Licensing and Government Use

Any Exar software ("Licensed Programs") based on Hifn Technology described in this document is furnished under a license and may be used and copied only in accordance with the terms of such license and with the inclusion of this copyright notice. Distribution of this document or any copies thereof and the ability to transfer title or ownership of this document's contents are subject to the terms of such license.

Such Licensed Programs and their documentation may contain public open-source software that would be licensed under open-source licenses. Refer to the applicable product release notes for open-source licenses and proprietary notices. Use, duplication, disclosure, and acquisition by the U.S. Government of such Licensed Programs is subject to the terms and definitions of their applicable license.

Disclaimer

Exar reserves the right to make changes to its products, including the contents of this document, or to discontinue any product or service without notice. Exar advises its customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied upon is current. Every effort has been made to keep the information in this document current and accurate as of the date of this document's publication or revision.

Limited Warranty

Exar warrants Products based on the Hifn Technology, including cards, against defects in materials and workmanship for a period of twelve (12) months from the delivery date. Exar's sole liability shall be limited to either, replacing, repairing or issuing credit, at its option, for the Product if it has been paid for. Exar will not be liable under this provision unless: (a) Exar is promptly notified in writing upon discovery of claimed defects by Buyer; (b) The claimed defective Product is returned to Exar, insurance and transportation charges prepaid, by Buyer; (c) The claimed defective Product is received within twelve (12) months from the delivery date; and (d) Exar's examination of the Product discloses to its satisfaction that the alleged defect was not caused by misuse, neglect, improper installation, repair, alteration, accident or other hazard. THIS WARRANTY DOES NOT COVER PRODUCT DAMAGE WHICH RESULTS FROM ACCIDENT, MISUSE, ABUSE, IMPROPER LINE VOLTAGE, FIRE, FLOOD, LIGHTNING OR OTHER ACTS OF GOD OR DAMAGE RESULTING FROM ANY MODIFICATIONS, REPAIRS OR ALTERATIONS PERFORMED OTHER THAN BY EXAR OR EXAR'S AUTHORIZED AGENT OR RESULTING FROM FAILURE TO STRICTLY COMPLY WITH EXAR'S WRITTEN OPERATING AND MAINTENANCE INSTRUCTIONS. BUYER ACKNOWLEDGES THAT THE PRODUCT ARE HIGHLY SENSITIVE ELECTRONIC PRODUCT REQUIRING SPECIAL HANDLING AND THAT THIS WARRANTY DOES NOT APPLY TO IMPROPERLY HANDLED PRODUCT. PRODUCT MANUFACTURED TO MEET BUYER'S SPECIFIC PERFORMANCE SPECIFICATIONS ACCEPTED BY EXAR ARE WARRANTED ONLY TO PERFORM IN CONFORMITY WITH SUCH SPECIFICATIONS, AND ARE WARRANTED ONLY AGAINST DEFECTS NOT RELATED TO SUCH SPECIFICATIONS IN ACCORDANCE WITH THE TERMS AND CONDITIONS SET FORTH HEREIN ABOVE.

Life Support Policy

Exar's Product are not authorized for use as critical components in life support devices or systems. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury or death to human life. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Buyer agrees to indemnify, defend and hold Exar harmless for any cost, loss, liability, or expense (including without limitation attorneys' fees and other costs of litigation or threatened litigation) arising out of violation of the above prohibition by Buyer or any person or entity receiving Exar's Product through Buyer.

Patent Infringement - Indemnification

Exar agrees, at its own expense, to defend Buyer from and against any claim, suit or proceeding, and to pay all judgments and costs finally awarded against Buyer by reason of claim, suit or proceeding insofar as it is based upon an allegation that the Product as furnished by Exar infringes any United States letter patent, provided that Exar is notified promptly of such claim in writing and is given authority and full and proper information and assistance (at Exar's expense) for defense of same. In case such Product are finally constituted an infringement and the use of Product is enjoined, Exar shall at its sole discretion and at its own expense: (1) procure for Buyer the right to continue using the Product; (2) replace or modify the same so that it becomes non-infringing; or (3) remove such Product and grant Buyer a credit for the depreciated value of the same.

Buyer shall have the right to employ separate counsel in any claim, suit or proceeding and to participate in the defense thereof, but the fees and expenses of Buyer's counsel shall not be borne by Exar unless: (1) Exar specifically so agrees; or (2) Exar, after written request and without cause, does not assume such defense. Exar shall not be liable to indemnify Buyer for any settlement effected without Exar's written consent, unless Exar failed, after notice and without cause, to defend such claim, suit or proceeding.

The indemnification shall not apply and Buyer shall indemnify Exar and hold it harmless from all liability or expense (including costs of suit and attorney's fees) if the infringement arises from, or is based upon Exar's

compliance with particular requirements of Buyer or Buyer's customer that differ from Exar's standard specifications (Custom Product) for the Product, or modifications or alterations of the Product, or a combination of the Product with other items not furnished or manufactured by Exar.

Buyer agrees that Exar shall not be liable for any collateral, incidental or consequential damages arising out of patent infringement.

The foregoing states the entire liability of Exar for patent infringement.

Motorola

The use of this product in stateful compression protocols (for example, PPP or multi-history applications) with certain configurations may require a license from Motorola. In such cases, a license agreement for the right to use Motorola patents (US05,245,614, US05,130,993) may be obtained directly from Motorola.

Patents

May include one or more of the following United States patents: 4,930,142; 4,996,690; 4,701,745; 5,003,307; 5,016,009; 5,126,739; 5,146,221; 5,414,425; 5,414,850; 5,463,390; 5,506,580; 5,532,694; 6,320,846; 6,816,459; 6,651,099; 6,665,725; 6,771,646; 6,789,116; 6,954,789; 6,839,751; 7,299,282; 7,260,558. Other patents pending.

Trademarks

Hi/fn®, MeterFlow®, MeterWorks®, and LZS®, are registered trademarks of Exar Corporation. Hifn™, Hifn Technology, FlowThrough™, BitWackr, and the Hifn logo are trademarks of Hi/fn, Inc. All other trademarks and trade names are the property of their respective holders.

IBM, IBM Logo, and IBM PowerPC are trademarks of International Business Machines Corporation in the United States, or other countries.

Microsoft, Windows, Windows XP, Windows Vista, Windows Server 2003, Windows Server 2008 and the Windows logo are trademarks of Microsoft Corporation in the United States, and/or other countries.

Intel QuickAssist is a trademark of Intel Corporation in the United States and in other countries.

Exporting

This product may only be exported from the United States in accordance with applicable Export Administration Regulations. Diversion contrary to United States laws is prohibited.

Exar Confidential

If you have signed a Exar Confidential Disclosure Agreement that includes this document as part of its subject matter, please use this document in accordance with the terms of the agreement. If not, please destroy the document.

Table of Contents

List of Figures

List of Tables

Preface

Welcome to the Data Sheet for Exar's *8201, 8202, 8203, 8204 (*820x), a high performance, low power, compression/encryption/hash acceleration processor. This document describes the 820x operation, data structures, data flow and specifications.

Please refer to the *8201, 8202, 8203, 8204 Errata* for release specific errata for the 820x device.

Audience

This document is intended for:

- Project managers
- System engineers
- Hardware and software development engineers
- Marketing and product managers

Prerequisite

Before proceeding, you should generally understand:

- PCI Express
- GZIP/Deflate, eLZS and AES algorithms
- Hash, MAC algorithms
- General networking concepts

Document Organization

This document is organized as follows:

Chapter 1, "Product Description" provides an overview of the 820x processor.

Chapter 2, "Operation" describes key features of the 820x operations.

Chapter 3, "Data Structures" defines the format of the data structures used by the 820x.

Chapter 4, "Data Flow" gives examples of the typical data flows within the 820x.

Chapter 5, "Modules" describes the internal 820x modules in more detail.

Chapter 6, "Register Definition" details the syntax and usage of the internal 820x registers.

Chapter 7, "PCIe Configuration Register Definition" details the syntax and usage of the PCIe registers.

Chapter 8, "Signal Description" defines the external interfaces for the 820x device.

Chapter 9, "Error Handling" describes the 820x error handling features.

Chapter 10, "DC Specifications" defines the 820x DC specifications.

Chapter 11, "AC Specifications" defines the 820x AC specifications.

Chapter 12, "Thermal Specifications" defines the 820x thermal specifications.

Chapter 13, "Package Specifications" defines the 820x package specifications.

Appendix A through Appendix D provide detailed information on how the algorithms are calculated. This material is included for reference only. Exar's SDK automatically implements these algorithms.

Related Documents

The following documents can be used as a reference to this document.

UG-0211 *8201, 8202, 8203, 8204 Hardware Design User Guide*

AN-0206 *Raw Acceleration API Performance Application Note*

AN-0207 *QuickAssist API Performance Application Note*

AN-0205 *Data Offload API Performance Application Note*

ER-0015 *8201, 8202, 8203, 8204 and DX 1845, 1835, 1825, 1710, 1720, 1730, 1740 Errata*

Customer Support

For technical support about this product, please contact your local Exar sales office, representative, or distributor.

For general information about Exar and Exar products refer to: www.exar.com

Glossary

Term	Definition
AAD	Additional Authentication Data
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
CM	Channel Manager
CPR	Command Pointer Ring
CRC	Cyclic Redundancy Check
CTR	Counter Mode
DES	Data Encryption Standard
DH	Diffie-Hellman key exchange protocol
DIF	Data Integrity Field for AES-XTS
DPC	Data Process Channel
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ECB	Electronic Codebook
ECC	Error correction code
ECDH	Elliptic-Curve Diffie-Hellman
ECDSA	Elliptic-Curve Digital Signature Algorithm
ECRC	End-to-End cyclic Redundancy Check
eLZS	Enhanced LZS
ESP	Encapsulating Security Payload
GMAC	Galois Message Authentication Code
GZIP	GNU ZIP
HMAC	Hash Message Authentication Code
IHV	Initial Hash Value
IPAD	Inner Padding
IPPCP	IP Payload Compression Protocol
IPsec	IP Security Protocol
IV	Initial Vector
JTAG	Joint Test Action Group
LZS	Lempel-Ziv Stac
MAC	Message Authentication Code
OPAD	Outer Padding
PHY	Physical-Layer interface - usually for PCI express

Term	Definition
PKP	Public Key Processor
PLL	Phase-Locked Loop
RC	PCIe Root Complex
RNG	Random Number Generator
RSA	Ron Rivest, Adi Shamir and Leonard Adleman
S0	Initial Value for Hash Engine GCM-MAC and GMAC calculation
SPI	Serial Peripheral Interface
SSL	Security Socket Layer
TLS	Transport Layer Security
UEFI	Unified Extensible Firmware Interface
XTS	XES-based Tweaked CodeBook mode with Cipher Text Stealing

1 Product Description

Exar's 820x is a high-performance, low-power, look-aside acceleration device that is ideal for both storage and networking markets. The 820x provides hardware acceleration of compression, encryption and hashing algorithms including GZIP/Deflate, LZS, AES, 3DES, SHA, MAC and some public key algorithms such as DH, RSA, ECC. The 820x has a PCIe 2.0 Gen1 compliant interface to communicate with the host system.

Figure 1-1 shows a typical 820x application example. The 820x can be employed on a half-height PCIe card to accelerate encryption, hash and LZS/GZIP/Deflate compression.

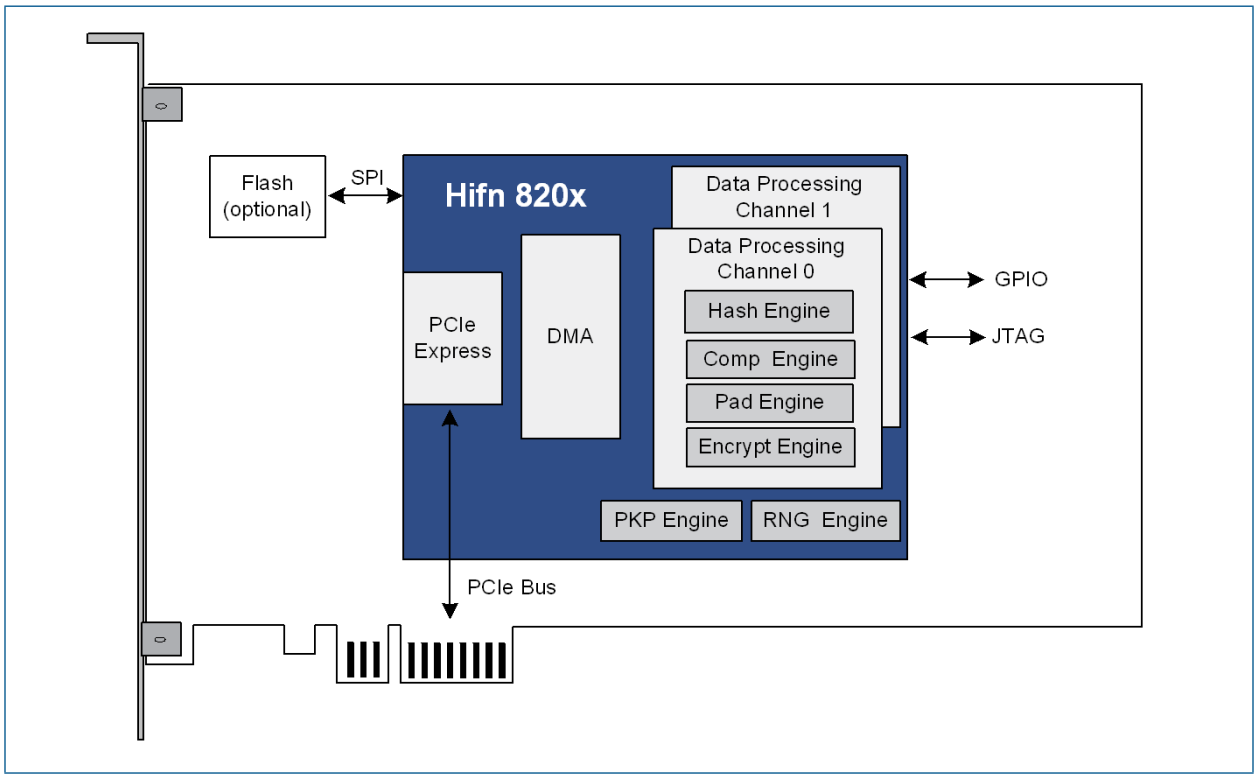


Figure 1-1. Application Example

1.1 Features

1.1.1 High Performance

- Dual channel compression, encryption, pad, and hash engines
- Compression/Decompression engine with LZS and GZIP/Deflate algorithms
- Encryption/Decryption engine supports:
  - AES-GCM, CBC, CTR and ECB with 128, 192 or 256 bit keys
  - AES-XTS with 256 or 512 bit keys

- 3DES
- DES (supported via 3DES with K1=K2=K3)

- Hash engine supports:
  - SHA1, SHA256
  - MD5
  - HMAC, GMAC, X-CBC-MAC and SSL3.0 MAC
- Public Key Processor
- Random Number Generator
- Compression, Encryption, Padding, MAC (Hash) in single pass
- Low Power
  - Static clock gating of channels
  - Dynamic clock gating for unused processing engines
  - Dynamic clock gating of unused algorithm core in processing engine

1.1.2 Flexible Design for Packet Processing

- Position of the Hash engine for calculating MAC is programmable
- Truncated data stream head and tail pointers in each processing engine for packet processing
- Programmable mute table to null the 16 byte packet header
- Programmable length MAC inserted into data stream

1.1.3 Engine Features

Table 1-1. 820x Engine Features

Engine	Features
LZS	Industry-standard LZS algorithm Enhanced LZS (eLZS) algorithm with anti-expansion compression
GZIP/Deflate	Complies with RFC 1951 (Deflate), and RFC 1952 (GZIP) Supports Static Huffman algorithm Supports Dynamic Huffman algorithm for high compression ratio
Encryption	Supports multiple algorithms: AES-GCM, -XTS, -CBC, -CTR and -ECB and 3DES Supports 128, 192 and 256 bit keys for AES (except XTS mode) Supports 256 and 512 bit keys for AES-XTS Supports stateful encryption operation

Table 1-1. 820x Engine Features

Engine	Features
Hash	Algorithms supported: HMAC-SHA1, HMAC-SHA256, HMAC-MD5, GMAC, X-CBC-MAC, SSL3.0-MAC Slice and file hash operation MAC inserted into the data stream Stateful hash and MAC operation
Pad	Supports padding with multiple algorithm in encode operation Remove padding in decode operation
PKP	RSA, DSA, DH, ECDH and ECDSA (elliptic curve) operations Support up to 8K-bits modular arithmetic and exponentiation

1.1.4 Command and Data Endian Conversion Modes

- No swap
- Byte swap in word
- Word swap, no byte swap in word
- Word swap, and byte swap in word

1.1.5 DMA Features

- Indirect command addressing
- Unlimited scatter-gather
- Maximum 16K commands in command pointer ring
- Two command pointer rings with round-robin arbitration (the priority of command pointer rings is handled by host software)
- Free pool ring to avoid result data overflow
- Small packet command structure mode

1.1.6 Data Integrity Features

- Input CRC32 check, output CRC32 generator
- ECRC protection through PCIe bus (requires host RC support)
- ECC or Parity protection for data path RAMs
- Real time verification for all data transformation (Encryption and Compression engines), and hash or MAC calculation (Hash engine)

1.1.7 Other features

- PCI Express x4, PCI Express x1 or PCI Express x2 interface

- JTAG support
- ACJTAG support for PCIe interface
- GPIO interface
- SPI interface to access a programmable device
- Thermal diode on chip to manage and prevent device overheating
- 196 ball HSBGA package (15 x 15 mm body, 1 mm ball pitch)

1.2 NIST Certificates

The 820x has passed the following NIST validation tests.

- 3DES: certificate #967
- AES: certificate #1414
- DSA: certificate #457
- ECDSA: certificate #181
- DRBG: certificate #53
- HMAC: certificate #834
- SHA: certificate #1284

1.3 Ordering Information

There are four speed variations and two process variations of the 820x device. The 8201 device is available for industrial applications.

Table 1-2. Part Numbers

Part Number	Process	Speed
8201CB	Commercial	0.8 Gbits/sec, 100 MBytes/sec
8201CB-F	Commercial, RoHS 6 compliant	0.8 Gbits/sec, 100 MBytes/sec
8201IB	Industrial	0.8 Gbits/sec, 100 MBytes/sec
8201IB-F	Industrial, RoHS 6 compliant	0.8 Gbits/sec, 100 MBytes/sec
8202CB	Commercial	1.6 Gbits/sec, 200 MBytes/sec
8202CB-F	Commercial, RoHS 6 compliant	1.6 Gbits/sec, 200 MBytes/sec
8203CB	Commercial	3.2 Gbits/sec, 400 MBytes/sec
8203CB-F	Commercial, RoHS 6 compliant	3.2 Gbits/sec, 400 MBytes/sec
8204CB	Commercial	6.4 Gbits/sec, 800 MBytes/sec
8204CB-F	Commercial, RoHS 6 compliant	6.4 Gbits/sec, 800 MBytes/sec
Note: The speed is based on measured AES-GCM and eLZS performance.		

2 Operation

Figure 2-1 shows a block diagram of the 820x. Each block is defined in Table 2-1.

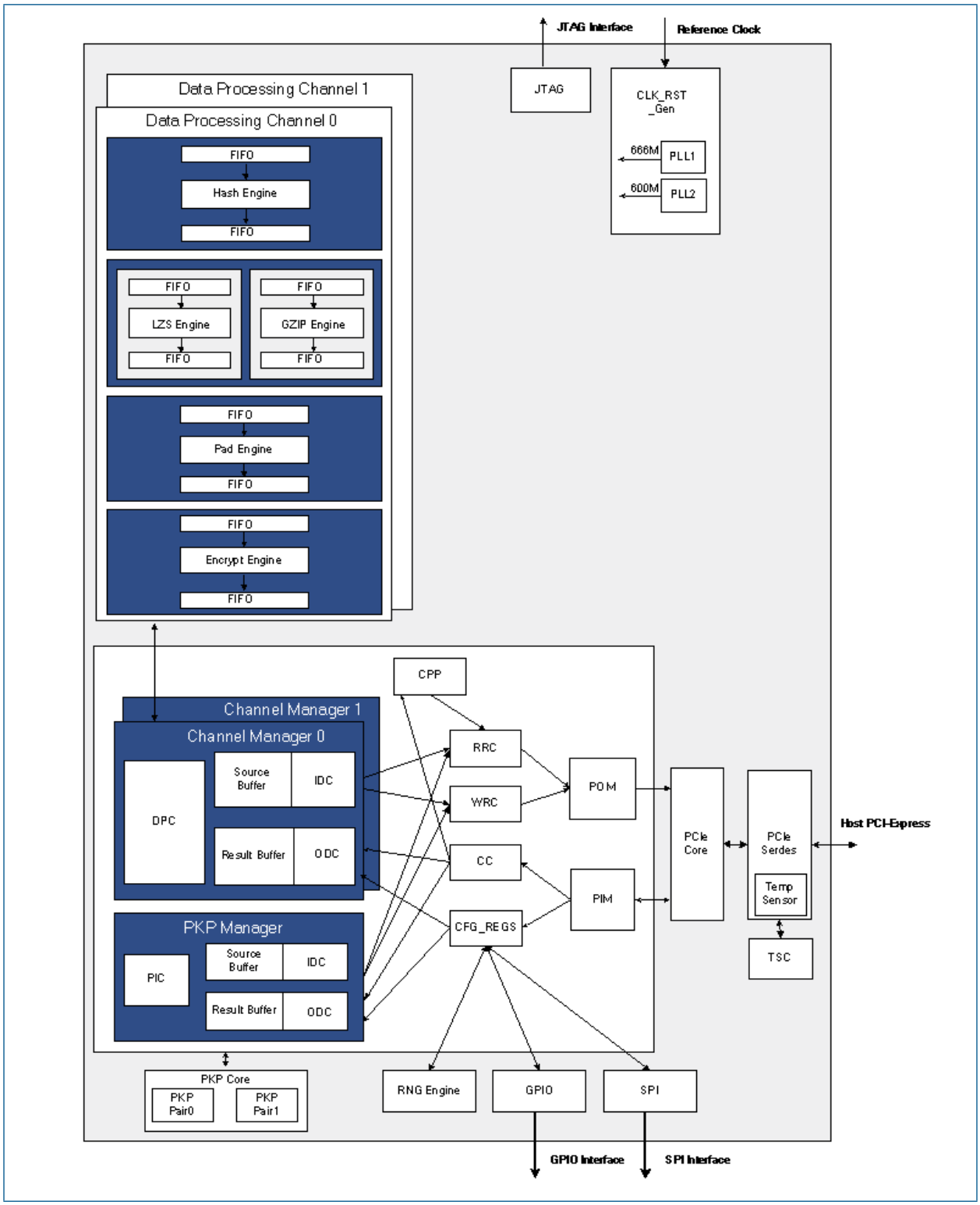


Figure 2-1. 820x Block Diagram

Table 2-1. Description of 820x Major Blocks

Block Name	Description
CC	Completion Controller Receives the completion data from the PCIe Inbound Manager, and then allocates the data to the proper Channel/PKP Manager according to the tags.
CFG_REGS	Configuration Registers
Channel Manager 0/1	Channel Manager 0 and Channel Manager 1 Controls all command processing: fetching command structures and data from host memory into the source buffer, transmitting results from the result buffer to host memory.
CLK_RST_Gen	Clock and Reset Generation Generates the clock and reset signals for all modules.
CPP	Command Pointer Ring Prefetch Prefetches the command pointer from the command ring.
Encrypt Engine	Encryption Engine Encrypts/Decrypts the data stream using the AES or 3DES algorithm.
GPIO	General Purpose IO
GZIP Engine	GZIP Engine Compresses/Decompresses the data stream using the GZIP/Deflate algorithms.
Hash Engine	Hash Engine Calculates the hash value or MAC value of the data stream.
LZS Engine	LZS Engine Compresses/Decompresses the data stream using the LZS or enhanced LZS (eLZS) algorithm.
Pad Engine	Pad Engine Adds/Removes padding data from the data stream.
PCIe Core	PCI express end point controller
PCIe Serdes	PCI express physical layer
PIM	PCIe Inbound Manager Receives the PCIe completion from the PCIe Core, and sends/receives memory read/memory write to/from the PCIe Core.
PKP Engine	Public Key Processor Engine The PKP engine consists of two pairs of public key processors.
PKP Manager	PKP Manager Controls instruction and operand data fetching from host memory to the PKP engine, and transmits the calculated result from the PKP engine to host memory.
POM	PCIe Outbound Manager Sends the PCIe write data and read requests to the PCIe Core.
RNG	Random Number Generator
RRC	Read Request Controller Arbitrates read requests from the CPP, Channel Manager 0, Channel Manager 1 and PKP Manager, and sends read requests to the PCIe Outbound Manager.

Table 2-1. Description of 820x Major Blocks

Block Name	Description
SPI	SPI interface that connects to an external programmable device
TSC	Temperature Sensor Controller Provides host software access to the temperature sensor parameters.
WRC	Write Request Controller Arbitrates the write requests from Channel Manager 0, Channel Manager 1 and PKP Manager, and sends the write requests to the PCIe Outbound Manager.

2.1 Data Integrity

The 820x provides robust data integrity with a combination of Error Code Correction (ECC), CRC, and Real Time Verification (RTV). These features combine to create strong data protection for encode and decode operations.

2.1.1 ECC & Parity Protection

All data in RAM is protected with 6-bit Error Code Correction (ECC6), 8-bit Error Code Correction (ECC8) and parity.

- ECC6 is used to protect 16 bit width RAM
- ECC8 is used to protect 64 bit width RAM
- Parity is used to protect the PCIe Core RAMs

ECC6 and ECC8 provide detection of all single, double, and triple bit errors.

2.1.2 CRC Protection

The host may configure the PCIe configuration register to either enable/disable ECRC protection. If enabled, the PCIe Core will generate the ECRC on the fly to protect data on the PCIe bus.

Data CRC can be generated by the host or by the 820x, depending on the configuration settings in the command structure. Please refer to the "CRC_EN" field of the Command Structure for more information (Section 3.1.2.2). The CRC will be verified by the 820x, and may be sent to the host or stripped from the data stream.

2.1.3 Real Time Verification

The Compression, Encryption and Hash engines contain internal real time verification logic that ensures all transforms are completed successfully and any detected errors are reported prior to the command completing.

Real time verification may be enabled or disabled via software. If enabled, real time verification will reduce the performance of small packet (< 2K bytes) LZS and GZIP/Deflate compression operations. Enabling real time verification will slightly increase the 820x power consumption (< 0.05W).

2.1.3.1 Compression Engine Real Time Verification

The Compression engine contains one compression core and one decompression core for LZS and GZIP/Deflate compression operations. Data for compression operations in the encode direction will be automatically decompressed to verify that the decompressed CRC matches the original CRC of the raw data. The Compression engine also employs an ECC in its FIFO.

For compression operations in the encode direction, raw data with an embedded CRC is compressed by the compression core. The compressed data is then decompressed to verify that the decompressed CRC matches the original input CRC. If the CRCs do not match, the Compression engine reports a CRC error to the host. To add further real time verification, a 6-bit ECC is added to the compressed data before it is written to the destination FIFO. This ECC is verified by the LZS Engine when reading data out from the destination FIFO.

For compression operations in the decode direction, the compressed data with an embedded CRC is decompressed in the decompression core. The decompressed CRC is verified by the decompression core, and if an error is detected it will be reported to the host. If CRC32 is not enabled, this CRC check is also disabled.

Figure 2-2 illustrates LZS real time verification in the Compression engine for encode and decode operations. In the following figures, a CRC in parenthesis, as in RAW (CRC), signifies that the CRC is embedded into the data. A CRC that is added to a value, as in RAW + CRC, signifies that the CRC is in a header, separate from the data.

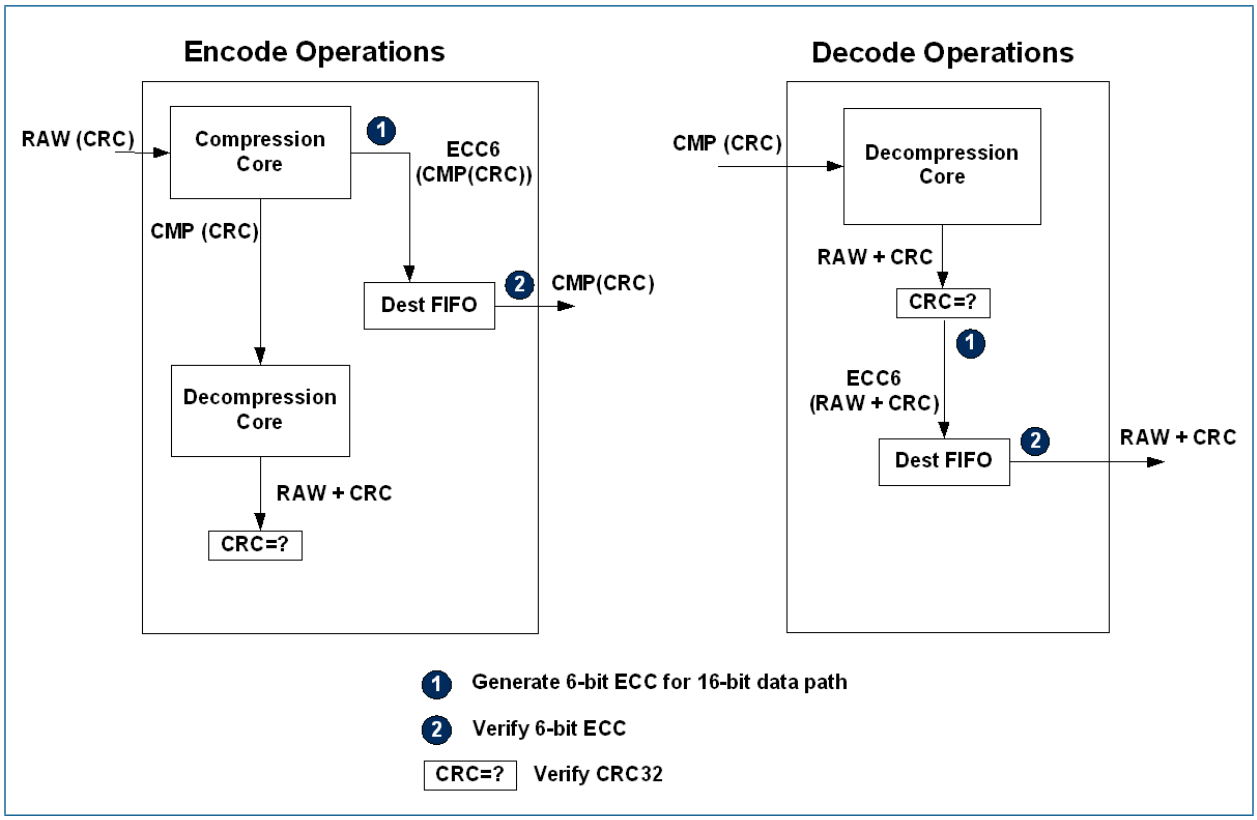


Figure 2-2. Compression Engine LZS Real Time Verification

Figure 2-3 illustrates GZIP/Deflate real time verification in the Compression engine. The GZIP/Deflate real time verification is similar to the LZS real-time verification except that the Compression engine does not generate the ECC.

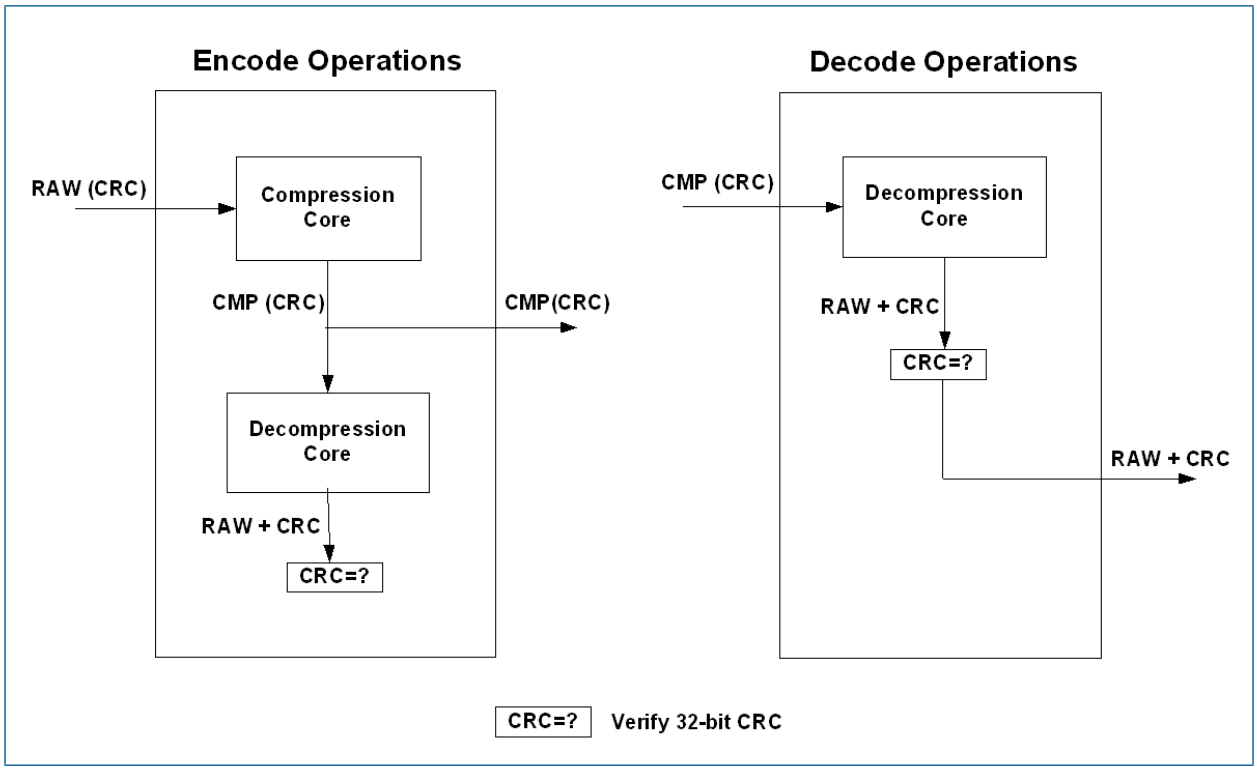


Figure 2-3. Compression Engine GZIP/Deflate Real Time Verification

2.1.3.2 Encryption Engine Real Time Verification

The Encryption engine contains two encryption cores and two decryption cores for AES and 3DES operations. The Encryption engine performs real time verification for both encode and decode operations by having both cores encrypt or decrypt the input data simultaneously. The results from both cores are compared and if an error is detected it will be reported to the host.

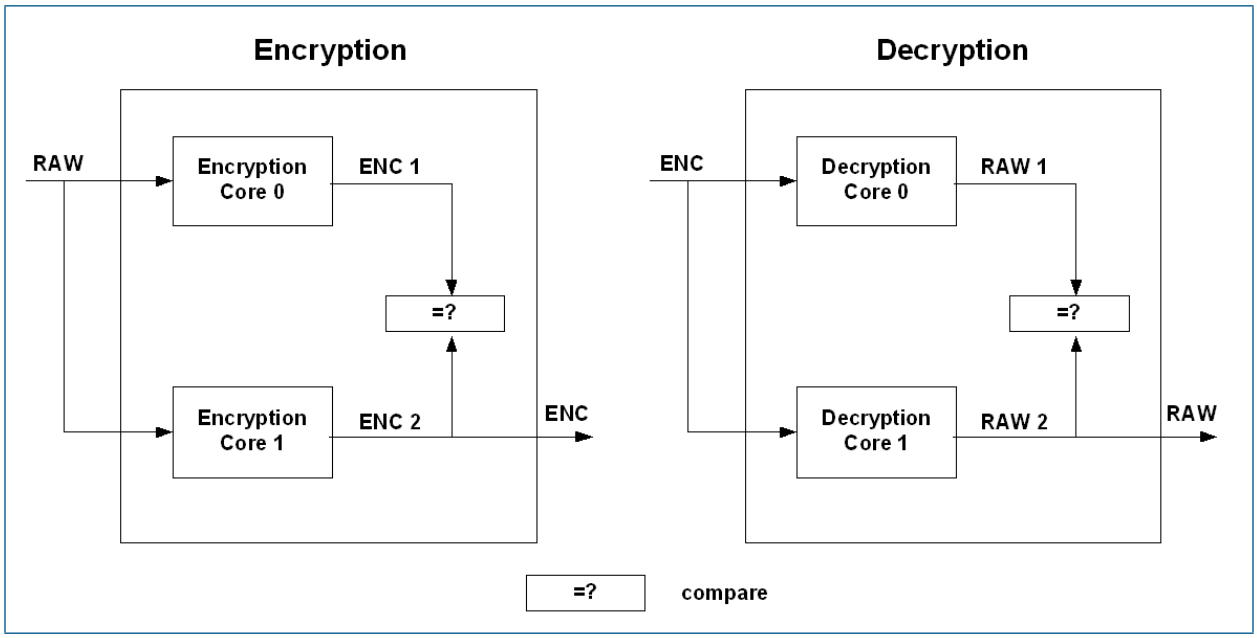


Figure 2-4. Encryption Engine Real Time Verification

2.1.3.3 Hash Engine Real Time Verification

The Hash engine contains two hash cores to implement real time verification of the data. For single File hash and Slice hash operations, both Hash cores calculate the hash/MAC simultaneously and the two results are compared. If an error is detected it will be reported to the host. For "File hash + Slice hash" operations, one core calculates the "File hash" and the other core calculates the "Slice Hash". Figure 2-5 shows the real time verification for File hash and Slice hash operations.

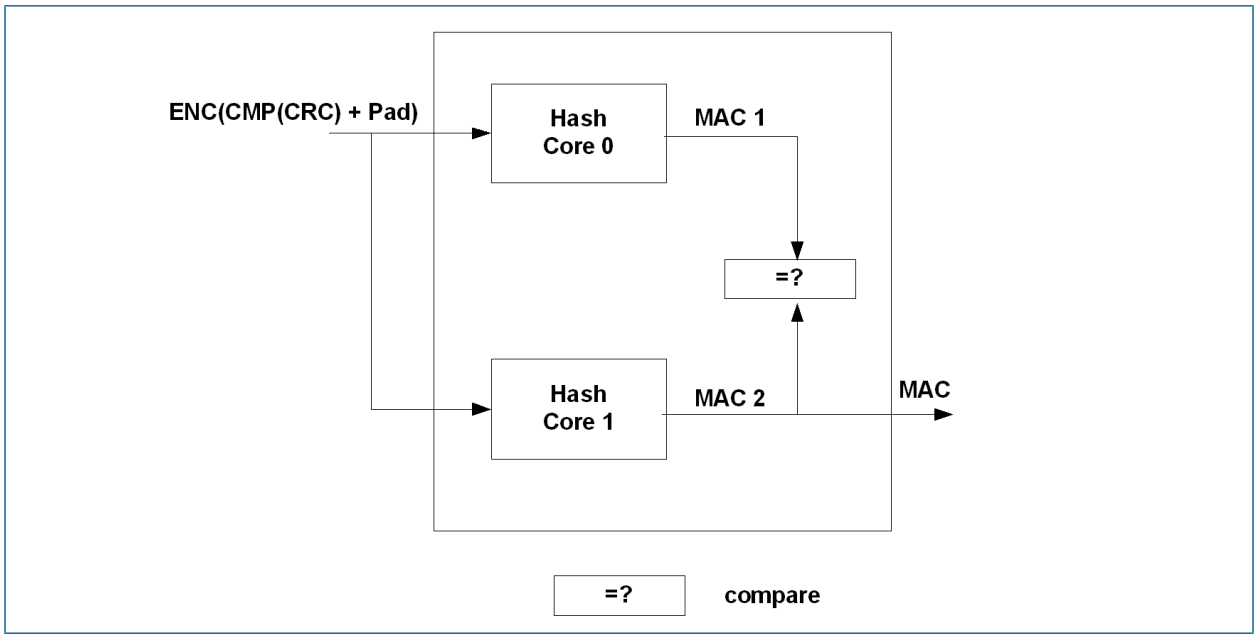


Figure 2-5. Hash Engine Real Time Verification

2.1.4 Data Integrity Model for Encode Operations

This section describes the data integrity model for encode operations using the parity, CRC, ECC and engine real time verification features described in the previous sections. Figure 2-6 illustrates the model described below.

Raw data with a PCIe standard CRC (ECRC) from the host enters the 820x over the PCIe bus through the 820x PCIe core. The PCIe core verifies the ECRC and generates parity before writing the data into RAM and into the PCIe Inbound Manager (PIM). The PIM verifies the parity and generates an 8-bit Error Correction Code (ECC8) and enters the data plus the ECC8 into the source buffer. When a Channel Manager reads the data from the source buffer, it first verifies the ECC, generates a CRC, and sends the raw data and CRC to the Compression engine. The Compression engine compresses the data and CRC, performs real time verification by decompressing the data and verifying the CRC, and sends the data to the Pad engine. The Pad engine adds padding to the data stream, and sends the padded data to the encryption engine (the Pad engine does not perform any real time verification). The Encryption engine encrypts the padded compressed data, performs real time verification, and sends the result to the Hash engine. The Hash engine calculates the MAC and performs real time verification on the hash result.

An 8-bit ECC is added to the output data of the Encryption engine and written into the result buffer. The Channel Manager reads the data from the result buffer, verifies the ECC, and sends the data to the PCIe Outbound Manager (POM). The POM adds parity before writing the data into the PCIe Core RAM. When data is read from the PCIe Core RAM by the PIM, the parity is first verified and an ECRC is generated before the final transformed data is sent to the host.

The example shown in Figure 2-6 where the 820x generates the embedded CRC rather than the host supplying it is the most popular configuration.

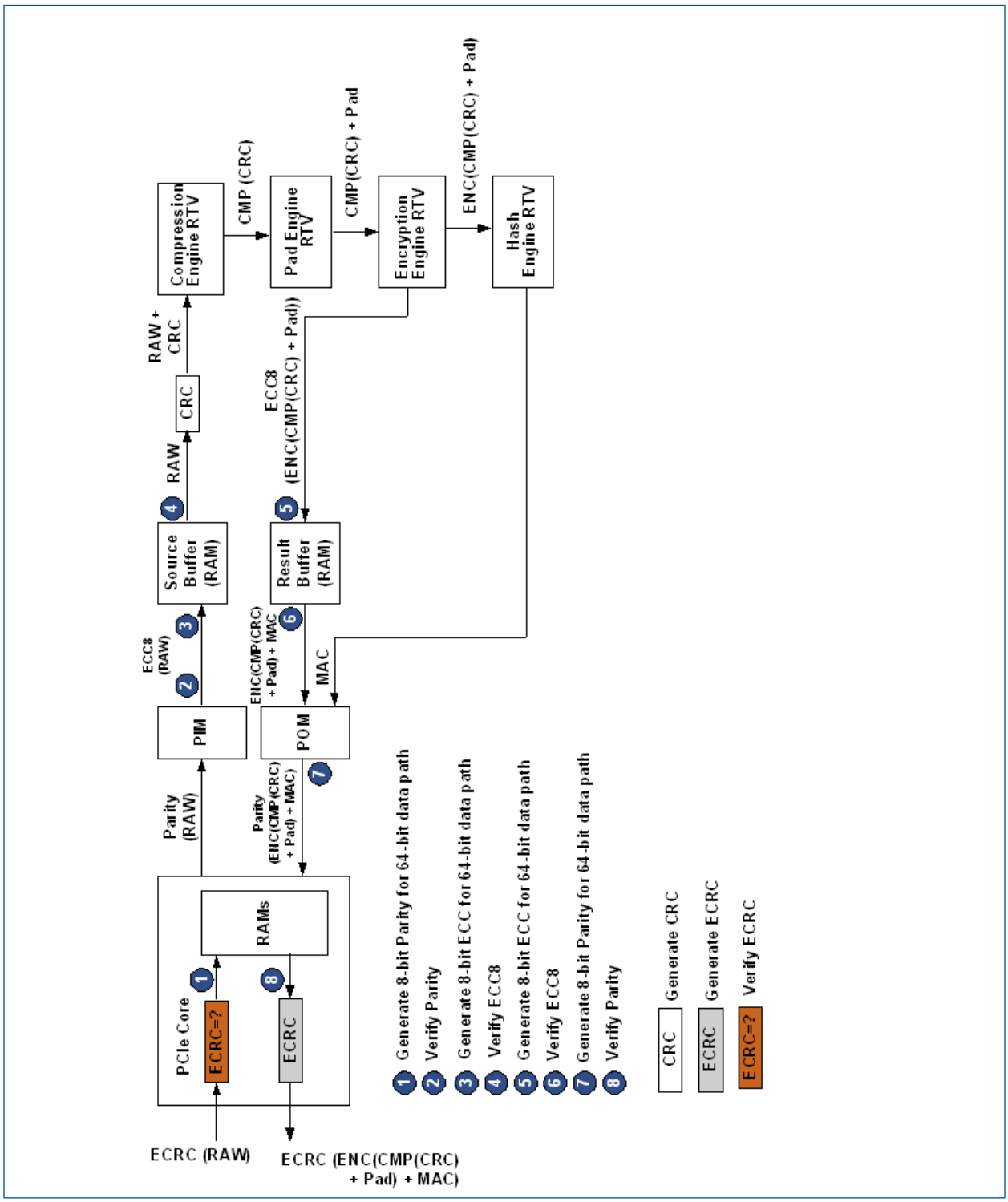


Figure 2-6. Encode Operation Real Time Verification

2.1.5 Data Integrity Model for Decode Operations

This section describes the data integrity model for decode operations using the parity, CRC, ECC and engine real time verification features described in the previous sections. Figure 2-7 illustrates the model described below.

Compressed, encrypted data with an embedded hash, and MAC arrive from the host over the PCIe bus with the standard PCIe CRC (ECRC) and enters the 820x through the 820x PCIe core. The PCIe core verifies the ECRC and generates parity before writing the data into RAM and into the PCIe Inbound Manager (PIM). The PIM verifies the parity and generates an 8-bit Error Correction Code (ECC8) and enters the data plus the ECC8 into the source buffer. When a Channel Manager reads the data from the source buffer, it first verifies the ECC, and the sends the encrypted, compression data with padding to the Hash engine and the Encryption engine. The Hash engine uses the embedded MAC and performs real time verification on the hash result. The Encryption engine decrypts the encrypted data and sends the padded compressed data result to the Pad engine. The Pad engine removes the padding from the data stream, and sends the compressed data to the compression engine (the Pad engine does not perform any real time verification). The compression engine decompresses the data and CRC, performs real time verification.

The CRC output from the compression engine is compared to the raw CRC, and, if verified, stripped from the raw data. An 8-bit ECC is added to the raw data and written into the result buffer. The Channel Manager reads the data from the result buffer, verifies the ECC, and sends the data to the PCIe Outbound Manager (POM). The POM adds parity before writing the data into the PCIe Core RAM. When data is read from the PCIe Core RAM by the PIM, the parity is first verified and an ECRC is generated before the final raw data is sent to the host.

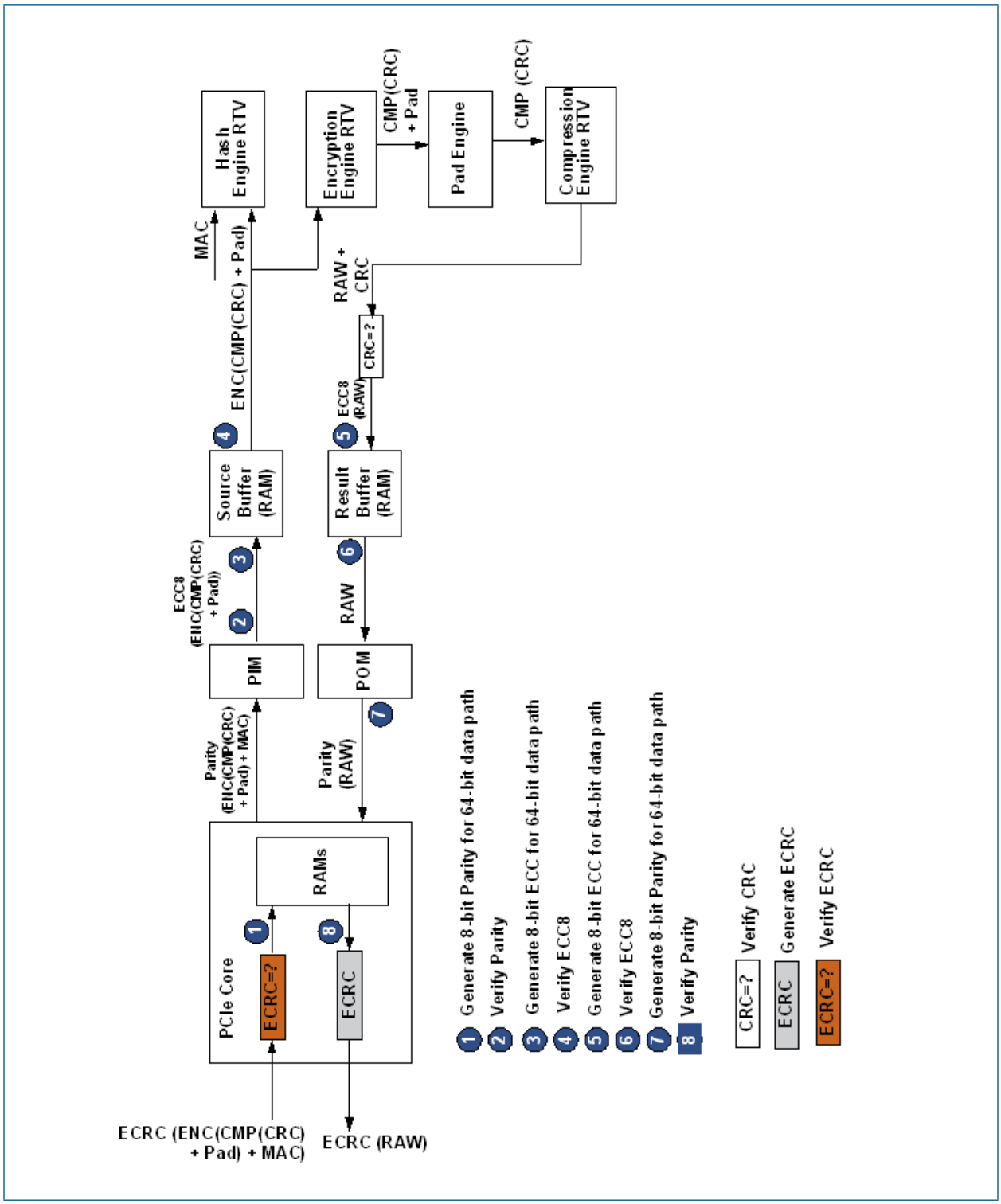


Figure 2-7. Decode Operation Real Time Verification

2.2 Clock Domains

The 820x logic operates with six main clock domains:

- PCIe Serdes, PCIe Core, DMA, Probe, Pad engine and JTAG (125MHz from PCIe Serdes PLL)
- Hash Engine (278MHz from CLK_RST_Gen)
- LZS Engine interface logic and Encrypt Engine (208MHz from CLK_RST_Gen)
- LZS Engine Core Logic (417MHz from CLK_RST_Gen)
- GZIP Engine (250MHz from CLK_RST_Gen)
- PK Core (375MHz from CLK_RST_Gen)

2.3 Clock Gating

Exar's 820x supports aggressive static and dynamic clock gating.

Static clock gating will disable the clock to the Channel Managers, Data Process Channels, PKP Manager, PKP Engine, and RNG if the corresponding register enable bits are set to zero. If the software disables a Channel Manager, the clock to all modules in this channel will also be disabled.

Dynamic clock gating will disable the clock for all unused engines in Data Process Channels for each command. For example, if one command only compresses data, the clock of the Pad Engine, Hash Engine and Encrypt Engine will be gated automatically.

The shaded regions in Figure 2-8 show which 820x blocks employ clock gating.

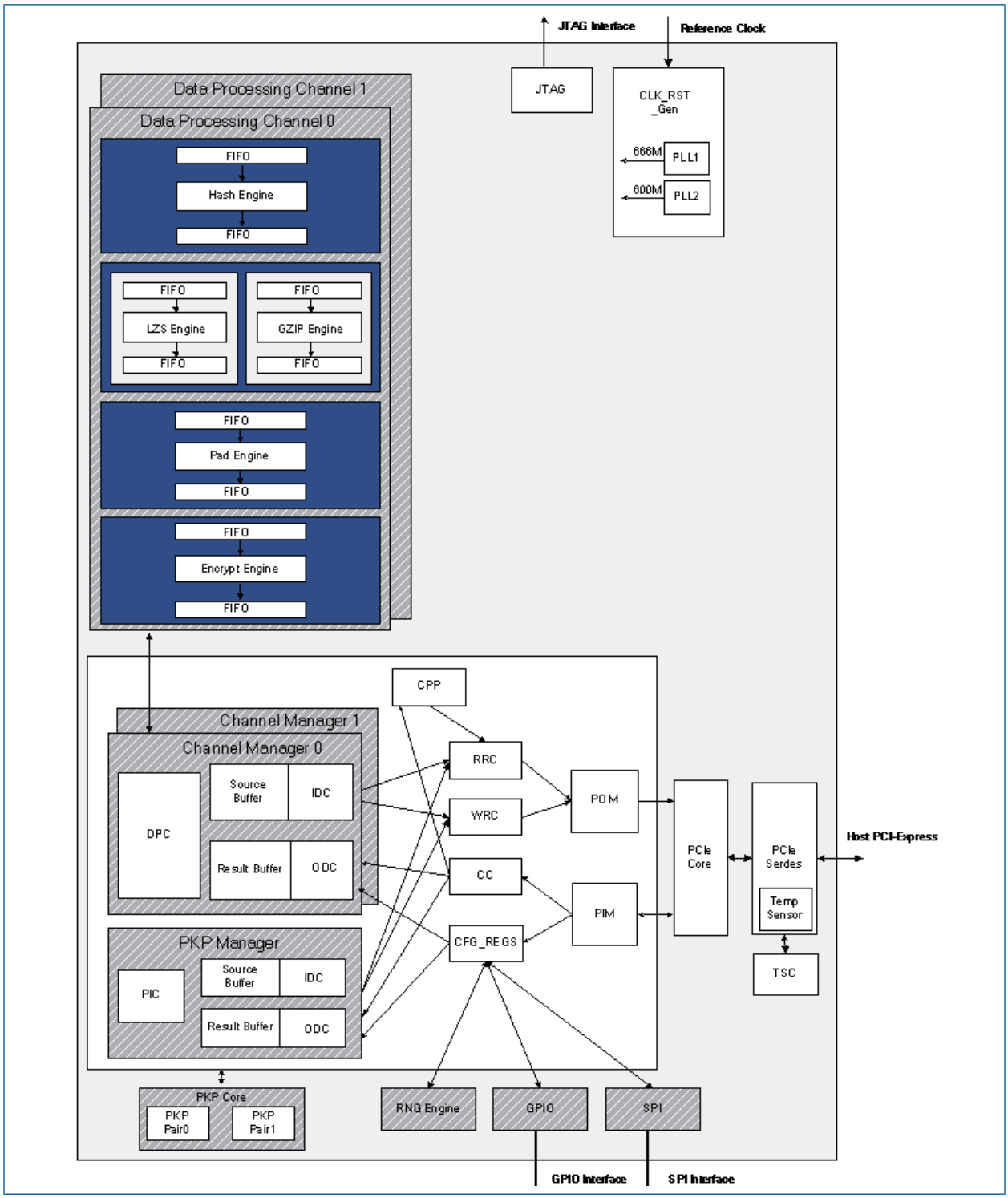


Figure 2-8. 820x Modules that can be Clock Gated

3 Data Structures

This section describes the 820x data structures for the command pointer ring, result ring, free pool ring, and the command structure format.

3.1 Command Pointer Ring

The command pointer ring is a linear array of command pointers which point to the actual command structures. The 820x uses only indirect command addressing to execute commands. By using indirect command addressing, the commands themselves may be located anywhere in host memory but must be 512-byte aligned.

The host software must maintain the command pointer ring and ensure that the command pointer ring base address is 8-byte aligned. The entries in the command pointer ring should be entered in consecutive order.

In 32-bit addressing mode, every command pointer is 4 bytes; in 64-bit addressing mode, every command pointer is 8 bytes.

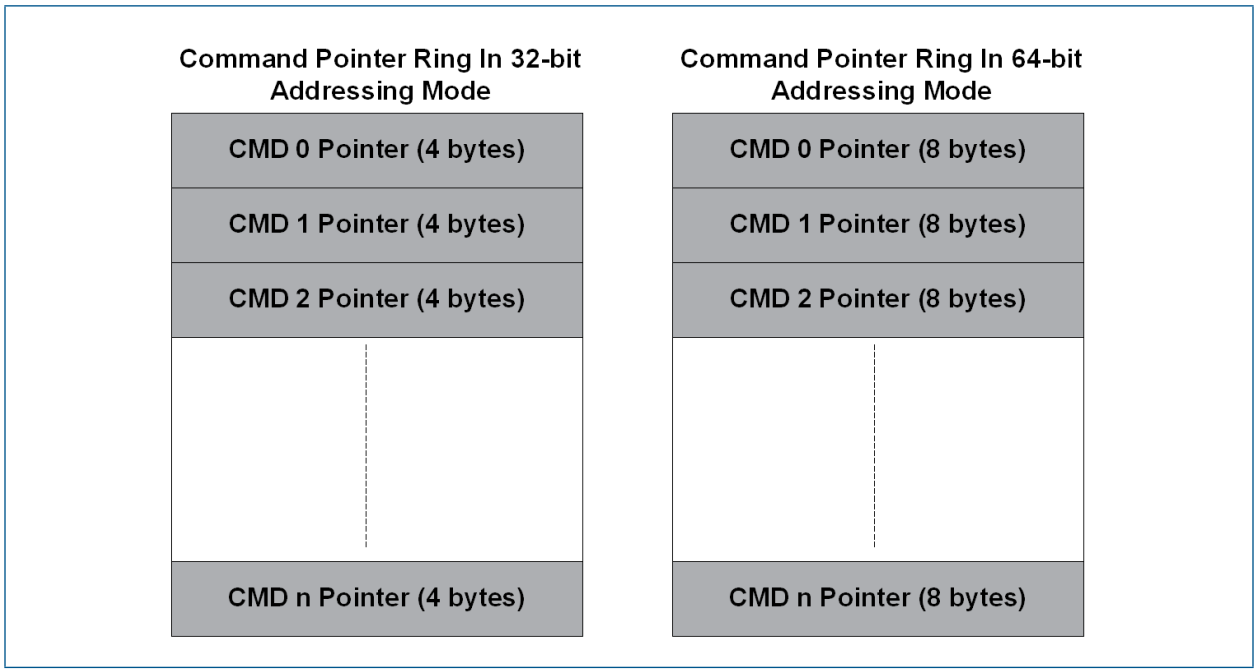


Figure 3-1. Command Pointer Ring Format

The maximum number of command pointers in the command pointer ring is configurable by the host using the DMA control registers, specifically the DMA Configuration Register (see Section 6.2.14, "DMA Configuration Register"). After the host software determines the memory location of the command pointer ring and command structure, it should update the 820x related DMA control registers. The 820x will fetch the command pointer, and then fetch the command structure. The detailed command execution flow is described in Section 3.3, "Command Operation Sequence".

Figure 3-2 illustrates a typical command pointer ring structure. The starting address of each command structure must fall on a 512-byte boundary, regardless of the number of descriptor pairs for each command structure.

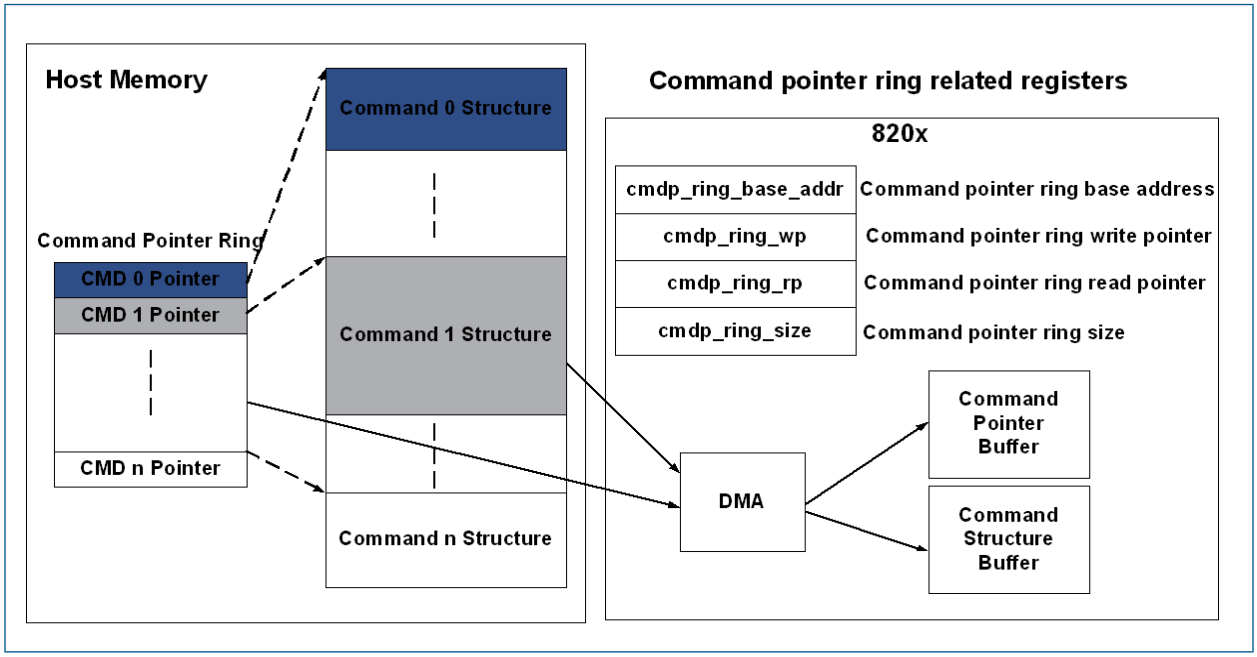


Figure 3-2. Command Pointer Ring

The 820x supports either a single or dual command ring mode, and the Express DX SDK uses the dual mode. In dual command ring mode, the host maintains two command pointer rings and two result rings are maintained in host memory. The Express DX Software Development Kit will place all higher priority commands in one command ring and the lower priority commands into the other command ring. However, since the 820x fetches commands from both command rings using a round-robin scheme with equal priority, there is no assurance that the high priority ring will execute commands faster than the low priority ring. If the host can assure that there are fewer high priority commands than low priority commands, a priority scheme can be supported.

The two command pointer rings and result rings have the same structures and configuration as shown in Figure 3-3.

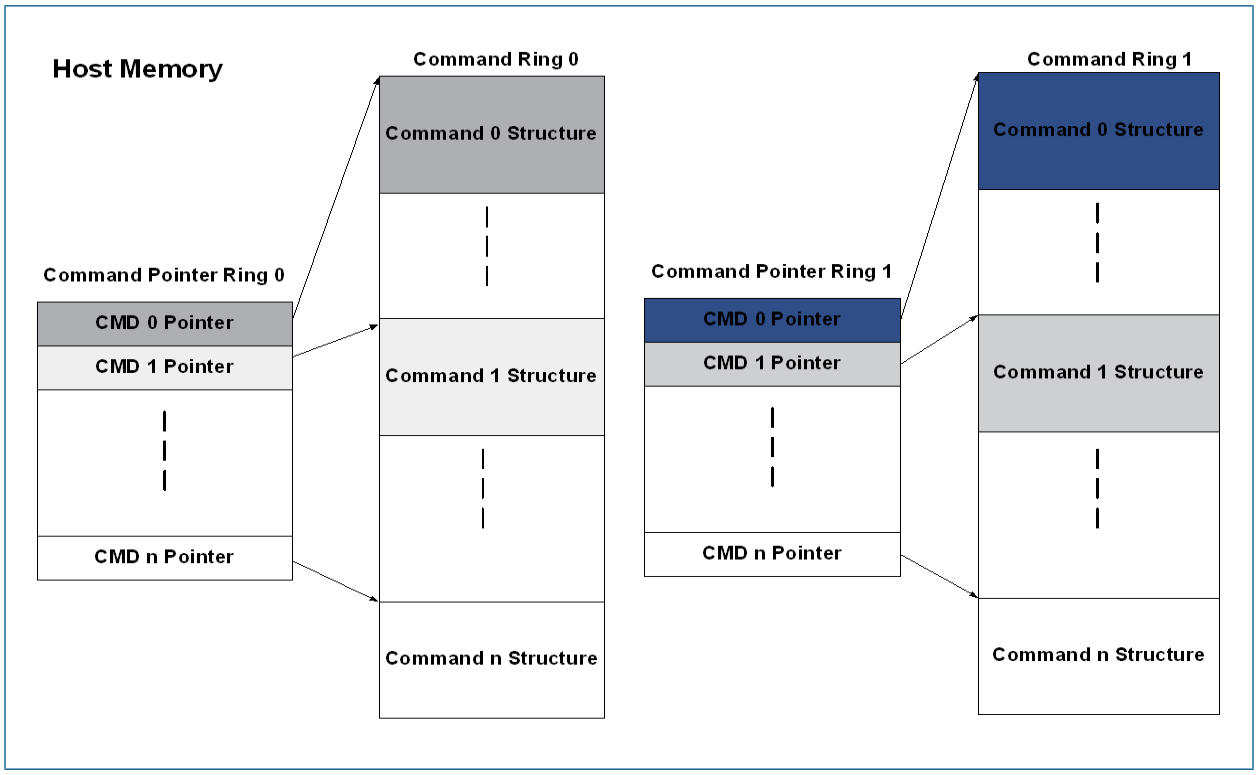


Figure 3-3. Dual Command Ring Mode

Command rings are implemented in PCIe host memory. The host writes a complete command structure as an entry in the command pointer ring. The 820x reads the command structure and interprets its various fields to perform the appropriate operations. Once the 820x has completed executing a command, it updates the appropriate entries within the corresponding command structure. The host then reads the entries in the command structure that have been updated by the 820x to fetch the results.

The 820x does not maintain a "full" bit for the command pointer ring; it is the responsibility of the host not to overflow the command ring.

3.1.1 Command Structure

The host must ensure that the starting address of all command structure entries are 512-byte aligned. This constraint will avoid the PCIe 4K boundary read request limitation because the 820x maximum command read request is 512 bytes.

The 820x supports two command structure modes: normal mode and small packet mode. The command structure mode is defined in the Desc_cmd_base descriptor. In normal mode, the data may be any size. In small packet mode, the size is reduced to improve small packet performance. The performance improvement of small packet mode is achieved by removing the data read request latency of 2000ns ~ 8000ns.

A normal mode command structure consists of a linear array of descriptors. The size of one command structure is only limited by the amount of host contiguous physical memory. Each command will have a unique command structure.

In normal mode, the command structure includes one result descriptor, one base command descriptor, one compression command descriptor, one encryption command descriptor, one hash command descriptor, one pad command descriptor, and several pairs of source and destination descriptors.

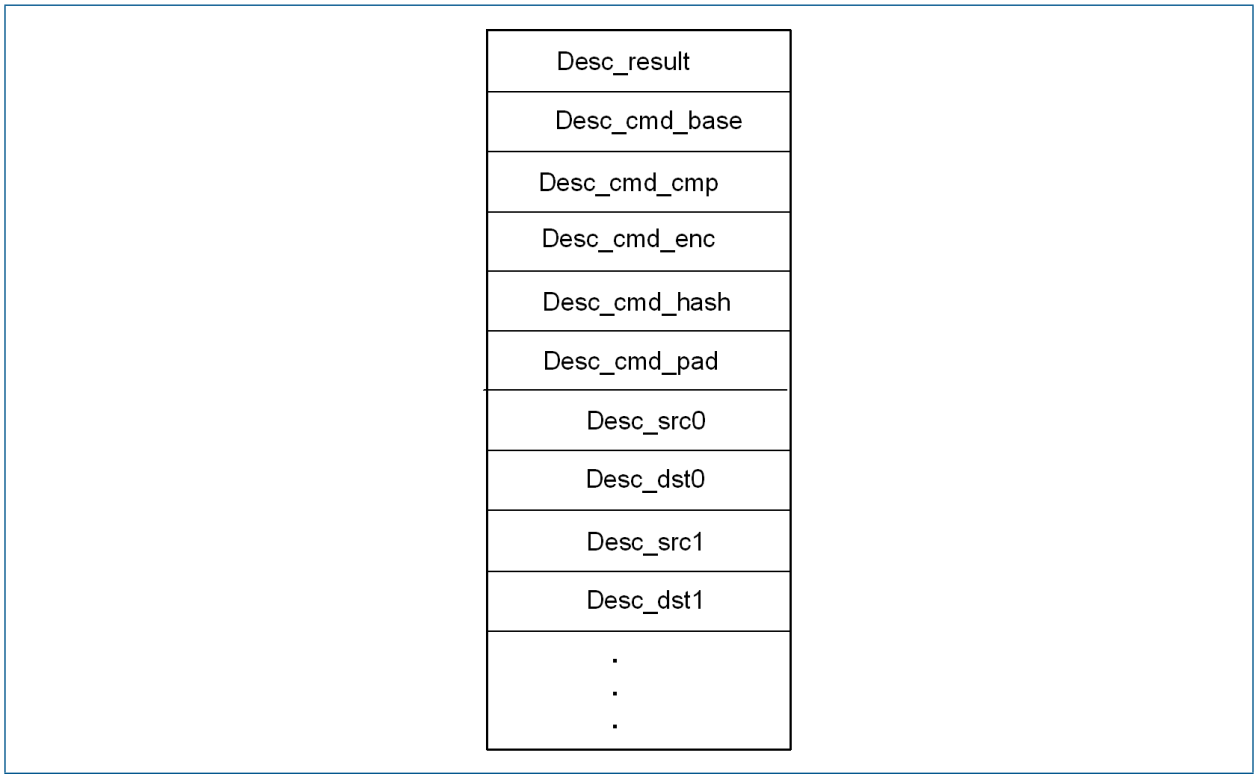


Figure 3-4. Normal Mode Command Structure

A small packet mode command structure includes one result descriptor, one base command descriptor, one compression command descriptor, one encryption command descriptor, one hash command descriptor, one pad command descriptor, two destination descriptors and data. The data field includes information fields and the source data.

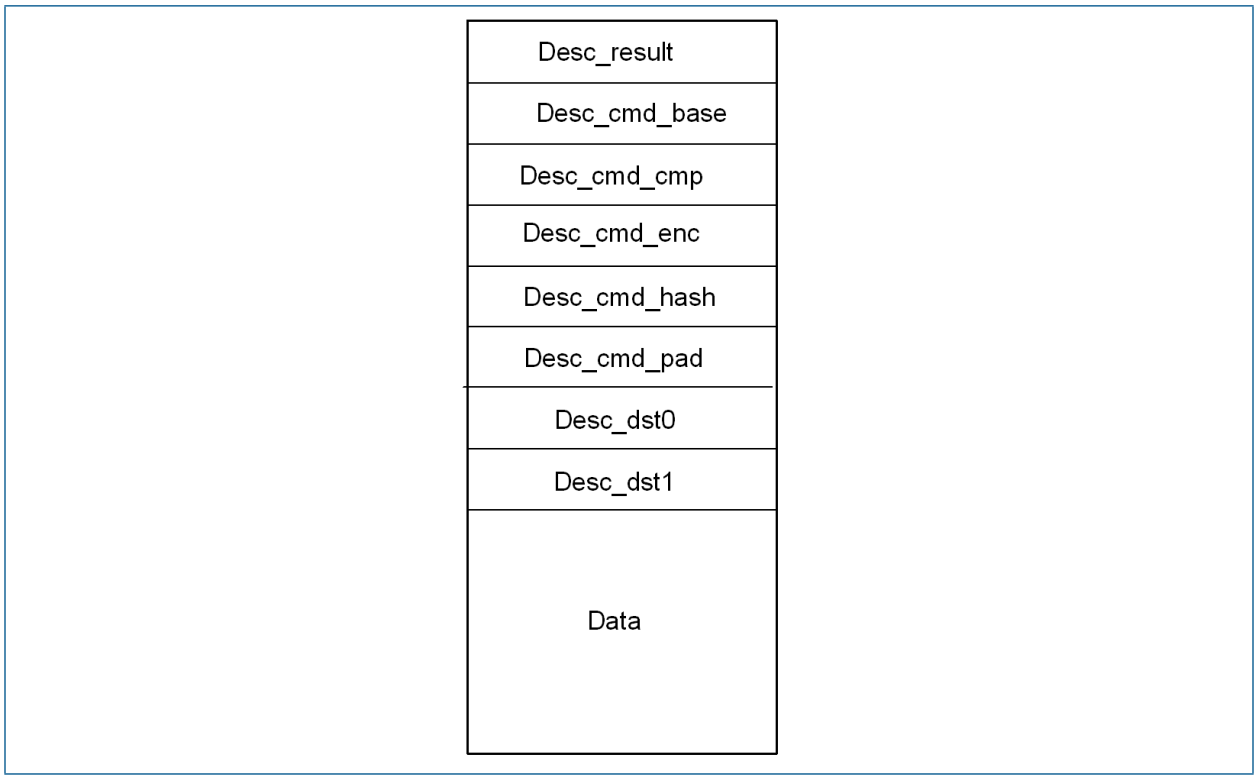


Figure 3-5. Small Packet Mode Command Structure

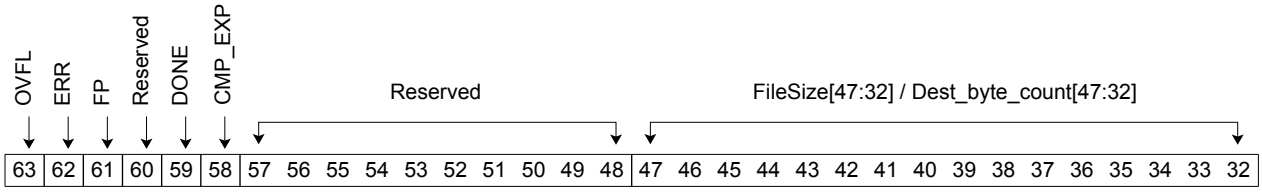
3.1.2 Command Descriptor Format

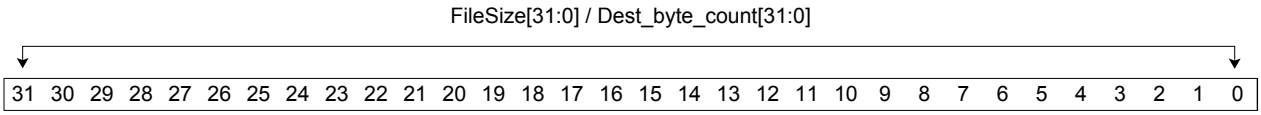
Descriptors are the building blocks of the 820x command structure.

Note: For all descriptors, reserved bits must be written as zeroes. Descriptors for unused engines must be filled with zeroes.

3.1.2.1 Desc_result

The result descriptor holds the command result. After the 820x completes a command, it will write to this descriptor. The Desc_result format is the same (8 bytes) for either 32-bit or 64-bit addressing modes.



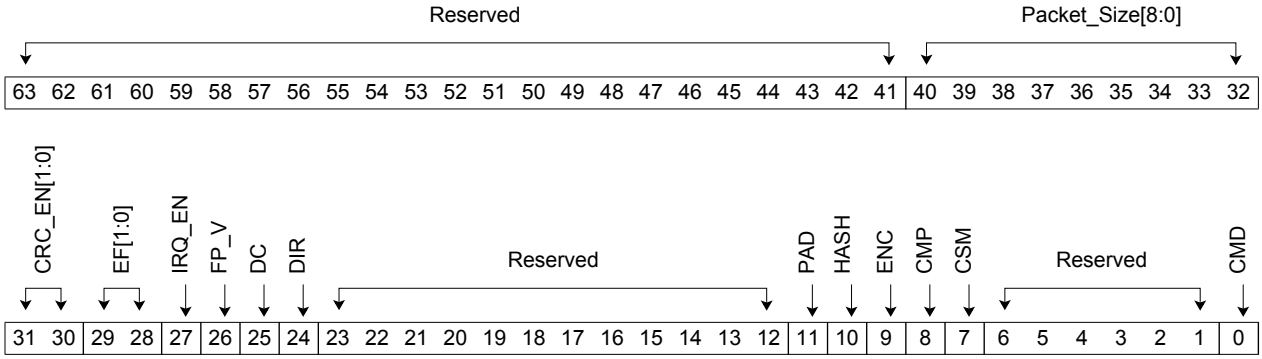


Field Name	Bits	Default	Description
OVFL	63	0	Overflow bit. The 820x will set this bit if the destination data size exceeded the total destination buffer space. This bit will not be set if the Free Pool is being used. 0 No destination buffer overflow occurred 1 Destination buffer overflow occurred
ERR	62	0	Error bit. The 820x will set this bit if any type of error occurred during command processing. 0 No error occurred 1 Error occurred
FP	61	0	Free Pool bit. If all destination buffers are consumed and the FP_V bit in the Desc_cmd_base is set, the 820x will set this bit to indicate it is using the Free Pool to store the result data. Once the 820x uses the Free Pool, the 820x will not longer set the OVFL bit. 0 820x is not using the free pool 1 820x is using the free pool
Reserved	60	0	Reserved
DONE	59	0	Done bit. This bit is written to and read by both the host and 820x. The host software must set this bit to one after the command structure is written. The 820x will write a zero to this but when it has completed a command. 0 820x has completed the command 1 Host has set up the command for the 820x
CMP_EXP	58	0	Compression Expansion bit. This bit defines how the compression expansion is calculated. 0 Len(cmp) + Head_size < Len(RAW) 1 Len(cmp) + Head_Size >= Len(raw)
Reserved	57:48	0	Reserved

Field Name	Bits	Default	Description
File_Size[47:0] / Dest_byte_count[47:0]	47:0	0	File Size / Destination Data Byte Count. This field contains the File Size or Destination Data Byte Count used to calculate the hash or MAC value of the last block of source data. The data in the source buffer includes information fields and raw data. The File size indicates the size of the raw data, excluding the information fields. This field is written to and read by both the host and the 820x. The host must write the file size to this field for file hash, AES-GCM, or MAC length padding operations. The 820x will overwrite this field with the Dest_byte_count[47:0] after the command completes.

3.1.2.2 Desc_cmd_base

This descriptor defines the command parameters that are common to all types of commands. The Desc_cmd_base format is 8 bytes for either 32-bit or 64-bit addressing modes.



Field Name	Bits	Default	Description
Reserved	63:41	0	Reserved

Field Name	Bits	Default	Description
Packet_size[8:0]	40:32	0	Packet Size. This field is only valid when Command Structure Mode, CSM, is set to small packet mode. The packet size will vary with the command addressing mode (32 or 64-bit addressing) and the size of the information field (see Section 3.1.3.5). This field indicates the real source data size, not including the information and descriptor fields. Note, for small packet mode, the total command size must be less than or equal to 512 bytes. The packet size range in bytes is: 1 - maximum_packet_size where Maximum_packet_size = 512 - (info field) - (descriptor field) | Total cmd structure size must be <= 512 Desc field | Info field | Source data | packet_size[8:0] |

Field Name	Bits	Default	Description
CRC_EN[1:0]	31:30	0	CRC Enable. The CRC enable bit is only valid if the disable count flag, DC, is set to one. The behavior of this bit depends if the command is for AES-XTS. For all commands except AES-XTS: 00 The input data does not contain a CRC; the 820x will not append or verify the CRC 01 For an encode command, the input data contains 4 bytes of CRC; the 820X will verify the CRC. For a decode command, the 820X will verify the CRC after decoding and send the data and the 4 bytes of CRC to the host. 10 For an encode command, the input data does not contain a CRC; the 820x will append and verify 4 bytes of CRC. For a decode command, the 820x will verify the CRC after decoding, strip the 4 bytes of CRC, and send only the raw data to the host 11 Reserved. For AES-XTS commands: 00 The input data does not contain a CRC; the 820x will not append or verify the CRC 01 For both encode or decode operations, the 820x will verify the source data CRC and update the CRC field of the DIF. 1x Reserved. DIF Format: 4 bytes / 2 bytes / 2 bytes Ref tag | Meta tag | CRC For a more detailed description of the CRC_EN bit's behavior for all commands except AES-XTS, please refer to Table 3-1.
EF[1:0]	29:28	0	Endian Format. Sets the data endian format. Refer to Figure 3-6 for an illustration of the swap options. 00 no swap 01 byte swap within word 10 word swap, no byte swap within word 11 word swap and byte swap within word
IRQ_EN	27	0	Interrupt Enable. If set, the 820x will interrupt the host when this command completes. 0 820x will not interrupt the host when this command completes 1 820x will interrupt the host when this command completes

Field Name	Bits	Default	Description
FP_V	26	0	Free Pool Valid. This bit is written to by the host to indicate if the 820x may use the Free Pool Note, if the Free Pool is not used and the destination buffers are not large enough to accommodate the resulting data, the 820x will set the OVFL bit in Desc_result descriptor. 0 Do not allow the 820x to use the free pool 1 Permit 820x to use the free pool
DC	25	0	Disable Count. If this bit is set, the 820x will process all raw data sent to it by the host and disregard the head and tail count values. If set, the host software does not need to set values for the source count and head counters; the 820x will ignore these fields. 0 All source and head counts in successive descriptors are valid. 1 Disable all source and head counts in successive descriptors
DIR	24	0	Disable Information Routing. Refer to Section 3.1.3.5, "Data Stream Information Fields" for more information. 0 820x will send the information fields to the host destination buffer for data integrity verification 1 820x will not send the information fields to the host destination buffer
Reserved	23:12	0	Reserved.
PAD	11	0	Pad Engine Enable/Disable. If disabled, data will pass through the Pad engine unaltered. 0 Disable the Pad engine 1 Enable the Pad engine
HASH	10	0	Hash Engine Enable/Disable. If disabled, data will pass through the Hash engine unaltered. 0 Disable the Hash engine 1 Enable the Hash engine
ENC	9	0	Encryption Engine Enable/Disable. If disabled, data will pass through the Encryption engine unaltered. 0 Disable the Encryption engine 1 Enable the Encryption engine
CMP	8	0	Compression Engine Enable/Disable. If disabled, data will pass through the Compression engine unaltered. 0 Disable the Compression engine 1 Enable the Compression engine

Field Name	Bits	Default	Description
CSM	7	0	Command Structure Mode. This bit sets the command structure mode to either normal or small packet mode. Refer to Section 3.1.1 for more information. 0 Normal mode 1 Small packet mode
Reserved	6:1	0	Reserved
CMD	0	0	Command Type. Indicates the command type: 0 Encode For compression, encryption and hash operations 1 Decode For decompression and decryption operations, and authentication using the hash engine to verify the MAC Note that the 820x supports stateful encryption and hash/MAC operations, but does not support stateful compression operations. The host software must not submit a compression operation when performing a stateful MAC operation.

Table 3-1. CRC Enable Behavior for all Commands except AES-XTS

CRC_EN Value	Operation	Host input to 820x	820x Output
00	N/A	Data	Data 820x does not verify or append a CRC
01	Encode	Data | 4 bytes CRC	Compressed (Data + CRC) 820x verifies CRC and compresses input data
	Decode	Compressed (Data + CRC)	Data | 4 bytes CRC 820x decompresses input data and verifies CRC

Table 3-1. CRC Enable Behavior for all Commands except AES-XTS

CRC_EN Value	Operation	Host input to 820x	820x Output
10	Encode	Data	Data | 4 bytes CRC ↓ Compressed (Data + CRC) 820x adds CRC and then compresses the input data and CRC
	Decode	Compressed (Data + CRC)	Data | 4 bytes CRC ↓ Data 820x decompresses input data, verifies CRC, and then removes CRC
11	Encode	Reserved	Reserved
	Decode	Reserved	Reserved

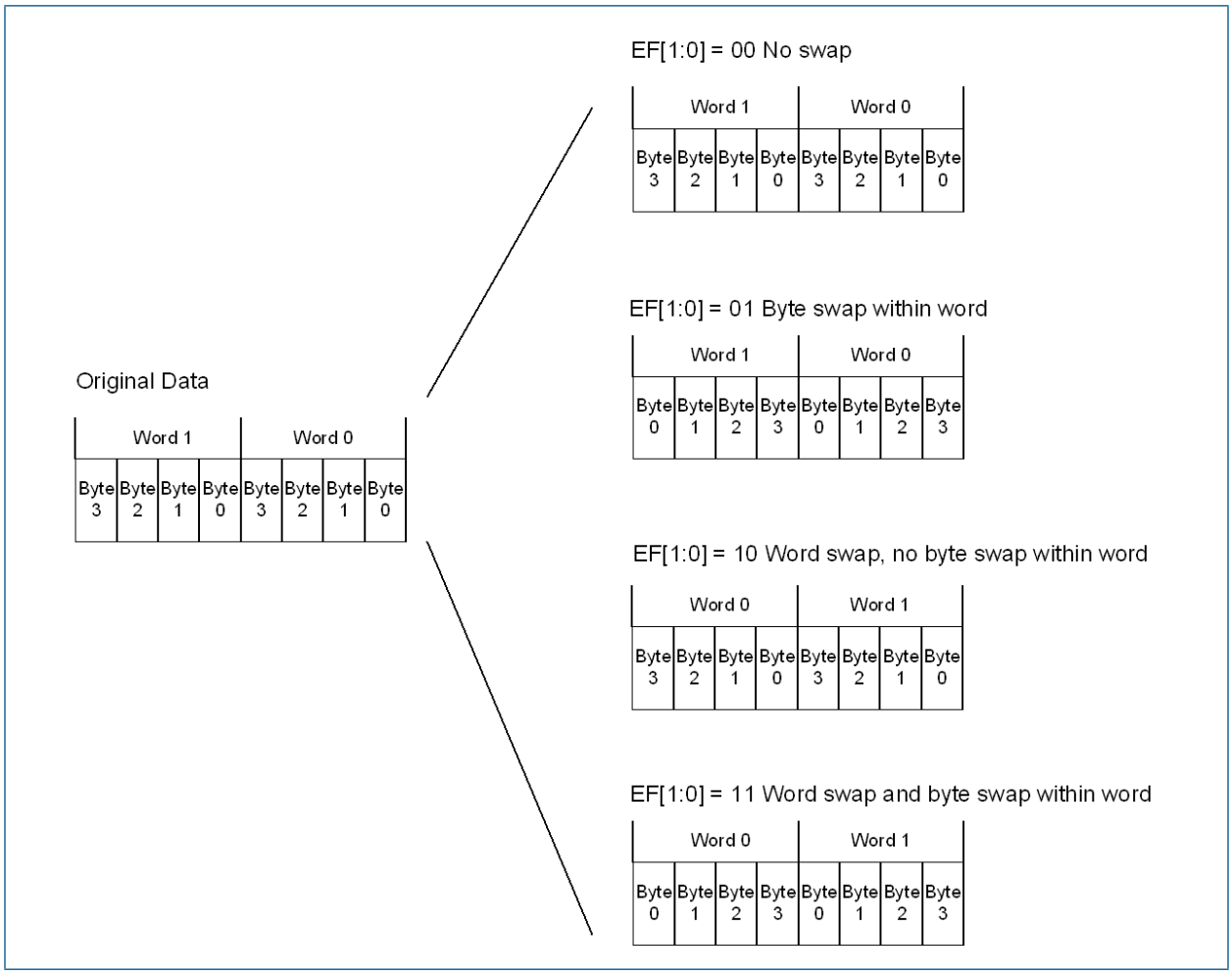
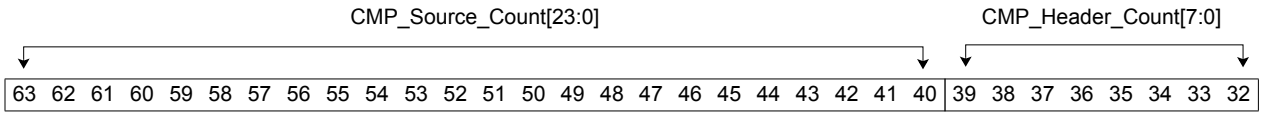
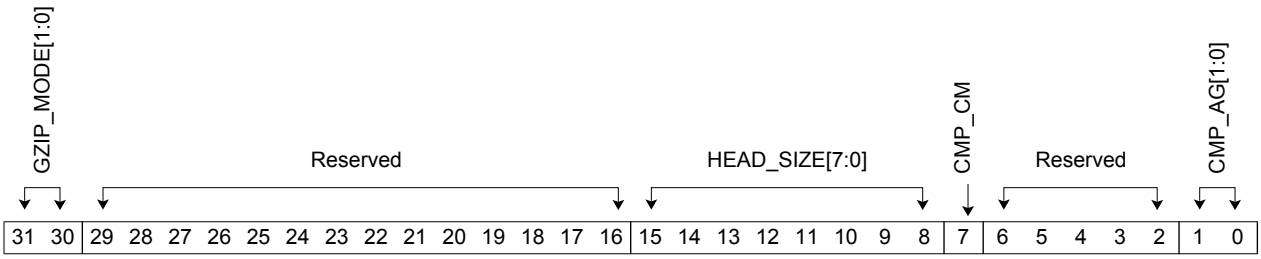


Figure 3-6. Endian Format Field Swap Options

3.1.2.3 Desc_cmd_cmp

This descriptor defines the compression engine related parameters. Desc_cmd_cmp is 8 bytes for either 32-bit or 64-bit addressing modes.



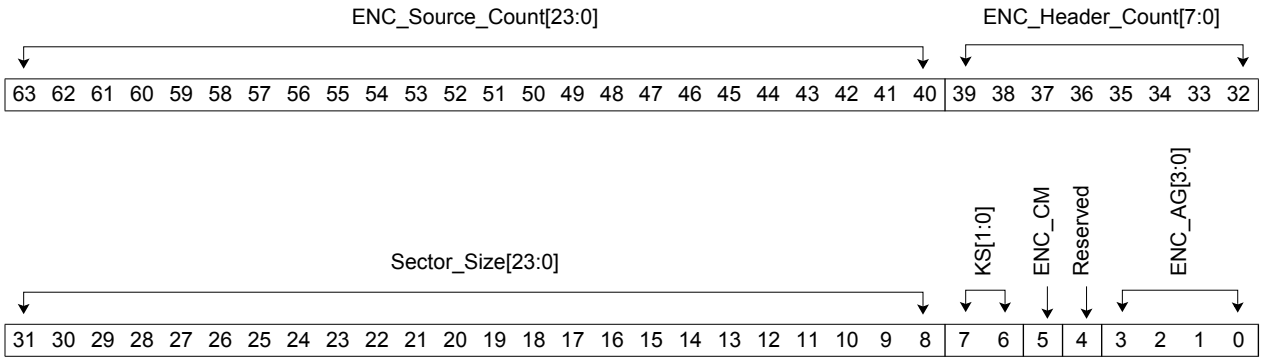


Field Name	Bits	Default	Description
CMP_Source_Count [23:0]	63:40	0	Compression Source Count. The number of bytes of compression data. The data stream following the header will be processed by the compression engine. The compression engine will stop processing data and return to passing through the remaining data in data stream after the Compression Source Counter reaches zero. The 820x Compression Source Counter begins to decrement after the Compression Head Counter has expired. The behavior of this field depends on the setting of compression count mode, CMP_CM. If CMP_CM = 0, 0 All data from Head Counter expired to EOB (end of block) need to be processed by the current engine 1 - 2^{24} 1 to 2^{24} bytes need to be processed by the current engine If CMP_CM =1, 0 - 2^{24} 0 to 2^{24} bytes after the last byte processed by the previous processing engine need to be processed by current engine
CMP_Header_Count [7:0]	39:32	0	Compression Header Count. This 8-bit field sets the number of 4-byte words that the compression engine will pass through before it begins processing the incoming data stream.
GZIP_MODE[1:0]	31:30	0	GZIP Operation Mode. 00 Static Huffman mode 01 Dynamic Huffman mode 10 Store mode 11 Reserved This field is only applicable on encode when CMP_AG[1:0] is set to GZIP or Deflate. In Store mode, a single block using BTYPE=00 (no compression) Deflate encoding is used.
Reserved	29:16	0	Reserved.

Field Name	Bits	Default	Description
HEAD_SIZE[7:0]	15:8	0	Head Size. The number of bytes required for the compression expansion calculation. 0 - 255: Head size in bytes The Compression engine will set the cmp_expansion bit (defined in the desc_result and result ring) to one if len(cmp) + Head_Size >= len(raw).
CMP_CM	7	0	Compression Count Mode. If this bit is set to zero, the compression source count in the 820x begins to decrement after the last header byte has been passed through the 820x. If this bit is set to one, the compression source counter begins to decrement after the last byte processed by the previous processing engine. Please refer to Section 3.1.3.10, "IPsec Packet Processing" for more information. 0 820x begins to decrement the compression source count after the last header byte 1 820x begins to decrement the compression source count after the last byte processed by the previous processing engine
Reserved	6:2	0	Reserved.
CMP_AG[1:0]	1:0	0	Compression Algorithm. Used to set the compression algorithm. 00 LZS algorithm 01 eLZS algorithm 10 GZIP algorithm 11 Deflate algorithm In GZIP mode, a standard GZIP file header and trailer are added on encode and verified/stripped on decode.

3.1.2.4 Desc_cmd_enc

This descriptor defines the encryption engine related parameters. Desc_cmd_enc is 8 bytes for either 32-bit or 64-bit addressing modes.



Field Name	Bits	Default	Description
ENC_Source_Count [23:0]	63:40	0	Encryption Source Count. The number of bytes of encryption data. The data stream following the header will be processed by the encryption engine. The encryption engine will stop processing data and return to passing through the remaining data in data stream after the Encryption Source Counter reaches zero. The 820x Encryption Source Counter begins to decrement after the Encryption Head Counter has expired. The behavior of this field depends on the setting of encryption count mode, ENC_CM. If ENC_CM = 0, 0 All data from Head Counter expired to EOB (end of block) need to be processed by the current engine 1 - 2^{24} 1 to 2^{24} bytes need to be processed by the current engine If ENC_CM =1, 0 - 2^{24} 0 to 2^{24} bytes after the last byte processed by the previous processing engine need to be processed by current engine
ENC_Header_Count [7:0]	39:32	0	Encryption Header Count. This 8-bit field sets the number of 4-byte words that the encryption engine will pass through before it begins processing the incoming data stream.
Sector_Size[23:0]	31:8	0	Sector Size. Sector size in 4 byte words. This field represents the sector size when ENC_AG[3:0] is set to the AES-XTS algorithm. Note that the sector size does not include the DIF (8 bytes) for XTS. Because the minimum sector size is 128 bytes, a sector size of 0x000000 to 0x00001F translates to 0x000020 words.
KS[1:0]	7:6	0	AES Key Size. 00 128 bit key 01 192 bit key 10 256 bit key 11 512 bit key (only valid for AES-XTS) Note: AES-XTS only supports 256 and 512 bit keys. Other AES modes support 128, 192 and 256 bits keys.

Field Name	Bits	Default	Description
ENC_CM	5	0	Encryption Count Mode. If this bit is set to zero, the encryption source count in the 820x begins to decrement after the last header byte has been passed through the 820x. If this bit is set to one, the encryption source counter begins to decrement after the last byte processed by the previous processing engine. Please refer to Section 3.1.3.10, "IPsec Packet Processing" for more information. 0 820x begins to decrement the encryption source count after the last header byte 1 820x begins to decrement the encryption source count after the last byte processed by the previous processing engine
Reserved	4	0	Reserved.
ENC_AG[3:0]	3:0	0	Encryption Algorithm. 0000 AES-GCM Hash engine should be set to AES-GCM-MAC algorithm 0001 AES-CBC 0010 AES-CTR 0011 AES-ECB 0100 AES-XTS Compression Engine must be disabled and Hash_OP[1:0] cannot be set any type of MAC operation 01013DES All other settings are reserved. Note: GMAC operations use the AES algorithm. If sharing logic with other AES modes, the GMAC will be implemented in the Encryption Engine.

3.1.2.5 Desc_cmd_hash

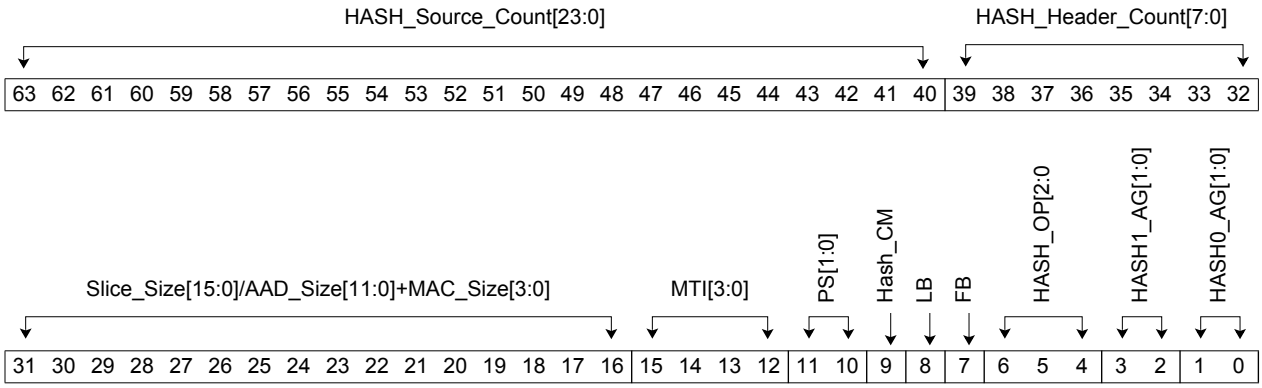
This descriptor defines the hash engine related parameters. Desc_cmd_hash is 8 bytes for either 32-bit or 64-bit addressing modes.

During a Slice Hash operation, the Hash engine will calculate the hash value for every slice of data. A single Slice Hash operation must contain more than one slice. The size of one slice data is defined by Slice_Size[15:0].

During a File Hash operation, the Hash engine calculates one hash value for the entire file. File Hash operations may be stateless or stateful. For a stateless file hash operation, one command operates on data from one file and the "file hash chaining value" of this command is the "File Hash" of the file (see Section 3.1.3.6). For a stateful file hash operation, one data file is operated on by several commands, and each command is assigned a file hash chaining value by the 820x. The file hash chaining value of the prior final command is used as the "File hash" of the file.

Note: Host software manages the stateful file hash operation flow. The software must wait for one data block of the file from the current command to finish before proceeding to the next block of the file in the next command. This is because the next command's IHV (initial hash vector) will use the previous command's file chaining hash value.

Please refer to Section 3.1.3.6, "Hash Buffer" for a detailed description of a hash operation.



Field Name	Bits	Default	Description
HASH_Source_Count [23:0]	63:40	0	Hash Source Count. The number of bytes of hash data. The data stream following the header will be processed by the hash engine. The hash engine will stop processing data and return to passing through the remaining data in data stream after the Hash Source Counter reaches zero. The 820x Hash Source Counter begins to decrement after the Hash Head Counter has expired. Note: This field is NOT valid when performing the last block of a stateful file hash operation. The behavior of this field depends on the setting of hash count mode, HASH_CM. If HASH_CM = 0, 0 All data from Head Counter expired to EOB (end of block) need to be processed by the current engine 1 - 2^{24} 1 to 2^{24} bytes need to be processed by the current engine If HASH_CM =1, 0 - 2^{24} 0 to 2^{24} bytes after the last byte processed by the previous processing engine need to be processed by current engine

Field Name	Bits	Default	Description
HASH_Header_Count [7:0]	39:32	0	Hash Header Count. This 8-bit field sets the number of 4-byte words that the hash engine will pass through before it begins processing the incoming data stream. Note: This field is NOT valid when performing the last block of a stateful file hash operation.

Field Name	Bits	Default	Description
Slice_Size[15:0] - or - bits [31:2] = AAD_Size[11:0] bits [19:16] = MAC_Size[3:0]	31:16	0	The value of this field depends on the type of hash operation. For slice hash operations (HASH_OP = 000 or 010), this field represents the hash slice size. The slice size is in bytes and must be a multiple of 64 bytes. Note: The total data size of one command can be an arbitrary number of bytes. The 820x will split the data into slices according to the Slice_Size. If the last slice is less than the Slice_Size, the 820x will pad the data with zeroes so that it is a multiple of 64 bytes. Each slice is treated as a complete hash operation (with length padding). The 820x will send the host the hash value per slice of data. For AES-GCM-MAC hash operations (HASH_OP = 111) this field represents the Additional Authenticated Data size, AAD_Size[11:0]. The AAD size values is in bytes with a range of 0x000 - 0xFFF, which translates into 0 - 4095 bytes Note: In GMAC mode, the host software does not need to fill out the AAD Size because the AAD will be written to the Hash engine as source data, and so the Hash engine will calculate the length by itself. Please refer to Section 3.1.3.8, "AES-GCM and GMAC Operations" for details on GMAC operations. For all hash MAC operations (HASH_OP = 011 or 1xx), this field represents MAC_Size[3:0]. For an encode operation, MAC_Size represents the number of words that will be inserted into the data stream. 0 MAC is not inserted into the data stream after calculated 1 - 15 Insert 2 to 16 words into the data stream The number of inserted MAC words for MD5 is 16 bytes, for SHA1 20 bytes, for SHA2 32 bytes. The 820x will truncate the MAC value according to the configuration before inserted the MAC into the data stream. For decode operations, MAC_Size represents the MAC size on the information bus (see Section 3.1.3.5, "Data Stream Information Fields"). The hash engine will compare the calculated MAC with the MAC on the information bus of this length. For example, if the calculated HMAC with SHA256 is 32 bytes and the MAC_Size is 24 bytes, the hash engine will compare the first 24 bytes of the calculated HMAC with the MAC on the information bus. This feature is useful in SSL. The MAC will be inserted at the end of the data stream that is processed by the hash engine.

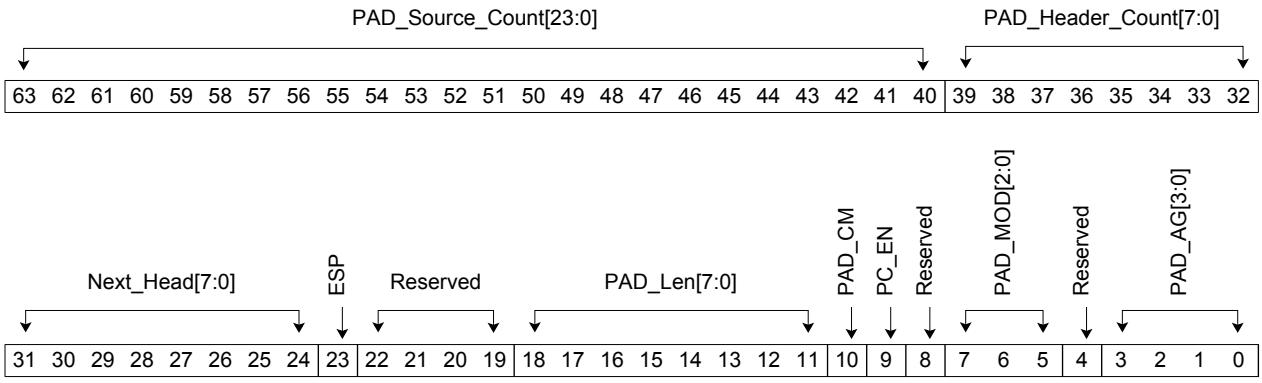
Field Name	Bits	Default	Description
MTI[3:0]	15:12	0	Mute Table Index. This field selects the mute table entries to be used on the first 16 bytes of the data stream processed by the Hash engine. 0000 Disable mute function 0001 Mute table entry 1 0010 Mute table entry 2 0011 Mute table entry 3 0100 Mute table entry 4 0101 Mute table entry 5 0110 Mute table entry 6 0111 Mute table entry 7 All other values are not defined. Please refer to Section 6.3.1, "Hash Engine Mute Table Entry Registers" for details.
PS[1:0]	11:10	0	Hash Engine Position This field's behavior depends on whether the operation is encode or decode. For encode operations, the hash engine is: 00 Before the compression engine 01 After the compression engine 10 After the pad engine 11 After the encryption engine For decode operations, the hash engine is: 00 Before the encryption engine 01 After the encryption engine 10 After the pad engine 11 After the compression engine Please refer to Chapter 4, "Data Flow" for more information.
HASH_CM	9	0	Hash Count Mode. If this bit is set to zero, the hash source count in the 820x begins to decrement after the last header byte has been passed through the 820x. If this bit is set to one, the hash source counter begins to decrement after the last byte processed by the previous processing engine. Please refer to Section 3.1.3.10, "IPsec Packet Processing" for more information. 0 820x begins to decrement the hash source count after the last header byte 1 820x begins to decrement the hash source count after the last byte processed by the previous processing engine

Field Name	Bits	Default	Description
LB	8	0	Last Block. The host should set this bit to indicate the last block of data. For stateful operations, data from one command is considered one block. 0 For stateful operations, if not last block 1 For stateless operations, or for stateful operations if last block Please refer to Section 3.1.3.9, "MAC Operations" for detailed usage of this bit.
FB	7	0	First Block. The host should set this bit to indicate the first block of data. For stateful operations, data from one command is considered one block. 0 For stateful operations, if not first block 1 For stateless operations, or for stateful operations if first block Please refer to Section 3.1.3.9, "MAC Operations" for detailed usage of this bit.
HASH_OP[2:0]	6:4	0	Hash operation. 000 Slice hash only, internal integrity checking 001 File hash only, internal integrity checking 010 Slice hash + file hash, no internal integrity checking 011 HMAC, internal integrity checking 100 SSL3.0 MAC, internal integrity checking 101 XCBC-MAC, internal integrity checking 110 GMAC, internal integrity checking; Encryption engine must be disabled 111 AES-GCM-MAC, internal integrity checking; Encryption engine must be set to AES-GCM algorithm For encode operations, if PS == 00, Hash_OP may be any one of the eight choices and the Hash engine will calculate the hash or MAC. If PS != 00, the Hash Engine will only calculate the MAC. For decode operations, Hash_OP may only be set to MAC operations (1xx). The Hash engine will calculate the MAC and compare the calculated value to the MAC in the data stream. Note: If the operation is SSL3.0 MAC, the source data in host memory should include "Seq. No. || Type || Length || Application Data". Please refer to Appendix D for details. Note: If the operation is XCBC-MAC, GMAC or AES-GCM-MAC, Hash0_AG and Hash1_AG are not valid since these MACs use the AES algorithm core.

Field Name	Bits	Default	Description
HASH1_AG[1:0]	3:2	0	Hash Core 1 algorithm. This field is only valid when Hash_OP[3:0] = 0010. In this case, Hash Core 0 will calculate the file hash while Hash Core 1 calculates the slice hash. For all other operations, Hash Core 1 is only used for real time verification and must be set to the same value as Hash Core 0. 00 SHA-1 01 SHA-256 10 MD5 11 Reserved
HASH0_AG[1:0]	1:0	0	Hash Core 0 algorithm. 00 SHA-1 01 SHA-256 10 MD5 11 Reserved

3.1.2.6 Desc_cmd_pad

This descriptor defines the pad engine related parameters. Desc_cmd_pad is 8 bytes for either 32-bit or 64-bit addressing modes.



Field Name	Bits	Default	Description
PAD_Source_Count [23:0]	63:40	0	Pad Source Count. The number of bytes of pad data. The data stream following the header will be processed by the pad engine. The pad engine will stop processing data and return to passing through the remaining data in data stream after the pad Source Counter reaches zero. The 820x pad Source Counter begins to decrement after the pad Head Counter has expired. The behavior of this field depends on the setting of hash count mode, PAD_CM. If PAD_CM = 0, 0 — All data from Head Counter expired to EOB (end of block) needs to be processed by the current engine 1 - 2^{24} — 1 to 2^{24} bytes need to be processed by the current engine If PAD_CM =1, 0 - 2^{24} — 0 to 2^{24} bytes after the last byte processed by the previous processing engine needs to be processed by current engine
PAD_Header_Count [7:0]	39:32	0	Pad Header Count. This 8-bit field sets the number of 4-byte words that the pad engine will pass through before it begins processing the incoming data stream.
Next_Head[7:0]	31:24	0	Next Header. This field is only valid when the ESP bit is set to one. The Next_Head[7:0] value is an IPv4/IPv6 protocol number describing the format of the Payload data field. The host software must this field for IPsec packet processing. For an encode operation, the host software must enter this field into the command structure. The Pad Engine will append it to the data stream. For a decode operation, the host software should set this field to zero when building the command structure. The 820x will update this field after the command completes.

Field Name	Bits	Default	Description
ESP	23	0	ESP Header Pad. For an encode operation, 0 The value of Next_Head[7:0] will not be inserted by the pad engine. 1 Pad engine will pad Next_Head[7:0] with "pad" and Pad_Len[7:0] for IPsec packet processing. The ESP Trailer will also be included in the Dest_byte_count. For a decode operation, 0 Pad engine will extract the Pad_Len[7:0] field from the data stream, and write the pad length to the host command structure after the command completes 1 Pad engine will extract Pad_Len[7:0] and Next_Head[7:0] fields from the data stream, and write the pad length and next header to the host command structure after the command completes Note: For both encode and decode operations, Pad_AG[3] must be set to one if ESP = 1. Regardless of the ESP value, the Pad Engine will always apply padding even if the data is already a multiple of the MOD-X. Please refer to Appendix A for more information about ESP IPsec processing. ESP Header Format for ESP = 0: 1 byte Payload | Pad | Pad Length ESP Header Format for ESP = 1: |----------ESP Trailer----------| 1 byte 1 byte Payload | Pad | Pad Length | Next Header
Reserved	22:19	0	Reserved.
Pad_Len[7:0]	18:11	0	Pad Length. This field is only valid for decode operations. For a decode operation, the pad engine removes the padding from the data stream. This field is the number of bytes that the pad engine stripped from the data. The pad engine will extract this field from the data stream. Note: The host software should set this field to zero when building the command structure. The 820x will update this field in the command structure after the command completes.

Field Name	Bits	Default	Description
PAD_CM	10	0	Pad Count Mode. If this bit is set to zero, the pad source count in the 820x begins to decrement after the last header byte has been passed through the 820x. If this bit is set to one, the pad source counter begins to decrement after the last byte processed by the previous processing engine. Please refer to Section 3.1.3.10, "IPsec Packet Processing" for more information. 0 820x begins to decrement the pad source count after the last header byte 1 820x begins to decrement the pad source count after the last byte processed by the previous processing engine
PAD_EN	9	0	Pad Check Enable. This bit is only valid for decode operations. 0 Pad engine will remove and not verify the padding values defined by PAD_AG[3:0] 1 Pad engine will remove and verify the padding values defined by PAD_AG[3:0]
Reserved	8	0	Reserved.
PAD_MOD[2:0]	7:5	0	Pad Modulo. This field is used to set the alignment of the payload after padding has been added. 000 MOD-4: the payload after padding is 4 bytes aligned 001 MOD-8: the payload after padding is 8 bytes aligned 010 MOD-16: the payload after padding is 16 bytes aligned 011 MOD-32: the payload after padding is 32 bytes aligned 100 MOD-64: the payload after padding is 64 bytes aligned 101 MOD-128: the payload after padding is 128 bytes aligned 110 MOD-256: the payload after padding is 256 bytes aligned 111 Reserved
Reserved	4	0	Reserved.

<table>
<tr><th>Field Name</th><th>Bits</th><th>Default</th><th>Description</th></tr>
<tr><td>PAD_AG[3:0]</td><td>3:0</td><td>0000</td><td>Pad Engine algorithm.
The behavior of this field depends if the operation is encode or decode and the setting of the ESP field (see Section 3.1.2.6).
Refer to Table 3-2 for the decode values for this field.
For all algorithms, the pad length is the number of bytes in the preceding Padding field, except for setting 1010 when the pad length is the number of bytes in the preceding Padding field plus the “pad length” byte.
PAD_AG[3] indicates whether the pad length field is included in the padding. If PAD_AG[3] = 0, the pad length field is not included, and if PAD_AG[3] = 1, the pad length field is included.
PAD_AG[3] = 0, without Length byte
<table><tr><td>Payload data</td><td>Pad</td></tr></table>
PAD_AG[3] = 1, with Length byte
<table><tr><td>Payload data</td><td>Pad</td><td>Pad Length</td></tr></table></td></tr>
</table>

Table 3-2. PAD_AG[3:0] Field Decoding

Encode/ Decode	ESP	PAD_AG [3:0]	Description
Encode	0	0000	Padding value is 0 - 255. For example, if 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 00, 01, 02, 03, 04.
		0001	Padding value 1 - 255 - 0. For example, If padding 256 bytes, the padding will be 1, 2, 3, ... 255, 0.
		0010	Padding with same value, the value is the number of bytes inserted. For example, If 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 05, 05, 05, 05, 05.
		0011	Padding will be all zeroes. For example, If 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 00, 00, 00, 00, 00.
		0100	Padding with number of bytes minus one inserted. For example, If 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 04, 04, 04, 04, 04.
		1000	Padding value 0 - 255, with pad length field. For example, If 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 00, 01, 02, 03, 04.
		1001	Padding value 1 - 255 - 0, with pad length field. For example, If 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 01, 02, 03, 04, 04.
		1010	Padding with same value, the value is the number of bytes inserted, with pad length field. For example, If 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 05, 05, 05, 05, 05.
		1011	Padding with all zero, with pad length field. For example, If 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 00, 00, 00, 00, 04.
		1100	Padding with number of bytes inserted minus 1, with pad length field. For example, If 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 04, 04, 04, 04, 04.
		All others	Reserved.

Table 3-2. PAD_AG[3:0] Field Decoding

Encode/ Decode	ESP	PAD_AG [3:0]	Description
Encode	1	0xxx	Not allowed
		1000	Padding value 0 - 255, with pad length field and next_header. For example, if 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 00, 01, 02, 03, <next_header>.
		1001	Padding value 1 - 255 - 0, with pad length field and next_header. For example, if 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 01, 02, 03, 03, <next_header>. Note: please refer to RFC 2406 (ESP) for this configuration.
		1010	Padding with same value, the value is the number of bytes inserted, with pad length field and next_header. For example, if 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 04, 04, 04, 04, <next_header>.
		1011	Padding with all zeroes, with pad length field and next_header. For example, if 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 00, 00, 00, 03, <next_header>.
		1100	Padding with number of bytes inserted minus 1, with pad length field and next_header. For example, if 5 bytes are required to pad out to modulo 8, 5 bytes will be inserted, the padding will be 03, 03, 03, 03, <next_header>.
		All others	Reserved.
Decode	0	0000	No verification and removal (no pad length field)
		0001	No verification and removal (no pad length field)
		0011	No verification and removal (no pad length field)
		0100	No verification and removal (no pad length field)
		1000	Verify the removed padding value 0 - 255 and the pad length field
		1001	Verify the removed padding value 1 - 255 - 0 and the pad length field
		1010	Verify the removed padding of number of bytes inserted, and the pad length field
		1011	Verify the removed padding value of all zeroes and the pad length field
		1100	Verify the removed padding of number of bytes inserted minus one, and the pad length field
		All others	Reserved.

Table 3-2. PAD_AG[3:0] Field Decoding

Encode/ Decode	ESP	PAD_AG [3:0]	Description
Decode	1	0xxx	Not valid.
		1000	Verify the removed padding value 0 - 255, pad length field; remove next_header
		1001	Verify the removed padding value 1 - 255 - 0, pad length field; remove next_header
		1010	Verify the removed padding of number of bytes inserted, pad length field; remove next_header
		1011	Verify the removed padding value of all zeroes, pad length field; remove next_header
		1100	Verify the removed padding value of all padding length value less one, pad length field; remove next_header
		All others	Reserved.

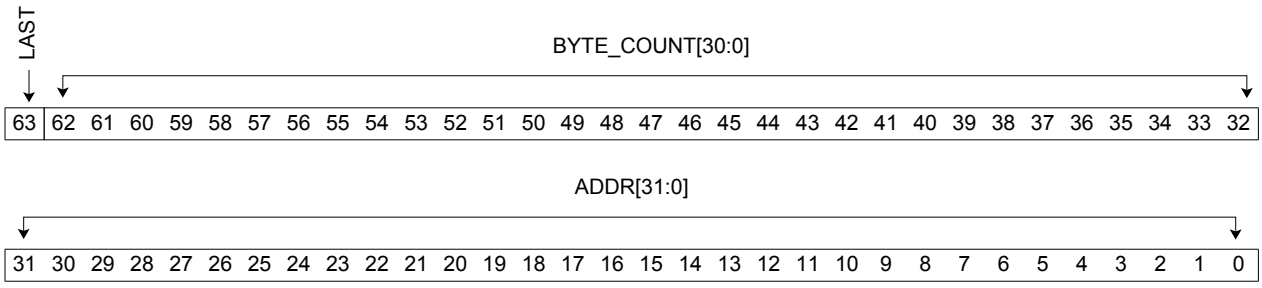
3.1.2.7 Desc_srcX and Desc_dstX

This source descriptor is only used in normal mode. The source data descriptors point to source data buffers. The source buffer address length may be an arbitrary number of bytes. The source buffer address is byte aligned except for when the source buffer is used to store the information fields for the "Key", "IHV", "MAC", or "IV", in which case its address should be 8-byte aligned. If a source buffer contains information fields, it should not contain source data.

The destination data descriptors point to the destination buffers. The destination buffer address and length must be on an 8 byte boundary. Usually, the destination buffers are used to store the result data, and must be fully consumed in order. For hash related operations, the first destination buffer is used to store hash values only.

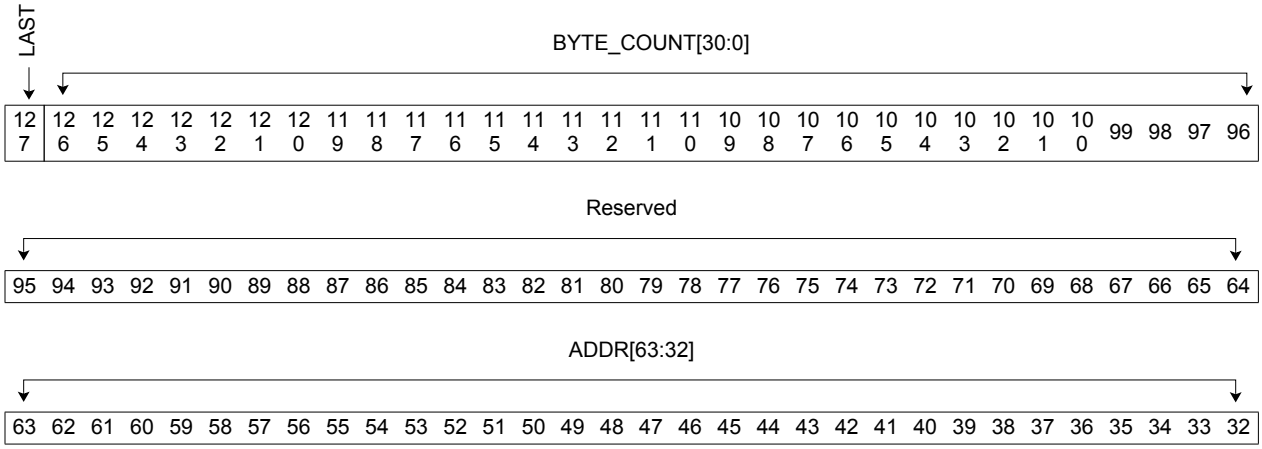
The source and destination descriptors both contain 8 bytes in the 32-bit addressing mode and 16 bytes in the 64-bit addressing mode.

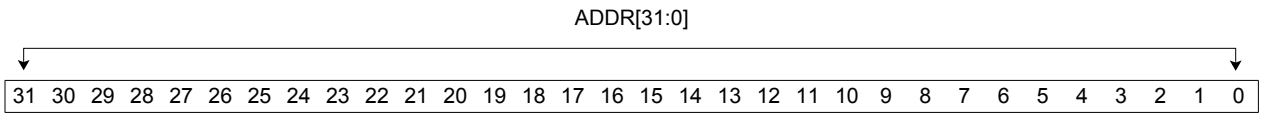
32-bit Addressing Mode Format:



Field Name	Bits	Default	Description
Last	63	0	Last. Used to identify the current descriptor as the last valid descriptor in the command. The host software must set this bit in the command structure. Although the source and destination descriptors must be in pairs in normal mode, the LAST flag need NOT be set in the same pair. For example, the source buffer LAST flag may be set at Dest_src2, but the destination descriptor buffer LAST flag is set at Dest_dst6. Note: In small packet mode, there are two destination descriptors but no source descriptors.
Byte_Count[30:0]	62:32	0	Byte Count. The size of the source or destination buffer in bytes. It indicates the size of the data block defined by this descriptor. For the source descriptor, the byte count is an arbitrary number of bytes. For the destination descriptor, the byte count must be a multiple of 8 bytes.
ADDR[31:0]	31:0	0	Address. The starting physical address of the source or destination descriptor. For source descriptors, including those that only contain AAD, the address is byte aligned. If the source buffers include the information fields “Key”, “IV”, “IHV”,”MAC”, the address must be on an 8-byte boundary. For destination descriptors, the address is always aligned on an 8-byte boundary.

64-bit Addressing Mode Format:





Field Name	Bits	Default	Description
Last	127	0	Last. Used to identify the current descriptor as the last valid descriptor in the command. The host software must set this bit in the command structure. Although the source and destination descriptors must be in pairs in normal mode, the LAST flag need NOT be set in the same pair. For example, the source buffer LAST flag may be set at Dest_src2, but the destination descriptor buffer LAST flag is set at Dest_dst6. Note: In small packet mode, there are two destination descriptors but no source descriptors.
Byte_Count[30:0]	126:96	0	Byte Count. The size of the source or destination buffer in bytes and indicates the block size defined by the descriptor. For the source descriptor, the byte count is an arbitrary number of bytes. For the destination descriptor, the byte count must be a multiple of 8 bytes.
Reserved	95:64	0	Reserved.
ADDR[63:0]	63:0	0	Address. The starting physical address of the source or destination descriptor. For source descriptors, including those that only contain AAD, the address is byte aligned. If the source buffers include the information fields “Key”, “IV”, “IHV”, ”MAC”, the address must be on an 8-byte boundary. For destination descriptors, the address must be aligned on an 8-byte boundary.

3.1.3 Command Structure Conventions

This section designates some command structure conventions, clarifies some confusing topics, and provides a better understanding of how a command structure is built.

All data streams described in this section are assumed to have the basic format shown below:

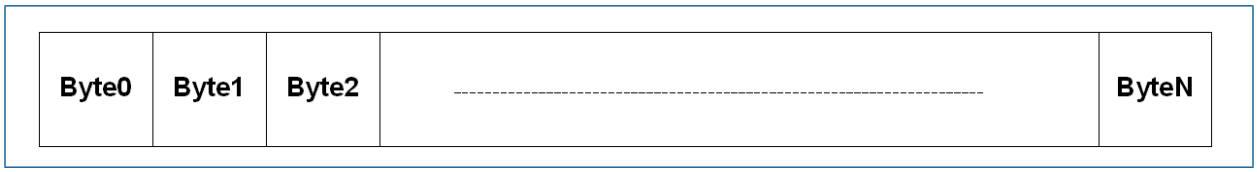


Figure 3-7. Basic Data Sequence

3.1.3.1 Initial Hash engine Value (IHV)

For a hash related operations, the host software must write the Initial Hash Value (IHV) for the hash engine in the proper place in the source buffer. IHV stands for not only the initial hash operation value, but also the partial MAC operation value in stateful MAC operation. Because a slice hash operation always uses the same default value, the host software need not write the IHV for slice hash operations. The descriptions below apply for file hash and all types of hash/MAC operations.

For a stateless hash operation or the first block of a stateful hash operation, the IHV is the default value defined by the hash algorithm. For a stateful hash/MAC operation after the first block, the host software should read the IHV (Partial hash/MAC result) from the hash buffer (the first destination buffer is used as a hash buffer in a hash related command).

Default Initial Hash Vector for Hash Operations

The default value and size of the IHV depend on the hash algorithm. In the 820x, the host software must pad the IHV to 256 bits for all algorithms before writing the IHV to the first source buffer.

The examples below illustrate the default IHV after padding for three common hash algorithms.

```
SHA1: IHV[0:255] ={67452301
    efcdab89
    98badcfe
    h10325476
    c3d2e1f0
    96'd0};

SHA256: IHV[0:255] = {6a09e667
    bb67ae85
    3c6ef372
    a54ff53a
    510e527f
    9b05688c
    1f83d9ab
    5be0cd19};

MD5: IHV[0:255] = {67452301
    efcdab89
    98badcfe
    10325476
    128'd0};
```

Partial HMAC Value for Stateful Operations

For a stateful HMAC operation, the host software should write the partial HMAC value into the IHV field for all blocks except the first.

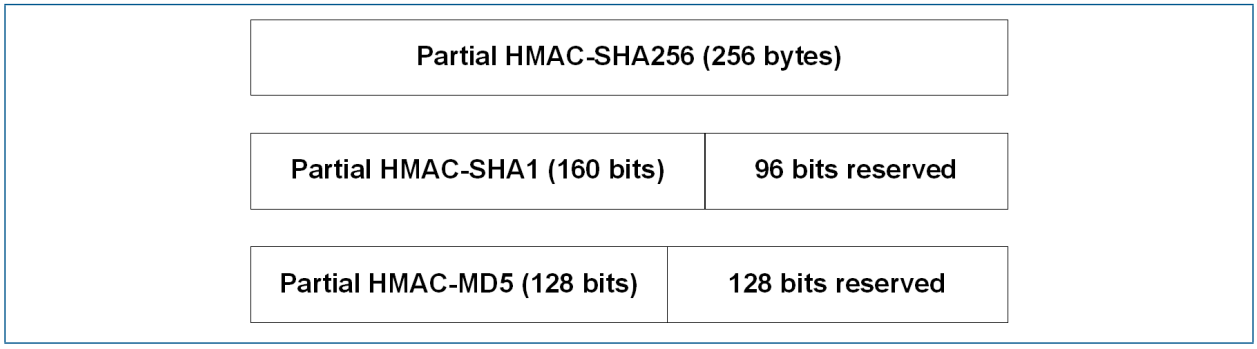


Figure 3-8. Partial IHV Field for a Stateful HMAC Operations

Partial XCBC-MAC Value for Stateful Operations

For a stateful XCBC-MAC operation, the host software should write the partial XMAC-MAC value into the IHV field for all blocks except the first.

Figure 3-9. Partial IHV Field for a Stateful XCBC-MAC Operations

Partial AES-GCM-MAC/GMAC Value for Stateful Operations

For a stateful AES-GCM-MAC or GMAC operation, the host software should write the partial AES-GCM-MAC/GMAC value into the IHV field for all blocks except the first. Please refer to Section 3.1.3.9, "MAC Operations" for a description of how to calculate s0.

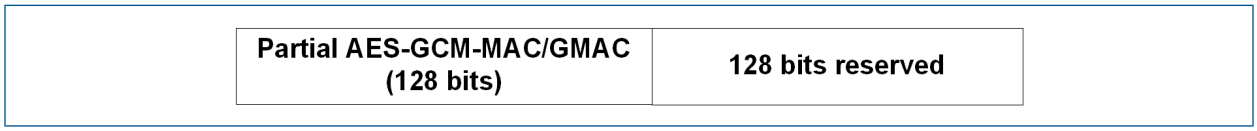


Figure 3-10. Partial IHV Field for a Stateful AES-GCM-MAC/GMAC Operations

Partial SSL3.0-MAC Value for Stateful Operations

For a stateful SSL3.0-MAC operation, the host software should write the partial SSL3.0-MAC value into the IHV field for all blocks except the first.

Because the IHV is not protected by a CRC, the 820x will transmit the IHV to the host destination buffer if the DIR bit in the Desc_cmd_base is set to one. The host software can then determine whether the IHV has been corrupted.

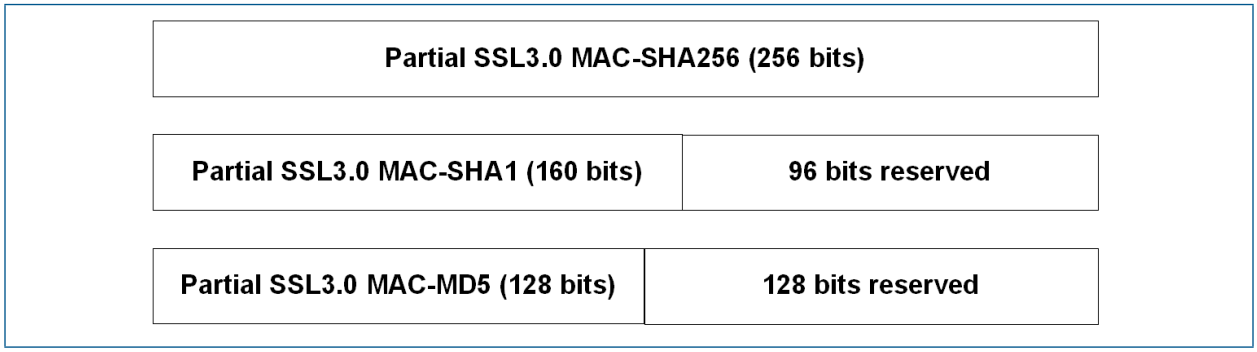


Figure 3-11. Partial IHV Field for a Stateful SSL3.0-MAC Operations

3.1.3.2 MAC

For encode operations, the host software configures the Hash Engine's position in the data processing channel and builds the command structure according to the application's requirement. The Hash Engine calculates the MAC value of the input data stream. The 820x writes the resulting MAC value to the first entry of the hash buffer.

For decode operations, the host software writes the MAC value into the proper location of the source buffer. The 820x Channel Manager will separate the MAC value from the source data stream and put it on the hash information bus (Please refer to Section 3.1.3.5, "Data Stream Information Fields" for details). The Hash Engine calculates the MAC value of the input data stream, compares it with the value in the information bus, and then reports the verified value or failure flag to the host. For a stateful MAC operation, the host software should also write the partial MAC value into the IHV field. The MAC value field in the first source buffer is fixed to 256 bits (32 bytes) as shown in Figure 3-12 below.

Because the MAC is not protected by a CRC, the 820x will transmit the MAC to the host destination buffer. The host software can then determine whether the MAC has been corrupted.

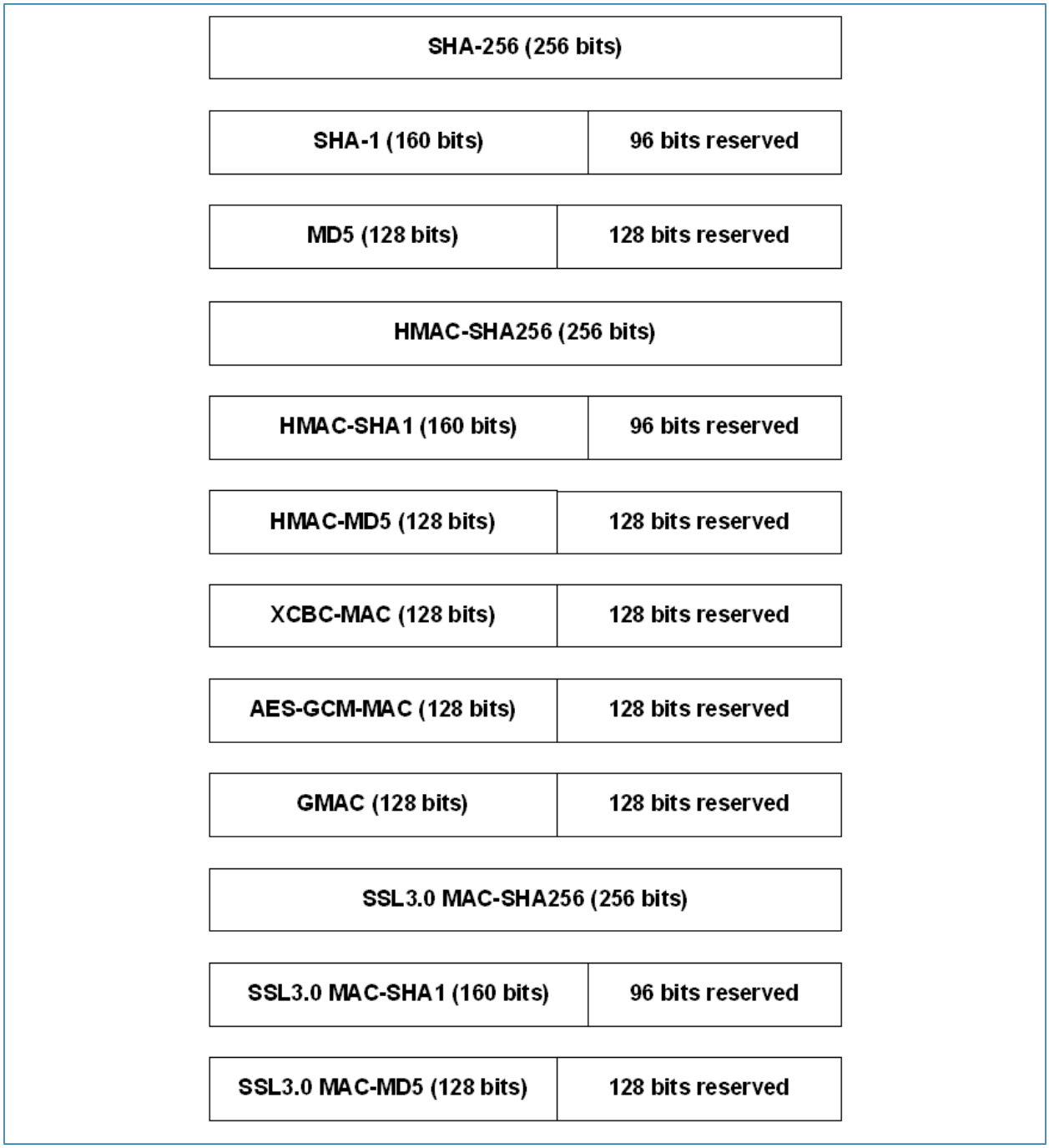


Figure 3-12. MAC Field Format

3.1.3.3 Initialization Vector (IV) and Additional Authenticated Data (AAD)

All AES related operations except AES-ECB require a 16 byte initialization vector (IV) for AES-CTR, AES-CBC, AES-XTS and GMAC, or a 32 byte initialization vector for AES-GCM, in the source buffer. The 16 byte IV for AES-XTS is actually the logic address of the disk sector. The Normal IV is the initial vector for the Encryption Engine AES algorithm and the S0_IV is the initialization vector for the Hash Engine MAC operation AES algorithm.

As shown in Figure 3-13 below, the 16 byte IV for GMAC operations is the S0_IV (J0) for the S0 calculation. AES-GCM requires 16 bytes of Normal_IV (J0) for the Encryption Engine and another 16 bytes of S0_IV for the Hash engine. The Normal_IV and S0_IV are the same value for stateless operation or for the first block of a stateful operation. The Normal_IV and S0_IV will be different for the middle blocks and last block of a stateful operation because the Normal_IV is not the first block initial value for the Encryption engine, but S0_IV is the first block initial value for the Hash engine.

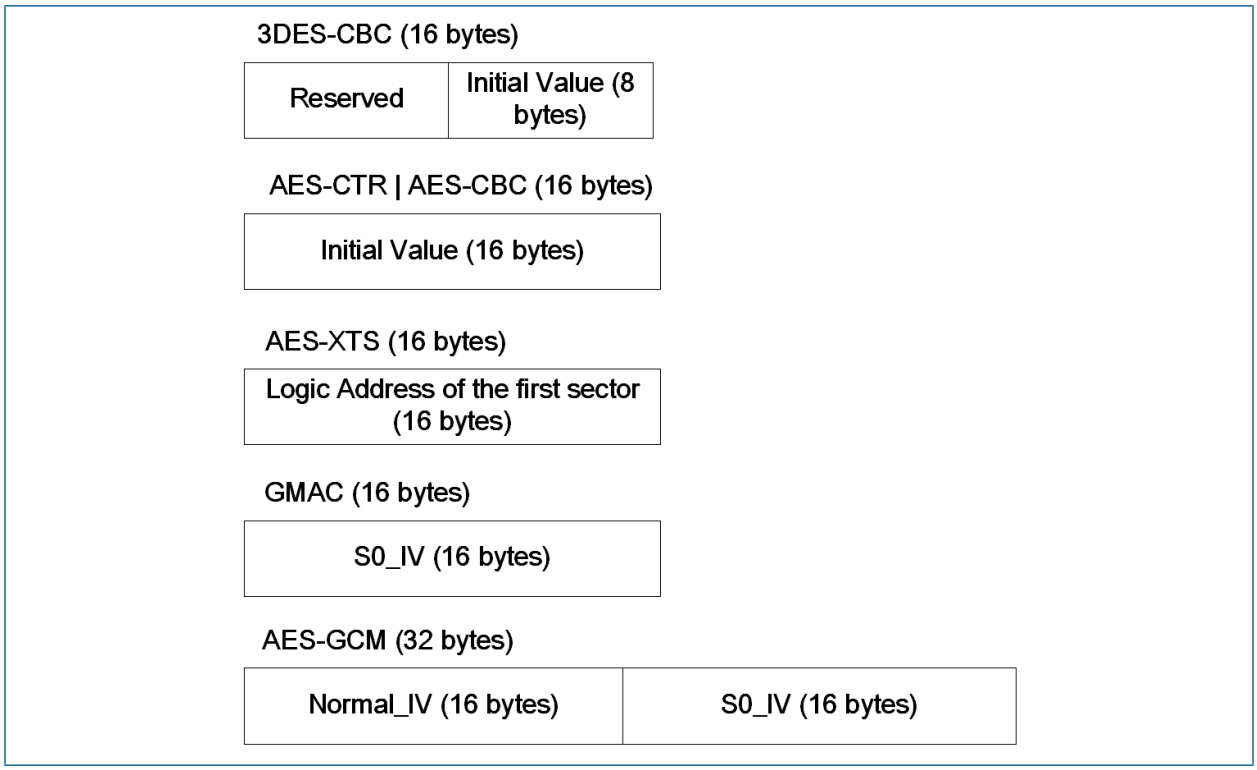


Figure 3-13. AES IV Lengths

Some AES-GCM related operations require Additional Authenticated Data (AAD) which is authenticated but not encrypted.

In normal mode, the AAD can be written into a single source buffer, or several source buffers. The length and address of a source buffer that only contains the AAD may be of arbitrary length.

In small packet mode, the host software must write the AAD into the command structure. The AAD length must be padded so that it is aligned on an 8-byte boundary. For example, if AAD_Size is equal to 57, the AAD field in the command structure would be 64 bytes (padded with 7 bytes of zeroes).

Because the IV and AAD are not protected by a CRC, the 820x will transmit the IV and AAD to the host destination buffer. The host software can then determine whether the IV or AAD have been corrupted.

3.1.3.4 Key Format

For Encryption and Hash related operations, the host software must write the encryption key to the 820x. The key must be located in the first source buffer and may have one of the five formats described in this section.

For all key formats described in this section, reserved fields and the CRC verified "Key + reserved fields" should be filled with zeroes when written by the host software.

The 820x will automatically verify the key CRC when reading the key, and report a "key CRC error" to the host if the CRC verification failed. Although the key is CRC protected, the 820x will still write the key to the host destination buffer if the DIR bit in the Desc_cmd_base descriptor is set to one.

Key Format 1: AES, 3DES

If the Encryption Engine and Hash Engine are both enabled and Hash_OP[3:0] is set to a Slice Hash, File Hash or Slice + File Hash operation, the Hash Engine will calculate the hash values without a key and the Encryption Engine will encrypt the data with a key. If the Encryption Engine is disabled, AES commands will not require a key. This format for these operations only includes the encryption key and requires 72 bytes.

If the Encryption engine and Hash engine are both enabled, and Hash_OP[3:0] is set to GMAC or AES-GCM-MAC, the Encryption engine and Hash engine will use the same AES key to separately calculate the cipher text and authentication value. This format for these operations only includes the AES key and requires 72 bytes.

The key format shown in Figure 3-14 is used for AES and 3DES operations.

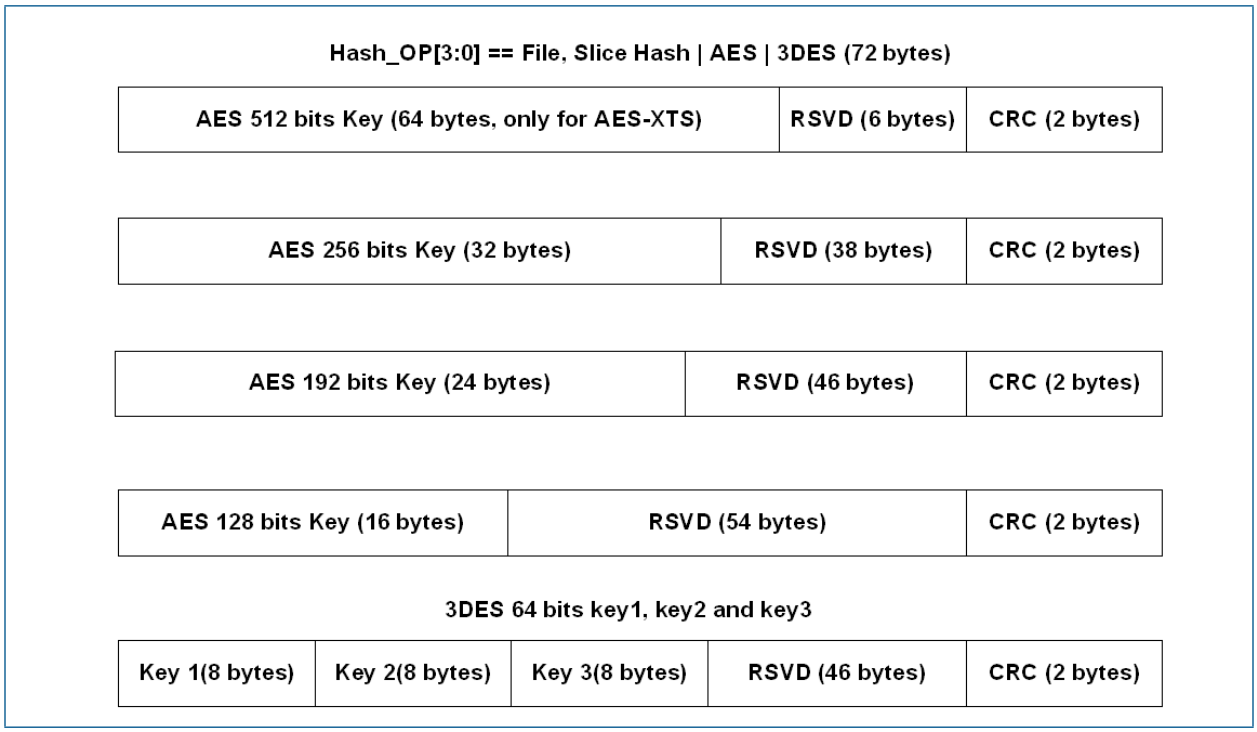


Figure 3-14. Key Format 1 (72 bytes)

Key Format 2: HMAC, SSL3.0-MAC

If the Encryption and Hash engines are enabled and Hash_OP[3:0] is set to "HMAC" or "SSL3.0-MAC (SHA256 or MD5)", the Hash engine will calculate the MAC, and the IPAD and OPAD fields must be generated by the host software according to the MAC key (please refer to Section 3.1.3.7, "IPAD & OPAD for HMAC & SSL3.0-MAC (SHA256 | MD5)" for details).

If the Encryption and Hash engines are disabled, commands will not require keys. If only the Encryption engine is disabled, the AES Key Field will be reserved. If only the Hash engine is disabled, the IPAD and OPAD fields will be reserved.

Note: If ENC_AG[3:0] is set to XTS, Hash_OP[3:0] cannot be set to any MAC operation. The 512 bit AES-XTS key is not permitted in this key format.

The key format shown in Figure 3-15 is used for HMAC and SSL3.0-MAC (SHA256 or MD5) operations. This format includes the AES key, IPAD and OPAD and requires 104 bytes.

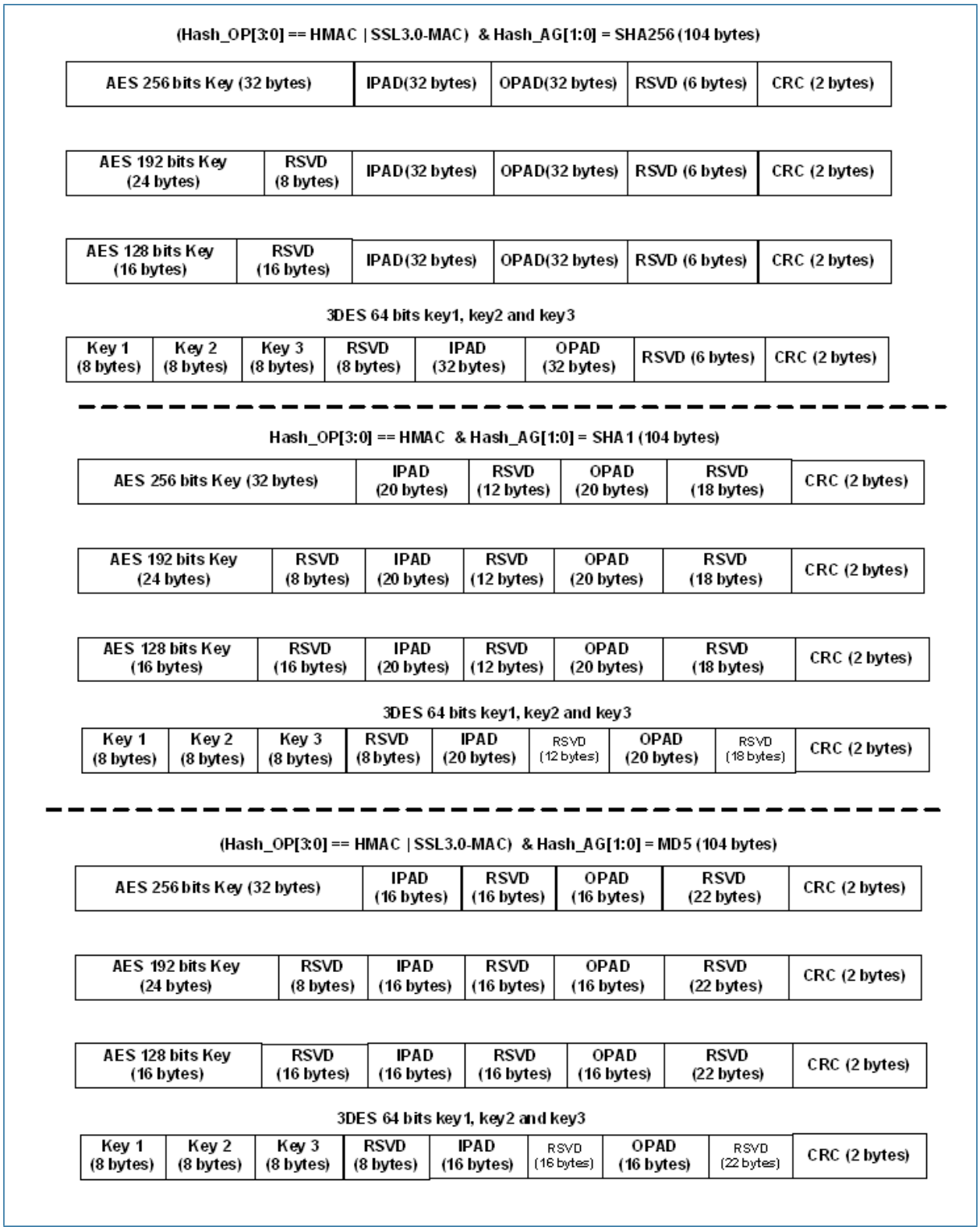


Figure 3-15. Key Format 2

Key Format 3: SSL3.0 MAC (SHA1)

If both the Encryption and Hash engines are enabled and Hash_OP[3:0] is set to "SSL3.0 MAC (SHA1)", the Hash engine will calculate the SSL 3.0 MAC and requires the MAC Key. Please refer to Section 3.1.3.9, "MAC Operations" for detail.

If both the Encryption and Hash engines are disabled, commands will not require keys. If only the Encryption engine is disabled, the AES key field will be reserved. If only the Hash Engine is disabled, the MAC key field will be reserved.

The key format shown in Figure 3-16 is used for SSL3.0 MAC (SHA1) operations. This format includes the AES key and the MAC key and requires 72 bytes.

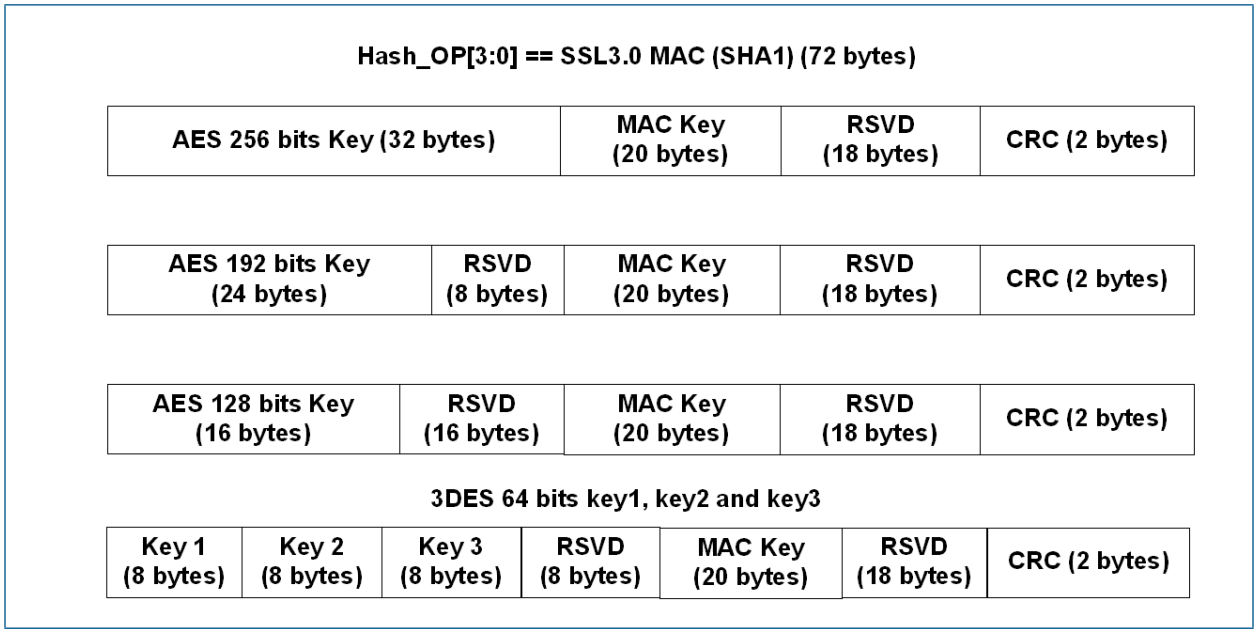


Figure 3-16. Key Format 3

Key Format 4: XCBC-MAC

If both the Encryption and Hash engines are enabled and Hash_OP[3:0] is set to "XCBC-MAC", the Hash engine will calculate the XCBC-MAC and requires a 128 bit XCBC-MAC Key.

If both the Encryption and Hash engines are disabled, commands will not require keys. If only the Encryption engine is disabled, the AES key field will be reserved. If only the Hash Engine is disabled, the XCBC-MAC key field will be reserved.

The key format shown in Figure 3-17 is used for XCBC-MAC operations. This format includes the AES key and the XCBC-MAC key and requires 72 bytes.

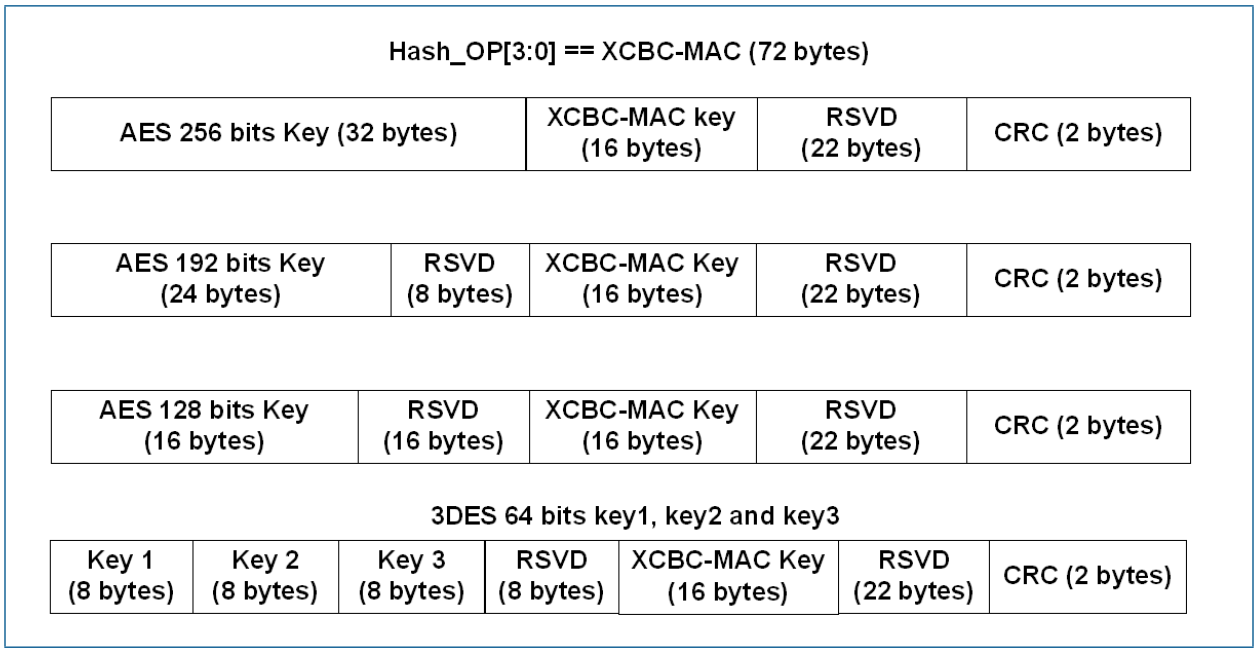
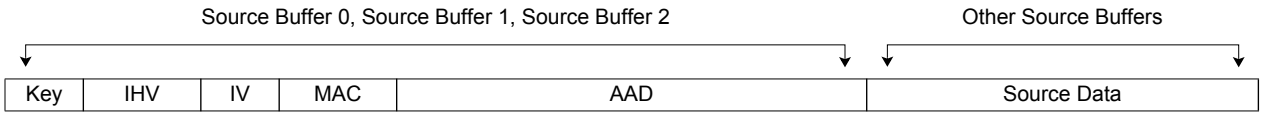


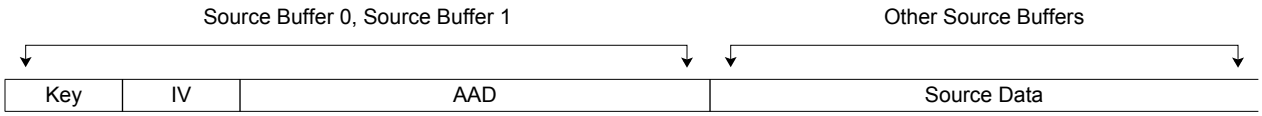
Figure 3-17. Key Format 4

3.1.3.5 Data Stream Information Fields

For some operations, the data stream will contain a mixture of the IHV, MAC, Key, IV and AAD information fields. The sequence for these information fields in the source buffer is: Key, IHV, IV, MAC, and AAD.



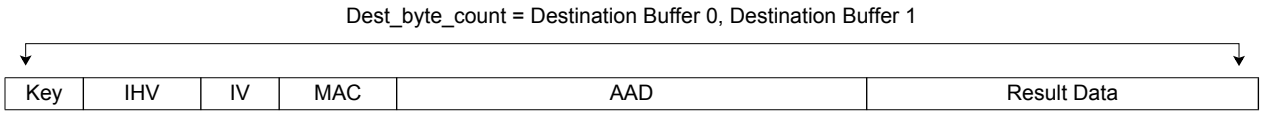
If a command does not require all fields, those fields may be removed from the data stream sequence. For example, a "Compression + encryption" command only requires the Key, IV and AAD information fields.



The 820x DMA will extract the information fields from the data stream and send them to the appropriate engines on the information bus. Please note the actual data stream sent into the 820x's data processing channels does NOT include these information fields. When calculating the "Head Count" and "Source Count" for the command structure, the host software should exclude the information fields.

All information bus fields except the AAD are aligned on an 8-byte boundary. Therefore, the length of the source buffer with information fields are 8-byte aligned, except for a source buffer that only contains the AAD. To simplify the software design, the input AAD is byte aligned, but the output AAD is 8-byte aligned.

The 820x will write all information fields to the host destination buffer if the DIR bit in the Desc_cmd_base descriptor is set to one. The host software can then determine if any of the information fields have been corrupted. All information fields are located in the header of the first destination buffer as shown in the example result data stream below. Dest_byte_count includes all resultant output from the 820x.



Source buffers that contain information fields also can be handled in a scatter-gather scheme to facilitate software packet processing. For example, the key field is a session related parameter, while the other information fields are packet related parameters. The host software can write the key into source buffer 0, setting the value for the whole session. The other information fields, whose values change according to the packet, can be saved into the remaining source buffers. In this way, the host software will not need to copy the key repeatedly to host memory for the hundreds of packets for that session.

3.1.3.6 Hash Buffer

The Hash buffer is a dedicated buffer for storing hash values. The 820x will write to the hash buffer before writing the command result entry into the destination buffer and the result ring. The host software can be assured that when a command result is available, the hash value for that command is also available.

The Hash buffer is actually the first destination buffer. If a command is hash related, the first destination buffer will be used to store only the hash values; all destination data will be written to host memory using the second destination buffer even if there is space remaining in the hash buffer.

The hash buffer starting address and length must be aligned to 32-bytes, and for kernel mode applications must be physically contiguous.

Every entry in the hash buffer is 256 bits to accommodate hash values of SHA-256, SHA-1, MD5 and all MACs, therefore the hash buffer size must be a multiple of 256 bits. The hash entry format is shown in Figure 3-18.

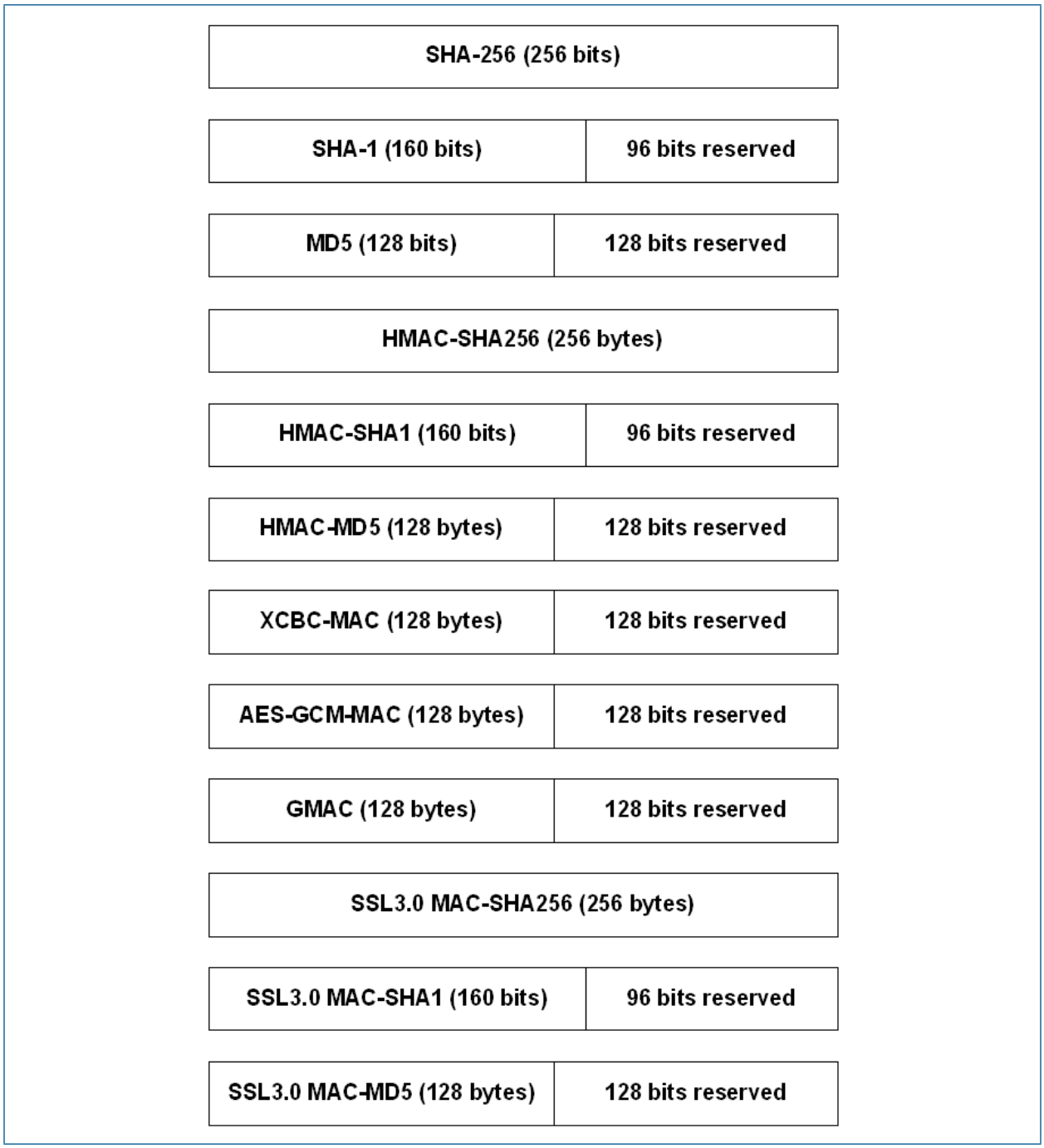


Figure 3-18. Hash Entry Format

The first entry is always used to store the file hash chaining value or the MAC value; the other entries are used to sequentially store the slice hash values.

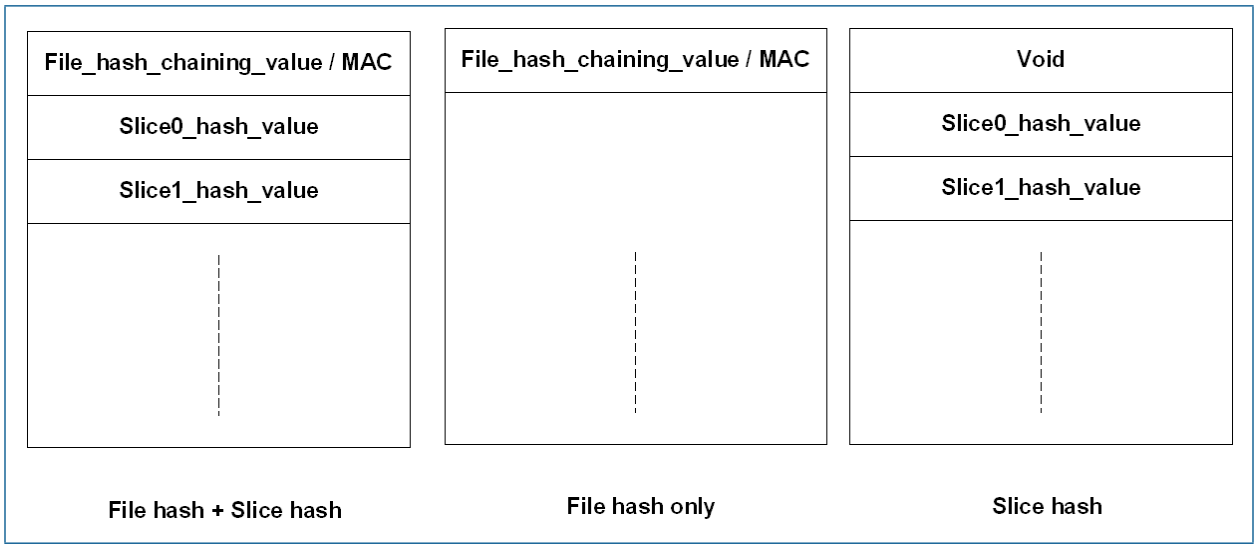


Figure 3-19. Hash Buffer Format

3.1.3.7 IPAD & OPAD for HMAC & SSL3.0-MAC (SHA256 | MD5)

SSL 3.0 MAC is a variation of the HMAC algorithm:

HMAC

MAC = H(K xor Pad2 || H(K xor Pad1 || M))

SSL 3.0 MAC

MAC = H(K || Pad2 || H(K || Pad1 || Seq. No. || Type || Length || Application Data))

The 820x uses an improved method to calculate the HMAC for better performance using the IPAD and OPAD:

820x HMAC

MAC(K,M) = H(OPAD, H(IPAD,M))

The host software computes the IPAD and OPAD and writes the values to the 820x as the key. The IPAD, derived from H(K xor Pad2) and the OPAD, derived from H(K xor Pad1) are easily generated in software using a single 512 bit block hash.

This improved method may also be applied to SSL3.0 MAC so that the IPAD is derived from H(K || Pad1) and the OPAD is derived from H(K || Pad2), the difference being the XOR and ||. If the host software works in SSL3.0 MAC mode, the input data should be "Seq. No. || Type || Length || Application Data".

However, there is a special case for SSL 3.0 MAC using the SHA-1 algorithm. The designer could easily make a mistake if the SHA-1 length of (K || Pad) is not 512 bits, and thus not reuse the HMAC flow.

The recommended approach for computing the HMAC and SSL 3.0 MAC is:

- For HMAC operations, the host software should pre-compute the IPAD and OPAD using H(K xor Pad1) and H(K xor Pad2) and then write the data to the 820x.
- For SSL3.0-MAC (MD5 or SHA-256) operations, the host software should pre-compute the IPAD and OPAD using H(K || Pad1) and H(K || Pad2) and then write "Seq. No. || Type || Length || Data" to the 820x.
- For SSL3.0-MAC (SHA-1) operations, the host software should not pre-compute the IPAD and OPAD and write K directly to the 820x using "Seq. No. || Type || Length || Data".

3.1.3.8 AES-GCM and GMAC Operations

AES-GCM operation is different from other AES operations. The output of an AES-GCM operation has two parts: a cipher text whose length is identical to the plain text, and an authentication tag.

The cipher text is an output of the encryption engine, and the authentication tag is an output of the hash engine. When performing AES-GCM operations, the host software must enable both the encryption and hash engines.

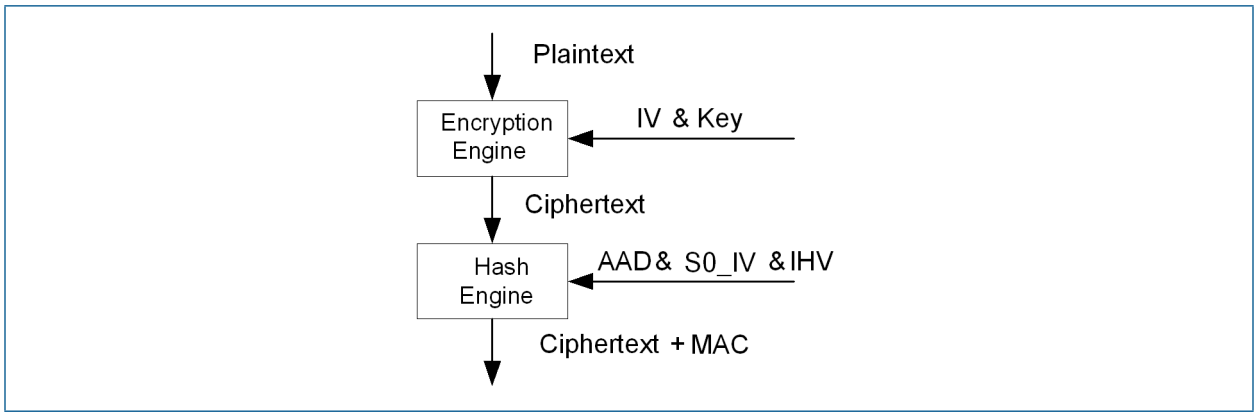


Figure 3-20. AES-GCM Implementation Illustration

GMAC is a special case of AES-GCM mode in which the cipher text size equals zero. When the 820x performs AES-GCM authenticated encryption, the Encryption engine performs the encryption operation and then the hash engine performs the authentication operation. For GMAC mode, the authentication occurs in the hash engine, using the AAD as the input data.

3.1.3.9 MAC Operations

The 820x Hash engine supports stateless and stateful MAC operations. The MAC operation is complex due to the number of parameters and operations, and the cooperation required between the 820x and the software. Table 3-3 identifies the parameters that must be set for different types of MAC operations.

Table 3-3. MAC Operation Table (Sheet 1 of 2)

MAC Type	FB	LB	IPAD	OPAD	SO_IV	IHV	AES_GCM Key	SSL 3.0-MAC Key	XCBC-MAC Key	File Size	Data	State
HMAC	1	1	√	√							Arbitrary length	Stateless
	1	0	√								512-bit aligned	First Block
	0	0				√					512-bit aligned	Middle Block
	0	1				√				√	Arbitrary length	Last Block
XCBC-MAC	1	1							√		Arbitrary length	Stateless
	1	0							√		128-bit aligned	First Block
	0	0				√			√		128-bit aligned	Middle Block
	0	1				√			√	√	Arbitrary length	Last Block
GCM-MAC	1	1			√		√				Arbitrary length	Stateless
	1	0					√				128-bit aligned	First Block
	0	0				√	√				128-bit aligned	Middle Block
	0	1			√	√	√			√	Arbitrary length	Last Block
GMAC	1	1			√						Arbitrary length	Stateless
	1	0									128-bit aligned	First Block
	0	0				√					128-bit aligned	Middle Block
	0	1			√	√				√	Arbitrary length	Last Block
SSL3.0-MAC (SHA256 MD5)	1	1	√	√							Arbitrary length	Stateless
	1	0	√								512-bit aligned	First Block
	0	0				√					512-bit aligned	Middle Block
	0	1		√		√				√	Arbitrary length	Last Block

Table 3-3. MAC Operation Table (Sheet 2 of 2)

MAC Type	FB	LB	IPAD	OPAD	SO_IV	IHV	AES_GCM Key	SSL 3.0-MAC Key	XCBC-MAC Key	File Size	Data	State
SSL3.0-MAC (SHA1)	1	1				√		√		X	Arbitrary length	Stateless
	1	0				√		√		X	512-bit aligned	First Block
	0	0				√				X	512-bit aligned	Middle Block
	0	1				√		√		√	Arbitrary length	Last Block

3.1.3.10 IPsec Packet Processing

The 820x has been optimized to maximize IPsec packet processing performance.

The 820x has four processing engines: compression, encryption, pad, and hash. The processing order depends on whether the operation is encode or decode and the position of the hash engine using the PS bits in the hash descriptor. Please refer to Chapter 4, "Data Flow" for a detailed description of the operation sequence.

Each processing engine may be enabled or disabled. A disabled processing engine simply passes data forward to the next processing unit without altering the data or modifying the context.

Each processing engine has two counters: a Header Counter and a Source Counter. The Header Counter is used to determine how many bytes the processing engine will pass through before processing the data. This counter is useful to skip header fields in many network communication protocols. The Source Counter determines how many bytes will be processed by that processing engine. The Source Counter may be programmed to start from the first byte to be processed, or after the last byte processed by the previous processing engine. This flexibility takes into account the variable output sizes produced by the compression and pad engines. Once the source count of a processing engine reaches zero, any remaining bytes in the input data stream will be passed through the processing engine without altering the data or modifying the context.

Figure 3-21 shows an example of how the 820x would processes an IPsec packet in conjunction with the host software. This IPsec example illustrates how to apply IPPCP and ESP in tunnel mode.

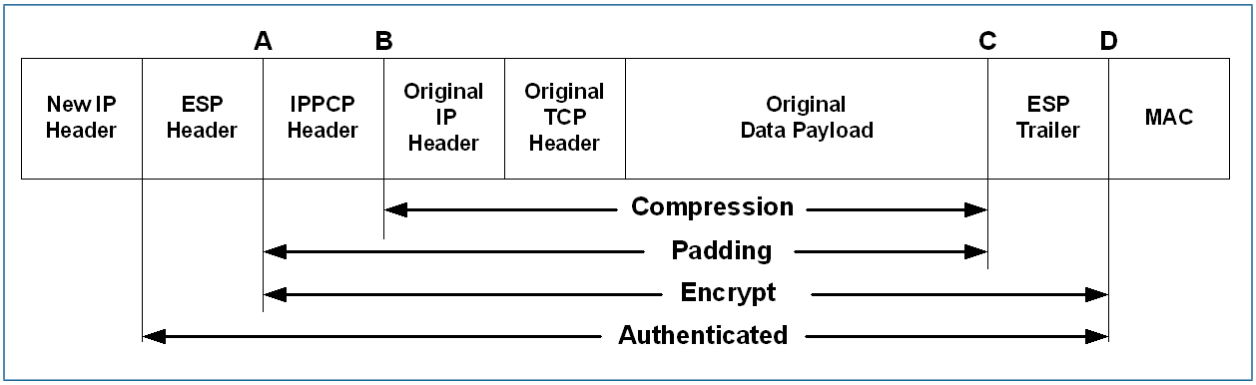


Figure 3-21. IPsec Example: Applying IPPCP and ESP in Tunnel Mode

The 820x command structure controls each processing engine by specifying which part of the block each engine is working on. Through careful selection of the Header Counter and Source Count values in each processing engine, relatively complex patterns of compression, encryption, padding and authentication can be performed. Table 3-4 below shows the values that the host software would set for processing this example packet.

Table 3-4. Command Structure Values for IPsec Example

Command Structure Field	Length / Value
Desc_cmd_cmp Descriptor	
CMP_Header_Count	B
CMP_CM	0
CMP_Source_Count	C-B
Desc_cmd_pad Descriptor	
Pad_Header_Count	A
Pad_CM	1
Pad_Source_Count	0
Desc_cmd_enc Descriptor	
ENC_Header_Count	A
ENC_CM	1
ENC_Source_Count	0
Desc_cmd_hash Descriptor	
Hash_Header_Count	0
PS	3
Hash_CM	1
Hash_Source_Count	0

For this example, the host software would construct the source data as:

ESP Header + IPPCP Header + Original IP Header + Original TCP Header + Original Data Payload

The 820x DMA would fetch the data and distribute it to the appropriate processing engines. Because the header of this data block is not to be compressed, the compression engine should skip over the ESP and IPPCP headers. Setting CMP_Header_Count to "B" represents the number of bytes of data the compression engine would pass through. The compression engine would begin compressing the Source Count bytes, "C-B".

The Pad, Encryption and Hash engines work in a similar fashion, except for the fact that in this example they have been set with CountMode of 1 and a SourceCount of 0. Each of these processing engines passes through the number of bytes in the Header Counter, and then process the same amount of data as processed by the previous engine.

3.1.4 Free Pool Ring

The host software may use the 820x Free Pool to store result data if all the command's destination buffers are consumed in order to avoid an overflow error. Figure 3-22 illustrates the Free Pool Ring.

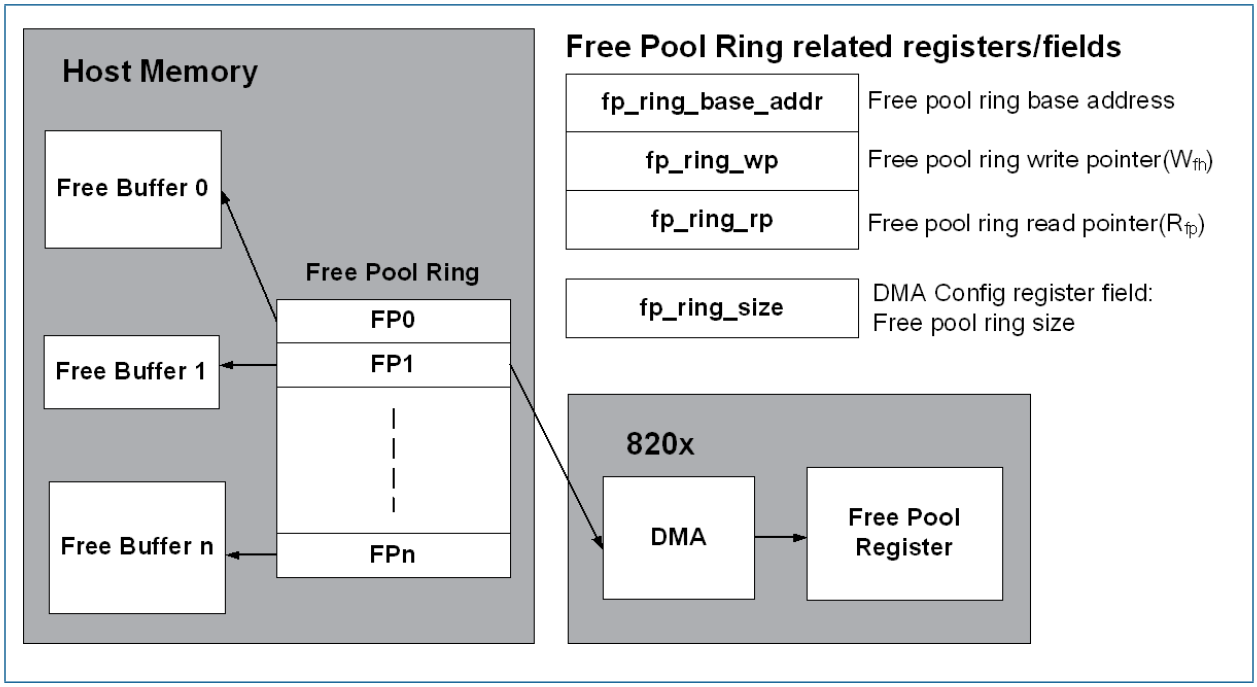
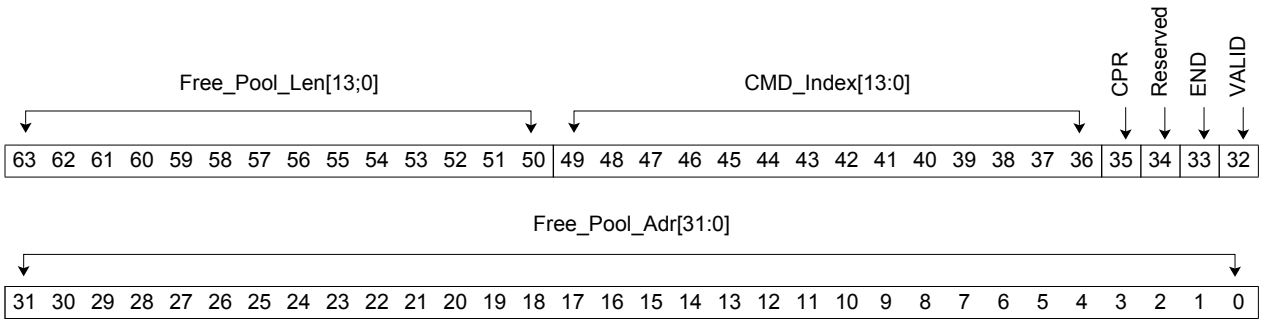


Figure 3-22. Free Pool Ring

The 820x does not maintain a "full" bit for the Free Pool Ring; it is the responsibility of the host not to overflow the free pool ring.

The Free Pool has same the format for both 32-bit or 64-bit addressing:



Field Name	Bits	Default	Description
Free_Pool_Len[13:0]	63:50	0	Free Pool Length. This field indicates the size of the block defined by the free pool descriptor. The size of the free pool descriptor buffer is 8-bytes.
CMD_Index[13:0]	49:36	0	Command Index. The index number of the command that used the Free Pool entry.
CPR	35	0	Command Pointer Ring. This bit identifies which command pointer ring used the free pool entry. 0 Free pool entry is from a command in Command Pointer Ring 0 1 Free pool entry is from a command in Command Pointer Ring 1
Reserved	34	0	Reserved
END	33	0	End. The command that used this entry is finished.
VALID	32	0	Entry Valid. This bit indicates that the free pool entry is valid. The host software will set this bit when writing a new free pool entry. The 820x will clear this bit after reading the free pool entry. 0 Free pool entry not valid 1 Free pool entry valid
Free_Pool_Adr[31:0]	31:0	0	Free Pool Buffer Starting Address. The starting physical address of the descriptor free pool. The Free Pool Entry format is the same for both 32-bit and 64-bit addressing. Even if operating in a 64-bit physical addressing environment, the 820x free pool physical addresses must be located in the first 4GB (0 - 0xffffffff). This field must be aligned on an 8 byte boundary.

Both the host and the 820x may read from and write to the Free Pool Ring; the host writes to the ring at initialization, and the 820x reads from the ring when needed. After the 820x has used a free pool entry, it will clear the Valid bit for that entry to inform the host that the entry was used by a Channel Manager. After the host has finished processing a

command and finds that the command used the free pool by reading the FP bit in the result descriptor, it should read the Free Pool Ring to determine how many entries were used by that command. The host can then read the result data for that command from the Free Pool buffers using the free pool address and length. Figure 3-24 illustrates a free pool ring example.

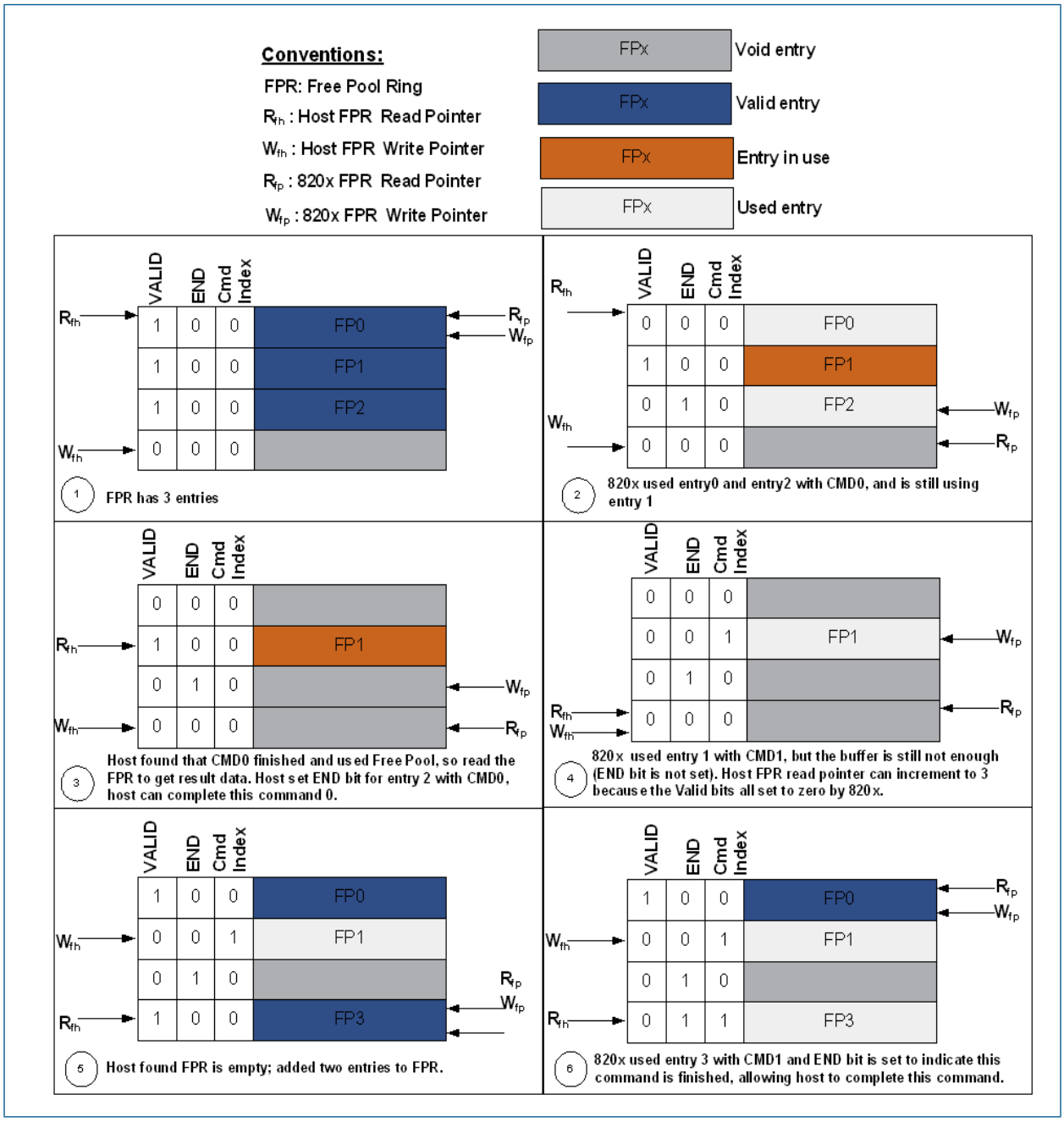


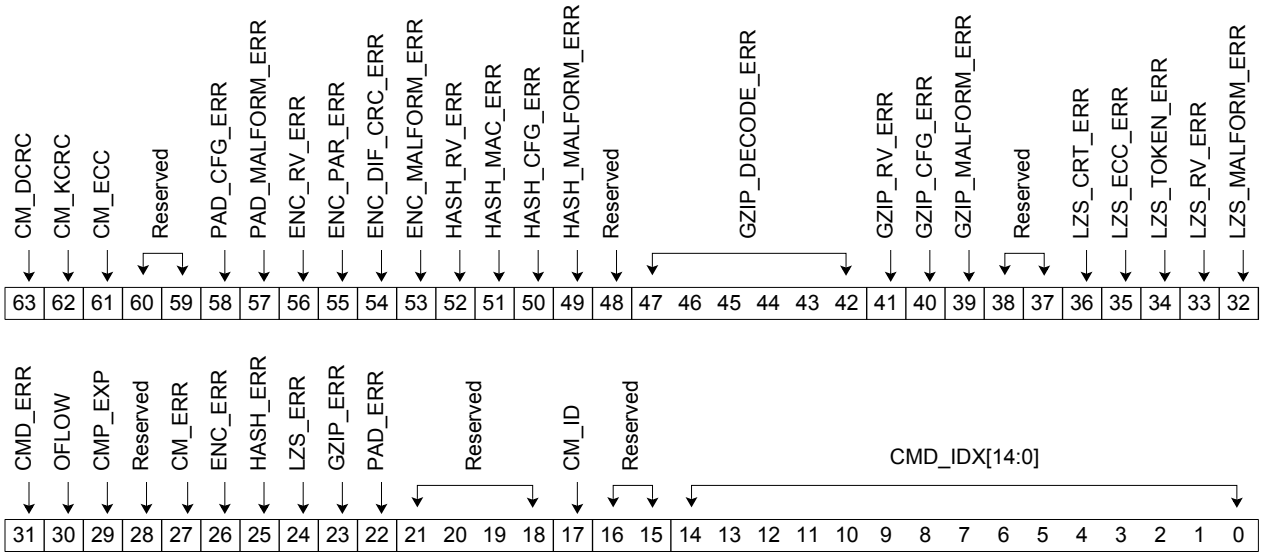
Figure 3-23. Free Pool Usage Example

3.2 Result Ring

The result ring contains a completed command's resultant data pointers and execution status (successful completion or errors occurred). The 820x writes to the result ring and the host reads from the result ring. The 820x will write to the result ring when a command completes and then update its result ring write pointer appropriately.

The size of the result ring is always identical to the size of command pointer ring. The result ring must be aligned to an 8-byte boundary and reside entirely in a physically contiguous range of memory.

In general in the case of an error, bits [31:17] of a result ring entry summarize the source of the error, while bits [63:32] provide a more detailed identification of the error. The error bits remain set until cleared by the host software.



Field Name	Bits	Default	Description
CM_DCRC	63	0	Channel Manager Data CRC error. 0 Channel Manager did not detect data CRC error 1 Channel Manager detected data CRC error
CM_KCRC	62	0	Channel Manager Key CRC error. 0 Channel Manager did not detect key CRC error 1 Channel Manager detected key CRC error
CM_ECC	61	0	Channel Manager ECC/Parity error. 0 Channel Manager did not detect ECC/Parity error on this channel 1 Channel Manager detected ECC/Parity error on this channel
Reserved	60:59	0	Reserved.

Field Name	Bits	Default	Description
PAD_CFG_ERR	58	0	Pad Engine Software Configuration Error. 0 Pad Engine did not detect software configuration error 1 Pad Engine detected software configuration error
PAD_MALFORM_ERR	57	0	Pad Engine Malformed Packet Error. 0 Pad Engine did not detect malformed packet error 1 Pad Engine detected malformed packet error
ENC_RV_ERR	56	0	Encryption Engine Real Time Verification Error. 0 Encryption Engine did not detect real time verification error 1 Encryption Engine detected real time verification error
ENC_PAR_ERR	55	0	Encryption Engine Parity Error. 0 No parity error in 3DES key 1 Encryption engine detected parity error in 3DES key
ENC_DIF_CRC_ERR	54	0	Encryption Engine DIF CRC Error in AES-XTS operation. 0 Encryption Engine did not detect DIF CRC error in AES-XTS operation 1 Encryption Engine detected DIF CRC error in AES-XTS operation
ENC_MALFORM_ERR	53	0	Encryption Engine Malformed Packet Error. 0 Encryption Engine did not detect malformed packet error 1 Encryption Engine detected malformed packet error
HASH_RV_ERR	52	0	Hash Engine Real Time Verification Error or HMAC Error. 0 Hash engine did not detect real time verification or HMAC error 1 Hash engine detected real time verification or HMAC error
HASH_MAC_ERR	51	0	Hash Engine MAC Check Error. 0 Hash engine did not detect MAC check error 1 Hash engine detected MAC check error
HASH_CFG_ERR	50	0	Hash Engine Software Configuration Error. 0 Hash Engine did not detect software configuration error 1 Hash Engine detected software configuration error

Field Name	Bits	Default	Description
HASH_MALFORM_ERR	49	0	Hash Engine Malformed Packet Error. 0 Hash Engine did not detect malformed packet error 1 Hash Engine detected malformed packet error
Reserved	48	0	Reserved.
GZIP_DECODE_ERR [5:0]	47:42	0	GZIP Error. Please refer to Table 3-5 for the decoding of this field.
GZIP_RV_ERR	41	0	GZIP Engine Real Time Verification Error. 0 GZIP engine did not detect real time verification error 1 GZIP engine detected real time verification error
GZIP_CFG_ERR	40	0	GZIP Engine Software Configuration Error. 0 GZIP Engine did not detect software configuration error 1 GZIP Engine detected software configuration error
GZIP_MALFORM_ERR	39	0	GZIP Engine Malformed Packet Error. 0 GZIP Engine did not detect malformed packet error 1 GZIP Engine detected malformed packet error
Reserved	38:37	0	Reserved.
LZS_CRT_ERR	36	0	LZS Engine Corruption Error. 0 LZS engine did not detect EOR/multiple record error 1 LZS engine detected EOR/multiple record error
LZS_ECC_ERR	35	0	LZS Engine ECC Error. 0 LZS engine did not detect ECC error in decode operation 1 LZS engine detected ECC error in decode operation
LZS_TOKEN_ERR	34	0	LZS Engine Token Error. This bit will be set if the LZS engine identifies an undefined token in the LZS engine data stream. A token at the end of a compressed block normally indicates the end of the compression data stream. 0 LZS engine did not detect token error 1 LZS engine detected token error
LZS_RV_ERR	33	0	LZS Engine Real Time Verification/CRC Error. real time verification error when encode, identifies CRC error when decode 0 LZS Engine did not detect real time verification error for a encode operation, or a CRC error for a decode operation 1 LZS Engine detected real time verification error for a encode operation, or a CRC error for a decode operation

Field Name	Bits	Default	Description
LZS_MALFORM_ERR	32	0	LZS Engine Malformed Packet Error. 0 LZS Engine did not detect malformed packet error 1 LZS Engine detected malformed packet error
CMD_ERR	31	0	Command Error. This bit summarizes the command error status. It will be set if any of the error bits, excluding the overflow error bit, in this entry are set. 0 No error occurred for this command 1 Error occurred for this command
OFLOW	30	0	Overflow. This bit indicates if an overflow error occurred in the destination data buffer. 0 No overflow error occurred for this command 1 Overflow error occurred for this command
CMP_EXP	29	0	Compression Expansion bit. This bit defines how the compression expansion is calculated. 0 Len(cmp) + head_size < len(raw) 1 Len(cmp) + Head_Size >= len(raw)
RSVD	28	0	Reserved.
CM_ERR	27	0	Channel Manager Error. This bit indicates if any error occurred in either channel manager. 0 No error occurred in either channel manager 1 Error occurred in one or both channel managers
ENC_ERR	26	0	Encryption Engine Error. This bit indicates if an error occurred in the Encryption engine. 0 No Encryption engine error occurred for this command 1 Encryption engine error occurred for this command
HASH_ERR	25	0	Hash Engine Error. This bit indicates if an error occurred in the hash engine. 0 No hash engine error occurred for this command 1 Hash engine error occurred for this command
LZS_ERR	24	0	LZS Engine Error. This bit indicates if an error occurred in the LZS engine. 0 No LZS engine error occurred for this command 1 LZS engine error occurred for this command

Field Name	Bits	Default	Description
GZIP_ERR	23	0	GZIP Engine Error. This bit indicates if an error occurred in the GZIP engine. 0 No GZIP engine error occurred for this command 1 GZIP engine error occurred for this command
PAD_ERR	22	0	Pad Engine Error. This bit indicates if an error occurred in the Pad engine. 0 No Pad engine error occurred for this command 1 Pad engine error occurred for this command
Reserved	21:18	0	Reserved.
CM_ID	17	0	Channel Manager Identifier. This bit identifies which Channel Manager reported the error. 0 Channel Manager 0 reported the error 1 Channel Manager 1 reported the error
Reserved	16:15	0	Reserved.
CMD_IDX	14:0	0	Command Index. The command index field contains a Valid bit and the index number of the completed command in the result ring. The bit position of the Valid bit depends on the command ring size, as described below. The Valid bit toggles each time the write pointer wraps the result ring. For a command ring size of 16K: bit 14 Valid bit if command ring size is 16K bit 13:0 Command index of completed command For a command ring size of 8K: bit 13 Valid bit if command ring size is 8K bit 12:0 Command index of completed command For a command ring size of 4K: bit 12 Valid bit if command ring size is 4K bit 11:0 Command index of completed command A similar pattern can be applied for command ring sizes < 4K.

Table 3-5 decodes the GZIP errors for the field GZIP_DECODE_ERR[5:0]. These errors indicate that the data sent to the 820x has been corrupted.

Table 3-5. GZIP Decode Error Table

Error Code Bits	Code Definition	6-bit signed Value	Comment
0X_XXXX	No error	0 -31	
11_1111	SYS_GENERAL_ERROR	-1	Not used.
11_1110	SYS_BAD_MAGIC_NUMBER	-2	Incorrect magic number
11_1101	SYS_BAD_GZIP_HEADER	-3	Bad compression mode or flags
11_1100	SYS_BAD_FILE_NAME	-4	Extra length greater than file length
11_1011	SYS_BAD_FCOMMENT	-5	Extra length greater than file length
11_1010	SYS_BAD_EXTRA	-6	Extra length greater than file length
11_1001	SYS_BAD_LENDIS	-7	Bad length or distance
11_1000	SYS_STC_INVALID_SYMBOL	-8	Static, invalid OP field
11_0111	SYS_STC_INVALID_DISTANCE	-9	Static, invalid OP field
11_0110	SYS_STO_LEN_MISMATCH	-10	STORED LEN (bypass mode) does not match NLEN
11_0101	SYS_BAD_BLOCK_TYPE	-11	Block type 3 detected
11_0100	SYS_BAD_NO_DYN_SUPP	-12	Attempt dynamic frame with no support. Only valid for cores built without dynamic support.
11_0011	SYS_DYN_NO_LAST_LENGTH	-13	No last length in dynamic mode
11_0010	SYS_DYN_NLEN_OR_NDIST	-14	NLEN > MAXLCODES or NDIST > MAXDCODES
11_0001	SYS_DYN_INVALID_CODE_LEN	-15	Invalid code lengths
11_0000	STS_DYN_TOO_MANY_LENGTH	-16	Invalid code lengths
10_1111	STS_DYN_INVALID_CODE_LEN	-17	Not used
10_1110	STS_DYN_TOO_MANY_LENS	-18	Not used
10_1101	STS_DYN_OVER_SUBSCRIBED	-19	Oversubscribed set of lengths
10_1100	STS_DYN_INCOMPLETE_SET	-20	Incomplete set of lengths
10_1011	STS_DYN_ENOUGH_NOT_ENOUGH	-21	Not enough space for dynamic table
10_1010	STS_DYN_UNEXPECTED_END_OF_STREAM	-22	No more data present in stream
10_1001 to 10_0000	Not defined	-23 to -32	Not used

The Valid bit in the CMD_IDX[14:0] field informs the host software that the corresponding command has finished and indicates that there are valid entries in the result ring. The value of the Valid bit toggles every time the result ring is completely used, i.e., the 820x write pointer has wrapped the ring. For example, for the first pass through the result ring the value of the Valid bit indicating valid entries is one. After the 820x's write pointer wraps

around the result ring, the Valid bit will be zero, and so on for every complete pass through the result ring. During initialization, the host software should write a zero to the Valid bit. This simple mechanism is sufficient to allow the host to differentiate unambiguously whether an entry in the result ring has been updated.

Figure 3-24 shows an example of how the result ring and command pointer ring are used to process a command and its results. To simplify the illustration, this example shows a command pointer ring only with 4 entries, but the actual minimum command pointer ring size is 16 entries.

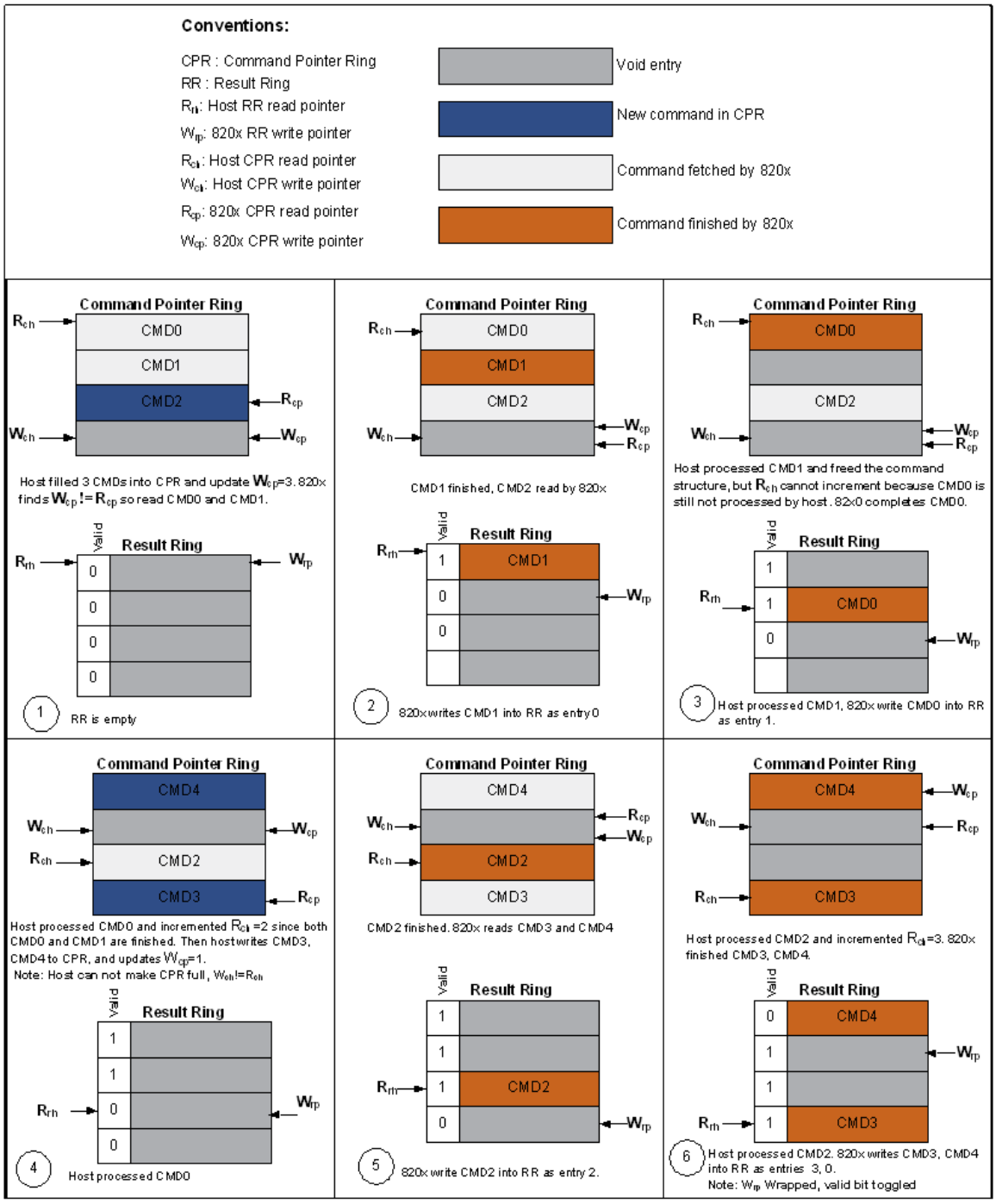


Figure 3-24. Result Ring Example

3.3 Command Operation Sequence

The 820x has a high performance DMA engine that supports two command pointer rings using a round-robin arbitration scheme. Figure 3-25 shows how the 820x and the host software work together to process a command. The flow is described in the figure below.

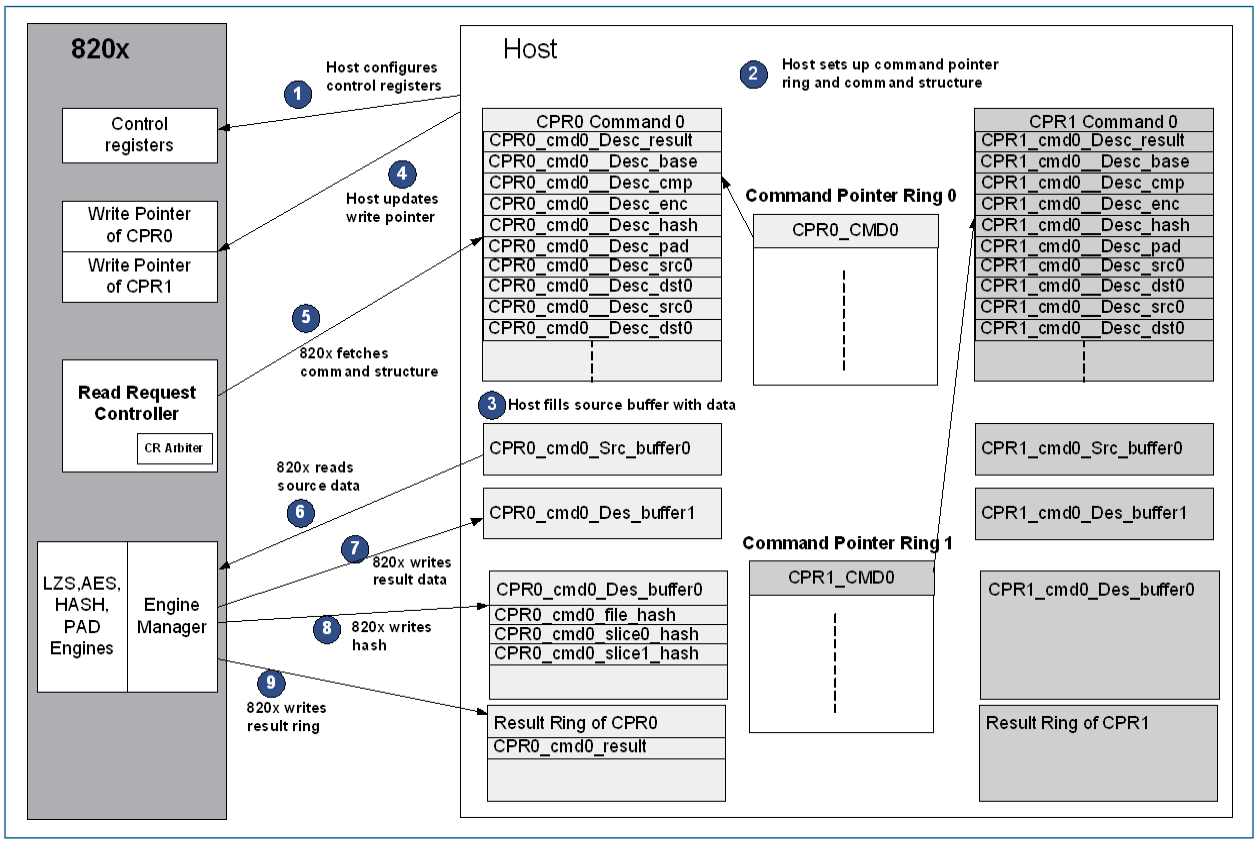


Figure 3-25. Command Process Flow Example

1. Host configures the 820x control registers (Refer to Chapter 6 for details).

2. Host sets up Command Pointer Ring 0 and Command Pointer Ring 1 and writes the command structures into host memory.

3. Host writes the source data into the source buffers.

4. Host writes the index number of the commands into the Command Pointer Ring using the 820x Command Pointer Ring Write Pointer (Wcp) register and updates its own Command Pointer Ring Write Pointer (Wch). The 820x maintains a Command Pointer Ring Read Pointer (Rcp) register which may be read by the host at any time. It is the responsibility of the host not to overflow the Command Pointer Ring by ensuring that the write pointer never advances to meet the read pointer. If several commands are ready for submission at the same time, the host may build the command structures for all these commands and then issue a single write to the write pointer register to improve efficiency.

5. If dual command pointer ring mode is enabled, the host should update the appropriate command write pointer of Command Pointer Ring 0 or Command Pointer Ring 1.

6. As long as the read and the write pointer registers for Command Pointer Ring 0 or Command Pointer Ring 1 are not equal, the 820x recognizes that commands are available to fetch from host memory. A command read request will be sent by the 820x Read Request Controller which will alternate sending the command to either Command Pointer Ring 0 or Command Pointer Ring 1. In this example, the command is sent to Command Pointer Ring 0.

7. The 820x reads the source descriptor to determine the location of the source data and the operation. The Compression, Decompression, Hash, and Pad engines process the data according to the control fields in the command structure.

8. The 820x writes the slice hash values and file chaining hash value to the first destination buffer if the command has Hash related operation.

9. After the Compression, Decompression, Hash, and Pad engines complete processing the data, the 820x writes the result data to the host destination buffers whose location is defined in the destination descriptor.

10. After the 820x writes the result data to host memory, it updates the appropriate command structure Result Ring. Because the 820x has two channels (each channel includes one group of Compression, Decompression, Hash, and Pad engines) which process commands in parallel, and the commands may have different sizes, it is very probable for commands to complete out of order. The Result Ring is used to keep track of the finished commands.

4 Data Flow

The 820x has four processing engines: Compression, Encryption, Pad, and Hash. Each processing engine may be enabled or disabled in the command descriptor base register (Section 3.1.2.2, "Desc_cmd_base"). A disabled processing engine simply passes data forward to the next processing engine without altering the data or modifying the context. The Pad engine is typically used for packet processing applications to pad or remove data; storage applications may disable the Pad engine to conserve power.

The order of processing depends on whether the operation is in the encode or decode direction. The positions of Compression, Encryption and Pad engines are fixed, but the position of the Hash engine is configurable using Hash Engine Position PS configuration bits in the hash descriptor command structure (Section 3.1.2.5, "Desc_cmd_hash").

In addition, in the data stream sent to a processing engine, the amount of header and trailer data skipped by the engine may be adjusted according to the Source_Count and Header_Count fields in the command descriptor structures for each engine. This feature can be used to accelerate packet processing.

4.1 Encode Operations Data Flow

4.1.1 Hash Engine before Compression Engine

Figure 4-1 shows the data flow when the position of the Hash Engine is before the Compression engine and all four engines are enabled.

1. The DMA fetches "RAW" data from the host according to the command structures, and sends the data to processing channel 0 or processing channel 1.

2. In parallel, processing channels 0 and 1 begin to process the "RAW" data for separate commands.

3. The Hash engine simultaneously calculates the hash values for the "RAW" data, and passes the "RAW" data to the Compression engine.

4. The Compression engine compresses the "RAW" data, and then sends the compressed RAW data "CMP" to the Pad engine.

5. The Pad engine passes through the "CMP" data and adds "Pad" to the data stream for the Encryption Engine.

6. The Encryption engine encrypts the "CMP + Pad" data, and then sends the result, ENC (CMP + Pad), to the DMA.

7. Finally, the DMA sends the result data, ENC (CMP +Pad), from the Encryption engine and the hash values from the Hash engine to host memory according to the command structures.

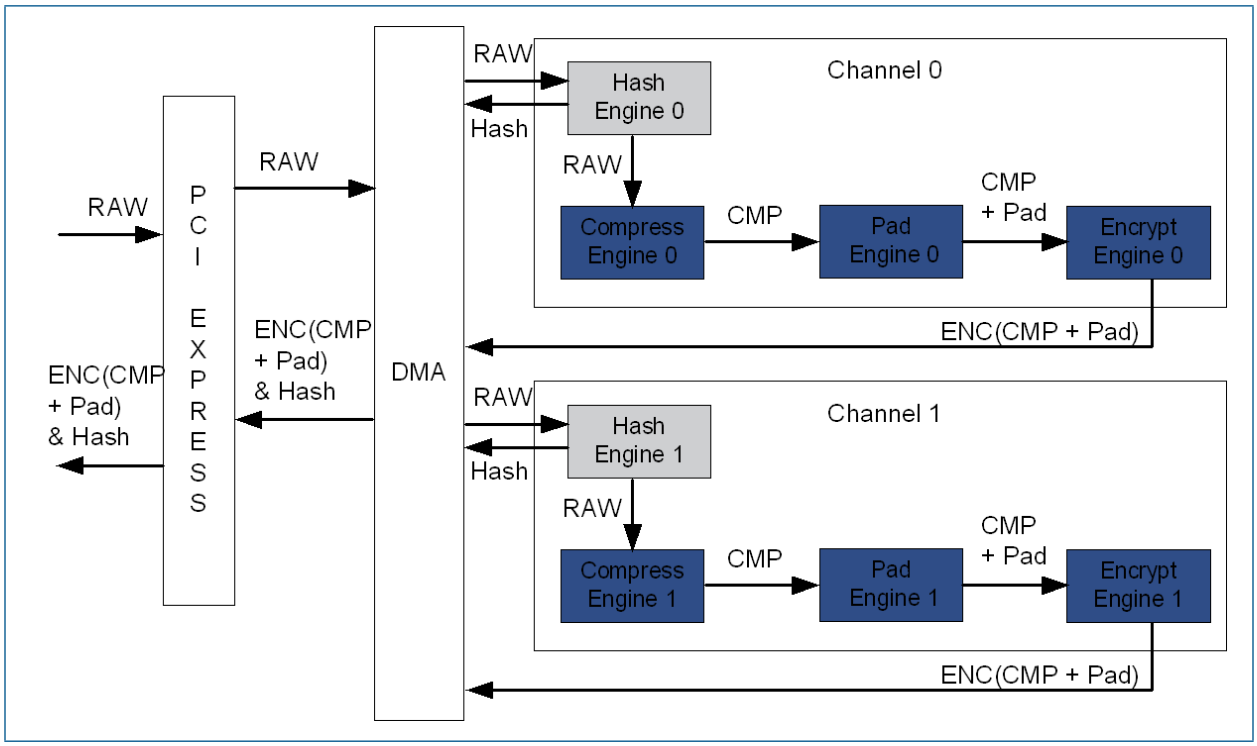


Figure 4-1. Encode Operation: Hash Engine before Compression Engine

In this figure, the Hash engine only calculates the hash values on the RAW data. The 820x can also calculate MAC values on the RAW data by setting the configuration fields appropriately in the command structure. The 820x can only calculate hash or MAC values when the Hash engine's position is before the Compression engine. In any other position, the Hash engine can only calculate MAC values.

4.1.2 Hash Engine after Compression Engine

Figure 4-2 shows the data flow when the Hash engine position is after the Compression engine and all four engines are enabled. In this configuration, the Hash engine calculates the MAC value on the CMP data instead of the RAW data. The data flow for the remaining engines is the same as described in Section 4.1.1, "Hash Engine before Compression Engine".

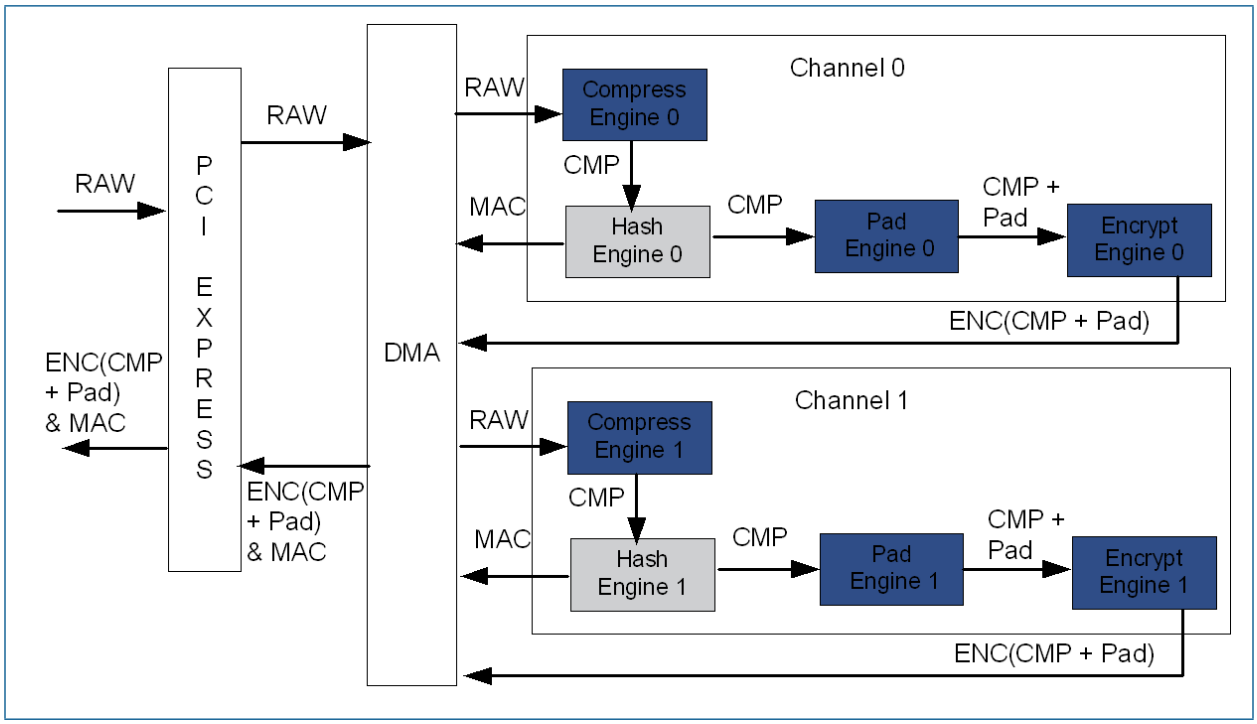


Figure 4-2. Encode Operation: Compression Engine before Hash Engine

4.1.3 Hash Engine after Pad Engine

Figure 4-3 shows the data flow when the Hash engine position is after the Pad engine and all four engines are enabled. In this configuration, the Hash engine calculates the MAC value of the compressed and padded data, CMP + Pad. The data flow for the remaining engines is the same as described in Section 4.1.1, "Hash Engine before Compression Engine".

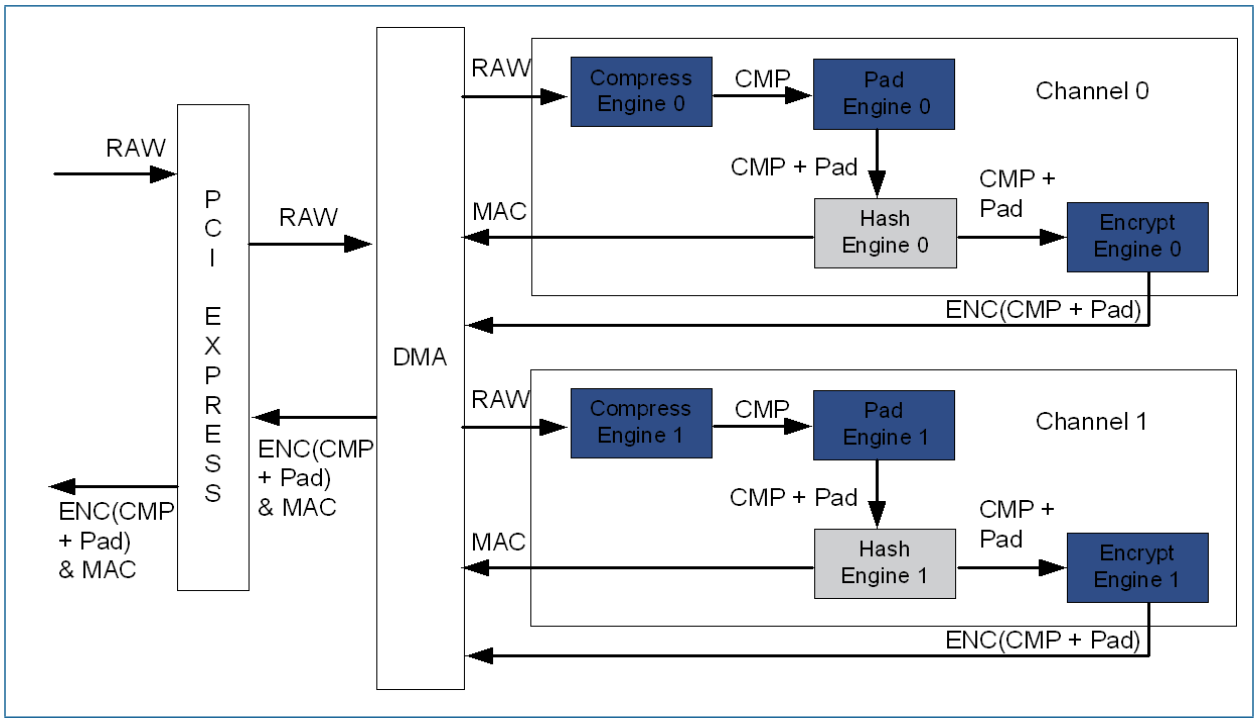


Figure 4-3. Encode Operation: Hash Engine after Pad Engine

4.1.4 Hash Engine after Encryption Engine

Figure 4-4 shows the data flow when the position of the Hash engine is after the Encryption engine and all four engines are enabled. In this configuration, the Hash engine calculates the MAC value on the encrypted, compressed data with padding, ENC(CMP + Pad). The data flow for the remaining engines is the same as described in Section 4.1.1, "Hash Engine before Compression Engine".

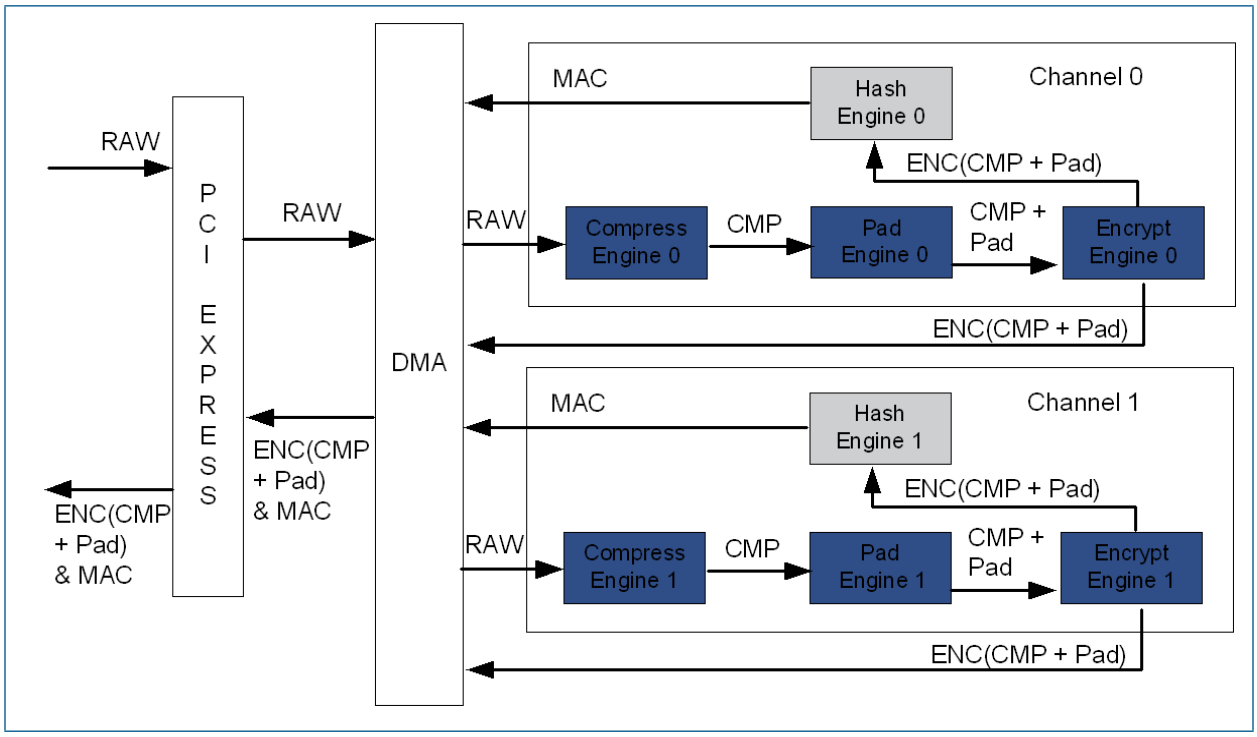


Figure 4-4. Encode Operation: Hash Engine after Encryption Engine

4.2 Decode Operations Data Flow

4.2.1 Hash Engine before Encryption Engine

Figure 4-5 shows the data flow when the position of the Hash Engine is before the Encryption engine and all four engines are enabled. The data flow is the reverse of the "Hash Engine after Encryption Engine" encode operation.

1. First, the DMA fetches the encrypted, compressed data with padding, ENC (CMP + Pad) & MAC, from the host according to the command structures, and sends the data to processing channel 0 or processing channel 1.

2. In parallel, processing channels 0 and 1 begin to process the "ENC (CMP + Pad) & MAC" data for separate commands.

3. The MAC value is sent to the Hash engine through the information bus, and the "ENC (CMP + Pad)" data is sent to the Hash engine through the data bus. Please refer to Section 3.1.3.5, "Data Stream Information Fields" for details.

4. The Hash engine calculates the MAC value of the "ENC (CMP + Pad)" data and compares the result to the MAC value on information bus to determine whether the input stream is valid, and also passes through the "ENC (CMP + Pad)" data to the Encryption engine.

5. The Encryption engine decrypts the "ENC (CMP + Pad)" data, and then sends the "CMP + Pad" data to the Pad engine.

6. The Pad engine passes through the "CMP" data to the Compression Engine, and removes the "Pad" data in the data stream.

7. The Compression engine decompresses the "CMP" data, and then sends the resulting "RAW" data to the DMA.

8. Finally, the DMA sends the result data to host memory according to command structures.

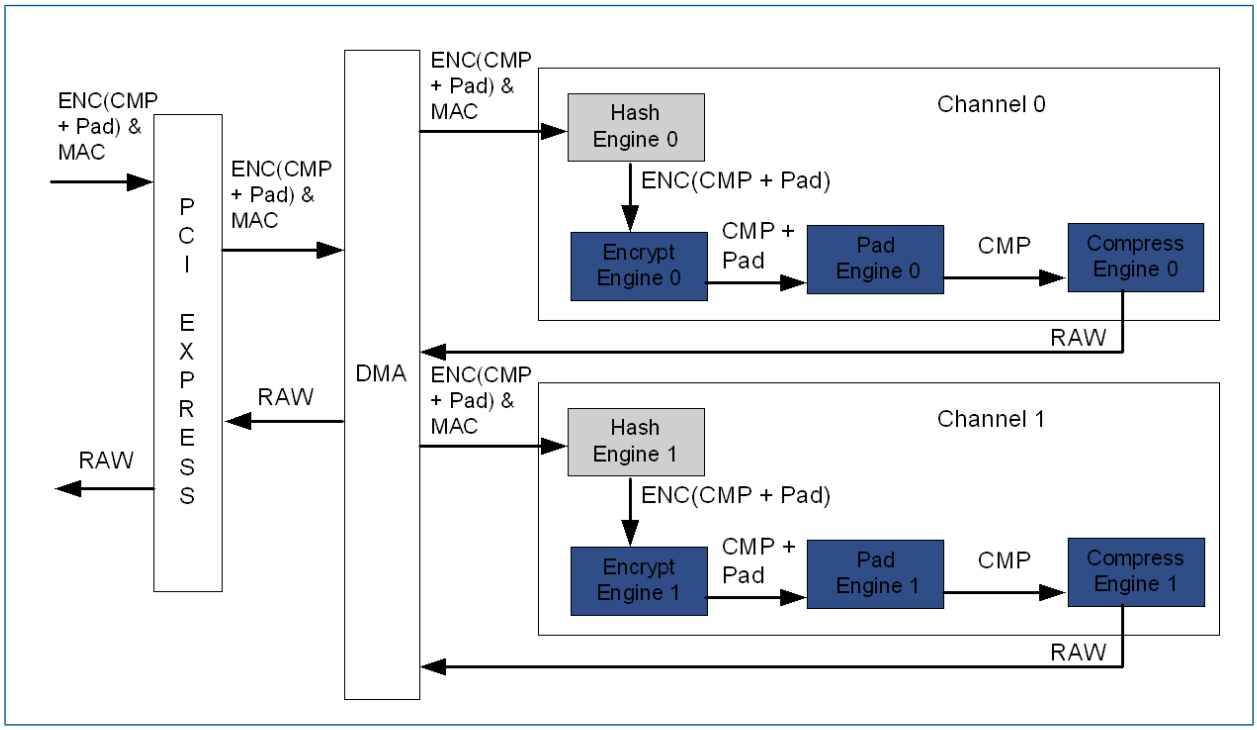


Figure 4-5. Decode Operation: Hash Engine before Encryption Engine

4.2.2 Hash Engine after Encryption Engine

Figure 4-6 shows the data flow when the position of the Hash engine is after the Encryption engine and all four engines are enabled. In this configuration, the Hash Engine calculates MAC value for the compressed data with padding, "CMP + Pad", and compares the calculated MAC value against the MAC value in the information bus. The data flow for the remaining engines is the same as described in Section 4.2.1, "Hash Engine before Encryption Engine". This data flow is the reverse of the encode operation "Hash Engine after Pad engine".

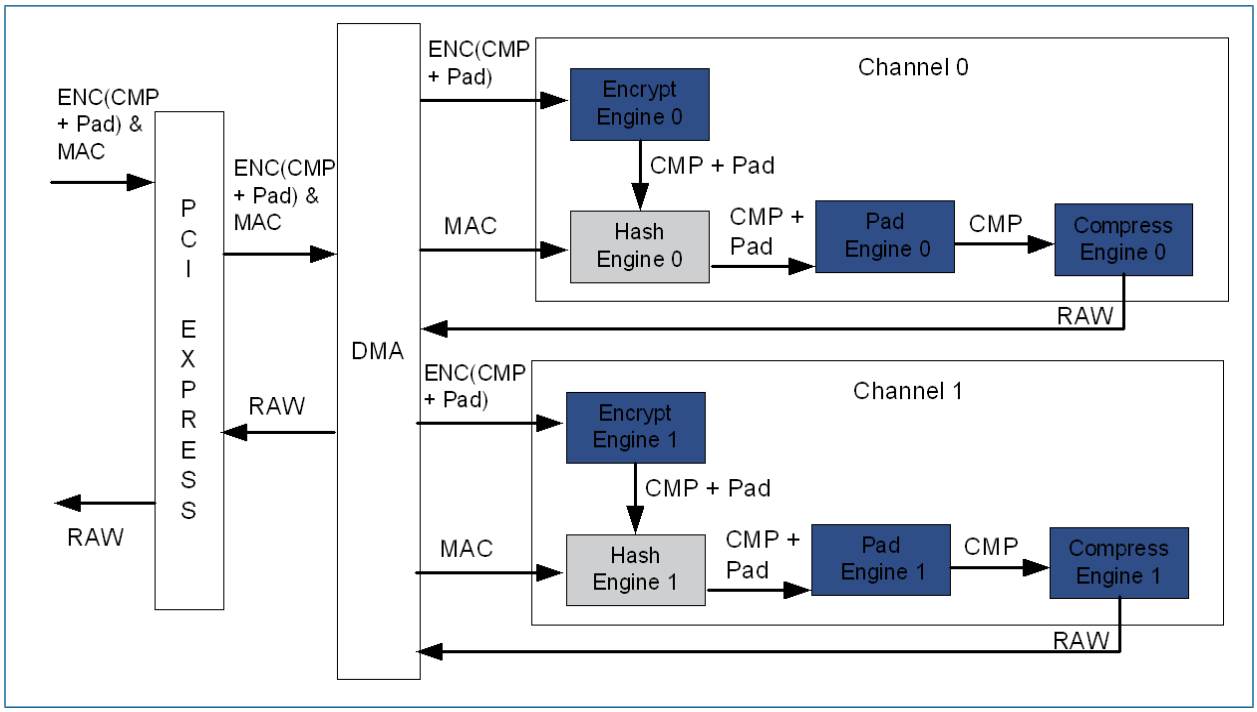


Figure 4-6. Decode Operation: Hash Engine after Encryption Engine

4.2.3 Hash Engine after Pad Engine

Figure 4-7 shows the data flow when the position of the Hash engine is after the Pad engine and all four engines are enabled. In this configuration, the Hash engine calculates the MAC value on the compressed "CMP" data and compares the calculated MAC value against the MAC value on the information bus. The data flow for the remaining engines is the same as described in Section 4.2.1, "Hash Engine before Encryption Engine". This flow is the reverse of the encode operation "Hash Engine after Compress Engine".

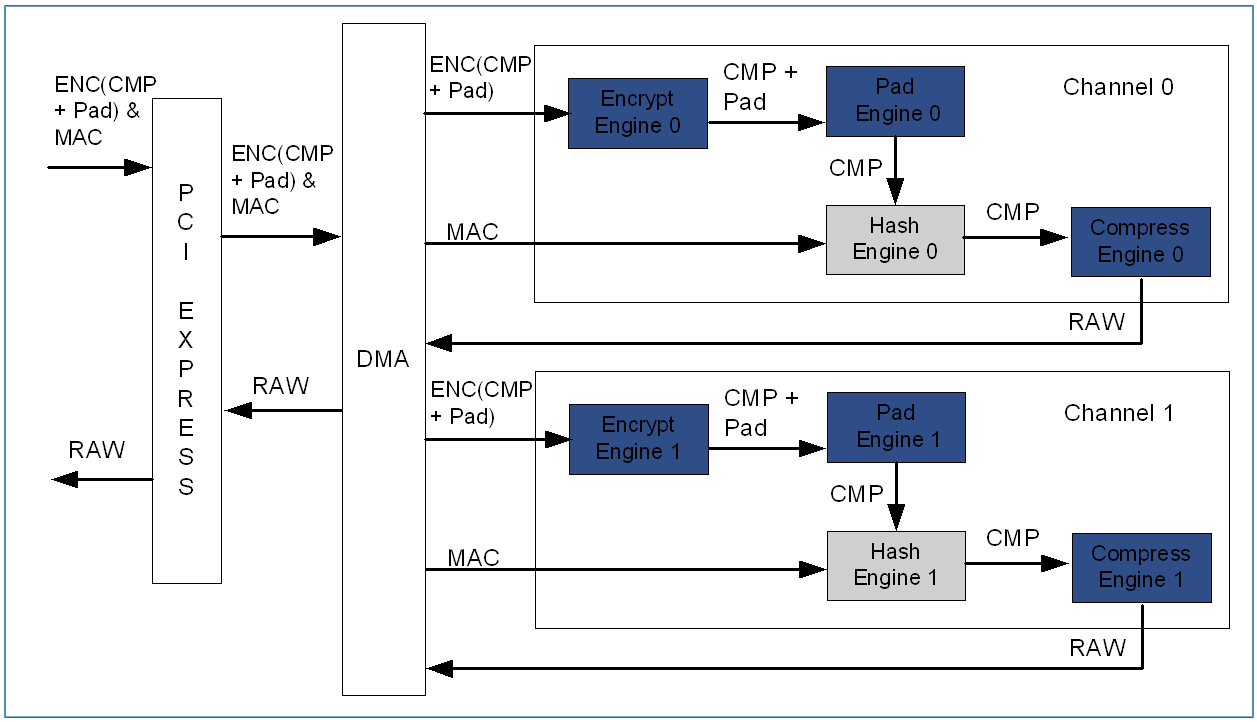


Figure 4-7. Decode Operation: Hash Engine after Pad Engine

4.2.4 Hash Engine after Compression Engine

Figure 4-8 shows the data flow when the position of the Hash engine is after the Compression engine and all four engines are enabled. In this configuration, the Hash engine calculates the MAC value of the RAW data and compares it against the MAC value on the information bus. The data flow for the remaining engines is the same as described in Section 4.2.1, "Hash Engine before Encryption Engine". The data flow is the reverse of the "Hash Engine before Compression Engine" encode operation.

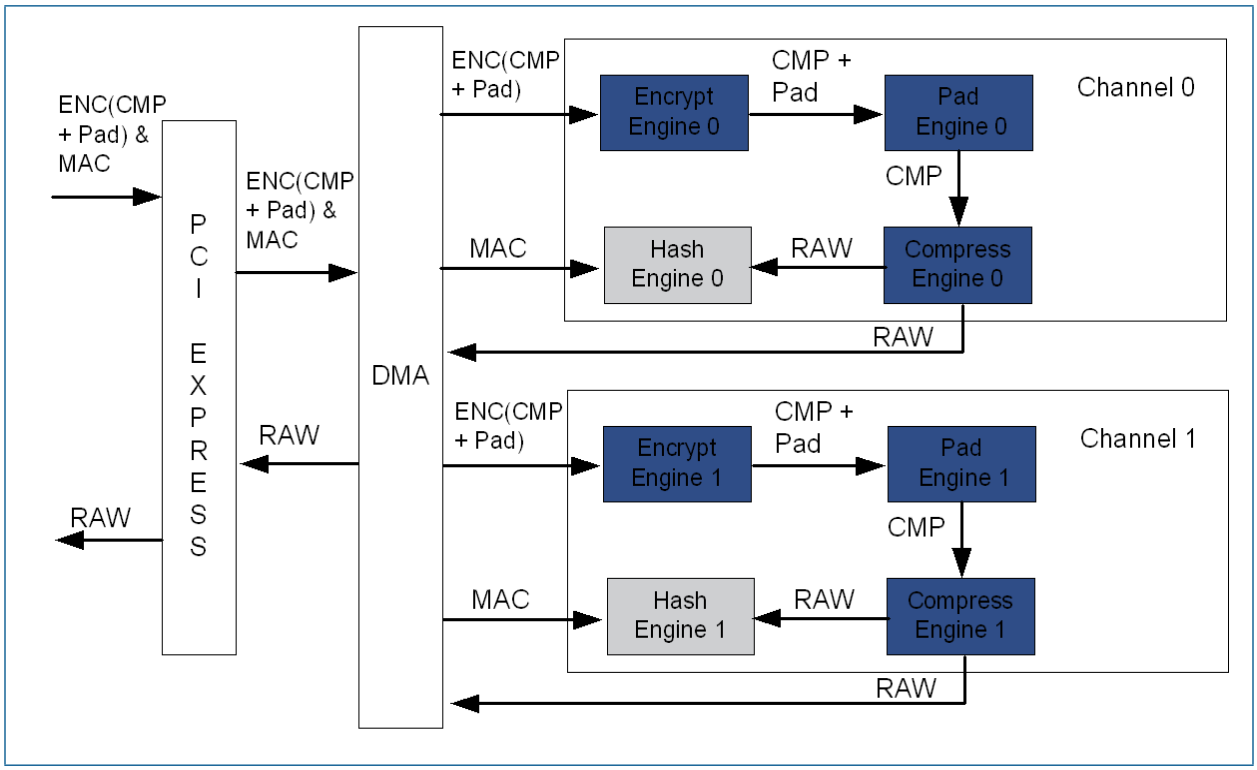


Figure 4-8. Decode Operation: Hash Engine after Compression Engine

5 Modules

This chapter gives a more detailed description of some of the key 820x internal modules: DMA, Configuration registers, Channel Managers, PKP Manager, PKP Core, RNG engine, Hash engine, LZS engine, GZIP engine, Pad engine, Encryption engine, and the clock generator.

5.1 DMA

5.1.1 PCIe Outbound Manager

The PCIe Outbound Manager (POM) provides a transmit interface between the PCIe Core and DMA modules. The main functions of the PIM are:

- Send write requests to the PCIe Core transmit interface
- Send read requests to the PCIe Core transmit interface
- Convert output data to the correct endian format

5.1.2 PCIe Inbound Manager

The PCIe Inbound Manager (PIM) provides a receive interface between the PCIe Core and DMA modules. The main functions of the PIM are:

- Decode the completion PCIe TLP (Transaction Layer Packet) from the PCIe Core completion receive interface, and send the completion data to Completion Controller
- Decode memory writes and memory reads from the PCIe Core ELBI interface (a local bus of the PCIe Core for reading or writing application-owned register space), and send write and read requests to the Configuration Register module
- Send interrupts to the host through the PCIe Core interface
- Convert input data to the correct endian format

5.1.3 Command Pointer Ring Prefetch

The Command Pointer ring Prefetch (CPP) module is used to prefetch command pointers from the host command pointer ring. The CPP implements a 128 byte Command Pointer Ring (CPR) buffer which can accommodate 16 command pointers in 64-bit addressing mode (8 bytes per pointer) and 32 command pointers in 32-bits addressing mode (4 bytes per pointer).

When the Command Pointer Ring buffer is half empty and the CPR Read Pointer is not equal to the CPR write pointer, the CPP module sends a read request to the Read Request controller (RRC, discussed in Section 5.1.4 below) to prefetch the maximum command pointer ring entries from host memory (8 entries for 64-bit addressing mode or 16 entries for 32-bit addressing mode). This CPP request has the highest priority in the RRC.

Figure 5-1 illustrates how the CPP pre-fetches 8 command pointers using 64-bit addressing. Initially, the host software writes 9 command pointers into the command pointer ring and updates the CPR write pointer for the 820x. At this point, the 820x CPR read pointer is zero because it has not read any of the commands. Once the CPP discovers that the CPR buffer is empty and CPR Read Pointer is not equal to the CPR write pointer, the 820x will fetch the 8 CPR entries and store them into the CPR buffer.

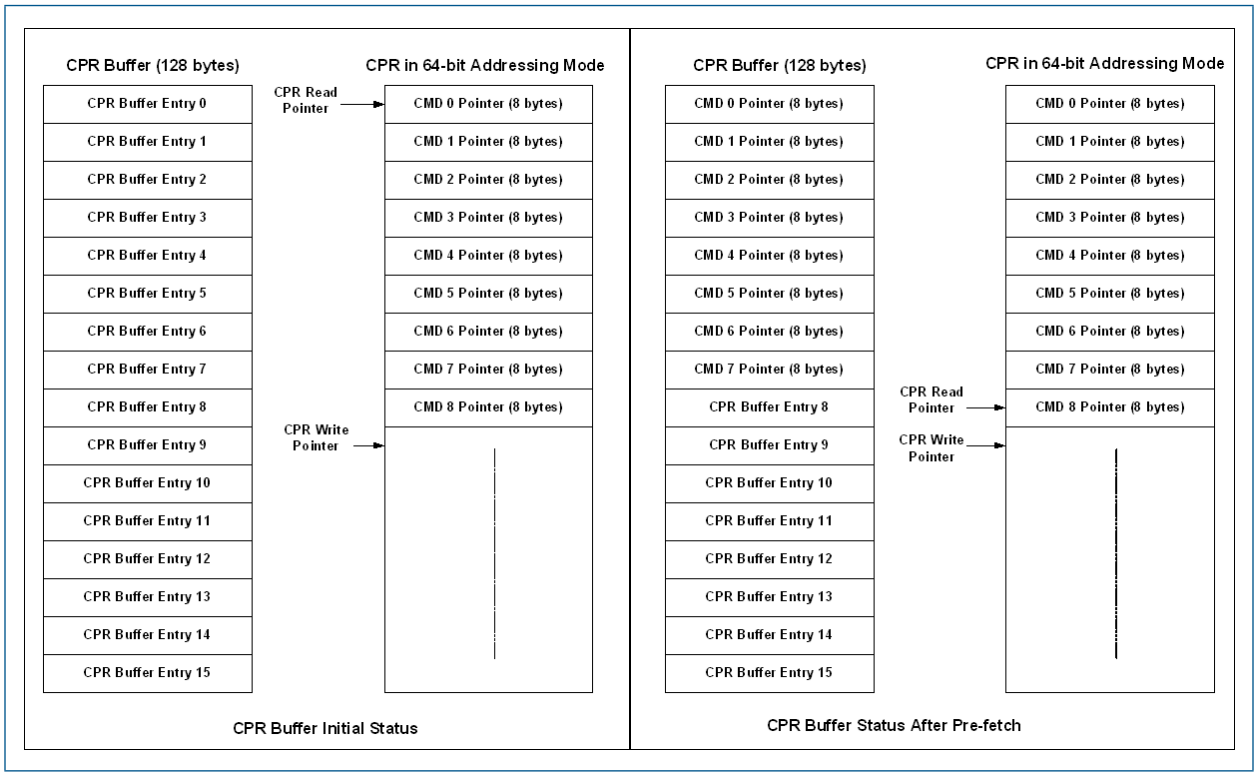


Figure 5-1. DMA Command Pointer Prefetch Example

Note: If dual command ring mode is enabled, the CPP will maintain two CPR buffers. The arbitration of the two CPR buffer read requests uses a round-robin scheme.

5.1.4 Read Request Controller

The Read Request controller (RRC) arbitrates the read requests from multiple sources, builds the TLP, and sends a read request to the POM. The main functions of the RRC are:

- Store the source descriptor starting address and byte count
- Arbitrate read requests from both channel managers, the PKP manager and the CPP
- Split the source descriptor into TLP read requests and send requests to the POM
- Arbitrate command process requests from the dual command ring using a round-robin scheme.
- Send out the command read request and then update the command pointer ring read pointer

- Push the source descriptor head and tail pointers into the pointer FIFO in the Channel Manager Inbound Data Controller (IDC) when sending source data read requests (The head and tail pointers indicate the valid bytes in a single quad-word as the source buffer starting address and length is arbitrary)

5.1.5 Write Request Controller

The Write Request controller (WRC) arbitrates the write requests from multiple sources, builds the TLP, and sends write requests to the POM. The main functions of the WRC are:

- Store the destination descriptor starting address and byte count
- Arbitrate destination write requests from both channel managers and the PKP manager
- Split the destination descriptor into TLP write requests and send those requests to the POM
- Generate a Free Pool write request if a Free Pool entry is used by the channel manager
- Generate the command structure result field and result ring write requests when a command completes
- Update the result ring write pointer after a command completes

5.1.6 Completion Controller

The Completion Controller (CC) distributes completion data to the proper destination according to the PCIe tags, which tag a completion. The main functions of the Completion Controller are:

- Decode the command structure and send the command to a channel manager
- Write the source descriptors to the RRC source descriptor buffer and destination descriptors to the WRC destination descriptor buffer
- Write the completion data to the corresponding channel manager's source buffer
- Maintain the read request record table

5.2 Configuration Registers

The Configuration Registers (CFG_REGS) module implements all 820x registers accessed by the host through the PCIe bus. The main functions of the Configuration Registers are:

- Implement all registers
- Respond to PCIe memory write and read request from the PIM
- Provide global configuration signals to all other modules

5.3 Channel Manager

There are two channel managers in the 820x. Each channel manager controls one Data Processing Channel. The DMA controller issues the read and write requests needed to execute a command and the control the data flow for the Data Processing Channels encode and decode operations.

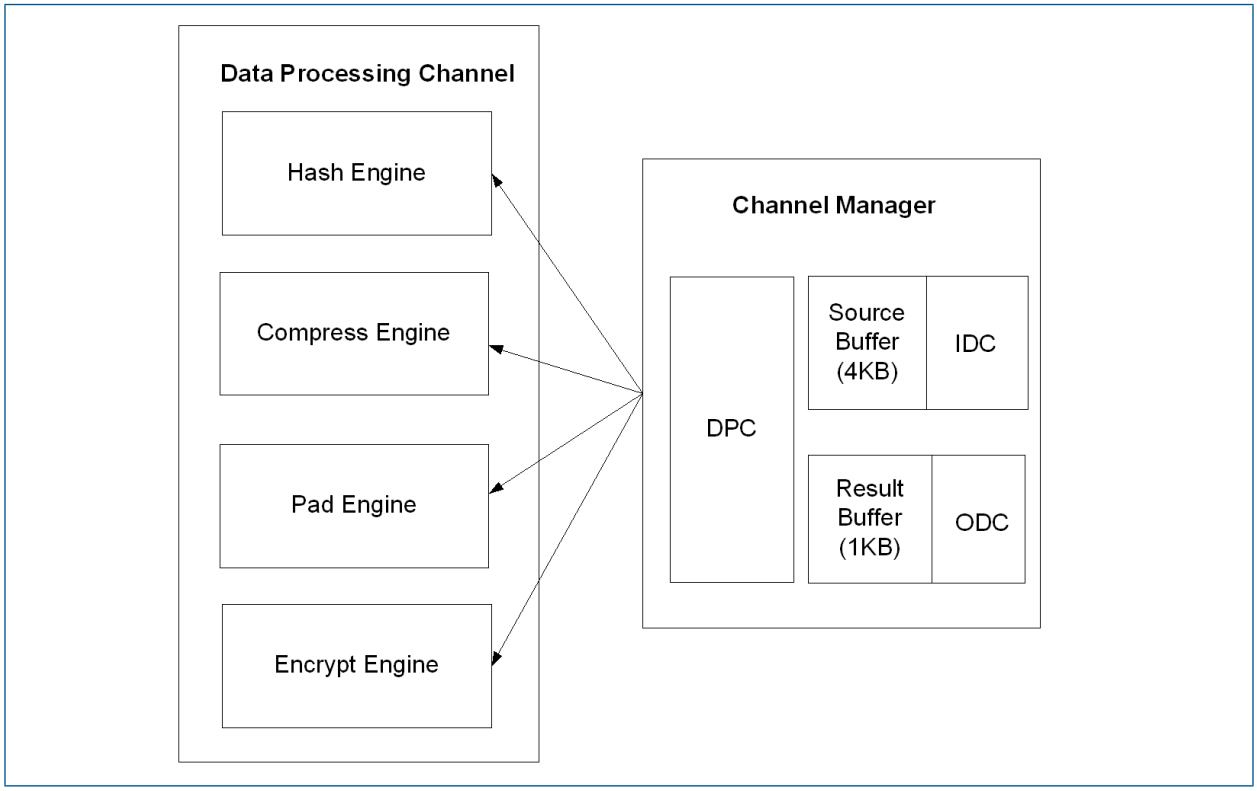


Figure 5-2. Channel Manager Block Diagram

5.3.1 Channel Manager Inbound Data Controller

The Inbound Data Controller (IDC) controls the command structure and fetching the source data. The main functions of the Inbound Data Controller are:

- Issue command structure read requests and put the command into the command buffer after the command completes
- Issue source descriptor read requests
- Issue source data read requests while maintaining the source data read request tags and ensuring the source buffer is available before issuing the source data read request (The IDC can generate a maximum 8 outstanding data read requests)
- Issue free pool descriptor read requests if the destination descriptors are all consumed

- Maintain two entries into the command buffers to improve performance by prefetching the next command's command structure and source data if all the current command’s source data is consumed
- Maintain the source descriptor head and tail pointer FIFO, which is used to indicate the valid bytes in a single quad word because the source buffer starting address and length is arbitrary

5.3.2 Channel Manager Source Buffer

The source buffer is implemented as a two-port 72-bit memory (64 bits of data and 8 bits of ECC). The source buffer memory is segmented into multiple blocks; each block comprises multiple cells. The cell size is fixed at 128-bytes, the minimum max_read_request_size.

When a Channel Manager requests a block of source data, the length will be the cell size, except the first and the last requests of a source descriptor because the starting address of a source descriptor is arbitrary aligned. The number of cells in a block are a power of 2 and depend on the parameter max_read_request_size which may be 128, 256, 512, 1K, or 2K bytes. Each Channel Manager has a 4K byte deep source buffer, and supports max_read_request_size equal 128, 256 or 512 bytes, therefore each block may consist of 1, 2, or 4 cells.

For data integrity, every 64 bits of data are protected by 8 bits of ECC.

Note: The default maximum read request size, max_read_request_size for the PCIe specification is 512 bytes. If the host configures the max read request size to 1KB or 2KB, the 820x will still send 512B read requests over the PCIe interface.

5.3.3 Channel Manager Outbound Data Controller

The Outbound Data Controller (ODC) controls the result data and command process status writes to host. The main functions of the Outbound Data Controller are:

- Issue data write requests to the POM when data in the result buffer exceeds the maximum read request size
- Issue destination descriptor read requests and free pool entry read requests
- Issue hash/MAC values write requests to the POM if used
- Issue free pool entry write requests if the free pool is used
- Issue command structure result field write requests and result ring write requests when a command completes
- Organize the errors reported by the processing engines

5.3.4 Channel Manager Result Buffer

The Result Buffer is used to temporarily store the destination data generated by the processing engines. Unlike source data, destination data is stored into the result buffer in the order that the commands complete and is fetched using the same sequence, therefore it mimics a FIFO.

The size of the write request payload must not exceed the parameter max_payload_size. The host software sets this parameter during initialization and is typically set to 128 bytes. If the result buffer is 1KB deep, it can accommodate 8 TLPs.

The result buffer is implemented as a dual-port 1K-byte memory. The data bus width is 72-bits (64 bits of data and 8 bits of ECC).

Note: The 820x supports a maximum payload size, max_payload_size, of 128, 256 or 512 bytes. If the host configures the max read request size to 1KB or 2KB, the 820x will still send 512B read requests over the PCIe interface.

5.3.5 Channel Manager Data Process Controller

The Data Process Controller (DPC) controls the data process flow and interface with all processing engines. The main functions of the Data Process Controller are:

- Separate the information bus from the source buffer, and feed the information bus field to the processing engines
- Read and align the data from the source buffer, and feed the source data to processing engines
- Verify the Key CRC if the command uses an the key
- For encode operations, append a CRC to the source data or verify the CRC of input data
- For decode operations, verify the data CRC
- Read the result data from the processing engines; align and write the data into the result buffer
- Control the encode and decode data flow

5.4 PKP Manager

The PKP Manager controls the flow of fetching instructions and operand data from host memory to the PKP engine, and the flow of transmitting the calculated results from the PKP engine to host memory.

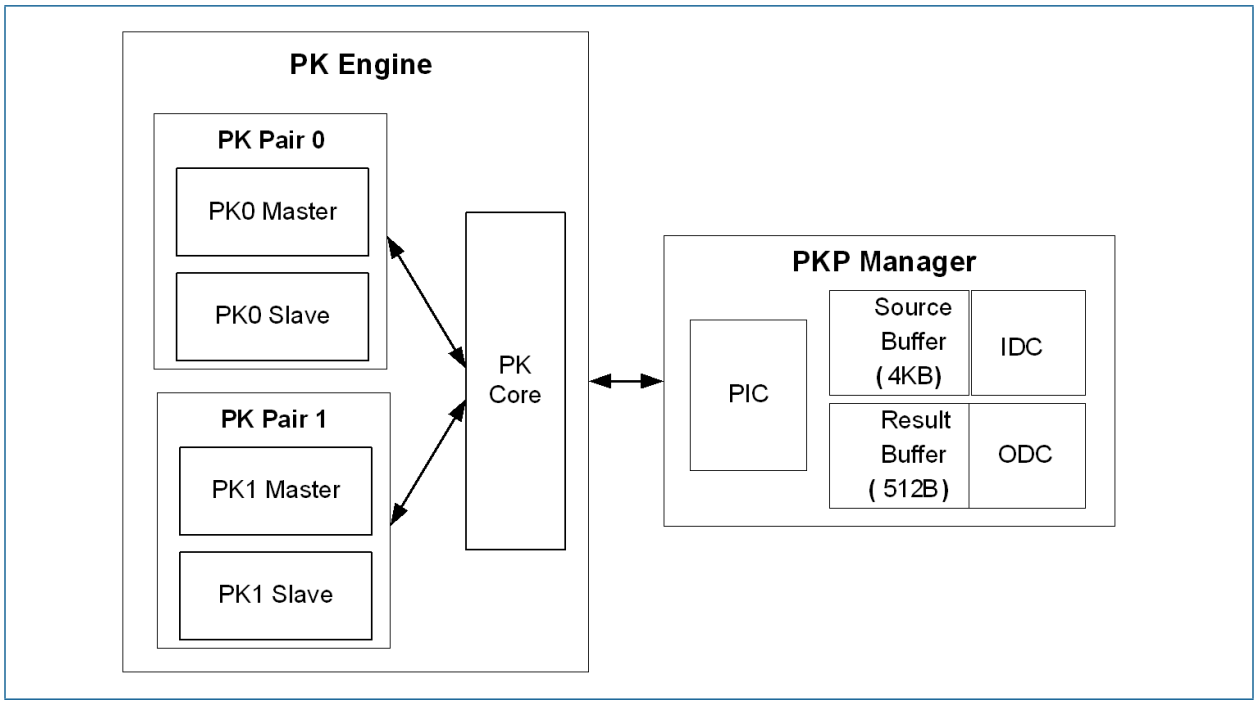


Figure 5-3. PKP Manager Block Diagram

5.4.1 PKP Inbound Data Controller

The PKP Inbound Data Controller (IDC) controls the PKP instruction and source data fetching. The IDC can generate a maximum of 8 outstanding data read requests. The main functions of the IDC are:

- Issue instruction and data read requests, maintain source data read request tags and ensure source buffer is available when issuing source data read requests
- Prefetch the next command's source data to improve performance

5.4.2 PKP Source Buffer

The PKP Source buffer is used to buffer the instruction and data of a PK operation, and is implemented as a two-port 72-bit memory. The memory is segmented into multiple blocks. Each block is comprised of multiple cells as described in Section 5.3.2, "Channel Manager Source Buffer". For data integrity, every 64 bits of data are protected by 8 bits of ECC.

5.4.3 PKP Outbound Data Controller

The PKP Outbound Data Controller (ODC) controls writing the result data to the host. The main function of the ODC is:

- Issue data write request to the POM when the data is available (the PKP result data is ready when the calculation is finished; the PIC will write the data into the result buffer continuously).

5.4.4 PKP Result Buffer

The PKP Result Buffer is used to temporarily store the calculated result generated by the PKP Engine. Unlike the source data, the destination data is stored in-sequence into the result buffer and is fetched using the same sequence, similar to a FIFO.

The PIC writes data to the Result Buffer after each PKP calculation completes. Whenever result data is available in the PKP Data Register, the PIC will continuously transfer the data to the result buffer until the Result Buffer is full.

Result buffer is implemented as a 72-bit RAM, with 64 bits of data and 8 bits of ECC.

5.4.5 PKP Interface Controller

The PKP Interface Controller (PIC) is the communication channel between the PKP Manager and the PKP Engine. The main functions of the PIC are:

- Read instructions from the source buffer, and transfer them to the PKP Engine Linear Instruction Registers (LIRs)
- Read data from the source buffer, and transfer them to the PKP Engine Data Registers
- Control the command execution flow

5.4.6 PKP Core

The PKP Core is IP from Athena. Please refer to Athena's *EXP-E5200 Public Key Cryptography Microprocessor* for more information.

5.5 RNG Engine

The RNG Engine is Exar IP. Please contact Exar if detailed information is required.

5.6 Hash Engine

The Hash Engine calculates the hash or MAC values. The Hash Engine comprises an input AFIFO, output AFIFO, Hash AFIFO, Hash interface controller and two Hash cores.

This engine can skip leading bytes and consume varying amounts of payload data to support current and future network security protocols. These options are provided on a per command basis.

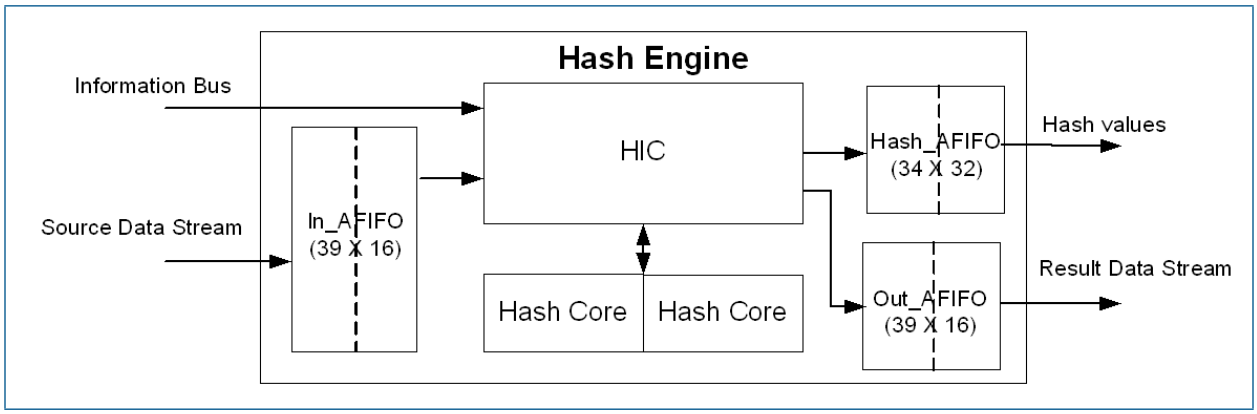


Figure 5-4. Hash Engine Block Diagram

Note, the Information Bus includes Hash_Head_Count, Hash_Source_Count from the Channel Manager.

5.6.1 Hash In_AFIFO & Out_AFIFO

The Hash In_AFIFO and Out_AFIFO are used to transfer data and control signals between the DMA clock domain (125M) and the Hash Engine clock domain (200M).

5.6.2 Hash_AFIFO

The Hash AFIFO is used to store the hash values calculated by the Hash cores, and synchronize the data transfer between the DMA clock domain (125M) and the Hash Engine clock domain (200M).

The Hash_AFIFO is implemented as a 34 x 32 bit flip-flop. A MD5 hash result is 128 bits (16 bytes) and occupies four entries (4 DWs) in the Hash_AFIFO; a SHA1 hash result is 160 bits (20 bytes) and occupies three entries (5 DWs) of the Hash_AFIFO; a SHA2 hash result is 256 bits (32 bytes) and occupies four entries (8 DWs) of the Hash_AFIFO. Therefore, the Hash_AFIFO can accommodate at least 4 hash result values.

5.6.3 Hash Interface Controller

The Hash Interface Controller (HIC) is the communication bridge between the In_AFIFO, Hash_AFIFO and Hash cores. The main functions of the HIC are:

- Read the source data from the In_AIFIO, truncate the header and tail from the data stream, and then send the truncated data to the Hash core according to Hash_Head_Count and Hash_Source_Count
- Control the real time verification flow
- Control the two Hash core's execution of the required operation
- Read the hash values from the Hash Cores and send them to the Hash_AFIFO

- Combine the source data with the MAC value, and then send it to the Out_AFIFO
- Report the Hash core status to the Channel Manager after each command completes

5.6.4 Hash Core

The Hash engine contains two Hash cores. Each core supports six operation modes:

- slice hash only
- file hash only
- slice hash + file hash
- MAC
- SSL3.0 MAC
- XCBC-MAC (with SHA1, SHA256, and MD5 algorithms).

5.6.4.1 Hash Core Operation Modes

Slice Hash only

In this mode, one core calculates the hash value, while the other is used for real time verification. Each slice is treated as a complete hash operation (with length padding).

File Hash only

In this mode, one core calculates the hash value, while the other is used for real time verification. Hash engine will load the initial hash value into the hash cores during initialization, and performs length padding on the last block of data (one command, one block data).

A File Hash operation performs the hash operation on the entire data block of one command. A File Hash command can be either stateless or stateful. If the hash of one file can be done in one command, it is considered stateless; otherwise if the hash of one file requires two or more commands, it is considered stateful.

Example of a stateless File Hash (one file per command):

The Hash core uses the default initial hash value, and adds length padding at the end of the data block. The output hash is considered complete.

Example of a stateful file hash (one file uses three commands):

a. The Hash core uses the default initial hash value for the data block of the first command, and does not add padding. The output hash is considered incomplete but is used as the IHV for the second command.

b. For the second command, the Hash core uses the IHV calculated from the first command, and does not add padding. The output hash is considered incomplete but is used as the IHV for the third command.

c. For the third command, the Hash core uses the IHV calculated from the second command, and pads the last data block. The output is considered complete and is the hash value of the entire file.

Slice Hash + File Hash

In this mode, one Hash core calculates the slice hash, while the other calculates the file hash. No internal data integrity checking is performed in this mode.

Figure 5-5 shows an application example of a Slice Hash + File Hash operation.

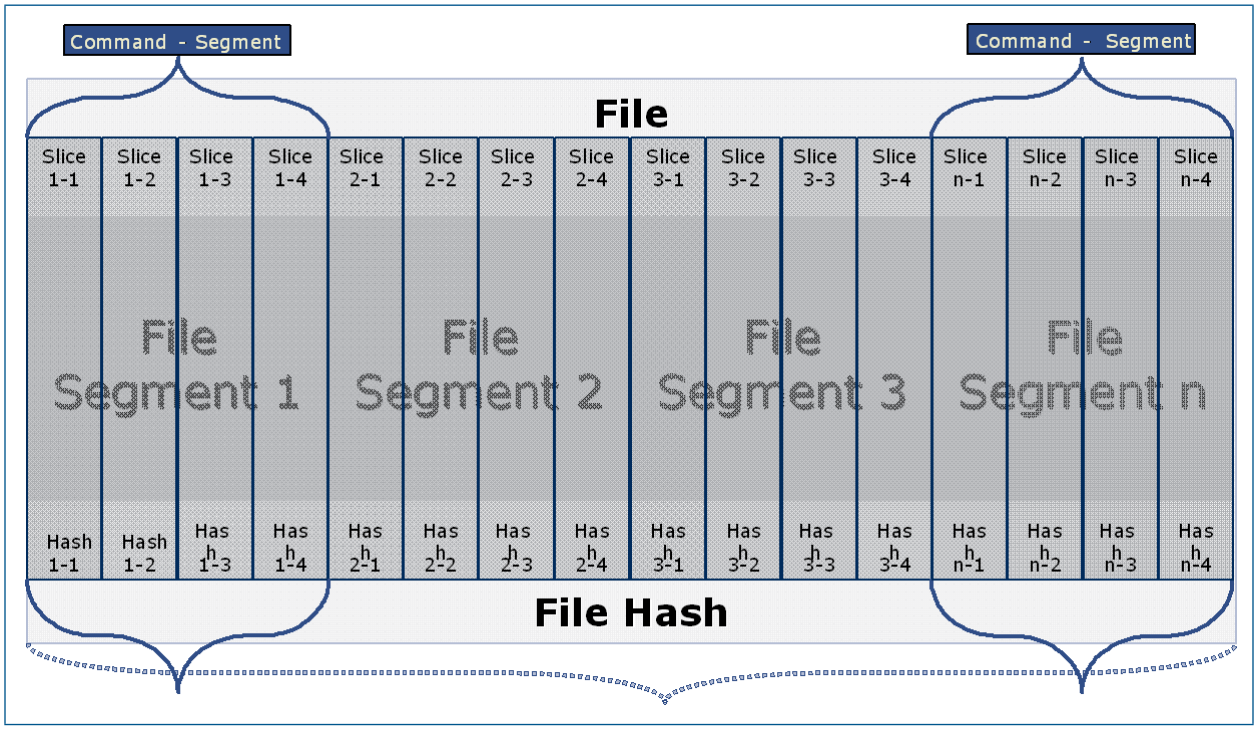


Figure 5-5. Application Example of Slice Hash + File Hash

MAC

The MAC is actually keyed hash functions within a keyed hash function. The general notation for a MAC message M using key K is:

MAC(K,M) = H(K xor opad, H(K xor ipad, M))

MAC(K,M) = H(OPAD, H(IPAD,M))

Where H is one of the hashing algorithms (SHA-1, SHA-256, or MD5), IPAD and OPAD are the pre-computed partial results of the 64-element array (of values 0x36 and 0x5c respectively) computed from the secured key (K) and IV (Initial Value). Either hashing algorithm provides automatic pad insertion and MAC defined length insertion on an arbitrary sized data block.

SSL3.0 MAC

In this mode, one Hash core calculates the SSL3.0 MAC, while the other performs real time verification.

The MAC in SSL3.0 is calculated as follows:

Hash(Write Secret || Pad2 || Hash(Write Secret || Pad1 || Seq. No. || Type || Length || Application Data))

Where:

Hash = SHA-1 or MD5

Write Secret = 20 bytes if SHA-1, 16 bytes if MD5; from Crypto Header

Pad1 = 48/40 bytes of 0x36 (MD5/SHA-1)

Pad2 = 48/40 bytes of 0x5C (MD5/SHA-1)

SeqNum = 8 byte Sequence Number from Crypto Header

Type = Type byte from SSL Header

Length = 16 bit Length of payload (no padding!)

The host software must combine "Seq. No. || Type || Length || Application Data" as source data, and the Hash Engine will calculate the SSL3.0 MAC.

XCBC-MAC

The AES-XCBC-MAC-96 algorithm is a variant of the basic CBC-MAC with obligatory 10* padding; however, AES-XCBC-MAC-96 is secure for messages of arbitrary length. The AES-XCBC-MAC-96 calculations require numerous encryption operations; this encryption MUST be accomplished using AES with a 128-bit key. Given a 128-bit secret key K, AES-XCBC-MAC-96 is calculated as follows for a message M that consists of n blocks, M[1]... M[n], in which the blocksize of blocks M[1]... M[n-1] is 128 bits and the blocksize of block M[n] is between 1 and 128 bits:

(1) Derive 3 128-bit keys (K1, K2 and K3) from the 128-bit secret

key K, as follows:

K1 = 0x01010101010101010101010101010101 encrypted with Key K

K2 = 0x02020202020202020202020202020202 encrypted with Key K

K3 = 0x03030303030303030303030303030303 encrypted with Key K

(2) Define E[0] = 0x00000000000000000000000000000000

(3) For each block M[i], where i = 1... n-1:

XOR M[i] with E[i-1], then encrypt the result with Key K1, yielding E[i].

(4) For block M[n]:

(a) If the blocksize of M[n] is 128 bits:

XOR M[n] with E[n-1] and Key K2, then encrypt the result with Key K1, yielding E[n].

(b) If the blocksize of M[n] is less than 128 bits:

(i) Pad M[n] with a single "1" bit, followed by the number of "0" bits (possibly none) required to increase M[n]'s blocksize to 128 bits.

(ii) XOR M[n] with E[n-1] and Key K3, then encrypt the result with Key K1, yielding E[n].

(5) The authenticator value is the leftmost 96 bits of the 128-bit E[n].

NOTE1: If M is the empty string, pad and encrypt as in (4b) to create M[1] and E[1]. This will never be the case for ESP or AH, but is included for completeness sake.

NOTE2: [CBC-MAC-2] defines K1 as follows:

K1 = Constant1A encrypted with Key K | Constant1B encrypted with Key K.

However, the second encryption operation is only needed for AES-XCBC-MAC with keys greater than 128 bits; thus, it is not included in the definition of AES-XCBC-MAC-96.

AES-XCBC-MAC-96 verification is performed as follows:

Upon receipt of the AES-XCBC-MAC-96 authenticator, the entire 128-bit value is computed and the first 96 bits are compared to the value stored in the authenticator field. Keying Material AES-XCBC-MAC-96 is a secret key algorithm. For use with either ESP or AH a fixed key length of 128-bits MUST be supported. Key lengths other than 128-bits MUST NOT be supported (i.e. only 128-bit keys are to be used by AES-XCBC-MAC-96).

5.6.4.2 Hash Core Algorithms

SHA1/SHA256 Core

The SHA-1/SHA-256 calculation core performs the secure hash algorithm on N x 512-bit blocks (N < 2k) and produces a single 160-bit/256-bit hash result, CVN+1, using the 512-bit input blocks and CVN, the previous hash result.

To calculate the hash value of an input message MCALC = Min + PadSHA-1 (PadSHA-1 includes lengthSHA-1), which by definition must be an integral number of 512-bit blocks {B0, B1, B2, …, BL}, the following sequence is performed:

1. To indicate the start of a new hash calculation, set the CV value to the SHA-1/SHA-256 initial value from a previously calculated partial result.
2. Write the first block to the calculation core. (16 x 32-bit words)
3. Wait for the hash value calculation to complete.

4. If the last block was written to the core, latch the hash value, otherwise, write the next block to the calculation core and go to step 3.

Figure 5-6 shows a block diagram of the SHA Core.

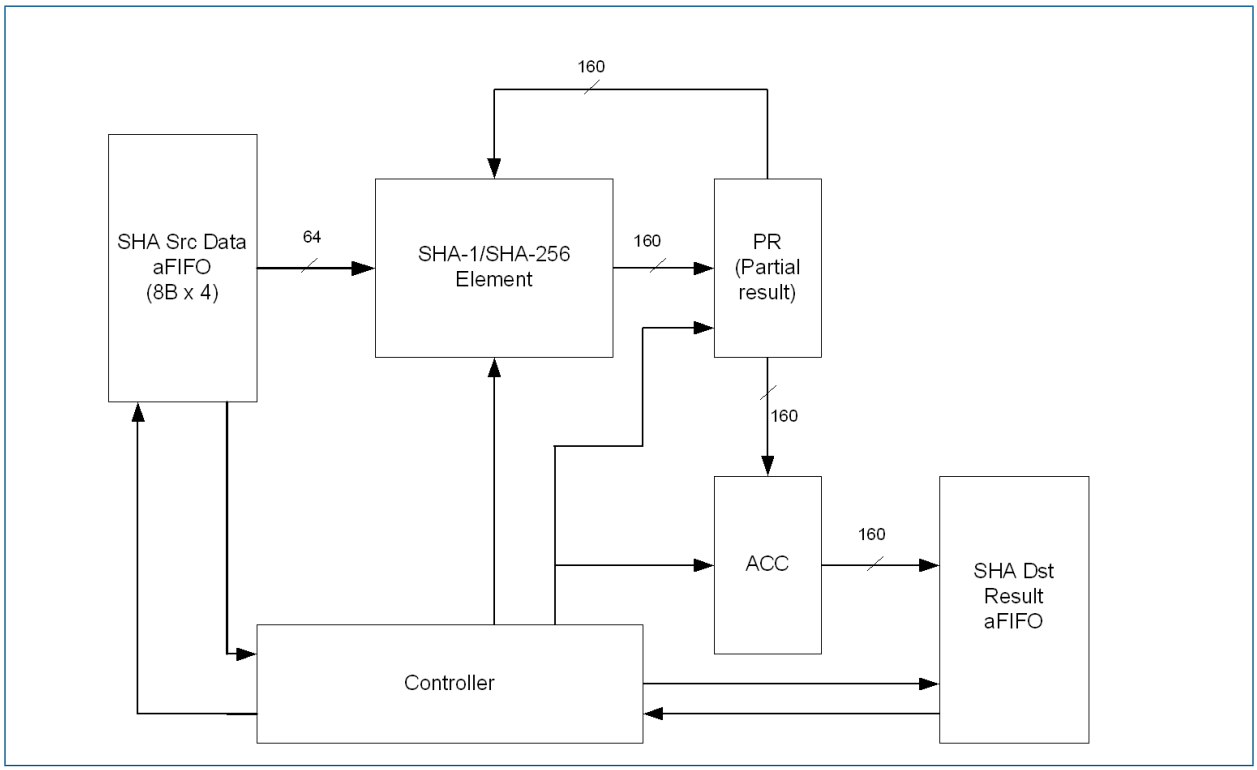


Figure 5-6. SHA Core Block Diagram

MD5 Core

The MD5 calculation core performs the MD5 message digest algorithm on a N x 512-bit blocks (N < 4k) and produces a single 128-bit message digest, CVN+1, using the 512-bit input blocks and CVN, the previous message digest.

To calculate the hash value of an input message MCALC= MIN + PadMD5 (PadMD5 includes lengthMD5), which by definition must be an integral number of 512-bit blocks {B0, B1, B2, …, BL}, the following sequence is performed:

1. To indicate the start of a new hash calculation, set the CV value to the MD5 initial value from a previously calculated partial result.

2. Write the first block to the calculation core. (16 x 32-bit words)

3. Wait for the hash value calculation to complete.

4. If the last block was written to the core, latch the hash value, otherwise, write the next block to the calculation core and go to step 3.

Figure 5-7 shows a block diagram of the MD5 Core.

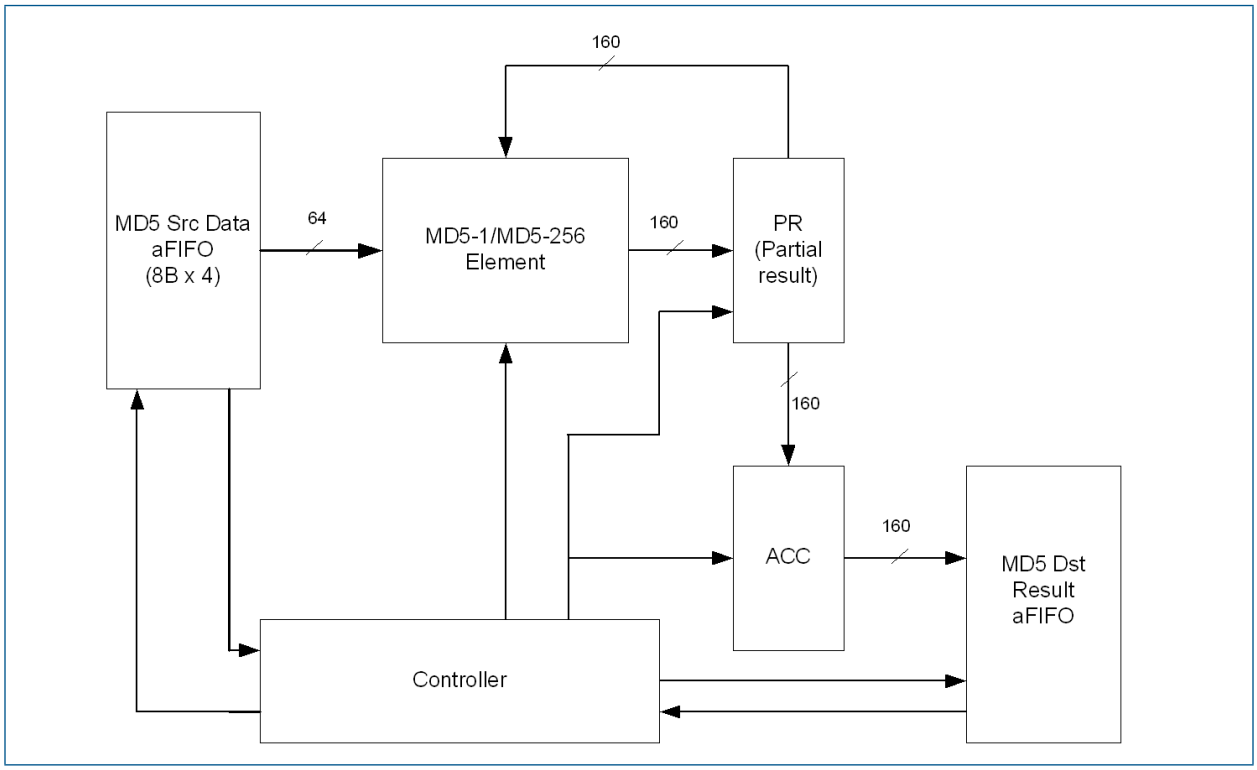


Figure 5-7. MD5 Core Block Diagram

5.7 LZS Engine

The LZS Engine compresses/decompresses source data using Exar's LZS and enhanced LZS (eLZS) algorithms. The LZS engine comprises an input AFIFO, output AFIFO, LZS interface controller and LZS core. This engine can skip leading bytes and consume varying amounts of payload data to support current and future network security protocols. These options are provided on a per command basis.

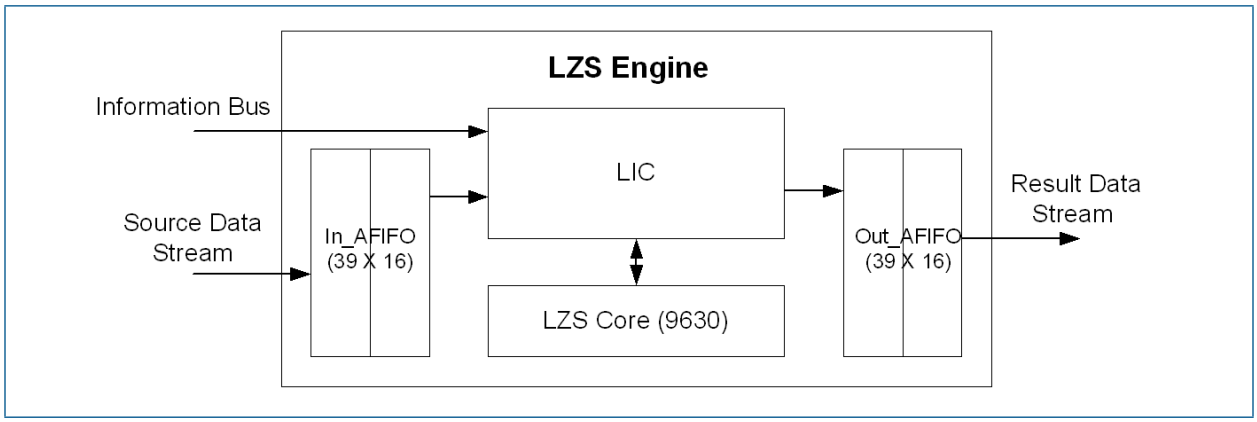


Figure 5-8. LZS Engine Block Diagram

Note: The Information Bus includes CMP_Head_Count and CMP_Source_Count from the Channel Manager.

5.7.1 LZS In_AFIFO & Out_AFIFO

The LZS In_AFIFO and Out_AFIFO are used to transfer data and control signals between the DMA clock domain (125M) and the LZS Engine interface controller clock domain (150M).

Both the In_AFIFO and Out_AFIFO are implemented as 39 x 16 bit flip-flops.

5.7.2 LZS Core Interface Controller

The LZS core Interface Controller (LIC) is the communication bridge between the In_AFIFO, Out_AFIFO, and LZS core. The main functions of the LIC are:

- Read the source data from the In_AIFIO, pass through the header and tail of the data stream to next engine (Out_AFIFO), and send the remaining data to the LZS core according to the CMP_Head_Count and CMP_Source_Count.
- Write to the LZS core command register at the beginning of each command.
- Read the result data from the LZS Core, merge the tail of the source data stream and the result data into a double word, and send it to the Out_AFIFO.
- Report the LZS core status to the Channel Manager after each command completes.

5.7.3 LZS Core

The LZS Core is Exar IP. The LZS core is a high performance lossless data compression processor that uses the industry-standard Lempel-Ziv-Stac (LZS®) compression algorithm, as well as the "Enhanced LZS" (eLZS) algorithm. The output of the compression engine is tied to the input of the decompression engine for compression operation verification.

The LZS Core uses an anti-expansion algorithm that prevents the output from expanding during compression. If the output expands during compression the original uncompressed input along with certain header bytes is passed through as the output.

5.8 GZIP Engine

GZIP Engine compresses/decompresses source data with the GZIP algorithm, it includes the input AFIFO, output AFIFO, GZIP interface controller and GZIP core.

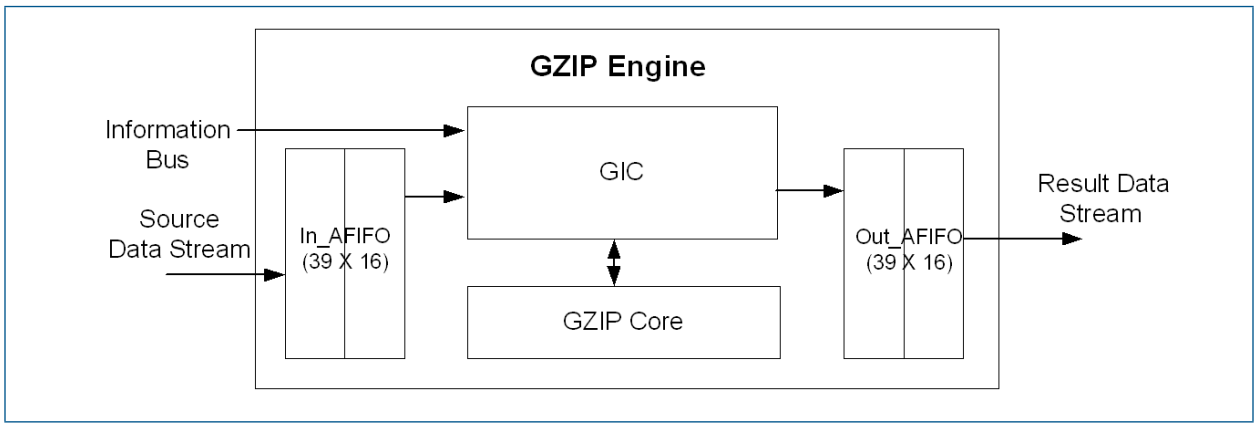


Figure 5-9. GZIP Engine Block Diagram

5.8.1 GZIP In_AFIFO & Out_AFIFO

The GZIP In_AFIFO and Out_AFIFO are used to transfer data and control signals between the DMA clock domain (125M) and the GZIP Engine interface controller clock domain (200M).

Both the In_AFIFO and Out_AFIFO are implemented as 39 x 16 bit flip-flops.

5.8.2 GZIP Core Interface Controller

The GZIP Core Interface Controller (GIC) is the communication bridge between the In_AFIFO, Out_AFIFO and GZIP core. The main functions of the GIC are:

- Read the source data from the In_AIFIO, pass through the header and tail of the data stream to next engine (Out_AFIFO), and send the remaining data to the GZIP core according to the CMP_Head_Count and CMP_Source_Count.
- Control the operation flow of GZIP core.
- Read the result data from the GZIP Core, merge the tail of the source data stream and the result data into a double word, and send it to the Out_AFIFO.
- Report the GZIP core status to Channel Manager after each command completes.

5.8.3 GZIP Core

The GZIP core complies with RFC 1951, and RFC 1952 and supports the Dynamic Huffman algorithm for a high compression ratio.

5.9 Pad Engine

The Pad Engine is used to add padding to the data stream for encode operations and remove padding for decode operations. This engine can skip leading bytes and consume varying amounts of payload data to support current and future network security protocols. These options are provided on a per command basis.

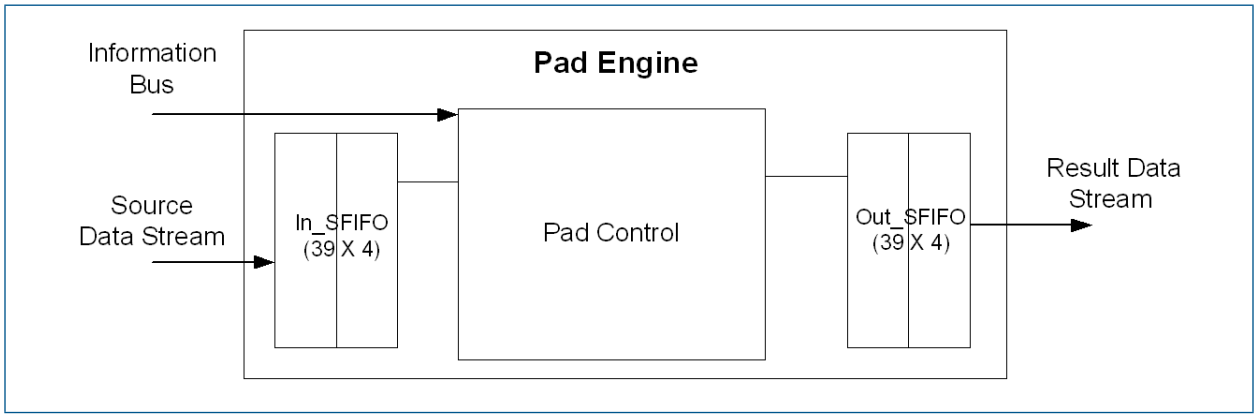


Figure 5-10. Pad Engine Block Diagram

The main functions of the Pad Engine are:

- For encode operations, receive the data stream from the Channel Manager, adding padding to the data stream according to the required algorithm.
- For encode operations, receive the data stream from Channel Manager, truncating the padding from the data stream.
- Extract the "Next Header" and "Pad Length" fields from data stream for decode operations.
- Detect Pad errors according to the pad algorithm.

5.10 Encryption Engine

The Encryption Engine encrypts/decrypts source data with either the AES or 3DES algorithm. The Encryption Engine is comprised of the input AFIFO, output AFIFO, Encryption interface controller, AES Core and 3DES Core. This engine can skip leading bytes and consume varying amounts of payload data to support current and future network security protocols. These options are provided on a per command basis.

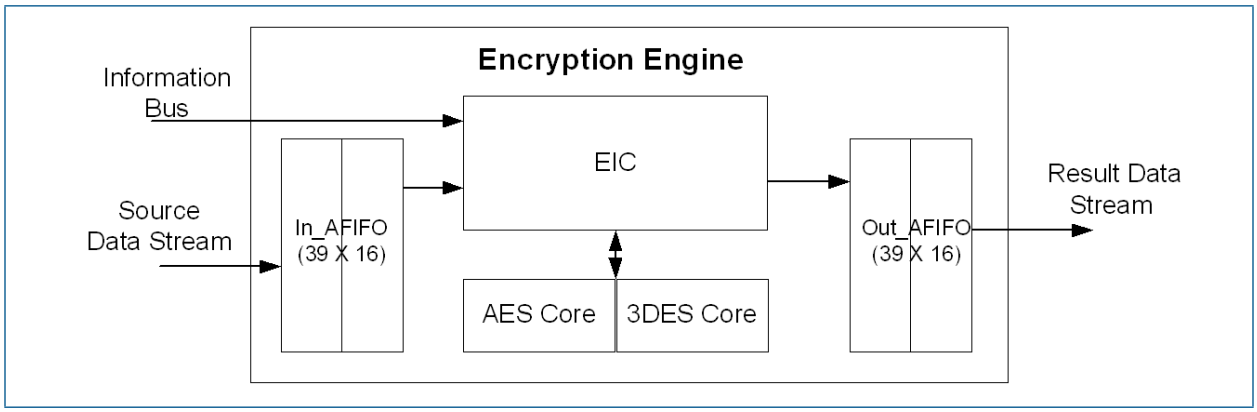


Figure 5-11. Encryption Engine Block Diagram

5.10.1 Encryption In_AFIFO & Out_AFIFO

The Encryption In_AFIFO and Out_AFIFO are used to transfer data and control signals between the DMA clock domain (125M) and the Encryption Engine interface controller clock domain (200M).

Both the In_AFIFO and Out_AFIFO are implemented as 39 x 16 bit flip-flops.

5.10.2 Encryption Interface Controller

The Encryption Interface Controller (EIC) is the communication bridge between the In_AFIFO, Out_AFIFO, AES core, and 3DES core. The main functions of the EIC are:

- Read the source data from the In_AIFIO, pass through the header and tail of data stream to the next engine (Out_AFIFO), and send the remaining data to the AES core/3DES core according to the ENC_Head_Count and ENC_Source_Count.
- Control the operation flow of the AES and 3DES cores.
- Read the result data from the AES Core/3DES cores, merge the tail of the source data stream and the result data into a double word, and send it to the Out_AFIFO.
- Report the AES and 3DES core status to the Channel Manager after each command completes.

5.10.3 Encryption AES and 3DES Cores

The AES and 3DES cores are Exar IP. The Encryption/Decryption engine supports AES-GCM, CBC, CTR and ECB with 128, 192 or 256 bit keys, AES-XTS with 256 or 512 bit keys, and 3DES.

5.11 Clock and Reset Generator

The CLK_RST_Gen module resides in the core of the 820x, and is responsible for the generation and management of all clocks and resets throughout the device. The major components of the CLK_RST_Gen module are:

- Main PLL x 1
- PCIE PHY PLL x 1
- Clock dividers, distribution and gating logic
- Reset generation logic

The Main PLL is used to multiply the input clock to a frequency of 600MHz, which is then divided down to 200MHz, 150MHz, and 300MHz for the processing engines. The PCIe PHY PLL is used to generate a 125MHz clock for most of the DMA related modules.

To conserve power, the CLK_RST_Gen modules uses a clock gating cell statically or dynamically to disable the clock to each data processing channel engine.

CLK_RST_Gen also generates the power on reset and software reset signals for all modules.

5.12 Serial Peripheral Interface

This section describes the 820x Serial Peripheral Interface (SPI) used to connect the 820x to a programmable device such as a Flash or EEPROM. Note that although the Express DX SDK supports the SPI, a programmable device is not required or recommended for embedded applications.

The *820x Hardware Design Guide*, UG-0211, contains more detailed information about using a programmable device with the 820x.

The 820x Serial Peripheral Interface can be used to connect the 820x to an external programmable device to allow the host to read/write the programmed data through the SPI registers. Figure 5-12 illustrates how the 820x may be connected to a programmable device.

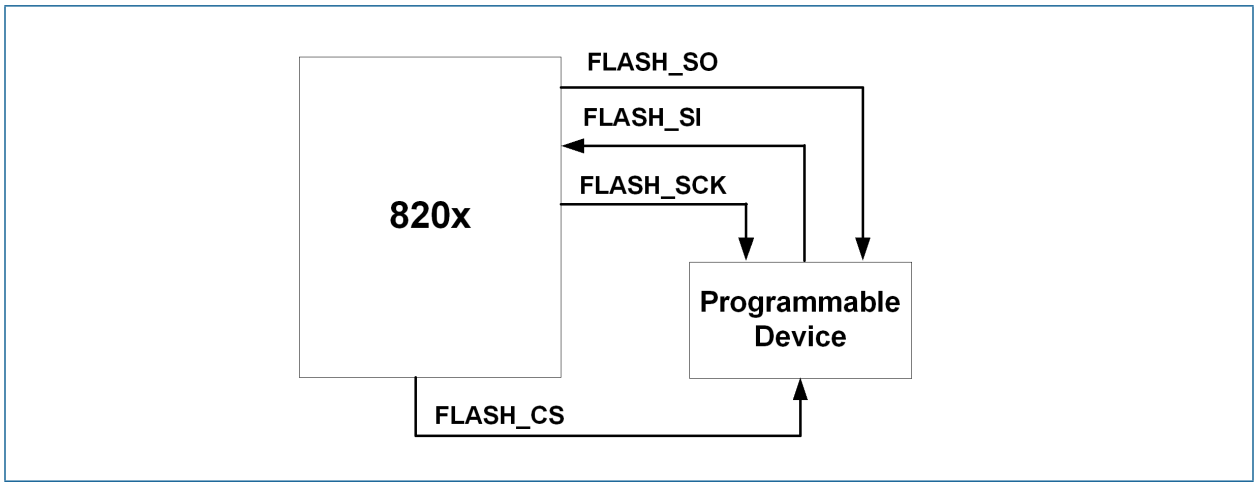


Figure 5-12. SPI Example Usage

Refer to the *820x Hardware Design Guide*, UG-0211, for the recommended programmable devices.

During initialization, the behavior of the programmable device is controlled by the external pins EXT_FLASH_CFG_EN and PCIE_PHY_CFG. Table 5-1 lists the possible combinations of these pins and the corresponding response of the 820x device. After initialization, the host may access an attached programmable device by enabling the SPI module clock via the "SPI Enable Register", regardless of the settings of these two external pins.

Table 5-1. Behavior of PCIE_PHY_CFG and EXT_FLASH_CFG_EN Pins during Initialization

PCIE_PHY_CFG	EXT_FLASH_CFG_EN	Description
0	0	The host will read the default values for the subsystem ID and vendor ID from the values that are stored in the PCIe "Sub-System ID Register" and "Vendor ID Register". The expansion BAR is not supported in this mode. The PHY settings will be read from the external pins PCIE_RXEQCTL[2:0], PCIE_TX_BOOST[3:0], PCIE_LOS_LVL[4:0], and PCIE_TX_LVL[4:0].
0	1	The host will read the values for the subsystem ID and vendor ID that are stored in the programmable device. The value for the expansion BAR will be read from the value stored in the PCIe "Expansion ROM Base Address Register". The PHY settings will be read from the external pins PCIE_RXEQCTL[2:0], PCIE_TX_BOOST[3:0], PCIE_LOS_LVL[4:0], and PCIE_TX_LVL[4:0].
1	0	The host will read the default values for the subsystem ID and vendor ID from the values that are stored in the PCIe "Sub-System ID Register" and "Vendor ID Register". The expansion BAR is not supported in this mode. The PHY settings will be read from the internal default values listed below. PCIE_RXEQCTL[11:0] = 12'b010010010010 PCIE_TX_BOOST[15:0] = 16'b1011101110111011 PCIE_LOS_LVL[4:0] = 5'b01100 PCIE_TX_LVL[4:0] = 5'b01100
1	1	The host will read the values for the subsystem ID and vendor ID that are stored in the programmable device. The value for the expansion BAR will be read from the value stored in the PCIe "Expansion ROM Base Address Register". PHY configuration is read from the external programmable device.

Table 5-2 lists the programmable device memory map. If the user chooses to store UEFI boot code in the programmable device, the starting address should be 1K.

Note that some of the PCIe parameters, namely PCIE_RXEQCTL[11:0], PCIE_TX_BOOST[15:0], have more resolution internally in the 820x device and in the programmable device than the external pins of the same name. The predefined fields are described in more detail after the table.

Table 5-2. Programmable Device Memory Map

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x00	Subsystem ID		Vendor ID	
0x04	[15:4] = PCIE_RXEQCTL[11:0] [3:0] = 4'b0000		[15:0] = PCIE_TX_BOOST[15:0]	
0x08	[7:0] = 8'h00	[7:5] = 3'b000 [4:0] = PCIE_LOS_LVL [4:0]	[7:5] = 3'b000 [4:0] = PCIE_TX_LVL[4:0]	Reserved
0x0C	Expansion ROM BAR Bits 31:1 Expansion ROM BAR Mask Indicates which Expansion ROM BAR bits to mask (make non-writable) from host software, which, in turn, determines the size of the BAR. For example, 0xFFF claims a 4096-byte BAR by masking bits 11:0 of the BAR from writing by host software.The maximum value is 0xFFFFFF because the maximum space that can be claimed by an Expansion ROM BAR is 16 MB. Bit 0 Expansion ROM BAR enable 0 = Expansion ROM BAR disabled 1 = Expansion ROM BAR enabled			

Subsystem and Vendor ID

The host reads the subsystem and vendor ID from the 820x device during initialization. The value that the 820x responds with depends on the setting of the pin EXT_FLASH_CFG_EN. If EXT_FLASH_CFG_EN is disabled, the 820x will respond with the Exar subsystem and vendor IDs that are hard-coded into the PCIe registers (“Vendor ID Register”, “Device ID Register”). If EXT_FLASH_CFG_EN is enabled, the 820x will respond with the subsystem and vendor ID values that are stored in the programmable device, thus allowing custom applications to respond with their own subsystem and vendor ID values.

PHY Settings

The 820x device configures the PHY during initialization. The values that the 820x uses to configure the PHY depend on the setting of the pins PCIE_PHY_CFG and EXT_FLASH_CFG_EN. If PCIE_PHY_CFG is disabled, the 820x will configure the PHY using the values set on the pins PCIE_RXEQCTL[2:0], PCIE_LOS_LVL[4:0], PCIE_TX_BOOST[3:0] and PCIE_TX_LVL[4:0]. If PCIE_PHY_CFG is enabled and EXT_FLASH_CFG_EN is disabled, the 820x will configure the PHY using the default internal register values. If PCIE_PHY_CFG is enabled and EXT_FLASH_CFG_EN is also enabled, the 820x will respond with the PHY values that are stored in the programmable device.

When programming the PHY settings into the programmable device, it is important to pay attention to the endian positions of these values.

Expansion ROM BAR

The Expansion ROM BAR is used by the host to directly access the programmable device attached to the 820x. The host reads the expansion ROM BAR value from the 820x device during initialization. The value that the 820x responds with depends on the setting of the

pin EXT_FLASH_CFG_EN. If EXT_FLASH_CFG_EN is enabled and the programmable device address 0x0C bit 0 is set to1 (expansion ROM enabled), the 820x will respond with the settings in the "Expansion ROM Base Address Register". If EXT_FLASH_CFG_EN is disabled, the 820x does not support expansion ROM BAR access.

5.12.1 SPI Register Operation Flow

Prior to configuring any other SPI registers, it is necessary to enable the Serial Peripheral Interface in the SPI Enable Register. Setting the SPI_EN bit in the "SPI Enable Register" enables the clock to the SPI controller module. Once the SPI module receives the clock, the host may access the programmable device, regardless of the settings of the pins EXT_FLASH_CFG_EN and PCIE_PHY_CFG.

If not using a recommended programmable device, the command definitions must be set once in the "SPI Command Configuration 0 Register" and "SPI Command Configuration 1 Register" during initialization by the host software. If using a recommended programmable device, these registers do not need to be set because the 820x will implement the default programmable device command definitions. Setting the SPI Command Address Register triggers the 820x SPI module to process the current SPI command.

The following diagram illustrates the process flow from the host point of view for accessing a programmable device.

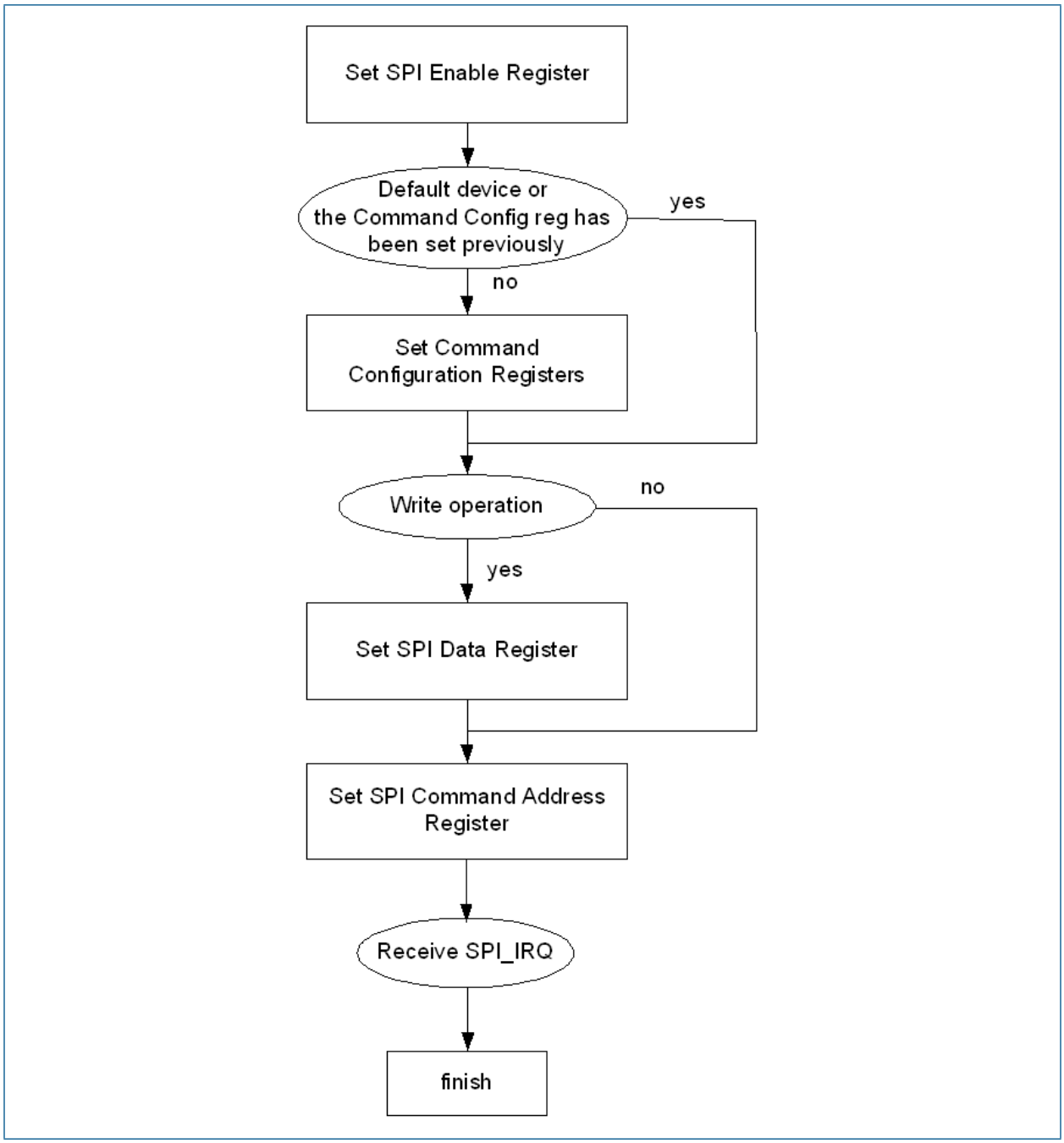


Figure 5-13. SPI Operation Example from Host Point of View

The sequence to write 4 bytes of data to the programmable device would be:

1. Enable the Serial Peripheral Interface using the "SPI Enable Register".

2. If not using a recommended programmable device, set the WREN[7:0], PP[7:0], and RDSR[7:0] command definitions in the "SPI Command Configuration 0 Register". This step can be skipped if using a recommended programmable device or if this register was already configured during initialization.

3. Write the 4 bytes of data to be programmed into the programmable device into the "SPI Data Register".

4. Set the CS, SPI_CMD[2:0], SPI_CLK_DIV[2:0], SPI_ADDR[23:0] fields in the "SPI Command Address Register". The 820x SPI module will then perform the write command.

5. When the write command is done, the 820x SPI module will issue a SPI_IRQ interrupt to the host, and simultaneously set the SPI_OP_DONE field and FLASH_WR_STATUS[7:0] field in the "SPI Status Register".

The sequence to read 4 bytes of data from the programmable device would be:

1. Enable the Serial Peripheral Interface using the "SPI Enable Register".

2. If not using a recommended programmable device, set the command definitions READ[7:0] in the "SPI Command Configuration 1 Register", and RDSR[7:0] in the "SPI Command Configuration 0 Register". This step can be skipped if using a recommended programmable device or if these registers were already configured during initialization.

3. Set the CS, SPI_CMD[2:0], SPI_CLK_DIV[2:0], and SPI_ADDR[23:0] fields in the "SPI Command Address Register". The 820x SPI module will then perform the read command.

4. When the read command is done, the 820x SPI module will issue a SPI_IRQ interrupt to the host, and simultaneously set the SPI_OP_DONE field in the "SPI Status Register". The host software can the read the data from the "SPI Data Register".

The sequence to erase data from the programmable device would be:

1. Enable the Serial Peripheral Interface using the "SPI Enable Register".

2. If not using a recommended programmable device, set the command definitions for WREN[7:0] and RDSR[7:0] fields in the "SPI Command Configuration 0 Register", and the BE[7:0] or SE[7:0] fields in the "SPI Command Configuration 1 Register". This step can be skipped if using a recommended programmable device or if these registers were already configured during initialization.

3. Set the CS, SPI_CMD[2:0], SPI_CLK_DIV[2:0], and SPI_ADDR[23:0] fields in the "SPI Command Address Register". The 820x SPI module will then perform the erase command.

4. When the erase command is done, the 820x SPI module will issue a SPI_IRQ interrupt to the host, and simultaneously set the SPI_OP_DONE and FLASH_WR_STATUS[7:0] (an erase command writes zeroes to the programmable device) fields in the "SPI Status Register".

The sequence to write a user defined command to the programmable device would be:

1. Enable the Serial Peripheral Interface using the "SPI Enable Register".

2. If not using a recommended programmable device, set the SPI_USER_CMD[7:0] field in the "SPI User Defined Register".

3. Set the CS, SPI_CMD[2:0], SPI_CLK_DIV[2:0], and SPI_ADDR[23:0] fields in the "SPI Command Address Register". The 820x SPI module will then perform the command.

4. When the command is done, the 820x SPI module will issue a SPI_IRQ interrupt to the host, and simultaneously set the SPI_OP_DONE field in the "SPI Status Register".

5.13 Temperature Sensor Controller

The 820x contains an internal temperature sensor that measures the die temperature using measured voltages from internal analog circuitry. These voltages are converted to a signal value through an ADC, allowing the values to be read by the host.

The 820x also contains a controller for the temperature sensor. The host may read/write directly to the 820x temperature sensor controller (TSC) registers. In contrast, the temperature sensor registers may only be accessed indirectly by the host via the temperature sensor controller registers. Refer to Section 6.8, "Temperature Sensor Controller Registers" for a complete description of the TSC registers.

The flow diagram in Figure 5-14 illustrates the procedure for measuring and calculating the 820x die temperature using the temperature sensor controller registers to read and write to the temperature sensor registers. Table 5-3 defines the TSC addresses that should be used to access the temperature sensor registers.

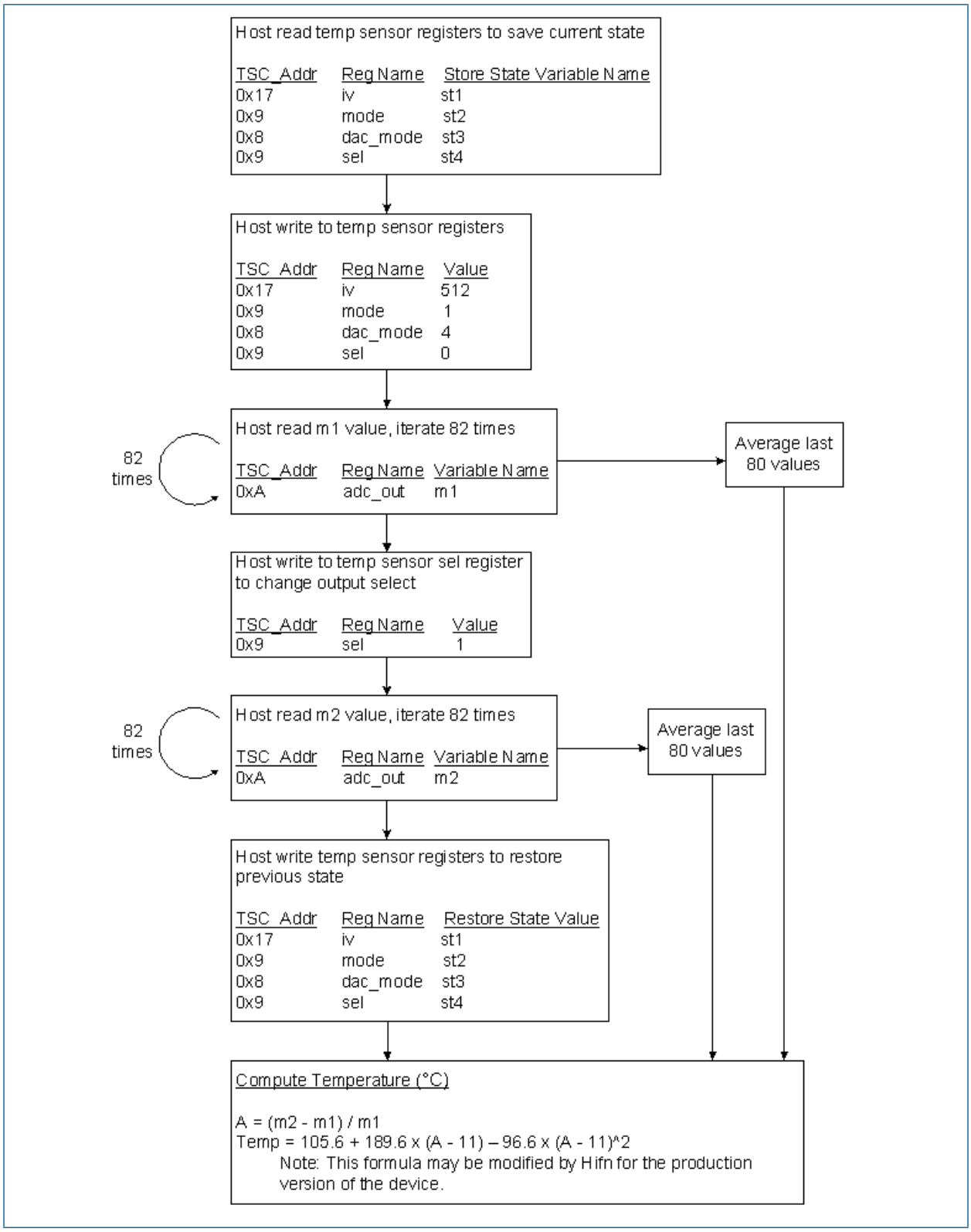


Figure 5-14. Die Temperature Measurement Procedure

Table 5-3. Temperature Sensor Register Map

Temperature Sensor Register Name	TSC_Addr	Bit Field
iv	0x17	[12:2]
mode	0x9	[1:0]
dac_mode	0x8	[14:12]
sel	0x9	[4]
adc_out	0xA	[9:0]

Note: The interval between two consecutive reads must be greater than 100 µs.

As shown in Figure 5-14, two equations are used to determine the die temperature:

```
a = (m2 - m1)/m1
Temperature = 105.6 + 189.6 x (a - 1.1) - 96.6 x (a - 1.1)^2
```

The host software can read the "m1" and "m2" values from the TSC Data register (Section 6.8.2), calculate the average over the last 80 values, and then calculate the temperature using the equations listed above.

For example, to read from a temperature sensor register:

1. Write the temperature sensor register address to TSC Address (0x8F0).
2. Write 0x4 to the TSC Command Register (0x8F8).
3. Poll the TSC_OP_Done bit in the TSC Command Register (0x8F8) until set.
4. Read the TSC Data Register (0x8F4) to get the temperature sensor value.

To write to a temperature sensor register

1. Write the value to be written to the temperature sensor register to the TSC Data Register (0x8F4).
2. Write the temperature sensor register address to the TSC Address Register (0x8F0).
3. Write 0x2 to the TSC Command Register (0x8F8).
4. Poll the TSC_OP_Done bit in the TSC Command Register (0x8F8) until set.

6 Register Definition

This section describes the 820x registers for configuration, DMA control, PKP control, and engine configuration registers.

The registers are mapped into 4K bytes of PCIe memory space that can be accessed through the PCIe bus. The absolute register address can be calculated using:

PCIe Address = PCIe base address + offset

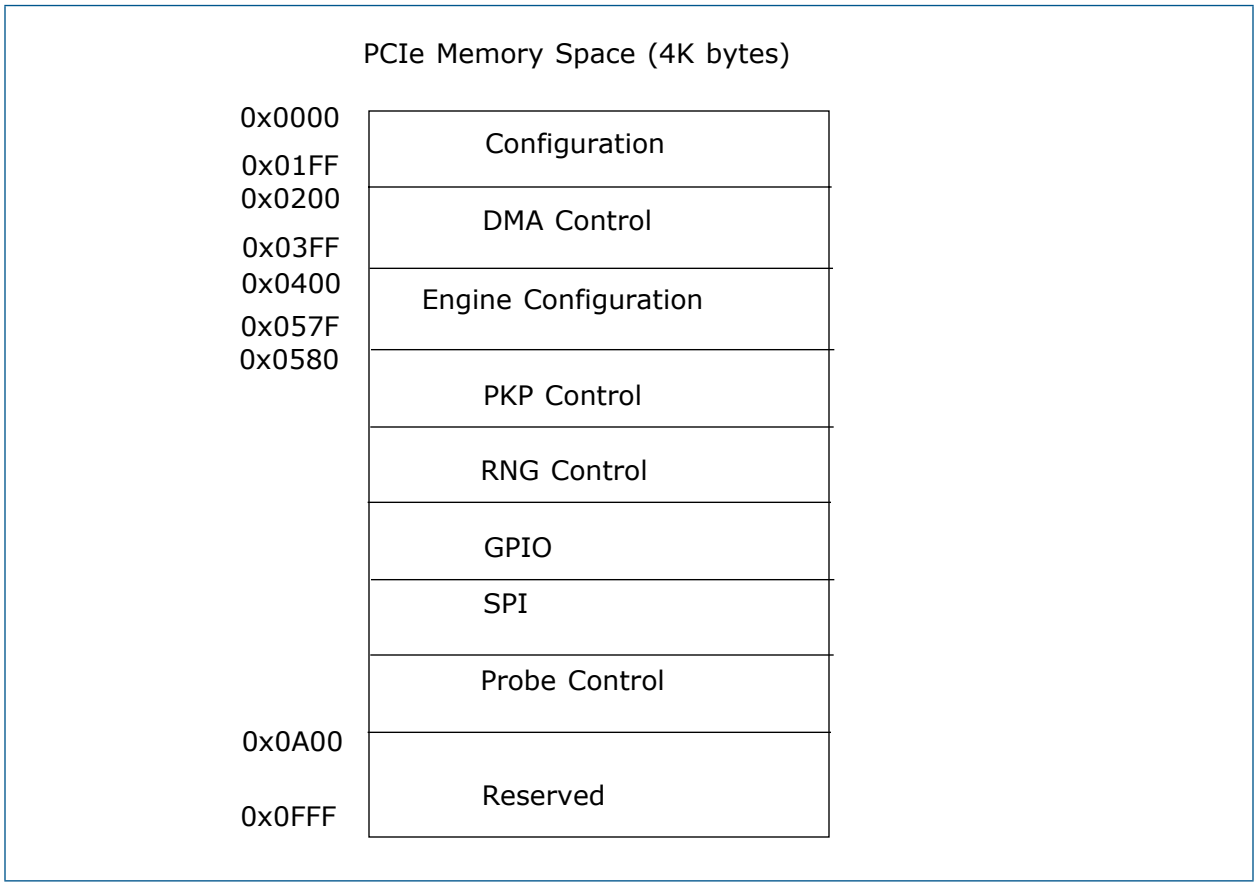


Figure 6-1. PCIe Memory Map

The host should not read/write to/from reserved registers. If the host writes to a reserved register, the write will be ignored by the 820x. Likewise, if the host reads a reserved register, the return value will be all zeros.

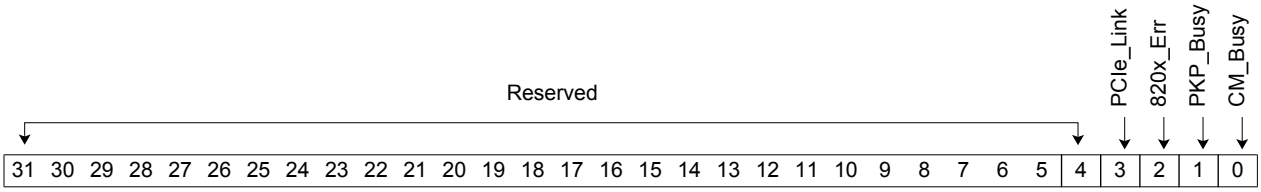
6.1 Configuration Registers

6.1.1 Status Register

The Status register provides the 820x internal status to host. These signals may be connected to LEDs as status indicators.

Type: Read only

Offset x‘0000’



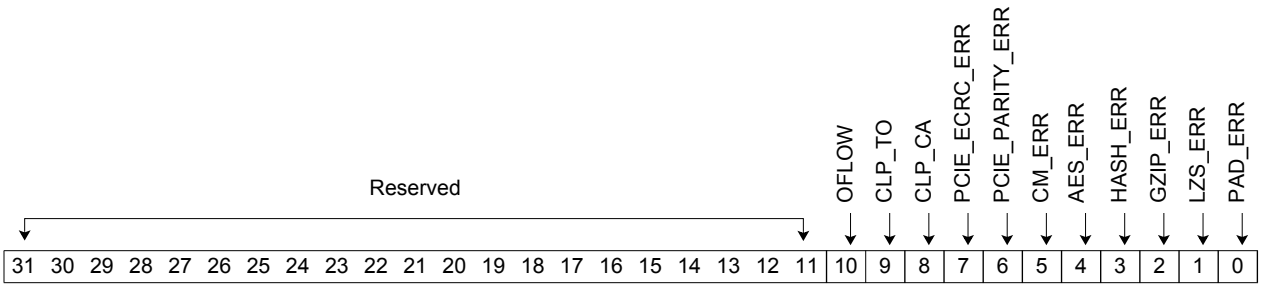
Field Name	Bits	Reset	Description
Reserved	31:4	0	Reserved.
PCIe_Link	3	0	PCIe Link Status. 0 PCIe link is down and not operational 1 PCIe link is up and operational
820x_Err	2	0	820x Status. 0 No 820x errors detected 1 820x error detected
PKP_Busy	1	0	Public Key Processor Status. 0 PKP Manager and PKP Engine not busy 1 PKP Manager and PKP Engine busy Note: the host software may only disable the PKP when the PKP is not busy.
CM_Busy	0	0	Channel Manager Status. 0 Channel Manager not busy 1 Channel Manager busy Note: the host software may only disable the Channel Manager when the Channel Manager is not busy.

6.1.2 820x Error Register

The 820x Error register provides additional information to the host if an 820x error occurs.

Type: Read/Write one to clear

Offset x‘0004’



Field Name	Bits	Reset	Description
Reserved	31:11	0	Reserved.
OFLOW	10	0	Destination buffer overflow. 0 No destination buffer overflow detected 1 Destination buffer overflow detected
CLP_TO	9	0	Completion Timeout. This bit will be set if outstanding read requests for completion packets do not return to the 820x within the timeout window valued defined in the PCIe specification. 0 No completion timeout error detected 1 Completion timeout error detected
CLP_CA	8	0	Host Completion Abort. This will be set if the host could not respond to a read request issued by the 820x. 0 No completion abort error detected 1 Completion abort error detected
PCIE_ECRC_ERR	7	0	PCIe Core ECRC Error. 0 No ECRC error detected by PCIe Core 1 ECRC error detected by PCIe Core
PCIE_PARITY_ERR	6	0	PCIe Core Parity Error. 0 No PCIe Core Parity error detected 1 PCIe Core Parity error detected

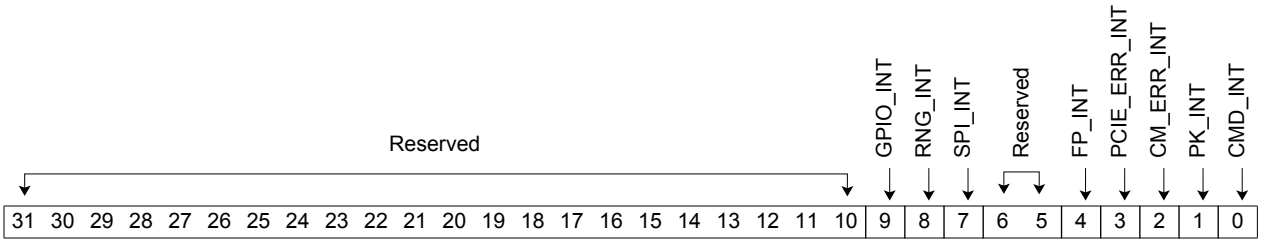
Field Name	Bits	Reset	Description
CM_ERR	5	0	Command Error. This bit will be set if any error is detected by the Channel Manager (see Section 6.2.15, "Channel Manager 0-1 Error Status Register") 0 No command error detected 1 Command error detected
AES_ERR	4	0	Encryption Engine Error. 0 No Encryption Engine error detected 1 Encryption Engine error detected
HASH_ERR	3	0	Hash Engine Error. 0 No Hash Engine error detected 1 Hash Engine error detected
GZIP_ERR	2	0	GZIP Engine Error. 0 No GZIP Engine error detected 1 GZIP Engine error detected
LZS_ERR	1	0	LZS Engine Error. 0 No LZS Engine error detected 1 LZS Engine error detected
PAD_ERR	0	0	PAD Engine Error. 0 No PAD Engine error detected 1 PAD Engine error detected

6.1.3 820x Interrupt Status Register

The 820x Interrupt Status register provides the 820x interrupt status to the host. Once an interrupt is set, it may be cleared by the host by either writing a one to that bit, by a hardware reset or a Miscellaneous Soft Reset (MISC_RST fromSection 6.1.5).

Type: Read/Write one to clear

Offset x‘0008’



Field Name	Bits	Reset	Description
Reserved	31:10	0	Reserved.
GPIO_INT	9	0	GPIO Interrupt. 0 No GPIO interrupt occurred 1 GPIO interrupt occurred
RNG_INT	8	0	Random Number Generator Interrupt. 0 No RNG interrupt occurred 1 RNG interrupt occurred
SPI_INT	7	0	Serial Peripheral Interface Interrupt. 0 No SPI interrupt occurred 1 SPI interrupt occurred
Reserved	6:5	0	Reserved.
FP_INT	4	0	Free Pool Interrupt. This interrupt will be set when a free pool descriptor is used by the 820x. 0 No Free Pool interrupt occurred 1 Free Pool interrupt occurred
PCIE_ERR_INT	3	0	PCIe Error Interrupt. The 820x was not able to recover from a PCIe error. 0 No PCIe Error interrupt occurred 1 PCIe Error interrupt occurred
CM_ERR_INT	2	0	Command Error Interrupt. The 820x detected a command error but that error was corrected internally. 0 No Command Error interrupt occurred 1 Command Error interrupt occurred

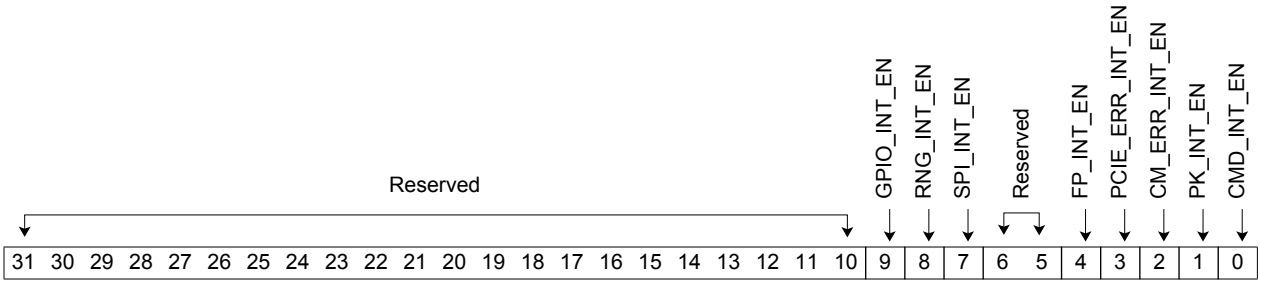
Field Name	Bits	Reset	Description
PK_INT	1	0	Public Key Processor Done Interrupt. 0 No PKP Done interrupt occurred 1 PKP Done interrupt occurred Warning: This interrupt should only be used for applications that perform only PK operations due to an errata associated with this bit (see the 820x Errata, ER-0015).
CMD_INT	0	0	Command Done Interrupt. Channel manager has completed a command. When this bit is set to one and the IRQ_EN bit in the command structure is set to one, the 820x will send this interrupt to the host after the command completes. 0 No Command Done interrupt occurred 1 Command Done interrupt occurred

6.1.4 820x Interrupt Enable Register

The 820x interrupt Enable register provides the capability for the host to enable/disable 820x interrupts. All interrupt enable bits in this register may be reset by a hardware reset or a Miscellaneous Soft Reset (MISC_RST fromSection 6.1.5). Refer to Section 6.1.3 for a detailed description of each interrupt.

Type: Read/Write

Offset x‘000C’



Field Name	Bits	Reset	Description
Reserved	31:10	0	Reserved.
GPIO_INT_EN	9	0	GPIO Interrupt Enable. 0 GPIO interrupt disabled 1 GPIO interrupt enabled
RNG_INT_EN	8	0	Random Number Generator Interrupt Enable. 0 RNG interrupt disabled 1 RNG interrupt enabled
SPI_INT_EN	7	0	Serial Peripheral Interface Interrupt Enable. 0 SPI interrupt disabled 1 SPI interrupt enabled
Reserved	6:5	0	Reserved.
FP_INT_EN	4	0	Free Pool Interrupt Enable. 0 Free Pool interrupt disabled 1 Free Pool interrupt enabled
PCIE_ERR_INT_EN	3	0	PCIe Error Interrupt Enable. 0 PCIe Error interrupt disabled 1 PCIe Error interrupt enabled
CM_ERR_INT_EN	2	0	Command Error Interrupt Enable. 0 Command Error interrupt disabled 1 Command Error interrupt enabled

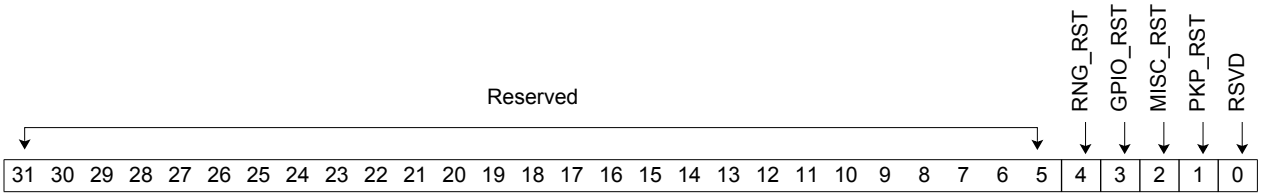
Field Name	Bits	Reset	Description
PK_INT_EN	1	0	Public Key Interrupt Enable. 0 Public Key interrupt disabled 1 Public Key interrupt enabled
CMD_INT_EN	0	0	Command Done Interrupt Enable. Channel manager has completed a command. 0 Command Done interrupt disabled 1 Command Done interrupt enabled

6.1.5 Soft Reset Register

The Soft Reset register provides the host the capability to reset the 820x during error recovery. The Soft Reset register will be cleared by the 820x after the reset is complete. After a soft reset, the 820x will wait 64 clock cycles to ensure that all modules are reset.

Type: Read/Write

Offset x‘0018’



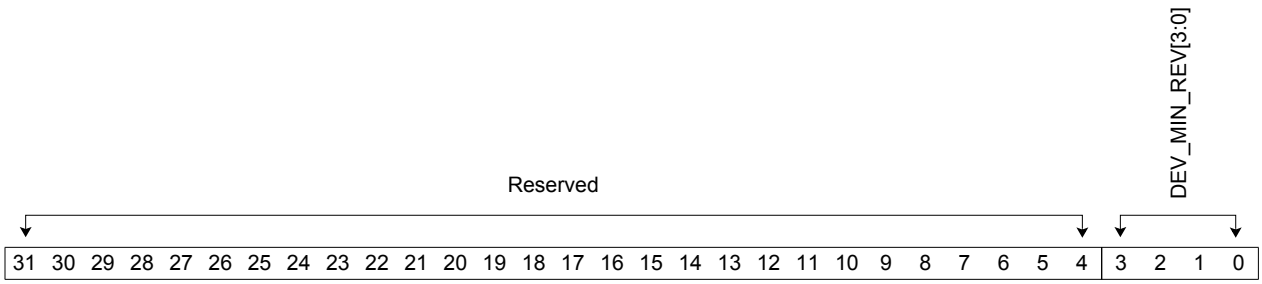
Field Name	Bits	Reset	Description
Reserved	31:5	0	Reserved.
RNG_RST	4	0	Random Number Generator Soft Reset. 0 Do not reset the RNG 1 Reset the RNG
GPIO_RST	3	0	General Purpose I/O Soft Reset. 0 Do not reset the GPIO 1 Reset the GPIO
MISC_RST	2	0	Miscellaneous Soft Reset. 0 Do not reset the misc modules 1 Reset all other modules except the GPIO, RNG, PKP Manager, PKP cores, PCIe core and PHY
PKP_RST	1	0	Public key Processor Soft Reset. 0 Do not reset the PKP 1 Reset the PKP manager and the two PKP pairs
Reserved	0	0	Reserved.

6.1.6 Device Minor Revision Register

The Device Minor Revision register reflects the device's minor revision number.

Type: Read only

Offset x'0020



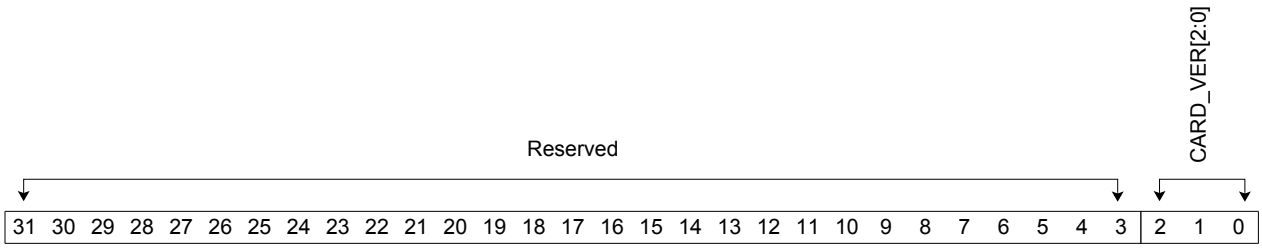
Field Name	Bits	Reset	Description
Reserved	31:4	0	Reserved.
DEV_MIN_REV[3:0]	3:0	0	Device Minor Revision Number. Please refer to the 820x Errata document, ER-0015, for the difference between the 820x revisions. 0000 Preliminary release 0001 First production release

6.1.7 Card Version Register

The Card Version register allows the host to read the version number of the card. The card version is read from pins BOARD_VERSION[2:0] that can be pulled high or low on the card.

Type: Read only

Offset x‘0100’



Field Name	Bits	Reset	Description
Reserved	31:3	0	Reserved.
CARD_VER[2:0]	2:0	0	Card Version.

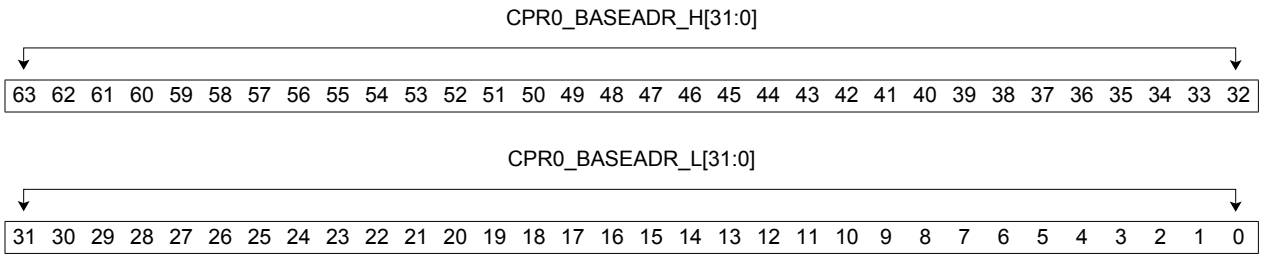
6.2 DMA Control Registers

6.2.1 Command Pointer Ring 0 Base Address Register

This register is the base address of the command pointer ring (CPR) 0 in host memory. The 820x uses this base address plus an offset to fetch a command pointer, and then uses the command pointer to fetch the corresponding command structure. The CPR base address must be 8-byte aligned, and the command pointer in CPR0 must be 512-byte aligned.

Type: Read/Write

Offset x‘0200’



Field Name	Bits	Reset	Description
CPR0_BASEADR_H [31:0]	63:32	0	Command pointer ring 0 base address high double word
CPR0_BASEADR_L [31:0]	31:0	0	Command pointer ring 0 base address low double word

6.2.2 Command Pointer Ring 0 Write Pointer Register

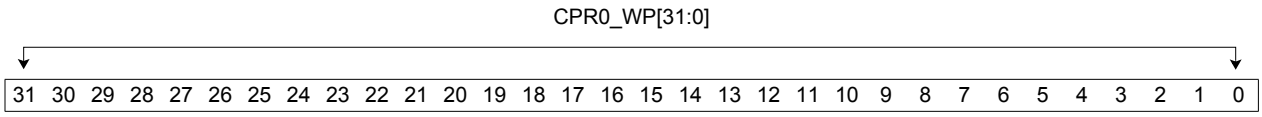
This register is a copy of the command pointer ring 0 write pointer in host memory. The host must update this register whenever new commands are added into the command pointer ring 0.

The 820x Command Ring 0 Write Pointer register stores the index number (i.e., 1, 2, 3…) of the corresponding command ring write pointer in host memory. The host updates this register whenever a new command is submitted to Command Ring 0. The 820x uses the stored index number (i.e., 1, 2, 3) to refer to the Command Pointer Ring 0 Base Address register (Section 6.2.1), and calculate the actual address of the newly submitted command structure in Command Ring 0.

Please refer to Section 3.3, "Command Operation Sequence" for more information about this register.

Type: Read/Write

Offset x'0208'



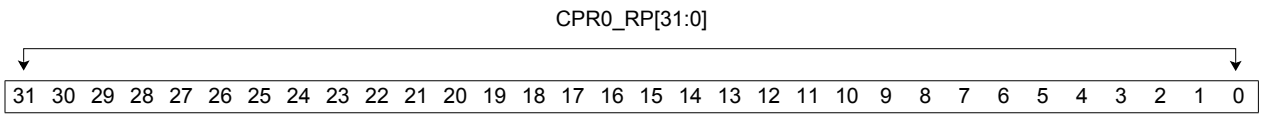
Field Name	Bits	Reset	Description
CPR0_WP[31:0]	31:0	0	Command pointer ring 0 write pointer

6.2.3 Command Pointer Ring 0 Read Pointer Register

This register is the 820x command pointer ring 0 read pointer. The host may read this register to determine how many entries in the command pointer ring 0 have been read by the 820x.

Type: Read/Write

Offset x'020C'



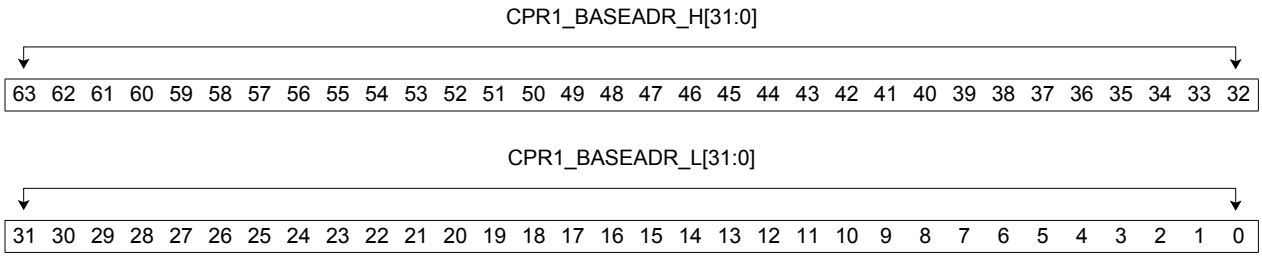
Field Name	Bits	Reset	Description
CPR0_RP[31:0]	31:0	0	Command pointer ring 0 read pointer

6.2.4 Command Pointer Ring 1 Base Address Register

This register is the base address of the command pointer ring (CPR) 1 in host memory. The 820x uses this base address plus an offset to fetch a command pointer, and then uses the command pointer to fetch the corresponding command structure. The CPR base address must be 8-byte aligned, and the command pointer in CPR1 must be 512-byte aligned.

Type: Read/Write

Offset x‘0210’



Field Name	Bits	Reset	Description
CPR1_BASEADR_H [31:0]	63:32	0	Command pointer ring 1 base address high double word
CPR1_BASEADR_L [31:0]	31:0	0	Command pointer ring 1 base address low double word

6.2.5 Command Pointer Ring 1 Write Pointer Register

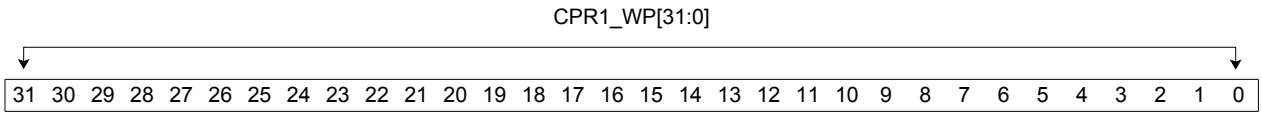
This register is a copy of the command pointer ring 1 write pointer in host memory. The host must update this register whenever new commands are added into the command pointer ring 1.

The 820x Command Ring 1 Write Pointer register stores the index number (i.e., 1, 2, 3…) of the corresponding command ring write pointer in host memory. The host updates this register whenever a new command is submitted to Command Ring 1. The 820x uses the stored index number (i.e., 1, 2, 3) to refer to the Command Pointer Ring 1 Base Address register (Section 6.2.4), and calculate the actual address of the newly submitted command structure in Command Ring 0.

Please refer to Section 3.3, "Command Operation Sequence" for more information about this register.

Type: Read/Write

Offset x'0218'



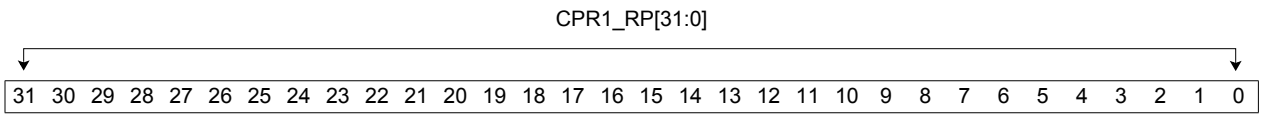
Field Name	Bits	Reset	Description
CPR1_WP[31:0]	31:0	0	Command pointer ring 1 write pointer

6.2.6 Command Pointer Ring 1 Read Pointer Register

This register is the 820x command pointer ring 1 read pointer. The host may read this register to determine how many entries in the command pointer ring 1 have been read by the 820x.

Type: Read/Write

Offset x'021C'



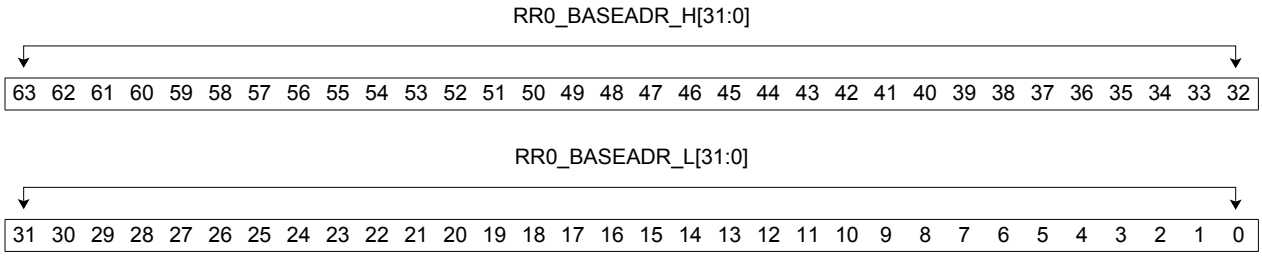
Field Name	Bits	Reset	Description
CPR1_RP[31:0]	31:0	0	Command pointer ring 1 read pointer

6.2.7 Result Ring 0 Base Address Register

This register is the result ring 0 base address in host memory. After the 820x finishes a command, it will write the result to the result ring. The Result Ring (RR) base address must be 8-byte aligned.

Type: Read/Write

Offset x'0220'



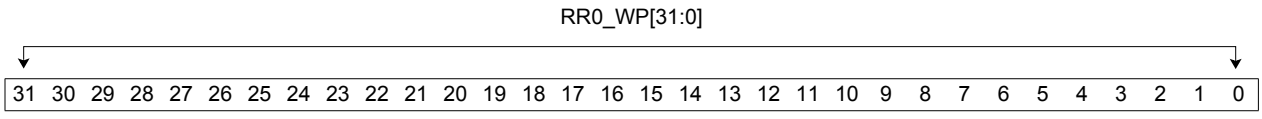
Field Name	Bits	Reset	Description
RR0_BASEADR_H [31:0]	63:32	0	Result ring 0 base address high double word
RR0_BASEADR_L [31:0]	31:0	0	Result ring 0 base address low double word

6.2.8 Result Ring 0 Write Pointer Register

This register is the 820x result ring 0 write pointer. The 820x automatically increments this register after successfully writing to the Result Ring 0. The host does not need to write to this register once the 820x is initialized. The host can read this register to determine the how many entries the 820x has written to the Result Ring 0. There is no corresponding Result Ring 0 Read Pointer register.

Type: Read/Write

Offset x'0228'



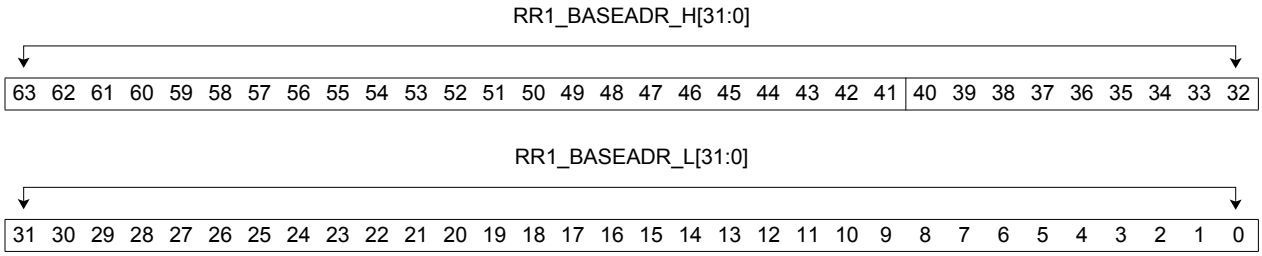
Field Name	Bits	Reset	Description
RR0_WP[31:0]	31:0	0	Result ring 0 write pointer.

6.2.9 Result Ring 1 Base Address Register

This register is the result ring 1 base address in host memory. After the 820x finishes a command, it will write the result to the result ring. The Result Ring (RR) base address is 8-byte aligned.

Type: Read/Write

Offset x‘0230’



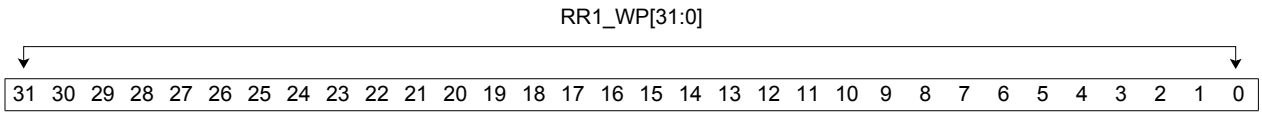
Field Name	Bits	Reset	Description
RR1_BASEADR_H [31:0]	63:32	0	Result ring 1 base address high double word.
RR1_BASEADR_L [31:0]	31:0	0	Result ring 1 base address low double word.

6.2.10 Result Ring 1 Write Pointer Register

This register is the 820x result ring 1 write pointer. The 820x automatically increments this register after successfully writing to the Result Ring 1. The host does not need to write to this register once the 820x is initialized. The host can read this register to determine the how many entries the 820x has written to the Result Ring 1. There is no corresponding Result Ring 1 Read Pointer register.

Type: Read/Write

Offset x‘0238’



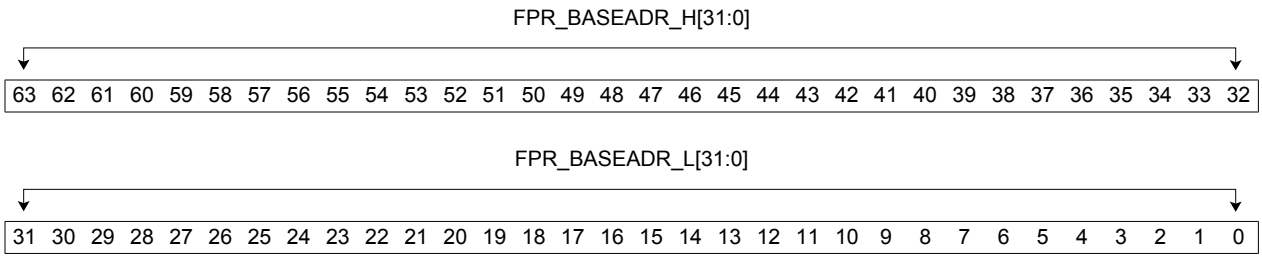
Field Name	Bits	Reset	Description
RR1_WP[31:0]	31:0	0	Result ring 1 write pointer.

6.2.11 Free Pool Ring Base Address Register

This register is the base address of the Free Pool Ring (FPR). The 820x uses this base address plus an offset to fetch a free pool entry.

Type: Read/Write

Offset x‘0240’



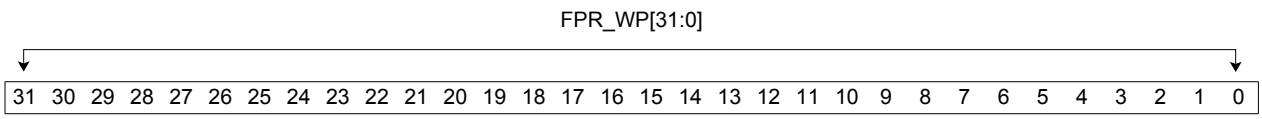
Field Name	Bits	Reset	Description
FPR_BASEADR_H [31:0]	63:32	0	Free pool ring base address high double word.
FPR_BASEADR_L [31:0]	31:0	0	Free pool ring base address low double word.

6.2.12 Free Pool Write Pointer Register

This register is a copy of the Free Pool Ring write pointer in host memory. The host must update this register when a new entry is added to the free pool ring.

Type: Read/Write

Offset x‘0248’



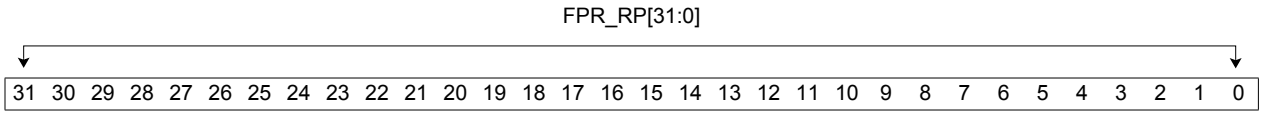
Field Name	Bits	Reset	Description
FPR_WP[31:0]	31:0	0	Free pool write pointer.

6.2.13 Free Pool Read Pointer Register

This register is the 820x Free Pool Ring read pointer. The host may read this register to determine how many free pool ring entries have been read by the 820x.

Type: Read only

Offset x‘024C’



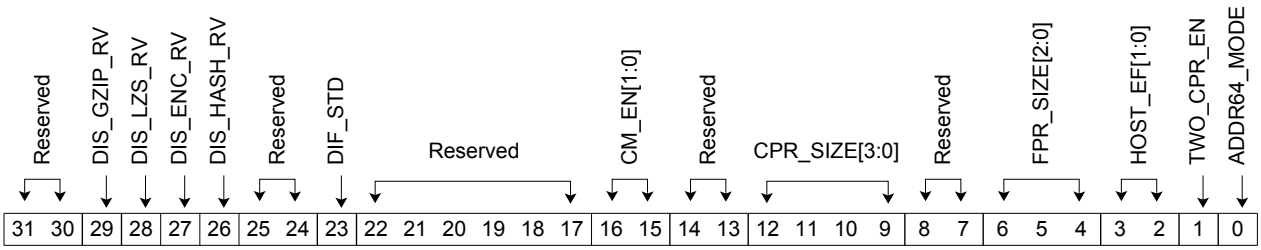
Field Name	Bits	Reset	Description
FPR_RP[31:0]	31:0	0	Free pool read pointer.

6.2.14 DMA Configuration Register

This register is used to set the DMA configuration parameters. The host must configure this register during system initialization.

Type: Read/Write

Offset x‘0250’



Field Name	Bits	Reset	Description
Reserved	31:30	0	Reserved.
DIS_GZIP_RV	29	0	Disable GZIP engine real time verification. When the GZIP real time verification core is disabled, the GZIP clock is also gated to save power. 0 Enable GZIP engine real time verification 1 Disable GZIP engine real time verification
DIS_LZS_RV	28	0	Disable LZS engine real time verification. When the LZS real time verification core is disabled, the LZS clock is also gated to save power. 0 Enable LZS engine real time verification 1 Disable LZS engine real time verification
DIS_ENC_RV	27	0	Disable Encryption engine real time verification. When the ENC real time verification core is disabled, the ENC clock is also gated to save power. 0 Enable ENC engine real time verification 1 Disable ENC engine real time verification
DIS_HASH_RV	26	0	Disable Hash engine real time verification. When the HASH real time verification core is disabled, the HASH clock is also gated to save power. 0 Enable HASH engine real time verification 1 Disable HASH engine real time verification
Reserved	25:24	0	Reserved.

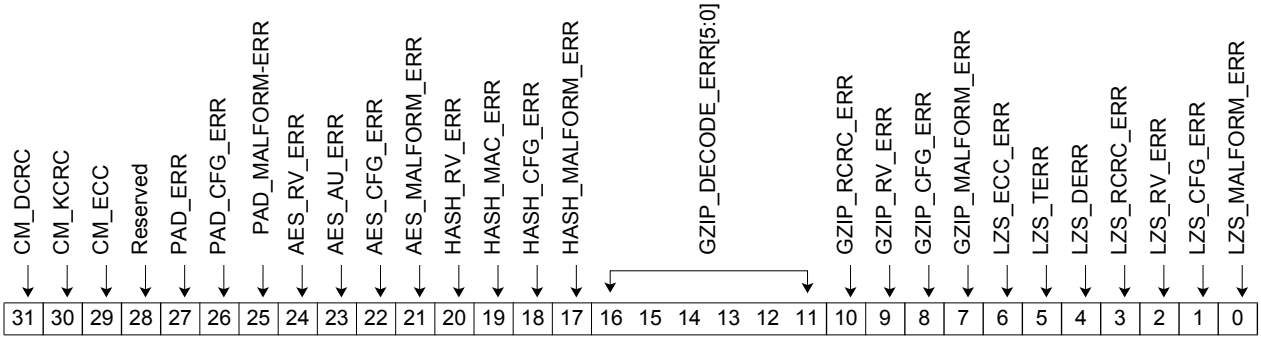
Field Name	Bits	Reset	Description
DIF_STD	23	0	DIF Standard Format Used to select the data integrity format (DIF) for AES-XTS mode, in particular the CRC byte order. 0 T10/03-310r0 1 T10/08-044r1 SBC-3
Reserved	22:17	0	Reserved.
CM_EN[1:0]	16:15	0	Channel Manager Enable. If set, the Channel Manager is enabled to process commands, otherwise, the Channel Manager will not process commands and its clock is disabled. 00 Disable Channel Manager 0 and Channel Manager 1 01 Enable Channel Manager 0 10 Enable Channel Manager 1 11 Enable Channel Manager 0 and Channel Manager 1
Reserved	14:13	0	Reserved.
CPR_SIZE[3:0]	12:9	0	Command Pointer Ring Size. Sets the maximum number of commands in the command pointer ring. Note: the command pointer ring size value is the same for command pointer ring 0 and command pointer ring 1. 0000 32 0001 64 0010 128 0011 256 0100 512 0101 1K 0110 2K 0111 4K 1000 8K 1001 16K 1010 - 1111 Reserved
Reserved	8:7	0	Reserved.

Field Name	Bits	Reset	Description
FPR_SIZE[2:0]	6:4	0	Free Pool Ring Size. Sets the maximum number of entries in the free pool ring. Note: the free pool is shared by commands in the CPR0 or CPR1. 000 4 001 8 010 16 011 32 100 64 101 128 110 256 111 512
HOST_EF[1:0]	3:2	0	Host Endian Format. Sets the endian format of the host command pointer ring, command structure, result ring and free pool ring. 00 No swap 01 Byte swap in word 10 Word swap, no byte swap in word 11 Word swap and byte swap in word
TWO_CPR_EN	1	0	Two Command Ring Pointer Enable. 0 Only Command Ring Pointer 0 enabled 1 Both Command Ring Pointer 0 and 1 enabled
ADDR64_MODE	0	0	DMA Addressing Mode. 0 32-bit addressing mode 1 64-bit addressing mode

6.2.15 Channel Manager 0-1 Error Status Register

If a 820x Channel Manager detects an error, the 820x will set the appropriate bit in this register. The Host may read the Channel Manager Error Status register to determine the source of the error. This information may also be read from the entry in the result ring. Because the Channel Managers continue to process commands after an error, the error bits in this register may be overwritten. It is recommended that the host validates the error bit with the result ring error from the command.

Type:	Read/Write to clear	
Offset	x'0260'	Channel Manager 0 error status
	x'0264'	Channel Manager 1 error status



Field Name	Bits	Reset	Description
CM_DCRC	31	0	Channel Manager Data CRC error. 0 Channel Manager did not detect data CRC error 1 Channel Manager detected data CRC error
CM_KCRC	30	0	Channel Manager Key CRC error. 0 Channel Manager did not detect key CRC error 1 Channel Manager detected key CRC error
CM_ECC	29	0	Channel Manager ECC/Parity error. 0 Channel Manager did not detect ECC/Parity error on this channel 1 Channel Manager detected ECC/Parity error on this channel
Reserved	28	0	Reserved.
PAD_ERR	27	0	Pad Engine Padding Error. 0 Pad Engine did not detect padding error 1 Pad Engine detected padding error

Field Name	Bits	Reset	Description
PAD_CFG_ERR	26	0	Pad Engine Software Configuration Error. 0 Pad Engine did not detect software configuration error 1 Pad Engine detected software configuration error
PAD_MALFORM_ERR	25	0	Pad Engine Malformed Packet Error. 0 Pad Engine did not detect malformed packet error 1 Pad Engine detected malformed packet error
AES_RV_ERR	24	0	Encryption Engine Real Time Verification Error. 0 Encryption engine did not detect real time verification error 1 Encryption engine detected real time verification error
AES_AU_ERR	23	0	Authentication Error in AES-GCM Mode. 0 Authentication error in AES-GCM mode not detected 1 Authentication error in AES-GCM mode detected
AES_CFG_ERR	22	0	Encryption Engine Software Configuration Error. 0 Encryption Engine did not detect software configuration error 1 Encryption Engine detected software configuration error
AES_MALFORM_ERR	21	0	Encryption Engine Malformed Packet Error. 0 Encryption Engine did not detect malformed packet error 1 Encryption Engine detected malformed packet error
HASH_RV_ERR	20	0	Hash Engine Real Time Verification Error or HMAC Error. 0 Hash engine did not detect real time verification or HMAC error 1 Hash engine detected real time verification or HMAC error
HASH_MAC_ERR	19	0	Hash Engine MAC Check Error. 0 Hash engine did not detect MAC check error 1 Hash engine detected MAC check error
HASH_CFG_ERR	18	0	Hash Engine Software Configuration Error. 0 Hash Engine did not detect software configuration error 1 Hash Engine detected software configuration error

Field Name	Bits	Reset	Description
HASH_MALFORM_ERR	17	0	Hash Engine Malformed Packet Error. 0 Hash Engine did not detect malformed packet error 1 Hash Engine detected malformed packet error
GZIP_DECODE_ERR [5:0]	16:11	0	GZIP Decode Error. For a description of this field, please refer to Table 3-5 on page 98.
GZIP_RCRC_ERR	10	0	GZIP Engine RCRC Error. 0 GZIP engine did not detect RCRC error in decode operation 1 GZIP engine detected RCRC error in decode operation
GZIP_RV_ERR	9	0	GZIP Engine Real Time Verification Error. 0 GZIP engine did not detect real time verification error 1 GZIP engine detected real time verification error
GZIP_CFG_ERR	8	0	GZIP Engine Software Configuration Error. 0 GZIP Engine did not detect software configuration error 1 GZIP Engine detected software configuration error
GZIP_MALFORM_ERR	7	0	GZIP Engine Malformed Packet Error. 0 GZIP Engine did not detect malformed packet error 1 GZIP Engine detected malformed packet error
LZS_ECC_ERR	6	0	LZS Engine ECC Error. 0 LZS engine did not detect ECC error 1 LZS engine detected ECC error
LZS_TERR	5	0	LZS Engine Token Error. 0 LZS engine did not detect token error 1 LZS engine detected token error
LZS_DERR	4	0	LZS Engine Data Error. 0 LZS engine did not detect data error 1 LZS engine detected data error
LZS_RCRC_ERR	3		LZS Engine RCRC Error. 0 LZS engine did not detect RCRC error in decode operation 1 LZS engine detected RCRC error in decode operation

Field Name	Bits	Reset	Description
LZS_RV_ERR	2		LZS Engine Real Time Verification Error. 0 LZS engine did not detect real time verification error 1 LZS engine detected real time verification error
LZS_CFG_ERR	1	0	LZS Engine Software Configuration Error. 0 LZS Engine did not detect software configuration error 1 LZS Engine detected software configuration error
LZS_MALFORM_ERR	0	0	LZS Engine Malformed Packet Error. 0 LZS Engine did not detect malformed packet error 1 LZS Engine detected malformed packet error

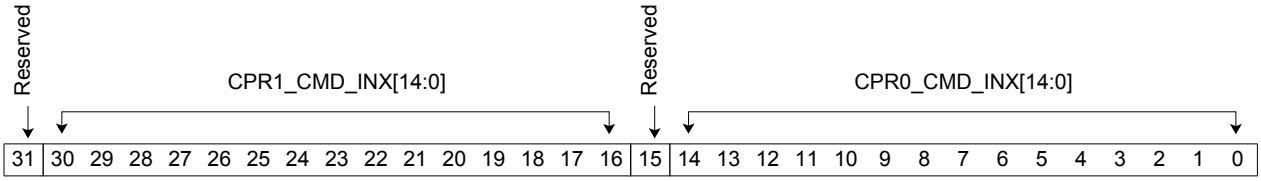
6.2.16 Channel Manager 0-1 Error Command Index Register

The Channel Manager 0-1 Error Command Index register records the last processing command index when the PCIe related error occurred. This record can be used to facilitate software error recovery.

Type: Read only

Offset x‘0268’ Channel Manager 0 error command index

x‘026C’ Channel Manager 1 error command index



Field Name	Bits	Reset	Description
Reserved	31	0	Reserved.
CPR1_CMD_INX[14:0]	30:16	0	Command Pointer Ring 1 Command Index [14:0].
Reserved	15	0	Reserved.
CPR0_CMD_INX[14:0]	14:0	0	Command Pointer Ring 0 Command Index [14:0]. If command ring size is 16K, bit 14 is the valid bit; If command ring size is 8K, bit 13 is the valid bit; If command ring size is 4K, bit 12 is the valid bit. A similar pattern may be applied for command ring sizes < 4K.

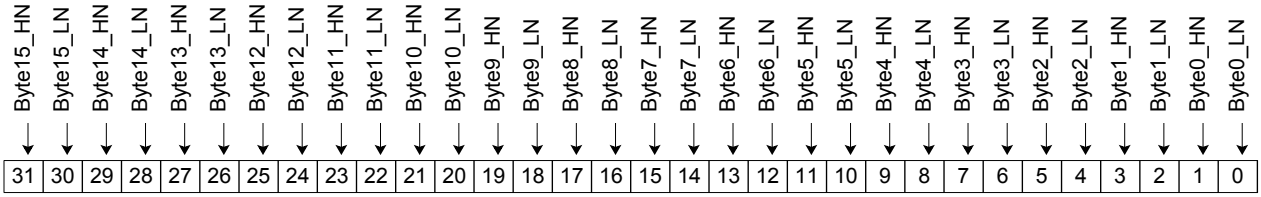
6.3 Engine Configuration Registers

6.3.1 Hash Engine Mute Table Entry Registers

The Hash Engine Mute Table Entry register provides a mask that controls the input to the Hash engine. The mask nulls specific segments of the data stream prior to being submitted to the Hash core. The masks are programmable by the host and are selected by the "MTI" field in command structure.

Each mute table entry is 32 bits wide to mute up to 16 bytes of the data stream. There are seven entries in the mute table. Each bit of the mute table entry masks one nibble (4-bit) of data, therefore a 32-bit register word supports a 16-byte header. The table below provides the map between the mute table entry bit and the 16-byte header nibble (4-bit).

Type:	Read/Write	
Offset	x'0400'	Mute table entry 1
	x'0404'	Mute table entry 2
	x'0408'	Mute table entry 3
	x'040C'	Mute table entry 4
	x'0410'	Mute table entry 5
	x'0414'	Mute table entry 6
	x'0418'	Mute table entry 7



Field Name	Bits	Reset	Description
Byte15_HN	31	0	Mute Data Stream Byte 15, bits [7:4] 0 Do not mute data stream byte 15, bits [7:4] 1 Mute data stream byte 15, bits [7:4]
Byte15_LN	30	0	Mute Data Stream Byte 15, bits [3:0] 0 Do not mute data stream byte 15, bits [3:0] 1 Mute data stream byte 15, bits [3:0]
. . .	. . .	. . .	. . .
Byte0_HN	1	0	Mute Data Stream Byte 0, bits [7:4] 0 Do not mute data stream byte 0, bits [7:4] 1 Mute data stream byte 0, bits [7:4]

Field Name	Bits	Reset	Description
Byte0_LN	0	0	Mute Data Stream Byte 0, bits [3:0]. 0 Do not mute data stream byte 0, bits [3:0] 1 Mute data stream byte 0, bits [3:0]

6.4 Public Key Processor Control Registers

The Public Key Processor (PKP) implements a group of registers, through which the host may control the PKP engine and execute complex computations. The PKP Manager fetches source data and instructions from host memory, and then sends the computation result to host memory according to the PKP command registers.

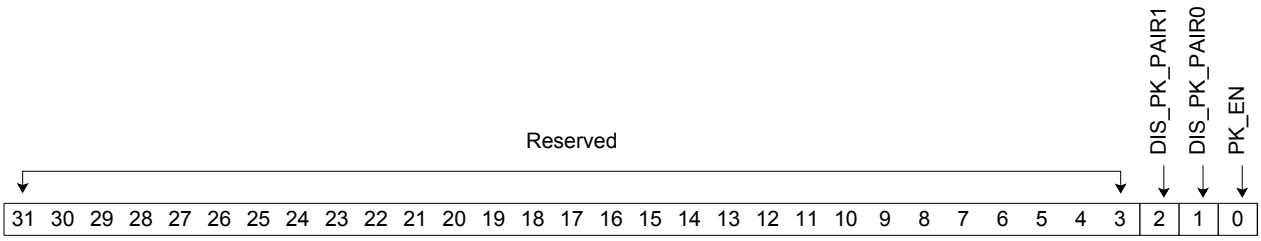
To increase performance, the 820x implements two pairs of PK cores; a pair consists of one master PK core and one slave PK core.

6.4.1 Public Key Enable Register

The Public Key Engine register is used to enable/disable either of the Public Key Engine pairs and the Public Key Engine.

Type: Read/Write

Offset x'0580'



Field Name	Bits	Reset	Description
Reserved	31:3	0	Reserved.
DIS_PK_PAIR1	2	0	Disable Public Key Pair 1. If the PK core 1 is disabled, the PK core 1 clock is also disabled to save power. 0 Enable Public Key Pair 1 1 Disable Public Key Pair 1
DIS_PK_PAIR0	1	0	Disable Public Key Pair 0. If the PK core 0 is disabled, the PK core 0 clock is disabled to save power. 0 Enable Public Key Pair 0 1 Disable Public Key Pair 0
PK_EN	0	0	Public Key Enable. Note: the host software should first enable the PKP, and then write commands to the PKP command entry. 0 Disable the PKP Manager and PKP Engine and clock 1 Enable the PKP Manager and PKP Engine

6.4.2 Public Key Command Entry Registers

The Public Key Command Entry registers control the eight PKP command entries in the 820x. Each entry includes one command register, one instruction register, one data register, and one result register as shown below.

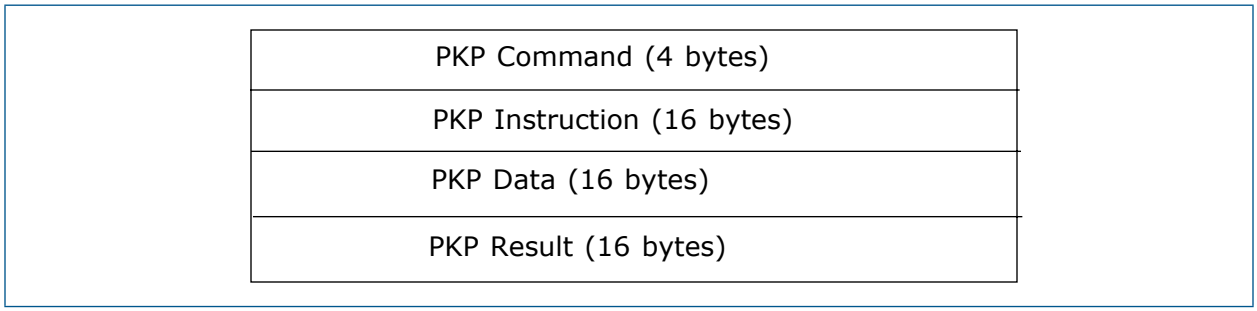
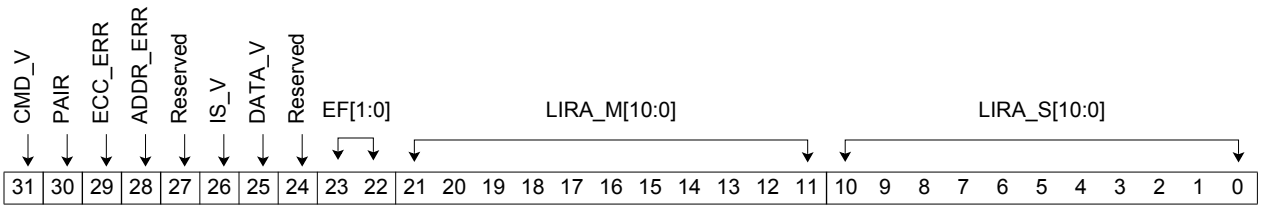


Figure 6-2. PKP Command Entry Format

Type:	Read/Write	
Offset	x‘0600’	Public Key entry 0 command register
	x‘0610’	Public Key entry 0 instruction register
	x‘0620’	Public Key entry 0 data register
	x‘0630’	Public Key entry 0 result register
	x‘0640’	Public Key entry 1 command register
	x‘0650’	Public Key entry 1 instruction register
	x‘0660’	Public Key entry 1 data register
	x‘0670’	Public Key entry 1 result register
	x‘0680’	Public Key entry 2 command register
	x‘0690’	Public Key entry 2 instruction register
	x‘06A0’	Public Key entry 2 data register
	x‘06B0’	Public Key entry 2 result register
	x‘06C0’	Public Key entry 3 command register
	x‘06D0’	Public Key entry 3 instruction register
	x‘06E0’	Public Key entry 3 data register
	x‘06F0’	Public Key entry 3 result register
	x‘0700’	Public Key entry 4 command register
	x‘0710’	Public Key entry 4 instruction register
	x‘0720’	Public Key entry 4 data register
	x‘0730’	Public Key entry 4 result register
	x‘0740’	Public Key entry 5 command register
	x‘0750’	Public Key entry 5 instruction register
	x‘0760’	Public Key entry 5 data register
	x‘0770’	Public Key entry 5 result register
	x‘0780’	Public Key entry 6 command register
	x‘0790’	Public Key entry 6 instruction register
	x‘07A0’	Public Key entry 6 data register
	x‘07B0’	Public Key entry 6 result register
	x‘07C0’	Public Key entry 7 command register
	x‘07D0’	Public Key entry 7 instruction register
	x‘07E0’	Public Key entry 7 data register
	x‘07F0’	Public Key entry 7 result register

6.4.2.1 Public Key Command Register Format

The Command register indicates the PKP command flag and status.

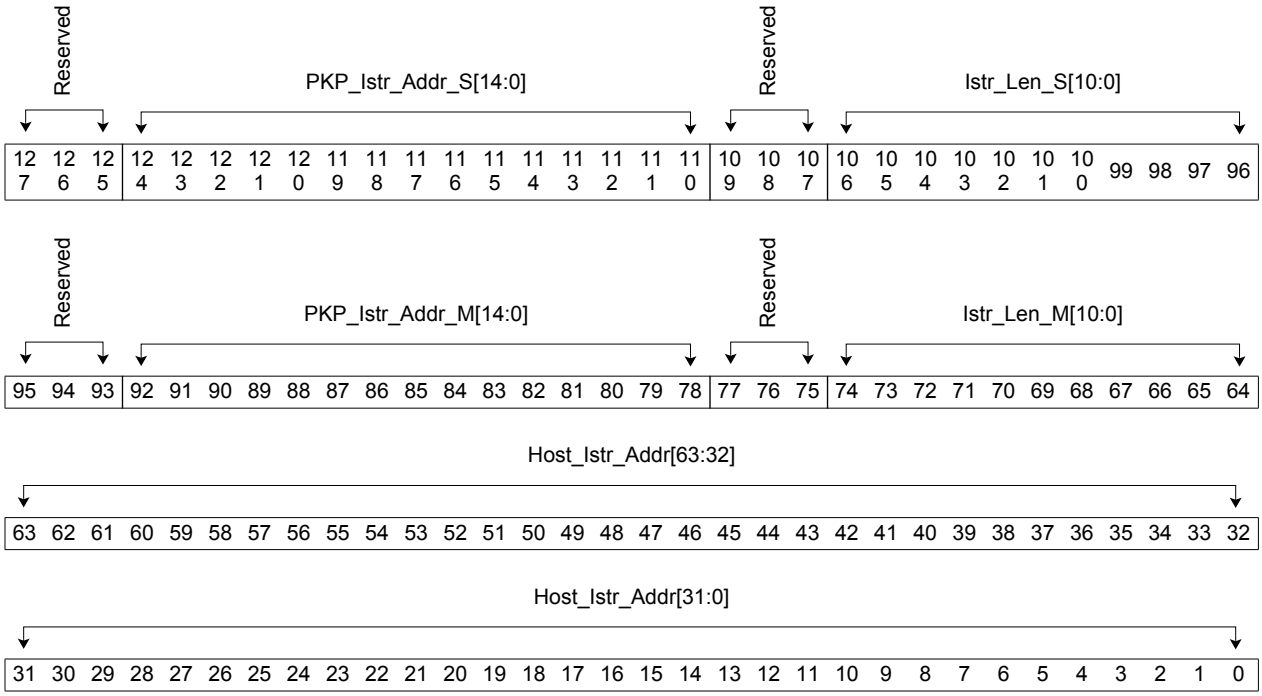


Field Name	Bits	Reset	Description
CMD_V	31	0	Command Valid. After this bit is set, the 820x will begin to process the command for this entry. Note: The host must set this bit to one *after* setting the Instruction Register, Data Register and Result Register. The 820x will clear this bit after the process is complete. The host must poll this register to determine whether a command has completed. 0 Command in this entry is not valid 1 Command in this entry is valid
PAIR	30	0	PK Pair Notification. After the 820x finishes a command, it will set this bit to inform the host which PK pair completed the current command. The host software need not configure this bit. 0 PK Pair 0 completed this command 1 PK Pair 1 completed this command
ECC_ERR	29	0	ECC error. 0 No ECC error in source buffer or destination buffer 1 ECC error in source buffer or destination buffer
ADDR_ERR	28	0	Host Access PKP Address out of range. This error is due to an error in software configuration. The 820x reports the error but takes no action.
Reserved	27	0	Reserved

Field Name	Bits	Reset	Description
IS_V	26	0	Instruction Register Valid. This bit is used to simplify the command operation process. Usually, the host loads the instruction programs in the first command, and then sets the LIRA in the following commands because the instruction programs have already been loaded in the PKP Linear Instruction Register by the first command. If this bit is set to one, the PKP Manager will wait until all PK pairs are idle, and then write the instruction to all PK pairs. This mechanism can simplify the instruction loading to all PK pairs, and maintain the same instruction field for all PK pairs. In other words, the 820x can balance the PK pair load, and the load balancing is transparent to the host software. 0 Instruction Register is not valid in this entry 1 820x requires load instruction in host memory into the PKP engine instruction register
DATA_V	25	0	Data Register Valid. The 820x will load data to one of the PK pairs when this bit is set to one. Typically, the host would set this bit to zero when writing to the PK pair instruction register for the first command that does not require any computation. 0 Data Register is not valid for this entry 1 Data Register is valid for this entry
Reserved	24	0	Reserved.
EF[1:0]	23:22	0	Instruction and Data Endian Format. 00 No swap 01 Byte swap in word 10 Word swap, no byte swap in word 11 Word swap and byte swap in word
LIRA_M[10:0]	21:11	0	PKP Master Linear Instruction Register (LIR) start address. The PKP Master will execute instruction code from this address.
LIRA_S[10:0]	10:0	0	PKP Slave Linear Instruction Register (LIR) start address. The PKP Slave will execute instruction code from this address.

6.4.2.2 Public Key Instruction Register Format

The Instruction register indicates the host source instruction buffer address, the PKP destination instruction register address, and the instruction size.

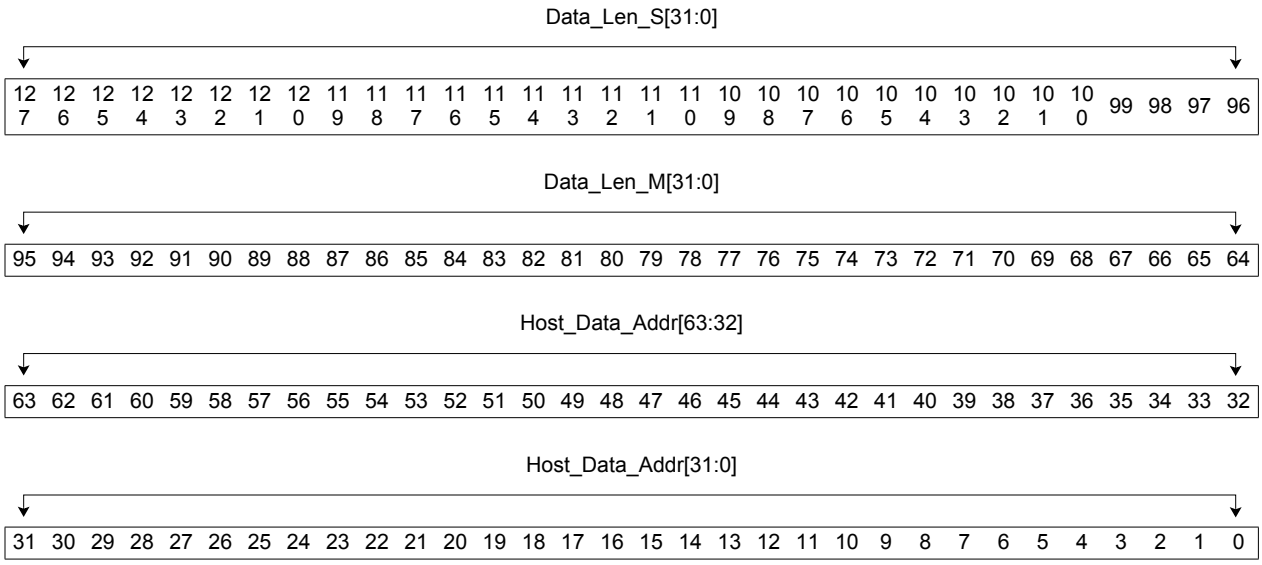


Field Name	Bits	Reset	Description
Reserved	127:125	0	Reserved.
PKP_Istr_Addr_S[14:0]	124:110	0	PKP Slave Linear Instruction Register (LIR) Start Address. This address must be 4-byte aligned because all PKP registers are double word aligned.
Reserved	109:107	0	Reserved.
Istr_Len_S[10:0]	106:96	0	PKP Slave Instruction Size. Istr_Len_M and Istr_len_S cannot both be equal to zero for any command when IS_V = 1 in the PK command register. The host instruction buffer size = Istr_len_M + Istr_Len_S. 0 No instruction for PKP Slave 1 - 2047 Number of double word instructions
Reserved	95:93	0	Reserved.
PKP_Istr_Addr_MS[14:0]	92:78	0	PKP Master Linear Instruction Register (LIR) Start Address. This address must be 4-byte aligned because all PKP registers are double word aligned.
Reserved	77:75	0	Reserved.

Field Name	Bits	Reset	Description
Istr_Len_M[10:0]	74:64	0	PKP Master Instruction Size. Istr_Len_M and Istr_len_S cannot both equal to zero for any command when IS_V = 1. The host instruction buffer size = Istr_len_M + Istr_Len_S. 0 No instruction for PKP Slave 1 - 2047 Number of double word instructions
Host_Istr_Addr[63:0]	63:0	0	Host Instruction Buffer Address. This address must be 8-byte aligned.

6.4.2.3 Public Key Data Register Format

The Public Key Data register indicates the source data size for PK Slave, the source data size for the PK Master, and the host source data buffer address. The host source buffer size is equal to the total source data size for PK Slave plus the total source data size for the PK Master. TSR, BER, FPR and MMR are PK core internal registers.

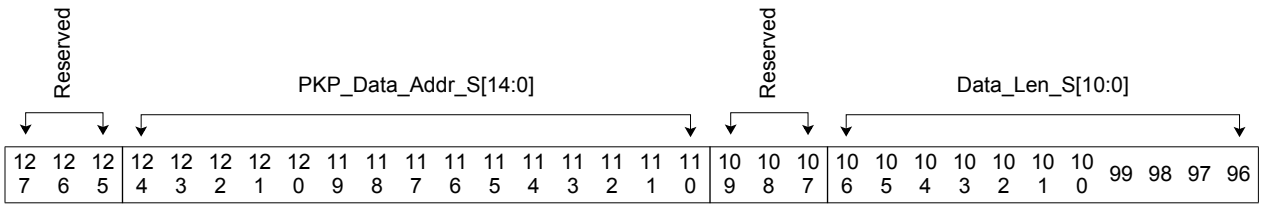


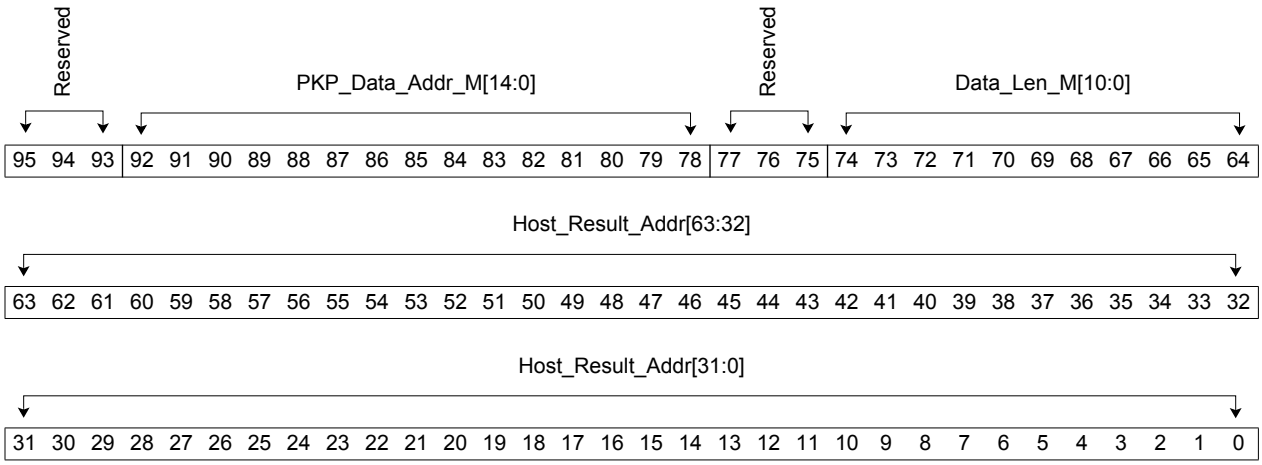
Field Name	Bits	Reset	Description
Data_Len_S[31:0]	127:96	0	PKP Slave Source Data Length. This field is a conglomeration of 4 subfields: TSR_CNT[1:0], FPR_LEN[9:0], MMR_LEN[9:0] and BER_LEN[9:0]. The total source data length is the sum of these 4 subfields. If TSR_CNT + FPR_LEN + MMR_LEN + BER_LEN = 0, there is no source data for the slave PKP. Bits [127:126] = TSR_CNT[1:0] PKP TSR data count 00 Zero 01 Same as BER_LEN 10 Same as MMR_LEN 11 Same as FPR_LEN Bits [125:116] = FPR_LEN[9:0] PKP FPR data count 0 Zero, no data for FPR register 1 - 10231 ~ 1023 double words (32 bits) Bits [115:106] = MMR_LEN[9:0] PKP MMR data count 0 Zero, no data for MMR register 1 - 10231 ~ 1023 double words (32 bits) Bits [105:96] = BER_LEN[9:0] PKP BER data count 0 Zero, no data for BER register 1 - 10231 ~ 1023 double words (32 bits)

Field Name	Bits	Reset	Description
Data_Len_M[31:0]	95:64	0	PKP Master Source Data Length. This field is a conglomeration of 4 subfields: TSR_CNT[1:0], FPR_LEN[9:0], MMR_LEN[9:0] and BER_LEN[9:0]. The total source data length is the sum of these 4 subfields. If TSR_CNT + FPR_LEN + MMR_LEN + BER_LEN = 0, there is no source data for the master PKP. Bits [95:94] = TSR_CNT[1:0] PKP TSR data count 00 Zero 01 Same as BER_LEN 10 Same as MMR_LEN 11 Same as FPR_LEN Bits [93:84] = FPR_LEN[9:0] PKP FPR data count 0 Zero, no data for FPR register 1 - 10231 ~ 1023 double words (32 bits) Bits [83:74] = MMR_LEN[9:0] PKP MMR data count 0 Zero, no data for MMR register 1 - 10231 ~ 1023 double words (32 bits) Bits [73:64] = BER_LEN[9:0] PKP BER data count 0 Zero, no data for BER register 1 - 10231 ~ 1023 double words (32 bits)
Host_Data_Addr[63:0]	63:0	0	Host Data Buffer Address This address must be 8-byte aligned.

6.4.2.4 Public Key Result Register

The Result register indicates the host destination result buffer address, the PKP source result register address, and the result size. The host result data buffer size is equal to Data_Len_M plus Data_Len_S.





Field Name	Bits	Reset	Description
Reserved	127:125	0	Reserved.
PKP_Data_Addr_S[14:0]	124:110	0	PKP Slave Result Data Register Start Address. This address must be 4-byte aligned.
Reserved	109:107	0	Reserved
Data_Len_S[10:0]	106:96	0	PKP Slave Result Data Size. This field must be 2 double word aligned. Data_Len_M and Data_Len_S cannot both equal to zero for any command when IS_V = 1. The host result data buffer size = Data_Len_M + Data_Len_S. 0 No Result Data for PKP Master 1 - 2047 Number of double word data
Reserved	95:93	0	Reserved.
PKP_Data_Addr_M[14:0]	92:78	0	PKP Master Result Data Register Start Address. This address must be 4-byte aligned.
Reserved	77:75	0	Reserved.
Data_Len_M[10:0]	74:64	0	PKP Master Result Data Size. This field must be 2 double word aligned. Data_Len_M and Data_Len_S cannot both equal to zero for any command when IS_V = 1. The host result data buffer size = Data_len_M + Data_Len_S. 0 No data for PKP Slave 1 - 2047 Number of double word data
Host_Result_Addr[63:0]	63:0	0	Host Result Data Buffer Address. This address must be 8-byte aligned.

6.4.3 Public Key Internal Registers

To access the PKP internal registers, please refer to the *EXP-E5200 Public Key Cryptography Microprocessor Datasheet* from Athena.

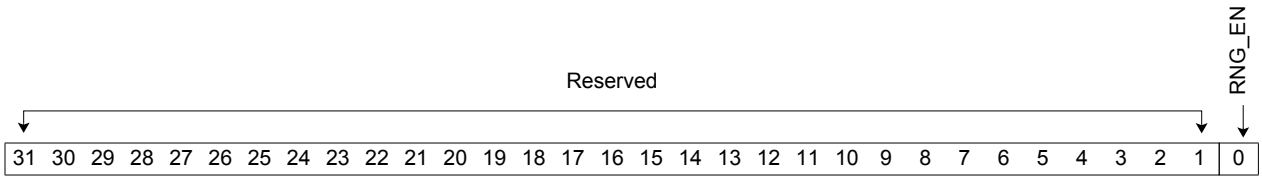
6.5 RNG Control Registers

6.5.1 RNG Enable Register

The RNG Enable register is used by the host to enable the 820x Random Number Generator.

Type: Read/Write

Offset x‘0800’



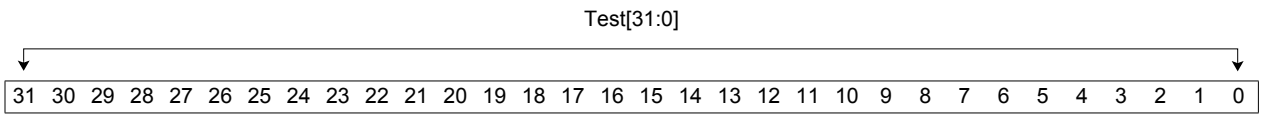
Field Name	Bits	Reset	Description
Reserved	31:1	0	Reserved.
RNG_EN	0	0	Random Number Generator Enable/Disable. 0 RNG and RNG clock disabled 1 RNG enabled

6.5.2 RNG Test Register

The RNG Test register allows the host software to perform read/write tests to the RNG address space. It serves no other function and causes no side effects. This register may be accessed at any time.

Type: Read/Write

Offset x‘0804’



Field Name	Bits	Reset	Description
Test[31:0]	31:0	0	Test data read/written from host into RNG memory space.

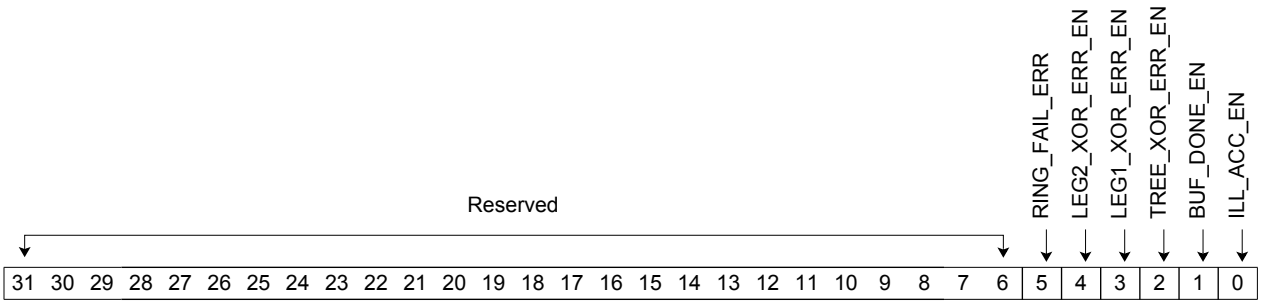
6.5.3 RNG Interrupt Enable Register

The RNG Interrupt Enable register allows the host software to enable/disable the 820x RNG interrupts. If a RNG Enable bit is set to one, the interrupt is enabled for that event. If the enable bit is cleared to zero, the interrupt will be disabled and will not interrupt the host, but the interrupt will appear in the RNG Interrupt Control / Status register as expected.

The LEG2_XOR_ERR, LEG1_XOR_ERR and TREE_XOR_ERR pertain to internal RNG logic. If any of these errors occur, please contact Exar customer support.

Type: Read/Write one to clear

Offset x'0808'



Field Name	Bits	Reset	Description
Reserved	31:6	0	Reserved.
RING_FAIL_EN	5	0	Ring Failure Interrupt Enable. Asserts an interrupt if any of the ring oscillators are not oscillating. 0 Disable ring failure interrupt 1 Enable ring failure interrupt
LEG2_XOR_ERR_EN	4	0	Leg 2 XOR Error Interrupt Enable. Asserts an interrupt if there was an error in the Leg 2 XOR RNG logic. 0 Disable Leg 2 XOR error interrupt 1 Enable Leg 2 XOR error interrupt
LEG1_XOR_ERR_EN	3	0	Leg 1 XOR Error Interrupt Enable. Asserts an interrupt if there was an error in the Leg 1 XOR RNG logic. 0 Disable Leg 1 XOR error interrupt 1 Enable Leg 1 XOR error interrupt
TREE_XOR_ERR_EN	2	0	Tree XOR Error Interrupt Enable. Asserts an interrupt if there was an error in the Tree XOR RNG logic. 0 Disable Tree XOR error interrupt 1 Enable Tree XOR error interrupt

Field Name	Bits	Reset	Description
BUF_DONE_EN	1	0	Buffer Done Flag Interrupt Enable. Asserts an interrupt if the Buffer_Done_Flag in the Interrupt Control / Status Register is set. 0 Disable Buffer Done Flag interrupt 1 Enable Buffer Done Flag interrupt
ILL_ACC_EN	0	0	Illegal Access Interrupt Enable. Asserts an interrupt if an illegal access to reserved memory space by one of the N8TBus interfaces was detected. 0 Disable Illegal Access interrupt 1 Enable Illegal Access interrupt

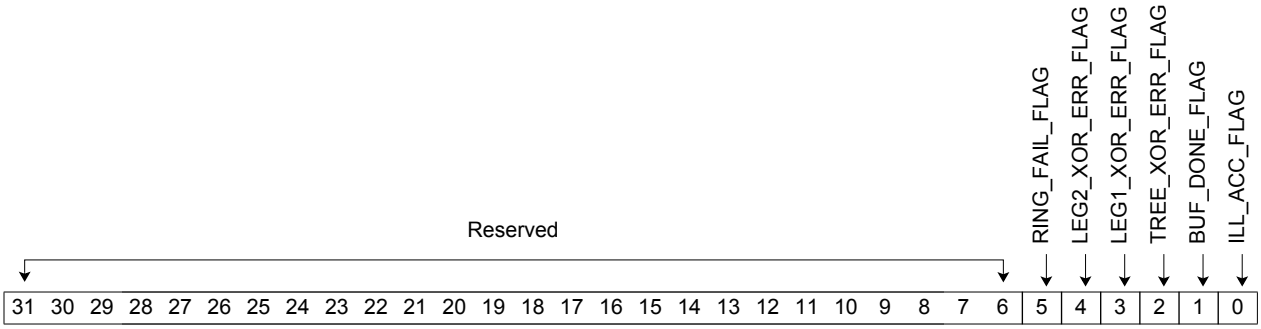
6.5.4 RNG Interrupt Control/Status Register

The RNG Interrupt Control/Status register allows the host software to read the 820x RNG interrupt status. The error bits in this registers will be set if that error event has occurred. If the error's corresponding enable bit is set in the RNG Interrupt Enable Register, an interrupt will be generated to the host.

The LEG2_XOR_ERR, LEG1_XOR_ERR and TREE_XOR_ERR pertain to internal RNG logic. If any of these errors occur, please contact Exar customer support.

Type: Read/Write one to clear

Offset x‘080C’



Field Name	Bits	Reset	Description
Reserved	31:6	0	Reserved.
RING_FAIL_FLAG	5	0	Ring Failure Flag. One or more ring oscillators failed to oscillate. 0 Ring failure not detected 1 Ring failure detected
LEG2_XOR_ERR_FLAG	4	0	Leg 2 XOR Error Flag. An error was detected in the RNG Leg 2 XOR logic. 0 Leg 2 XOR error not detected 1 Leg 2 XOR error detected
LEG1_XOR_ERR_FLAG	3	0	Leg 1 XOR Error Flag. An error was detected in the RNG LEG 1 XOR logic. 0 Leg 1 XOR error not detected 1 Leg 1 XOR error detected
TREE_XOR_ERR_FLAG	2	0	Tree XOR Error Interrupt Flag. An error was detected in the RNG Tree XOR logic. 0 Tree XOR error not detected 1 Tree XOR error detected

Field Name	Bits	Reset	Description
BUF_DONE_FLAG	1	0	Buffer Done Flag Buffer has been filled with random seeds, i.e., a 1-to-0 transition of the Buffer_Go/Busy bit was detected. 0 Buffer Done Flag condition not detected 1 Buffer Done Flag condition detected
ILL_ACC_FLAG	0	0	Illegal Access Flag An access to reserved memory space by one of the N8TBus interfaces was detected. 0 Illegal Access not detected 1 Illegal Access detected

6.5.5 RNG Buffer Control/Status Register

The RNG Buffer Control/Status register provides buffer control and status information. This register may be read at any time. However, writing to this register is only allowed when Buffer_Go/Busy is zero, otherwise the access will be treated as an access to reserved memory space.

The Sample Interval should generally be set such that at the currently specified bit rate (see RNG configuration register Bit Rate field in Section 6.5.7) at least 32 bits have been sampled between sample intervals. Setting the Sample Interval larger helps with increasing entropy of the random numbers, at the cost of generating random numbers at a smaller rate. For example,

Sample Interval >= 32 * (bit rate + 1)

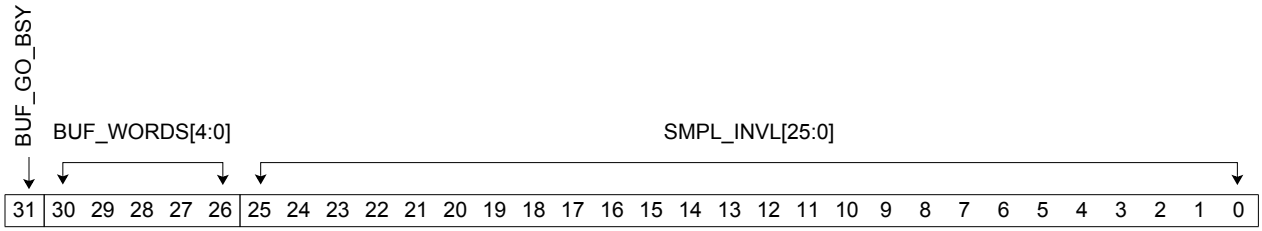
should always be true for normal operation. While

Sample Interval ~= 128 * (bit rate + 1)

is recommended for a reasonable entropy versus performance compromise.

Type: Read/Write Buffer_Words field is Read only

Offset x'0810'

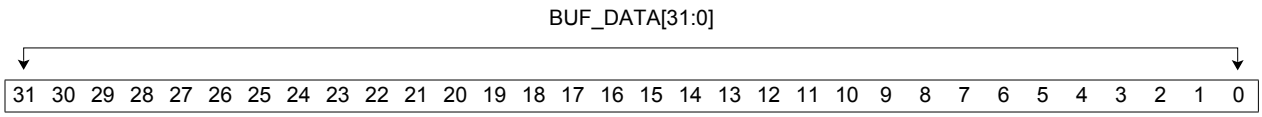


Field Name	Bits	Reset	Description
BUF_GO_BSY	31	0	Buffer Go/Busy. Writing a one to this bit causes the RNG buffer to be filled with 16 new random seeds, regardless of the number of unread seeds still in the Buffer (if any). This bit will remain set until the operation is complete, at which time it will return to zero. 0 RNG Buffer ready to be filled 1 Initiates filling RNG Buffer with new random seed
BUF_WORDS[4:0]	30:26	0	Buffer Words. The number of random seeds available in the buffer. This field will be decremented on each valid access of the Buffer Data Register (Section 6.5.6).
SMPL_INVL[25:0]	25:0	0	Sample Interval. Number of clocks to skip between capturing the 32 bit output of the Seed Generator into the RNG buffer.

6.5.6 RNG Buffer Data Register

The contents of the RNG Buffer may be read, one word at a time, through this register. This register must never be written; any writes will be treated as an access to Reserved memory space. This register may be read only when RNG Buffer Control Status register (Section 6.5.5) bits BUF_GO/BSY is zero *and* BUF_WORDS is nonzero; otherwise the read will be treated as an access to Reserved memory space.

Type:	Read only	When BUF_GO/BSY is zero AND BUF_WORDS is nonzero
Offset	x'0814'	



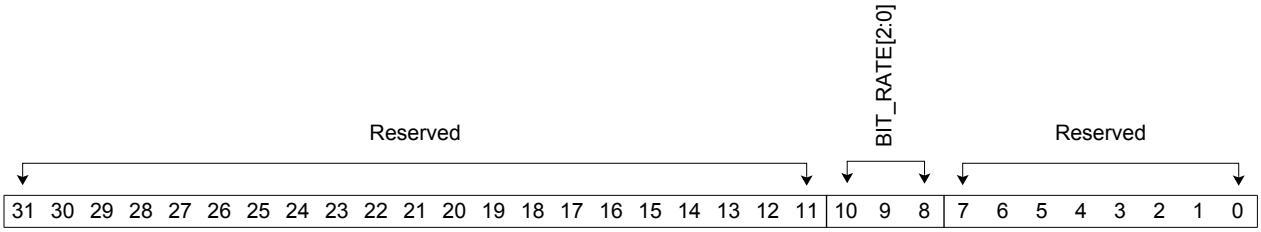
Field Name	Bits	Reset	Description
BUF_DATA[31:0]	31:0	0	Data from RNG seed buffer.

6.5.7 RNG Configuration Register

The RNG Configuration register is used to set the RNG bit rate. The bit rate is used to determine the RNG sample rate, as described in Section 6.5.4, "RNG Interrupt Control/ Status Register".

Type: Read/write

Offset x'0824'



Field Name	Bits	Reset	Description
RSVD	31:11	0	Reserved.
BIT_RATE[2:0]	10:8	000	Bit Rate. The number of 125MHz clock cycles between bit samples in the serial-to-parallel conversion / whitening LFSR. 0 125 Mbps (continuous) 1 62.5 Mbps 2 41.7 Mbps 3 31.25 Mbps 4 25 Mbps 5 20.8 Mbps 6 17.9 Mbps 7 15.63 Mbps
RSVD	7:0	0	Reserved.

6.6 GPIO Registers

The GPIO registers provide flexibility to configure the 820x GPIO pins. The operation sequence for configuring the GPIO registers is:

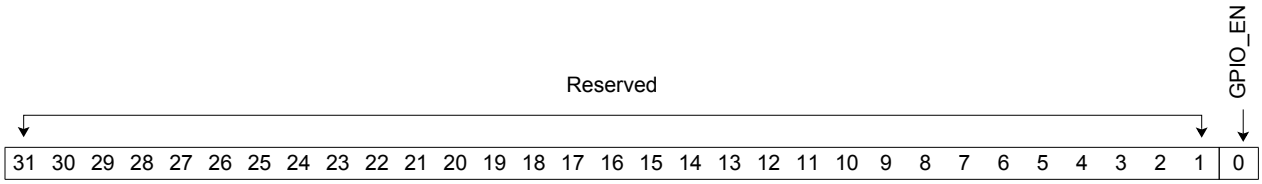
1. Set the GPIO port direction to either input or output.
2. Configure the interrupt mode if needed.
3. Read data from the GPIO EXT register if the GPIO are configured as input; write data to the GPIO SW DATA register if the GPIO are configured as output.

6.6.1 GPIO Enable Register

The GPIO Enable register is used by the host to enable the 820x General Purpose I/O registers.

Type: Read/Write

Offset x‘0850’ GPIO Enable Register



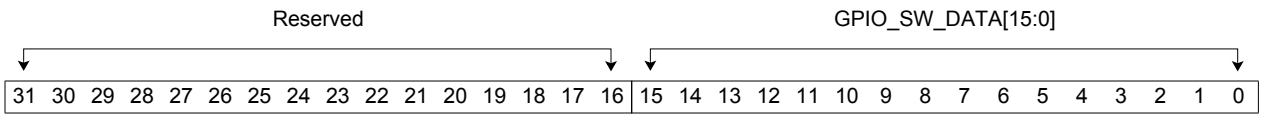
Field Name	Bits	Reset	Description
Reserved	31:1	0	Reserved.
GPIO_EN	0	0	General Purpose I/O Enable/Disable. 0 GPIO and GPIO clock disabled 1 GPIO enabled

6.6.2 GPIO Software Data Register

If the data direction bits in the GPIO Data Direction register are set to "output" mode and the control field is set to "Software" mode, then data written to the GPIO Software Data register are output to the 820x general purpose I/O pins. The value read back is equal to the last value written to this register.

Type: Read/Write

Offset x'0854'



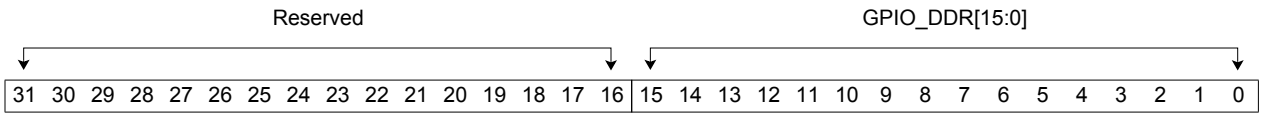
Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_SW_DATA [15:0]	15:0	GPIO_SW_ RESET	GPIO Software Data.

6.6.3 GPIO Data Direction Register

The GPIO Data Direction register is used to control the direction of the data bits in the GPIO Software register. Each bit may be independently set as either input or output. The default direction is input.

Type: Read/Write

Offset x'0858'



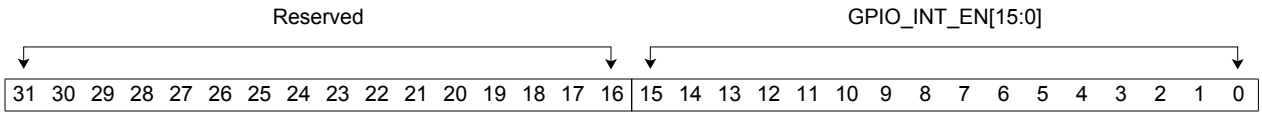
Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_DDR [15:0]	15:0	0	GPIO Data Direction. The direction for each bit can be set as: 0 Input 1 Output

6.6.4 GPIO Interrupt Enable Register

The GPIO Interrupt Enable register allows each GPIO bit to be independently configured as an interrupt. By default, using the GPIO ports as interrupts is disabled. If a one is written to a bit of this register, that GPIO bit will become an interrupt. Otherwise, it operates as a normal GPIO port. Interrupts will be disabled if the corresponding Data Direction register bit is set to Output or if its Mode is set to Hardware.

Type: Read/Write

Offset x‘0860’



Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_INT_EN [15:0]	15:0	0	GPIO Interrupt Enable. Each GPIO bit can be configured as: 0 Normal GPIO port (default) 1 Interrupt

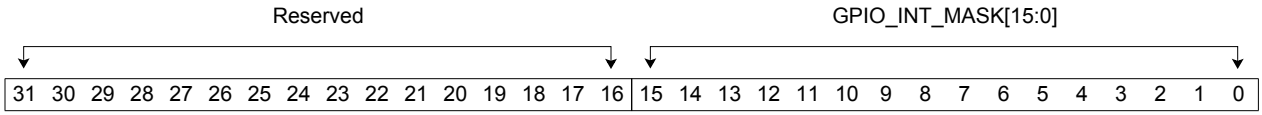
6.6.5 GPIO Interrupt Mask Register

The GPIO Interrupt Mask register is used to mask the GPIO interrupts. By default, all interrupts bits are unmasked.

If a one is written to a bit in this register, that bit will be masked from generating an interrupt. The host may be read this register to determine the masked/unmasked status of each bit.

Type: Read/Write

Offset x‘0864’



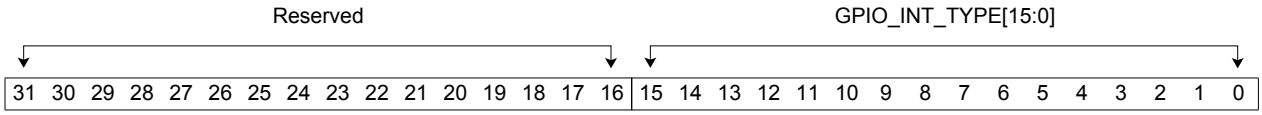
Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_INT_MASK [15:0]	15:0	0	GPIO Interrupt Mask. Each GPIO bit can be masked as: 0 Unmasked (default) 1 Masked

6.6.6 GPIO Interrupt Type Register

The GPIO Interrupt Type register controls whether the interrupt is level-sensitive or otherwise edge-sensitive. Each bit may be independently set.

Type: Read/Write

Offset x‘0868’



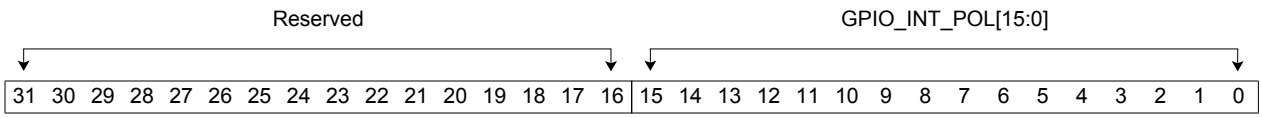
Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_INT_TYPE [15:0]	15:0	0	GPIO Interrupt Type. Each GPIO bit can be set as: 0 Level-sensitive (default) 1 Edge-sensitive

6.6.7 GPIO Interrupt Polarity Register

The GPIO Interrupt Polarity register controls the polarity of the edge or level sensitivity for each GPIO bit. If a zero is written to a bit of this register, the polarity for that GPIO interrupt bit will be configured as falling-edge or active-low; otherwise, if a one is written, the bit will be configured as rising-edge or active-high. Each bit may be independently set.

Type: Read/Write

Offset x‘086C’



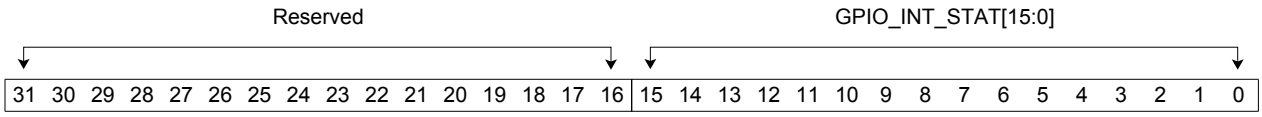
Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_INT_POL [15:0]	15:0	0	GPIO Interrupt Polarity. Each GPIO bit can be set as: 0 Active-low (default) 1 Active-high

6.6.8 GPIO Interrupt Status Register

The GPIO Interrupt Status register may be read by the host to determine the GPIO interrupt status. This register will reflect any masking that has been set using the GPIO Interrupt Mask register.

Type: Read/Write

Offset x‘0870’



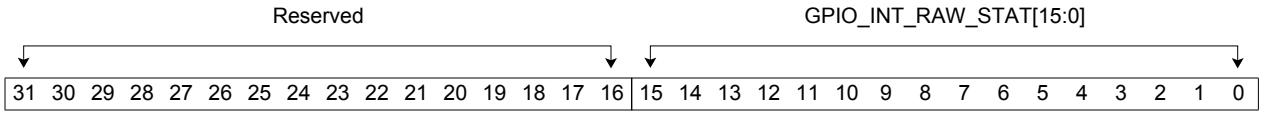
Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_INT_STAT [15:0]	15:0	0	GPIO Interrupt Status.

6.6.9 GPIO Interrupt Raw Status Register

The GPIO Interrupt Raw Status register may be read by the host to determine the GPIO interrupt status. This register does not reflect any masking that has been set using the GPIO Interrupt Mask register.

Type: Read/Write

Offset x'0874'



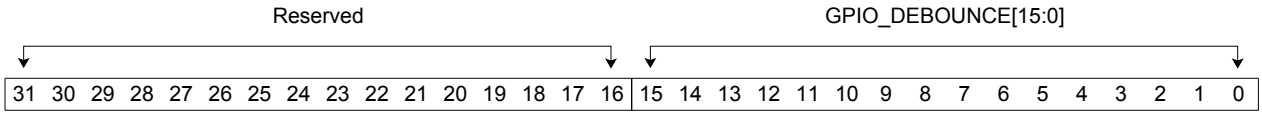
Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_INT_RAW STAT [15:0]	15:0	0	GPIO Interrupt Raw Status.

6.6.10 GPIO De-Bounce Register

The GPIO De-Bounce register controls whether an external signal that is the source of an interrupt needs to be debounced to remove any spurious glitches. Writing a one to a bit in this register enables the debouncing circuitry. Once enabled, a signal must be valid for two periods of an external clock before it is internally processed.

Type: Read/Write

Offset x'0878'



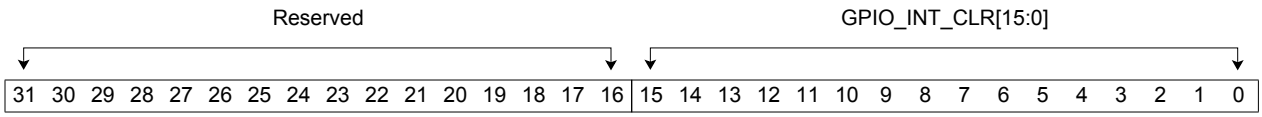
Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_DEBOUNCE [15:0]	15:0	0	GPIO De-bounce enable/disable. 0 Disable debounce (default) 1 Enable debounce

6.6.11 GPIO Interrupt Clear Register

The GPIO Interrupt Clear register controls the clearing of edge type interrupts. If a one is written into a bit of this register, the interrupt corresponding to that bit will be cleared.

Type: Read/Write

Offset x‘087C’



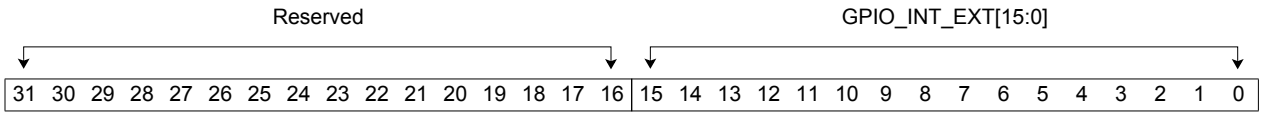
Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_INT_CLR [15:0]	15:0	0	GPIO Interrupt Clear. 0 Do not clear interrupt (default) 1 Clear interrupt

6.6.12 GPIO Interrupt Ext Register

If the data direction of the GPIO port is configured as “input,” this register can be used to read the values on the GPIO port. When the data direction of GPIO Port is set as “output,” reading this location reads the data register for that port.

Type: Read/Write

Offset x‘0880’



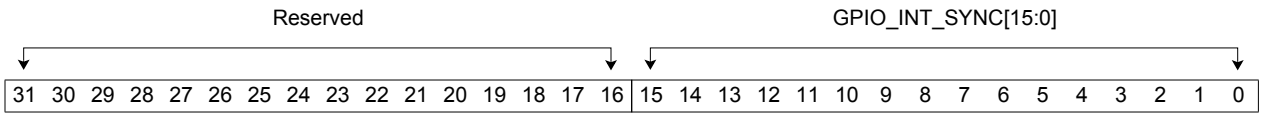
Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_INT_EXT [15:0]	15:0	0	External GPIO Interrupt. Data dir=input Read GPIO port Data dir=output Read data register for that GPIO port

6.6.13 GPIO Interrupt Sync Register

The GPIO Interrupt Sync register controls whether level sensitive interrupts are synchronized to pclk_intr.

Type: Read/Write

Offset x‘0884’



Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
GPIO_INT_SYNC [15:0]	15:0	0	GPIO Sync. 0: Do not synchronize to pclk_intr 1: Synchronize to pclk_intr

6.7 Serial Peripheral Interface Registers

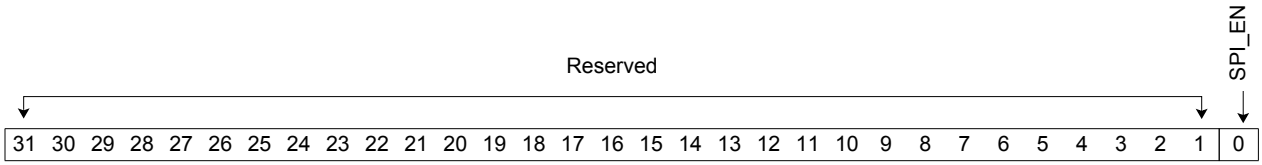
Refer to Section 5.12 for an overview of the SPI and for the sequence to access these registers.

6.7.1 SPI Enable Register

The SPI Enable register is used by the host to enable the clock to the 820x Serial Peripheral Interface module. Once SPI_EN is set, the host may access the programmable device regardless of the settings of the pins EXT_FLASH_CFG_EN and PCIE_PHY_CFG.

Type: Read/Write

Offset x'08C0'



Field Name	Bits	Reset	Description
Reserved	31:1	0	Reserved.
SPI_EN	0	0	Serial Peripheral Interface Enable/Disable. 0 SPI interface disabled 1 SPI interface enabled

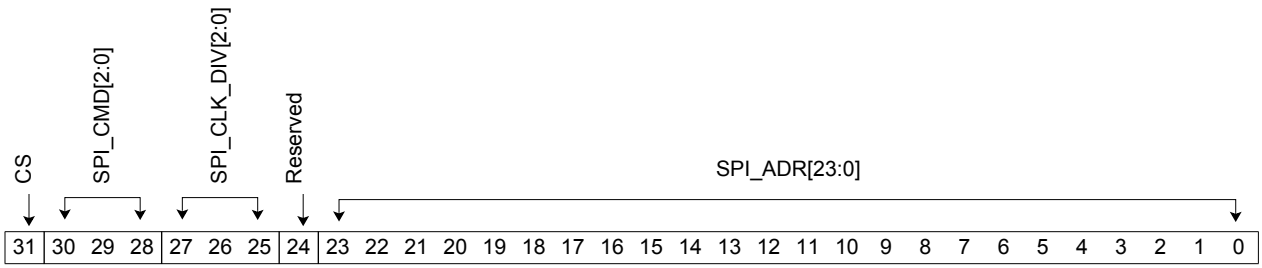
6.7.2 SPI Command Address Register

The SPI Enable register is used by the host to initiate the SPI command. Once the host writes to this register, the 820x will issue a SPI write/read operation to the external programmable device. For a write operation, the host should write the data to the "SPI Data Register" before writing to this register. For a read operation, the read data will be stored in "SPI Data Register" after the SPI_OP_Done bit in the "SPI Status Register" is set.

The programmable device command definitions in the "SPI Command Configuration 0 Register" and "SPI Command Configuration 1 Register" should have already been confirmed/set before writing to this register.

Type: Read/Write

Offset x'08C4'



Field Name	Bits	Reset	Description
CS	31	0	Chip Select. 0 Reserved 1 Current operation is for the programmable device
SPI_CMD[2:0]	30:28	001	Serial Peripheral Interface Command. This field is used to set the type of SPI command. If CS is also set, writing to this register will execute the command. 000 Write 4 bytes of data to the programmable device 001 Read 4 bytes of data from the programmable device 010 Read Identification (one byte manufacture ID and one byte Device ID) 100 Sector Erase (sector size depends on programmable device type) 101 User defined command (the "SPI User Defined Register" defines the detailed command. The 820x will simply write the command to the programmable device. All other values are reserved.

Field Name	Bits	Reset	Description
SPI_CLK_DIV[2:0]	27:25	011	SPI Clock Divisor. The SPI clock is equal to 125MHZ/SPI_CLK_DIV. 000 No division 001 Divide by 2 010 Divide by 4 011 Divide by 8 100 Divide by 16 101 Divide by 32 110 Divide by 64 111 Divide by 128
Reserved	24	0	Reserved.
SPI_ADR[23:0]	23:0	0	Programmable device Physical Address. Valid only when CS = 1. This is the sector address if the SPI operation is set to "sector erase".

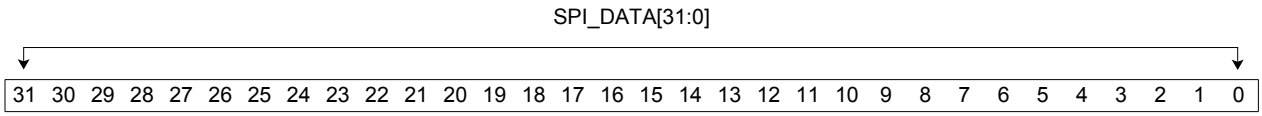
6.7.3 SPI Data Register

The SPI Data register can be used by the host to read and write data through the 820x Serial Peripheral Interface.

The MSB must be written to bit [31] and the LSB written to bit [0]. The MSB will be transmitted first on the SPI and LSB will be transmitted last.

Type: Read/Write

Offset x'08C8'



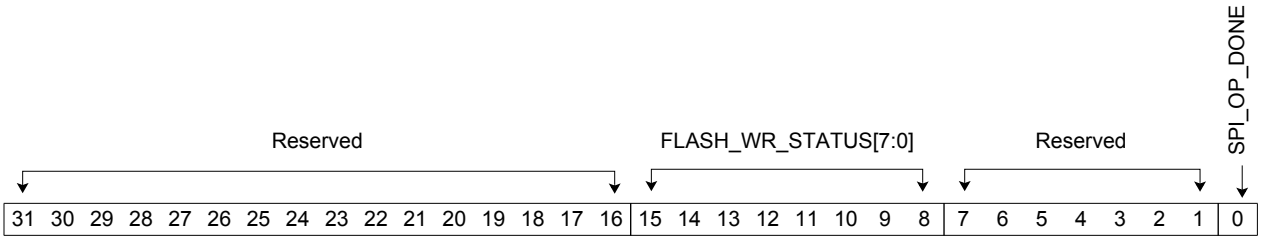
Field Name	Bits	Reset	Description
SPI_DATA[31:0]	31:0	0	Serial Peripheral Interface Data. For SPI write operations, the host should write the data to this register before writing to the "SPI Command Address Register". For SPI read operations, the 820x will write the data to this register when the operation is done.

6.7.4 SPI Status Register

The SPI Status register can be read by the host to determine when a SPI read or write operation on the Serial Peripheral Interface has completed. This register may also be used to read the programmable device's write status.

Type: Read only

Offset x'08CC'



Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
FLASH_WR_STATUS[7:0]	15:8	0	Programmable device Write Status. This field is only valid for 4 byte *write* operations. This field is a copy of the programmable device's status register. Please refer to the programmable device's specification for details regarding this field. The 820x will read this field to determine if the write operation has completed.
Reserved	7:1	0	Reserved.
SPI_OP_DONE	0	1	SPI Operation Done. This bit is read only from the host. When a SPI read/write operation is done, the 820x will assert this signal. When the host writes to the "SPI Command Address Register", the 820x will automatically clear this bit. 0 SPI read/write operation not done 1 SPI read/write operation done

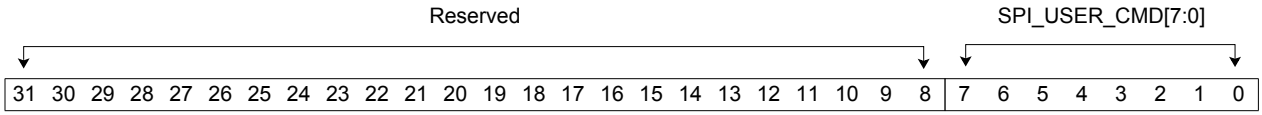
6.7.5 SPI User Defined Register

The SPI User Defined register can be used by the host to write custom commands to the device(s) attached to the Serial Peripheral Interface. Please refer to the specifications for the attached devices for a list of predefined commands.

For example, if a Spansion S25FL064A device is attached to the SPI interface, the user may issue a deep power down mode command (B9h). The host would write 'B9h' to this field, and then set SPI_CMD to '100' in the SPI Command Address Register. The 820x will then write the power down mode command to the programmable device.

Type: Read only

Offset x'08D0'



Field Name	Bits	Reset	Description
Reserved	31:8	0	Reserved.
SPI_USER_CMD [7:0]	7:0	0	SPI User Defined Command. This field can be used by the host to write any valid command defined by the device attached to the SPI interface.

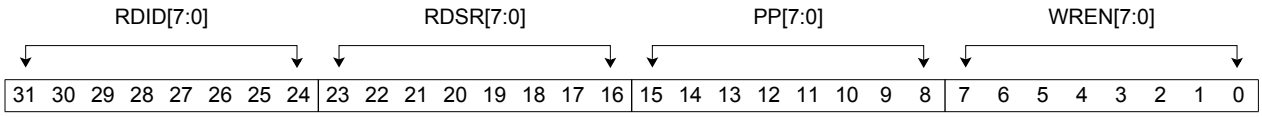
6.7.6 SPI Command Configuration 0 Register

The SPI Command Configuration Register 0 can be used by the host to store the command definitions that the 820x will use to access the external programmable device. The default values of this register apply to the recommended programmable devices (see the *820x Hardware Design User Guide*, UG-0211). The user should confirm that the default values are compatible with the programmable device and initialize this register accordingly.

If not using one of the recommended programmable devices, the host must set the command definitions for the programmable device in this register. This register simply holds the programmable device command definitions; the command will not be executed until writing to the "SPI Command Address Register".

Type: Read/Write

Offset x'08D4'



Field Name	Bits	Reset	Description
RDID[7:0]	31:24	0x90	Read Identification. The command definition to read the programmable device's vendor ID and device ID. The default value is the command definition for the recommended devices. Note: there is an erratum for this field. Refer to the 820x Errata document, ER-0015, for more information.
RDSR[7:0]	23:16	0x05	Read Status Command. The command definition to read the programmable device's status register. The default value is the command definition for the recommended devices.
PP[7:0]	15:8	0x02	Page Program command. The command definition for a page program command on the programmable device. The default value is the command definition for the recommended devices.
WREN[7:0]	7:0	0x06	Write Enable. The command definition to set the write enable for the programmable device. The default value is the command definition for the recommended devices.

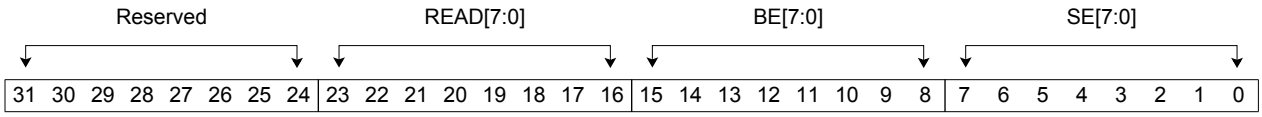
6.7.7 SPI Command Configuration 1 Register

The SPI Command Configuration Register 1 can be used by the host to store the command definitions that the 820x will use to access the external programmable device. The default values of this register apply to the recommended programmable devices. The user should confirm that the default values are compatible with the programmable device and initialize this register accordingly.

If not using one of the recommended programmable devices, the host must set the command definitions for the programmable device in this register. This register simply holds the programmable device command definitions; the command will not be executed until writing to the "SPI Command Address Register".

Type: Read/Write

Offset x'08D8'



Field Name	Bits	Reset	Description
Reserved	31:24	0	Reserved.
READ[7:0]	23:16	0x03	Read Command. The command definition to read from the programmable device. The default value is the command definition for the recommended devices.
BE[7:0]	15:8	0xC7	Bulk Erase. The command definition to erase the entire programmable device. The default value is the command definition for the recommended devices.
SE[7:0]	7:0	0x20	Sector Erase. The command definition to erase the a sector from the programmable device. The default value is the command definition for the recommended devices.

6.8 Temperature Sensor Controller Registers

The temperature sensor controller registers are used to access the internal temperature sensor registers.

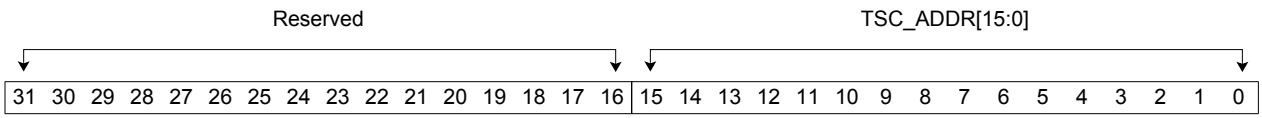
Refer to Section 5.13, "Temperature Sensor Controller" for a definition of the parameters "m1", "m2", and "a", and how they are used to calculate the 820x die temperature.

6.8.1 TSC Address Register

The TSC Address register is used to set the temperature sensor register address. The host should set the TSC address before a reading or writing to the 820x temperature sensor.

Type: Read/Write

Offset x'08F0'



Field Name	Bits	Reset	Description
Reserved	31:16	0	Reserved.
TSC_ADDR[15:0]	15:0	0	Temperature Sensor Address.

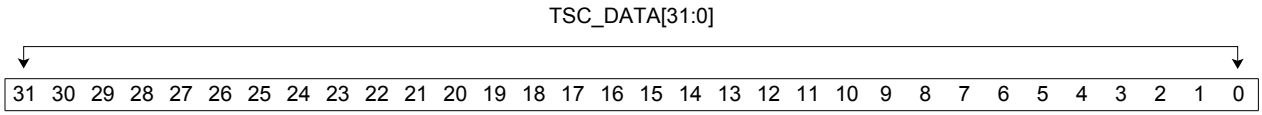
6.8.2 TSC Data Register

The TSC Data register is used to store the data read from the temperature sensor or store the data to be written to the temperature sensor. Note that the host should set the TSC address using the TSC Address register before issuing a read or write register command.

Refer to Section 5.13, "Temperature Sensor Controller" for a definition of the parameters "m1", "m2", and "a", and how they are used to calculate the 820x die temperature.

Type: Read/Write

Offset x'08F4'



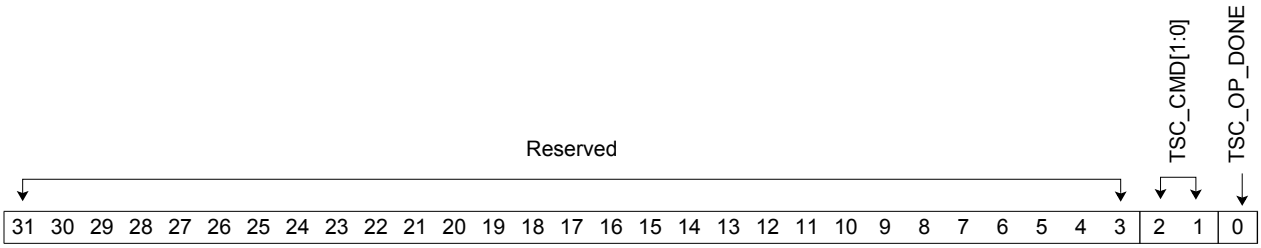
Field Name	Bits	Reset	Description
TSC_DATA[31:0]	31:0	X'FFFF_FFFF	Temperature Sensor Data. Read/Write TSC data For a TSC write operations, the host should write the data to this register before setting the TSC command register. When a TSC read operation is done (TSC_OP_DONE = 0), the 820x will write the temperature parameter "m1" to bits [9:0] and parameter "m2" to bits [25:16].

6.8.3 TSC Command Register

The TSC Command register can be used by the host to read or write data from/to the internal 820x PHY register pointed to by the TSC Address register.

Type: Read/Write TSC_OP_DONE is read only

Offset x‘08F8’



Field Name	Bits	Reset	Description
Reserved	31:3	0	Reserved.
TSC_CMD[1:0]	2:1	00	Temperature Sensor Controller Command. The host software should write to this field to trigger a temperature sensor register operation. 00 Reserved 01 Register Write 10 Register Read 11 Reserved For a “Register Write” command, the 820x will write TSC_Data[15:0] to the temperature sensor register with address TSC_ADDR. Note, the host must first write to the TSC_Addr and TSC_Data registers before issuing this command. For a “Register Read” command, the 820x will read from the temperature sensor register with address TSC_Addr, and write the value to TSC_Data[15:0]. Note, the host must write to TSC_Addr before issuing the command, and then read TSC_Data after the TSC_OP_Done bit is set by the 820x.
TSC_OP_DONE	0	1	Temperature Sensor Controller Operation Done. This bit is read only from the host. 0 820x will automatically clear this bit after the host writes to this register. 1 820x will assert this bit when a TSC read/write operation is done

7 PCIe Configuration Register Definition

This section describes the 820x PCIe configuration registers. The registers are mapped into 4K bytes of PCIe configuration space that can be accessed through the PCIe bus. The offset values in this section are given in hex.

The host should not read/write to/from reserved registers. If the host writes to a reserved register, the write will be ignored by the 820x. Likewise, if the host reads a reserved register, the return value will be all zeros.

Note that the value read from the PCIe registers may not be the 820x default value because the PCIe controller may write to the PCIe configuration registers before the application software can access these registers.

Table 7-1 lists the register types definitions used to describe the PCIe configuration registers in this chapter. Fields marked as 'Read Only' usually indicate the 820x capability and may not be altered by the host; fields marked as 'Read/Write' may be altered by the host (usually BIOS or OS) for special purposes.

Table 7-1. Register Type Definitions

Register Type	Description
RO	Read only. Register bits are read-only and cannot be altered by software.
ROS	Sticky - Read only. Registers are read-only and cannot be altered by software. Registers are not initialized or modified by hot reset.
RW	Read/Write. Register bits are read-write and may be either set or cleared by software to the desired state.
RW1C	Read-only status, Write-1-to-clear status. Register bits indicate status when read, a set bit indicating a status event may be cleared by writing a 1. Writing a 0 to RW1C bits has no effect.
RWS	Sticky - Read-Write. Registers are read-write and may be either set or cleared by software to the desired state. Bits are not initialized or modified by hot reset.
HWINIT	Hardware Initialized. Register bits are initialized by firmware or hardware mechanisms such as pin strapping or serial EEPROM. Bits are read-only after initialization and can only be reset with a power on reset.

Figure 7-1 illustrates the 820x PCIe Configuration registers.

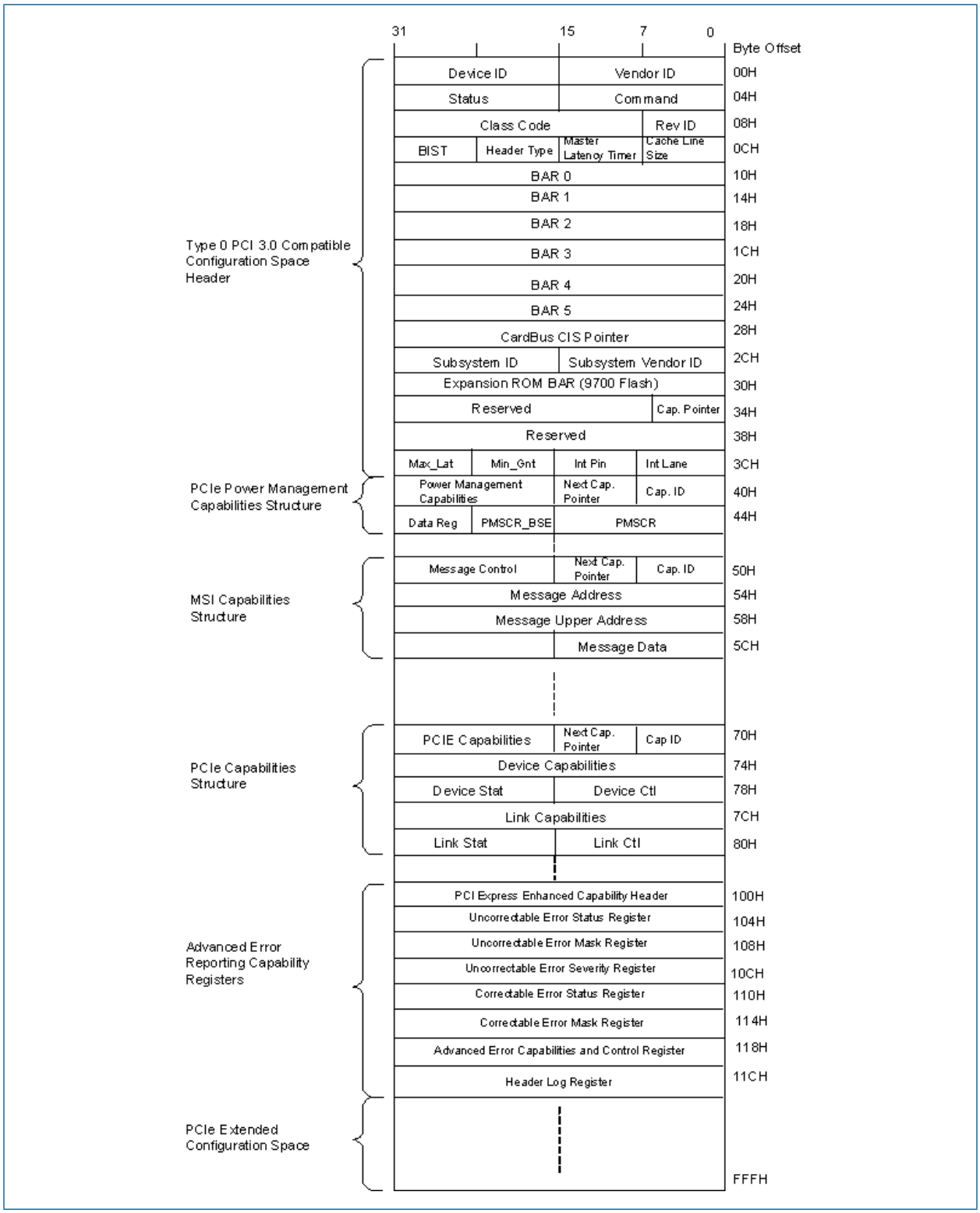


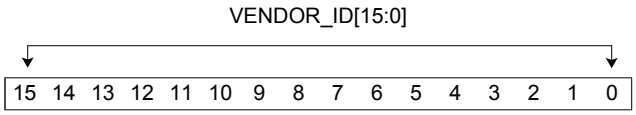
Figure 7-1. 820x PCI-Express Configuration Space

7.1 Type 0 PCIe Compatible Configuration Space

7.1.1 Vendor ID Register

The Vendor ID register identifies Exar as the manufacturer of the 820x device.

Offset x‘0000’



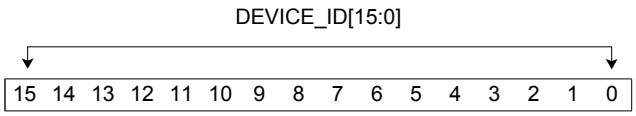
Field Name	Bits	Type	820x Value	Description
VENDOR_ID[15:0]	15:0	RO	0x13A3	This 16-bit register identifies Exar as the manufacturer of the 820x. The value hardwired in this read-only register is assigned by a central authority (the PCI SIG) that controls issuance of the numbers.

7.1.2 Device ID Register

The Device ID register identifies the 820x.

Type: Read only

Offset x‘0002’

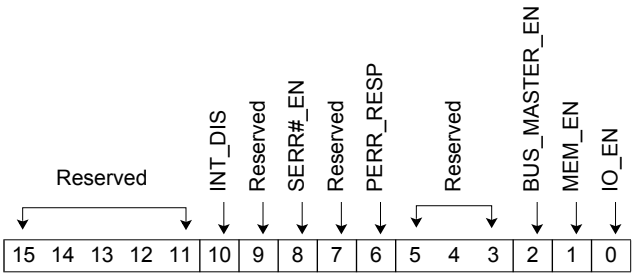


Field Name	Bits	Type	820x Value	Description
DEVICE_ID[15:0]	15:0	RO	preset value will match the chip version	This 16-bit value is assigned by Exar and identifies the 820x. In conjunction with the Vendor ID and Revision ID PCIe registers, and the Minor Revision 820x register, the Device ID can be used to locate a 820x-specific driver for the 820x. 0x0033 = 8201 device 0x0034 = 8202 device 0x0035 = 8203 device 0x0037 = 8204 device

7.1.3 Command Register

The Command register is used to enable or disable The I/O space, Memory space, Bus Master, Parity Error Response, System Errors, and Interrupts.

Offset x‘0004’



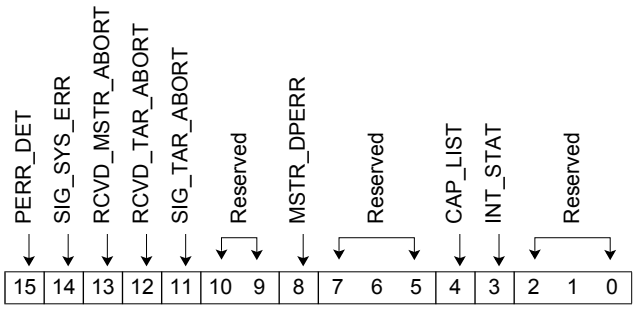
Field Name	Bits	Type	820x Value	Description
Reserved	15:11	RO	0	Reserved.
INT_DIS	10	RW	0	Interrupt Disable. Controls whether the 820x can generate INTx interrupt messages. 0 820x enabled to generate INTx interrupt messages. 1 820x's disabled to generate INTx interrupt messages is disabled. If the 820x had already transmitted an Assert_INTx emulation interrupt messages and this bit is then set, the 820x must transmit a corresponding Deassert_INTx message for each previously transmitted assert message Note that INTx emulation interrupt messages forwarded by Root and Switch Ports from devices downstream of the Root or Switch Port are not affected by this bit.
Reserved	9	RO	0	Reserved.
SERR#_EN	8	RW	0	System Error Enable. This active low bit enables or disables the reporting of errors detected by the 820x to the Root Complex. 0 820x may send detected errors to the Root Complex 1 820x may not send detected errors to the Root Complex
Reserved	7	RO	0	Reserved.

Field Name	Bits	Type	820x Value	Description
PERR_RESP	6	RW	0	Parity Error Response. This bit is used to enable or disable the Master Data Parity Error bit in the Status register. 0 Disable MSTR_DPERR 1 Enable MSTR_DPERR
Reserved	5:3	RO	0	Reserved.
BUS_MASTER_EN	2	RW	0	Bus Master Enable. 0 Disables the 820x from issuing memory or IO requests, and from generating MSI messages. 1 Enables the 820x to issue memory or IO requests, including MSI messages. Requests other than memory or IO requests are not controlled by this bit.
MEM_EN	1	RW	0	Memory Address Space Decoder Enable. 0 Memory decoder is disabled and Memory transactions targeting the 820x are not recognized. 1 Memory decoder is enabled and Memory transactions targeting the 820x are accepted.
IO_EN	0	RW	0	IO Address Space Decoder Enable. 0 IO decoder is disabled and IO transactions targeting the 820x are not recognized. 1 IO decoder is enabled and IO transactions targeting the 820x are accepted.

7.1.4 Status Register

The Status register is used to report errors and interrupts from the 820x to the host. The read only fields of this register are automatically updated by the 820x to reflect their internal status.

Offset x‘0006’



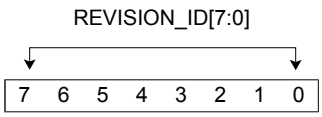
Field Name	Bits	Type	820x Value	Description
PERR_DET	15	RO	0	Parity Error Detected. Regardless of the state the Parity Error Enable bit in the 820x's Command register, this bit is set if the 820x receives a Poisoned TLP. 0 820x has not received a Poisoned TLP. 1 820x received a Poisoned TLP.
SIG_SYS_ERR	14	RO	0	Signaled System Error. 0 820x has not sent a fatal or non-fatal message. 1 The 820x sent an ERR_FATAL or ERR_NONFATAL message, and the SERR Enable bit in the Command register is set to one.
RCVD_MSTR_ABORT	13	RO	0	Received Master Abort. 0 820x has not sent a master abort message. 1 820x received a Completion with Unsupported Request Completion Status.
RCVD_TAR_ABORT	12	RO	0	Received Target Abort. 0 820x has not sent a received target abort message. 1 820x received a Completion with Completer Abort Completion Status.

Field Name	Bits	Type	820x Value	Description
SIG_TAR_ABORT	11	RO	0	Signaled Target Abort. 0 820x has not sent a signal target abort message. 1 The 820x, acting as a Completer, terminated a request by issuing Completer Abort Completion Status to the Requester.
Reserved	10:9	RO		Reserved
MSTR_DPERR	8	RO	0	Master Data Parity Error. The Master Data Parity Error bit is set by a 820x if the Parity Error Enable bit is set in the Command register and either of the following two conditions occurs: -the 820x receives a poisoned Completion. -the 820x poisons a write request. If the Parity Error Enable bit is cleared, the Master Data Parity Error status bit is never set. 0 No Master Data Parity Error occurred. 1 Master Data Parity Error occurred.
Reserved	7:5	RO	0	Reserved.
CAP_LIST	4	RO	1	Capabilities List. Indicates the presence of one or more extended capability register sets in the lower 48 dwords of the 820x's PCI-compatible configuration space. 0 Capabilities List not present. 1 Capabilities List present.
INT_STAT	3	RO	0	Interrupt Status. Indicates that the 820x has an interrupt request outstanding (that is, the 820x transmitted an interrupt message that is waiting to be serviced). Note that INTx emulation interrupts forwarded by Root and Switch Ports from devices downstream of the Root or Switch Port are not reflected in this bit. Note: this bit is only associated with INTx messages, and has no meaning if the device is using Message Signaled Interrupts. 0 820x has no interrupt request outstanding. 1 820x has an interrupt request outstanding.
Reserved	2:0	RO	N/A	Reserved

7.1.5 Revision ID Register

The Revision ID register identifies the major revision number assigned to each 820x device. Please refer to Section 6.1.6 for the minor revision number of the 820x device.

Offset x‘0008’

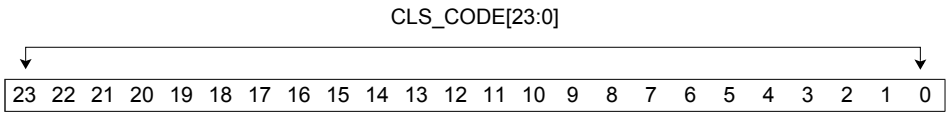


Field Name	Bits	Type	820x Value	Description
REVISION_ID[7:0]	7:0	RO	0x00	This 8-bit value is assigned by Exar to identify the revision number of the 820x. 0 = First major revision of the 820x

7.1.6 Class Code Register

The Class Code register defines the 820x encryption/decryption controller.

Offset x‘0009’



Field Name	Bits	Type	Default	Description
CLS_CODE[23:0]	23:0	RO	0x108000	Encryption/Decryption controller.

7.1.7 Cache Line Size Register

Offset x‘000C’

The Cache Line Size register is implemented as a read-write field for legacy compatibility purposes but has no impact on the 820x's functionality. The typical read value is 0x10.

7.1.8 Master Latency Timer Register

Offset x‘000D’

The Master Latency Timer register is implemented for legacy compatibility purposes but has no impact on the 820x's functionality. This register is hard-wired to 0x00.

7.1.9 Header Type Register

Offset x‘000E’

The Header Type register is a read-only optional register whose value is hard-wired to 0x00.

7.1.10 BIST Register

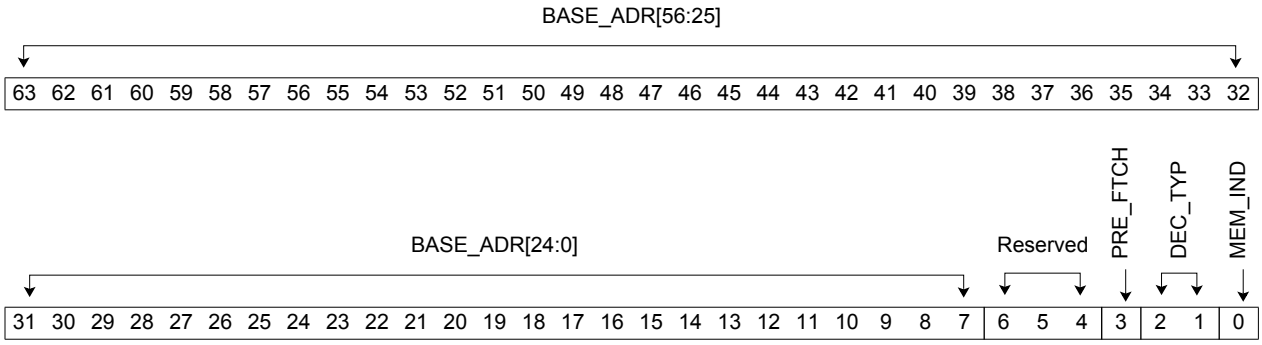
Offset x‘000F’

The BIST register is used for control and status of the BIST function. This register is hard-wired to 0x00.

7.1.11 Base Address Register 0, 1

The Base Address 0 register provides the 820x base address on a 4KB boundary and some memory configuration parameters. This address in memory space is where the standard 820x register set will reside and be accessed. For 32-bit systems, the base address is set using only BAR 0. For 64-bit systems, the base address is set using both BAR 0 and BAR 1.

Offset x‘0010’



Field Name	Bits	Type	820x Value	Description
BASE_ADR[56:0]	63:7	RW	0	Base Address. [63:32] Only used for 64-bit 4KB memory base address [31:7] Indicates 32-bit 4KB memory base address
Reserved	6:4	RO	0	Reserved.
PRE_FTCH	3	RO	0	Prefetchable memory. 0 Non-Prefetchable memory 1 Prefetchable memory
DEC_TYP	2:1	RW	00 for a 32-bit system; 10 for a 64-bit system	Decoder Type. 00 32 bit decoder; locate memory anywhere in lower 4GB; use only BAR0 01 Reserved 10 64-bit decoder; locate memory anywhere in 264 memory space; uses BAR0 and BAR1 11 Reserved
MEM_IND	0	RO	0	Memory Space Indicator. x0 Base Address Register0 is a memory address decoder x1 Base Address Register0 is an IO address decoder

7.1.12 Base Address Register 2-5

The Base Address Registers 2-5 are not used by the 820x, are read only, and will be read as all zeroes.

7.1.13 Cardbus CIS Pointer Register

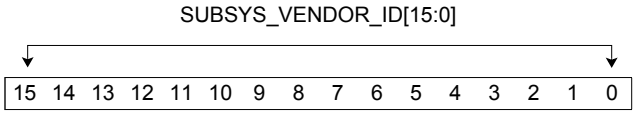
Offset x‘0028’

This optional read-only register is used by devices that contain a CardBus and PCIe interface. The 820x does not support Cardbus so this register is hard-wired to 0x00000000.

7.1.14 Sub-System Vendor ID Register

The Sub-System Vendor ID register identifies Exar as the manufacturer of the 820x device. This value is hard coded in the 820x device.

Offset x‘002C’

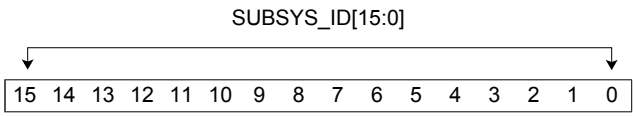


Field Name	Bits	Type	820x Value	Description
SUBSYS_VENDOR_ID[15:0]	15:0	RO	0x13A3	This 16-bit register identifies the manufacturer of the subsystem. The value hardwired in this read-only register is assigned by a central authority (the PCI SIG) that controls issuance of the numbers.

7.1.15 Sub-System ID Register

The Sub-System ID register identifies the 820x. This value is hard coded in the 820x device.

Offset x‘002E’

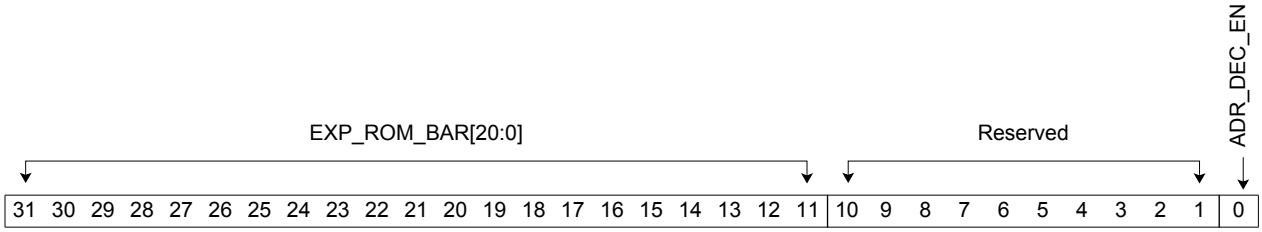


Field Name	Bits	Type	820x Value	Description
SUBSYS_ID[15:0]	15:0	RO	0x0036	This 16-bit value is assigned by the subsystem manufacturer and identifies the type of subsystem.

7.1.16 Expansion ROM Base Address Register

The ROM Base address register (ROM BAR) provides the 820x expansion ROM Base address and address decode enable. This register sets the address space in memory for the attached programmable device.

Offset x‘0030’ If the Expansion ROM is enabled, bits [23:11] are RO to reserve 16M space.



Field Name	Bits	Type	820x Value	Description
EXP_ROM_BAR [20:0]	31:11	RW		Expansion ROM Base Address. [31:24] The base address of the expansion ROM If the Expansion ROM is enabled, bits [23:11] are RO to reserve 16M space. Bits [23:0] should be R/W if using the Express DX SDK.
Reserved	10:1	RO	0	Reserved.
ADR_DEC_EN	0	RW	0	Address Decode Enable. 0 Disable expansion ROM 1 Enable expansion ROM This bit should be set low if using the Express DX SDK.

7.1.17 Capabilities Pointer Register

Offset x‘0034’

The Capabilities Pointer register is used to point to a linked list of capabilities implemented by the 820x. This read-only register value is hard-wired to 0x40, the Power Management Capabilities structure.

7.1.18 Interrupt Line Register

Offset x‘003C’

The Interrupt Line register communicates interrupt line routing information to device drivers and operating systems. Values in this register are programmed by system software and are system architecture specific. The typical read value is 0x0B.

7.1.19 Interrupt Pin Register

Offset x‘003D’

The Interrupt Pin register identifies the legacy interrupt Message(s) used by the 820x. This register is a read-only register whose value will be read as 0x01, indicating the 820x uses INTA.

7.1.20 Min_Gnt Register

Offset x‘003E’

This legacy register is a read-only register whose value is hard-wired to 0x00.

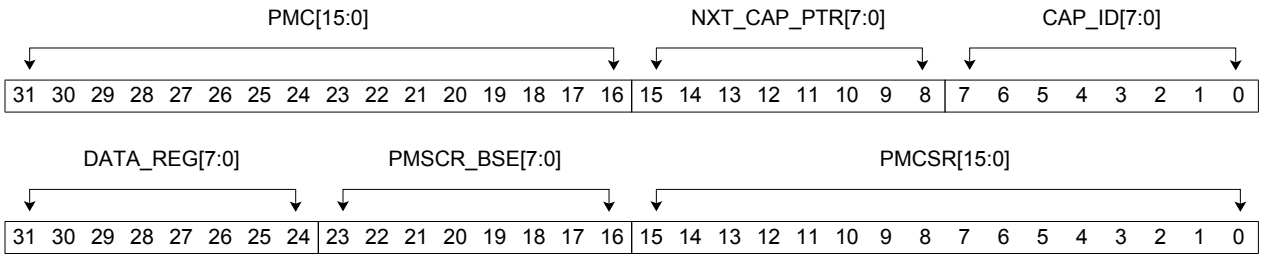
7.1.21 Max_Lat Register

Offset x‘003F’

This legacy register is a read-only register whose value is hard-wired to 0x00.

7.2 Power Management Capabilities Registers

The Power Management registers indicate the 820x power management capabilities.



7.2.1 Capability ID Register

The Capability Identifier register, when read by system software as 01h, indicates that the data structure currently being pointed to is the PCI Power Management data structure.

Offset x‘0040’

Field Name	Bits	Type	820x Value	Description
CAP_ID	7:0	RO	0x01	Power Management Capability ID. 01h = identifies the linked list item as being the PCI Power Management registers

7.2.2 Next Capabilities Pointer Register

The Next Capabilities register describes the location of the next item in the 820x’s capability list. The value given is an offset into the 820x’s PCI Configuration Space.

Offset x‘0041’

Field Name	Bits	Type	820x Value	Description
NXT_CAP_PTR	7:0	RO	0x50	Power Management Next Capabilities Pointer. 50h = PCIe MSI capabilities structure

7.2.3 Power Management Capabilities Register

The Power Management Capabilities register is a 16-bit read-only register which provides information on the capabilities of the function related to power management. The information in this register is generally static and known at design time.

Offset x‘0042’

Field Name	Bits	Type	820x Value	Description
PME Support	15:11	RO	0x0B	PME Support. Indicates the PM states supported by the 820x. A one in a bit indicates the 820x is capable of sending a Power Management Event (PME) message. A zero in a bit indicates PME notification is not supported in the respective PM state. Bit PM State 11 D0 (supported by 820x) 12 D1 (supported by 820x) 13 D2 (not supported by 820x) 14 D3hot (supported by 820x) 15 D3cold (not supported by 820x)
D2 Support	10	RO	0	D2 Support. 0 = D2 PM state not supported
D1 Support	9	RO	1	D1 Support. 1 = D1 PM state supported
Aux Current	8:6	RO	111	Aux Current. The Aux_Current field reports the 3.3Vaux current requirements for the 820x. The 820x reports 375mA max.
Device-Specific Initialization	5	RO	0	Device-Specific Initialization. A one in this bit indicates that immediately after entry into the D0 Uninitialized state, the 820x requires additional configuration above and beyond setup of its PCI configuration Header registers before the Class driver can use the 820x. Microsoft OSs do not use this bit. Rather, the determination and initialization is made by the Class driver.
Reserved	4	RO	0	Reserved.
PME Clock	3	RO	0	The 820x does not use PME Clock.
Version Field	2:0	RO	0x3	Version Field. This field indicates the version of the PCI Bus PM Interface spec that the 820x complies with.

7.2.4 Power Management Control/Status Register

The Power Management Control/Status (PMCSR) register is used to manage the PCI function's power management state as well as to enable/monitor Power Management Events (PMEs).

Offset x'0044'

Field Name	Bits	Type	820x Value	Description
PME_STAT	15	RW1C	0	PME Status. This bit is set to "0" because the 820x does not support PME# generation from D3cold.
DAT_SCALE	14:13	RO	0	Data Scale. This field is set to "00b" in the 820x because the PM Data register is not implemented.
DAT_SEL	12:9	RW	0000	Data Select. This field is set to "0000b" in the 820x because the PM Data register is not implemented.
PME_EN	8	RW	0	PME Enable. This bit is set to "0" because the 820x does not support PME# generation from D3cold.
Reserved	7:4	RO	0	Reserved.
NO_SFT_RST	3	RO	0	No Soft Reset. The 820x sets this bit to 0 to indicate it will perform an internal reset upon transitioning from D3hot to D0 via software control of the PowerState bits. Configuration Context is lost when performing the soft reset. Upon transition from the D3hot to the D0 state, full reinitialization sequence is needed to return the device to D0 Initialized.
Reserved	2	RO	0	Reserved.

Field Name	Bits	Type	820x Value	Description
PWR_STATE	1:0	RW	00	Power State. This 2-bit field is used both to determine the current power state of the 820x and to set the 820x into a new power state. The definition of the field values is given below. 00b - D0 01b - D1 10b - D2 11b - D3hot (not supported) If software attempts to write an unsupported state to this field, the write operation must complete normally on the bus; however, the data is discarded and no state change will occur.

7.2.5 PMCSR-BSE Register

Offset x‘0046’

The PMCSR PCI-to-PCI Bridge Support Extensions register does not apply to the 820x. When read, the register will return zeroes.

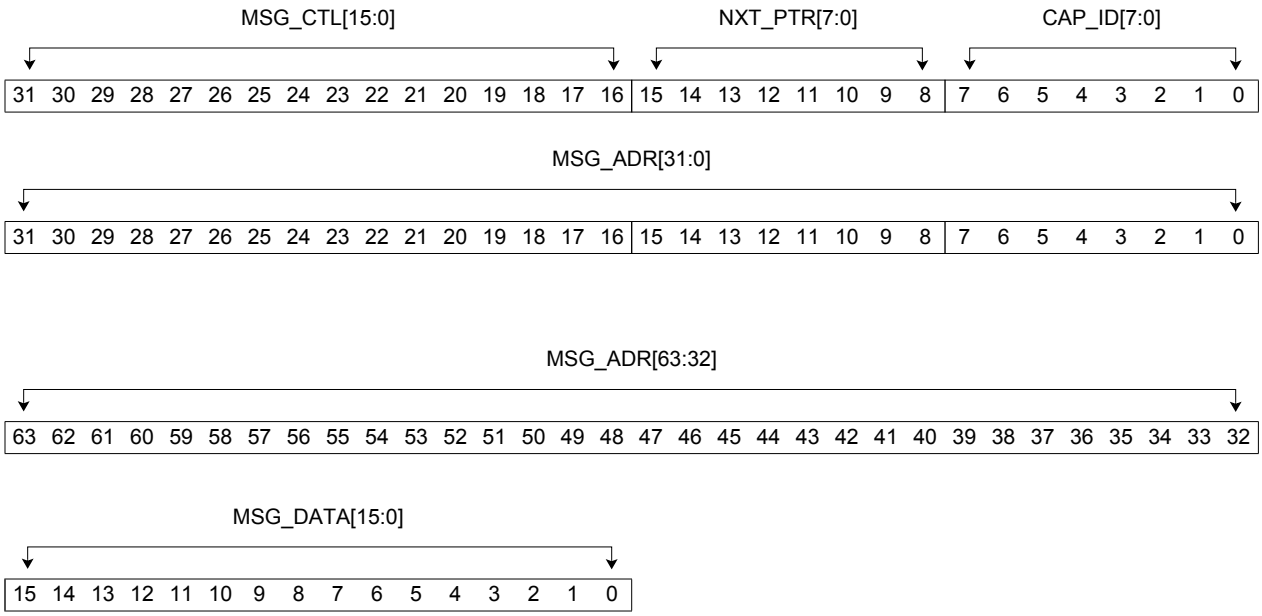
7.2.6 Data Register

Offset x‘0047’

The Data register is not used by the 820x but may be written to by system software. When read, the register will typically return all zeroes.

7.3 MSI Capability Registers

The MSI Capability registers indicate the 820x Message Signaled Interrupt capabilities. The 820x uses a 64-bit Message Address.



7.3.1 Capability ID Register

The Capability Identifier register, when read by system software as 05h, indicates that the 820x is MSI capable.

Offset x‘0050’

Field Name	Bits	Type	820x Value	Description
CAP_ID	7:0	RO	0x05	MSI Capability. 05h = MSI capable

7.3.2 Next Capabilities Pointer Register

The Next Capabilities register describes the location of the next item in the 820x's capability list. The value given is an offset into the 820x's PCI Configuration Space.

Offset x'0051'

Field Name	Bits	Type	820x Value	Description
NXT_PTR	7:0	RO	0x70	MSI Next Pointer. 70h = PCIe capabilities structure

7.3.3 Message Control Register

Offset x‘0053’

Field Name	Bits	Type	820x Value	Description
Reserved	15:9	RO	0	Reserved.
PER_VEC_MSK	8	RO	0	Per-vector masking. The 820x does not support MSI per-vector masking. 0 = Per-vector masking not supported
64_ADR_CAP	7	RO	1	64-bit Address. The 820x is capable of generating a 64-bit message address. 1 = 64-bit address capable
MULT_MESS_EN	6:4	RW	000	Multiple Message Enable. System software writes to this field to indicate the number of allocated vectors (equal to or less than the number of requested vectors). 000 = 1 vector allowed
MULT_MESS_CAP	3:1	RO	000	Multiple Message Capable. System software reads this field to determine the number of requested vectors. 000 = 1 vector requested
MSI Enable	0	RW	0	MSI Enable. System configuration software sets this bit to enable MSI. 0 = 820x is prohibited from using MSI to request service 1 = If the MSI-X Enable bit in the MSI-X Message Control register is 0, the 820x is permitted to use MSI to request service.

7.3.4 Message Address Register

Offset x‘0054 - x0058’

Field Name	Bits	Type	820x Value	Description
MSG_ADR	63:2	RW		MSI Message Address. If the Message Enable bit (bit 0 of the Message Control register) is set, the contents of this register specifies the DWORD aligned address (AD[63:02]) for the MSI memory write transaction. AD[1:0] are driven to zero during the address phase.
Reserved	1:0	RO	00	Reserved. Always returns 0 on read. Write operations have no effect.

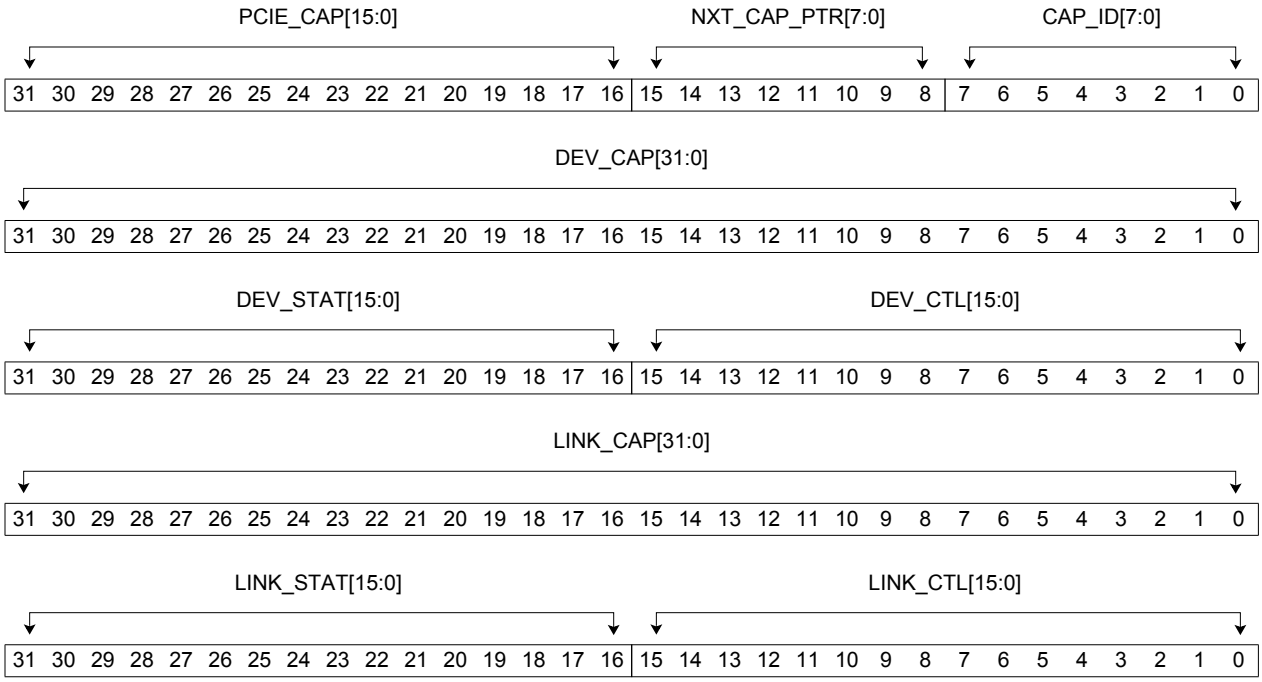
7.3.5 Message Data Register

Offset x‘005C’

Field Name	Bits	Type	820x Value	Description
MSG_DATA	15:0	RW		MSI Message Data. If the Message Enable bit (bit 0 of the Message Control register) is set, the message data is driven onto the lower word (MSG_ADR[15:0]) of the memory write transaction’s data phase. MSG_ADR[31:16] are driven to zero during the memory write transaction’s data phase. C/BE[3:0]# are asserted during the data phase of the memory write transaction.

7.4 PCI Express Capability Registers

The PCI Express Capability registers are required for PCI Express devices. The capability structure is a mechanism for enabling PCI software transparent features requiring support on legacy operating systems. In addition to identifying a PCI Express device, the PCI Express Capability structure is used to provide access to PCI Express specific Control/Status registers and related Power Management enhancements.



7.4.1 Capability ID Register

The Capability Identifier register, when read by system software as 10h, indicates that the 820x is PCIe capable.

Offset x‘0070’

Field Name	Bits	Type	820x Value	Description
CAP_ID[7:0]	7:0	RO	0x10	PCIe Capable. 10h = 820x is PCIe capable

7.4.2 Next Capabilities Pointer Register

The Next Capabilities register describes the location of the next item in the 820x's capability list. The value given is an offset into the 820x's PCI Configuration Space.

Offset x'0071'

Field Name	Bits	Type	820x Value	Description
NXT_PTR[7:0]	7:0	RO	0x00	Next Capability Pointer. 00h = Advanced Error structure

7.4.3 PCIe Capabilities Register

The PCI Express Capabilities (PCIE_CAP) register identifies the PCI Express device type and its associated capabilities.

Offset x'0072'

Field Name	Bits	Type	820x Value	Description
RSVD	15:14	RO	00	Reserved.
INT_MSG_NUM[4:0]	13:9	RO	0x0	Interrupt Message Number. This field is not used by the 820x.
SLOT_IMPL	8	HWINIT	0	Slot Implemented. This bit is not used by the 820x.
DEV_PORT_TYP[3:0]	7:4	RO	0000	Device/Port Type. Indicates the type of PCI Express logical device. 0000 = 820x is PCI Express Endpoint device
CAP_VER[3:0]	3:0	RO	01	Capability Version. Indicates the PCI Express capability structure version number.

7.4.4 Device Capabilities (DEV_CAP) Register

The Device Capabilities register identifies the PCI Express device specific capabilities.

Offset x‘0074’

Field Name	Bits	Type	820x Value	Description
Reserved	31:28	RO	0	Reserved.
Captured Slot Power Limit Scale[1:0]	27:26	RO		Captured Slot Power Limit Scale. This field specifies the scale used for the Slot Power Limit Value, which is set by the Set_Slot_Power_Limit Message for the 820x. Most systems will read 0x0, which means 1.0x.
Captured Slot Power Limit Value[7:0]	25:18	RO		Captured Slot Power Limit Value. This field is used in combination with the Slot Power Limit Scale value to specify the upper limit on the power supplied by the slot. The power limit (in Watts) is calculated by multiplying the value in this field by the value in the Slot Power Limit Scale field. This value is set by Set_Slot_Power_Limit Message for the 820x. Most systems will read 0x19, which represents 25W.
Reserved	17:15	RO	0x0	Reserved.
Power Indicator Present	14	RO	0	Power Indicator Present. When set to one, indicates a Power Indicator is implemented on the card or module. Valid for the following PCI Express device types: Express Endpoint device Legacy Express Endpoint device Switch upstream port Express-to-PCI/PCI-X bridge
Attention Indicator Present	13	RO	0	Attention Indicator Present. When set to one, indicates an Attention Indicator is implemented on the card or module. Valid for the following PCI Express device Types: Express Endpoint device Legacy Express Endpoint device Switch upstream port Express-to-PCI/PCI-X bridge

Field Name	Bits	Type	820x Value	Description
Attention Button Present	12	RO	0	Attention Button Present. When set to one, indicates an Attention Button is implemented on the card or module. Valid for the following PCI Express device types: Express Endpoint device Legacy Express Endpoint device Switch upstream port Express-to-PCI/PCI-X bridge
Endpoint L1s Acceptable Latency[2:0]	11:9	RO	0x3	Endpoint L1s Acceptable Latency. Acceptable latency that an Endpoint can withstand due to the transition from L1 state to the L0 state. This value is an indirect indication of the amount of the Endpoint's internal buffering. Power management software uses this value to compare against the L1 Exit Latencies reported by all components in the path between this Endpoint and its parent Root Port to determine whether ASPM L1 entry can be used with no loss of performance. The 820x requires an endpoint L1 acceptable latency of 8µs maximum.
Endpoint L0s Acceptable Latency[2:0]	8:6	RO	0x3	Endpoint L0s Acceptable Latency. Acceptable total latency that an Endpoint can withstand due to the transition from the L0s state to the L0 state. This value is an indirect indication of the amount of the Endpoint's internal buffering. Power management software uses this value to compare against the L0s exit latencies reported by all components in the path between this Endpoint and its parent Root Port to determine whether ASPM L0s entry can be used with no loss of performance. The 820x requires an endpoint L0 acceptable latency of 4µs maximum.
Extended Tag Field Supported	5	RO	0x0	Extended Tag Field Supported. Max supported size of the Tag field when this function acts as a Requester. 0 = 5-bit Tag field supported (max of 32 outstanding request per Requester). 1 = 8-bit Tag field supported (max of 256 outstanding request per Requester). If 8-bit Tags are supported and will be used, this feature is enabled by setting the Extended Tag Field Enable bit in the Device Control register to one.

Field Name	Bits	Type	820x Value	Description
Phantom Functions Supported[1:0]	4:3	RO	0x0	Phantom Functions Supported. When the device within which a function resides does not implement all eight functions, a non-zero value in this field indicates that this is so. Assuming all functions are not implemented and that the programmer has set the Phantom Function Enable bit in the Device Control register, a function may issue request packets using its own function number as well as one or more additional function numbers. This field indicates the number of msbs of the function number portion of Requester ID that are logically combined with the Tag identifier. 00 = The Phantom Function feature is not available within this device. 01 = The msb of the function number in the Requestor ID is used for Phantom Functions. The device designer may implement functions 0-3. When issuing request packets, Functions 0, 1, 2, and 3 may also use function numbers 4, 5, 6, and 7, respectively, in the packet's Requester ID. 10 = The two msbs of the function number in the Requestor ID are used for Phantom Functions. The device designer may implement functions 0 and 1. When issuing request packets, Function 0 may also use function numbers 2, 4, and 6 in the packet's Requester ID. Function 1 may also use function numbers 3, 5, and 7 in the packet's Requester ID. 11 =All three bits of the function number in the Requestor ID are used for Phantom Functions. The device designer must only implement Function 0 (and it may use any function number in the packet's Requester ID).
Max Payload Size Supported[2:0]	2:0	RO	0x2	Max Payload Size Supported. Defines the Max data payload size that the 820x supports for TLPs. The 820x supports 512 bytes max payload size.

7.4.5 Device Control (DEV_CTL) Register

The Device Control register controls PCI Express device specific parameters. The system software may read and write the fields of this register to control the operation of the 820x.

Offset x‘0078’

Field Name	Bits	Type	820x Value	Description
RSVD	15	RO	00	Reserved.
MAX_RD_REQ_SZ [2:0]	14:12	RW	010	Maximum Read Request Size. This field sets the maximum Read Request size for the Device as a Requester. The Device must not generate read requests with size exceeding the set value. Supported 820x read request sizes: 000 = 128 bytes 001 = 256 bytes 010 = 512 bytes
EN_NO_SNOOP	11	RW	1	Enable No Snoop. This bit is set to 1 in the 820x, indicating the 820x is permitted to set the No Snoop bit in the Requester Attributes of transactions it initiates that do not require hardware enforced cache coherency.
AUX_PWR_EN	10	RWS	0	Auxiliary (AUX) Power PM Enable. This bit is set to 0 by the 820x, disabling the 820x to draw AUX power independent of PME AUX power.
PHANT_EN	9	RW	0	Phantom Functions Enable. This bit is hardwired to 0 as the 820x device does not implement this capability.
EXT_TAG_EN	8	RW	0	Extended Tag Field Enable. This bit is set to 0 in the 820x to disable the 820x from using an 8-bit Tag field as a requester.

Field Name	Bits	Type	820x Value	Description
MAX_PAY_SZ[2:0]	7:5	RW	000	Maximum Payload Size. This field sets maximum TLP payload size for the device/function. As a Receiver, the device must handle TLPs as large as the set value; as Transmitter, the device must not generate TLPs exceeding the set value. Supported 820x payload sizes: 000 = 128 bytes max payload size 001 = 256 bytes max payload size 010 = 512 bytes max payload size
EN_RLX_ORD	4	RW	0	Enable Relaxed Ordering. This bit is hardwired to 0 as the 820x device never sets the Relaxed Ordering attribute in transactions it initiates as a requester.
NON_SPT_REQ_REP_EN	3	RW	0	Unsupported Request Reporting Enable. This bit, in conjunction with other bits, controls the signaling of Unsupported Requests by sending Error Messages.
FTL_ERR_REP_EN	2	RW	0	Fatal Error Reporting Enable. This bit, in conjunction with other bits, controls sending ERR_FATAL Messages.
NON_FTL_ERR_REP_EN	1	RW	0	Non-Fatal Error Reporting Enable. This bit, in conjunction with other bits, controls sending ERR_NONFATAL Messages.
COR_ERR_REP_EN	0	RW	0	Correctable Error Reporting Enable. This bit, in conjunction with other bits, controls sending ERR_COR Messages.

7.4.6 Device Status Register

The Device Status register provides information about PCI Express device specific parameters.

Offset x‘007A’

Field Name	Bits	Type	820x Value	Description
RSVD	15:6	RO	00	Reserved.
TRNS_PEND	5	RO	0	Transactions Pending. This bit when set indicates that the 820x has issued Non- Posted Requests which have not been completed. The 820x reports this bit cleared only when all outstanding Non-Posted Requests have completed or have been terminated by the Completion Timeout mechanism.
AUX_PWR_DET	4	RO	0	AUX Power Detected. This bit is set to 0 in the 820x, indicating the 820x does not use AUX power.
UNSUP_REQ_DET	3	RW1C	0	Unsupported Request Detected. This bit indicates that the 820x received an Unsupported Request. Errors are logged in this register regardless of whether error reporting is enabled or disabled in the Device Control register.
FTL_ERR_DET	2	RW1C	0	Fatal Error Detected. This bit indicates the status of the fatal errors detected. Errors are logged in this register regardless of whether error reporting is enabled or disabled in the Device Control register. Errors are logged in this register regardless of the settings of the correctable error mask register in the Advanced Error Handling registers.
NON_FTL_ERR_DET	1	RW1C	0	Non-Fatal Error Detected. This bit indicates the status of the non-fatal errors detected. Errors are logged in this register regardless of whether error reporting is enabled or disabled in the Device Control register. Errors are logged in this register regardless of the settings of the correctable error mask register in the Advanced Error Handling registers.

Field Name	Bits	Type	820x Value	Description
COR_ERR_DET	0	RW1C	0	Correctable Error Detected. This bit indicates the status of the correctable errors detected. Errors are logged in this register regardless of whether error reporting is enabled or disabled in the Device Control register. Errors are logged in this register regardless of the settings of the correctable error mask register in the Advanced Error Handling registers.

7.4.7 Link Capabilities Register

The Link Capabilities register identifies PCI Express Link specific capabilities.

Offset x‘007C’

Field Name	Bits	Type	820x Value	Description
Port Number[7:0]	31:24	RO	0	Port Number. The 820x has a single PCIe port; the port number is zero.
Reserved	23:18	RO	0	Reserved.
L1 Exit Latency[2:0]	17:15	RO	0x4	L1 Exit Latency. Indicates the L1 exit latency for the Link (i.e., the length of time this Port requires to complete a transition from L1 to L0). The 820x L1 Exit latency required is 8µs to 16µs.
L0 Exit Latency[2:0]	14:12	RO	0x3	L0 Exit Latency. Indicates the L0s exit latency for the Link (i.e., the length of time this Port requires to complete a transition from L0s to L0). The 820x L0 Exit latency required is 256ns to less than 512ns.
Active State Link PM Support[1:0]	11:10	RO	0x3	Active State Power Management (ASPM) Support. Indicates the level of ASPM supported on this Link. The 820x supports L0 and L1.
Max Link Width[5:0]	9:4	RO	0x4	Max Link Width. The 820x max link width is x4.
Max Link Speed[3:0]	3:0	RO	0x1	Max Link Speed. 0001 = 2.5 Gb/s

7.4.8 Link Control Register

The Link Control register controls PCI Express Link specific parameters.

Offset x‘0080’

Field Name	Bits	Type	820x Value	Description
Reserved	15:9	RO	0	Reserved.
EN_CLK_PWR_MAN	8	RW	0	Enable Clock Power Management. This bit is hardwired to 0 as the 820x device does not support Clock Power Management.
EXT_SYNC	7	RW	0	Extended Synch. This bit when set forces the transmission of additional ordered sets when exiting the L0s state and when in the Recovery state. This bit is not used by the 820x.
CLK_CFG	6	RW	~~1~~0	Common Clock Configuration. This bit when set indicates that the 820x and the component at the opposite end of this Link are operating with a distributed common reference clock.
RTN_LINK	5	RW	0	Retrain Link. This field is not applicable and is reserved for the 820x device.
LINK_DIS	4	RW	0	Link Disable. This field is not applicable and is reserved for the 820x device.
RCB	3	RW	0	Read Completion Boundary. This bit is hardwired to 0 as the 820x device does not support RCB.
Reserved	2	RO	0	Reserved.
ASPM_CTL[1:0]	1:0	RW	00	Active State Power Management (ASPM) Control. This field controls the level of ASPM supported on the given PCI Express Link. 820x supported values are: 00 = Disabled

7.4.9 Link Status Register

The Link Status register provides information about PCI Express Link specific parameters.

Offset x‘0082’

Field Name	Bits	Type	820x Value	Description
Reserved	15:14	RO	0	Reserved.
DAT_LINK_ACT	13	RO	0	Data Link Layer Link Active. This bit indicates the status of the Data Link Control and Management State Machine. It returns a 1b to indicate the DL_Active state, 0b otherwise. This bit must be implemented if the corresponding Data Link Layer Active Capability bit is implemented. Otherwise, this bit must be hardwired to 0b.
SLT_CLK_CFG	12	HWINIT	1	Slot Clock Configuration. This bit indicates that the 820x uses the same physical reference clock that the platform provides on the connector.
LINK_TRN	11	RO	0	Link Training. This field is not applicable to the 820x device and is hardwared to zero.
UNDEF	10	RO	0	Undefined. This legacy bit is no longer used.
NEG_LINK_WDTH [5:0]	9:4	RO		Negotiated Link Width. This field indicates the negotiated width of the given PCI Express Link. Defined encodings are: 000001 = x1 000010 = x2 000100 = x4 (typical value) 001000 = x8 001100 = x12 010000 = x16 100000 = x32 All other encodings are reserved. The value in this field is undefined when the Link is not up.
LINK_SP[3:0]	3:0	RO	0001	Link Speed. This field indicates the negotiated Link speed of the given PCI Express Link. Defined encodings are: 0001b 2.5 Gb/s PCI Express Link

7.5 Advanced Error Reporting Capability Registers

The Advanced Error Reporting Capability registers is an optional extended capability that may be implemented by PCI Express devices supporting advanced error control and reporting.

ENH_CAP_ HEAD[31:0]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

UNC_ERR_STAT[31:0]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

UNC_ERR_MASK[31:0]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

UNC_ERR_SEV[31:0]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

COR_ERR_STAT[31:0]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

COR_ERR_MASK[31:0]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ERR_CAP_CTL[31:0]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

HEAD_LOG[127:96]

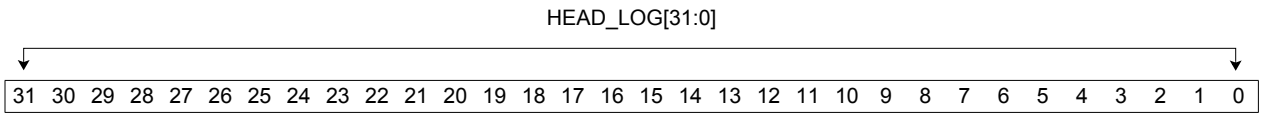
127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96

HEAD_LOG[95:64]

95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64

HEAD_LOG[63:32]

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32



7.5.1 Enhanced Capability Header Register

Offset x‘0100’

Field Name	Bits	Type	820x Value	Description
NXT_CAP_OFF[10:0]	31:20	RO	000h	Next Capability Offset. This field contains the offset to the next PCI Express capability structure or 000h if no other items exist in the linked list of capabilities. 000h = Terminating list of capabilities
CAP_VER[3:0]	19:16	RO	1h	Capability Version. This field is the version number of the capability structure present.
EXT_CAP_ID[15:0]	15:0	RO	0001h	PCI Express Extended Capability ID. The Extended Capability ID for the Advanced Error Reporting Capability is 0001h.

7.5.2 Uncorrectable Error Status Register

The Uncorrectable Error Status register indicates error detection status of individual errors on a 820x device. An individual error status bit that is set indicates that a particular error was detected; software may clear an error status by writing a one to the respective bit.

Offset x‘0104’

Field Name	Bits	Type	820x Value	Description
Reserved	31:21	RO	0	Reserved.
REQ_ERR_STAT	20	RW1CS	0	Unsupported Request Error Status.
ECRC_ERR_STAT	19	RW1CS	0	ECRC Error Status (Optional).
TLP_STAT	18	RW1CS	0	Malformed TLP Status.
RV_OVF_STAT	17	RW1CS	0	Receiver Overflow Status (Optional).
CMPL_STAT	16	RW1CS	0	Unexpected Completion Status.
CMPL_ABORT_STAT	15	RW1CS	0	Completer Abort Status (Optional).
CMPL_TMOUT_STAT	14	RW1CS	0	Completion Timeout Status.
FC_ERR_STAT	13	RW1CS	0	Flow Control Protocol Error Status (Optional).
POIS_TLP_STAT	12	RW1CS	0	Poisoned TLP Status.
Reserved	11:6	RO	0	Reserved.
DWN_ERR_STAT	5	RW1CS	0	Surprise Down Error Status (Optional).
DLNK_PROT_STAT	4	RW1CS	0	Data Link Protocol Error Status.
Reserved	3:1	RO	0	Reserved.
UNDEF	0	RO	0	Undefined.

7.5.3 Uncorrectable Error Mask Register

The Uncorrectable Error Mask register controls reporting of individual errors by the device to the PCI Express Root Complex via a PCI Express error Message. A masked error (respective bit set to 1b in the mask register) is not logged in the Header Log register, does not update the First Error Pointer, and is not reported to the PCI Express Root Complex by an individual device.

Offset x‘0108’

Field Name	Bits	Type	820x Value	Description
Reserved	31:21	RO	0	Reserved.
REQ_ERR_MSK	20	RWS	0	Unsupported Request Error Mask.
ECRC_ERR_MSK	19	RWS	0	ECRC Error Mask (Optional).
TLP_MSK	18	RWS	1	Malformed TLP Mask.
RV_OVF_MSK	17	RWS	1	Receiver Overflow Mask (Optional).
CMPL_MSK	16	RWS	0	Unexpected Completion Mask.
CMPL_ABORTMSK	15	RWS	0	Completer Abort Mask (Optional).
CMPL_TMOUT_MSK	14	RWS	0	Completion Timeout Mask.
FC_ERR_MSK	13	RWS	1	Flow Control Protocol Error Mask (Optional).
POIS_TLP_MSK	12	RWS	0	Poisoned TLP Mask.
Reserved	11:6	RO	0	Reserved.
DWN_ERR_MSK	5	RWS	1	Surprise Down Error Mask (Optional).
DLNK_PROT_MSK	4	RWS	0	Data Link Protocol Error Mask.
Reserved	3:1	RO	0	Reserved.
UNDEF	0	RO	0	Undefined.

7.5.4 Uncorrectable Error Severity Register

The Uncorrectable Error Severity register controls whether an individual error is reported as a Nonfatal or Fatal error. An error is reported as fatal when the corresponding error bit in the severity register is set. If the bit is cleared, the corresponding error is considered non-fatal.

Offset x‘010C’

Field Name	Bits	Type	820x Value	Description
Reserved	31:21	RO	0	Reserved.
REQ_ERR_SEV	20	RWS	0	Unsupported Request Error Severity.
ECRC_ERR_SEV	19	RWS	0	ECRC Error Severity (Optional).
TLP_SEV	18	RWS	1	Malformed TLP Severity.
RV_OVF_SEV	17	RWS	1	Receiver Overflow Severity (Optional).
CMPL_SEV	16	RWS	0	Unexpected Completion Severity.
CMPL_ABORT_SEV	15	RWS	0	Completer Abort Severity (Optional).
CMPL_TMOUT_SEV	14	RWS	0	Completion Timeout Severity.
FC_ERR_SEV	13	RWS	1	Flow Control Protocol Error Severity (Optional).
POIS_TLP_SEV	12	RWS	0	Poisoned TLP Severity.
Reserved	11:6	RO	0	Reserved.
DWN_ERR_SEV	5	RWS	1	Surprise Down Error Severity (Optional).
DLNK_PROT_SEV	4	RWS	1	Data Link Protocol Error Severity.
Reserved	3:1	RO	0	Reserved.
UNDEF	0	RO	0	Undefined.

7.5.5 Correctable Error Status Register

The Correctable Error Status register reports error status of individual correctable error sources on a 820x device. When an individual error status bit is set, it indicates that a particular error occurred; software may clear an error status by writing a 1 to the respective bit.

Offset x‘0110’

Field Name	Bits	Type	820x Value	Description
Reserved	31:14	RO	0	Reserved.
ADV_ERR_STAT	13	RW1CS	0	Advisory Non-Fatal Error Status.
REPLY_TMOUT_STAT	12	RW1CS	0	Replay Timer Timeout Status.
Reserved	11:9	RO	0	Reserved.
REPLAY_NUM_STAT	8	RW1CS	0	REPLAY_NUM Rollover Status.
BAD_DLLP_STAT	7	RW1CS	0	Bad DLLP Status.
BAD_TLP_STAT	6	RW1CS	0	Bad TLP Status.
Reserved	5:1	RO	0	Reserved.
RX_ERR_STAT	0	RW1CS	0	Receiver Error Status.

7.5.6 Correctable Error Mask Register

The Correctable Error Mask register controls reporting of individual correctable errors by the 820x to the PCI Express Root Complex via a PCI Express error Message. A masked error (respective bit set in mask register) is not reported to the PCI Express Root Complex by an individual device.

Offset x‘0114’

Field Name	Bits	Type	820x Value	Description
Reserved	31:14	RO	0	Reserved.
ADV_ERR_MSK	13	RWS	1	Advisory Non-Fatal Error Mask.
REPLY_TMOUT_MSK	12	RWS	0	Replay Timer Timeout Mask.
Reserved	11:9	RO	0	Reserved.
REPLAY_NUM_MSK	8	RWS	0	REPLAY_NUM Rollover Mask.
BAD_DLLP_MSK	7	RWS	0	Bad DLLP Mask.
BAD_TLP_MSK	6	RWS	0	Bad TLP Mask.
Reserved	5:1	RO	0	Reserved.
RX_ERR_MSK	0	RWS	0	Receiver Error Mask.

7.5.7 Advanced Error Capabilities and Control Register

Offset x‘0118’

Field Name	Bits	Type	820x Value	Description
Reserved	31:9	RO	0	Reserved.
ECRC_CHK_CAP	8	RWS	0	ECRC Check Enable. This bit when set enables ECRC checking.
ECRC_CHK_CAP	7	RO	1	ECRC Check Capable. This bit indicates that the 820x is capable of checking ECRC.
ECRC_EN	6	RWS	0	ECRC Generation Enable. This bit when set enables ECRC generation.
ECRC_CAP	5	RO	1	ECRC Generation Capable. This bit indicates that the 820x is capable of generating ECRC
FRST_ERR_PTR[4:0]	4:0	ROS	0	First Error Pointer. The First Error Pointer is a read-only register that identifies the bit position of the first error reported in the Uncorrectable Error Status register.

8 Signal Description

8.1 PCI Express Interface

The 820x provides a PCIe x4 interface for communicating with the host.

Table 8-1. PCIe Interface Signal Definition

Signal Name	I/O Type	Signal Type	Description
PCIE_TXP[3:0]	Output	Analog	PCIe compliant differential positive transmit outputs for lanes 0/1/2/3. No termination is required for these pins.
PCIE_TXN[3:0]	Output	Analog	PCIe compliant differential negative transmit outputs for lanes 0/1/2/3. No termination is required for these pins.
PCIE_RXP[3:0]	Input	Analog	Differential positive receive inputs for lanes 0/1/2/3. No termination is required for these pins.
PCIE_RXN[3:0]	Input	Analog	Differential negative receive inputs for lanes 0/1/2/3. No termination is required for these pins.
PCIE_REFCLK_P	Input	HCSL	PCIe compliant 100 MHz differential positive reference clock.
PCIE_REFCLK_N	Input	HCSL	PCIe compliant 100 MHz differential negative reference clock.
PCIE_REFRES	Input	Analog	PCIE reference resistor to calibrate the RX and TX termination. To calibrate the Rx and Tx termination resistors to 50Ω, connect a 191Ω 1% precision external resistor, ResRef, to the ResRef pin. The resistor should be 1% tolerant and can be in any form factor. The board trace connecting the ResRef pin to the external resistor constitutes parasitic resistance and should be minimized. Power dissipation in the component is < 20 mW peak during a calibration that lasts < 20 µs. At all other times, power dissipation is zero. Do not connect a capacitor to the ResRef pin.
PCIE_RXEQCTL[2:0]	Input	LVCMOS25_33	Receiver equalization control
PCIE_LOS_LVL[4:0]	Input	LVCMOS25_33	Loss of signal detection control for PCIE PHY

Table 8-1. PCIe Interface Signal Definition

Signal Name	I/O Type	Signal Type	Description
PCIE_TX_BOOST[3:0]	Input	LVCMOS25_33	Transmit boost control for PCIE PHY. The transmitter can be programmed to provide a pre-emphasis or de-emphasis boost. The boost is achieved by reducing the drive level of a non-transitioning bit with respect to a transitioning bit. These pins can be transitioned asynchronously. The amount of boost the transmitter supplies is programmable up to 5.75 dB in increments of ~0.37 dB. Boost = -20log(1 - (PCIE_TX_BOOST[3:0] + 0.5)/32) dB To produce 3.5 dB of boost as specified in the PCIe base specification, set tx_boost[3:0] to 4'b1011. This is one step higher than the calculated value because the calculated value is at the IP pads and does not take into account any package loss. A value of 4'b0000 produces 0 db of boost.
PCIE_TX_LVL[4:0]	Input	LVCMOS25_33	Transmit level control for PCIE PHY. Fine resolution setting of transmit signal level, common to all lanes connected to a single clock module. Setting the maximum Tx amplitude to greater than 1 V peak-to-peak differential results in overdrive (applying 1.2 V) to the thin-gate output transistors.
PERST_N	Input	LVCMOS25_33	PCIe reset. Assertion of this active low signal will reset the entire 820x device. This signal should be connected to PCIe slot reset. 0: Reset the 820x device 1: No PCIe reset
PCIE_PHY_CFG	Input	LVCMOS25_33	PCIe PHY configuration. The behavior of this bit depends on the value of EXT_FLASH_CFG_EN. Refer to Section 5.12 for a complete description.

8.2 Miscellaneous Interface

Table 8-2. Miscellaneous Interface Signal Definition

Signal Name	I/O Type	Signal Type	Description
POR_N	Input	LVCMOS25_33	Power on reset. Active low power on reset. When asserted, this bit will reset the entire 820x device including the PCIE core sticky registers. Due to an erratum, this signal should be connected to PERST_N (see the 820x Errata document, ER-0015). 0: Reset entire chip 1: No reset
PCIE_LINKUP	Output	LVCMOS25_33	PCIe Link Status This pad may be connected to a LED. 0: PCIe link down 1: PCIe link up
PLL_LOCK	Output	LVCMOS25_33	PLL Lock Status. Device internal PLL clock output frequency lock status. 0: One or both of the internal clock generation PLLs and/or the PCIe PHY PLL failed to successfully lock. 1: Both of the internal clock generation PLLs and the PCIe PHY PLL successfully locked.
BOARD_VERSION [2:0]	Input	LVCMOS25_33	Board Version Users may drive this signal to annotate the version of the board containing the 820x device.
PLL0_REF_CLK	Input	LVCMOS25_33	66.6667 MHz PLL0 reference clock
PLL1_REF_CLK	Input	LVCMOS25_33	66.6667 MHz PLL1 reference clock
VRET_PLL0	Input	Analog	VRET for PLL0 An external bypass capacitor should be connected between VDDA_PLL0 and this signal. See Figure 8-1 for an example of an external PLL connection circuit.
VRET_PLL1	Input	Analog	VRET for PLL1 An external bypass capacitor should be connected between VDDA_PLL0 and this signal. See Figure 8-1 for an example of an external PLL connection circuit.
EXT_FLASH_CFG_EN	Input	LVCMOS25_33	External Programmable Device Configuration Enable. The behavior of this bit depends on the value of PCIE_PHY_CFG. Refer to Section 5.12 for a complete description.

Figure 8-1 provides an example circuit for the PLL.

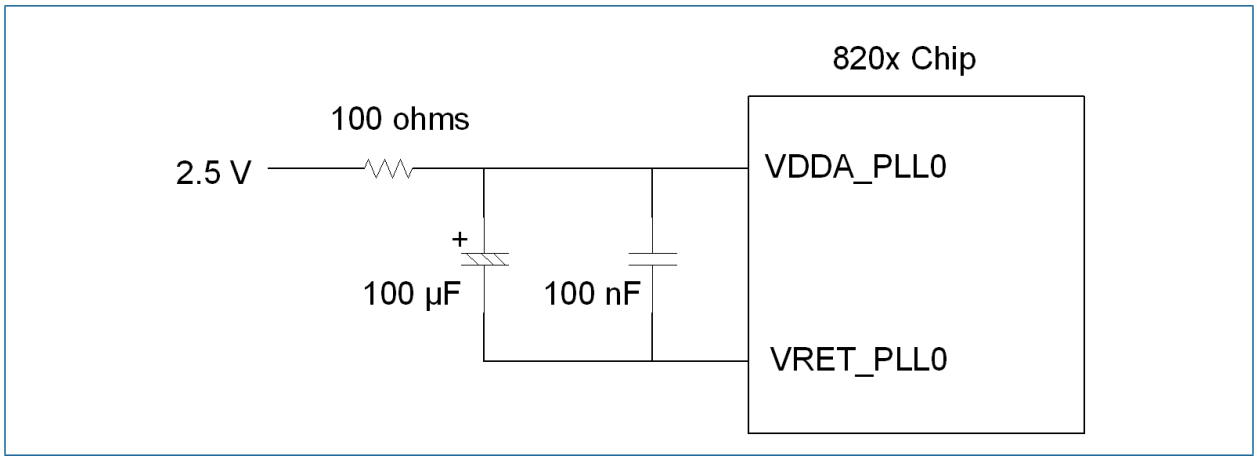


Figure 8-1. Example PLL Circuit

8.3 GPIO/Probe Interface

Table 8-3. Miscellaneous Interface Signal Definition

Signal Name	I/O Type	Signal Type	Description
GPIO[15:0]	Input/Output	LVCMOS25_33	General purpose IOs

8.4 SPI Interface

This section describes the Serial Peripheral Interface (SPI) interface. The Express DX SDK supports the SPI, however, a programmable device is not required or recommended for embedded applications.

Table 8-4. SPI Interface Signal Definition

Signal Name	I/O Type	Signal Type	Description
FLASH_SO	Output	LVCMOS25_33	Serial output. Output data from the 820x to the programmable device device.This output should not be connected if not using a programmable device.

Table 8-4. SPI Interface Signal Definition

Signal Name	I/O Type	Signal Type	Description
FLASH_SI	Input	LVCMOS25_33	Serial input. Programmable device input data driven from the output of the programmable device. This input pin should be tied high or low if not using a programmable device.
FLASH_SCK	Output	LVCMOS25_33	Clock output to the Programmable device. This output should be left unconnected if not using a programmable device.
FLASH_CS	Output	LVCMOS25_33	Chip select output signal from the 820x to the programmable device. This output should be left unconnected if not using a programmable device.

8.5 JTAG Interface

Table 8-5. JTAG Interface Signal Definition

Signal Name	I/O Type	Signal Type	Description
JTAG_TCK	Input	LVCMOS25_33	JTAG test clock
JTAG_TDI	Input	LVCMOS25_33	JTAG test data in, internally pulled up
JTAG_TDO	Output	LVCMOS25_33	JTAG test data out
JTAG_TMS	Input	LVCMOS25_33	JTAG test mode select, internally pulled up
JTAG_TRST_N	Input	LVCMOS25_33	JTAG reset, internally pulled up Connect to an external pull-down resistor if JTAG testing is desired. There is an errata for this pin that causes other circuitry in the chip to be reset by this input. JTAG_TRST_N can be tied to PERST_N if JTAG testing is not desired (see the *820x Errata document*, ER-0015).

8.6 Power and Ground Interface

Table 8-6. Power and Ground Interface Description (Sheet 1 of 2)

Pin Name	Description	Voltage Level
VDD10	Core digital power supply	1.0V
VDD25_33	This power domain can be connected with 2.5V or 3.3V. If 2.5V power domain is used, all IO signals in that domain will not be 3.3V tolerant	2.5V or 3.3V
VDD10A	Analog power supply for PCIE PHY	1.0V
VDD25A	Analog power supply for PCIE PHY	2.5V

Table 8-6. Power and Ground Interface Description (Sheet 2 of 2)

Pin Name	Description	Voltage Level
VDDA_PLL0	Analog power supply for on chip PLL0	2.5V
VDDA_PLL1	Analog power supply for on chip PLL1	2.5V
VSS	Digital ground	GND

9 Error Handling

9.1 Error Detection Methods

There are three methods the host software can use to detect if errors have occurred in the 820x:

1. Read the "ERR" bit of the Desc_result descriptor in the command structure. This bit will be set if any error has occurred.

2. Read the result status in the result ring.

3. Poll the 820x error status register.

If the host is only interested in whether an error has occurred, the best method would be to read the "ERR" bit. If the host is interested in detailed error information for the current command, the best method would be to read the result status in the result ring. If the host is interested in the type of error that occurred without regard to the command that created the error, the best method would be to read the 820x error status and Channel Manager error status registers, where any errors are recorded until it is cleared by host.

9.2 Error Categories

There are two error categories for the 820x:

Category 1 errors: Channel Manager Detected Error

> All errors in this category will not block the 820x's operation even if the errors occur while processing a command. These errors will corrupt the command result. Refer to "Channel Manager 0-1 Error Status" register for details.

Category 2 errors: PCIe Core Reported Error

> Error in this category would include completion timeout, completion abort, parity error and ECRC error. All errors in this category will block the 820x's operation because the 820x cannot access the completion data.

9.3 Error Handling Mechanism

The 820x error handing mechanism adapts well to network and storage applications with minimum system overhead.

Self recovery from Category 1 errors:

> The 820x will recover from these errors because these errors will not block the 820x's operation. The 820x will continue processing commands and set the error status bits in the command structure and result ring for the command that errored. The host software can then reschedule the failed command according to the result information.

Host reset 820x for Category 2 errors:

The 820x cannot recover from these errors because the errors will not allow the command to complete.

1. If this category of error is detected by the 820x, the 820x will terminate all commands in both 820x's channels, and then wait for the host's soft reset.

2. The host software may use one of the three methods described above to detect the 820x error. Once the error has been detected, the host should poll the CM_BUSY bit of the Status register before resetting the 820x.

3. When the CM_BUSY bit is cleared to zero by the 820x, the host software writes to the Soft Reset register to reset the entire 820x chip except the PCIe core.

4. THe host software then reschedules all terminated commands.

The 820x command pointer ring read pointer and result ring write pointer will be writable during the error handling phase to facilitate the software error recovery flow.

10 DC Specifications

10.1 Absolute Maximum Ratings

Table 10-1. Absolute maximum ratings

Symbol	Description	Min	Max	Units
VDD_25_33	This power domain may be connected to either a 2.5V or 3.3V supply. If a 2.5V power domain is used, all IO signals in that domain are not 3.3V tolerant.	VSS-0.5	4.0	V
VDD_10	DC supply 1.0V Core Voltage	VSS-0.2	1.2	V
VDD_25A	2.5V analog power supply for PCIe PHY	VSS-0.4	3.0	V
VDD_10A	1.0V analog power supply for PCIe PHY	VSS-0.2	1.2	V
VDDA_PLL0	2.5V analog power supply for on chip PLL0	VSS-0.4	3.0	V
VDDA_PLL1	2.5V analog power supply for on chip PLL1	VSS-0.4	3.0	V
T_{STG}	Storage Temperature	-40	130	°C

Caution

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions may affect device reliability.

10.2 Recommended Operating Conditions

Table 10-2. Recommended operating conditions

Symbol	Description	Min	Max	Units
Tj	Junction Temperature	-40	125	°C
Ta	Ambient Operating Temperature for 8201I	-40	85	°C

Notes:

1. For normal device operation, adhere to the limits in this table. Sustained operations of a device at conditions exceeding these values, even if they are within the absolute maximum rating limits, may result in permanent device damage or impaired device reliability. Device functionality to stated DC and AC limits is not guaranteed if conditions exceed recommended operating conditions.
2. Recommended operation conditions require accuracy of the power supplies as described in Section 10.3.
3. The ambient operating temperature listed above only applies to the industrial grade 8201I device.
4. The ambient operating temperature for all devices must be managed with any combination of device speed rating, heatsink, or airflow to guarantee Tj is not exceeded.

10.3 Power Supplies

The following subsections provide details concerning the digital and analog power supply connections on the 820x device. Generally speaking, analog supplies can be derived from the same power source as the digital supplies, but are more noise-sensitive and require additional filtering and careful layout/routing.

10.3.1 Digital Power Supplies

Table 10-3. VDD_25_33 Power Supply Requirements

Parameter	Description	Min	Max	Units
Operating range	Voltage range for nominal operating conditions for 2.5V	2.25	2.75	V
	Voltage range for nominal operating conditions for 3.3V	2.97	3.63	V
Rise time	Time from 10% to 90% mark	0.1	10	ms
Overshoot	Maximum overshoot allowed		80	mV
Ripple	Maximum voltage ripple		60	mV

Table 10-4. VDD_10 Power Supply Requirements

Parameter	Description	Min	Max	Units
Operating range	Voltage range for nominal operating conditions	0.95	1.05	V
Rise time	Time from 10% to 90% mark	0.1	10	ms
Overshoot	Maximum overshoot allowed		50	mV
Ripple	Maximum voltage ripple		30	mV

10.3.2 Analog Power Supplies

Table 10-5. VDD_25A Power Supply Requirements

Parameter	Description	Min	Max	Units
Operating range	Voltage range for nominal operating conditions	2.38	2.63	V
Rise time	Time from 10% to 90% mark	0.1	10	ms
Overshoot	Maximum overshoot allowed		80	mV
Ripple	Maximum voltage ripple		60	mV

Table 10-6. VDD_10A Power Supply Requirements

Parameter	Description	Min	Max	Units
Operating range	Voltage range for nominal operating conditions	0.95	1.05	V
Rise time	Time from 10% to 90% mark	0.1	10	ms
Overshoot	Maximum overshoot allowed		50	mV
Ripple	Maximum voltage ripple		30	mV

Table 10-7. VDDA_PLL0, VDDA_PLL1 Power Supply Requirements

Parameter	Description	Min	Max	Units
Operating range	Voltage range for nominal operating conditions	2.38	2.63	V
Rise time	Time from 10% to 90% mark	0.1	10	ms
Overshoot	Maximum overshoot allowed		80	mV
Ripple	Maximum voltage ripple		60	mV

10.4 Power Sequencing

The 820X power supplies do not have any sequencing requirements. However, all 1.0 V supplies should be powered on simultaneously.

10.5 Power Consumption

Table 10-8. 8204 Current and Power Per Power Domain

Power Supply	Nominal Voltage(V)	TYP Current (mA)	Max Current (mA)	TYP Power (mW)	Max Power[2] (mW)
VDD_25_33[1]	2.5	40	43	100	113
	3.3	40	43	132	149
VDD_10	1.0	1,700	3,320	1,700	3,486
VDD_25A	2.5	70	70	175	184
VDD_10A	1.0	60	80	60	84
VDDA_PLL0	2.5		1		2.6
VDDA_PLL1	2.5		1		2.6

Table 10-8. 8204 Current and Power Per Power Domain

Power Supply	Nominal Voltage(V)	TYP Current (mA)	Max Current (mA)	TYP Power (mW)	Max Power[2] (mW)
Total using 2.5v nominal of VDD_25_33		1,870	3,515	2,035	3,872
Total using 3.3v nominal of VDD_25_33		1,870	3,515	2,067	3,908
Note: 1. VDD_25_33 current listed does not include any current delivered to external loads driven by the 820x. 2. Max power is calculated as 105% of nominal, at 125°C junction temperature.					

Table 10-9. 8203 Current and Power Per Power Domain

Power Supply	Nominal Voltage(V)	TYP Current (mA)	Max Current (mA)	TYP Power (mW)	Max Power[2] (mW)
VDD_25_33[1]	2.5	30	43	75	113
	3.3	30	43	99	149
VDD_10	1.0	1,160	2,530	1,160	2,657
VDD_25A	2.5	70	70	175	184
VDD_10A	1.0	60	80	60	84
VDDA_PLL0	2.5		1		2.6
VDDA_PLL1	2.5		1		2.6
Total using 2.5v nominal of VDD_25_33		1,320	2,725	1,470	3,042
Total using 3.3v nominal of VDD_25_33		1,320	2,725	1,494	3,078
Note: 1. VDD_25_33 current listed does not include any current delivered to external loads driven by the 820x. 2. Max power is calculated as 105% of nominal, at 125°C junction temperature.					

Table 10-10. 8202 Current and Power Per Power Domain

Power Supply	Nominal Voltage(V)	TYP Current (mA)	Max Current (mA)	TYP Power (mW)	Max Power[2] (mW)
VDD_25_33[1]	2.5	30	43	75	113
	3.3	30	43	99	149
VDD_10	1.0	765	2,170	765	2,279
VDD_25A	2.5	70	70	175	184
VDD_10A	1.0	60	80	60	84
VDDA_PLL0	2.5		1		2.6
VDDA_PLL1	2.5		1		2.6

Table 10-10. 8202 Current and Power Per Power Domain

Power Supply	Nominal Voltage(V)	TYP Current (mA)	Max Current (mA)	TYP Power (mW)	Max Power[2] (mW)
Total using 2.5v nominal of VDD_25_33		925	2,365	1,075	2,664
Total using 3.3v nominal of VDD_25_33		925	2,365	1,099	2,700
Note: 1. VDD_25_33 current listed does not include any current delivered to external loads driven by the 820x. 2. Max power is calculated as 105% of nominal, at 125°C junction temperature.					

Table 10-11. Commercial 8201 Current and Power Per Power Domain

Power Supply	Nominal Voltage(V)	TYP Current (mA)	Max Current (mA)	TYP Power (mW)	Max Power[2] (mW)
VDD_25_33[1]	2.5	30	43	75	113
	3.3	30	43	99	149
VDD_10	1.0	580	1,970	580	2,069
VDD_25A	2.5	70	70	175	184
VDD_10A	1.0	60	80	60	84
VDDA_PLL0	2.5		1		2.6
VDDA_PLL1	2.5		1		2.6
Total using 2.5v nominal of VDD_25_33		740	2,165	890	2,454
Total using 3.3v nominal of VDD_25_33		740	2,165	914	2,490
Note: 1. VDD_25_33 current listed does not include any current delivered to external loads driven by the 820x. 2. Max power is calculated as 105% of nominal, at 125°C junction temperature.					

Table 10-12. Industrial 8201 Current and Power Per Power Domain

Power Supply	Nominal Voltage(V)	TYP Current (mA)	Max Current (mA)	TYP Power (mW)	Max Power[2] (mW)
VDD_25_33[1]	2.5	30	43	75	113
	3.3	30	43	99	149
VDD_10	1.0	580	1,500	580	1,575
VDD_25A	2.5	70	70	175	184
VDD_10A	1.0	60	80	60	84
VDDA_PLL0	2.5		1		2.6

Table 10-12. Industrial 8201 Current and Power Per Power Domain

Power Supply	Nominal Voltage(V)	TYP Current (mA)	Max Current (mA)	TYP Power (mW)	Max Power[2] (mW)
VDDA_PLL1	2.5		1		2.6
Total using 2.5v nominal of VDD_25_33		740	1695	890	1,961
Total using 3.3v nominal of VDD_25_33		740	1695	914	1,997
Note: 1. VDD_25_33 current listed does not include any current delivered to external loads driven by the 820x. 2. Max power is calculated as 105% of nominal, at 125°C junction temperature.					

10.6 I/O Characteristics

Table 10-13. Normal IO Characteristics

Symbol	Description	Min	TYP	Max	Units
V_{IH}	DC Input High Voltage	0.7 * VDD_25_33		1.1 * VDD_25_33	V
V_{IL}	DC Input Low Voltage	- 0.1 * VDD_25_33		0.3 * VDD_25_33	V
VDD_25_33	I/O supply voltage at 2.5V	2.25	2.5	2.75	V
	I/O supply voltage at 3.3V	2.97	3.3	3.63	V
R_{pull}	External resistor value for pull down or pull up pins	1K		10K	Ω

Table 10-14. PCIe PHY Transmitter Characteristics

Symbol	Description	Min	TYP	Max	Units
V_{CTM}	Transmit common mode voltage	400		600	mV
Z_D	Differential output impedance	85		115	Ω

Table 10-15. PCIe PHY Receiver Characteristics

Symbol	Description	Min	TYP	Max	Units
$V_{MIN\_RX\_EYE\_HEIGHT}$	Minimum Rx eye height (differential peak-to-peak)			175	mV
Z_{IN}	Differential input impedance	85		115	Ω
PPM	Tolerance	-350		350	ppm

11 AC Specifications

11.1 Reset Timing

The 820x has three reset pins, two of which contain an internal Schmitt circuit.

PERST_N This reset pin with hysteresis should be connected to PCIe reset.

POR_N This is an asynchronous power on reset with hysteresis.

Due to a erratum, to ensure that the 820x operates correctly on all platforms, POR_N and PERST_N must be tied to the PCIe reset.

JTAG_TRST_N This is an asynchronous reset for JTAG and other logic.

There is an errata for the JTAG_TRST_N pin that requires that JTAG_TRST_N is driven low during power on reset. If JTAG testing is desired, JTAG_TRST_N may be connected to a pull-down resistor, otherwise, JTAG_TRST_N should be connected to PERST_N and POR_N for normal operation.

11.2 PLL Clock Input

Table 11-1. PLL0 and PLL1 Reference Clock Requirements

Symbol	Description	Min	Typ	Max	Units
F_{RefClk}	Input clock frequency	10	66.66	73	MHz
$D.C._{RefClk}$	Duty Cycle	45		55	%
$J_{CLK-REF}$	Input Jitter (peak-to-peak)		0.4		ns
Frequency tolerance	Input clock frequency deviation			150	ppm
Aging	Input clock long time deviation			+/- 5	ppm/ year
T_{rdy}	Lock time		0.25		µs
Notes: 1. Spread spectrum clocks are not supported. 2. The 820x performance scales as a percentage of the typical input clock frequency. For example, if the input clock frequency is 50 MHz, the performance will be reduced by 50/66 or 75%.					

11.3 SPI Interface Timing

This section describes the Serial Peripheral Interface timing.

The SPI does not conform to the JEDEC standard for the all command definitions, in particular, the RDID command. Refer to the *820x Hardware Design Guide*, UG-0211, for a table of supported devices and their capabilities.

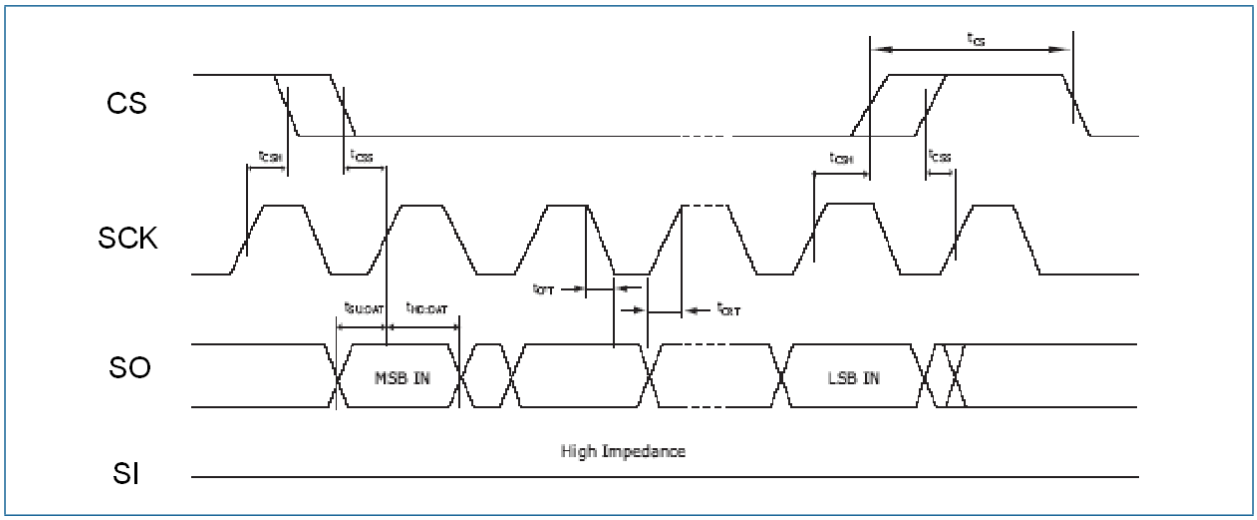


Figure 11-1. SPI Write Timing

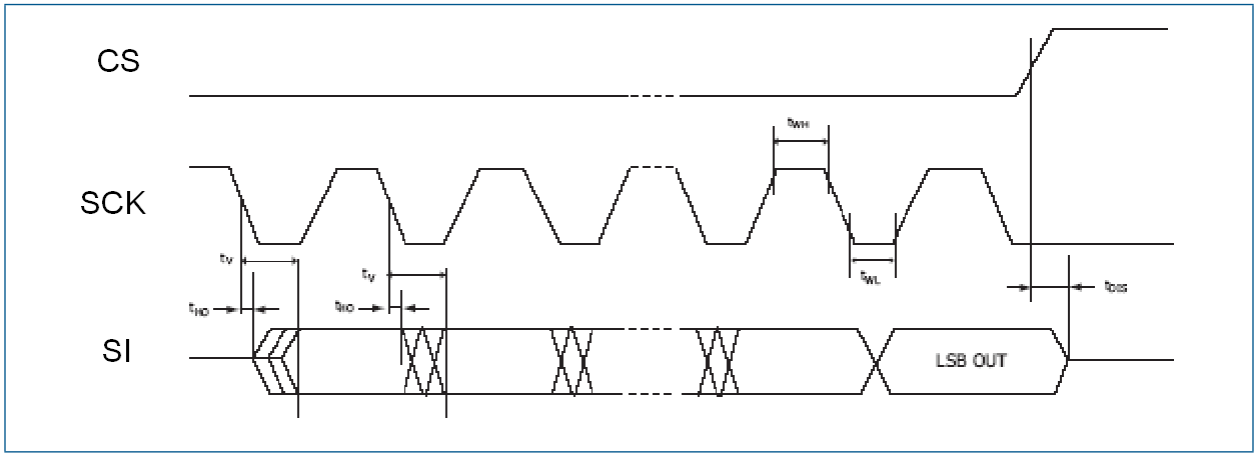


Figure 11-2. SPI Read Timing

Table 11-2. SPI Interface AC Characteristics

Symbol	Description	Min	Typ	Max	Units
t_{WH}	Programmable device SCK high time	9			ns
t_{WL}	Programmable device SCK low time	9			ns
t_{CSS}	Programmable device CS setup time	5			ns
t_{CSH}	Programmable device CS hold time	5			ns

Table 11-2. SPI Interface AC Characteristics

Symbol	Description	Min	Typ	Max	Units
t_V	Programmable device SI valid time	0		10	ns
t_{HO}	Programmable device SI hold time	0			ns
$t_{HD:DAT}$	Programmable device SO hold time	5			ns
$t_{SU:DAT}$	Programmable device SO setup time	5			ns
Note: The parameters in this table were measured with a 30pf load.					

11.4 JTAG Interface Timing

The 820x is designed to support the IEEE 1149.1 JTAG standard.

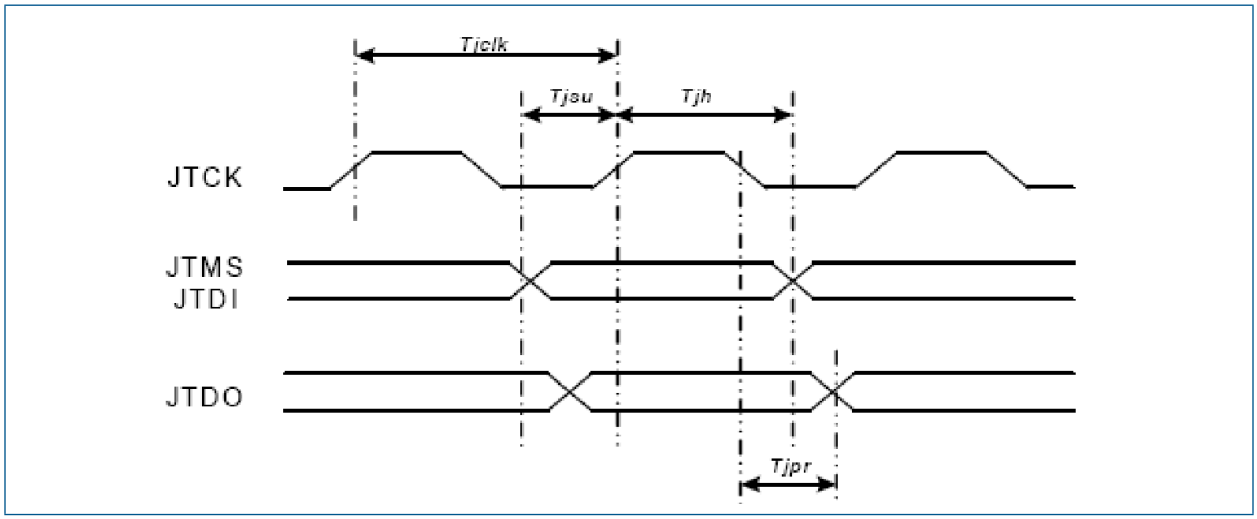


Figure 11-3. JTAG Timing

Table 11-3. JTAG Interface AC Characteristics

Symbol	Description	Min	Max	Units
T_{jclk}	JTAG clock frequency		20	ns
T_{th}	JTAG_TMS and JTAG_TDI hold time	10		ns
T_{jsu}	JTAG_TMS and JTAG_TDI setup time	10		ns
T_{jpr}	JTAG_TDO propagation delay		15	ns

11.5 PCIe Interface Timing

The 820x PCIe interface is fully compliant to the PCI Express Electromechanical Specification Revision 2.0 standard.

12 Thermal Specifications

Each application will require thermal analysis based on its system environment.

Table 12-1. Thermal operating conditions

Symbol	Description	Min	Max	Units
Tj	Junction Temperature (applies to commercial parts)	-40	125	°C
Ta	Ambient Temperature (applies to industrial 8201I part only)	-40	85	°C

Notes:

1. For normal device operation, adhere to the limits in this table. Sustained operations of a device at conditions exceeding these values, even if they are within the absolute maximum rating limits, may result in permanent device damage or impaired device reliability. Device functionality to stated DC and AC limits is not guaranteed if conditions exceed recommended operating conditions.

2. Recommended operation conditions require accuracy of the power supplies as described in Section 10.3.

Table 12-2. Thermal Specifications for Commercial Parts

Parameter	Max
Junction Temperature (Tj)	125 °C
Internal thermal resistance, (θ_{jc})	6.5 °C/W
Thermal resistance, (θ_{ja}) at 0 m/s airflow	20.4 °C/W
Thermal resistance, (θ_{ja}) at 1 m/s airflow	18 °C/W
Thermal resistance, (θ_{ja}) at 2 m/s airflow	17 °C/W
Temperature correlation (Ψ jt)	4.2 °C/W

12.1 Thermal Sensor Controller

The 820x contains a thermal sensor controller that can be used to read or monitor the 820x die temperature. Please refer to Section 5.13, "Temperature Sensor Controller"" and Section 6.8, "Temperature Sensor Controller Registers"" for detailed information.

13 Package Specifications

This chapter provides general and mechanical package information, as well as ball assignment drawings.

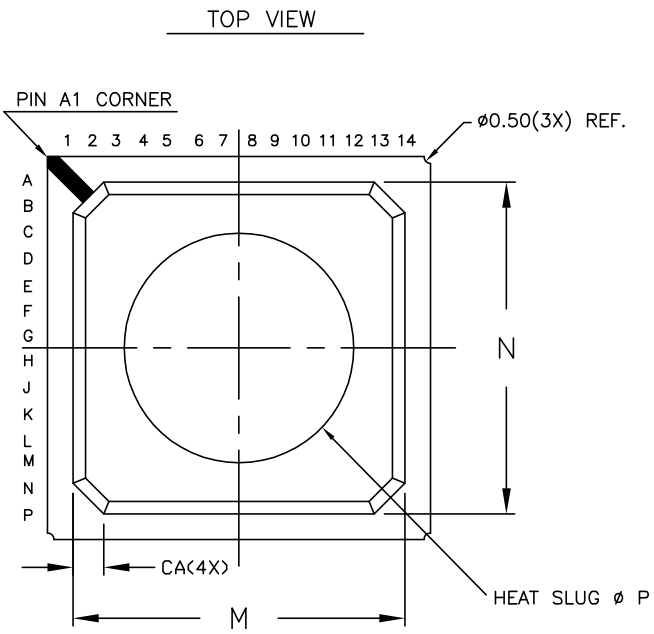
13.1 General Information

Table 13-1. General Package Information

Package Information	Description
Package Type	HSBGA
Ball Count	196
Package Size	15mm x 15mm
Pitch	1.0 mm

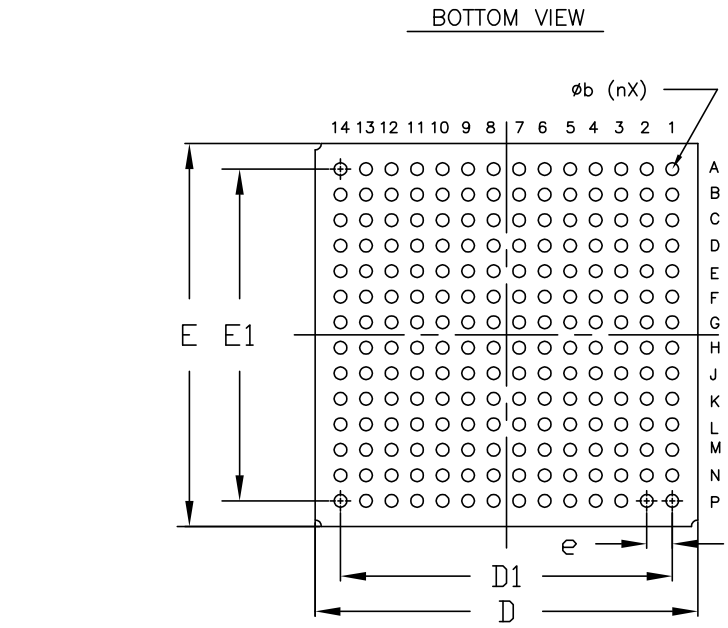
13.2 Mechanical Information

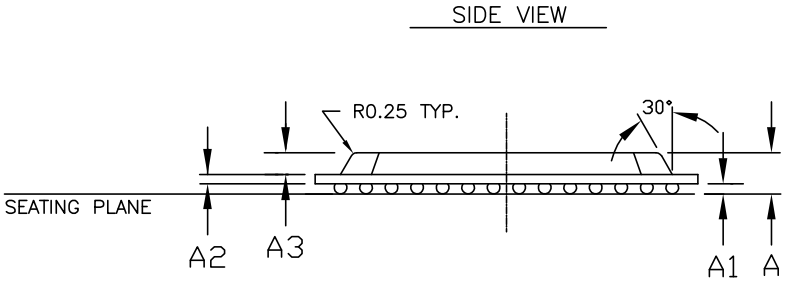
The 820x is packaged as a 196 ball HSBGA (15 x 15 mm body, 1 mm ball pitch). The package dimensions from a top view are given in Figure 13-2. The package dimensions from a bottom and side view are shown in Figure 13-2.



196–Ball HSBGA						
SYMBOLS	DIMENSIONS IN MM (Control Unit)			DIMENSIONS IN INCH (Reference Unit)		
	MIN	NOM	MAX	MIN	NOM	MAX
A	1.62	1.81	2.00	0.064	0.071	0.079
A1	0.30	0.40	0.50	0.012	0.016	0.020
A2	0.560 REF			0.022 REF		
A3	0.850 REF			0.033 REF		
D	14.80	15.00	15.20	0.583	0.591	0.598
D1	13.00 BSC			0.512 BSC		
E	14.80	15.00	15.20	0.583	0.591	0.598
E1	13.00 BSC			0.512 BSC		
b	0.40	0.50	0.60	0.016	0.020	0.024
e	1.00 BSC			0.039 BSC		
M	12.80	13.00	13.20	0.504	0.512	0.520
N	12.80	13.00	13.20	0.504	0.512	0.520
P	8.00	8.50	9.00	0.315	0.335	0.354
CA	1.20X45° REF			0.047X45° REF		
n	196			196		

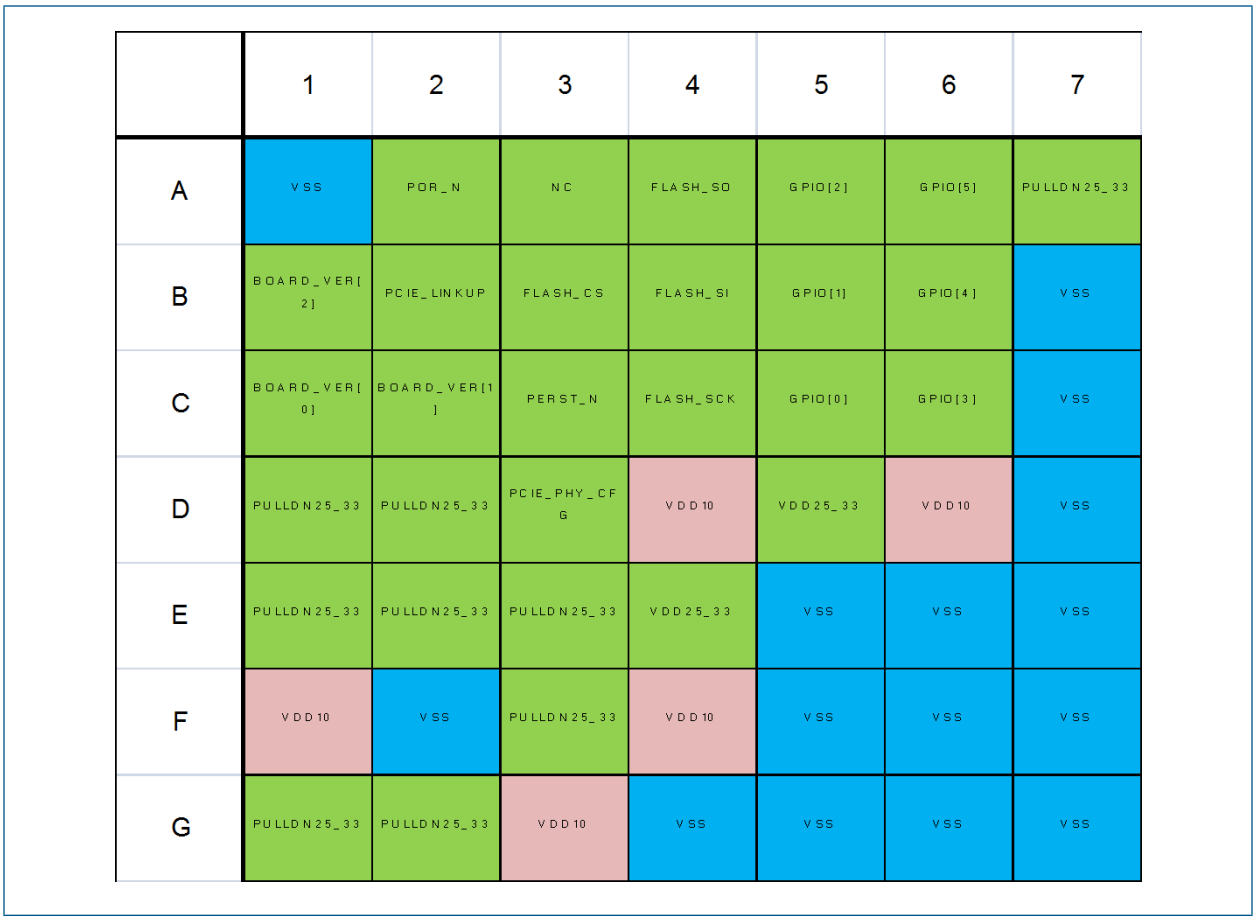
Figure 13-1. Top View Package Dimensions





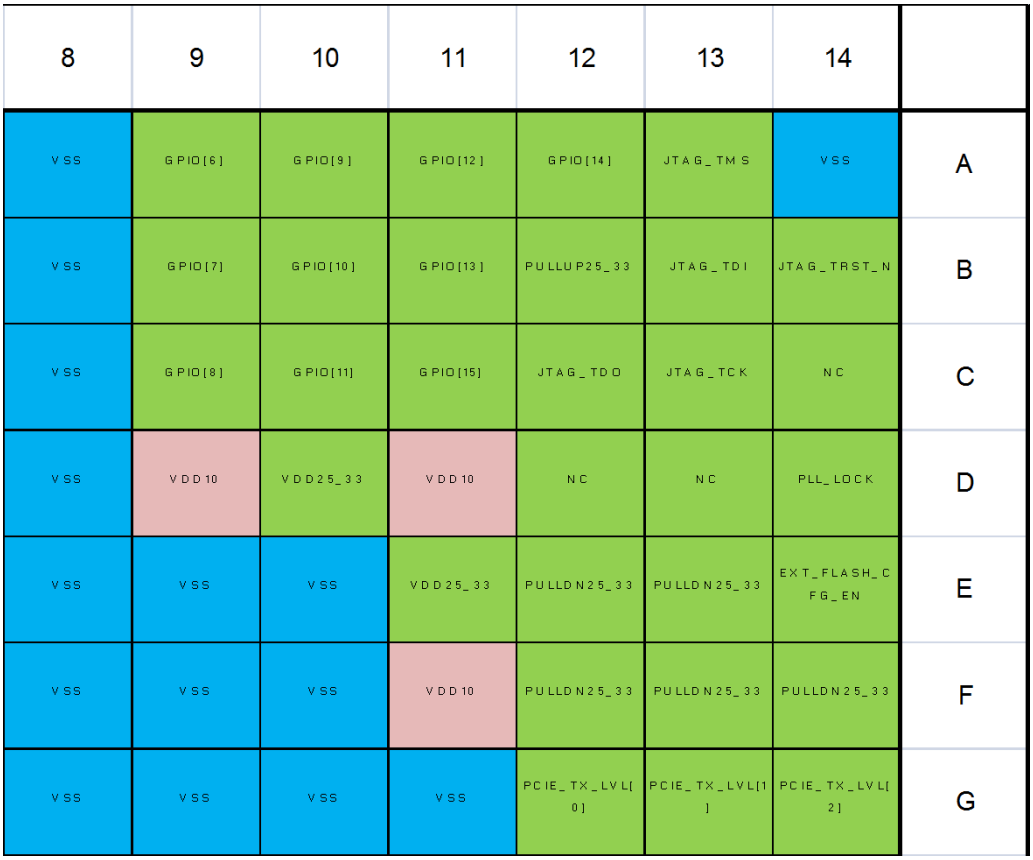
196–Ball HSBGA						
SYMBOLS	DIMENSIONS IN MM (Control Unit)			DIMENSIONS IN INCH (Reference Unit)		
	MIN	NOM	MAX	MIN	NOM	MAX
A	1.62	1.81	2.00	0.064	0.071	0.079
A1	0.30	0.40	0.50	0.012	0.016	0.020
A2	0.560 REF			0.022 REF		
A3	0.850 REF			0.033 REF		
D	14.80	15.00	15.20	0.583	0.591	0.598
D1	13.00 BSC			0.512 BSC		
E	14.80	15.00	15.20	0.583	0.591	0.598
E1	13.00 BSC			0.512 BSC		
b	0.40	0.50	0.60	0.016	0.020	0.024
e	1.00 BSC			0.039 BSC		

Figure 13-2. Bottom View and Side View Package Dimensions



	1	2	3	4	5	6	7
A	VSS	POR_N	NC	FLASH_SO	GPIO[2]	GPIO[5]	PULLDN25_33
B	BOARD_VER[2]	PCIE_LINKUP	FLASH_CS	FLASH_SI	GPIO[1]	GPIO[4]	VSS
C	BOARD_VER[0]	BOARD_VER[1]	PERST_N	FLASH_SCK	GPIO[0]	GPIO[3]	VSS
D	PULLDN25_33	PULLDN25_33	PCIE_PHY_CFG	VDD10	VDD25_33	VDD10	VSS
E	PULLDN25_33	PULLDN25_33	PULLDN25_33	VDD25_33	VSS	VSS	VSS
F	VDD10	VSS	PULLDN25_33	VDD10	VSS	VSS	VSS
G	PULLDN25_33	PULLDN25_33	VDD10	VSS	VSS	VSS	VSS

Figure 13-3. Ball Map Drawing - Top View - Upper Left Quadrant



8	9	10	11	12	13	14	
VSS	GPIO[6]	GPIO[9]	GPIO[12]	GPIO[14]	JTAG_TMS	VSS	A
VSS	GPIO[7]	GPIO[10]	GPIO[13]	PULLUP25_33	JTAG_TDI	JTAG_TRST_N	B
VSS	GPIO[8]	GPIO[11]	GPIO[15]	JTAG_TDO	JTAG_TCK	NC	C
VSS	VDD10	VDD25_33	VDD10	NC	NC	PLL_LOCK	D
VSS	VSS	VSS	VDD25_33	PULLDN25_33	PULLDN25_33	EXT_FLASH_CFG_EN	E
VSS	VSS	VSS	VDD10	PULLDN25_33	PULLDN25_33	PULLDN25_33	F
VSS	VSS	VSS	VSS	PCIE_TX_LVL[0]	PCIE_TX_LVL[1]	PCIE_TX_LVL[2]	G

Figure 13-4. Ball Map Drawing - Top View - Upper Right Quadrant

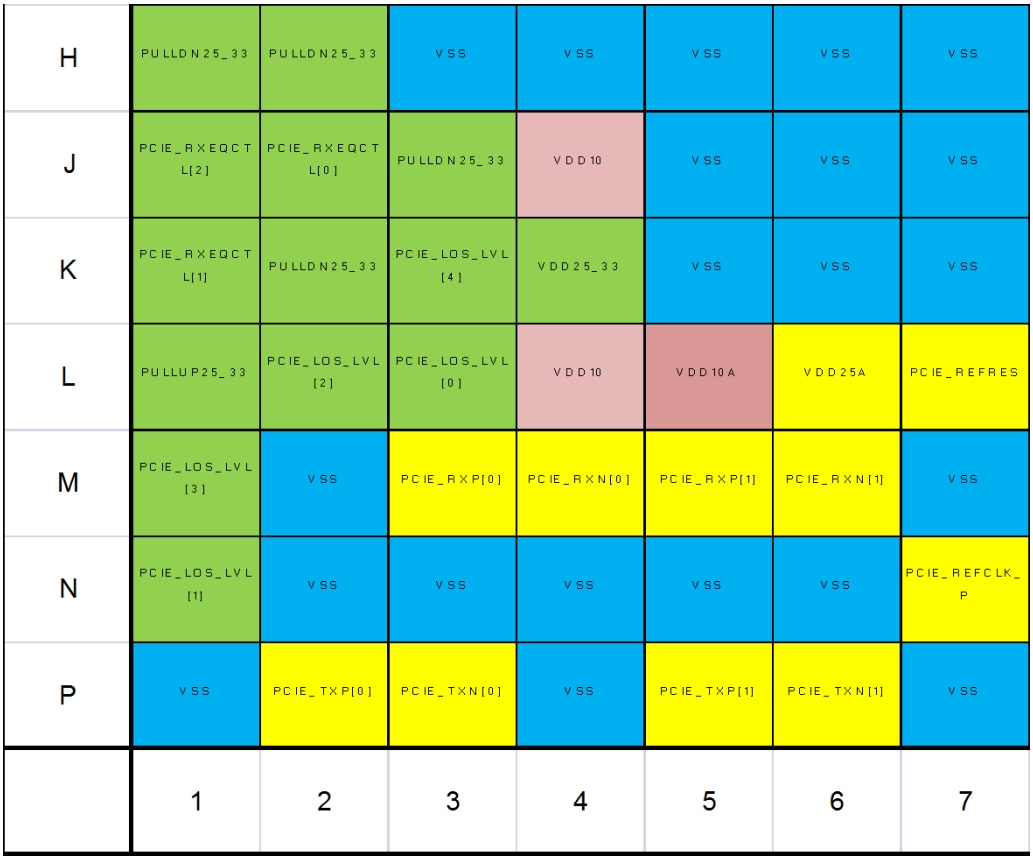


Figure 13-5. Ball Map Drawing - Top View - Lower Left Quadrant

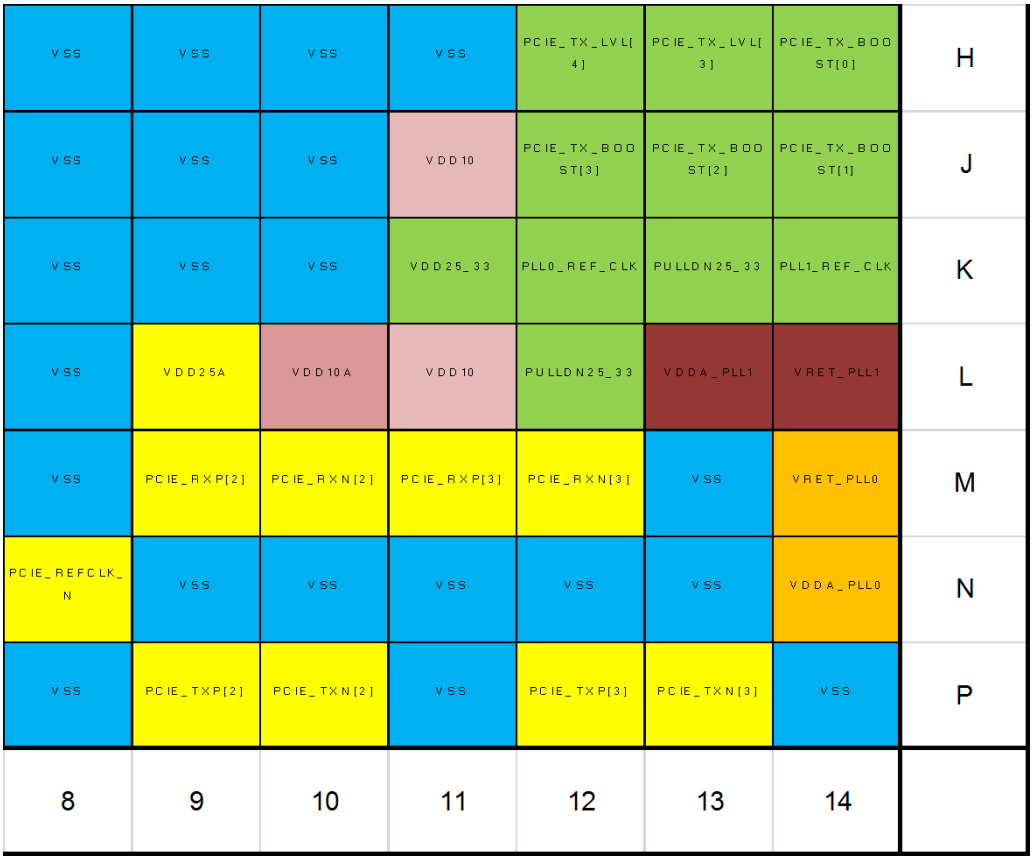


Figure 13-6. Ball Map Drawing - Top View - Lower Right Quadrant

Table 13-2. Alphabetical Ball List

Signal	Ball
BOARD_VERSION[0]	C1
BOARD_VERSION[1]	C2
BOARD_VERSION[2]	B1
EXT_FLASH_CFG_EN	E14
FLASH_CS	B3
FLASH_SCK	C4
FLASH_SI	B4
FLASH_SO	A4
GPIO[0]	C5
GPIO[1]	B5
GPIO[10]	B10
GPIO[11]	C10
GPIO[12]	A11
GPIO[13]	B11
GPIO[14]	A12
GPIO[15]	C11
GPIO[2]	A5
GPIO[3]	C6
GPIO[4]	B6
GPIO[5]	A6
GPIO[6]	A9
GPIO[7]	B9
GPIO[8]	C9
GPIO[9]	A10
JTAG_TCK	C13
JTAG_TDI	B13
JTAG_TDO	C12
JTAG_TMS	A13
JTAG_TRST_N	B14
NC	A3
NC	C14
NC	D12
NC	D13
PCIE_LINKUP	B2
PCIE_LOS_LVL[0]	L3
PCIE_LOS_LVL[1]	N1
PCIE_LOS_LVL[2]	L2
PCIE_LOS_LVL[3]	M1
PCIE_LOS_LVL[4]	K3
PCIE_PHY_CFG	D3
PCIE_REFCLK_N	N8
PCIE_REFCLK_P	N7
PCIE_REFRES	L7
PCIE_RXEQCTL[0]	J2
PCIE_RXEQCTL[1]	K1
PCIE_RXEQCTL[2]	J1
PCIE_RXN[0]	M4
PCIE_RXN[1]	M6
PCIE_RXN[2]	M10
PCIE_RXN[3]	M12
PCIE_RXP[0]	M3
PCIE_RXP[1]	M5
PCIE_RXP[2]	M9
PCIE_RXP[3]	M11
PCIE_TX_BOOST[0]	H14
PCIE_TX_BOOST[1]	J14
PCIE_TX_BOOST[2]	J13
PCIE_TX_BOOST[3]	J12
PCIE_TX_LVL[0]	G12
PCIE_TX_LVL[1]	G13
PCIE_TX_LVL[2]	G14
PCIE_TX_LVL[3]	H13
PCIE_TX_LVL[4]	H12
PCIE_TXN[0]	P3
PCIE_TXN[1]	P6
PCIE_TXN[2]	P10
PCIE_TXN[3]	P13
PCIE_TXP[0]	P2
PCIE_TXP[1]	P5
PCIE_TXP[2]	P9
PCIE_TXP[3]	P12
PERST_N	C3
PLL_LOCK	D14
PLL0_REF_CLK	K12
PLL1_REF_CLK	K14
POR_N	A2
PULLDN25_33	A7
PULLDN25_33	D1
PULLDN25_33	D2
PULLDN25_33	E1
PULLDN25_33	E2
PULLDN25_33	E3
PULLDN25_33	E12
PULLDN25_33	E13
PULLDN25_33	F3
PULLDN25_33	F12
PULLDN25_33	F13
PULLDN25_33	F14
PULLDN25_33	G1
PULLDN25_33	G2
PULLDN25_33	H1
PULLDN25_33	H2
PULLDN25_33	J3
PULLDN25_33	K2
PULLDN25_33	K13
PULLDN25_33	L12
PULLUP25_33	B12
PULLUP25_33	L1
VDD10	D4
VDD10	D6
VDD10	D9
VDD10	D11
VDD10	F1
VDD10	F4
VDD10	F11
VDD10	G3
VDD10	J4
VDD10	J11

Signal	Ball
VDD10	L4
VDD10	L11
VDD10A	L5
VDD10A	L10
VDD25_33	D5
VDD25_33	D10
VDD25_33	E4
VDD25_33	E11
VDD25_33	K4
VDD25_33	K11
VDD25A	L6
VDD25A	L9
VDDA_PLL0	N14
VDDA_PLL1	L13
VRET_PLL0	M14
VRET_PLL1	L14
VSS	A1
VSS	A8
VSS	A14
VSS	B7
VSS	B8
VSS	C7
VSS	C8
VSS	D7
VSS	D8
VSS	E5
VSS	E6
VSS	E7
VSS	E8
VSS	E9
VSS	E10
VSS	F2
VSS	F5
VSS	F6
VSS	F7
VSS	F8
VSS	F9

Signal	Ball
VSS	F10
VSS	G4
VSS	G5
VSS	G6
VSS	G7
VSS	G8
VSS	G9
VSS	G10
VSS	G11
VSS	H3
VSS	H4
VSS	H5
VSS	H6
VSS	H7
VSS	H8
VSS	H9
VSS	H10
VSS	H11
VSS	J5
VSS	J6
VSS	J7
VSS	J8
VSS	J9
VSS	J10
VSS	K5
VSS	K6
VSS	K7
VSS	K8
VSS	K9
VSS	K10
VSS	L8
VSS	M2
VSS	M7
VSS	M8
VSS	M13
VSS	N2
VSS	N3

Signal	Ball
VSS	N4
VSS	N5
VSS	N6
VSS	N9
VSS	N10
VSS	N11
VSS	N12
VSS	N13
VSS	P1
VSS	P4
VSS	P7
VSS	P8
VSS	P11
VSS	P14

Table 13-3. Numeric Ball List

Signal	Ball
VSS	A1
GPIO[9]	A10
GPIO[12]	A11
GPIO[14]	A12
JTAG_TMS	A13
VSS	A14
POR_N	A2
NC	A3
FLASH_SO	A4
GPIO[2]	A5
GPIO[5]	A6
PULLDN25_33	A7
VSS	A8
GPIO[6]	A9
BOARD_VERSION[2]	B1
GPIO[10]	B10
GPIO[13]	B11
PULLUP25_33	B12
JTAG_TDI	B13
JTAG_TRST_N	B14
PCIE_LINKUP	B2
FLASH_CS	B3
FLASH_SI	B4
GPIO[1]	B5
GPIO[4]	B6
VSS	B7
VSS	B8
GPIO[7]	B9
BOARD_VERSION[0]	C1
GPIO[11]	C10
GPIO[15]	C11
JTAG_TDO	C12
JTAG_TCK	C13
NC	C14
BOARD_VERSION[1]	C2
PERST_N	C3

Signal	Ball
FLASH_SCK	C4
GPIO[0]	C5
GPIO[3]	C6
VSS	C7
VSS	C8
GPIO[8]	C9
PULLDN25_33	D1
VDD25_33	D10
VDD10	D11
NC	D12
NC	D13
PLL_LOCK	D14
PULLDN25_33	D2
PCIE_PHY_CFG	D3
VDD10	D4
VDD25_33	D5
VDD10	D6
VSS	D7
VSS	D8
VDD10	D9
PULLDN25_33	E1
VSS	E10
VDD25_33	E11
PULLDN25_33	E12
PULLDN25_33	E13
EXT_FLASH_CFG_EN	E14
PULLDN25_33	E2
PULLDN25_33	E3
VDD25_33	E4
VSS	E5
VSS	E6
VSS	E7
VSS	E8
VSS	E9
VDD10	F1
VSS	F10

Signal	Ball
VDD10	F11
PULLDN25_33	F12
PULLDN25_33	F13
PULLDN25_33	F14
VSS	F2
PULLDN25_33	F3
VDD10	F4
VSS	F5
VSS	F6
VSS	F7
VSS	F8
VSS	F9
PULLDN25_33	G1
VSS	G10
VSS	G11
PCIE_TX_LVL[0]	G12
PCIE_TX_LVL[1]	G13
PCIE_TX_LVL[2]	G14
PULLDN25_33	G2
VDD10	G3
VSS	G4
VSS	G5
VSS	G6
VSS	G7
VSS	G8
VSS	G9
PULLDN25_33	H1
VSS	H10
VSS	H11
PCIE_TX_LVL[4]	H12
PCIE_TX_LVL[3]	H13
PCIE_TX_BOOST[0]	H14
PULLDN25_33	H2
VSS	H3
VSS	H4
VSS	H5

Signal	Ball
VSS	H6
VSS	H7
VSS	H8
VSS	H9
PCIE_RXEQCTL[2]	J1
VSS	J10
VDD10	J11
PCIE_TX_BOOST[3]	J12
PCIE_TX_BOOST[2]	J13
PCIE_TX_BOOST[1]	J14
PCIE_RXEQCTL[0]	J2
PULLDN25_33	J3
VDD10	J4
VSS	J5
VSS	J6
VSS	J7
VSS	J8
VSS	J9
PCIE_RXEQCTL[1]	K1
VSS	K10
VDD25_33	K11
PLL0_REF_CLK	K12
PULLDN25_33	K13
PLL1_REF_CLK	K14
PULLDN25_33	K2
PCIE_LOS_LVL[4]	K3
VDD25_33	K4
VSS	K5
VSS	K6
VSS	K7
VSS	K8
VSS	K9
PULLUP25_33	L1
VDD10A	L10
VDD10	L11
PULLDN25_33	L12
VDDA_PLL1	L13

Signal	Ball
VRET_PLL1	L14
PCIE_LOS_LVL[2]	L2
PCIE_LOS_LVL[0]	L3
VDD10	L4
VDD10A	L5
VDD25A	L6
PCIE_REFRES	L7
VSS	L8
VDD25A	L9
PCIE_LOS_LVL[3]	M1
PCIE_RXN[2]	M10
PCIE_RXP[3]	M11
PCIE_RXN[3]	M12
VSS	M13
VRET_PLL0	M14
VSS	M2
PCIE_RXP[0]	M3
PCIE_RXN[0]	M4
PCIE_RXP[1]	M5
PCIE_RXN[1]	M6
VSS	M7
VSS	M8
PCIE_RXP[2]	M9
PCIE_LOS_LVL[1]	N1
VSS	N10
VSS	N11
VSS	N12
VSS	N13
VDDA_PLL0	N14
VSS	N2
VSS	N3
VSS	N4
VSS	N5
VSS	N6
PCIE_REFCLK_P	N7
PCIE_REFCLK_N	N8
VSS	N9

Signal	Ball
VSS	P1
PCIE_TXN[2]	P10
VSS	P11
PCIE_TXP[3]	P12
PCIE_TXN[3]	P13
VSS	P14
PCIE_TXP[0]	P2
PCIE_TXN[0]	P3
VSS	P4
PCIE_TXP[1]	P5
PCIE_TXN[1]	P6
VSS	P7
VSS	P8
PCIE_TXP[2]	P9

Appendix A: IPsec Encapsulating Security Payload (ESP) Format

This section describes the format of the IPsec ESP fields as well as the encryption and authentication coverage of these fields.

Table A-1. IPsec ESP Packet Field Description

Section	Field Name	Size (bytes)	Description	Encryption Coverage	Authentication Coverage
ESP Header	SPI	4	Security Parameter Index (SPI): A 32-bit value that is combined with the destination address and security protocol type to identify the security association to be used for this datagram. See the topic on security associations for more details.		√
	Sequence Number	4	Sequence Number: A counter field initialized to zero when a security association is formed between two devices, and then incremented for each datagram sent using that SA. This is used to provide protection against replay attacks.		√
Payload	Payload Data	variable	Payload Data: The encrypted payload data, consisting of a higher layer message or encapsulated IP datagram. May also include support information such as an initialization vector, required by certain encryption methods.	√	√
ESP Trailer	Padding	variable (0 to 255)	Additional padding bytes included as needed for encryption or for alignment.	√	√
	Pad Length	1	The number of bytes in the preceding Padding field.	√	√
	Next Header	1	Contains the protocol number of the next header in the datagram. Used to chain together headers.	√	√
ESP Authentication Data		variable	This field contains the Integrity Check Value (ICV) resulting from the application of the optional ESP authentication algorithm.		

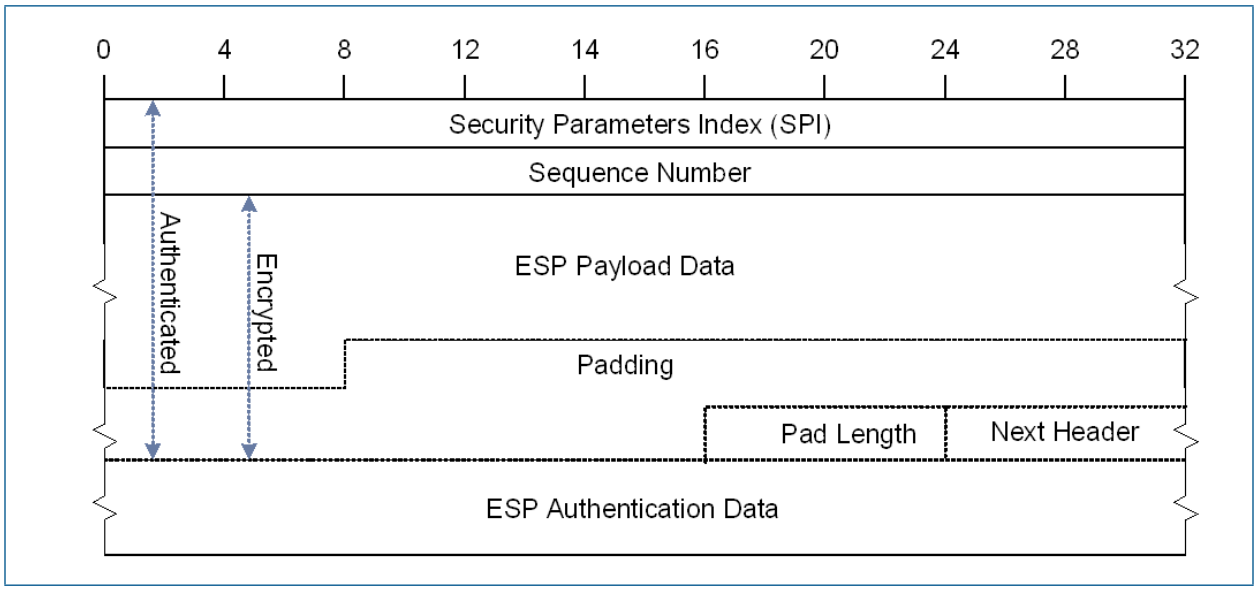


Figure A-1. IPsec ESP Packet Format

Appendix B: CRC Algorithms

B.1 Key CRC and XTS DIF

AES-XTS mode DIF CRC and other key CRCs use the CRC16 algorithm below:

$$F(x) = x16 + x15 + x11 + x9 + x8 + x7 + x5 + x4 + x2 + x + 1$$

The initial value for the function is "0xFFFF" (refer to t10-dif-03, http://www.t10.org).

Note that the generated CRC for the key need to be swapped.

```
uint dif_crc16(uint bCnt,const void *data,uint crcVal) {
  uint i,j;
  const u08b *p = (const u08b *)data;

  assert((crcVal & ~0xFFFF) == 0);
  for (i=0;i<bCnt;i++) {
    crcVal ^= (p[i] << 8);
    for (j=0;j<8;j++)
      crcVal = (crcVal << 1) ^ ((crcVal & 0x8000) ? 0x18BB7 : 0);
  }
  assert((crcVal & ~0xFFFF) == 0);
  return  crcVal;
}
```

B.2 Data CRC

Data CRCs use the CRC32 algorithm below:

$$F(x) = x32+x26+x23+x22+x16+x12+x11+x10+x8+ x7+x5+x4+x2+x+1$$

The initial value of the function is "0xFFFF_FFFF".

```
string CalcLCRC(const string Data)
{
  string result = "11111111111111111111111111111111";
  int byte_num = Data.length() / 8;
  string _crc, crc32;

  int poly[13] = { 6,9,10,16,20,21,22,24,25,27,28,30,31 };
  int src = 0 , dst = 0;

  while( dst < Data.length() )
  {
    if( Data.at(dst + 7 - 2*(dst%8)) != result.at(src) ){
      result.push_back('1');
      // xor poly 0x04c11db7
      for( int i=0; i<13; i++ )
        result.at( src + poly[i] ) = result.at( src + poly[i] ) == '0'?'1':'0';
    }else   result.push_back('0');
    src++ , dst++;
```

```
  }
  // TODO:: rewire result
  crc32 =  result.substr (src , 32 ) ;
  for ( int i = 0; i < 4 ; i ++ )
    for ( int j = 0; j < 8; j ++ )
      _crc = _crc + crc32.at( 8 * i + 7 - j );
  for ( int i = 0; i < 32; i ++ )
    _crc.at( i ) = ( _crc.at(i) == '0' ? '1' : '0' );
  return  _crc;
}
```

Appendix C: AES CBC/XTS FK Generation

For more information on the AES algorithm, please refer to the FIPS-197 Specification for the Advanced Encryption Standard (AES).

C.1 Key Expansion

The key expansion algorithm shown below operates on 32 bit word segments of the keys, where Nr is the number of rounds (iterations) of the AES algorithm, Nk is the number of 32 bit words in the key (4 for 128 bit keys), Nb is the number of words in a block (4 for this implementation).

```
KeyExpansion[Rounds+1][128](Key[128|256])
{
  reg[32] temp;

  if (sizeof(Key)==256)
    { KeyExpansion[0], KeyExpansion[1]} = Key;
  else    // 128 bit key
    KeyExpansion[0] = Key;
  for(k = sizeof(Key)==256 ? 2 : 1; k<=Rounds; k=k+1) {
    temp = ByteSub(RotByte(KeyExpansion[k-1][31:0]));
    if (sizeof(Key)==256) begin
      lastkey = ExpandedKey[k-2];
      if (k%2==0)
        temp = temp^{Rcon(k/2 - 1),24'h000000};
    end else begin  // 128 bit key
      lastkey = KeyExpansion[k-1];
      temp = temp^{Rcon(k-1),24'h000000};
    end
    temp = temp ^ lastkey[127:96];
    KeyExpansion[k][127:96] = temp;
    temp = temp ^ lastkey[95:64];
    KeyExpansion[k][95:64] = temp;
    temp = temp ^ lastkey[63:32];
    KeyExpansion[k][63:32] = temp;
    temp = temp ^ lastkey[31:0];
    KeyExpansion[k][31:0] = temp;
  }
}
```

For 256-bit keys, each 128 bit key expansion array member is used as the key by each round of the encryption/decryption algorithm. In this case, (Nr+1)/2 256-bit expanded keys are required.

C.2 Encryption Algorithm

An AES encryption algorithm involves an initial key expansion followed by a user specified number of encryption rounds represented here by Nr. Nr is one of the parameters specified in the input control word and is typically 10 for 128 bit keys, 14 for 256 bit keys, and a maximum of 32.

```
AES_encrypt(State[128], CipherKey[128/256], Rounds)
{
      array[128] ExpandedKey[Rounds+1];

ExpandedKey = KeyExpansion(CipherKey) ;
State = Text ^ ExpandedKey[0];
For( i=1 ; i<Rounds ; i++ ) {
  State = MixColumn(ShiftRow(ByteSub(State))) ^ ExpandedKey[i];
      }
Output = ShiftRow(ByteSub(State)) ^ ExpandedKey[Rounds];
}
```

Note that since the expanded keys are used in increasing order, the key expansion algorithm can run concurrently with the encryption algorithm.

C.3 Decryption Algorithm

The AES decryption algorithm is similar to the encryption algorithm, in that it has the same basic form and uses the same expanded key register array. What is different is that transformation functions in the decryption round are inverses of those used in the encryption case. The inverse functions are very similar in form to their forward counterparts. The decryption algorithm also uses keys in the reverse order from the expanded key array. This causes the decryption algorithm to run slightly slower than because all of the expansion keys must be calculated before decryption begins.

```
AES_decrypt(State[128], CipherKey[128/256], Rounds)
{
      array[128] ExpandedKey[Rounds+1];

ExpandedKey = KeyExpansion(CipherKey) ;
State = Text ^ ExpandedKey[Rounds];
For( i=Rounds-1 ; i>0 ; i-- ) {
State = InvMixColumn(InvShiftRow(InvByteSub(State)) ^ ExpandedKey[i]);
      }
Output = InvShiftRow(InvByteSub(State)) ^ ExpandedKey[0];
}
```

The AES Encryption key is different from the Decrypt key because the pre-calculated decryption key can improve AES performance. Refer to AES Spec §5.2 Key Expansion.

The C code for generating the FK in YD simulation environment can be reused.

LAN security standard IEEE 802.1ae (MACSec) and NIST SP 800-38D use AES cipher in the GCM mode, while the disk/tape encryption standard IEEE P1619 D16 uses the XTS mode. Legacy storage designs used the CBC mode, NIST SP 800-38A.

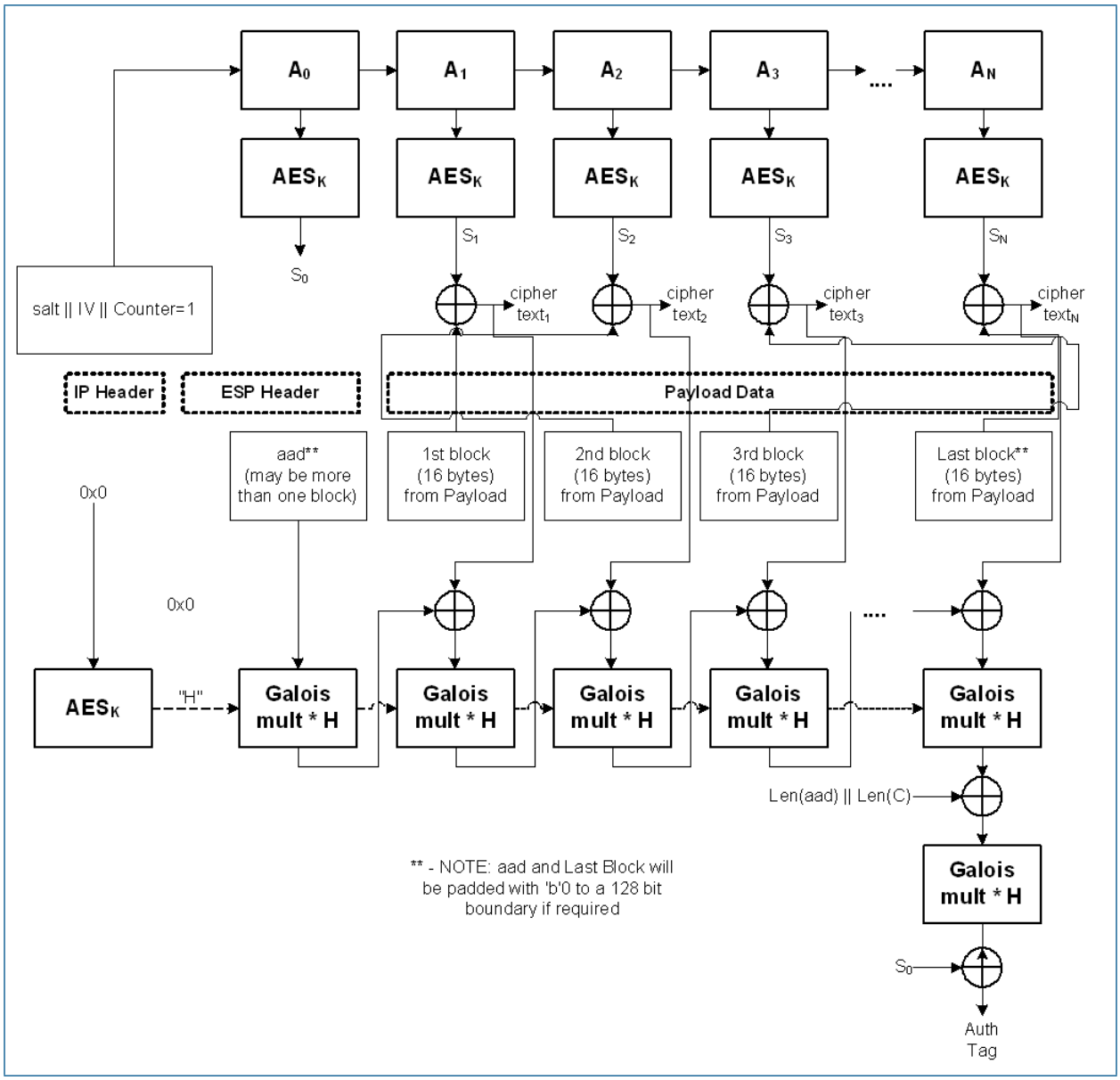


Figure C-1. GCM Mode

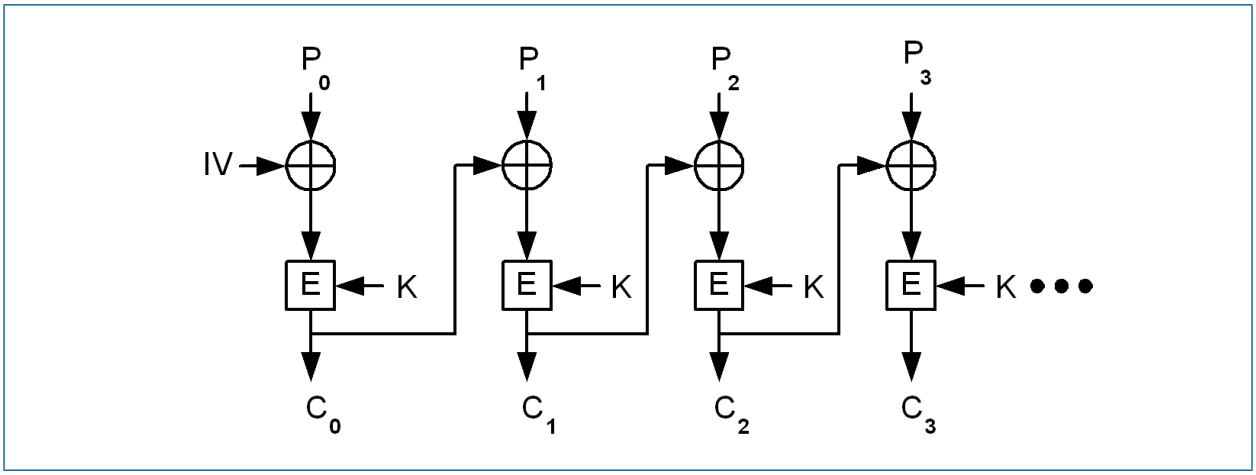


Figure C-2. CBC Mode

Appendix D: MAC and IPAD and OPAD

D.1 MAC in SSL3.0

The MAC in SSL3.0 is calculated as follows:

> Hash(Write Secret || Pad2 || Hash(Write Secret || Pad1 || Seq. No. || Type || Length || Application Data))

Where:

Hash = SHA-1 or MD5

Write Secret = 20 bytes if SHA-1, 16 bytes if MD5; from Crypto Header

Pad1 = 48/40 bytes of 0x36 (MD5/SHA-1)

Pad2 = 48/40 bytes of 0x5C (MD5/SHA-1)

SeqNum = 8 byte Sequence Number from Crypto Header

Type = Type byte from SSL Header

Length = 16 bit Length of payload (no padding)

D.2 MAC in TLS1.0

The MAC in TLS 1.0 is calculated as follows:

> HMACHash(MACWrite Secret, Seq. No. || Type || Version || Length || Application Data)

Where:

HMACHash = HMAC-SHA-1 or HMAC-MD5

Write Secret = 20 bytes if SHA-1, 16 bytes if MD5. Note that the Mustang HMAC operation uses a precomputed IPAD/OPAD formulation of the Write Secret; that is, the Ipad=hash([WriteSecret]XOR[0x36...]) and opad=hash([WriteSecret]XOR[0x5C...])

SeqNum = 8 byte Sequence Number from Crypto Header

Type = Type byte from SSL Header

Version = 16 bit Version from SSL Header (SSL3.0 = 3,0; TLS1.0 = 3,1)

Length = 16 bit Length of payload (no padding!)

Note: the Length available in the final output packet SSL/TLS header is the length including padding. The Length used in the MAC calculation, however, is the SSL.compressedfragment length, and does NOT include padding.

D.3 MAC in DTLS 1.0

The MAC in DTLS 1.0 is calculated as follows:

HMACHash(MACWrite Secret, Epoch # || Seq. # || Type || Version || Length || Application Data)

Where:

HMACHash = HMAC-SHA-1 or HMAC-MD5

Write Secret = 20 bytes if SHA-1, 16 bytes if MD5. Note that the Mustang HMAC operation uses a precomputed IPAD/OPAD formulation of the Write Secret; that is, the Ipad=hash([WriteSecret]XOR[0x36...]) and opad=hash([WriteSecret]XOR[0x5C...])

Epoch # = 2 byte explicit Epoch Number from DTLS Header (provided by PP)

Seq# = 6 byte explicit Sequence Number from DTLS Header (provided by PP)

Type = Type byte from DTLS Header

Version = 16 bit Version from DTLS Header (DTLS1.0 = 254,255)

Length = 16 bit Length of payload (before encryption, so no padding)

Note: the Length available in the final output packet SSL/TLS header is the length including padding. The Length used in the MAC calculation, however, is the DTLS.compressedfragment length, and DOES not include padding.

D.4 IPAD & OPAD Generation

The pre-computed function is the compressed digest produced by hashing the secure key (K) padded with zeroes to form a 512-bit block exclusive-or'd with the given IPAD value (0x36363636) or OPAD value (0x5c5c5c5c). The hashing function is loaded with the IV (Initial Vector = 0x01234567, 89abcdef, fedcba98, 76543210 for MD5 or = 0x67452301, efcdab89, 98badcfe, 10325476, c3d2e1f0 for SHA-1 or 0x6a09e667, bb67ae85, 3c6ef372, a54ff53a, 510e527f, 9b05688c, 1f83d9ab, 5be0cd19 for SHA-256). The figure below shows the flow of the pre-computed IPAD and OPAD values.

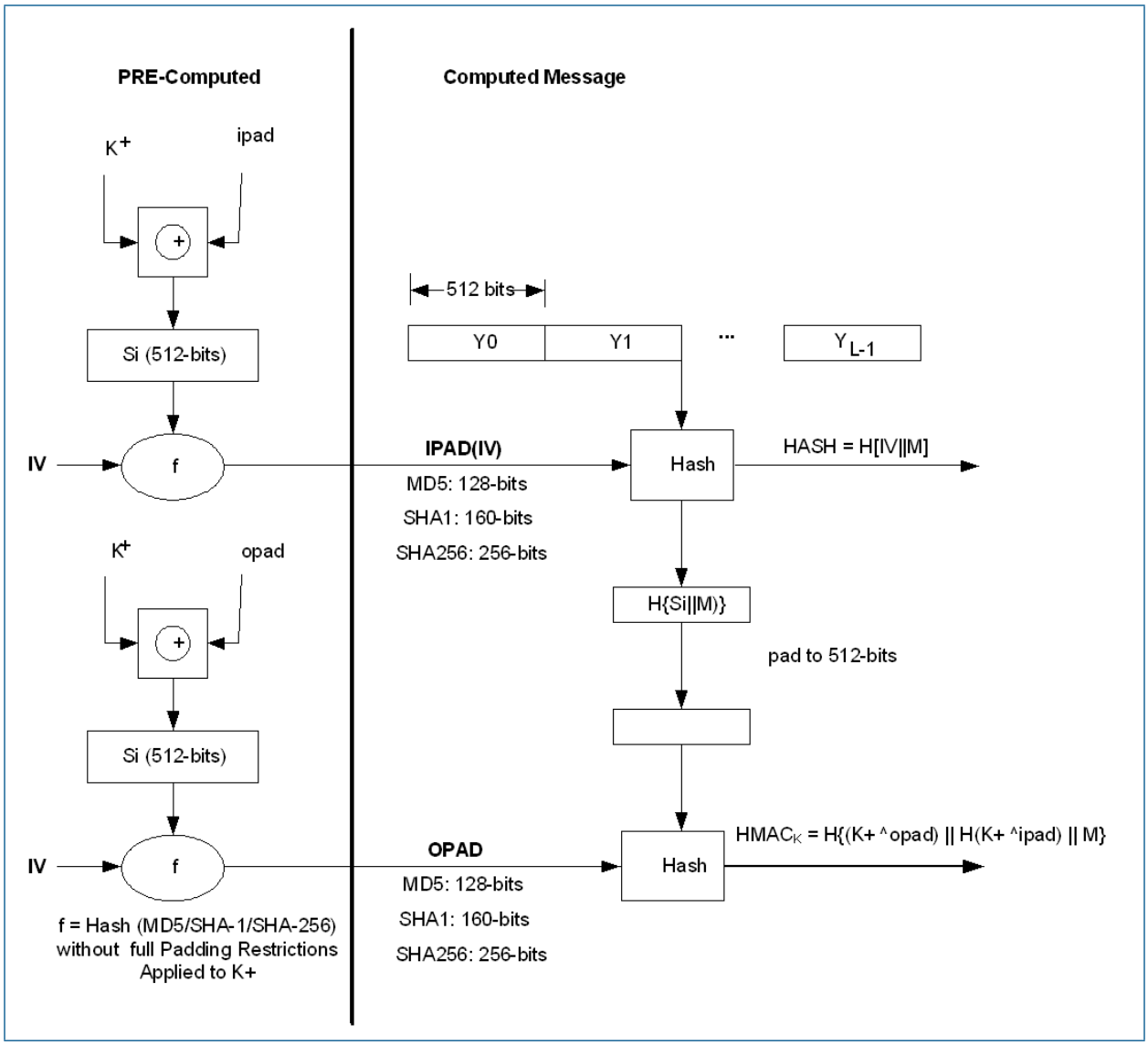


Figure D-1. IPAD and OPAD Generation

Document Revision History

This section lists the additions, deletions, and modifications made to this document for each release of this document.

I.1 Document Revision A

Update 1. Initial release.

I.2 Document Revision B

Update 1. Section 1.2 Part Numbers: added this new section.

Update 2. Chapter 2 Operation: added internal temperature sensor controller to Figure 2-1 and Table 2-1.

Update 3. Section 3.1.3.3 IV and AAD, Figure 3-13 AES IV Lengths: added 3DES-CBC.

Update 4. Section 3.2 Result Ring: updated bits in result ring.

Update 5. Section 5.12 Temperature Sensor Controller: added this new section.

Update 6. Section 6.7 Serial/Parallel Interface regs: removed all references to the external thermal device on the SPI bus. Removed thermal registers.

Update 7. Section 6.8 Temperature Sensor Controller Regs: added this new section.

Update 8. Section 7.1 PCIE Interface: renamed pcie_tx and pcie_rx signals. Added pcie_refclk_p, pcie_refclk_n, pcie_resref.

Update 9. Section 7.2 Misc Interface: removed CM_busy, PKP_busy, 820x_err, clk_gating_disable, phy_refclk_sel, thermal_int_n, and added pll0_ref_clk, pll1_ref_clk.

Update 10. Section 7.6 Test Interface: scan_enable is one bit wide, removed pll_out0 and pll_out1.

Update 11. Section 9.1 Clock Domains: added CPIe_PHY.

Update 12. Section 10.2 Digital Power Supplies: added this new section.

Update 13. Section 10.3 Analog Power Supplies: added this new section.

Update 14. Section 10.4 Power Sequencing: new content for this section.

Update 15. Chapter 11 Thermal Specifications: added thermal resistance and temp correlation parameter.

Update 16. Section 12 Package Specifications: added package drawing, ball map drawing, and alphabetical and numerical ball lists.

Update 17. Moved Sections 10.1 Clock Specification and Section 10.2 Clock Gating to Chapter 2 Operations to be expanded upon later. Added AC and DC Specifications. Replaced mechanical drawing with latest update. Removed comments for TX_LVL and PLL0_REF_CLK signals. Changed VDDA_PLL_0 to VDDA_PLL0 in Table 13-1 alphabetical Ball List. Corrected padding example in Table 3.2. Checked for old pin names in doc.

Update 18. Section 2.2 remove PCIe_PHY and update speed numbers for clock domains. Fixed figure in section 3.2. Section 6.1.1 add bit for Over_heated status and add to diagram. Section 6.7 changed pin labels in figure 6-3. Section 6.7.7 changed flash_status to byte_prog in diagram. Added Spansion device to list

of recommended devices in Section 6.7. Section 6.8 changed parameter "1" to "a". Changed power consumption numbers for VDD_10 in tables 9.8, 9.9, 9.10, 9.11. Removed duplicate thermal resistance table in Chap 12.

I.3 Document Revision C

Update 1. Section 1.2 Ordering Information: added "IB" to all part numbers.

Update 2. Section 1.1.3 Engine Features: removed CCM encryption.

Update 3. Section 3.2 Result Ring: changed bit 28 from OVER_HEAT to Reserved.

Update 4. Section 5.12 Temperature Sensor Controller: updated this section to include flow diagram, examples and to remove automatic temperature detection.

Update 5. Section 6.1.1 Status Register: removed over_heated bit and made reserved.

Update 6. Section 6.1.2 820x Error Register: corrected bit postitions.

Update 7. Section 6.1.3 820x Interrupt Status Register: changed THERM_INT_N to Reserved; changed PKP_INT to Reserved.

Update 8. Section 6.1.4 820x Interrupt Enable Register: changed THERM_INT_N_EN to Reserved; changed PKP_INT_EN to Reserved.

Update 9. Section 6.1.5 Command Completion Interrupt Interval Register: removed this register.

Update 10. Section 6.1.6 Command Completion Interrupt Number Register: removed this register.

Update 11. Section 6.4.2.1 Public Key Command Register Format: added "host must poll this register to know whether the command is finished"; changed IRQ to Reserved.

Update 12. Section 6.5.7 RNG Config Register: added this new register.

Update 13. Section 6.6 GPIO Registers: corrected all GPIO registers to reflect 16-bit GPIO instead of 32-bit.

Update 14. Section 6.8 Temperature Sensor Controller registers: modified this section to remove automatic temp detection.

Update 15. Chapter 11 Thermal Specs: removed ambient operating temperature data from Table 11-1.

Update 16. Section 13 Errata: added errata #2, 3.

I.4 Document Revision D

Update 1. Section 3.1.3.6 Hash Buffer: added alignment restriction.

Update 2. Section 6.1.6 Die Revision Reg: added this new register.

Update 3. Section 8.1 Miscellaneous Interface: updated Figure 8-1.

Update 4. Section 10.5 Power Consumption: updated power values or all 820x devices.

Update 5. Errata: added errata 4.

I.5 Document Revision 00

Update 1. Section 1.1.7 Other Features: added note that Flash interface is not supported by the DX SDK.

Update 2. Section 1.2 Ordering Information: removed industrial 8202, 8203, and 8204 parts.

Update 3. Chapter 2 Operation: added note that SPI interface is not supported by the DX SDK.

Update 4. Section 3.1.2.6 Desc_cmd_pad: added description of how to prevent 820x from hanging if the source and header counts are not set correctly.

Update 5. Section 6.1.5 Soft Reset Register: changed bit 0 from PCIE_RST to Reserved.

Update 6. Section 6.7 Serial/Parallel Interface Registers: added Flash memory map and notes that the Flash is not supported by the DX SDK.

Update 7. Section 7.1.16 Expansion ROM Base Address Register: added notes that the Flash is not supported by the DX SDK.

Update 8. Section 8.2 Miscellaneous Interface: renamed EXT_FLASH_EN to EXT_FLASH_CFG_EN and added a new description.

Update 9. Section 8.4 Flash Interface: added notes that the Flash is not supported by the DX SDK.

Update 10. Section 10.5 Power Consumption: added Table 10-12 listing the industrial power specs for the 8201I device.

Update 11. Section 11.3 Flash Interface Timing: added notes that the Flash is not supported by the DX SDK.

I.6 Document Revision 01

Update 1. Section 1.2 NIST Certificates: added this new section.

Update 2. Section 10.2 Recommended Operating Conditions: added ambient operating temperature for 8201I.

Update 3. Section 3.1.2 Command Descriptor Format: corrected the description for Compression Header Count, Encryption Header Count, Hash Head Count, and Pad Header Count from “double words” to “4-byte words”.

Update 4. Section 3.1.2.3 Desc_cmd_cmp: updated notes for GZIP_MODE, CMP_AG. Changed CMP_AG[1:0]=11 from Reserved to Deflate.

Update 5. Section 3.1.2.7 Desc_srcX and Desc_dstX: removed references to GZIP History.

Update 6. Updated text referenced to GZIP to GZIP/Deflate throughout.

I.7 Document Revision 02

Update 1. Section 1.2 NIST Certificates: corrected AES cert number.

Update 2. Section 6.7.3 SPI Command Address Register: corrected decode values for SPI_CMD. Corrected SPI_ADR only valid when CS=1.

I.8 Document Revision 03

Update 1. Section 6.4.2.3 Public Key Data Register Format: removed reference to future document.

Update 2. Section 13.2 Mechanical Information: updated package drawings to make them more readable.

I.9 Document Revision 04

Update 1. Removed all comments about the SPI interface not being supported by the Express DX SDK. Changed all references to Flash to programmable device.

Update 2. Section 3.1 Command Pointer Ring: clarified that command structures must start on 512-byte boundaries.

Update 3. Section 5.12 Serial to Parallel Interface: created this new section from intro to SPI registers. Added more explanation for how to access the programmable device.

Update 4. Section 6.1.3 820x Interrupt Status Register: changed bit 1 from Reserved to PKP_INT and added warning about errata for this bit.

Update 5. Section 6.2.4 DMA Configuration Register: changed bit [23] from Reserved to DIF_STD.

Update 6. Section 6.2.8 Result Ring 0 Write Pointer Register: clarified that host does not write to this register.

Update 7. Section 6.2.10 Result Ring 1 Write Pointer Register: clarified that host does not write to this register.

Update 8. Section 6.7 SPI Command Configuration 1 Register: added field READ_DATA[7:0] to bits [23:16].

Update 9. Section 7.4.8 Link Control Register: changed default value of CLK_CFG from 1 to 0.

Update 10. Chapter 8 PCI Express Interface: corrected PCIe RX, TX and Ref Clock signal types to HCSL.

Update 11. Section 10.5 Power Consumption: corrected VDDA PLL0 and PLL1 typ and max current. Updated power totals in all tables.

Update 12. Section 11.1 Reset Timing: corrected JTAG_TRSTN conditions for reseting the device.

I.10 Document Revision 05

Update 1. Section 3.1.2 Command Descriptor Format: added note that descriptors for unused engines must be written as zeroes.

Update 2. Section 8.1 PCI Express Interface: removed from PERST_N description that this signal will not reset the PCIe sticky registers. Added note that PERST_N should be connected to POR_N.

Update 3. Section 8.2 Miscellaneous Interface: add text that POR_N should be tied to PERST_N.

Update 4. Section 8.5 JTAG Interface: added text that JTAG_RST_N can be tied to PERST_N.

Update 5. Section 11.1 Reset Timing: added text to tie PERST_N, POR_N and JTAG_TRST all to PCIe reset.

Update 6. Section 11.2 PLL Clock Input: added note that performance scales with input clock frequency.

Update 7. Section 12 Thermal Specifications: added ambient temp for 8201I part.

48720 Kato Road
Fremont, CA 94538
p: 510.668.7000
www.exar.com