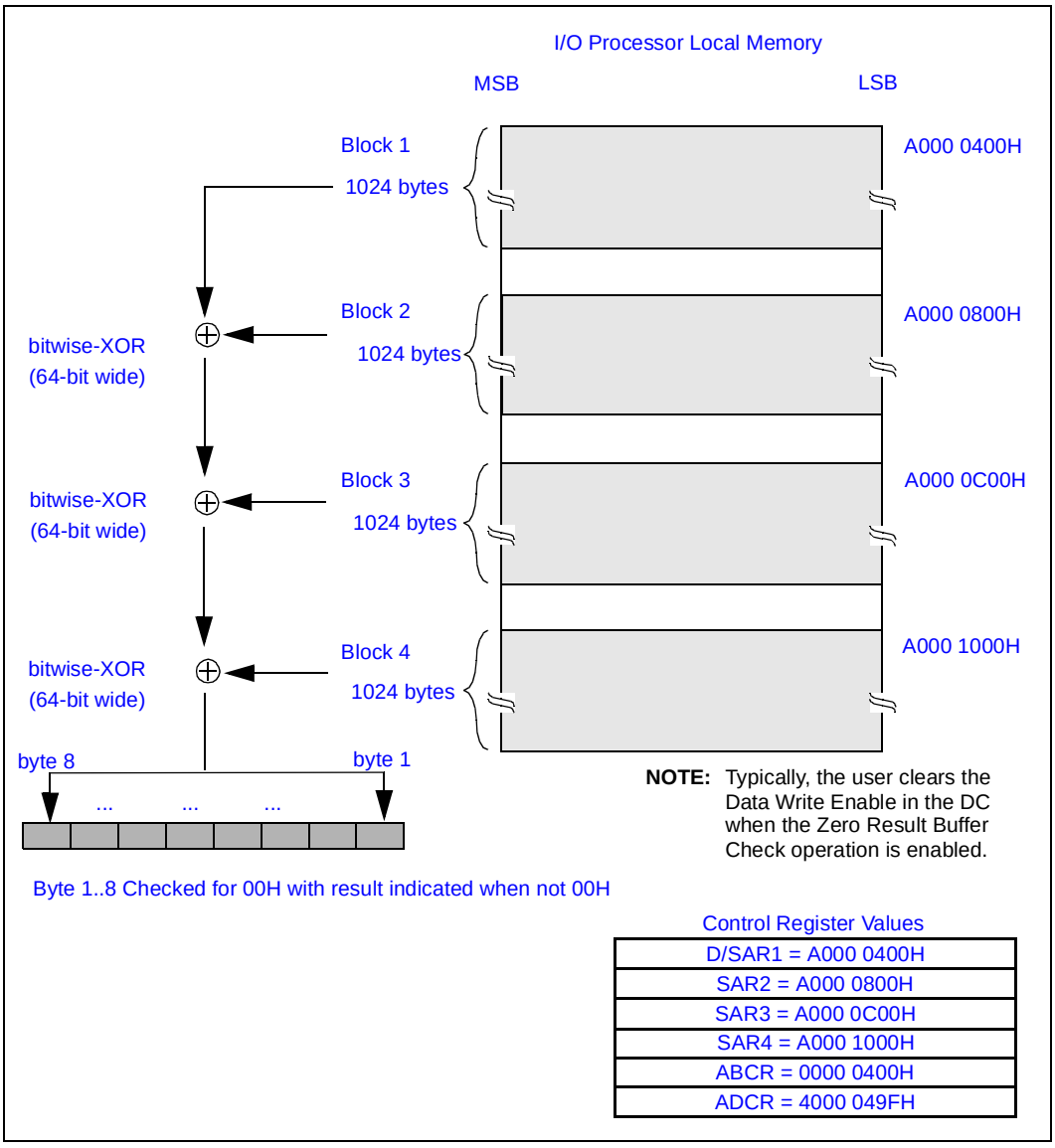
2. The Application Accelerator transfers the first eight bytes of data from the address pointed at by the Second Source Address Register (SAR2).
    a. The XOR command must be selected for SAR2, so that the boolean unit performs the XOR operation on the data currently existing in the two store queues (SAR1) with the data being transferred from memory (SAR2).
3. The boolean unit performs the bit-wise XOR algorithm on the input operands. The input operands are the first eight bytes of data read from SAR1 (bytes 1-8) which are stored in the store queues and the first eight bytes of data just read from SAR2 (bytes 1-8).
4. The XOR-ed result is transferred to both store queues and stored in the first eight bytes (bytes 1-8) overwriting previously stored data.
5. The Application Accelerator transfers the next eight bytes of data (bytes 9-16) from address pointed at by the Second Source Address Register (SAR2).
6. The boolean unit performs the bit-wise XOR algorithm on the input operands. The input operands are the next eight bytes of data read from SAR1 (bytes 9-16 stored in the queue) and the eight bytes of data read from SAR2 in Step-5.
7. Step-5 and Step-6 (Data transfer and XOR) are repeated until all data pointed at by SAR1 is XOR-ed with the corresponding data pointed at by SAR2. The two store queues now both contain a buffer full of XOR-ed data, the source addresses for which were specified in SAR1 and SAR2.
8. Steps 2-7 are repeated with the Horizontal Source address used for the next source data with the following exceptions
    a. Only the Horizontal Store Queue is overwritten with the new XOR-ed result.
    b. Upon completion, the Horizontal Store Queue holds the bit-wise XOR of Source 1 (SAR1), Source 2 (SAR2) and the Horizontal Source (SAR_H).
    c. The Diagonal Store Queue remains unchanged.
9. Once Step 8 is completed, the XOR operation is complete for the first full buffer of the Horizontal XOR operation. The Destination Write Enable Bit in the Accelerator Descriptor Control Register (ADCR) must be set. The data in the horizontal store queue is written to local memory at the address pointed to by the Horizontal Destination Address Register (DAR_H).

10. Steps 2-7 are then repeated with the Diagonal Source address used for the next source data with the following exceptions
    a. Only the Diagonal Store Queue is overwritten with the new XOR-ed result.
    b. Upon completion, the Diagonal Store Queue holds the bit-wise XOR of Source 1 (SAR1), Source 2 (SAR2) and the Diagonal Source (SAR_D).
    c. The Horizontal Store Queue remains unchanged.
11. Once Step 10 is completed, the XOR operation is complete for the first full buffer of the Diagonal XOR operation. The Destination Write Enable Bit in the Accelerator Descriptor Control Register (ADCR) must be set. The data in the diagonal store queue is written to local memory at the address pointed to by the Diagonal Destination Address Register (DAR_D).
12. Steps 1-11 are repeated until all the bytes of data have undergone the *Dual-XOR-transfer* operation.

1.5.5 Zero Result Buffer Check

The AA can be used to verify parity across memory blocks specified by the SARx registers. XOR operation descriptors are used to specify the memory blocks on which the AA performs the Zero Result Buffer Check. Figure 21 illustrates a Zero Result Buffer Check performed by the AA. After processing all source data, the AA updates the Transfer Complete and Result Buffer Not Zero bit of the eighth word of the descriptor (ADCR) pointed to by the ADAR in local memory.

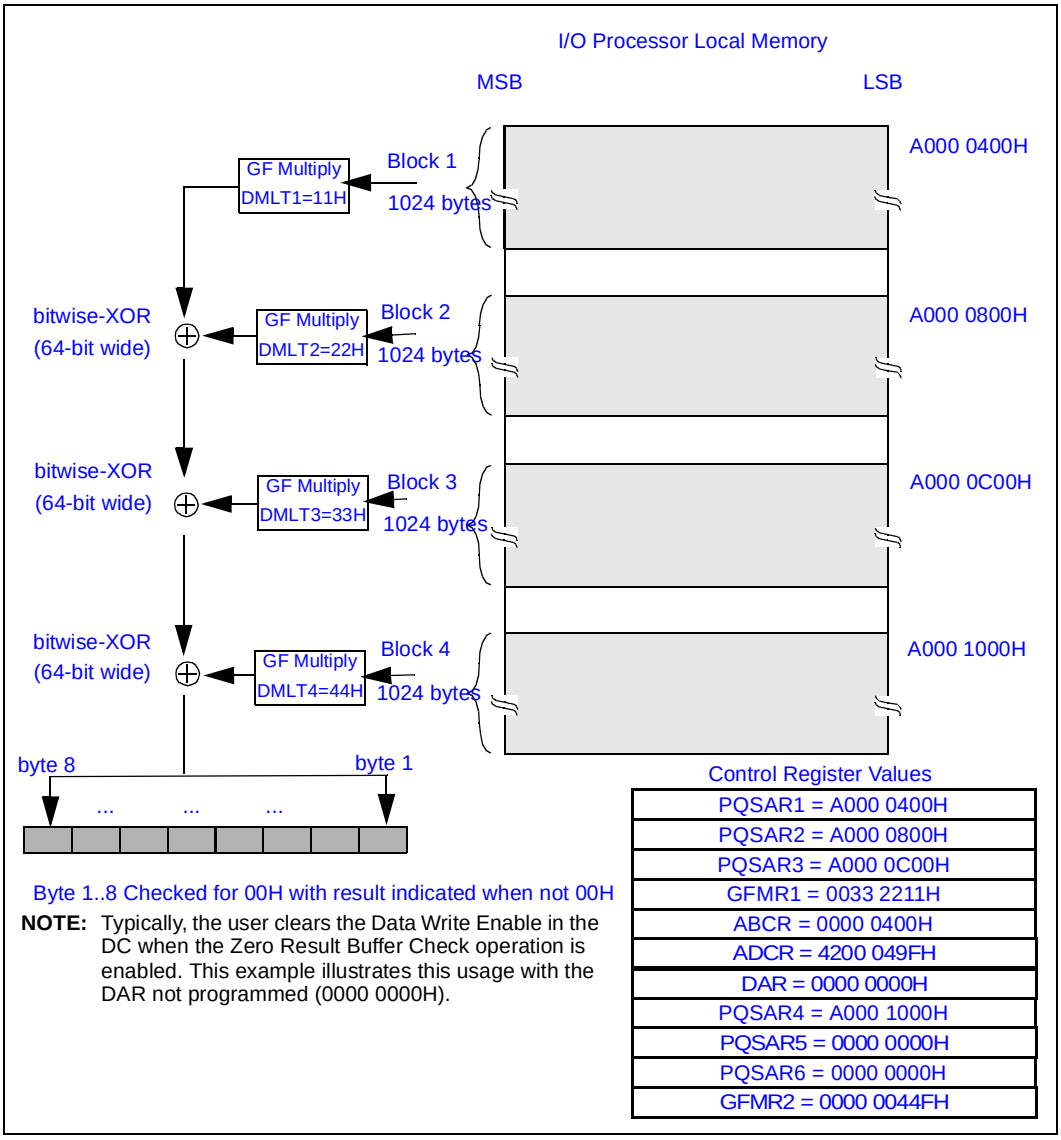
Figure 21. An Example of Zero Result Buffer Check



1.5.6 Zero Result Buffer Check with P+Q RAID-6

The AA can be used to verify check values for P+Q RAID-6 implementations. P+Q XOR operation descriptors are used to specify the memory blocks on which the AA performs the Zero Result Buffer Check. Figure 22 illustrates a P+Q RAID-6 Zero Result Buffer Check performed by the AA. After processing all source data, the AA updates the Transfer Complete and Result Buffer Not Zero bit of the eighth word of the descriptor (ADCR) pointed to by the ADAR in local memory. If the Destination Write Enable bit is set, the result buffer is also stored to the memory location pointed to by the DAR.

Figure 22. An example of Zero Result Buffer Check with P+Q RAID-6

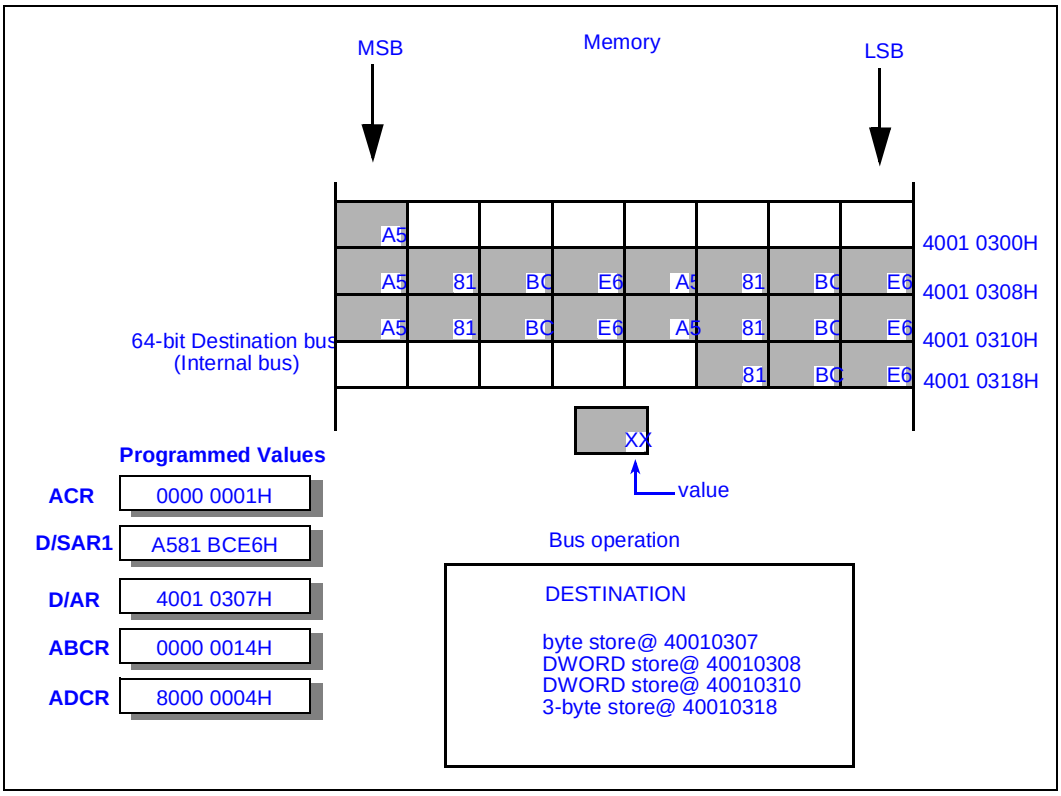


NOTE: Typically, the user clears the Data Write Enable in the DC when the Zero Result Buffer Check operation is enabled. This example illustrates this usage with the DAR not programmed (0000 0000H).

1.5.7 Memory Block Fill Operation

The AA can be used to write a constant value to a memory block in the 80331 local memory. As with XOR operations, descriptors are used to specify the memory blocks to which the AA writes the data contained in the Data / Source Address Register1. All memory block fill operations are controlled by chain descriptors located in the Intel XScale® core local memory. Figure 23 illustrates a Block Fill Operation to an arbitrary destination address.

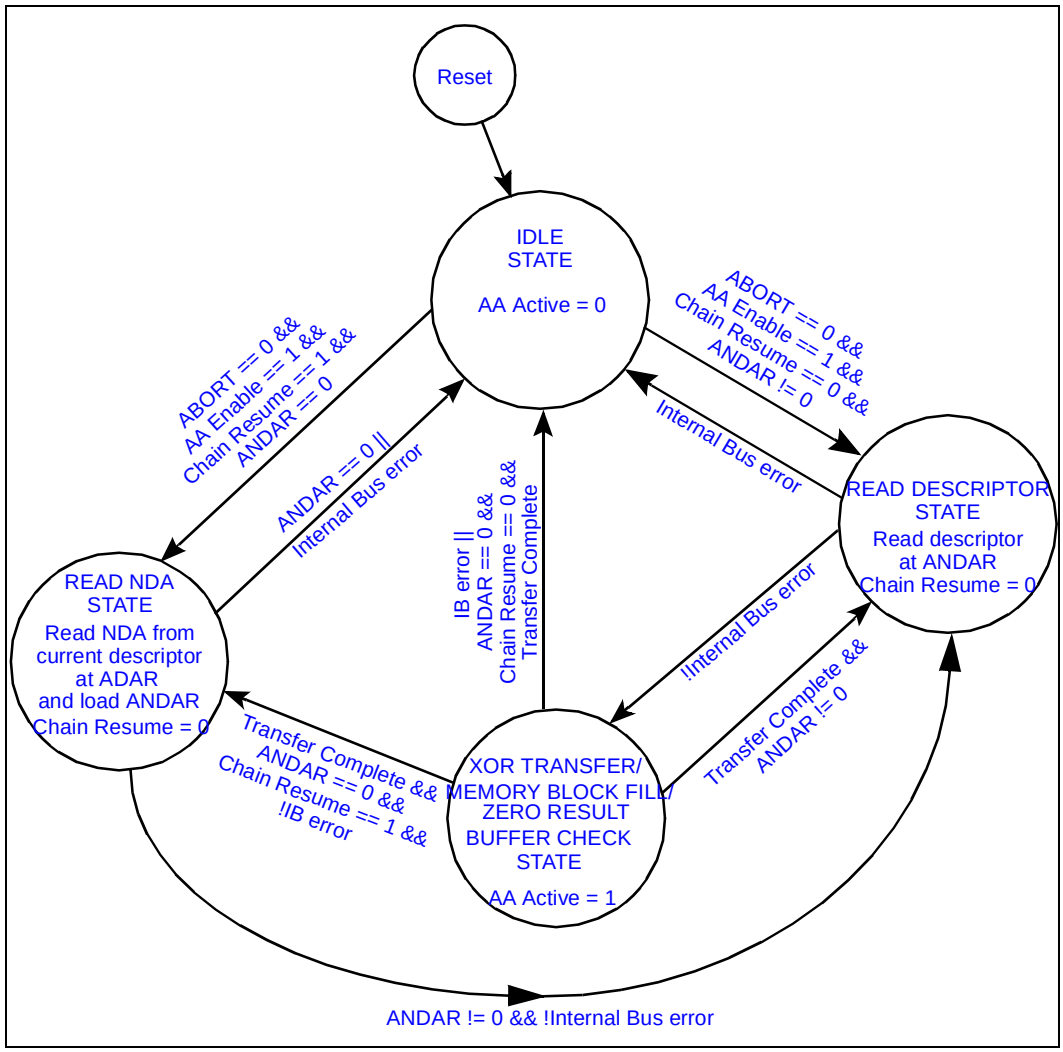
Figure 23. Example of a Memory Block Fill Operation



1.6 Programming Model State Diagram

The AA programming model diagram is shown in Figure 24. Error condition states are not shown.

Figure 24. Application Accelerator Programming Model State Diagram



1.7 Application Accelerator Priority

The internal bus arbitration logic determines which internal bus master has access to the 80331 internal bus. The Application Accelerator has an independent Bus Request/Grant signal pair to the internal bus arbitration logic. The "Arbitration Unit" in the *Intel® 80331 I/O Processor Developer's Manual*, describes in detail the priority scheme between all of the bus masters on the internal bus.

In addition, the internal bus arbitration unit has a Multi-Transaction timer (Section 13.4.3, "Multi-Transaction Timer Register 2 - MTTR2" on page 656, in the *Intel® 80331 I/O Processor Developer's Manual*) that affects the throughput of the AA. The default value for MTT2 of 152 clocks was chosen to ensure that once an internal bus agent (in this case the AA) is granted the internal bus that it is guaranteed an opportunity to burst data into DDR SDRAM memory. However, when the bus is busy the AA loses grant before the burst is completed. This means that the AA is able to complete only one burst for each arbitration cycle.

Alternatively, the user may wish to increase the value of MTT2 to guarantee that two or more bursts are able to complete within an arbitration cycle.

For example, assuming 1 Kbyte bursts and a 64-bit memory subsystem, an MTT2 setting of 192 clocks would be sufficient to support two 1 Kbyte bursts for an AA single arbitration cycle.

Warning: Increasing the MTT2 value may also increase the latency to peripheral memory mapped registers or PCI addresses for the Intel XScale® core on the average. Before changing the MTT2 value, it's imperative that the overall impact to the performance of the application is considered.

1.8 Packing and Unpacking

The Application Accelerator contains a hardware data packing and unpacking unit to support data transfers between unaligned source and destination addresses. Source and destination addresses can either be unaligned or aligned on natural boundaries. The packing unit optimizes data transfers to and from 32 and 64-bit memory. It reformats data words for the correct bus data width. When the read data needs to be packed or unpacked, the data is held internally and does not need to be re-read.

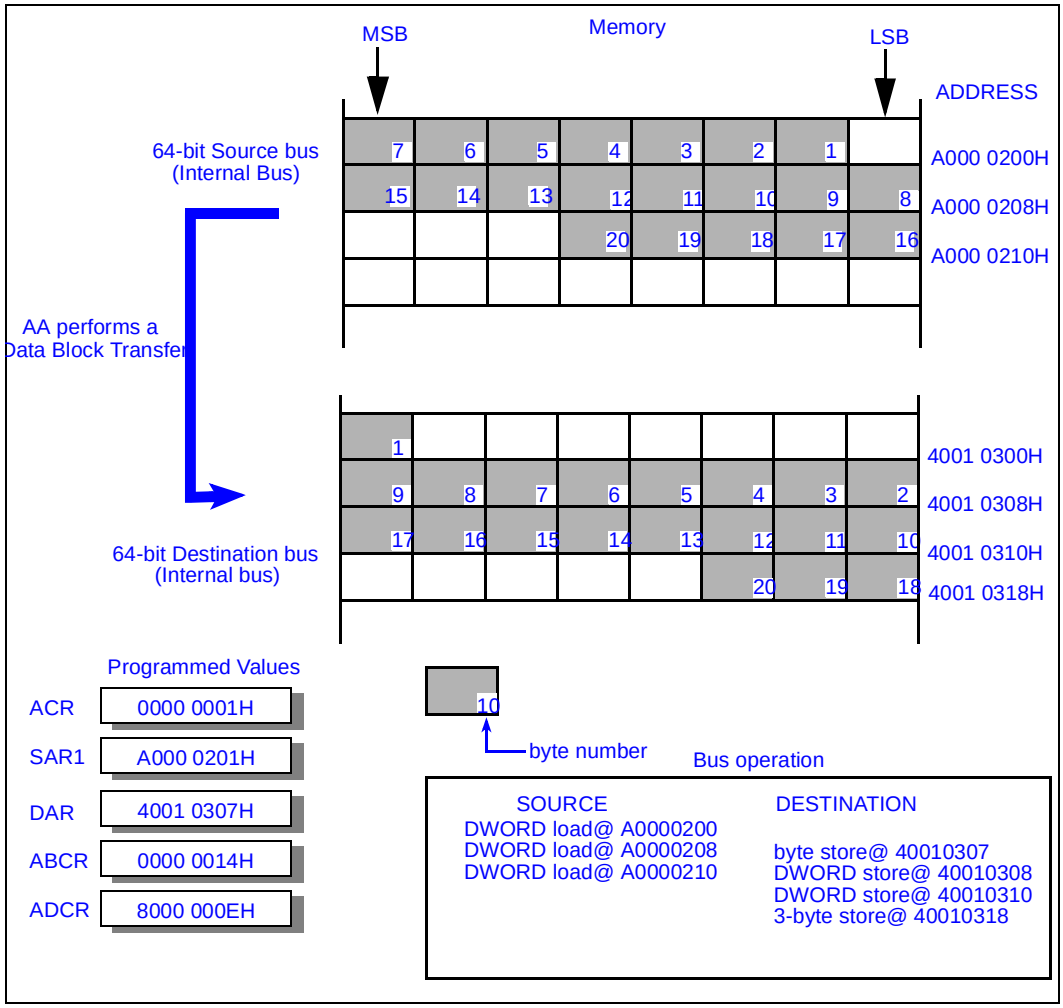
Aligned data transfers fall on natural boundaries. For example; DWORDs are aligned on 8-byte boundaries and words are aligned on 4-byte boundaries. Data transfers take place in two instances:

- The source and destination addresses are both aligned.
- All or some source addresses are unaligned and the destination address is aligned or unaligned.

1.8.1 64-bit Unaligned Data Transfers

Figure 26 illustrates a data transfer between unaligned 64-bit, source and destination addresses.

Figure 25. Optimization of an Unaligned Data Transfer



1.9 Programming the Application Accelerator

The operations for Application Accelerator software falls into the following categories:

- AA initialization
- Suspend AA
- Appending Descriptors
- Resume AA Operation

An example for each category is shown in the following sections as pseudo code flow.

The AA control register provides independent control each time the AA is configured. This provides the greatest flexibility to the applications programmer.

The most efficient method for operating the AA is to use the *Chain Resume* capability described in Section 1.3.4, “Descriptor Chaining”. To use of the *Chain Resume* capability for appending descriptors to chains in normal operation, an initial AA descriptor must be executed. This initialization step is described in Section 1.9.1, “Application Accelerator Initialization”. The example AA operations provided later in this section use the *Chain Resume* capability as follows:

- Store Descriptor in Local Memory
- Append Descriptor to Chain
- Resume AA Operation

1.9.1 Application Accelerator Initialization

The AA is designed to have independent control of the interrupts, enables, and control. The initialization consists of virtually no overhead as shown in Figure 27.

Figure 26. Pseudo Code: Application Accelerator Initialization

```
ACR = 0x0000 0000 ; Disable the application accelerator
Call setup_accelerator
```

The following example illustrates how AA initialization S/W prepares the AA for descriptor *Chain Resume* operation. Initializing the AA for chaining requires an initial descriptor be created and executed. This descriptor is then the start of the chain, and future descriptors are appended to this descriptor to create the chain. This descriptor is a NULL descriptor, requiring no source or destination data buffers be allocated. To start an operation, software simply sets the AA Enable bit in the “Accelerator Control Register - ACR” (see in the *Intel® 80331 I/O Processor Developer’s Manual*) as shown in Figure 27.

Figure 27. Pseudo Code: Application Accelerator Chain Resume Initialization

```
; Set up descriptor in Intel XScale® core local memory at address d
d.nda  = 0         /* No chaining */
d.D/SAR1 = 0x0000 0000/* Source address of Data Block 1     */
d.SAR2 = 0x0000 0000/* Source address of Data Block 2     */
d.SAR3 = 0x0000 0000/* Source address of Data Block 3     */
d.SAR4 = 0x0000 0000/* Source address of Data Block 4     */
d.DAR  = 0x0000 0000/* Destination address of XOR-ed data */
d.ABCR = 0x0       /* Byte Count of zero */
d.ADCR = 0x000 0000/* Null Descriptor, No Interrupt*/

; Start operation
ANDAR = &d ; Setup descriptor address
ACR = 0x0000 0001 ; Set AA Enable bit
```

1.9.2 Suspending and Resuming the Application Accelerator

The Application Accelerator unit provides the ability to suspend the current state without losing status information. The AA resumes without requiring application software to restore the previous configuration. The example shown in Figure 28 describes pseudo-code for suspending the ongoing operation and then restarting.

Figure 28. Pseudo Code: Suspend Application Accelerator

```
;Suspend Application Accelerator
ACR = 0x0000 0000  ; Suspend ongoing AA transfer

;Restart Application Accelerator
ACR = 0x0000 0001  ; Restart AA operation
```

1.9.3 Appending Descriptor for XOR Operations

The example shown in Figure 29 describes the pseudo code for initiating an XOR operation with the AA. The examples illustrates appending the XOR operation to an existing chain, and taking advantage of the Chain Resume capability as described in Section 1.9.2, “Suspending and Resuming the Application Accelerator”.

Figure 29. Pseudo Code: XOR Transfer Operation

```
; Set up descriptor in Intel XScale® core local memory at address d
d.nda  = 0         /* No chaining */
d.D/SAR1 = 0xA000 0400/* Source address of Data Block 1     */
d.SAR2 = 0xA000 0800/* Source address of Data Block 2     */
d.SAR3 = 0xA000 0C00/* Source address of Data Block 3     */
d.SAR4 = 0xA000 1000/* Source address of Data Block 4     */
d.DAR  = 0xB000 0100/* Destination address of XOR-ed data */
d.ABCR = 1024      /* Byte Count of 1024 */
d.ADCR = 0x8000 049F/* Direct fill data from Block 1      */
                   /* XOR with data from Block 2,Block 3 and
                      Block 4                            */
                   /* Store the result and interrupt processor */

; Append descriptor to end of last chain at address c
c.nda = d

; Resume AA operation
ACR = 0x00000003   ; Set AA Enable and Resume bits
```

1.9.4 Appending Descriptor for Dual XOR Operations

The example shown in Figure 30 describes the pseudo code for initiating a Dual XOR operation with the AA. The examples illustrates appending the Dual XOR operation to an existing chain, and taking advantage of the Chain Resume capability as described in Section 1.9.2, "Suspending and Resuming the Application Accelerator" on page 57.

Figure 30. Pseudo Code: Dual XOR Transfer Operation

```
; Set up descriptor in Intel XScale® core local memory at address d
d.nda  = 0         /* No chaining */
d.D/SAR1 = 0xA000 0400/* Source address of Data Block 1     */
d.SAR2 = 0xA000 0800/* Source address of Data Block 2     */
d.SAR3 = 0xA000 0C00/* Source address of Horizontal Data Block */
d.SAR4 = 0xA000 1000/* Source address of Diagonal Data Block */
d.DAR_H = 0xB000 0100/* Destination address of Horizontal XOR-ed data */
d.ABCR = 1024      /* Byte Count of 1024 */
d.ADCR = 0x8800 049F/* Dual XOR Operation */
                   /* Required: Direct fill from Block 1 */
                   /* XOR enabled for Blocks 2, 3 and 4 */
                   /* Store the results */
                   /* Optional: interrupt processor */
d.DAR_D = 0xB000 4100/* Destination address of Diagonal XOR-ed data */

; Append descriptor to end of last chain at address c
c.nda = d

; Resume AA operation
ACR = 0x00000003   ; Set AA Enable and Resume bits
```

1.9.5 Appending Descriptor for Memory Block Fill Operations

The example shown in Figure 31 describes the pseudo code for initiating a Memory Block Fill operation with the AA.

Figure 31. Pseudo Code: Memory Block Fill Operation

```
; Set up descriptor in Intel XScale® core local memory at address d
d.nda  = 0         /* No chaining */
d.D/SAR1 = 0xA000 0400/* Immediate data used for block write*/
d.DAR  = 0xB000 0100/* Address of the memory block to be written*/
d.ABCR = 1024      /* Byte Count of 1024 */
d.ADCR = 0x8000 0005/* Memory Write Block using data in D/SAR1*/
                   /* Store the result and interrupt processor */

; Append descriptor to end of last chain at address c
c.nda = d

; Resume AA operation
ACR = 0x00000003   ; Set AA Enable and Resume bits
```

1.9.6 Appending Descriptor for Zero Result Buffer Check

The example shown in Figure 32 describes the pseudo code for initiating an XOR operation with the AA.

Figure 32. Pseudo Code: Zero Result Buffer Check Operation[a]

```
; Set up descriptor in Intel XScale® core local memory at address d
d.nda  = 0         /* No chaining */
d.D/SAR1 = 0xA000 0400/* Source address of Data Block 1     */
d.SAR2 = 0xA000 0800/* Source address of Data Block 2     */
d.SAR3 = 0xA000 0C00/* Source address of Data Block 3     */
d.SAR4 = 0xA000 1000/* Source address of Data Block 4     */
d.ABCR = 1024      /* Byte Count of 1024 */
d.ADCR = 0x4000 049F/* Direct fill data from Block 1      */
                   /* XOR with data from Block 2,Block 3 and
                      Block 4                            */
                   /* Check Result, Write Status (ADCR) and interrupt processor */

; Append descriptor to end of last chain at address c
c.nda = d

; Resume AA operation
ACR = 0x00000003   ; Set AA Enable and Resume bits
```

a. Notice that ADCR.dwe is cleared and that the DAR is not programmed. The reason is that for Zero Result Buffer Check operations, there is no need to write out a destination parity stripe.

1.10 Interrupts

The Application Accelerator can generate an interrupt to the Intel XScale® core. The *Interrupt Enable* bit in the Accelerator Descriptor Control Register (ADCR.ie) determines whether the AA generates an interrupt upon successful, error-free completion. Error conditions described in Section 1.11 also generate an interrupt. The AA has one interrupt output connected to the PCI and Peripheral Interrupt Controller described in Chapter 17, "Interrupt Controller Unit and IOAPIC".

Once the AA is enabled, the AA loads the chain descriptor fields into the respective registers. A special case exists when data write enable is clear, then an interrupt is generated (when enabled) after the descriptor is fetched and processed as defined by the block control fields in the ADCR. Table 5 summarizes the status flags and conditions when interrupts are generated in the Accelerator Status Register (ASR).

Table 5. AA Interrupts

Interrupt Condition	Accelerator Status Register (ASR) Flags				Interrupt Generated?	
	Active	End of Transfer	End of Chain	IB Master Abort	Interrupt Enabled	Interrupt Disabled
(Data Write Enable == 0 || byte count == 0) && (ANDAR != NULL || Resume == 1) (End of Transfer)	1	1	0	0	Y	N
(Data Write Enable == 0 || byte count == 0) && ANDAR == NULL && Resume == 0 (End of Chain)	0	0	1	0	Y	N
IB Master Abort	0	0	0	1	Y	Y
IB Target Abort	0	0	0	0	N	N

Note: End-of-Transfer and End-of-Chain flags is set only when Interrupt Enable is set. When Interrupt Enable is clear, then the above flags are always set to 0. End-of-Transfer Interrupt and End of Chain Interrupt can only be reported in the ASR when the descriptor fetch and processing completed without any reportable errors. However, multiple error conditions may occur and be reported together. Also, because the AA does not stop after reporting the End-of-Transfer interrupt, an IB master-abort error may occur before the End-of-Transfer interrupt is serviced and cleared.

1.11 Error Conditions

Master Aborts that occur during a transfer are recorded by the Application Accelerator.

When an error occurs, the actions taken are detailed below:

- The AA ceases the ongoing transfer for the current chain descriptor and clear the *Application Accelerator Active* flag in the ASR.
- The AA does not read any new chain descriptors.
- The AA sets the error flag in the Accelerator Status Register. For example; when an IB master-abort occurred during a transfer, the channel sets bit 5 in the ASR.
- The AA signals an interrupt to the Intel XScale® core.
- The Application Accelerator does not restart the transfer after an error condition. It is the responsibility of the application software to reconfigure the AA to complete any remaining transfers.

Note: Target-aborts during AAU reads result from multi-bit ECC errors that are recorded by the MCU. Refer to Chapter 8, “Memory Controller” for details on error handling in this instance. For correct operation of the AAU, user software has to disable the AAU before clearing the error condition. Furthermore, the AAU needs to be re-enabled by writing a 1 to the AA Enable bit before initiating a new operation.

1.12 Power-up/Default Status

Upon power-up, an external hardware reset, the Application Accelerator Registers are initialized to their default values.

1.13 Register Definitions

The Application Accelerator Unit contains forty two memory-mapped registers for controlling its operation. There is read/write access only to the Accelerator Control Register, Accelerator Status Register, the Accelerator Next Descriptor Address Register, and the three Extended Descriptor Control Registers. All other registers are read-only and are loaded with new values from the chain descriptor whenever the AA reads a chain descriptor from memory.

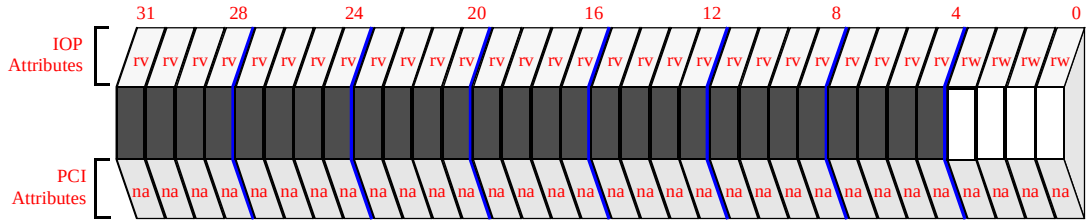
Table 6. Application Accelerator Unit Registers

Section, Register Name - Acronym (page)
Section 1.13.1, “Accelerator Control Register - ACR” on page 63
Section 1.13.2, “Accelerator Status Register - ASR” on page 64
Section 1.13.3, “Accelerator Descriptor Address Register - ADAR” on page 65
Section 1.13.4, “Accelerator Next Descriptor Address Register - ANDAR” on page 66
Section 1.13.5, “Data / Source Address Register1 - D/SAR1/PQSAR1” on page 673
Section 1.13.6, “Source Address Registers 2..32 - SAR2..32” on page 68
Section 1.13.7, “P+Q RAID-6 Source Address Registers 2..16 - PQSAR2..16” on page 706
Section 1.13.8, “P+Q RAID-6 Galois Field Multiplier Registers 1..5 - GFMR1..5” on page 71
Section 1.13.9, “Destination Address Register - DAR” on page 73
Section 1.13.10, “Accelerator Byte Count Register - ABCR” on page 74
Section 1.13.11, “Accelerator Descriptor Control Register - ADCR” on page 75
Section 1.13.12, “Extended Descriptor Control Register 0 - EDCR0” on page 79
Section 1.13.13, “Extended Descriptor Control Register 1 - EDCR1” on page 81
Section 1.13.14, “Extended Descriptor Control Register 2 - EDCR2” on page 83

1.13.1 Accelerator Control Register - ACR

The Accelerator Control Register (ACR) specifies parameters that dictate the overall operating environment. The ACR should be initialized prior to all other AA registers following a system reset. Table 7 shows the register format. This register can be read or written while the AA is active.

Table 7. Accelerator Control Register - ACR



Internal bus address
FFFF E800H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

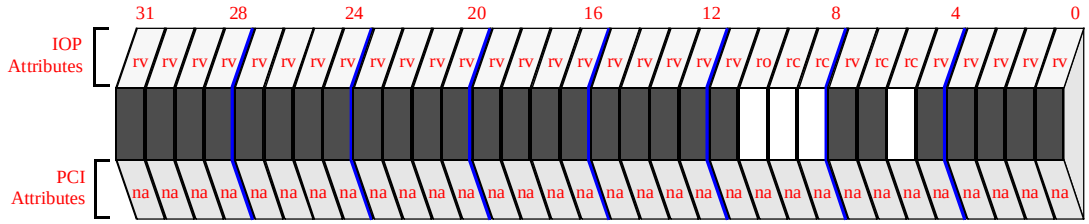
Bit	Default	Description
31:04	0	Reserved
03	0_2	P+Q RAID-6 Enable - when set, causes the AA to process descriptors defined for P+Q RAID-6, which includes a Galois Field Multiply byte value for each source. NOTE: This bit can only be changed when the AA is idle (ASR.10=0). Changing the state of this bit while the AA is active (ASR.10=1) will result in unpredictable results. 0 = Disabled - All source data is passed directly from the internal bus to the AA. 1 = Enabled - All source data is operated on by the GF Multiply function when being fetched from memory, before being passed to the AA.
02	0_2	512 Byte Buffer Enable - when set, causes the AA to use only 512 bytes of 1 KB data buffer while processing all descriptors.
01	0_2	Chain Resume - when set, causes the AA to resume chaining by re-reading the current descriptor located at the address in the Accelerator Descriptor Address Register when the AA is idle (AA Active bit in the ASR is clear) or when the AA completes a transfer. This bit is cleared by hardware when either: • The AA completes a transfer and the Accelerator Next Descriptor Address Register is non-zero. In this case, the AA proceeds to the next descriptor in the chain. • The AA re-reads the chain descriptor located at the address in the Accelerator Descriptor Address Register and loads the Next Descriptor Address of that descriptor into the Accelerator Next Descriptor Address Register
00	0_2	AA Enable - When set, the AA enables transfers. When clear, the AA disables any transfer. Clearing this bit when the AA is active suspends the current transfer at the earliest opportunity by halting all internal bus transactions. The AA does not initiate any new transfers when this bit is cleared. Data held in queues remains valid. Setting the bit after the AA is suspended causes the AA to resume the previously ongoing transfer.

1.13.2 Accelerator Status Register - ASR

The Accelerator Status Register (ASR) contains status flags that indicate status. This register is typically read by software to examine the source of an interrupt. See Section 1.11 for a description of the error conditions that are reported in the ASR. See Section 1.10 for a description of interrupts caused by the Application Accelerator.

When an AA error occurs, application software should check the status of Accelerator Active flag before processing the interrupt.

Table 8. Accelerator Status Register - ASR



Internal bus address
FFFF E804H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

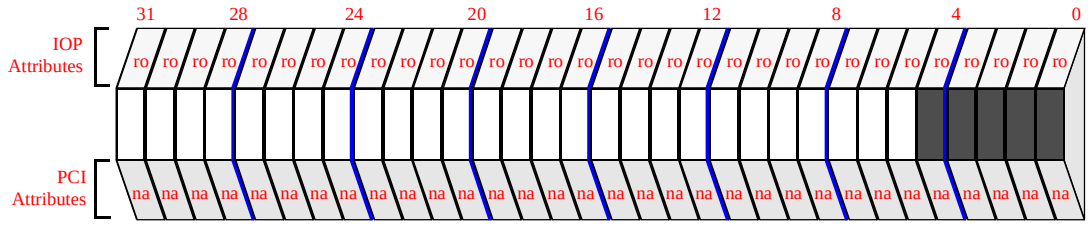
Bit	Default	Description
31:11	000000H	Reserved
10	0_2	Accelerator Active Flag - indicates the AA is either active (in use) or idle (available). When set, indicates the AA is in use and actively performing an operation. When clear, indicates the channel is idle and available to be configured for a new operation. The AA clears the Accelerator Active flag when the previously configured transfer completes as a result of: • byte count reached zero and last chain descriptor is encountered (NULL value detected for Next Descriptor Address in chain descriptor) • Internal Bus Errors • Last chain descriptor is processed (NULL value detected for Next Descriptor Address in chain descriptor) and write enable is zero. The Accelerator Active flag is set once a Chain Descriptor is read from memory.
09	0_2	End of Transfer Interrupt Flag - set when the AA has signalled an interrupt to the Intel XScale® core after processing a descriptor but it is not the last descriptor in a chain.
08	0_2	End of Chain Interrupt Flag - set when the channel has signalled an interrupt to the Intel XScale® core after processing a descriptor that is the last in a chain.
07:06	0_2	Reserved
05	0_2	This bit is set when a Master-abort occurs during a transaction when the AAU is the master on the internal bus.
04:00	0_2	Reserved

1.13.3 Accelerator Descriptor Address Register - ADAR

The Accelerator Descriptor Address Register (ADAR) contains the address of the current chain descriptor in local memory. This read-only register is loaded when a new chain descriptor is read. Table 9 depicts the ADAR. Depending on the number of sources, the chain descriptors are required to be aligned on different address boundaries. These include four sources on an eight word address boundary, eight sources on a 16 word address boundary, 16 sources on a 32 word address boundary, and 32 sources on a 64 word address boundary.

Note: In the above paragraph, the term “word” refers to a DWORD.

Table 9. Accelerator Descriptor Address Register - ADAR



Internal bus address
FFFF E808H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:05	000 0000H	Current Descriptor Address - local memory address of the current chain descriptor read by the Application Accelerator.
04:00	$0\ 0000_2$	Reserved

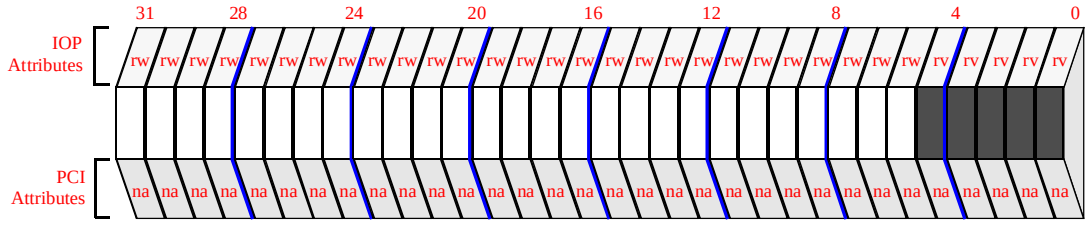
1.13.4 Accelerator Next Descriptor Address Register - ANDAR

The Accelerator Next Descriptor Address Register (ANDAR) contains the address of the next chain descriptor in local memory. When starting a transfer, this register contains the address of the first chain descriptor. Table 10 depicts the Accelerator Next Descriptor Address Register.

All chain descriptors are aligned on an eight DWORD boundary. The AA may set bits 04:00 to zero when loading this register.

Note: The *Accelerator Enable* bit in the ACR and the *Accelerator Active* bit in the ASR must both be clear prior to writing the ANDAR. Writing a value to this register while the AA is active may result in undefined behavior.

Table 10. Accelerator Next Descriptor Address Register - ANDAR



Bit	Default	Description
31:05	000 0000H	Next Descriptor Address - local memory address of the next chain descriptor to be read by the Application Accelerator.
04:00	$0\ 0000_2$	Reserved

1.13.5 Data / Source Address Register1 - D/SAR1/PQSAR1

The Data / Source Address Register (D/SAR1/PQSAR1) contains a 32-bit, local memory address or immediate data to be written in case of Memory Block Fill operations. The ADCR register (Table 17) controls the operation performed on data block referenced by this register. The local memory address space is a 32-bit, byte addressable address space.

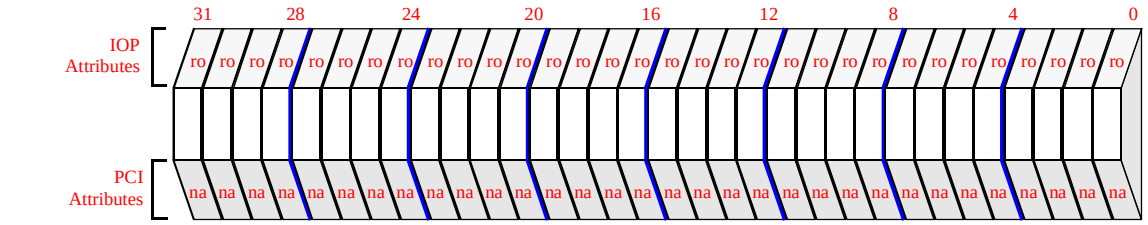
Reading the D/SAR1/PQSAR1 register once the AA has started a chain descriptor returns the current source address or immediate data to be written in case of Memory Block Fill operations.

Once an operation is initiated, these registers contain the current source addresses. For example; when the Byte Count is initially 4096 bytes and the AA has completed the operation on the first three 1K-byte data blocks, the value in register SAR1/PQSAR1 is the equal to the programmed descriptor value + 3072 (SAR1 + 3072).

During Memory Block Fills the register always contains the data to be written and does not change.

Table 11 shows the Data / Source Address Register1/P+Q RAID-6 SAR1. This read-only register is loaded when a chain descriptor is read from memory.

Table 11. Data / Source Address Register - SAR1/PQSAR1



Internal bus address
SAR1 FFFF E810H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0000 0000H	For the XOR command - Local Address - The local source address. For the Memory Block Fill Command - Data to be written to the memory block.

1.13.6 Source Address Registers 2..32 - SAR2..32

The Source Address Registers 2..32 (SAR2..32) contain 32-bit, local memory addresses. There are 31 Source Address Registers (SAR2 - SAR32). Each of these registers is loaded with address of blocks of data to be operated upon by the AA. The ADCR, EDCR0, EDCR1, and EDCR2 registers control the operation performed on each data block referenced by the registers (SAR2 - SAR32). The local memory address space is a 32-bit, byte addressable address space.

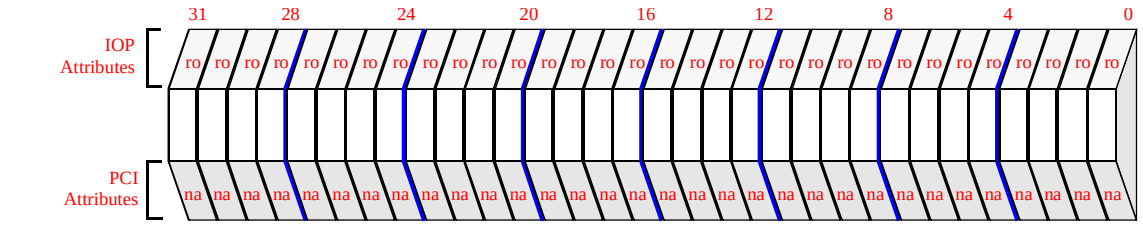
Reading SARx registers once AA has started a chain descriptor returns the current source addresses. Once an operation is initiated, these registers contain current source addresses. For example; when Byte Count is initially 4096 bytes and AA has completed operation on the first three 1K-byte data blocks, the value in register SARx is the equal to the programmed descriptor value + 3072 (SARx + 3072).

For *Dual-XOR-transfer*s SAR3 is the Horizontal Source Address for the XOR result for Horizontal XOR result, and SAR4 is the Diagonal Source Address for the Diagonal XOR result. Also, SAR5 is the Diagonal Destination Address for the Diagonal XOR result.

Note: For P+Q RAID-6 Mode, refer to section Section 1.13.7, “P+Q RAID-6 Source Address Registers 2..16 - PQSAR2..16” on page 70 for source addresses 2 through 16 and Section 1.13.8, “P+Q RAID-6 Galois Field Multiplier Registers 1..5 - GFMR1..5” on page 71 for P+Q RAID-6 Multiplier Word definitions.

Table 12 shows the Source Address Register2..32. These read-only registers are loaded when a chain descriptor is read from memory.

Table 12. Source Address Register2..32 - SAR2..32



	Internal bus address		Internal bus address
SAR2	FFFF E814H	SAR18	FFFF E868H
SAR3	FFFF E818H	SAR19	FFFF E86CH
SAR4	FFFF E81CH	SAR20	FFFF E870H
SAR5	FFFF E82CH	SAR21	FFFF E874H
SAR6	FFFF E830H	SAR22	FFFF E878H
SAR7	FFFF E834H	SAR23	FFFF E87CH
SAR8	FFFF E838H	SAR24	FFFF E880H
SAR9	FFFF E840H	SAR25	FFFF E888H
SAR10	FFFF E844H	SAR26	FFFF E88CH
SAR11	FFFF E848H	SAR27	FFFF E890H
SAR12	FFFF E84CH	SAR28	FFFF E894H
SAR13	FFFF E850H	SAR29	FFFF E898H
SAR14	FFFF E854H	SAR30	FFFF E89CH
SAR15	FFFF E858H	SAR31	FFFF E8A0H
SAR16	FFFF E85CH	SAR32	FFFF E8A4H
SAR17	FFFF E864H		

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0000 0000H	Local Address - The local source address For Dual XOR operations, the following applies: • SAR3 is the Horizontal local source address (SAR_H) • SAR4 is the Diagonal local source address (SAR_D) • SAR5 contains the destination address of the Diagonal XOR result (DAR_D)

1.13.7 P+Q RAID-6 Source Address Registers 2..16 - PQSAR2..16

Note: The following definition applies only when P+Q RAID-6 mode is enabled. When P+Q RAID-6 is NOT enabled, refer to Section 1.13.6, "Source Address Registers 2..32 - SAR2..32" on page 68 for definition and internal bus addresses of Source Address Registers

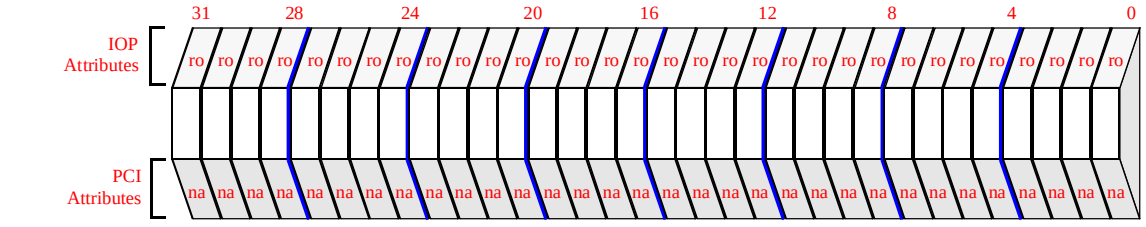
The P+Q RAID-6 Source Address Register2..16 (PQSAR2..16) contain 32-bit, local memory addresses. There are 16 P+Q RAID-6 Source Address Registers (PQSAR1..PQSAR16). Each of these registers is loaded with address of blocks of data to be operated upon by the AA when P+Q RAID-6 Mode is enabled. The ADCR, EDCR0, and EDCR1 registers control the operation performed on each data block referenced by the registers (PQSAR1..PQSAR16). The local memory address space is a 32-bit, byte addressable address space.

Reading PQSARx registers once AA has started a chain descriptor returns the current source addresses. Once an operation is initiated, these registers contain current source addresses. For example; when Byte Count is initially 4096 bytes and AA has completed operation on the first three 1K-byte data blocks, the value in register PQSARx is the equal to the programmed descriptor value + 3072 (PQSARx + 3072).

Table 13 shows the P+Q RAID-6 Source Address Registers 2..16. These read-only registers are loaded when a chain descriptor is read from memory.

Note: See Section 1.13.8 for definition of the Data Multipliers in P+Q RAID-6 mode.

Table 13. P+Q RAID-6 Source Address Registers 2..16 - PQSAR2..16



	Internal bus address		Internal bus address
PQSAR2	FFFF E814H	PQSAR10	FFFF E850H
PQSAR3	FFFF E818H	PQSAR11	FFFF E854H
PQSAR4	FFFF E82CH	PQSAR12	FFFF E858H
PQSAR5	FFFF E830H	PQSAR13	FFFF E864H
PQSAR6	FFFF E834H	PQSAR14	FFFF E868H
PQSAR7	FFFF E840H	PQSAR15	FFFF E86CH
PQSAR8	FFFF E844H	PQSAR16	FFFF E870H
PQSAR9	FFFF E848H		

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0000 0000H	• Local Address - The local source address

1.13.8 P+Q RAID-6 Galois Field Multiplier Registers 1..5 - GFMR1..5

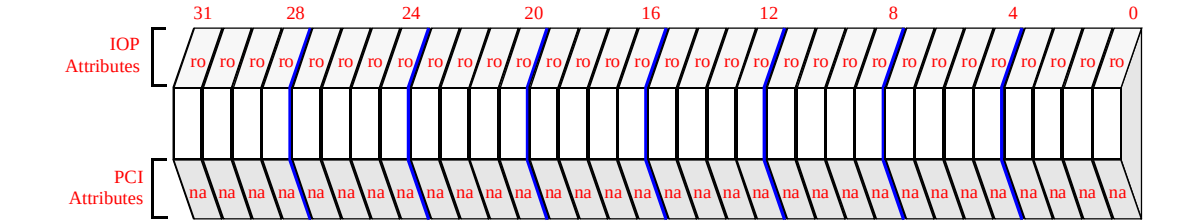
Note: The following definition applies only when P+Q RAID-6 mode is enabled. When P+Q RAID-6 is NOT enabled, refer to Section 1.13.6, "Source Address Registers 2..32 - SAR2..32" on page 68 for definition and internal bus addresses of Source Address Registers

The P+Q RAID-6 Galois Field Multiplier Registers 1..5 (GFMR1..5) contain the 8-bit multiplier values. There are 16 Data Multipliers distributed through the five Data Multiplier Words (GFMR1..GFMR5). Each of these registers is loaded with data multiplier values to be used by the AA GF Multiply Function when P+Q RAID-6 Mode is enabled. The ADCR, EDCR0, and EDCR1 registers control the operation performed on each source data block.

Table 14 shows the Galois Field Multiplier Registers GFMR[1:5]. These read-only registers are loaded when a chain descriptor is read from memory.

Note: See Section 1.13 for definition of the Data Integrity Source Addresses in P+Q RAID-6 Source Address.

Table 14. Galois Field Multiplier Registers 1..5 - GFMR1..5 (Sheet 1 of 2)

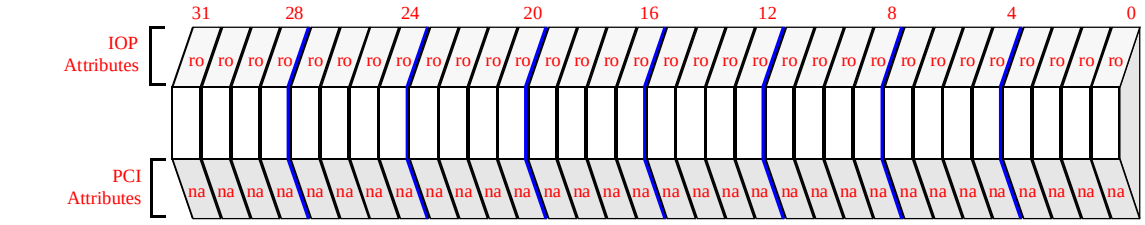


	Internal bus address
GFMR1	FFFF E81CH
GFMR2	FFFF E838H
GFMR3	FFFF E84CH
GFMR4	FFFF E85CH
GFMR5	FFFF E874H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:24	00H	Data Multiplier - Data Multiplier Byte used by the P+Q RAID-6 function (GF Multiply) with source data from corresponding PQSARx, when P+Q RAID-6 mode is enabled. • GFMR1 - reserved • GFMR2 - reserved • GFMR3 - reserved • GFMR4 - reserved • GFMR5 - Data Multiplier 16 (DMLT16)
23:16	00H	Data Multiplier - Data Multiplier Byte used by the P+Q RAID-6 function (GF Multiply) with source data from corresponding PQSARx, when P+Q RAID-6 mode is enabled. • GFMR1 - Data Multiplier 3 (DMLT3) • GFMR2 - Data Multiplier 6 (DMLT6) • GFMR3 - Data Multiplier 9 (DMLT9) • GFMR4 - Data Multiplier 12 (DMLT12) • GFMR5 - Data Multiplier 15 (DMLT15)

Table 14. Galois Field Multiplier Registers 1..5 - GFMR1..5 (Sheet 2 of 2)



	Internal bus address
GFMR1	FFFF E81CH
GFMR2	FFFF E838H
GFMR3	FFFF E84CH
GFMR4	FFFF E85CH
GFMR5	FFFF E874H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:8	00H	Data Multiplier - Data Multiplier Byte used by the P+Q RAID-6 function (GF Multiply) with source data from corresponding PQSARx, when P+Q RAID-6 mode is enabled. • GFMR1 - Data Multiplier 2 (DMLT2) • GFMR2 - Data Multiplier 5 (DMLT5) • GFMR3 - Data Multiplier 8 (DMLT8) • GFMR4 - Data Multiplier 11 (DMLT11) • GFMR5 - Data Multiplier 14 (DMLT14)
7:0	00H	Data Multiplier - Data Multiplier Byte used by the P+Q RAID-6 function (GF Multiply) with source data from corresponding PQSARx, when P+Q RAID-6 mode is enabled. • GFMR1 - Data Multiplier 1 (DMLT1) • GFMR2 - Data Multiplier 4 (DMLT4) • GFMR3 - Data Multiplier 7 (DMLT7) • GFMR4 - Data Multiplier 10 (DMLT10) • GFMR5 - Data Multiplier 13 (DMLT13)

1.13.9 Destination Address Register - DAR

The Destination Address Register (DAR) contains a 32-bit, local memory address. The DAR may also contain an address targeting the ATU outbound windows for writing the AAU result to the PCI bus. During operations, this address is the destination address in local memory where data will be stored. The 80331 local memory address space is a 32-bit, byte addressable address space. When programming the result to be on the PCI bus, this address is one of the ATU outbound windows, which results in a 32-bit or 64-bit PCI address depending on the window addressed.

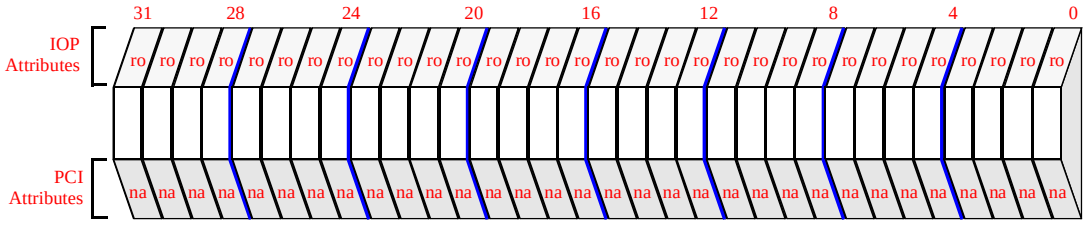
During Dual XOR operations, this address points to the memory block to be written with the Horizontal XOR result.

During Memory Block Fill operations, this address points to the memory block to be written with the constant value contained in the D/SAR1 register.

Reading the DAR once the AA has started a chain descriptor returns the current destination address. For example; during an XOR operation when the Byte Count is initially 4096 bytes and the AA has completed the *XOR-transfer* operation on the first three 1K-byte data blocks, the value in the Destination Address Register (DAR) will be equal to the programmed descriptor value + 3072 (DAR + 3072).

Table 15 shows the Destination Address Register. This read-only register is loaded when a chain descriptor is read from memory

Table 15. Destination Address Register - DAR



Internal bus address
FFFF E820H

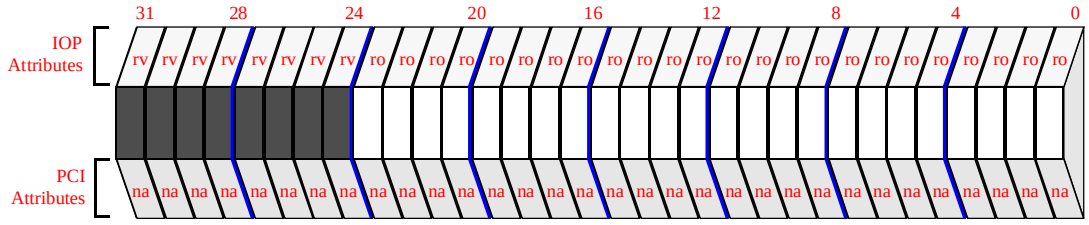
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000H	Destination Address - The result destination address in local memory or PCI Outbound windows. Local Address - The local destination address. For Dual XOR operations, DAR contains the destination address of the Horizontal XOR result (DAR_H).

1.13.10 Accelerator Byte Count Register - ABCR

The Accelerator Byte Count Register (ABCR) contains the number of bytes to transfer for an operation. This is a read-only register that is loaded from the Byte Count word in a chain descriptor. It allows for a maximum transfer of 16 Mbytes. A value of zero is a valid byte count and results in no read or write cycles being generated to the Memory Controller Unit. No cycles are generated on the internal bus.

Table 16. Accelerator Byte Count Register - ABCR



Internal bus address
FFFF E824H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

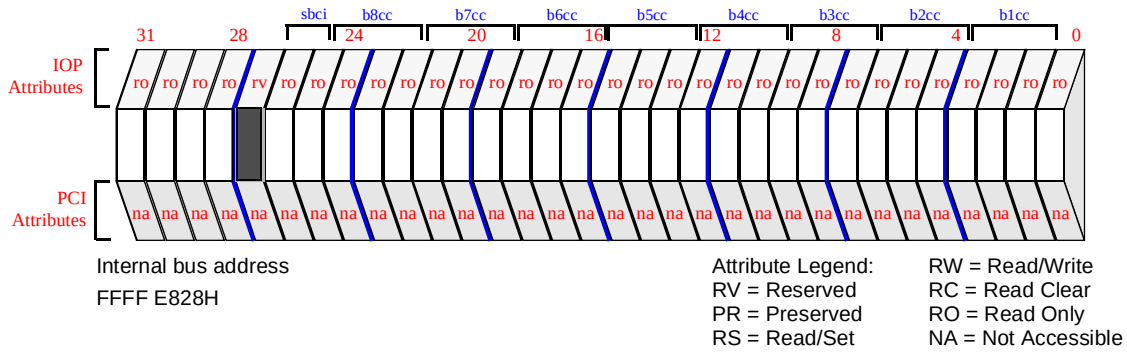
Bit	Default	Description
31:24	00H	Reserved
23:00	000000H	Byte Count - is the number of bytes to transfer for an operation.

Note: Anytime this register is read, it contains the number of bytes left to transfer on the internal bus. Note that during an operation valid data may be present in the Application Accelerator store queue. This register is decremented by 1 through 8 for every successful transfer from the store queue to the destination location. During *Memory Block Fills* this register is decremented by 1 through 8 for every successful write operation. Table 16 shows the Accelerator Byte Count Register. The byte count value is not required to be aligned to a DWORD boundary (i.e., the byte count value can be a DWORD aligned, short aligned, or byte aligned).

1.13.11 Accelerator Descriptor Control Register - ADCR

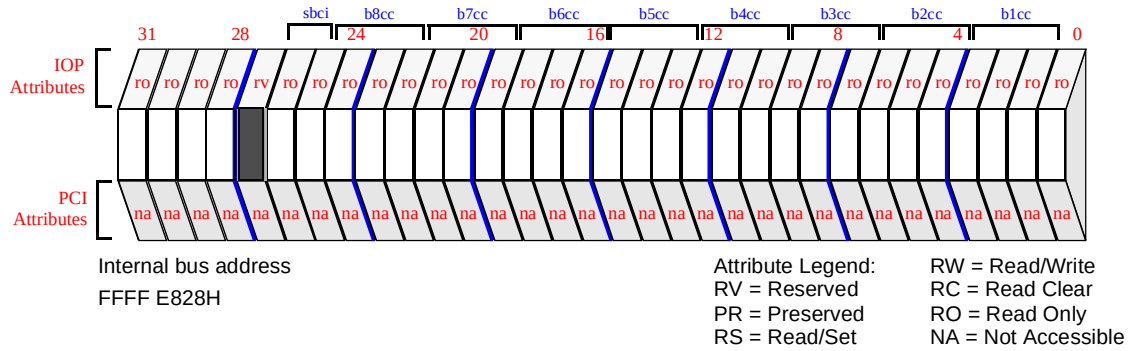
The Accelerator Descriptor Control Register contains control values for data transfer on a per-chain descriptor basis. This read-only register is loaded when a chain descriptor is read from memory. These values may vary from chain descriptor to chain descriptor. The AAU determines whether a mini-descriptor is appended to the end of the current chain descriptor by examining bits 26:25. Table 17 shows the definition of the Accelerator Descriptor Control Register.

Table 17. Accelerator Descriptor Control Register - ADCR (Sheet 1 of 4)



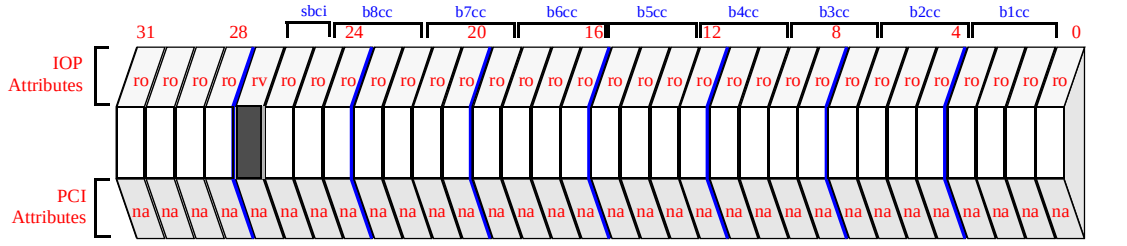
Bit	Default	Description
31	0_2	Destination Write Enable (dwe)- Determines whether data present in the store queue is written out to local memory. When set, data in the queue is written to the address specified in the Destination Address Register (DAR) after performing the specified operation on data referenced by the SARx registers. When clear, data is held in the queue. **NOTE:** This bit must be set for *Dual-XOR-transfers*. **NOTE:** This bit should be SET when Descriptor Byte Count is larger than the AA buffer size. When the ABCR register contains a value greater than the buffer size and this bit is cleared, the AAU only reads the first complete buffer of data and perform the specified function. It does not read the remaining bytes specified in the ABCR. Further, the AAU proceeds to process the next chain descriptor when it is specified.
30	0_2	Zero Result Buffer Check Enable - When this bit is set the AA checks for an all-zero result buffer across the data blocks specified by the SARx registers.
29	0_2	Result Buffer Not Zero- This bit is set when the result buffer computed across the data blocks specified by the SARx registers results in a non-zero value. **NOTE:** The AA updates this status in memory **only** by updating the Descriptor Control Word of the current descriptor (the eighth word of the descriptor pointed to by the ADAR).
28	0_2	Transfer Complete - This bit is set when the AA completes the processing of a descriptor with Zero Result Buffer Check enabled (i.e., bit 30 of the ADCR is set). **NOTE:** The AA updates this status in memory **only** by updating the Descriptor Control Word of the current descriptor (the eighth word of the descriptor pointed to by the ADAR).
27	0_2	Reserved Dual XOR Operation - Defines the descriptor as a Dual XOR format when set. See Section 1.3.2.5, “Dual-XOR-Transfer Descriptor Format” on page 22 for details. The Supplemental Block Control Interpreter field must also be set for AA to fetch mini-descriptor.

Table 17. Accelerator Descriptor Control Register - ADCR (Sheet 2 of 4)



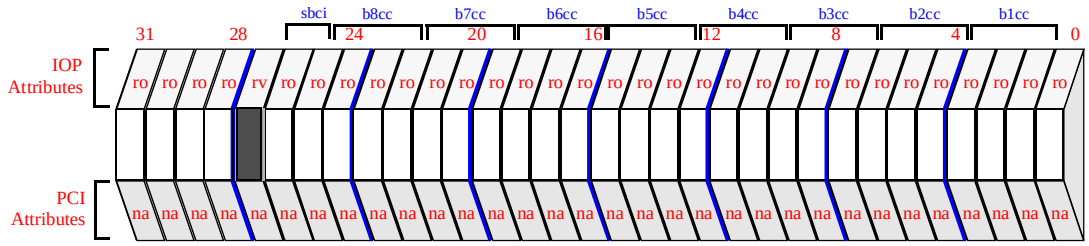
Bit	Default	Description
26:25	00	Supplemental Block Control Interpreter - This bit field specifies the number of additional descriptor segments beyond the principle descriptor on which the operation is executed. 00 Principle Descriptor only - This specifies that no additional descriptor words exist. The AA only reads the principle descriptor to initialize the first eight AA descriptor registers. Set for up to 4 sources for XOR, or up to 3 sources for P+Q RAID-6. 01 Mini-Descriptor - This specifies that there are up to 4 additional words. The AA therefore reads the mini-descriptor to initialize four additional registers. Set for up to 8 sources for XOR, or Dual-XOR operation, or up to 6 sources for P+Q RAID-6. 10 Extended Descriptor 0 - This specifies that there are up to nine additional descriptor words. The AA therefore reads the mini-descriptor and one extended-descriptor to initialize a total of twenty-one registers. Set for up to 16 sources for XOR, or up to 12 sources for P+Q RAID-6 . 11 Extended Descriptors 1 and 2 - This specifies that there are up to eighteen additional descriptor words. The AA therefore reads the mini-descriptor and three extended-descriptors to initialize registers a total of thirty-nine registers. Set for up to 32 sources for XOR, or up to 16 sources for P+Q RAID-6 .
24:22	0	**Block 8 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR8 register. 000 Null command - This implies that Block 8 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 8 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
21:19	0	**Block 7 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR7 register. 000 Null command - This implies that Block 7 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 7 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
18:16	0	**Block 6 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR6 register. 000 Null command - This implies that Block 6 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 6 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved

Table 17. Accelerator Descriptor Control Register - ADCR (Sheet 3 of 4)



Bit	Default	Description
15:13	0	**Block 5 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR5 register. 000 Null command - This implies that Block 5 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 5 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
12:10	0	**Block 4 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR4 register. 000 Null command - This implies that Block 4 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 4 Data is transferred to the Application Accelerator to execute the XOR function. (required for *Dual-XOR-transfer*s) All other values are reserved
09:07	0	**Block 3 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR3 register. 000 Null command - This implies that Block 3 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 3 Data is transferred to the Application Accelerator to execute the XOR function. (required for *Dual-XOR-transfer*s) All other values are reserved
06:04	0	**Block 2 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR2 register. 000 Null command - This implies that Block 2 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 2 Data is transferred to the Application Accelerator to execute the XOR function. (required for *Dual-XOR-transfer*s) All other values are reserved

Table 17. Accelerator Descriptor Control Register - ADCR (Sheet 4 of 4)

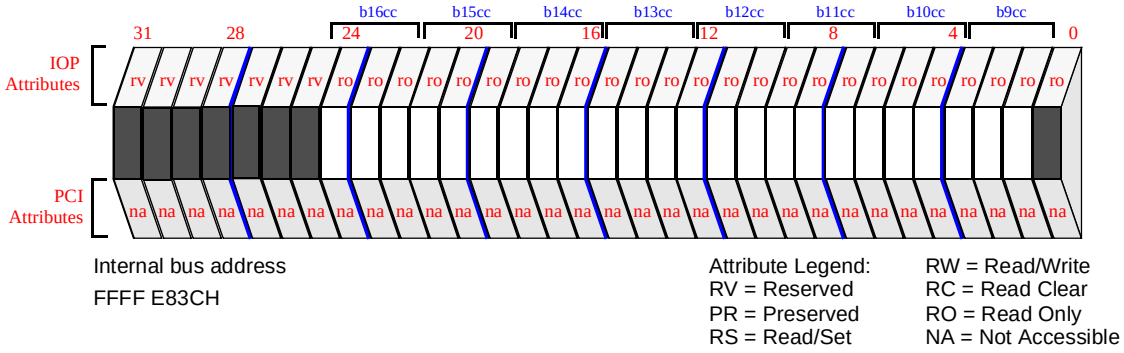


Bit	Default	Description
03:01	0	**Block 1 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by D/SAR1 register (for XOR command) or with the data contained in the D/SAR1 (for Memory Block Fill command). 000 Null command - This implies that Block 1 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 1 Data is transferred to the Application Accelerator to execute the XOR function. 010 Memory Block Fill command - This implies that the memory block specified by the DAR is filled with the constant specified by the D/SAR1register. 111 Direct Fill - This implies that Block 1 Data is transferred directly from local memory to the store queue. (required for *Dual-XOR-transfers*) All other values are reserved
00	0	**Interrupt Enable** - When set, the Application Accelerator generates an interrupt to the Intel XScale® core upon completion of a transfer. When clear, no interrupt is generated.

1.13.12 Extended Descriptor Control Register 0 - EDCR0

The Extended Descriptor Control Register 0 contains control values for data transfer on a per-chain descriptor basis. This read-only register is loaded when a chain descriptor that requires a minimum of 16 Source Addresses is read from memory. The values in EDCR0 define the command/control value for SAR16 - SAR9. The AAU determines whether an extended descriptor requiring the use of EDCR0 is appended to the end of the current chain descriptor by examining bits 26:25 of the Accelerator Descriptor Control Register. Table 18 shows the definition of the Extended Descriptor Control Register 0.

Table 18. Extended Descriptor Control Register 0 - EDCR0 (Sheet 1 of 2)

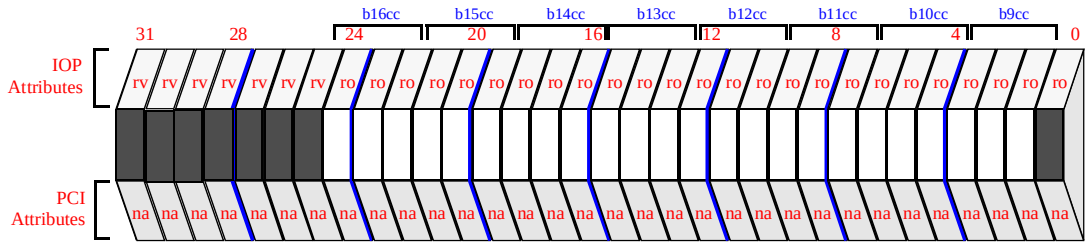


Internal bus address
FFFF E83CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:25	0_2	Reserved
24:22	0	**Block 16 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR16 register. 000 Null command - This implies that Block 16 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 16 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
21:19	0	**Block 15 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR15 register. 000 Null command - This implies that Block 15 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 15 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
18:16	0	**Block 14 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR14 register. 000 Null command - This implies that Block 14 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 14 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved

Table 18. Extended Descriptor Control Register 0 - EDCR0 (Sheet 2 of 2)



Internal bus address
FFFF E83CH

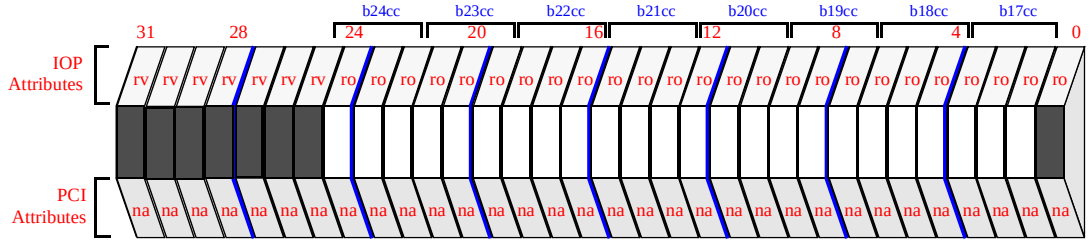
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:13	0	**Block 13 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR13 register. 000 Null command - This implies that Block 13 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 13 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
12:10	0	**Block 12 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR12 register. 000 Null command - This implies that Block 12 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 12 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
09:07	0	**Block 11 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR11 register. 000 Null command - This implies that Block 11 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 11 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
06:04	0	**Block 10 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR10 register. 000 Null command - This implies that Block 10 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 10 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
03:01	0	**Block 9 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR9 register. 000 Null command - This implies that Block 9 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 9 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
00	0	Reserved.

1.13.13 Extended Descriptor Control Register 1 - EDCR1

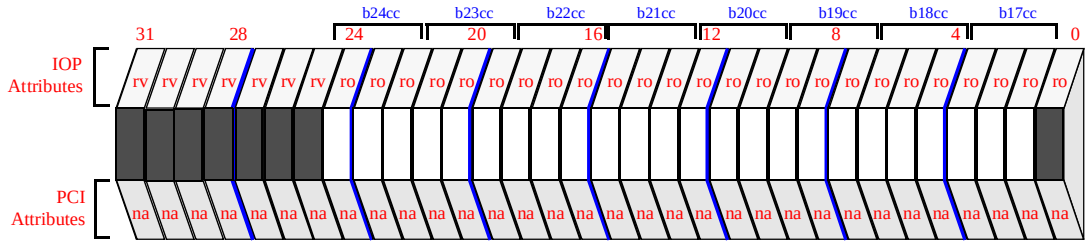
The Extended Descriptor Control Register 1 contains control values for data transfer on a per-chain descriptor basis. This read-only register is loaded when a chain descriptor that requires 32 Source Addresses is read from memory. The values in EDCR1 define the command/control value for SAR24 - SAR17. The AAU determines whether an extended descriptor requiring the use of EDCR1 is appended to the end of the current chain descriptor by examining bits 26:25 of the Accelerator Descriptor Control Register. Table 19 shows the definition of the Extended Descriptor Control Register 1.

Table 19. Extended Descriptor Control Register 1 - EDCR1 (Sheet 1 of 2)



Bit	Default	Description
31:25	0_2	Reserved
24:22	0	**Block 24 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR24 register. 000 Null command - This implies that Block 24 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 24 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
21:19	0	**Block 23 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR23 register. 000 Null command - This implies that Block 23 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 23 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
18:16	0	**Block 22 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR22 register. 000 Null command - This implies that Block 22 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 22 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved

Table 19. Extended Descriptor Control Register 1 - EDCR1 (Sheet 2 of 2)



Internal bus address
FFFF E860H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

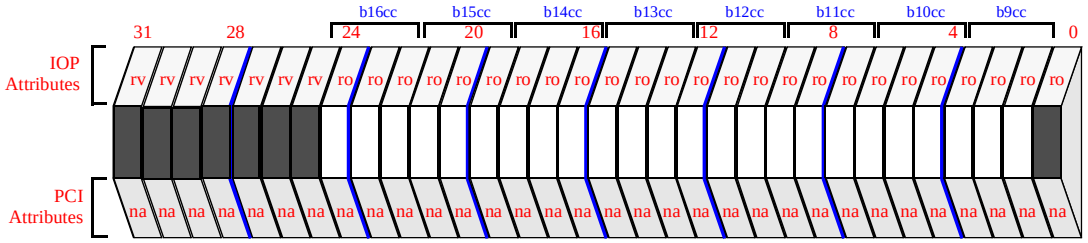
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:13	0	**Block 21 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR21 register. 000 Null command - This implies that Block 21 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 21 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
12:10	0	**Block 20 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR20 register. 000 Null command - This implies that Block 20 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 20 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
09:07	0	**Block 19 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR19 register. 000 Null command - This implies that Block 19 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 19 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
06:04	0	**Block 18 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR18 register. 000 Null command - This implies that Block 18 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 18 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
03:01	0	**Block 17 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR17 register. 000 Null command - This implies that Block 17 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 17 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
00	0	Reserved.

1.13.14 Extended Descriptor Control Register 2 - EDCR2

The Extended Descriptor Control Register 2 contains control values for data transfer on a per-chain descriptor basis. This read-only register is loaded when a chain descriptor that requires 32 Source Addresses is read from memory. Values in EDCR2 define the command/control value for SAR32 - SAR25. The AAU determines whether an extended descriptor requiring the use of EDCR2 is appended to the end of the current chain descriptor by examining bits 26:25 of the Accelerator Descriptor Control Register. Table 20 shows the definition of the Extended Descriptor Control Register 2.

Table 20. Extended Descriptor Control Register 2 - EDCR2 (Sheet 1 of 2)

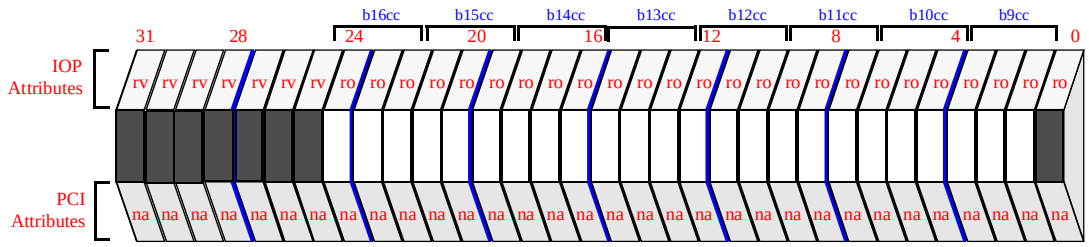


Internal bus address
FFFF E884H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:25	0_2	Reserved
24:22	0	**Block 32 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR32 register. 000 Null command - This implies that Block 32 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 32 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
21:19	0	**Block 31 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR31 register. 000 Null command - This implies that Block 31 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 31 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
18:16	0	**Block 30 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR30 register. 000 Null command - This implies that Block 30 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 30 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
15:13	0	**Block 29 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR29 register. 000 Null command - This implies that Block 29 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 29 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved

Table 20. Extended Descriptor Control Register 2 - EDCR2 (Sheet 2 of 2)



Internal bus address
FFFF E884H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
12:10	0	**Block 28 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR28 register. 000 Null command - This implies that Block 28 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 28 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
09:07	0	**Block 27 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR27 register. 000 Null command - This implies that Block 27 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 27 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
06:04	0	**Block 26 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR26 register. 000 Null command - This implies that Block 26 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 26 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
03:01	0	**Block 25 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR25 register. 000 Null command - This implies that Block 25 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001 XOR command - This implies that Block 25 Data is transferred to the Application Accelerator to execute the XOR function. All other values are reserved
00	0	Reserved.