PRODUCTION SPECIFICATION

BCM5802

Single-Chip Security Processor

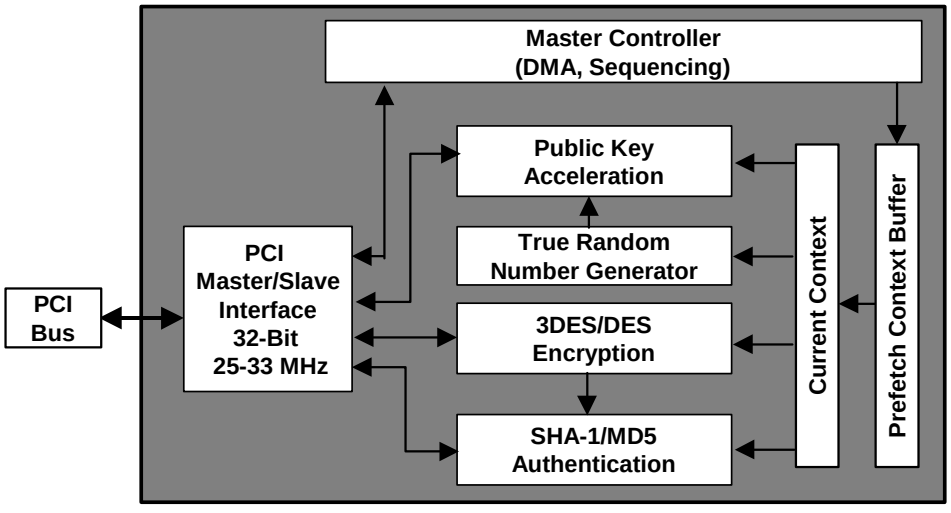
GENERAL DESCRIPTION

The BCM5802 Security Processor provides industry-standard IETF IPsec encryption and authentication acceleration as well as IKE/SSL/TLS key setup acceleration. Engine throughput is over 150 Mbps with 3DES strong encryption and MD5/SHA1 authentication enabled. Sustained in-system performance with all features enabled ranges up to 100 Mbps for crypto/authentication acceleration and 30 1024-bit Diffie-Hellman (180-bit exponent) key setups per second. The BCM5802 is ideal for cost-sensitive devices, including cable modem access systems, xDSL access systems, T1/T3 line security, and 10/100 Mbps ethernet interfaces.

The BCM5802 includes a built-in PCI 2.2 compliant interface for easy hardware interfacing. It requires zero external support components, enabling tremendous system cost savings, and it features a streamlined high-performance programming model for easy software integration.

FEATURES

- High-performance, low-cost security processor integrating full IPsec acceleration
- Supports DES, 3DES, HMAC-SHA1 and HMAC-MD5
- 100 Mbps IPsec (3DES, SHA1) in-system performance, with new Security Association (SA) per packet
- Unlimited SA support via system memory
- Extensive hardware support for IKE/SSL/TLS key setup acceleration
- Public key acceleration unit supports over 30 Diffie-Hellman key exchanges per second
- Compatible with SSH IPsec and IKE software
- True hardware random number generator
- Supports multi-packet processing and pre-fetch of packet data and context
- Aggressive pre-fetch DMA allows multi-packet, multi-threaded, DMA processing with single PCI writes
- Full performance maintained independent of any reasonable PCI latency
- PCI 2.2 interface, 32-bit, 33 MHz
- Low-power 3.3V design in 0.35µ CMOS technology
- 144-pin DQFP package



Functional Block Diagram

16215 Alton Parkway • P.O. Box 57013 • Irvine, California 92619-7013 • Phone: 949-450-8700 • Fax: 949-450-8710

REVISION HISTORY

Revision #	*Date*	*Change Description*
5802-DS00-R	09-25-00	Initial release.
5802-DS01-R	11-15-00	Added lead pitch and lead width dimensions to package dimensions table.
5802-DS02-R	07-27-01	• Made text changes in “Pin Definitions” table. • Made text changes in “Overview of Software Interface.” • Added two new bullets under “Invalid Encryption/Authentication Operations.” • Updated “PCI Configuration Registers” table. • Updated “DMA Control and Status Registers” table. • Updated and added items under “Electrical and Timing Specifications” section.
5802-DS03-R	07-03-02	Changed access for bits 24:23 in Table 29 on page 37.

Broadcom Corporation
P.O. Box 57013
16215 Alton Parkway
Irvine, California 92619-7013

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Section 1: Functional Description

OVERVIEW

This document describes the BCM5802 security processor. The BCM5802 provides high-performance, low-cost IPsec/IKE/SSL/TLS security. The device is especially attractive for high-volume, cost-sensitive access products and telecommuter solutions running over xDSL, cable modem, T1 line, T3 line, and 10/100 Mb ethernet interfaces.

KEY FEATURES AND STATISTICS

The feature set of the BCM5802 is optimized to enable cryptographic acceleration for protocols such as IPsec and IKE/SSL/TLS acceleration. High in-system performance, low system cost and ease of software development are key goals of the BCM5802. The following table lists the key features and statistics of the BCM5802.

Table 1: BCM5802 Key Features and Statistics

Supply	3.3V supply, 3.3V-driven, and 5V-tolerant I/O.
Engine throughput, 3DES + MD5/SHA1	>150 Mbps, all features on.
System throughput, 3DES+MD5/SHA1	100 Mbps.
System throughput, DH (1024b Mod, 180b Exp)	30 key setup/s.
System throughput, DSA (1024b public key, 160b private key)	50 signing/s and 25 verification/s.
System throughput, 1024-bit RSA	20 private key operation/s.
External memory usage	No additional memory required.
External clock supply	No additional clock required. The chip is driven by PCI clock.
External bus	PCI 2.2, 25-33 MHz, 32-bit, 3.3V, and 5V.
Package	144-pin DQFP.
Technology	0.35 µm, 5LM standard-cell logic process.

STREAMLINED HARDWARE INTERFACE

- Direct connect to 32-bit PCI 2.2 bus running at 25-33 MHz, 3.3V, or 5V PCI
- Zero external components: no external memory, no clock chips/oscillators, no EEPROM
- Ideally suited for a shared PCI bus: latency-tolerant design, programmable burst size

IETF IPSEC COMPLIANT ACCELERATION

- 3DES CBC encryption and decryption in accordance with FIPS 46-3 and FIPS 81.
- HMAC-MD5-96 and HMAC-SHA1-96 authentication in accordance with RFC2403, RFC2404 and FIPS 180-1. Automatic generation of MD5/SHA1 padding.
- Single-pass encryption and authentication via pipelined application of algorithms over payload in accordance with RFC2402 and RFC2406.
- Automatic sequencing of encryption and authentication: Encrypt first for outbound packets, authenticate first for inbound packets in accordance with RFC2401.

IETF IKE

- 768-bit and 1024-bit Diffie-Hellman key generations for IKE handshake according to RFC2409
- 512-bit, 768-bit and 1024-bit RSA signing and verification for IKE handshake
- 1024-bit DSA signing and verification for IKE handshake according to FIPS 186-2
- True random number generation for IKE keys using on-chip random number generator

SECURE SOCKET LAYER (SSL) V 3.0, TRANSPORT LAYER SECURITY (TLS)

- 512-bit, 768-bit, and 1024-bit RSA public key and private key processing
- 512-bit, 768-bit, and 1024-bit Diffe-Hellman session key generation
- DES and Triple-DES bulk encryption capability
- 1024-bit DSA signing and verification
- HMAC-MD5/SHA1 bulk authentication according to RFC2104

STREAMLINED, FLEXIBLE SOFTWARE COMMAND AND PACKET INTERFACE

- Flexible command interface allows exchange of multiple packets or public key setups with one PCI write
- Zero latency command buffer switching via double-buffered master command register
- Support for big and little endian environments
- Host CPU intervention not required between packets or between key setups
- Intelligent, autonomous DMA descriptor based interface to minimize software load
- Scatter/Gather support to eliminate packet data or key setup data copying–handles fragmented data directly
- Support for any number of IPsec security association contexts, limited only by system memory

ADDITIONAL PERFORMANCE ENHANCING FEATURES

- Latency-tolerant design optimized for shared PCI bus environments. The BCM5802 leverages PCI burst mode access capability, up to a maximal burst size of 64 bytes.
- Aggressive pre-fetch of command and packet data.
- Full performance is maintained independent of any reasonable PCI latency.

ADVANCED TESTABILITY FEATURES

- 100% testability of on-chip RAM cells via BIST circuitry
- JTAG boundary scan for board-level testing

BCM5802 ADDITIONAL FEATURES TO BCM5801

The BCM5802 adds a number of features as compared to the BCM5801. The notable additional features are:

- Diffie-Hellman, RSA, and DSA key setup execution unit to accelerate the public key operations.
- True random number generator (RNG) functional unit to generate secure private keys for Diffie-Hellman key exchanges and DSA signatures.
- 1024-bit register files to hold the large public key data.
- The BCM5802 is completely pin and register compatible with BCM5801, and is completely backwards register compatible with the BCM5801.

OPTIMAL APPLICATION AREAS

The BCM5802 enables high-speed security support for a variety of cost-sensitive applications and markets, including no compromise VPN support, secure telecommuting and remote access. Specific applications areas are as follows:

- Secure telecommuting and SOHO access devices based on cable or xDSL modem
- Secure enterprise T1 and T3 access devices
- Secure LAN access devices
- PC-based VPN accelerator boards

PROCESSING OVERVIEW

The BCM5802 security processor manages IPsec packets in the following stages:

1. Fetch command context and data via descriptors.
2. If packet is inbound, authenticate then decrypt in pipelined fashion.
3. If packet is outbound, encrypt then authenticate in pipelined fashion.
4. Write (via descriptors) output data and authentication codes if applicable.

The command, data descriptor, packet data and context data fetch phases are completely overlapped with engine processing. Output packet data writeback is completely overlapped as well.

The following figure illustrates a high-level view of the BCM5802 packet processing.

Note **Multiple sets of input packets can be specified via a single command descriptor (single PCI write).**

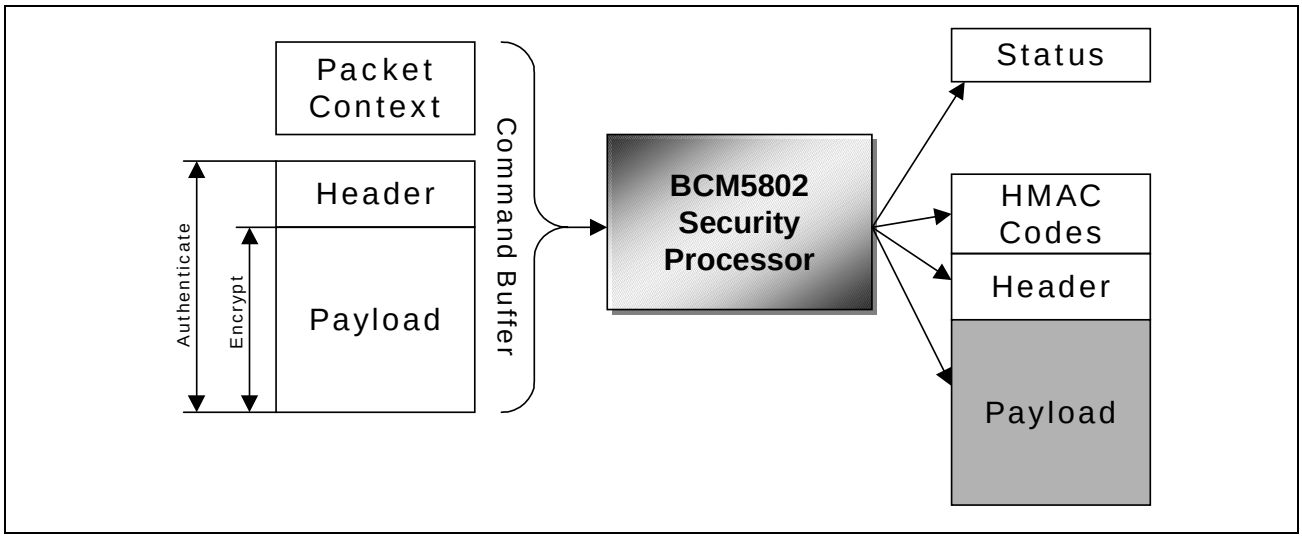


Figure 1: Packet Processing Overview

The BCM5802 provides SSL/TLS key exchange using RSA in the following stages:

1. Fetch command context, including keys and message through DMA.
2. If the required operation is private key decryption, use the private key RSA algorithm with pre-computed components generated using the Chinese Remainder Theorem.
3. If the required operation is public key encryption, use the public RSA algorithm.
4. Write the decrypted/encrypted message to the output buffer.

The BCM5802 generates keys using the Diffie-Hellman algorithm for IKE handshake in the following stages:

1. Fetch command context and message through DMA.
2. If the required operation is to generate a message to another party (g^x mod n), generate a random number from the random number generator unit on the chip and then perform the modular exponentiation with the generated random number as the exponent on the chip.
3. If the required operation is to generate the shared key from the message received (Y^x mod n), perform the modular exponentiation with a previously generated random number on the chip. The random number is a part of the command context through DMA.
4. Write the output including the generated random number to the output buffer.

The BCM5802 performs authentication using the DSA algorithm for an IPsec session during IKE handshake in the following stages:

1. Fetch command context and message through DMA.
2. If the required operation is to sign message, generate a random number and compute *r* and *s* values using SHA-1 and key setup execution units.
3. If the required operation is to verify signature, compute *v* value using SHA-1 and key setup execution units.
4. Write the output to the output buffer.

Section 2: Hardware System Interface and Performance

APPLICATION EXAMPLES

The BCM5802 is ideally suited for cost-sensitive applications such as VPN appliances, SOHO routers and appliances, and IPsec acceleration. The following figure illustrates a system architecture concept that integrates the BCM5802 as a VPN accelerator. This architecture allows wire-speed support of secure VPN for a minimal incremental system cost.

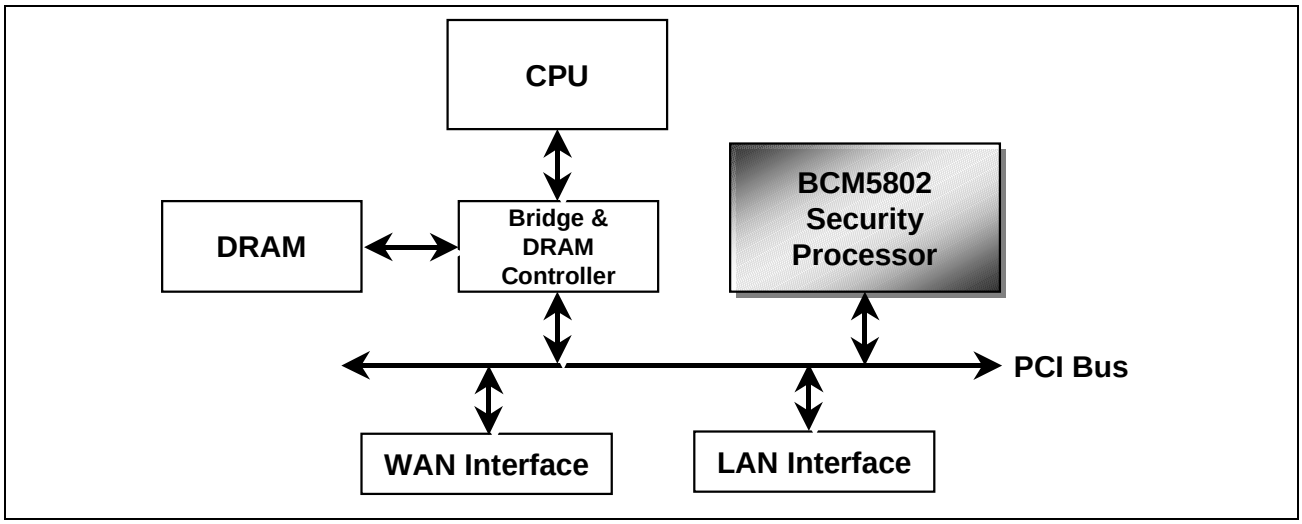


Figure 2: Architecture Concept

The BCM5802 enables very low-cost PCI-based cards that can accelerate IPsec processing up to T3 rate. The following figure shows the architecture of a BCM5802-based accelerator card. The accelerator card also provides key setup acceleration on the chip as well as a hardware random number generator to generate secret keys.

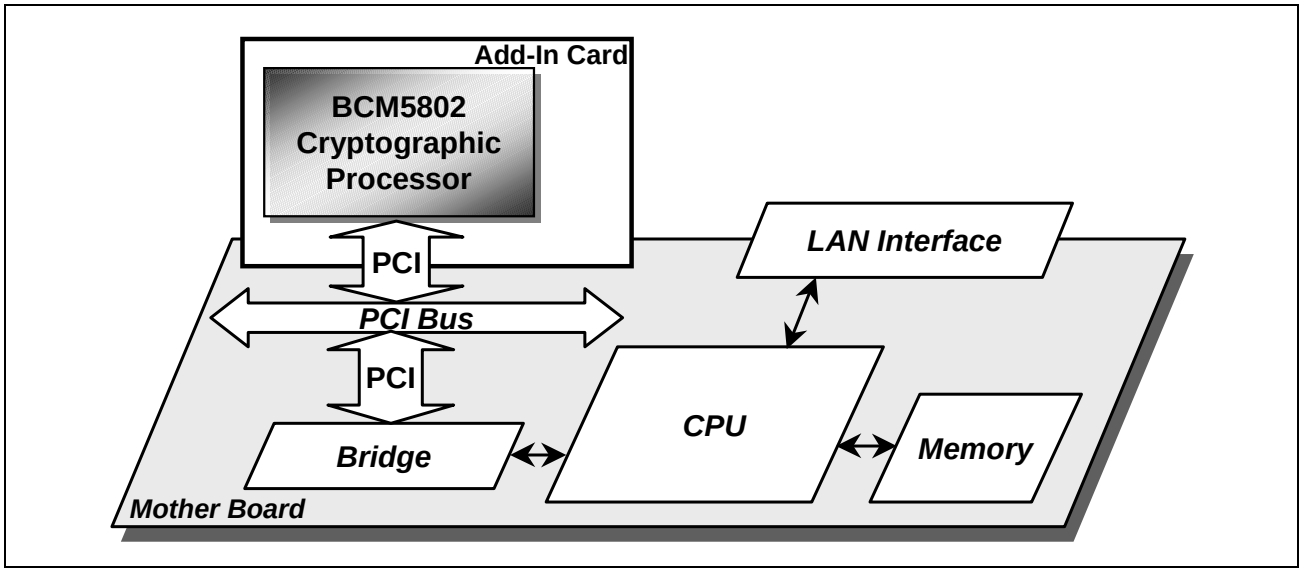


Figure 3: PCI IPsec Accelerator Board - Architecture Concept

HARDWARE INTERFACE

The only interface to the BCM5802 is a 32-bit PCI 2.2-compliant bus and a clock input. The following sections describe the key features of the hardware interface.

Support for Both PCI 3.3V and PCI 5V Signaling Environments

Single supply voltage of 3.3V ±5%. Because I/O pins for the BCM5802 are 5V tolerant, the BCM5802 can be used in both PCI 3.3V and PCI 5V environments.

Latency Tolerant Design

Descriptor for command as well as data buffers are pre-fetched to reduce the impact of PCI arbitration and system latency upon overall performance. Large burst sizes (up to a maximum of 64 bytes) are used when possible to fetch descriptor, command and packet payload data. Command context data is pre-fetched. Payload data is also pre-fetched and written back in posted fashion.

Support for PCI Clock Rates from 25-33 MHz

PCI clock rates from 25-33 MHz are supported. In general, lower clock rates and higher PCI system latencies have little impact on system performance, owing to aggressive data pre-fetch.

IN-SYSTEM PERFORMANCE ANALYSIS

PCI bus clock and latency have little effect on total BCM5802 system performance. This is because the chip aggressively pre-fetches and writes back descriptors, command buffers, context parameters and packet data. This aggressive pre-fetch enables the chip to run encryption and authentication engines at their full potential despite system latencies. Standard shared PCI bus implementations that run at 20-33 MHz with per-access latencies in the range of 1 ms to 1.5 ms enable the BCM5802 to run at full speed.

The chip core clock rate has a major impact on performance. Broadcom recommends that the BCM5802 be clocked at 33 MHz, which is the high end of the core clock frequency, in systems where maximal performance is desired. The chip core clock can be either directly copied from the PCI clock for reduced system cost, or generated asynchronously via an external oscillator for maximal performance.

The values shown in the following table indicate outbound packet Mbps performance for 3DES, HMAC-SHA1, with new the Security Association per packet.

Table 2: Performance Table (Mbits/second)

PCI Clock Frequency	*Packet Sizes (Bytes)*				
	64	*256*	*512*	*1024*	*2048*
33 MHz	28	67	89	104	113

Section 3: Hardware Signal Definition Table

The BCM5802 is housed within a 144-pin DQFP package with a 28 mm x 28 mm body size. The pin definitions are shown in the following table.

Table 3: PCI Interface Pin Definitions

Name	*I/O*	*Pin #*	*Description*
AD[31]	IO	20	PCI multiplexed address/data bus.
AD[30]	IO	21	PCI multiplexed address/data bus.
AD[29]	IO	23	PCI multiplexed address/data bus.
AD[28]	IO	24	PCI multiplexed address/data bus.
AD[27]	IO	25	PCI multiplexed address/data bus.
AD[26]	IO	27	PCI multiplexed address/data bus.
AD[25]	IO	28	PCI multiplexed address/data bus.
AD[24]	IO	29	PCI multiplexed address/data bus.
AD[23]	IO	33	PCI multiplexed address/data bus.
AD[22]	IO	35	PCI multiplexed address/data bus.
AD[21]	IO	36	PCI multiplexed address/data bus.
AD[20]	IO	37	PCI multiplexed address/data bus.
AD[19]	IO	38	PCI multiplexed address/data bus.
AD[18]	IO	39	PCI multiplexed address/data bus.
AD[17]	IO	41	PCI multiplexed address/data bus.
AD[16]	IO	42	PCI multiplexed address/data bus.
AD[15]	IO	59	PCI multiplexed address/data bus.
AD[14]	IO	60	PCI multiplexed address/data bus.
AD[13]	IO	62	PCI multiplexed address/data bus.
AD[12]	IO	63	PCI multiplexed address/data bus.
AD[11]	IO	65	PCI multiplexed address/data bus.
AD[10]	IO	66	PCI multiplexed address/data bus.
AD[9]	IO	67	PCI multiplexed address/data bus.
AD[8]	IO	68	PCI multiplexed address/data bus.
AD[7]	IO	71	PCI multiplexed address/data bus.
AD[6]	IO	72	PCI multiplexed address/data bus.
AD[5]	IO	73	PCI multiplexed address/data bus.
AD[4]	IO	75	PCI multiplexed address/data bus.
AD[3]	IO	76	PCI multiplexed address/data bus.
AD[2]	IO	77	PCI multiplexed address/data bus.
AD[1]	IO	79	PCI multiplexed address/data bus.

Table 3: PCI Interface Pin Definitions

Name	I/O	Pin #	Description
AD[0]	IO	80	PCI multiplexed address/data bus.
PCI_CLK	I	8	PCI clock, 25-33 MHz.
GNT#	I	17	PCI bus grant allowing the chip to use the bus.
FRAME#	IO	45	PCI frame, indicates the beginning and duration of a master transfer.
IRDY#	IO	46	PCI initiator ready.
TRDY#	IO	47	PCI target ready.
DEVSEL#	IO	49	PCI device select, asserted by an access target.
STOP#	IO	50	PCI stop, requesting that the current master stop an active transfer.
PERR#	IO	53	PCI parity error.
SERR#	IO	54	PCI system error, open drain.
PAR	IO	55	PCI parity.
REQ#	O	19	PCI bus request.
RESET#	I	16	PCI reset, tri-states all PCI outputs.
INT#	O	15	PCI interrupt output, open drain.
IDSEL	I	32	PCI Initialization Device Request, used for PCI configuration cycles.
CBE#[3]	IO	31	PCI command/byte enable, provides PCI bus command and data byte enables.
CBE#[2]	IO	43	PCI command/byte enable, provides PCI bus command and data byte enables.
CBE#[1]	IO	58	PCI command/byte enable, provides PCI bus command and data byte enables.
CBE#[0]	IO	70	PCI command/byte enable, provides PCI bus command and data byte enables.
VCC	I	51	Must be pulled up to VCC (PCI LOCK_).
VCC	Power pins, must be connected to a 3.3V source: 10, 18, 26, 40, 48, 57, 61, 74, 81, 90, 92, 93, 102, 103, 109, 110, 126, 127, 133, 134, 135, 143, 144.		
GND	Ground pins: 5, 12, 14, 22, 30, 34, 44, 52, 56, 64, 69, 78, 84, 85, 89, 99, 100, 106, 107, 108, 116, 117, 118, 119, 122, 123, 136, 137, 139, 140.		
AVCC1	I	94	Analog VCC for 4x PLL. Connect to 3.3V.
AGND1	I	98	Analog ground for 4x PLL.
AVCC2	I	9	Analog VCC for deskew PLL. Connect to 3.3V.
AGND2	I	7	Analog ground for deskew PLL.
VIO	I	111	PCI clamp voltage bias. Connect to 3.3V for 3.3V signaling environments. Connect to 5V for 5V signaling environments.
EXPORT	I	138	EXPORT pin (high = 56-bit encryption; low = strong encryption). Internally pulled up.
TEST	I	1	Test pin, internally pulled down, should be grounded for regular operation. When TEST is high, all outputs are tri-stated.
TRST#	I	131	Internally pulled up. Should be connected to ground for normal operation. Used for boundary scan JTAG testing.
TMS	I	120	Test mode select for JTAG boundary scan. Internally pulled up. Should be connected to VCC for normal operation.
TCK	I	6	Test mode clock for JTAG boundary scan. Internally pulled up. Unused in normal operation; connect to either high or low static level.

Table 3: PCI Interface Pin Definitions

Name	I/O	Pin #	Description
TDI	I	13	Test data in for JTAG boundary scan. Internally pulled up. Unused in normal operation; connect to either high or low static level.
TDO	O	121	Test data out for JTAG boundary scan. Unused in normal operation.
RNGOSC	I	113	Optional random number generator oscillator. Internally grounded. It can be Ex-ORed with internal oscillator to provide random number source.
Don't Connect	The pins used for product testability and not used by customers. Leave them unconnected: 2, 3, 11, 82, 83, 86, 87, 91, 112, 114, 141.		
All other pins are No Connects, and can be left floating or connected.			

PINOUT DIAGRAM

The following figure shows the BCM5802 pin diagram.

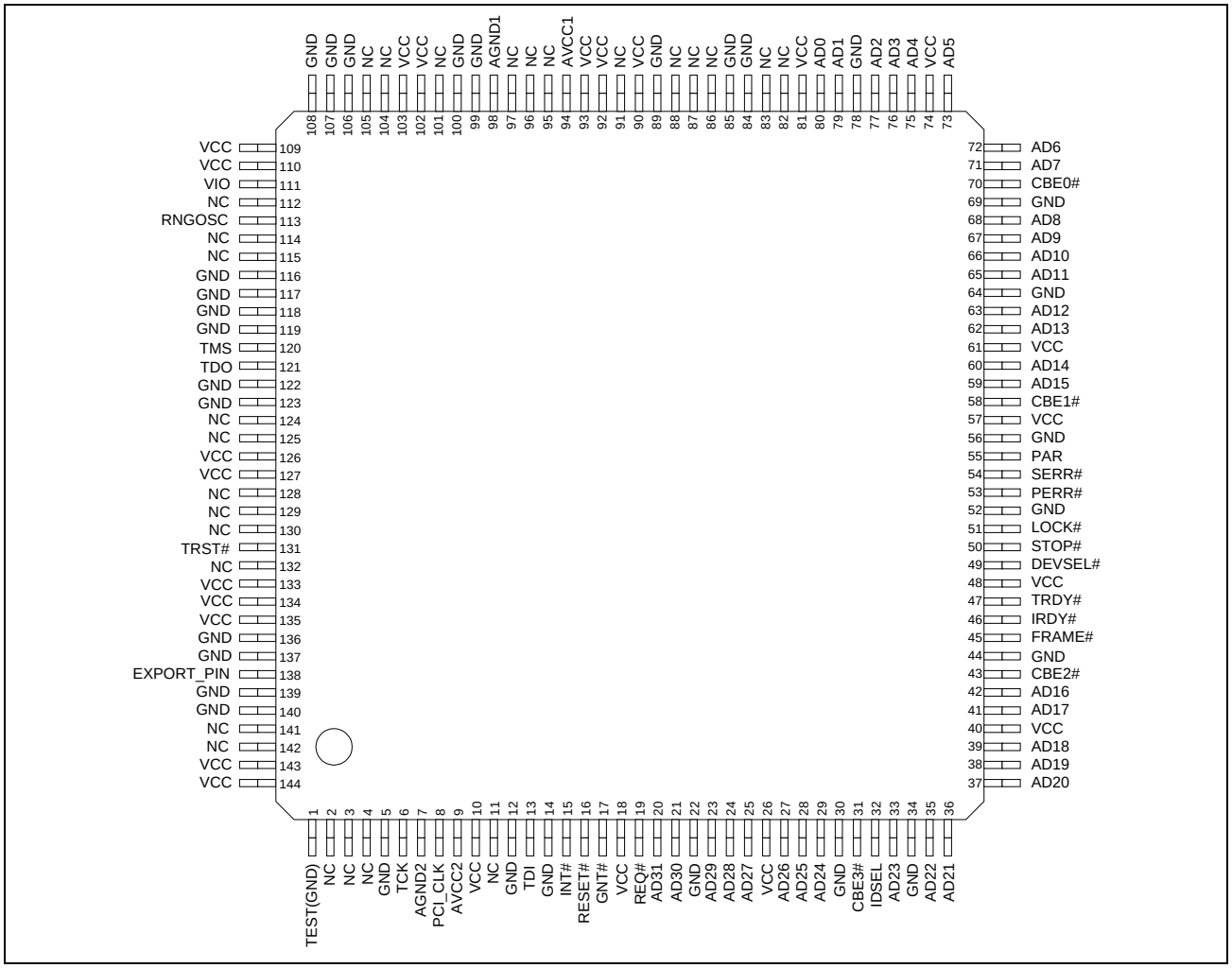


Figure 4: BCM5802 Pin Diagram

Section 4: Software Programming Model

This section specifies the programming model of the BCM5802, shows a sample software processing loop, and provides detailed descriptions of the on-chip registers.

Overview of Software Interface

The major features of the BCM5802 software interface are as follows:

- Autonomous chip operation via an intelligent, descriptor-based DMA interface that minimizes the software processing load.
- Avoid packet or key setup data copying under any condition.
- Supports input packet fragmentation (at an IP level as well as in terms of memory allocation for packet data). Input fragments can be of any size (down to 1 byte), and can be aligned on any byte boundary.
- Supports output packet fragmentation (at an IP level as well as in terms of memory allocation for packet data). Output fragment size can be controlled in one of two configurable ways: 1) through a length field with each output data descriptor, or 2) through a global output data buffer length field. This offers the flexibility of using a fixed output fragment size, or of setting fragment size on a per-packet basis. Output fragments must be aligned on 32-bit word boundaries, and must be multiples of a 32-bit word in size.
- Permits flexibility with respect to the granularity of communication between the CPU and the chip. The CPU can instruct the chip to process several packets or key setups via a single PCI write. This allows the host CPU to select the degree of overlap between software and chip processing–one packet or key setup, several packets or key setups, or a very large number of packets or key setups.
- Permits different security processing to be applied to each and every packet or key setup, even though several packets or key setups may be part of a common master command structure.
- Flexible support for all IPsec formats, including ESP, AH and combinations with and without tunneling
- Flexible support for IKE, SSL, and TLS protocols, including DH, RSA, and DSA algorithms

The host CPU queues up any number of packets or key setups in system memory, and passes a pointer to a master command structure that identifies these packets or key setups to the chip. After the chip processes all the packets or key setups as specified, it then returns status to the CPU via a done flag per packet, and if enabled, via an interrupt upon global completion of all packets or all key setups within a master command structure.

A processing context structure is associated with each packet/key setup that allows various packets/key setups to be processed differently even though they are all part of a common master command structure. In addition, data from each packet can be fragmented on input (gather function) and on output (scatter function) in the IPsec crypto/authentication operations.

While there are no data buffer alignment constraints (such as byte alignment only), there are specific constraints upon command and context structure alignment as detailed under memory structures.

The following figure shows an overview of the various structures and linkages used to forward packet/key setup data to the chip. Fields indicated by an @ sign correspond to pointers. The # PKT field in the master command structure allows up to 2^{16}-1 packets to be queued up for processing (the high order 16 bits of this field are not used). The output fields within each entry in a master command buffer specify the start of a buffer chain into which output (encrypted or decrypted) data is written.

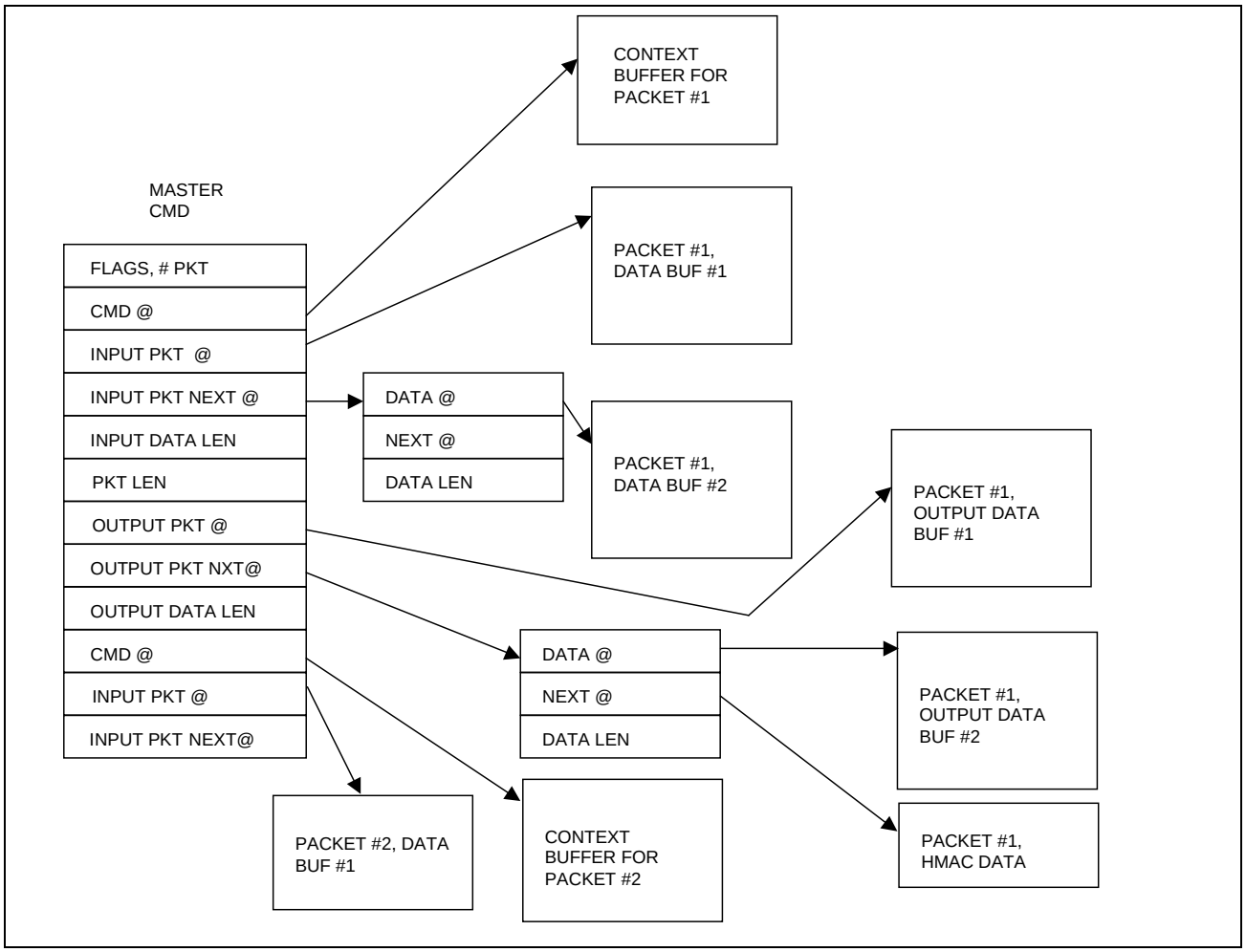


Figure 5: Structures and Linkages Used to Forward Packet/Key Setup Data to Chip

The master command structure is a single point of communication between the host CPU and the chip. Chip processing of any number of packets is initiated by writing the address of a master command structure to the on-chip master command address register (Master Command Register 1). The chip signals completion of processing by writing status information to the flags entry at the beginning of the master command structure and by posting an interrupt per master command structure (if enabled).

Note

The NEXT@ field of the last output data buffer pointer is never used to access data for IPsec crypto/authentication operations. This field instead contains the address of a buffer to which HMAC information is written to or read from, if HMAC processing is specified for a given packet. For HMAC-MD5, the entire 16 bytes of hash result is written to the buffer. For HMAC-SHA1, the entire 20 bytes of hash result is written to the buffer. For the IPsec HMAC-96, the software must discard the last four bytes of the data for HMAC-MD5 and the last eight bytes of the data for HMAC-SHA1.

For key setup operations, the same MCR structure is used as for IPsec crypto/authentication operations. The only difference is that chip processing of any number of key setups is initiated by writing the address of a master command structure to a different on-chip master command address register (Master Command Register 2). Both operations still share DMA Control Register, Status Register, and Error Address Register.

MEMORY STRUCTURES

All structures used for communication between the CPU and the chip are defined by their .h pseudo-code C language representation.

For IPsec crypto/authentication processing, the only alignment restriction placed upon all command and descriptor (not packet data) memory structures is that they must start on 32-bit (4-byte) boundaries. Beyond that, aligning structures to their natural boundaries may increase performance in certain systems.

IPSEC CRYPTO/AUTHENTICATION PROCESSING DATA STRUCTURE

```
-------------------------------------------------------------------------------------------
/* LITTLE ENDIAN command structures for uBSec Chip */
typedef unsigned char u8; /* 8-bit data type */
typedef unsigned short u16; /* 16-bit data type */

/* Data Buffer chain entry */
typedef struct DataBufChain_struct {
    unsigned char *dataAddr;
    struct DataBufChain_struct *next;
    u16 dataLength;
    u16 reserved;
} DataBufChain;

/* Context buffer */
typedef struct PktCtxBuf_struct {
    /* Keys for 3DES -- three keys of 8 bytes each (56 bits plus parity) */
    uint cryptokeys[6];
    /*
        * Pre-computed HMAC inner & outer state
        * (2x16B for MD5, 2x20B for SHA1).
    */
        uint HMACInnerState[5];//HMACInnerState[0-3] for MD5, HMACInnerState[0-4] for SHA1
        uint HMACOuterState[5];//HMACOuterState[0-3] for MD5, HMACOuterState[0-4] for SHA1
        /*
            * Crypto IV (copied from payload if explicit, byte swapped if needed)
        */
        uint computedIV[2];
        /*
        * Processing control flags

        */
        unsigned int reserved:12; /* Reserved */
        unsigned int auth:2; /* MD5, SHA1, None */
        unsigned int inbound:1; /* Inbound packet */
        unsigned int crypto:1; /* 3DES-CBC or None */
            /* Offset to skip authenticated but non-encrypted
            header words. Goes to start of IV data. In units of 32-bit words */
        u16 cryptoOffset;
```

```
} PktCtxBuf;
/* Master command record */
typedef struct MasterCmd_struct {
       u16 numPkt; /* Number of Packets in this MCR*/
       u16 flags; /* Completion and error status from chip, per MCR */
          /* flags[0] = 1 if processing of the MCR is finished
                 0 otherwise
             flags[1] = 1 if an error occurred
                 0 if no error occurred
             flags[7:2]: reserved
             flags[15:8] = error code if an error occurred (i.e. flags[1] == 1),
                 undefined otherwise*/
       /* Following 5 fields occur once per packet in the MCR */
       uint firstPktCMDAddr;
       DataBufChain firstPktData; /* First descriptor for input packet data */
       u16 reserved; /* Includes per packet done status */
       u16 pktLength;
       DataBufChain firstOutputData; /* First descriptor for output packet data */
       /* Followed by as many sets of above 5 fields as there
       are packets in this MCR */
} MasterCmd;
```

An implicit (pre-computed) IV is never used as part of the HMAC computation—even if specified. However, an explicit IV is always part of the authentication computation. Further details regarding IV material handling follow the pictorial illustration of the packet context structure.

The following is the data structure (.h file) for key setup processing.

IKE/SSL/TLS KEY SETUP PROCESSING DATA STRUCTURE

```
-------------------------------------------------------------------------------------------
/* LITTLE ENDIAN command structures for uBSec Chip */
typedef unsigned char u8; /* 8-bit data type */
typedef unsigned short u16; /* 16-bit data type */
typedef unsigned int u32; /* 32-bit data type */

/* Data Buffer chain entry */
typedef struct DataBufChain_struct {
   unsigned char *dataAddr;
   struct DataBufChain_struct *next;
   u16 dataLength;
   u16 reserved;
} DataBufChain;

/* Context buffer */
/* Different algorithms have different command context buffers */

/*Diffie-Hellman Send*/
typedef struct DH_SEND_CtxCmdBuf_struct {
   u16 total_command_structure_length;
   u16 operation_type; /* Send mode for DH (0x1) */
   u16 rng_enable; /* Private key x generated by RNG or provided by SW
          rng_enable = 0x0 -> x provided by SW
          rng_enable = 0x1 -> x generated by RNG */
      u16 private_key_length; /* Private key x length in bits*/
      u16 generator_length; /*Generator g length in bits*/
      u16 modulus_length; /* Modulus N Length in bits */
u32 N[(modulus_length <= 512)? 16 : (modulus_length <= 768)? 24 : 32]; /* Modulus N
*/
      u32 g[(modulus_length <= 512)? 16 : (modulus_length <= 768)? 24 : 32]; /* Generator
g */
          /* Private key is stored in the data buffer */
} DH_SEND_CtxCmdBuf;

/*Diffie-Hellman Receive*/
typedef struct DH_REC_CtxCmdBuf_struct {
   u16 total_command_structure_length;
   u16 operation_type; /* Receive mode for DH (0x2) */
   u16 exponent_length; /* Exponent (private key x) length in bits */
   u16 modulus_length; /* Modulus N Length in bits */

   u32 N[(modulus_length <= 512)? 16 : (modulus_length <= 768)? 24 : 32]; /* Modulus N */
} DH_REC_CtxCmdBuf;

/*Public Key RSA*/
typedef struct Pub_RSA_CtxCmdBuf_struct {
   u16 total_command_structure_length;
   u16 operation_type; /* Public mode for RSA (0x3) */
   u16 exponent_length; /* Exponent E length in bits*/
   u16 modulus_length; /* Modulus N Length in bits */
   u32 N[modulus_length <= 512)? 16 : (modulus_length <= 768)? 24 : 32]; /*
Modulus N */
   u32 E [exponent_length + 31)/32]; /* Exponent E */
} Pub_RSA_CtxCmdBuf:

/*Private Key RSA*/
typedef struct Pri_RSA_CtxCmdBuf_struct {
   u16 total_command_structure_length;
```

```
    u16 operation_type; /* Private mode for RSA (0x4) */
    u16 q_length; /* Prime q length in bits */
    u16 p_length; /* Prime p Length in bits */

    u32 p[max_length <= 256 ? 8 : max_length <= 384 ? 12 : 16]; /* Prime p */
    u32 q[max_length <= 256 ? 8 : max_length <= 384 ? 12 : 16]; /* Prime q */
    u32 dp[max_length <= 256 ? 8 : max_length <= 384 ? 12 : 16];/* CRT private exponent dp
    */
    u32 dp[max_length <= 256 ? 8 : max_length <= 384 ? 12 : 16];/* CRT private exponent dq
    */
    u32 pinv[max_length <= 256 ? 8 : max_length <= 384 ? 12 : 16]; /* CRT coefficient */

} Pri_RSA_CtxCmdBuf;

where max_length = (p_length > q_length) ? p_length : q_length;

/*DSA signing */
typedef struct DSA_SIGN_CtxCmdBuf_struct {
    u16 total_command_structure_length;
    u16 operation_type; /* Signing mode for DSA (0x5) */

    u16 sha1_enable; /* hash of message performed by SHA1 unit or provided by SW
        sha1_enable = 0x0 -> hash provided by SW
        sha1_enable = 0x1 -> hash performed by SHA1 unit */
    u16 reserved;

    u16 rng_enable; /* Random number k generated by RNG or provided by SW
        rng_enable = 0x0 -> k provided by SW
        rng_enable = 0x1 -> k generated by RNG */

        u16 p_length; /* Modulus p length in bits */

        u32 q[5]; /* Modulus q */
        u32 p[(p_length <= 512)? 16 : (p_length <= 768)? 24 : 32]; /* Modulus p */
        u32 g[(p_length <= 512)? 16 : (p_length <= 768)? 24 : 32]; /* Generator g */
        u32 x[5]; /* Private key x */
} DSA_SIGN_CtxCmdBuf;

/*DSA Verification */
typedef struct DSA_VERIFY_CtxCmdBuf_struct {
    u16 total_command_structure_length;
    u16 operation_type; /* Verification mode for DSA (0x6)*/
    u16 sha1_enable; /* hash of message performed by SHA1 unit or provided by SW
        sha1_enable = 0x0 -> hash provided by SW
        sha1_enable = 0x1 -> hash performed by SHA1 unit */
    u16 reserved;
    u16 reserved;
    u16 p_length; /* Modulus p length in bits */

    u32 q[5]; /* Modulus q */
    u32 p[(p_length <= 512)? 16 : (p_length <= 768)? 24 : 32]; /* Modulus p */
    u32 g[(p_length <= 512)? 16 : (p_length <= 768)? 24 : 32]; /* Generator g */
    u32 y[(p_length <= 512)? 16 : (p_length <= 768)? 24 : 32]; /* Public key y */
} DSA_VERIFY_CtxCmdBuf

/* RNG Bypass */
typedef struct RNG_BYPASS_CtxCmdBuf_struct {
    u16 total_command_structure_length; /* 64 bytes long as required by PCI access */
    u16 operation_type; /* Bypass RNG mode for RNG (0x41) */
} RNG_BYPASS_CtxCmdBuf

/* RNG SHA1 */
```

```
typedef struct RNG_SHA1_CtxCmdBuf_struct {
    u16 total_command_structure_length; /* 64 bytes long as required by PCI access */
    u16 operation_type; /* RNG-SHA1 modes for RNG (0x42)*/
} RNG_SHA1_CtxCmdBuf

/*Modular Addition Atomic Operation*/
typedef struct ModAdd_CtxCmdBuf_struct {
    u16 total_command_structure_length;
    u16 operation_type; /* ModAdd (0x43)*/
    u16 reserved;
    u16 modulus_length; /* Modulus N Length in bits */

    u32 N[(modulus_length <= 512)? 16 : (modulus_length <= 768)? 24 : 32]; /* Modulus N */

} ModAdd_CtxCmdBuf;

/*Modular Subtraction Atomic Operation*/
typedef struct ModSub_CtxCmdBuf_struct {
    u16 total_command_structure_length;
    u16 operation_type; /* ModSub (0x44) */
    u16 reserved;
    u16 modulus_length; /* Modulus N Length in bits */

        u32 N[(modulus_length <= 512)? 16 : (modulus_length <= 768)? 24 : 32]; /* Modulus N
        */

} ModSub_CtxCmdBuf;

/*Modular Multiplication Atomic Operation*/
typedef struct ModMul_CtxCmdBuf_struct {
    u16 total_command_structure_length;
    u16 operation_type; /* ModMul (0x45) */
    u16 reserved;
    u16 modulus_length; /* Modulus N Length in bits */

    u32 N[(modulus_length <= 512)? 16 : (modulus_length <= 768)? 24 : 32]; /* Modulus N */

} ModMul_CtxCmdBuf;

/*Modular Reduction Atomic Operation */
typedef struct ModRem_CtxCmdBuf_struct {
    u16 total_command_structure_length;
    u16 operation_type; /* ModRem (0x46) */
    u16 message_length; /* Message M Length in bits */
    u16 modulus_length; /* Modulus N Length in bits */

    u32 N[(modulus_length <= 512)? 16 : (modulus_length <= 768)? 24 : 32]; /* Modulus N */

} ModRem_CtxCmdBuf;

/*Modular Exponentiation Atomic Operation */
typedef struct ModExp_CtxCmdBuf_struct {
    u16 total_command_structure_length;
    u16 operation_type; /* ModExp (0x47) */
    u16 exponent_length; /* Exponent E Length in bits */
    u16 modulus_length; /* Modulus N Length in bits */

    u32 N[(modulus_length <= 512)? 16 : (modulus_length <= 768)? 24 : 32]; /* Modulus N */

} ModExp_CtxCmdBuf;

/*Modular Inverse Atomic Operation */
typedef struct ModInv_CtxCmdBuf_struct {
    u16 total_command_structure_length;
```

```
    u16 operation_type; /* ModInv (0x48)*/
    u16 reserved;
    u16 modulus_length; /* Modulus N Length in bits */

    u32 N[(modulus_length <= 512)? 16 : (modulus_length <= 768)? 24 : 32]; /* Modulus N */
    u32 E[(modulus_length + 31)/32]; /* Exponent (N-2) */

} ModInv_CtxCmdBuf;

/* Master command record */
typedef struct MasterCmd_struct {
    u16 numKeysetup; /* Number of Key setups in this MCR*/
    u16 flags; /* Completion/error status from chip, per MCR */
        /* flags[0] = 1 if processing of the MCR is finished
            0 otherwise
        flags[1] = 1 if an error occurred
            0 if no error occurred
                flags[7:2]: reserved
                flags[15:8] = error code if an error occurred (i.e. flags[1] == 1),
                    undefined otherwise
        */
    /*
        * Following 5 fields occur once per key setup in the MCR
    */
    uint firstKeySetupCMDAddr;
    DataBufChain firstKeySetupData; /* First descriptor for input key setup data */
    u16 reserved;
    u16 dLength; /* Total length of the input data for the first key setup */
    DataBufChain firstOutputData; /* First descriptor for output key setup data */
    /*
        * Followed by as many sets of above 5 fields as there
        * are key setups in this MCR
    */
} MasterCmd;
```

PICTORIAL ILLUSTRATIONS OF MEMORY STRUCTURES

The tables below illustrate memory-based structures used for CPU to chip communication. Fields in quotes refer to structure names from the description on the previous pages.

IPsec ESP and AH (Bulk Encryption and Authentication) Processing

Data Buffer Chain Entries. This structure is used to build up a linked list of data buffers for every input and output packet. Each entry in the linked list points at a data buffer that contains actual packet data, a next field that points to the next descriptor entry in the linked list, and a length field that contains the number of bytes stored in the data buffer.

Table 4: Data Buffer Chain Entries

MSB																														LSB
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
Data Buffer Address dataAddr																															
Next entry in linked list of data buffers next																															
Reserved																Data buffer length dataLen														

Master Command Record. This structure is used to hand off a number of packets to the chip for processing. The structure is variable-length, and contains up to 2^{16}-1 sets of fields where each field describes one packet. This degree of flexibility allows the host CPU to queue up any number of packets, and to initiate hardware processing of all queued up packets via a single PCI write.

Table 5: Master Command Record

MSB																LSB															
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
Flags																# Packets in this MCR															
Command context address for 1st packet firstPktCMDAddr																															
Data Buffer Address dataAddr for 1st packet																															
Next entry in linked list of data buffers for 1st packet next																															
Reserved																Data buffer length dataLen 1st pkt															
Length for 1st packet pktLength																Reserved															
Output Buffer Address dataAddr for 1st packet																															
Next entry in linked list of Output buffers for 1st packet next																															
Reserved																Output buffer length dataLen 1st pkt															
Command context address for 2nd to Nth packet pktCMDAddr																															
Data Buffer Address dataAddr for 2nd to Nth packet																															
Next entry in linked list of data buffers for 2nd to Nth packet next																															
Reserved																Data buffer length dataLen 2-Nth pkt															
Length for 2-Nth packet pktLength																Reserved															
Output Buffer Address dataAddr for 2-Nth packet																															
Next entry in linked list of Output buffers for 2-Nth packet next																															
Reserved																Output buf length dataLen 2-Nth pkt															

Packet Context Buffer. This structure defines IPsec crypto and authentication processing to be applied on a per packet basis.

Table 6: Packet Context Buffer

MSB																														LSB
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
Crypto 3DES keying material, (24 bytes, high word of k1)																															
Crypto 3DES keying material (24 bytes, low word of k1)																															
Crypto 3DES keying material (high word of k2)																															
Crypto 3DES keying material (low word of k2)																															
Crypto 3DES keying material (high word of k3)																															
Crypto 3DES keying material (low word of k3)																															
HMAC Hash Inner state (high word) HMACInnerState																															
HMAC Hash Inner state																															
HMAC Hash Inner state																															
HMAC Hash Inner state (low word for MD5)																															
HMAC Hash Inner state (low word only for SHA1)																															
HMAC Hash Outer state (high word) HMACOuterState																															
HMAC Hash Outer state																															
HMAC Hash Outer state																															
HMAC Hash Outer state (low word for MD5)																															
HMAC Hash Outer state (low word only for SHA1)																															
3DES Computed IV (8 bytes, high word)																															
3DES Computed IV (8 bytes, low word)																															
Payload auth to Crypto offset cryptoOffset in 32-bit words																Crypto	Inbound	Auth (2)	Reserved											

The crypto bit must be 0 for no crypto, or 1 for 3DES-CBC. DES modes are generated by setting three consecutive 3DES keys to be equal.

The authentication value must be set as follows:

00	No authentication
01	HMAC-MD5
10	HMAC-SHA1
11	Invalid

Generation of Cryptography Initial Vector (IV). The cryptographic IV is always read from the context structure associated with a given packet. This implies that for situations where the IPsec explicit IV mode is used, the host CPU must copy IV material from packet payload to the context structure. If needed, the host may have to perform byte swapping on the IV to convert between big and little endian.

For IPsec explicit IV packets, cryptoOffset must point to the word following IV material, and the IV must be copied into packet payload as well as into the context structure. This ensures that the IV is part of the HMAC computation. For IPsec implicit IV packets, cryptoOffset must point to the first encrypted payload word, and the IV is not part of packet payload, hence is automatically left out of the HMAC computation.

Key Setup Processing

Data Buffer Chain Entries. This structure is used to build up a linked list of data buffers for every input and output message. Each entry in the linked list points at a data buffer that contains actual key set up data, a next field that points to the next descriptor entry in the linked list, and a length field that contains the number of bytes stored in the data buffer.

Unlike IPsec ESP and AH processing, key setup operations do not involve packet fragmentation. The linked list in each set of key setup is used to access different data needed for key setup computations. For Diffie-Hellman algorithms used in the IKE protocol, both the public key *Y* received from a party with whom the secret is shared and its own secret key *x* are required to compute the shared secret. In this case, the first entry points to Y data buffer. The second entry in the data buffer points to a structure that contains the pointer to *x* data buffer.

Table 7: Data Buffer Chain Entries

MSB																														LSB
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
Data Buffer Address dataAddr																															
Next entry in linked list of data buffers next																															
Reserved																Data buffer length dataLen															

Master Command Record. This structure is used to hand off a number of key setups to the chip for processing. The structure is variable-length, and contains up to 2^{16}-1 sets of fields where each field describes one key setup. This degree of flexibility allows the host CPU to queue up any number of key setups, and to initiate hardware processing of all queued up key setup sessions via a single PCI write. When using the Diffie-Hellman algorithm to generate shared secrets, two key setup operations must be performed. The first operation is to generate a public key to be sent to a party with whom the secret is shared. The second operation is to generate the shared secret using the received public key from the party. Two sets of fields are needed to complete the generation of a shared secret.

Table 8: Master Command Record

MSB 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	LSB 0
Flags																# key setups in this MCR															
Command context address for 1st key setup firstKeySetupCMDAddr																															
Data Buffer Address dataAddr for 1st key setup																															
Next entry in linked list of data buffers for 1st key setup next																															
Reserved																Data buf length dataLen 1st key setup															
Length for 1st key setup data dLength																Reserved															
Output Buffer Address dataAddr for 1st key setup																															
Next entry in linked list of Output buffers for 1st key setup next																															
Reserved																Output buf length dataLen 1st key setup															
Command context address for 2nd to Nth key setup KeySetupCMDAddr																															
Data Buffer Address dataAddr for 2nd to Nth key setup																															
Next entry in linked list of data buffers for 2nd to Nth key setup next																															
Reserved																Data buf length dataLen 2-Nth key setup															
Length for 2-Nth keysetup dLength																Reserved															
Output Buffer Address dataAddr for 2-Nth key setup																															
Next entry in linked list of Output buffers for 2-Nth key setup next																															
Reserved																Output buf length dataLen 2-Nth key setup															

Context Buffer. This structure defines DH/RSA/DSA processing to be applied on a per key setup basis.

Table 9: Diffie-Hellman Public Key Generation (X = g^x mod N) Command Context

MSB 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	LSB 0
Operation Type Diffie-Hellman Public Key Operation (0x01)																Total Command Context Structure Length															
Random Number x Length																x provided by SW/x generated by RNG															
Modulus N Length																Base g Length															
Modulus N (512, 768, 1024 bits, lowest word)																															
Modulus N (512, 768, 1024 bits, 2nd lowest word)																															
……																															
Modulus N (512, 768, 1024 bits, highest word)																															
Base g (512, 768, 1024 bits, lowest word of key)																															
Base g (512, 768, 1024 bits, 2nd lowest word of key)																															
……																															
Base g (512, 768, 1024 bits, highest word of key)																															

Table 10: Diffie-Hellman Shared Secret Generation (K=Y^x mod N) Command Context

MSB 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	LSB 0
Operation Type Diffie-Hellman Shared Secret Generation Operation (0x02)																Total Command Context Structure Length															
Modulus N Length																Exponent (private key) x Length															
Modulus N (512, 768, 1024 bits, lowest word)																															
Modulus N (512, 768, 1024 bits, 2nd lowest word)																															
……																															
Modulus N (512, 768, 1024 bits, highest word)																															

Table 11: RSA Public Key Command Context

MSB 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	LSB 0
Operation Type RSA Public Key Operation (0x03)																Total Command Context Structure Length															
Modulus N Length																Exponent E Length															
Modulus N - RSA keying material, (512, 768, 1024 bits, lowest word of key)																															
Modulus N - RSA keying material, (512, 768, 1024 bits, 2nd lowest word of key)																															
……																															
Modulus N - RSA keying material, (512, 768, 1024 bits, highest word of key)																															
Exponent E - RSA keying material, (lowest word of key)																															
Exponent E - RSA keying material, (2nd lowest word of key)																															
……																															
Exponent E - RSA keying material, (highest word of key)																															

Table 12: RSA Private Key Command Context

MSB 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	LSB 0
Operation Type RSA Private Key Operation with CRT (0x04)																Total Command Context Structure Length															
Prime p Length																Prime q Length															
Prime p - RSA keying material, (256, 384, 512 bits, lowest word of parameter)																															
Prime p - RSA keying material, (256, 384, 512 bits, 2nd lowest word of parameter)																															
……																															
Prime p - RSA keying material, (256, 384, 512 bits, highest word of parameter)																															
Prime q - RSA keying material, (256, 384, 512 bits, lowest word of parameter)																															
Prime q - RSA keying material, (256, 384, 512 bits, 2nd lowest word of parameter)																															
……																															
Prime q - RSA keying material, (256, 384, 512 bits, highest word of parameter)																															
CRT Private Exponent dp - RSA keying material, (256, 384, 512 bits, lowest word of parameter)																															
CRT Private Exponent dp - RSA keying material, (256, 384, 512 bits, 2nd lowest word of parameter)																															
……																															
CRT Private Exponent dp - RSA keying material, (256, 384, 512 bits, highest word of parameter)																															
CRT Private Exponent dq - RSA keying material, (256, 384, 512 bits, lowest word of parameter)																															
CRT Private Exponent dq - RSA keying material, (256, 384, 512 bits, 2nd lowest word of parameter)																															
……																															
CRT Private Exponent dq - RSA keying material, (256, 384, 512 bits, highest word of parameter)																															
CRT Coefficient pinv - RSA keying material, (256, 384, 512 bits, lowest word of parameter)																															
CRT Coefficient pinv - RSA keying material, (256, 384, 512 bits, 2nd lowest word of parameter)																															
……																															
CRT Coefficient pinv - RSA keying material, (256, 384, 512 bits, highest word of parameter)																															

Table 13: DSA Signing Command Context

MSB																LSB															
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
Operation Type DSA Signing (0x05)																Total Command Context Structure Length															
Reserved																Message Hash Provided/Generated															
Modulus p Length																Random Number k provided/RNG generated															
Modulus q (160 bits, lowest word)																															
Modulus q (160 bits, 2nd lowest word)																															
……																															
Modulus q (160 bits, highest word)																															
Modulus p (512, 768, or 1024 bits, lowest word)																															
Modulus p (512, 768, or 1024 bits,2nd lowest word)																															
……																															
Modulus p (512, 768, or 1024 bits, highest word)																															
Base g (512, 768, or 1024 bits, lowest word of key)																															
Base g (512, 768, or 1024 bits, 2nd lowest word of key)																															
……																															
Base g (512, 768, or 1024 bits, highest word of key)																															
Private key y (512, 768, or 1024 bits, lowest word)																															
Private key y (512, 768, or 1024 bits, 2nd lowest word)																															
……																															
Private key y (512, 768, or 1024 bits, highest word)																															

Table 14: DSA Verification Command Context

MSB 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	LSB 0
Operation Type DSA verification operation (0x06)																Total Command Context Structure Length															
Reserved																Message Hash provided/generated															
Modulus p Length																Reserved															
Modulus q (160 bits, lowest word)																															
Modulus q (160 bits, 2nd lowest word)																															
……																															
Modulus q (160 bits, highest word)																															
Modulus p (512, 768, or 1024 bits, lowest word)																															
Modulus p (512, 768, or 1024 bits, 2nd lowest word)																															
……																															
Modulus p (512, 768, or 1024 bits, highest word)																															
Base g (lowest word of key)																															
Base g (2nd lowest word of key)																															
……																															
Base g (highest word of key)																															
Public key y (512, 768, 1024 bits, lowest word)																															
Public key y (512, 768, 1024 bits, 2nd lowest word)																															
……																															
Public key y (512, 768, 1024 bits, highest word)																															

Table 15: RNG Direct Test Command Context

MSB 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	LSB 0
Operation Type RNG Direct Test Operation (0x41)																Total Command Context Structure Length (minimum length is 64 bytes)															

Table 16: RNG-SHA1 Test Command Context

MSB 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	LSB 0
Operation Type RNG-SHA1 Test Operation (0x42)																Total Command Context Structure Length (minimum length is 64 bytes)															

Table 17: ModAdd Command Context (C = (A+B) mod N)

MSB 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	LSB 0
Operation Type Modular Addition Operation (0x43)																Total Command Context Structure Length															
Modulus N Length																Reserved															
Modulus N (512, 768, 1024 bits, lowest word)																															
Modulus N (512, 768, 1024 bits, 2nd lowest word)																															
……																															
Modulus N (512, 768, 1024 bits, highest word)																															

Table 18: ModSub Command Context (C = (A-B) mod N)

MSB 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	LSB 0
Operation Type Modular Subtraction Operation (0x44)																Total Command Context Structure Length															
Modulus N Length																Reserved															
Modulus N (512, 768, 1024 bits, lowest word)																															
Modulus N (512, 768, 1024 bits, 2nd lowest word)																															
……																															
Modulus N (512, 768, 1024 bits, highest word)																															

Table 19: ModMul Command Context (C = A*B mod N)

MSB 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	LSB 0
Operation Type Modular Multiplication Operation(0x45)																Total Command Context Structure Length															
Modulus N Length																Reserved															
Modulus N (512, 768, 1024 bits, lowest word)																															
Modulus N (512, 768, 1024 bits, 2nd lowest word)																															
……																															
Modulus N (512, 768, 1024 bits, highest word)																															

Table 20: ModRem Command Context (C = M mod N)

MSB 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	LSB 0
Operation Type Modular Reduction Operation(0x46)																Total Command Context Structure Length															
Modulus N Length																Message M Length															
Modulus N (512, 768, 1024 bits, lowest word)																															
Modulus N (512, 768, 1024 bits, 2nd lowest word)																															
……																															
Modulus N (512, 768, 1024 bits, highest word)																															

Table 21: ModExp Command Context (C = M^E mod N)

MSB 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	LSB 0
Operation Type Modular Exponentiation Operation(0x47)																Total Command Context Structure Length															
Modulus N Length																Exponent E Length															
Modulus N (512, 768, 1024 bits, lowest word)																															
Modulus N (512, 768, 1024 bits, 2nd lowest word)																															
……																															
Modulus N (512, 768, 1024 bits, highest word)																															

Table 22: ModInv Command Context (C = M^{-1} mod N = M^{N-2} mod N)

MSB 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	LSB 0
Operation Type Modular Inverse Operation(0x48)																Total Command Context Structure Length															
Modulus N Length																Reserved															
Modulus N (512, 768, 1024 bits, lowest word)																															
Modulus N (512, 768, 1024 bits, 2nd lowest word)																															
……																															
Modulus N (512, 768, 1024 bits, highest word)																															
Exponent N-2 (512, 768, 1024 bits, lowest word)																															
Exponent N-2 (512, 768, 1024 bits, 2nd lowest word)																															
……																															
Exponent N-2 (512, 768, 1024 bits, highest word)																															

The selection of IPsec crypto/authentication operation versus IPsec key setup operation can be made on a per MCR basis. Within one MCR, no mix of crypto/authentication and IPsec key setup operations is allowed. The mode the current MCR operates on is determined by which DMA register the MCR address is written into. If it is written into the first DMA register (Master Command Record 1), then the chip performs crypto/authentication operations. If it is written into the fifth DMA register (Master Command Record 2), then the chip performs key setup operations.

The Operation Type bits must be set as follows:

- 0x01 – Diffie-Hellman public key generation operation
- 0x02 – Diffie-Hellman shared secret generation operation
- 0x03 – RSA public key operation
- 0x04 – RSA private key operation (RSA operation with Chinese Remainder Theory)
- 0x05 – DSA signing operation
- 0x06 – DSA verification operation
- 0x41 – RNG direct test mode
- 0x42 – RNG-SHA1 test mode
- 0x43 – Modular Addition
- 0x44 – Modular Subtraction
- 0x45 – Modular Multiplication
- 0x46 – Modular Reduction (Remainder)
- 0x47 – Modular Exponentiation
- 0x48 – Modular Inverse
- Other values – Reserved for future use

The number of entries a command context has depends on Operation Type and number of bits used for the operation. The total_command__context_length field provides the total number of bytes required for the command context structure for a given key setup or an atomic arithmetic operation. Since the minimum number of bytes required for a PCI access is 64 bytes, the field should have 64 bytes for the RNG test modes.

For DH public key generation and DSA signing operation, either the on-chip Random Number Generator can be used to generate x for DH and k for DSA or else the values can be obtained from the software. If they are generated by RNG, the Provided/RNG Generated (RNG Enable) bits in command context are set to one. Otherwise, they are set to 0. If they are provided by the application software, then they are stored in data buffers. The chip retrieves them during processing of MCR structure.

For DSA signing and verification operations, message hash can either be provided by software (CPU does the hashing) or be performed by SHA1 unit on the chip. If hash is done by SHA1, the Message Hash Provided/Generated (SHA1 Enable) bits are set to one. Otherwise, they are set to zero. Either the message or the message hash is stored in the input data buffer. The chip retrieves them during MCR structure processing.

For DH send mode, both public key and private key are generated and stored in the output data buffers in a linked list fashion.

For DH receive mode, both public key and private key are provided for shared secret computation and stored in the input data buffers in a linked list fashion.

For DSA signing mode, both r and s are generated and stored in output data buffers in a linked list fashion.

For DSA verification mode, both r and s are provided by application and stored in input data buffers in a linked list fashion.

For RNG bypass and RNG-SHA1 modes, there is no input data buffer required and one output data buffer containing the random numbers. The length of the data buffer is contained in the output buffer length field in MCR.

For atomic operations ModAdd, ModSub, ModMul, ModRem, ModExp, and ModInv, the modulus is passed to the chip via command context structure and other operands are stored in the input data buffers in a linked list fashion. In typical applications, modulus does not change for each operation. For ModInv, a modular inverse operation was converted to a modular exponentiation operation. Because of that, (N-2) is stored where N is the modulus, in the command context.

The following table shows the data chaining in the MCR structure for various key setup algorithms. Symbol *A* ➔ *B* is used to represent that the next field in data buffer A points to the data buffer for B.

Table 23: MCR Input/Output Data Buffer Chaining

Algorithms	*Input Data Chaining*	*Output Data Chaining*
DH Send	**Private Key *x* Provided by SW.** If the private key is generated by RNG, no input data is needed. Input data buffer length is zero.	Public Key data buffer ➔ Private Key data buffer
DH Receive	**Public Key data buffer ➔ Private Key data buffer .** The SW driver must keep track of the corresponding private keys to generate the shared secret.	Shared secret buffer
RSA Public Key	Message data buffer	Message data buffer
RSA Private Key	Message data buffer	Message data buffer
DSA Signing	**m data buffer ➔ Random number k Provided by SW.** If k is generated by RNG, only message data is stored in the input data buffer. The M data buffer can contain multiple fragments. In this case, random number k provided by software follows the last fragment of m data buffer. The dlength field of the key setup is the total length (in bytes) of m data buffer (does not include the random number k). However, the fragments other than the last one must be integer multiple of 512 bits. The last fragment can be in any length.	r parameter data buffer ➔ s parameter data buffer
DSA Verification	**m data buffer ➔ r parameter data buffer ➔ s parameter data buffer.** The M data buffer can contain multiple fragments. In this case, r parameter data buffer follows the last fragment of m data buffer. The dlength field of the key setup is the total length (in bytes) of m data buffer (does not include r and s parameter data buffers). However, the fragments other than the last one must be integer multiple of 512 bits. The last fragment can be in any length.	v parameter buffer
RNG Bypass Mode	None	Random number buffer
RNG SHA1 Randomized Mode	None	Random number buffer
ModAdd ((A+B) mod N)	A data buffer ➔ B data buffer	Output data buffer
ModSub ((A-B) mod N)	A data buffer ➔ B data buffer	Output data buffer
ModMul (A*B mod N)	A data buffer ➔ B data buffer	Output data buffer
ModRem (A mod N)	*A* data buffer	Output data buffer
ModExp (A^E mod N)	A data buffer ➔ E data buffer	Output data buffer
ModInv (A^{-1} mod N)	*A* data buffer	Output data buffer

ALIGNMENT RESTRICTIONS

The following table shows alignment requirements for all memory-resident data in IPsec crypto/authentication operations.

Table 24: Memory-Resident Data Alignment Requirements in IPsec Crypto/Authentication Operations

Memory-Resident Data Type	*Alignment Requirement, Size Requirement*
Packet Payload Data	
Packet Input Data Buffers (per descriptor)	None (byte), None (byte)
Packet Output Data Buffers (per descriptor)	32-bit, length multiple of 32 bits
Control and Command Structures	
Descriptors (Input and Output)	32-bit, fixed size (3 words of 32 bits)
Command Context Structure	32-bit, fixed size (19 words of 32 bits)
Master Command Record	32-bit, variable size (1 + #pkts*8 32-bit words)

The flexibility with respect to input packet payload data allows extreme combinations to be supported. For instance, a packet with 16,000 bytes of input payload data could be described as a chain of 16,000 descriptors, with each descriptor holding one single byte. The BCM5802 handles such an extreme situation correctly from a functional standpoint, albeit with reduced performance from the huge number of descriptor fetches.

The following table shows alignment requirements for all memory-resident data in DH/RSA/DSA operations.

Table 25: Memory-Resident Data Alignment Requirements in DH/RSA/DSA Operations

Memory-Resident Data Type	*Alignment Requirement, Size Requirement*
Packet Payload Data	
Input Data Buffers (per descriptor)	32-bit, length multiple of 32 bits
Output Data Buffers (per descriptor)	32-bit, length multiple of 32 bits
Control and Command Structures	
Descriptors (Input and Output)	32-bit, fixed size (3 words of 32 bits)
Command Context Structure	32-bit, fixed size (variable words of 32 bits)
Master Command Record	32-bit, variable size (1 + #key setup*8 32-bit words)

Because IKE/SSL/TLS key setups operate at or above Layer 4 of the network stack, users have full control of the data memory allocation. Aligning data at the 32-bit boundary is relatively easy to do for software.

INVALID ENCRYPTION/AUTHENTICATION OPERATIONS

This section details scenarios that the software should never request the chip to process. These can cause unknown results being written to memory, or possibly a chip hang condition.

- Zero-length packets: These can arise in several ways, all of which should be avoided. One way is to have a zero total packet length in a MCR structure. Another is to have a non-zero packet length, but to set the crypto offset equal to or greater than the entire length of the packet.
- Zero-length descriptors: All data buffer entries in input and output descriptor chains should have a non-zero length. Similarly, requesting the chip to use a zero output fragment size from the output fragment register would lead to unpredictable results.
- Erroneous parameter specifications: Situations such as illegal authentication specifiers, misaligned structure members, and misaligned output packet payload data, should be guaranteed to never occur.
- Output descriptors that point to misaligned output data buffers: All output data should be aligned on 32-bit boundaries.
- Output descriptors that indicate an output buffer byte length that is not a multiple of four: All output data buffers must have a length that is multiple of 32-bits.
- Non-zero crypto offset with crypto disabled.
- Packets with both authentication and crypto disabled.
- Packets with crypto disabled, but with an output descriptor chain of length > 1 specified: For packets that have no crypto output (hence *must* have an authentication output), there must be one, and exactly one output descriptor specified in the Master Command Record. Only the next field of this descriptor is used to write out the HMAC codes. Other fields of this descriptor (in particular the data buffer address and size) are ignored.
- Incorrect packet size for cryptography: Whenever 3DES is enabled, the length of input data to be encrypted must be a multiple of eight bytes. The input data length is calculated as total packet size minus the number of 32-bit dwords specified by the crypto offset context field. Giving the chip a crypto data length that is not a multiple of eight bytes could hang the chip. IPsec padding guarantees that this never happens.
- Crypto offset that leads to a data length for encryption or decryption that is not multiple of 64-bits: For instance, a crypto offset of one word with a total packet length of 40 words would force the crypto unit to process 39 words, which is not a multiple of eight bytes. However, a crypto offset of one word with a packet length of 41 words is fine, as is a crypto offset of two words with a packet length of 40 words.
- Non-zero crypto offset for packets that do not have both crypto and authentication enabled: If authentication is disabled, the crypto offset *must* be set to zero. Crypto offset can not be used as a programmable skip length for crypto-only packets.
- Writing to the MCR register with PCI master mode disabled: Doing so causes the control microcode to start processing and hang, waiting for a PCI master mode access that never begins.
- The #Packet or #Key Setup in the first field in an MCR cannot be zero.
- The Flags field (second field) in an MCR must be zeroed out before sending the MCR pointer to DMA register on the BCM5802.

BCM5802 REGISTERS

The BCM5802 registers are divided into two categories.

1. PCI configuration registers implement control and status information that is specific to the PCI bus, as well as registers required by the PCI specification revision 2.2.
2. DMA control and status registers correspond to master command, data and packet context fetch and write back operations.

Unused bits read as an unknown value which could be zero or one, and should be masked off prior to further processing. Unused bits should be written as zeroes. The following mnemonics are used to describe the types of access allowed for each register bit:

- RW: bit is read/write
- WO: bit is write only
- RO: read only bit (i.e. status flag)
- RSVD: reserved bit, ignore upon read, write 0s upon write

A value of X upon reset means that the state of the register is undefined and should not be relied upon after a reset occurs.

PCI CONFIGURATION REGISTERS

The BCM5802 provides PCI 2.2-compliant configuration space registers as follows. In addition, the BCM5802 uses PCI Memory BAR0 for all slave control and status registers. The registers use a total memory space of 64 KB in one memory BAR region. This region is non-pre-fetchable, and must be relocated only in 32-bit space.

Configuration registers not shown in the table below are reserved.

Table 26: PCI 2.2-Compliant Configuration Space Registers

ADDR	31 Bits 16		15 Bits 00	
0x00	Device ID		Vendor ID	
0x04	Status		Command	
0x08	Class code			Rev ID
0x0C	BIST	Header Type	Master Latency Timer	Cache line Size
0x10	Memory BAR0			
0x2C	Subsystem ID		Subsystem Vendor ID	
0x3C	MAX_LAT	MIN_GNT	Interrupt Pin	Interrupt Line
0x40	Reserved		Retry Timeout	TRDY Timeout

The various registers within PCI configuration space are as follows.

Table 27: PCI Configuration Registers

Bits	Access	Reset	Purpose
PCI Vendor ID: 0x00			
15:0	RO	14E4	Hard-wired device identifier (0x14E4), Broadcom ID assigned by PCISIG.
PCI Device ID: 0x02			
31:16	RO	5802	Hard-wired device identifier (0x5802).
PCI Command Register: 0x04			
15:10	RSVD	0	Reserved.
9	RW	0	Fast back to back master enable.
8	RW	0	System error enable.
7	RSVD	0	Reserved.
6	RW	0	Parity error enable.
5	RSVD	0	Reserved.
4	RW	0	Memory write and Invalidate enable.
3	RSVD	0	Reserved.
2	RW	0	Bus master enable.
1	RW	0	Memory access enable.
0	RW	0	I/O access enable (ignored, leave at 0).
PCI Status Register: 0x04			
31	RO	0	Detect parity error.
30	RO	0	Signaled system error.
29	RO	0	Received master abort status.
28	RO	0	Received target abort status.
27	RO	0	Signaled target abort status.
26:25	RO	01	DEVSEL timing.
24	RO	0	Data parity detected.
23	RO	1	Fast back-to-back capable status.
22	RSVD	0	Reserved.
21	RO	0	66-MHz capable.
20:16	RSVD	0	Reserved.
PCI Rev ID: 0x08			
7:0	RO	01/E1	Hard-wired device revision identifier (0x01 for domestic version and 0xE1 for export version).

Table 27: PCI Configuration Registers (Cont.)

Bits	Access	Reset	Purpose
PCI Class Code Register: 0x08			
31:8	RO	0B4000	**Class code value (hard-wired).** 0x0B4000 (processor class, coprocessor subclass).
PCI BIST Register, Cache line, Master Latency, Header: 0x0C			
31	RO	0	**BIST capable.** The BCM5802 is not capable of performing PCI configuration BIST operation.
30	RW	0	**BIST Start.** Not supported on BCM5802.
29:28	RO	0	Reserved.
27:24	RO	0	**BIST completion code.** Not supported on BCM5802.
23:16	RW	0	Header type.
15:0	RW	0	Master latency timer.
7:0	RW	0	Cache line size.
PCI Memory BAR: 0x10			
31:0	RW	0xFFFF0000	Memory Base Address Register, 64 KB region, non-prefetchable, relocate in 32-bit space only.
PCI MAX_LAT, MIN_GNT, Interrupt: 0x3C			
31:24	RO	0	PCI MAX_LAT parameter.
23:16	RO	0	Length of burst period MIN_GNT.
15:8	RO	0x1	Interrupt pin register.
7:0	RW	0	Interrupt line register.
PCI Retry Timeout, TRDY Timeout: 0x40			
15:8	RW	0x80	Number of retries that the PCI interface performs.
7:0	RW	0x80	TRDY timeout value.

DMA Control and Status Registers

The DMA registers control how master command structures, packet context and packet data are fetched and then stored after processing. All of the following registers are located in PCI Memory BAR0 space. A second MCR register has been added in the BCM5802 to handle the key setup operations. The BCM5802 is completely compatible with the BCM5801 for crypto/authentication operations. The BCM5801 software driver also works on the BCM5802 without modification.

Table 28: PCI Memory BAR0 Space DMA Registers

ADDR	31 Bits 16	15 Bits 00
0x00	Master Command Record 1@	
0x04	DMA Control	
0x08	DMA Status	
0x0C	DMA Error Address	
0x10	Master Command Record 2@	

The following table shows the DMA control and status registers.

Table 29: DMA Control and Status Registers

Bits	Access	Reset	Purpose
DMA Master Command Record 1@: 0x00			
31:0	RW	X	Writing the address of a valid Master Command Record to this register causes crypto/authentication processing of the packets within that record to begin. This register must only be written when the MCR_FULL bit of the DMA Status register is 0. This register is double buffered, such that the MCR_FULL bit goes to zero very quickly after an initial write. This allows the CPU to write a second MCR address value to this register, effectively queuing up to MCR structures for back to back processing with zero latency. Reset state is Unknown. Do not write if PCI master mode is disabled.
DMA Control: 0x04			
31	RW	0	**RESET.** Software reset. Normally, it is unset. If software detects hanging or other undesirable states of BCM5802, it sets this bit to reset. After writing 1 to this bit, you must wait 30 PCI clocks before the chip can be accessed again.
30	RW	0	**MCR2INT_EN.** Enable interrupt per MCR for MCR2. An interrupt is generated every time an entire MCR completes processing. This is the preferred operational mode. Resets to 0.
29	RW	0	**MCR1INT_EN.** Enable interrupt per MCR for MCR1. An interrupt is generated every time an entire MCR completes processing. This is the preferred operational mode. Resets to 0.
28	RSVD	0	Reserved.
27	RSVD	1	Reserved. Do not change its reset value.
26	RSVD	1	Reserved. Do not change its reset value.
25	RW	0	**DMAERR_EN.** Enable interrupt upon DMA master access error.

Table 29: DMA Control and Status Registers (Cont.)

Bits	Access	Reset	Purpose
24:23	WO	00	**RNG_MODE** • 00: 1 bit random number per one slow clock cycle. • 01: 1 bit random number per four slow clock cycles • 10: 1 bit random number per eight slow clock cycles • 11: 1 bit random number per sixteen slow clock cycles
15:0	RSVD	0	Reserved.
DMA Status: 0x08			
31	RO	0	Master access in progress. Resets to 0.
30	RO	0	**MCR1_FULL flag.** Master Command Address register is full. When this flag is 1, the CPU must not write to the MCR1@register. When this flag is 0, the PCU may write a value to the MCR1@register to request processing of a master command structure. Resets to 0.
29	RW	0	**MCR1_INTR.** Completion interrupt status of per-MCR interrupt for MCR1. Cleared by writing a 1 to this bit position. ***Note:*** This bit accurately reflects processing status, even if the corresponding interrupt bit is disabled (in which case a PCI interrupt is not generated). This bit is sticky until cleared explicitly. Resets to 0.
28	RW	0	**DMAERR_INTR.** Interrupt status for MCR DMA master access error. Sticky until software reset (DMA control bit 31 is set to 1) or hardware reset. This bit accurately reflects status even if the corresponding interrupt enable bit is off (in which case a PCI interrupt is not generated). Resets to 0.
27	RO	0	**MCR2_FULL flag.** Master Command Address register is full. When this flag is 1, the CPU must not write to the MCR2@ register. When this flag is 0, the CPU may write a value to the MCR2@ register to request processing of a master command structure. Resets to 0.
26	RW	0	**MCR2_INTR.** Completion interrupt status of per-MCR interrupt for MCR2. Cleared by writing a 1 to this bit position. ***Note:*** This bit accurately reflects processing status (in which case a PCI interrupt is not generated). This bit is sticky until cleared explicitly. Resets to 0.
DMA Error Address: 0x0C			
31:2	RO	X	Address of master access that resulted in a PCI fault (32b word address). Reset state unknown.
1	RO	X	1 = faulted master access was a read, 0 = was a write. Reset state unknown.
DMA Master Command Record 2@: 0x10			
31:0	RW	X	Writing the address of a valid Master Command Record to this register causes key setup processing of the data within that record to begin. This register must only be written when the MCR_FULL bit of the DMA Status register is 0. This register is double buffered, such that the MCR_FULL bit goes to zero very quickly after an initial write to this register. This allows the CPU to write a second MCR address value to this register, effectively queuing up to MCR structures for back-to-back processing with zero latency. Reset state is unknown. Do not write if PCI master mode is disabled.

Section 5: Electrical and Timing Specifications

Table 30: Electrical and Timing Specifications

Parameter	*Typical*	*Description*
PCI Compliance	3.3V and 5V	Over the range of 25-33 MHz PCI clocks
Supply Voltage	3.3V ±5%	
Power Consumption	1.2W	Typical power consumption at 33 MHz
I/O Buffers	3.3V	
Operating Temperature	0-70C	Within the commercial temperature range
Timing Specification for the I/O Pins		Follows the PCI 2.2 timing specification
The BCM5802 works in both 3.3V and 5V PCI environments		

Table 31: PCI Pin DC Specifications

Symbol	*Parameter*	*Condition*	*Min*	*Max*	*Units*
V_{CC}	Supply Voltage		3.135	3.465	V
V_{IH}(FRAME#)	Input High Voltage for FRAME# pin		$0.52V_{CC}$	V_{CC} + 0.5	V
V_{IH}(PERR#)	Input High Voltage for PERR# pin		$0.52V_{CC}$	V_{CC} + 0.5	V
V_{IH}	Input High Voltage for all other pins		$0.50V_{CC}$	V_{CC} + 0.5	V
V_{IL}	Input Low Voltage		-0.5	$0.3V_{CC}$	V
V_{IPU}	Input Pull-up Voltage		$0.7V_{CC}$		V
V_{OH}	Output High Voltage	I_{OUT} = -0.5 mA	$0.9V_{CC}$		V
V_{OL}	Output Low Voltage	I_{OUT} = 1.5 mA		$0.1V_{CC}$	V
C_{IN}	Input Pin Capacitance		5	12	pF
C_{CLK}	PCI_CLK Pin Capacitance			8	pF
L_{PIN}	Pin Inductance			20	nH

FRAME# and PERR# pins violated V_{IH} PCI specification very slightly at the corners of the operating temperature range. All other pins are within the PCI DC Specifications. All the pins, including FRAME# and PERR#, satisfy the PCI Timing Specifications.

Section 6: Mechanical Information

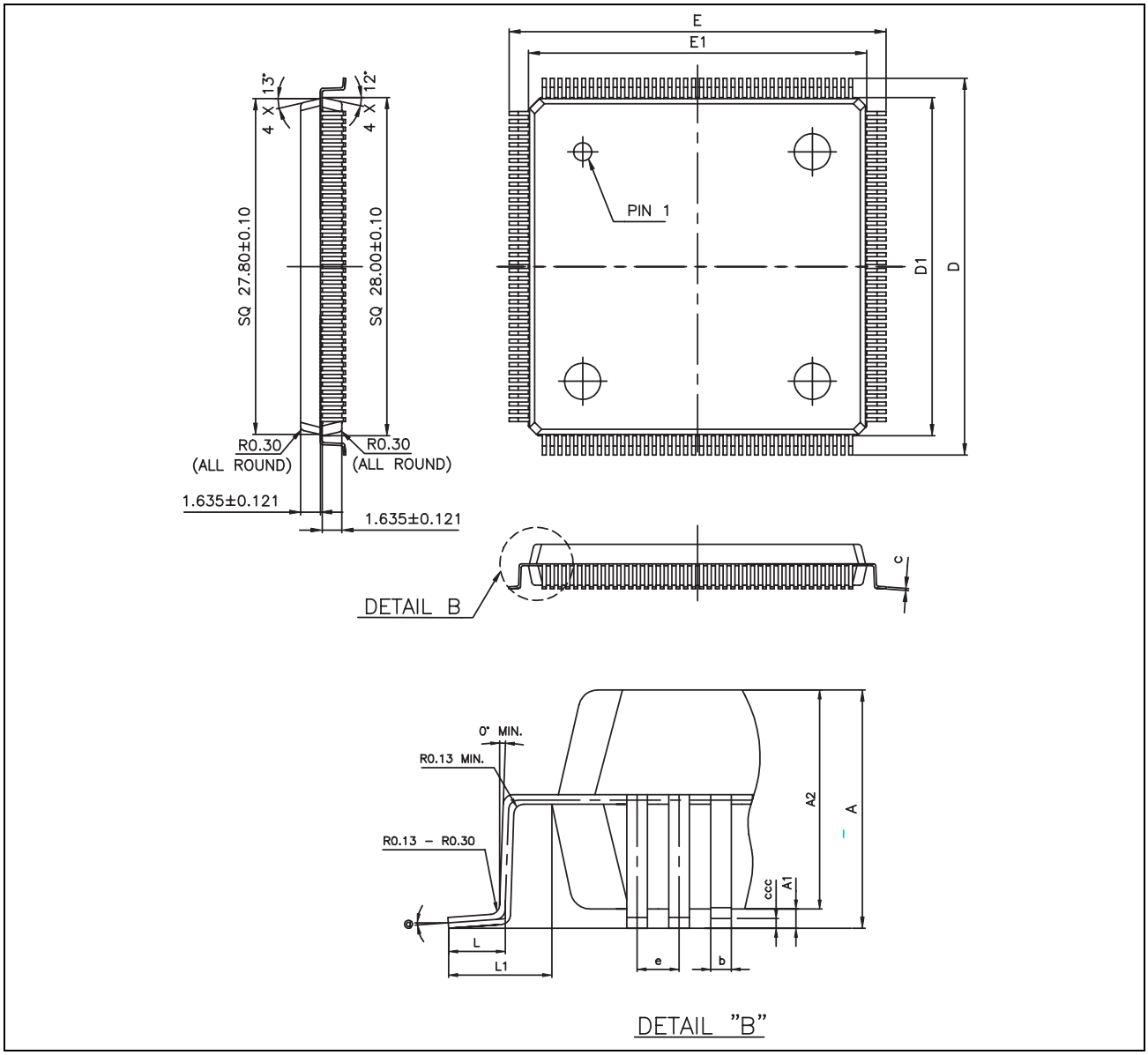


Figure 6: 144-Pin DQFP Package Drawing

Table 32: 144-Pin DQFP Package Dimensions

Symbol	*Dimension*	*Remarks*
ccc	max. 0.102 (0.004)	Planarity
ddd	max. 0.127 (0.005)	Bent Lead
c	0.13 - 0.23	Lead Thickness
L	0.88 (±0.15)	Foot Length
L1	1.60 (REF)	—
E1	28.0 (±0.10)	Package Length
E	31.2 (±0.25)	Lead to Lead Length
D1	28.0 (±0.10)	Package Width
D	31.2 (±0.25)	Lead to Lead Width
A2	3.42 (±0.25)	Package Thickness
A1	min. 0.25	Standoff
A	max. 4.07	Overall Height
e	0.65 basic	Lead Pitch
b	0.22 - 0.38	Lead Width

Broadcom Corporation

Broadcom Corporation
P.O. Box 57013
16215 Alton Parkway
Irvine, California 92619-7013