Application Note

AN2492/D
Rev. 0, 6/2003

MPC184 Descriptor Programmer's Guide—PCI View

Geoff Waters
Security Applications
Michael Torla
Security Design

This application note is offered as a supplement to the *MPC184 Security Co-Processor User's Manual, PCI Interface*, to assist the user in understanding and creating descriptors in the event the user has more specific requirements than those covered by the MPC184 device driver. This application note will be more useful if the reader is already basically familiar with the MPC184 architecture, as explained in the user's manual. All descriptor and execution unit references are shown in little endian format consistent with the PCI version of the MPC184 user's manual.

The following topics are addressed:

1 Data Packet Descriptor Overview

The MPC184 has bus mastering capability on either 32-bit PCI or the PowerQUICC 8xx bus to off-load data movement and encryption operations from a host processor. As the system controller, the host processor maintains a record of current secure sessions and the corresponding keys and contexts of those sessions. Once the host has determined a security operation is required, it can either directly write keys, context, and data to the MPC184 (MPC184 in target mode), or the host can create a 'data packet descriptor' to guide the MPC184 through the security operation, with the MPC184 acting as a bus master. The descriptor can be created in main memory, any memory local to the MPC184, including 8 Kbytes of on-chip gpRAM, or written directly to the data packet descriptor buffer in the MPC184 crypto-channel.

2 Descriptor Structure

The MPC184 data packet descriptors are conceptually similar to descriptors used by most devices with DMA capability. See Figure 1 for a conceptual data packet descriptor. The descriptors are fixed length (64 bytes), and consist of sixteen 32-bit fields. Descriptors begin with a header, which describes the security operation to be performed and the mode that the execution unit will be set to while performing the operation.

The header is followed by seven data length/data pointer pairs. Data length indicates the amount of contiguous data to be transferred. This amount cannot exceed 32 Kbytes. The data pointer refers to the address of the data which the MPC184 fetches. In this case, data is broadly interpreted to mean keys, context, additional pointers, or the actual plain text to be permuted.

Figure 1 shows an example data packet descriptor.

31   0
Descriptor Header
R/W
Pointer 1
Length 2
Pointer 2
Length 3
Pointer 3
Length 4
Pointer 4
Length 5
Pointer 5
Length 6
Pointer 6
Length 7
Pointer 7
Next Descriptor Pointer

Figure 1. Example Data Packet Descriptor

3 Descriptor Header

Descriptors are created by the host to guide the MPC184 through required cryptographic operations. The descriptor header defines the operations to be performed, the mode for each operation, and the ordering of the inputs and outputs in the body of the descriptor. The MPC184 device drivers allow the host to create proper headers for each cryptographic operation. Figure 2 shows the descriptor header.

	31 … 20	19 … 8	7 … 4	3 … 2	1	0
Field	Op_0	Op_1	Desc_TYPE	RSVD	ST	DN
Reset	0x0000_0000					
R/W	R/W					
Addr	Channel_1 0x02080, Channel_2 0x03080, Channel_3 0x04080, Channel_4 0x05080					

Figure 2. Descriptor Header

Table 1 defines the header bits.

Table 1. Header Bit Definitions

Bits	Name	Description
31:20	Op_0	Op_0 contains two sub fields, EU_Select and Mode_Data. Figure 3 shows the sub field detail. EU_SELECT[31:28]—Programs the channel to select a primary EU of a given type. Table 2 lists the possible EU_SELECT values. MODE_DATA[27:20]—Programs the primary EU mode data. The mode data is specific to the chosen EU. This data is passed directly to bits 7:0 of the specified EU mode register.
19:8	Op_1	Op_1 contains two sub fields, EU_Select and Mode_Data. Figure 3 shows the sub field detail. EU_SELECT[19:16]—Programs the channel to select a secondary EU of a given type. Table 2 lists the possible EU_SELECT values. MODE_DATA[15:8]—Programs the secondary EU mode data. The mode data is specific to the chosen EU. This data is passed directly to bits 7:0 of the specified EU mode register. **Note:** The MDEU is the only valid secondary EU. Values for Op1 EU_SELECT other than ‘MDEU’ or ‘no secondary CHA selected’ will result in an ‘unrecognized header’ error condition. Selecting MDEU for both primary and secondary EU will also create an error condition.
7:4	Desc_Type	Descriptor type—Each type of descriptor determines the following attributes for the corresponding data length/pointer pairs: the direction of the data flow; which EU is associated with the data; and which internal EU address is used. Table 9 lists the valid types of descriptors.
3:2	—	Reserved—set to zero
1	ST	Snoop type—Selects which of the two types of available snoop modes applies to the descriptor. See Figure 11 for a graphical representation of the snooping concept. 0 Snoop output data mode 1 Snoop input data mode In ‘snoop input data mode,’ while the bus transaction to write data into the input FIFO of the primary EU is in progress, the secondary EU (always MDEU) will snoop the same data into its input FIFO. In ‘snoop output data mode’, the secondary EU (always MDEU) will snoop data into its input FIFO during the bus transaction to read data out of the output FIFO of the primary EU.

Table 1. Header Bit Definitions (continued)

Bits	Name	Description
0	DN	DONE_NOTIFICATION_FLAG—Done Notification Flag Setting this bit indicates whether to perform notification on completion of this descriptor. The notification can take the form of an interrupt or modified header write back or both depending on the state of the INTERRUPT_ENABLE and WRITEBACK_ENABLE control bits in the crypto-channel configuration register. 0 Do not signal DONE on completion of this descriptor (unless globally programmed to do so via the crypto-channel configuration register) 1 Signal DONE on completion of this descriptor **Note:** The MPC184 can be programmed to perform DONE notification on completion of each descriptor, completion of any descriptor, or completion of the final descriptor in a chain. This bit provides for the second case. When the crypto-channel is requesting a write of the descriptor header back to system memory, the least significant byte (little endian) of the header will always read as set to 0xFF, and the remaining 24 bits will not be changed.

Figure 3 shows the two sub fields of Op_x.

11–8	7–0
Op_x	
EU_SELECT	MODE_DATA

Figure 3. Op_x Sub Field

Op0 EU_SELECT values of ‘no primary EU selected’ or ‘reserved EU’ will result in an ‘unrecognized header error’ condition during processing of the descriptor header. Also, the primary EU selected by the Op0 EU_SELECT field may only be DEU, AESU, or AFEU when a valid secondary EU is selected. For this case, all other values of Op0 EU_SELECT will result in an ‘unrecognized header’ error condition. The full range of permissible EU_Select values is shown in Table 2.

Table 2. EU_Select Values

Value	EU Select
0000	No EU selected
0001	AFEU
0010	DEU
0011	MDEU
0100	RNG
0101	PKEU
0110	AESU
Others	Reserved EU

4 Execution Unit Mode Data

The MPC184 execution units are programmed via the descriptor header. The Mode_Data portion of Op_X field in the descriptor header is written to bits 7:0 of the mode register in the execution unit selected by the EU_Select field in Op_X. A complete explanation of the execution unit registers can be found in Chapter 5 of the *MPC184 Security Co-Processor User's Manual, PCI Interface*, however, the mode register for each EU is provided in this section for convenience.

4.1 PKEU Mode Register

This register specifies the internal PKEU routine to be executed. For the root arithmetic routines, PKEU has the capability to perform arithmetic operations on subsegments of the entire memory. This is particularly useful for operations such as ECDH (elliptic curve Diffie-Hellman) key agreement computation. By using regAsel and regBsel, for example, parameter memory A subsegment 2 can be multiplied into parameter memory B subsegment 1. Figure 4 and Figure 5 detail two definitions.

	31 7	6 0
Field	Reserved	Mode
Reset	0	0
R/W	R/W	
Addr	PKEU 0x10000	

	31 0
Field	Reserved
Reset	0
R/W	R/W
Addr	PKEU 0x10004

Figure 4. PKEU Mode Register: Definition 1

	31 7	6 4	3 0
Field	Reserved	Mode	regsel
Reset	0	0	0
R/W	R/W		
Addr	PKEU 0x10000		

	31 0
Field	Reserved
Reset	0
R/W	R/W
Addr	PKEU 0x10004

Figure 5. PKEU Mode Register: Definition 2

Table 3 lists mode register routine definitions. Parameter memories are referred to for the base address, as shown.

Table 3. Mode Register Routine Definitions

Routine	Mode [6:4]	Mode [3:2]	Mode [1:0]
Reserved	000	00	00
Clear memory	000	0	01
Modular exponentiation	000	00	10
R^2 mod N	000	00	11
R_nR_p mod N	000	01	00
F_p affine point multiplication	000	01	01
F_2m affine point multiplication	000	01	10
F_p projective point multiplication	000	01	11
F_2m projective point multiplication	000	10	00
F_p point addition	000	10	01
F_p point doubling	000	10	10
F_2m point addition	000	10	11
F_2m point doubling	000	11	00
F_2m R^2 CMD	000	11	01
F_2m INV CMD	000	11	10
MOD INV CMD	000	11	11
Modular addition	001	regAsel [1] 00 = A0 01 = A1 10 = A2 11 = A3	regBsel [1] 00 = B0 01 = B1 10 = B2 11 = B3
Modular subtraction	010		
Modular multiplication with single reduction	011		
Modular multiplication with double reduction	100		
Polynomial addition	101		
Polynomial multiplication with single reduction	110		
Polynomial multiplication with double reduction	111		

[1] In this case, regAsel and regBsel refer to the specific segment of parameter memory A and B.

4.2 DEU Mode Register

The DEU mode register contains 3 bits which are used to program the DEU, as shown in Figure 6. It also reflects the value of burst size, which is loaded by the crypto-channel during normal operation with the MPC184 as an initiator. Burst size is not relevant to target mode operations, where an external host pushes and pulls data from the execution units.

The mode register is cleared when the DEU is reset or re-initialized. Setting a reserved mode bit will generate a data error. If the mode register is modified during processing, a context error will be generated.

	31–11	10–8	7–3	2	1	0
Field	Reserved	Burst Size	Reserved	CE	TS	ED
Reset	0	0	0	0	0	0
R/W	R/W					
Addr	DEU 0x0A000					

	31–0
Field	Reserved
Reset	0
R/W	R/W
Addr	DEU 0x0A004

Figure 6. DEU Mode Register

Table 4 describes the DEU mode register signals.

Table 4. DEU Mode Register Signals

Bits	Signal	Description
31:11	—	Reserved
10:8	Burst size	The MPC184 implements flow control to allow larger than FIFO sized blocks of data to be processed with a single key/IV. The DEU signals to the crypto-channel that a ‘burst size’ amount of data is available to be pushed to or pulled from the FIFO. **Note:** The inclusion of this field in the DEU mode register is to avoid confusing a user who may read this register in debug mode. Burst size should not be written directly to the DEU.
7:3	—	Reserved
2	CBC/ECB	If set, DEU operates in cipher-block-chaining mode. If not set, DEU operates in electronic codebook mode. 0 ECB mode 1 CBC mode
1	Triple/single DES	If set, DEU operates the triple DES algorithm; if not set, DEU operates the single DES algorithm. 0 Single DES 1 Triple DES
0	Encrypt/decrypt	If set, DEU operates the encryption algorithm; if not set, DEU operates the decryption algorithm. 0 Perform decryption 1 Perform encryption

4.3 AFEU Mode Register

The AFEU mode register contains 3 bits which are used to program the AFEU, as shown in Figure 7. It also reflects the value of burst size, which is loaded by the crypto-channel during normal operation with the MPC184 as an initiator. Burst size is not relevant to target mode operations, where an external host pushes and pulls data from the execution units.

The mode register is cleared when the AFEU is reset or re-initialized. Setting a reserved mode bit will generate a data error. If the mode register is modified during processing, a context error will be generated.

4.3.1 Host-Provided Context via Prevent Permute

In the default mode of operation, the host provides the key and key size to the AFEU. The initial memory values in the S-box are permuted with the key to create new S-box values, which are used to encrypt the plaintext.

If the ‘prevent permute’ mode bit is set, the AFEU will not require a key. Rather, the host will write the context to the AFEU and message processing will occur using the provided context. This mode is used to resume processing of a message using the already permuted S-box. The context may be written through the FIFO if the ‘context source’ mode bit is set.

4.3.2 Dump Context

This mode may be independently specified in addition to host-provided context mode. In this mode, once message processing is complete and the output data is read, the AFEU will make the current context data available for reads via the output FIFO.

NOTE

After the initial key permute to generate a context for an AFEU encrypted session, all subsequent messages will re-use that context, such that it is loaded, modified during the encryption, and unloaded, similar to the use of a CBC initialization vector in DES operations. A new context is generated (via key permute) according to a rekeying interval specified by the security protocol. Context should never be loaded to encrypt a message if a key is loaded and permuted at the same time.

	31–11	10–8	7–3	2	1	0
Field	Reserved	Burst Size	Reserved	CS	DC	PP
Reset	0	0	0	0	0	0
R/W	R/W					
Addr	AFEU 0x08000					

	31–0
Field	Reserved
Reset	0
R/W	R/W
Addr	AFEU 0x08004

Figure 7. AFEU Mode Register

Table 5 describes the AFEU mode register signals.

Table 5. AFEU Mode Register Signals

Bits	Signal	Description
31:11	—	Reserved
10:8	Burst size	The MPC184 implements flow control to allow larger than FIFO sized blocks of data to be processed with a single key/context. The AFEU signals to the crypto-channel that a ‘burst size’ amount of data is available to be pushed to or pulled from the FIFO. **Note:** The inclusion of this field in the AFEU mode register is to avoid confusing a user who may read this register in debug mode. Burst size should not be written directly to the AFEU.
7:3	—	Reserved
2	Context source	If set, this causes the context to be moved from the input FIFO into the S-box prior to starting encryption/decryption. Otherwise, context should be directly written to the context registers. Context source is only checked if the prevent permute bit is set. 0 Context not from FIFO 1 Context from input FIFO
1	Dump context	If set, this causes the context to be moved from the S-box to the output FIFO following assertion of AFEU’s done interrupt. 0 Do not dump context 1 After cipher, dump context
0	Prevent permute	Normally, AFEU receives a key and uses that information to randomize the S-box. If reusing a context from a previous descriptor or if in static assignment mode, this bit should be set to prevent AFEU from reperforming this permutation step. 0 Perform S-box permutation 1 Do not permute

4.4 MDEU Mode Register

The MDEU mode register, shown in Figure 8, contains 8 bits which are used to program the MDEU. It also reflects the value of burst size, which is loaded by the crypto-channel during normal operation with the MPC184 as an initiator. Burst size is not relevant to target mode operations, where an external host pushes and pulls data from the execution units.

The mode register is cleared when the MDEU is reset or re-initialized. Setting a reserved mode bit will generate a data error. If the mode register is modified during processing, a context error will be generated.

Figure 8 shows the MDEU mode register.

	31–11	10–8	7	6–5	4	3	2	1–0
Field	Reserved	Burst Size	Cont	--	INT	HMAC	PD	ALG
Reset	0							
R/W	R/W							
Addr	MDEU 0x0C000							

	31–0
Field	Reserved
Reset	0
R/W	R/W
Addr	MDEU 0x0C004

Figure 8. MDEU Mode Register

Table 6 describes the MDEU mode register signals.

Table 6. MDEU Mode Register

Bits	Signal	Description
31:11	—	Reserved
10:8	Burst size	The MPC184 implements flow control to allow larger than FIFO sized blocks of data to be processed with a single key/context. The MDEU signals to the crypto-channel that a 'burst size' amount of data is available to be pushed to the FIFO. **Note:** The inclusion of this field in the MDEU mode register is to avoid confusing a user who may read this register in debug mode. Burst size should not be written directly to the MDEU.
7	Cont	Continue (Cont)—Used during HMAC/HASH processing when the data to be hashed is spread across multiple descriptors. 0 Don't Continue—operate the MDEU in auto completion mode 1 Preserve context to operate the MDEU in continuation mode
6:5	—	Reserved
4	INT	Initialization Bit (INT)—Cause an algorithm-specific initialization of the digest registers. Most operations will require this bit to be set. Only static operations that are continuing from a known intermediate hash value would not initialize the registers. 0 Do not initialize 1 Initialize the selected algorithm's starting registers
3	HMAC	Identifies the hash operation to execute: 0 Perform standard hash 1 Perform HMAC operation. This requires a key and key length information.

Table 6. MDEU Mode Register (continued)

Bits	Signal	Description
2	PD	If set, configures the MDEU to automatically pad partial message blocks. 0 Do not autopad 1 Perform automatic message padding whenever an incomplete message block is detected
1:0	ALG	Message digest algorithm selection 00 SHA-160 algorithm (full name for SHA-1) 01 SHA-256 algorithm 10 MD5 algorithm 11 Reserved

4.4.1 Recommended Settings for MDEU Mode Register

The most common task likely to be executed via the MDEU is HMAC generation. HMACs are used to provide message integrity within a number of security protocols, including IPSec and SSL/TLS. When the HMAC is being generated by a single dynamic descriptor (the MDEU acting as sole or secondary EU), the following mode register bit settings should be used:

Continue—Off

Initialize—On

HMAC—On

Autopad—On

When the HMAC is being generated for a message that is spread across a chain of static descriptors, the following mode register bit settings should be used:

- First Descriptor:

  Continue—On

  Initialize—On

  HMAC—On

  Autopad—Off

- Middle Descriptor(s):

  Continue—On

  Initialize—Off

  HMAC—Off

  Autopad—Off

- Final Descriptor

  Continue—Off

  Initialize—Off

  HMAC—On

  Autopad—On

Additional information on descriptors can be found in Chapter 6 of the *MPC184 Security Co-Processor User's Manual, PCI Interface*.

4.5 RNG Mode Register

The RNG mode register is used to control the RNG. One operational mode, randomizing, is defined. Writing any other value than 0 to 7:0 results in a data error interrupt that is reflected in the RNG interrupt status register. The mode register also reflects the value of burst size, which is loaded by the crypto-channel during normal operation with the MPC184 as an initiator. Burst size is not relevant to target mode operations, where an external host pushes and pulls data from the execution units.

The mode register is cleared when the RNG is reset or re-initialized. The RNG mode register is shown in Figure 9.

	31 … 11	10 … 8	7 … 0
Field	Reserved	Burst Count	Reserved
Reset	0		
R/W	R/W		
Addr	0x0E000		

	31 … 0
Field	Reserved
Reset	0
R/W	R/W
Addr	0x0E004

Figure 9. RNG Mode Register

Table 7 describes the RNG mode register signals.

Table 7. RNG Mode Register Definitions

Bits	Signal	Description
31:11	—	Reserved, must be set to zero.
10:8	Burst Count	The MPC184 implements flow control to allow larger than FIFO sized blocks of data to be processed with a single key/context. The RNG signals to the crypto-channel that a ‘burst size’ amount of data is available to be pulled from the FIFO. **Note:** The inclusion of this field in the RNG mode register is to avoid confusing a user who may read this register in debug mode. Burst size should not be written directly to the RNG.
7:0	—	Reserved

4.6 AESU Mode Register

The AESU mode register contains 4 bits which are used to program the AESU. It also reflects the value of burst size, which is loaded by the crypto-channel during normal operation with the MPC184 as an initiator. Burst size is not relevant to target mode operations, where an external host pushes and pulls data from the execution units.

The mode register is cleared when the AESU is reset or re-initialized. Setting a reserved mode bit will generate a data error. If the mode register is modified during processing, a context error will be generated. The AESU mode register is shown in Figure 10.

	31–11	10–8	7–4	3	2–1	0
Field	Reserved	Burst Size	Reserved	RDK	CM	ED
Reset	0					
R/W	R/W					
Addr	AESU 0x12000					

Figure 10. AESU Mode Register

Table 8 describes the AESU mode register signals.

Table 8. AESU Mode Register Signals

Bits	Signal	Description
31:11	—	Reserved
10:8	Burst size	The MPC184 implements flow control to allow larger than FIFO sized blocks of data to be processed with a single key/context. The AESU signals to the crypto-channel that a 'burst size' amount of data is available to be pushed to or pulled from the FIFO. **Note:** The inclusion of this field in the AESU mode register is to avoid confusing a user who may read this register in debug mode. Burst size should not be written directly to the AESU.
7:4	—	Reserved
3	RDK	Restore Decrypt Key (RDK)—Specifies that key data write will contain pre-expanded key (decrypt mode only). See note on use of RDK bit. 0 Expand the user key prior to decrypting the first block 1 Do not expand the key. The expanded decryption key will be written following the context switch.
2-1	CM	Cipher Mode: Controls which cipher mode the AESU will use in processing: 00 ECB—Electronic codebook mode 01 CBC—Cipher block chaining mode 10 Reserved 11 CTR—Counter mode
0	Encrypt/Decrypt	If set, AESU operates the encryption algorithm; if not set, AESU operates the decryption algorithm. 0 Perform decryption 1 Perform encryption

NOTE: Restore Decrypt Key

In most networking applications, the decryption of an AES protected packet will be performed as a single operation. However, if circumstances dictate that the decryption of a message should be split across multiple descriptors, the AESU allows the user to save the decrypt key, and the active AES context, to memory for later re-use. This saves the internal AESU processing overhead associated with regenerating the decryption key schedule (approximately 12 AESU clock cycles for the first block of data to be decrypted).

The use of RDK is completely optional, as the input time of the preserved decrypt key may exceed the approximate 12 cycles required to restore the decrypt key for processing the first block.

To use RDK, the following procedure is recommended:

- The descriptor type used in decryption of the first portion of the message is ‘0100—AESU Key Expand Output.’ The description mode must be ‘Decrypt.’ See Chapter 4 in the *MPC184 Security Co-Processor User’s Manual, PCI Interface*, for more information. The descriptor will cause the MPC184 to write the contents of the context and key registers (containing the expanded decrypt key) to memory.
- To process the remainder of the message, use a ‘normal’ descriptor type (descriptor type selected based on the need for simultaneous HMAC generation, etc.), and set the ‘restore decrypt key’ mode bit. Load the context registers and the expanded decrypt key with previously saved key and context data from the first message. The key size is written as before (16, 24, or 32 bytes).

5 Descriptor Type Field

The MPC184 accepts 13 fixed format descriptors. The descriptor type field in the descriptor header informs the crypto-channel of the ordering of the inputs and outputs defined by the length/pointer pairs in the descriptor body. The MPC190 (a previous Motorola security co-processor with mastering capability) allowed the user to define (within limits) the order in which keys, context, and data were fetched by the MPC190 prior to processing. The MPC184 descriptor type field advises the crypto-channel of the predetermined ordering of keys, context, and null fields. The ordering of inputs and outputs in the length/pointer pairs (as defined by descriptor type) is shown in Table 10.

Table 9 shows the permissible values for the descriptor type field in the descriptor header. Note that not all descriptor types are operationally useful, some exist for test and debug reasons, and to provide flexibility in dealing with evolving security standards. The cryptographic transforms required by most security protocols use types 0001 and 0010.

Table 9. Descriptor Types

Value	Descriptor Type	Notes
0000	Reserved	—
0001	common_nonsnoop_no_afeu	Common, nonsnooping, non-PKEU, non-AFEU
0010	hmac_snoop_no_afeu	Snooping, HMAC, non-AFEU
0011	non_hmac_snoop_no_afeu	Snooping, non-HMAC, non-AFEU
0100	aseu_key expand_output	Non-snooping, non HMAC, AESU, expanded key out
0101	common_nonsnoop_afeu	Common, nonsnooping, AFEU
0110	hmac_snoop_afeu	Snooping, HMAC, AFEU (no context out)
0111	non_hmac_snoop_afeu	Snooping, non-HMAC, AFEU
1000	pkeu_mm	PKEU-MM
1001	pkeu_ec	PKEU-EC
1010	pkeu_static_ec_point	PKEU static-EC point (completes operand loading and executes)

Table 9. Descriptor Types (continued)

1011	pkeu_static_ec_parameter	PKEU static-EC parameter (preloads EC operands)
1100	Reserved	—
1101	Reserved	—
1110	hmac_snoop_afeu_ key_in	AFEU context out available
1111	hmac_snoop_afeu_ctx_in	AFEU context out available

Table 10 shows how the length/pointer pairs should be used with the various descriptor types to load keys, context, and data into the execution units, and how the required outputs should be unloaded. Note that some outputs are optional.

Table 10. Descriptor Length/Pointer Mapping

Descriptor Type	L/P 1	L/P 2	L/P 3	L/P 4	L/P 5	L/P 6	L/P 7
0000	Null	Null	Null	Null	Null	Null	Null
0001	Null	IV	Key	Data in	Data out	IV out	MAC out
0010	HMAC key	HMAC data	Key	IV	Data in	Data out	HMAC/context out
0011	MD Ctx in	IV	Key	Data in	Data out	IV out	MD/context out
0100	Null	IV	Key	Data in	Data out	IV out	Key out via FIFO
0101	Null	IV in via FIFO	Key	Data in	Data out	IV out via FIFO	MD/context out
0110	HMAC key	HMAC data	Key	IV in via FIFO	Data in	Data out	HMAC/context out
0111	MD Ctx in	IV in via FIFO	Key	Data in	Data out	IV out via FIFO	MD/context out
1000	B	A	E	N	B out	Null	Null
1001	B	A	Key	N	B1 out	Null	Null
1010	A0	A1	A2	B1 out	B2 out	B3 out	Null
1011	A3	B0	B1	Key	N	Null	Null
1100	Null	Null	Null	Null	Null	Null	Null
1101	Null	Null	Null	Null	Null	Null	Null
1110	HMAC key	HMAC data	Key	Data in	Data out	IV out via FIFO	HMAC/context out
1111	HMAC key	HMAC data	IV	Data in	Data out	IV out via FIFO	HMAC/context out

5.1 Descriptor Type 0001

Descriptor type 0001 is used for a wide variety of functions, most of which do not require all the length/pointer fields to be used. A few non-obvious uses of this descriptor type are highlighted in Table 11.

Table 11. Descriptor Type 0001 Length/Pointer Mapping

Descriptor Type	L/P 1	L/P 2	L/P 3	L/P 4	L/P 5	L/P 6	L/P 7	Use
0001	Null	Null	Null	Null	Data out	Null	Null	RNG only
0001	Null	Ctx-in (opt)	Null	Data in	Null	Hash out	Null	Hash only
0001	Null	Ctx-in (opt)	HMAC Key	Data in	Null	HMAC out	Null	HMAC only
0001	Null	IV	Key	Data in	Data out	IV out	MAC out	Self integrity checking operations

For RNG operations, there is no key, context, or data to send in to the MPC184, so the only relevant pointer is the one which causes random data to be written from the RNG output FIFO to memory.

For HMAC only operations, the HMAC key should be loaded, followed by the data. The HMAC itself is written out via L/P 6. If an HMAC calculation is spread across multiple descriptors, all descriptors after the first would need to load the MDEU context registers via L/P 2. This requires the first descriptor to output the MDEU context or message digest, rather than an HMAC, with L/P 6.

Certain protocols do not rely on the HMAC function provided by the MDEU to generate MACs, or message integrity check values.

5.2 Snoop Type Bit

As mentioned in Table 1, bit 1 controls the type of ‘snooping’ which must occur between the primary and secondary EU. The rationale for ‘in-snooping’ vs. ‘out-snooping’ is found in security protocols which perform both encryption and integrity checking, such IPSec. When transmitting an IPSec ESP packet, the encapsulator must encrypt the packet payload, then calculate an HMAC over the header plus encrypted payload. Because the MDEU cannot generate the HMAC without the output of the primary EU (the one performing encryption, typically the DEU or AESU), the MDEU must ‘out-snoop.’

When receiving an IPSec packet, the decapsulator must calculate the HMAC over the encrypted portion of the packet prior to decryption. This allows the MDEU to source its data from the input FIFO of the primary EU, without waiting for the primary EU to finish its task.

Note that slightly different portions of an IPSec packet would pass through the primary and secondary EUs, in both the in-snooping and out-snooping cases. These offsets are dealt with by providing different starting pointers and byte lengths to the channel in the body of the descriptor.

An overview of the snooping concept is shown in Figure 11.

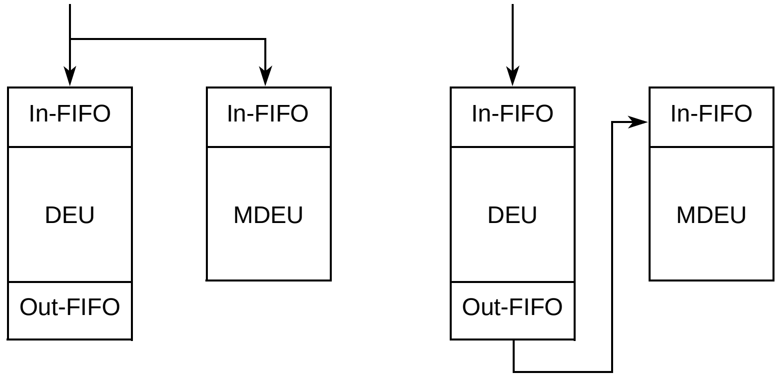

Figure 11. Snooping

5.3 Done Notification Bit

The done notification bit in the MPC184 descriptor header acts as a manual override to the crypto-channel configuration register's NOTIFICATION_TYPE bit. The NOTIFICATION_TYPE bit determines whether the MPC184 will advise the system (via interrupt or header writeback) that it is DONE with an operation after every descriptor, or after a chain of descriptors. Setting the notification bit in the descriptor header is unnecessary and redundant if NOTIFICATION_TYPE is set to 'end-of-descriptor,' but if set to 'end-of-chain,' the notification bit in the header can be quite useful as an intermediate notification.

The DONE notification can take the form of an interrupt or modified header writeback or both, depending on the state of the INTERRUPT_ENABLE and WRITEBACK_ENABLE control bits in the crypto-channel configuration register.

When the channel signals DONE via header writeback, the least significant byte (little endian) of the original header (at its original location in system memory) will always read as set to 0xFF, and the remaining 24 bits will not be modified. MPC184-initiated PCI writes can occur only on 64-bit word boundaries, but reads can occur on any byte boundary. Writing back a header read from a non-64-bit word boundary will yield unpredictable results.

6 Descriptor Length and Pointer Fields

The length and pointer fields represent one of seven data length/pointer pairs. Each pair defines a block of data in system memory. The length field gives the length of the block in bytes. The maximum allowable number of bytes is 32 Kbytes. A value of zero loaded into the length field indicates that this length/pointer pair should be skipped and processing continue with the next pair.

The pointer field contains the address, in PCI address space, of the first byte of the data block. Transfers from the PCI bus with the pointer address set to zero will have the length value written to the EU, and no data fetched from the PCI bus.

NOTE

Certain public key operations require information about data length, but not the data itself. Figure 12 shows the descriptor length field.

	31 ... 16	15 ... 0
Field	Reserved	Data Length Field
Reset	0	
R/W	R/W	

Figure 12. Descriptor Length Field

Table 12 shows the descriptor length field mapping.

Table 12. Descriptor Length Field Mapping

Bits	Name	Reset Value	Description
31:16	—	0	Reserved, set to zero
15:0	Data field length	0	**Note:** The maximum length this field can be set to 32 Kbytes. Under host control, a channel can be temporarily locked static, and 'data only' descriptors can be chained to fetch blocks larger than 32 Kbytes in 32-Kbyte sub-blocks without key/context switching, until the large original block has been completely ciphered. Length fields also indicate the size of items to be written back to memory on completion of security processing in the MPC184.

Figure 13 shows the descriptor pointer field.

	31 ... 0
Field	Data Field X Pointer
Reset	0
R/W	R/W

Figure 13. Descriptor Pointer Field

Table 13 shows the descriptor pointer field mapping.

Table 13. Descriptor Pointer Field Mapping

Bits	Name	Reset Value	Description
31:0	Data field pointer	0	The data pointer field contains the address, in PCI address space, of the first byte of the data packet for either read or writeback. Transfers from the PCI bus with pointer address set to zero will be skipped. **WARNING** MPC184-initiated PCI writes can occur only on 64-bit word boundaries, but reads can occur on any byte boundary. Writing back a header read from a non-64-bit word boundary will yield unpredictable results.

Following the length/pointer pairs is the next descriptor field, which contains the pointer to the next descriptor in memory. On completion of processing of the current descriptor, this value, if non-zero, is used to request a PCI burst read of the next-data-packet descriptor. This automatic load of the next descriptor is referred to as descriptor chaining. Figure 14 displays the next descriptor pointer field.

	31 … 0
Field	Next Descriptor Pointer
Reset	0
R/W	R/W

Figure 14. Next Descriptor Pointer Field

Table 14 describes the descriptor pointer field mapping.

Table 14. Descriptor Pointer Field Mapping

Bits	Name	Reset Value	Description
31:0	Next descriptor pointer	0	The next descriptor pointer field contains the address, in PCI address space, of the next descriptor to be fetched if descriptor chaining is enabled. **WARNING** The next descriptor pointer address must be modulo-8 aligned if writeback is enabled as the method of DONE notification.

7 Descriptor Chaining

Descriptor chaining provides a measure of decoupling between host CPU activities and the status of the MPC184. Rather than waiting for the MPC184 to signal DONE, and arbitrating for the PCI bus in order to write directly to the next-data-packet descriptor in the crypto-channel, the host can simply create new descriptors in memory, and chain them to descriptors which have not yet been fetched by the MPC184 by filling the next-data-packet field with the address of the newly created descriptor. Whether or not processing continues automatically following next-descriptor fetch and whether or not an interrupt is generated depends on the programming of the crypto-channel's configuration register.

See Section 7.1.1, "Crypto-Channel Configuration Register (CCCR)," in the *MPC184 Security Co-Processor User's Manual, PCI Interface*, for additional information on how the MPC184 can be programmed to signal and act on completion of a descriptor.

NOTE

It is possible to insert a descriptor into an existing chain; however, great care must be taken when doing so.

Figure 15 shows a conceptual chain, or linked list, of descriptors.

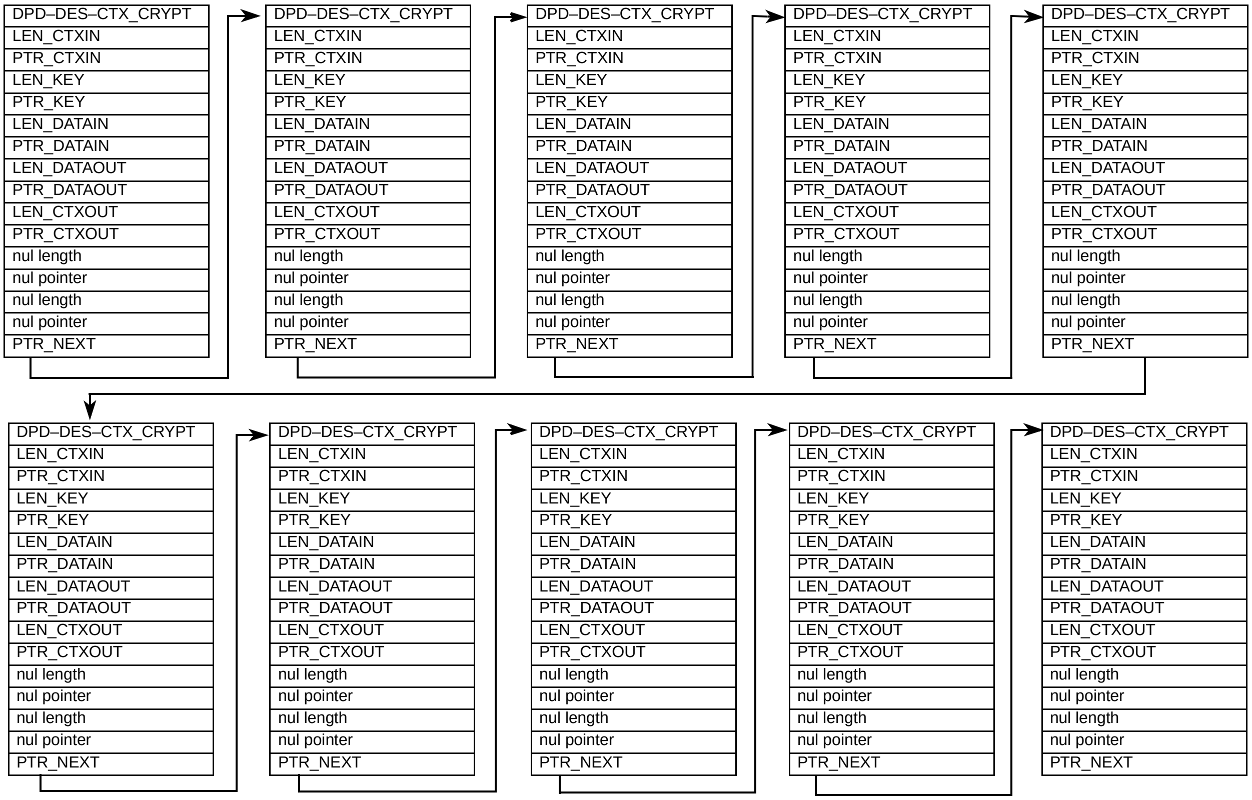

Figure 15. Chain of Descriptors

7.1 Null Fields

On occasion, a descriptor field may not be applicable to the requested service. With seven length/pointer pairs, it is possible that not all descriptor fields will be required to load the required keys, context, and data. (Some operations don't require context, others may only need to fetch a small, contiguous block of data.) Therefore, when processing data packet descriptors, the MPC184 will skip any pointer entirely that has an associated length of zero.

8 Descriptor Classes

The MPC184 has two general classes of descriptors: static, which refers to a relatively unchanging usage of MPC184 resources, and dynamic, which refers to a continually changing usage model.

8.1 Static Descriptors

Recall that the MPC184 has six execution units and four crypto-channels. The EUs can be statically assigned, dedicating them to a particular crypto-channel. Certain combinations of EUs can be statically assigned to the same crypto-channel to facilitate multi-operation security processes, such as IPSec ESP mode. When the system traffic model permits its use, static assignment can offer significant performance improvements over dynamic assignment by avoid key and context switching per packet.

Static descriptors split the operations to be performed during a security operation into separate descriptors. The first descriptor is typically only used to set the EU mode, and load the key and context. The second (and multiple subsequent) descriptor contains length/pointer pairs to the data to be permuted. Because the key and context are unchanging over multiple packets (or descriptors), the series of short reads and writes required to setup and tear down a session are avoided. This savings, along with the crypto-channel having dedicated execution units, can represent a noticeable performance improvement.

Note that there is no mechanism for resetting an EU automatically when statically assigned, or when assignment is changed from static to dynamic. Therefore, it is recommended that the drivers always reset an EU just prior to removing a static assignment to it to prevent the previously used context from polluting another encryption stream.

For example, statically assigning a DEU to a particular crypto-channel permits the DEU to retain context between data packets. The following descriptors, listed in Table 15 through Table 17, support context retention. Table 15 defines the first DPD_3DES_CBC_Encrypt descriptor in the static chain.

Table 15. Actual Descriptor DPD_Type 0001_3DES_CBC_Encrypt

Field	Value/Type	Description
Header	0x2070_0010	DPD_Type 0001_3DES_CBC_Encrypt
LEN_1	Length	Null
PTR_1	Pointer	Null
LEN_2	Length	Number of bytes of IV to be written to DEU IV register (always 8)
PTR_2	Pointer	PCI address of IV
LEN_3	Length	Number of bytes of key to be written to DEU key register (must be 16 or 24)
PTR_3	Pointer	PCI address of key
LEN_4	Length	Number of bytes to be ciphered
PTR_4	Pointer	PCI address of data to be ciphered
LEN_5	Length	Bytes to be written (should be equal to length of data-in)
PTR_5	Pointer	PCI address where ciphered data is to be written
LEN_6	Nul	Null
PTR_6	Nul	Null
LEN_7	Nul	Null
PTR_7	Nul	Null
PTR_NEXT	Pointer	Pointer to next descriptor

Table 16 defines the second (or *N* middle) DPD_3DES_CBC_Encrypt descriptor in the static chain. Note that the IV and key are not loaded, as they remain in the DEU key and IV register.

Table 16. Actual Descriptor DPD_Type 0001_3DES_CBC_Encrypt

Field	Value/Type	Description
Header	0x2070_0010	DPD_Type 0001_3DES_CBC_Encrypt
LEN_1	Length	Null
PTR_1	Pointer	Null

Table 16. Actual Descriptor DPD_Type 0001_3DES_CBC_Encrypt (continued)

Field	Value/Type	Description
LEN_2	Length	Null
PTR_2	Pointer	Null
LEN_3	Length	Null
PTR_3	Pointer	Null
LEN_4	Length	Number of bytes to be ciphered
PTR_4	Pointer	PCI address of data to be ciphered
LEN_5	Length	Bytes to be written (should be equal to length of data-in)
PTR_5	Pointer	PCI address where ciphered data is to be written
LEN_6	Nul	Null
PTR_6	Nul	Null
LEN_7	Nul	Null
PTR_7	Nul	Null
PTR_NEXT	Pointer	Pointer to next descriptor

Table 17 defines the final DPD_3DES_CBC_Encrypt descriptor in the static chain. Note that the IV and key are not loaded, as they remain in the DEU key and IV register. The IV may be optionally unloaded at the conclusion of the descriptor. On completion of this descriptor, the EU should be reset, and the EU should be released via a write to the EU assignment control register in the controller.

Table 17. Actual Descriptor DPD_Type 0001_3DES_CBC_Encrypt

Field	Value/Type	Description
Header	0x2070_0010	DPD_Type 0001_3DES_CBC_Encrypt
LEN_1	Length	Null
PTR_1	Pointer	Null
LEN_2	Length	Null
PTR_2	Pointer	Null
LEN_3	Length	Null
PTR_3	Pointer	Null
LEN_4	Length	Number of bytes to be ciphered
PTR_4	Pointer	PCI address of data to be ciphered
LEN_5	Length	Bytes to be written (should be equal to length of data-in)
PTR_5	Pointer	PCI address where ciphered data is to be written
LEN_6	Length	(Optional) Number of bytes of IV to be written to PCI memory space (always 8)
PTR_6	Pointer	(Optional) PCI address where IV is to be written
LEN_7	Nul	Null

Table 17. Actual Descriptor DPD_Type 0001_3DES_CBC_Encrypt (continued)

Field	Value/Type	Description
PTR_7	Nul	Null
PTR_NEXT	Pointer	Pointer to next descriptor

8.2 Dynamic Descriptors

In a typical networking environment, packets from innumerable sessions arrive fairly randomly. The host must determine which security association applies to the current packet and encrypt or decrypt without any knowledge of the security association of the previous or next packet. This situation calls for the use of dynamic descriptors.

When under dynamic assignment, an EU must be used under the assumption that a different crypto-channel (with a different context) may have just used the EU and that another crypto-channel (with yet another context) may use that EU immediately after the current crypto-channel has released the EU.

The descriptor shown in Table 18 completely sets up the DEU for an encryption operation; loads the keys, context, and data; writes the permuted data back to memory; and (optionally) writes the altered context (IV) back to memory. (This may be necessary when DES is operating in CBC mode.) On completion of the descriptor, the DEU is automatically cleared and released.

Table 18. Representative Descriptor DPD_Type 0001_3DES_CBC_Encrypt

Field	Value/Type	Description
Header	0x2070_0010	DPD_Type 0001_3DES_CBC_Encrypt
LEN_1	Length	Null
PTR_1	Pointer	Null
LEN_2	Length	Number of bytes of IV to be written to DEU IV register (always 8)
PTR_2	Pointer	PCI address of IV
LEN_3	Length	Number of bytes of key to be written to DEU key register (must be 16 or 24)
PTR_3	Pointer	PCI address of key
LEN_4	Length	Number of bytes to be ciphered
PTR_4	Pointer	PCI address of data to be ciphered
LEN_5	Length	Bytes to be written (should be equal to length of data-in)
PTR_5	Pointer	PCI address where ciphered data is to be written
LEN_6	Length	(Optional) Number of bytes of IV to be written to PCI memory space (always 8)
PTR_6	Pointer	(Optional) PCI address where IV is to be written
LEN_7	Nul	Null
PTR_7	Nul	Null
PTR_NEXT	Pointer	Pointer to next descriptor

Note that the descriptor header value is the same as the value used in the static assignment example. The descriptor header does not determine static vs. dynamic assignment (this is a difference from the MPC190). In the MPC184, static assignment is entirely controlled by the EU assignment control register in the controller (see Chapter 8 in the *MPC184 Security Co-Processor User's Manual*, *PCI Interface*, for more information on the EUACR.) When an EU is statically assigned to a channel, it will use keys and context from the current descriptor for the following descriptor, until the EU is reset and released from static assignment. Releasing an EU, then resetting it, is not recommended, as any channel with an outstanding request for an EU of the type being released could be dynamically assigned the EU before the previous key and context was cleared by the reset. When a channel has been dynamically assigned an EU, the channel will automatically reset the EU before releasing it for use by another channel.

9 Additional Examples

In the following sections are descriptor examples of some common cryptographic transforms. Also provided are tables of derivative descriptor headers for closely related transforms.

9.1 Dynamically Assigned 3DES-HMAC-SHA-1 Decrypt (Inbound IPSec ESP)

Table 19 shows a dynamic descriptor example of an inbound IPSec ESP transform.

Table 19. Representative Descriptor DPD_Type 0010_3DES_CBC_HMAC_SHA-1_Decrypt

Field	Value/Type	Description
Header	0x2063_1C22	DPD_Type 0010_3DES_CBC_HMAC_SHA-1_Decrypt
LEN_1	Length	Number of bytes of HMAC key to be written to MDEU key register
PTR_1	Pointer	PCI address of HMAC key
LEN_2	Length	Number of bytes to be HMAC'd
PTR_2	Pointer	PCI address of data to be HMAC'd
LEN_3	Length	Number of bytes of key to be written to DEU key register (must be 16 or 24)
PTR_3	Pointer	PCI address of key
LEN_4	Length	Number of bytes of IV to be written to DEU IV register (always 8)
PTR_4	Pointer	PCI address of IV
LEN_5	Length	Number of bytes of ciphertext to be decrypted
PTR_5	Pointer	PCI address of ciphertext to be decrypted
LEN_6	Length	Number of bytes of plaintext to be written out to memory (should be equal to length of data-in)
PTR_6	Pointer	PCI address where plaintext is to be written
LEN_7	Length	Number of bytes of HMAC to be written to PCI memory space (always 20 for HMAC-SHA-1)

Table 19. Representative Descriptor DPD_Type 0010_3DES_CBC_HMAC_SHA-1_Decrypt (continued)

Field	Value/Type	Description
PTR_7	Pointer	PCI address where HMAC is to be written
PTR_NEXT	Pointer	Pointer to next descriptor

The descriptor header encodes the information required to select the DEU for Op_0, and the MDEU for Op_1. The Op_0 mode data configured the DEU to operate in 3DES, CBC, decrypt mode. The Op_1 mode data configured the MDEU to operate in HMAC-SHA-1 mode. Because all the data necessary to calculate the HMAC in a single dynamic descriptor is available, initialize, and autopad are set, while continue is off.

The descriptor header also encodes the descriptor type 0010, which defines the input and output ordering for 'hmac_snoop_no_afeu.' The HMAC key is loaded first, followed by the length and pointer to the data over which the HMAC will be calculated. The 3DES key is loaded next, followed by the 3DES IV. The number of bytes to be ciphered and starting address will be an offset of the number of bytes being HMAC'd. The data to be decrypted and HMAC'd is only brought into the MPC184 a single time, with the DEU and MDEU only reading the portion that matches the starting address and byte length in the length/pointer fields corresponding to their data of interest.

Ciphertext is brought into the DEU input FIFO, with the MDEU in-snooping the portion of the data it has been told to process. As the decryption continues, the plaintext fills the DEU output FIFO, and this data is written back to system memory as needed. When the final byte of data to be HMAC'd has been processed through the MDEU, the descriptor will cause the MDEU to write the HMAC to the indicated area in PCI memory. The MPC184 will write the entire 20 bytes HMAC-SHA-1 to PCI memory, and the host will compare the most significant 12 bytes of the HMAC generated by the MPC184 with the HMAC which was received with the inbound packet. If the HMACs match, the packet integrity check passes.

The next descriptor pointer is optional, and if a next descriptor is indicated, that descriptor may be completely unrelated to the operation performed by the descriptor shown in Table 19.

9.2 Dynamically Assigned 3DES-HMAC-SHA-1 Encrypt (Outbound IPSec ESP)

Table 20 shows a dynamic descriptor example of an outbound IPSec ESP transform.

Table 20. Representative Descriptor DPD_Type 0010_3DES_CBC_HMAC_SHA-1_Decrypt

Field	Value/Type	Description
Header	0x2073_1C20	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Encrypt
LEN_1	Length	Number of bytes of HMAC key to be written to MDEU key register
PTR_1	Pointer	PCI address of HMAC key
LEN_2	Length	Number of bytes to be HMAC'd
PTR_2	Pointer	PCI address of data to be HMAC'd
LEN_3	Length	Number of bytes of key to be written to DEU key register (must be 16 or 24)
PTR_3	Pointer	PCI address of key

Table 20. Representative Descriptor DPD_Type 0010_3DES_CBC_HMAC_SHA-1_Decrypt (continued)

Field	Value/Type	Description
LEN_4	Length	Number of bytes of IV to be written to DEU IV register (always 8)
PTR_4	Pointer	PCI address of IV
LEN_5	Length	Number of bytes of plaintext to be encrypted
PTR_5	Pointer	PCI address of plaintext to be encrypted
LEN_6	Length	Number of bytes of ciphertext to be written out to memory (should be equal to length of data-in)
PTR_6	Pointer	PCI address where ciphertext is to be written
LEN_7	Length	Number of bytes of HMAC to be written to PCI memory space (always 20)
PTR_7	Pointer	PCI address where HMAC is to be written
PTR_NEXT	Pointer	Pointer to next descriptor

The descriptor header encodes the information required to select the DEU for Op_0, and the MDEU for Op_1. The Op_0 mode data configured the DEU to operate in 3DES, CBC, encrypt mode. The Op_1 mode data configured the MDEU to operate in HMAC-SHA-1 mode. Because all the data necessary to calculate the HMAC in a single dynamic descriptor is available, initialize, and autopad are set, while continue is off.

The descriptor header also encodes the descriptor type 0010, which defines the input and output ordering for ‘hmac_snoop_no_afeu.’ The HMAC key is loaded first, followed by the length and pointer to the data over which the HMAC will be calculated. The 3DES key is loaded next, followed by the 3DES IV. The number of bytes to be encrypted and starting address will be an offset of the number of bytes being HMAC’d. The data to be encrypted and HMAC’d is only brought into the MPC184 a single time, with the DEU and MDEU only reading the portion that matches the starting address and byte length in the length/pointer fields corresponding to their data of interest.

Plaintext is brought into the DEU input FIFO, with the MDEU out-snooping the portion of the data it has been told to process. As the encryption continues, the ciphertext fills the DEU output FIFO, and this data is written back to system memory as needed. When the final byte of data to be HMAC’d has been processed through the MDEU, the descriptor will cause the MDEU to write the HMAC to the indicated area in PCI memory. The MPC184 will write the entire 20 bytes HMAC-SHA-1 to PCI memory, and the host will append the most significant 12 bytes of the HMAC generated by the MPC184 to the packet as the authentication trailer. Common practice in IPSec ESP with 3DES-CBC is to use the last 8 bytes of the ciphertext as the IV for the next packet. If this is the case, the host should copy the last 8 bytes of the ciphertext to the Security Association database entry for this particular session before transmitting the packet.

The next descriptor pointer is optional, and if a next descriptor is indicated, that descriptor may be completely unrelated to the operation performed on the descriptor shown in Table 20.

9.3 Dynamically Assigned HMAC-MD-5 (Inbound/Outbound IPSec AH)

Table 21 shows a dynamic descriptor example of an inbound/outbound IPSec AH transform.

Table 21. Representative Descriptor DPD_Type 0001_HMAC-MD-5

Field	Value / Type	Description
Header	0x31E0_0010	DPD_Type 0001_HMAC_MD-5
LEN_1	Length	Null
PTR_1	Pointer	Null
LEN_2	Length	Null
PTR_2	Pointer	Null
LEN_3	Length	Number of bytes of HMAC key to be written to MDEU key register
PTR_3	Pointer	PCI address of HMAC key
LEN_4	Length	Number of bytes of data to be written to MDEU input FIFO
PTR_4	Pointer	PCI address of data
LEN_5	Length	Null
PTR_5	Pointer	Null
LEN_6	Length	Number of bytes of HMAC to be written out to memory (always 16 MD-5)
PTR_6	Pointer	PCI address where HMAC is to be written
LEN_7	Length	Null
PTR_7	Pointer	Null
PTR_NEXT	Pointer	Pointer to next descriptor

The descriptor header encodes the information required to select the MDEU for Op_0, and no EU for Op_1. The Op_0 mode data configured the MDEU to operate in HMAC-MD-5 mode. Because all the data necessary to calculate the HMAC in a single dynamic descriptor is available, initialize, and autopad are set, while continue is off.

The descriptor header also encodes the descriptor type 0001, which defines the input and output ordering for ‘common_nonsnoop_no_afeu.’ This is the descriptor type used for most operations which don’t require a secondary EU. Following some null pointers, the HMAC key is loaded, followed by the length and pointer to the data over which the HMAC will be calculated.

The data is brought into the MDEU input FIFO, and when the final byte of data to be HMAC’d has been processed through the MDEU, the descriptor will cause the MDEU to write the HMAC to the indicated area in PCI memory. The MPC184 will write the entire 16 bytes HMAC-MD-5 to PCI memory, and depending on whether the packet is inbound or outbound, the host will either insert the most significant 12 bytes of the HMAC generated by the MPC184 into the packet header (outbound) or compare the HMAC generated by the MPC184 with the HMAC which was received with the inbound packet (obviously inbound). If the HMACs match, the packet integrity check passes.

The next descriptor pointer is optional, and if a next descriptor is indicated, that descriptor may be completely unrelated to the operation performed by the descriptor shown in Table 21.

Table 22 shows today’s most commonly used IPSec descriptor headers. In all the descriptor headers shown, the MDEU performs auto padding.

Table 22. Common IPSec Dynamic Descriptor Headers

Value/Type	Description
0x2003_1E22	DPD_Type 0010_DES_ECB_HMAC_MD-5 Decrypt
0x2013_1E20	DPD_Type 0010_DES_ECB_HMAC_MD-5 Encrypt
0x2003_1C22	DPD_Type 0010_DES_ECB_HMAC_SHA-1 Decrypt
0x2013_1C20	DPD_Type 0010_DES_ECB_HMAC_SHA-1 Encrypt
0x2043_1E22	DPD_Type 0010_3DES_ECB_HMAC_MD-5 Decrypt
0x2053_1E20	DPD_Type 0010_3DES_ECB_HMAC_MD-5 Encrypt
0x2043_1C22	DPD_Type 0010_3DES_ECB_HMAC_SHA-1 Decrypt
0x2053_1C20	DPD_Type 0010_3DES_ECB_HMAC_SHA-1 Encrypt
0x2023_1E22	DPD_Type 0010_DES_CBC_HMAC_MD-5 Decrypt
0x2033_1E20	DPD_Type 0010_DES_CBC_HMAC_MD-5 Encrypt
0x2023_1C22	DPD_Type 0010_DES_CBC_HMAC_SHA-1 Decrypt
0x2033_1C20	DPD_Type 0010_DES_CBC_HMAC_SHA-1 Encrypt
0x2063_1E22	DPD_Type 0010_3DES_CBC_HMAC_MD-5 Decrypt
0x2073_1E20	DPD_Type 0010_3DES_CBC_HMAC_MD-5 Encrypt
0x2063_1C22	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Decrypt
0x2073_1C20	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Encrypt
0x31C0_0010	DPD_Type 0001_HMAC_SHA-1
0x31D0_0010	DPD_Type 0001_HMAC_SHA-256
0x31E0_0010	DPD_Type 0001_HMAC_MD-5

Table 23 shows today’s AES descriptors as they will be used for IPSec and SRTP. In all the descriptor headers shown, the MDEU performs auto padding.

Table 23. Additional Multi-Op Dynamic Descriptor Headers

Value / Type	Description
0x6083_1E22	DPD_Type 0010_AES_ECB_HMAC_MD-5 Decrypt
0x6093_1E20	DPD_Type 0010_AES_ECB_HMAC_MD-5 Encrypt
0x6083_1C22	DPD_Type 0010_AES_ECB_HMAC_SHA-1 Decrypt
0x6093_1C20	DPD_Type 0010_AES_ECB_HMAC_SHA-1 Encrypt
0x6083_1D22	DPD_Type 0010_AES_ECB_HMAC_SHA-256 Decrypt
0x6093_1D20	DPD_Type 0010_AES_ECB_HMAC_SHA-256 Encrypt
0x60A3_1E22	DPD_Type 0010_AES_CBC_HMAC_MD-5 Decrypt
0x60B3_1E20	DPD_Type 0010_AES_CBC_HMAC_MD-5 Encrypt
0x60A3_1C22	DPD_Type 0010_AES_CBC_HMAC_SHA-1 Decrypt
0x60B3_1C20	DPD_Type 0010_AES_CBC_HMAC_SHA-1 Encrypt

Table 23. Additional Multi-Op Dynamic Descriptor Headers (continued)

Value / Type	Description
0x60A3_1D22	DPD_Type 0010_AES_CBC_HMAC_SHA-256 Decrypt
0x60B3_1D20	DPD_Type 0010_AES_CBC_HMAC_SHA-256 Encrypt
0x60E3_1E22	DPD_Type 0010_AES_CTR_HMAC_MD-5 Decrypt
0x60E3_1E20	DPD_Type 0010_AES_CTR_HMAC_MD-5 Encrypt
0x60E3_1C22	DPD_Type 0010_AES_CTR_HMAC_SHA-1 Decrypt
0x60E3_1C20	DPD_Type 0010_AES_CTR_HMAC_SHA-1 Encrypt
0x60E3_1D22	DPD_Type 0010_AES_CTR_HMAC_SHA-256 Decrypt
0x60E3_1D20	DPD_Type 0010_AES_CTR_HMAC_SHA-256 Encrypt

9.4 Statically Assigned 3DES-HMAC-SHA-1 Decrypt (Inbound IPSec ESP)

This example, shown in Table 24, is designed to contrast the dynamic descriptor shown in Table 19. For whatever reason, the data to be decrypted and authenticated is not available in a single contiguous block, or the total data size is larger than 32 Kbytes. The user must statically assign a DEU and MDEU to a channel before launching this descriptor chain.

The first descriptor loads the appropriate keys and context, while the N middle descriptors continue processing data. The final descriptor decrypts the final data, and allows the HMAC calculation to complete.

Table 24. Representative First Descriptor DPD_Type 0010_3DES_CBC_HMAC_SHA-1_Decrypt

Field	Value / Type	Description
Header	0x2063_9822	DPD_Type 0010_3DES_CBC_HMAC_SHA-1_Decrypt
LEN_1	Length	Number of bytes of HMAC key to be written to MDEU key register
PTR_1	Pointer	PCI address of HMAC key
LEN_2	Length	Number of bytes to be hashed
PTR_2	Pointer	PCI address of data to be hashed
LEN_3	Length	Number of bytes of key to be written to DEU key register (must be 16 or 24)
PTR_3	Pointer	PCI address of key
LEN_4	Length	Number of bytes of IV to be written to DEU IV register (always 8)
PTR_4	Pointer	PCI address of IV
LEN_5	Length	Number of bytes to be ciphered
PTR_5	Pointer	PCI address of data to be ciphered
LEN_6	Length	Bytes to be written (should be equal to length of data-in)
PTR_6	Pointer	PCI address where ciphered data is to be written
LEN_7	Nul	Null

Table 24. Representative First Descriptor DPD_Type 0010_3DES_CBC_HMAC_SHA-1_Decrypt (continued)

Field	Value / Type	Description
PTR_7	Nul	Null
PTR_NEXT	Pointer	Pointer to next descriptor

The first descriptor header encodes the information required to select the DEU for Op_0, and the MDEU for Op_1. The Op_0 mode data configured the DEU to operate in 3DES, CBC, decrypt mode. The Op_1 mode data configured the MDEU to operate in SHA-1 mode. Because all the data necessary to calculate the HMAC is not present, the first static descriptor is set to initialize, continue, and HMAC, while autopad is off.

The descriptor header also encodes the descriptor type 0010, which defines the input and output ordering for ‘hmac_snoop_no_afeu.’ The HMAC key is loaded first, followed by the length and pointer to the data over which the initial hash will be calculated. The 3DES key is loaded next, followed by the 3DES IV. The number of bytes to be ciphered and starting address will be an offset of the number of bytes being hashed. The data to be decrypted and hashed is only brought into the MPC184 a single time, with the DEU and MDEU only reading the portion that matches the starting address and byte length in the length/pointer fields corresponding to their data of interest.

Ciphertext is brought into the DEU input FIFO, with the MDEU snooping the portion of the data it has been told to process. As the decryption continues, the plaintext fills the DEU output FIFO, and this data is written back to system memory as needed. Because it has been told to expect more data (HMAC off, continue on), the descriptor must not attempt to output the contents of the MDEU message digest register.

The next descriptor pointer should point to the descriptor shown in Table 25.

Table 25. Representative Middle Descriptor DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Decrypt

Field	Value/Type	Description
Header	0x2063_8022	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 decrypt
LEN_1	Nul	Null
PTR_1	Nul	Null
LEN_2	Length	Number of bytes to be Hashed
PTR_2	Pointer	PCI address of data to be hashed
LEN_3	Nul	Null
PTR_3	Nul	Null
LEN_4	Nul	Null
PTR_4	Nul	Null
LEN_5	Length	Number of bytes to be ciphered
PTR_5	Pointer	PCI address of data to be ciphered
LEN_6	Length	Bytes to be written (should be equal to Length of data-in)
PTR_6	Pointer	PCI address where ciphered data is to be written
LEN_7	Nul	Null

Table 25. Representative Middle Descriptor DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Decrypt (continued)

Field	Value/Type	Description
PTR_7	Nul	Null
PTR_NEXT	Pointer	Pointer to next descriptor

The middle descriptor header encodes the information required to select the DEU for Op_0, and the MDEU for Op_1. The Op_0 mode data configured the DEU to operate in 3DES, CBC, decrypt mode. The Op_1 mode data configured the MDEU to operate in SHA-1 mode. Because all the data necessary to calculate the HMAC is still not present, the middle static descriptor is set to continue, while initialize, HMAC, and autopad are off.

The descriptor header also encodes the descriptor type 0010, which defines the input and output ordering for ‘hmac_snoop_no_afeu.’ The HMAC key is already loaded, and does not need to be reloaded. The length and pointer to the data over which the initial hash will be calculated must be provided for this descriptor. The 3DES key and IV are already loaded, and need not be reloaded.

Ciphertext is brought into the DEU input FIFO, with the MDEU snooping the portion of the data it has been told to process. As the decryption continues, the plaintext fills the DEU output FIFO, and this data is written back to system memory as needed. Because it has been told to expect more data (HMAC off, continue on), the descriptor must not attempt to output the contents of the MDEU message digest register.

The next descriptor pointer should point to the descriptor shown in Table 26.

Table 26. Representative Final Descriptor DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Decrypt

Field	Value/Type	Description
Header	0x2063_8C22	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 decrypt
LEN_1	Nul	Null
PTR_1	Nul	Null
LEN_2	Length	Number of bytes to be hashed
PTR_2	Pointer	PCI address of data to be hashed
LEN_3	Nul	Null
PTR_3	Nul	Null
LEN_4	Nul	Null
PTR_4	Nul	Null
LEN_5	Length	Number of bytes to be ciphered
PTR_5	Pointer	PCI address of data to be ciphered
LEN_6	Length	Bytes to be written (should be equal to length of data-in)
PTR_6	Pointer	PCI address where ciphered data is to be written
LEN_7	Nul	Null
PTR_7	Nul	Null
PTR_NEXT	Nul	Null

The final descriptor header encodes the information required to select the DEU for Op_0, and the MDEU for Op_1. The Op_0 mode data configured the DEU to operate in 3DES, CBC, decrypt mode. The Op_1 mode data configured the MDEU to operate in SHA-1 mode. Because the final data necessary to calculate the HMAC is now present, the final static descriptor is set to HMAC and autopad, while continue and initialize are off.

The descriptor header also encodes the descriptor type 0010, which defines the input and output ordering for ‘hmac_snoop_no_afeu.’ The HMAC key is already loaded, and doesn’t need to be reloaded. The length and pointer to the data over which the initial hash will be calculated must be provided for this descriptor. The 3DES key and IV are already loaded, and need not be reloaded.

Ciphertext is brought into the DEU input FIFO, with the MDEU snooping the portion of the data it has been told to process. As the decryption continues, the plaintext fills the DEU output FIFO, and this data is written back to system memory as needed. Because it has been told it has the final data for HMAC calculation (HMAC on, continue off), the descriptor must output the contents of the MDEU message digest register to the indicated address in system memory. The MPC184 will write the entire 20-byte HMAC-SHA-1 to PCI memory, and depending on the security protocol in question, the host will compare the most significant *x* bytes of the HMAC generated by the MPC184 with the HMAC sent with the packet.

The next descriptor pointer should be null, as the channel should not fetch another descriptor until the EUs have been reset. The static assignment of the current EUs need not end, if the channel is expected to need the same EUs to operate on a similar static chain belonging to a difference secure session.

Table 27 shows today’s most commonly used IPSec descriptor headers. In all the descriptor headers shown, the MDEU performs auto padding for the final data block, as needed.

Table 27. Common IPSec Static Descriptor Headers

Value/Type	Description
0x2003_9A22	DPD_Type 0010_DES_ECB_HMAC_MD-5 Decrypt First
0x2003_8222	DPD_Type 0010_DES_ECB_HMAC_MD-5 Decrypt Middle
0x2003_8E22	DPD_Type 0010_DES_ECB_HMAC_MD-5 Decrypt Last
0x2013_9A22	DPD_Type 0010_DES_ECB_HMAC_MD-5 Encrypt First
0x2013_8220	DPD_Type 0010_DES_ECB_HMAC_MD-5 Encrypt Middle
0x2013_8E20	DPD_Type 0010_DES_ECB_HMAC_MD-5 Encrypt Last
0x2003_9822	DPD_Type 0010_DES_ECB_HMAC_SHA-1 Decrypt First
0x2003_8022	DPD_Type 0010_DES_ECB_HMAC_SHA-1 Decrypt Middle
0x2003_8C22	DPD_Type 0010_DES_ECB_HMAC_SHA-1 Decrypt Last
0x2013_9820	DPD_Type 0010_DES_ECB_HMAC_SHA-1 Encrypt First
0x2013_8020	DPD_Type 0010_DES_ECB_HMAC_SHA-1 Encrypt Middle
0x2013_8C20	DPD_Type 0010_DES_ECB_HMAC_SHA-1 Encrypt Last
0x2043_9A22	DPD_Type 0010_3DES_ECB_HMAC_MD-5 Decrypt First
0x2043_8222	DPD_Type 0010_3DES_ECB_HMAC_MD-5 Decrypt Middle
0x2043_8E22	DPD_Type 0010_3DES_ECB_HMAC_MD-5 Decrypt Last

Table 27. Common IPSec Static Descriptor Headers (continued)

Value/Type	Description
0x2053_9A22	DPD_Type 0010_3DES_ECB_HMAC_MD-5 Encrypt First
0x2053_8220	DPD_Type 0010_3DES_ECB_HMAC_MD-5 Encrypt Middle
0x2053_8E20	DPD_Type 0010_3DES_ECB_HMAC_MD-5 Encrypt Last
0x2043_9822	DPD_Type 0010_3DES_ECB_HMAC_SHA-1 Decrypt First
0x2043_8022	DPD_Type 0010_3DES_ECB_HMAC_SHA-1 Decrypt Middle
0x2043_8C22	DPD_Type 0010_3DES_ECB_HMAC_SHA-1 Decrypt Last
0x2053_9820	DPD_Type 0010_3DES_ECB_HMAC_SHA-1 Encrypt First
0x2053_8020	DPD_Type 0010_3DES_ECB_HMAC_SHA-1 Encrypt Middle
0x2053_8C20	DPD_Type 0010_3DES_ECB_HMAC_SHA-1 Encrypt Last
0x2023_9A22	DPD_Type 0010_DES_CBC_HMAC_MD-5 Decrypt First
0x2023_8222	DPD_Type 0010_DES_CBC_HMAC_MD-5 Decrypt Middle
0x2023_8E22	DPD_Type 0010_DES_CBC_HMAC_MD-5 Decrypt Last
0x2033_9A22	DPD_Type 0010_DES_CBC_HMAC_MD-5 Encrypt First
0x2033_8220	DPD_Type 0010_DES_CBC_HMAC_MD-5 Encrypt Middle
0x2033_8E20	DPD_Type 0010_DES_CBC_HMAC_MD-5 Encrypt Last
0x2023_9822	DPD_Type 0010_DES_CBC_HMAC_SHA-1 Decrypt First
0x2023_8022	DPD_Type 0010_DES_CBC_HMAC_SHA-1 Decrypt Middle
0x2023_8C22	DPD_Type 0010_DES_CBC_HMAC_SHA-1 Decrypt Last
0x2033_9820	DPD_Type 0010_DES_CBC_HMAC_SHA-1 Encrypt First
0x2033_8020	DPD_Type 0010_DES_CBC_HMAC_SHA-1 Encrypt Middle
0x2033_8C20	DPD_Type 0010_DES_CBC_HMAC_SHA-1 Encrypt Last
0x2063_9A22	DPD_Type 0010_3DES_CBC_HMAC_MD-5 Decrypt First
0x2063_8222	DPD_Type 0010_3DES_CBC_HMAC_MD-5 Decrypt Middle
0x2063_8E22	DPD_Type 0010_3DES_CBC_HMAC_MD-5 Decrypt Last
0x2073_9A22	DPD_Type 0010_3DES_CBC_HMAC_MD-5 Encrypt First
0x2073_8220	DPD_Type 0010_3DES_CBC_HMAC_MD-5 Encrypt Middle
0x2073_8E20	DPD_Type 0010_3DES_CBC_HMAC_MD-5 Encrypt Last
0x2063_9822	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Decrypt First
0x2063_8022	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Decrypt Middle
0x2063_8C22	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Decrypt Last
0x2073_9820	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Encrypt First
0x2073_8020	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Encrypt Middle
0x2073_8C20	DPD_Type 0010_3DES_CBC_HMAC_SHA-1 Encrypt Last

10 SSLv3.1/TLS1.0 Processing

The MPC184 is capable of assisting in SSL record layer processing, however, for SSL v3.0 and earlier, this support is limited to acceleration of the encryption only. The MDEU does not calculate the version of HMAC required by early versions of SSL. SSLv3.1 and TLSv1.0 use the same HMAC version as IPSec (specified in RFC2104), which the MPC184 MDEU supports, allowing it to off-load both bulk encryption and authentication from the host processor.

SSLv3.1 and TLSv1.0 (henceforth, referred to as TLS) record layer encryption/decryption is more complicated for hardware than IPSec, due to the order of operations mandated in the protocol. TLS performs the HMAC function first, then attaches the HMAC (which is variable size) to the end of the payload data. The payload data, HMAC, and any padding added after the HMAC are then encrypted. Parallel encryption and authentication of TLS records cannot be performed using the MPC184 snooping mechanisms which works for IPSec.

Performing TLS record layer encryption and authentication with the MPC184 requires two descriptors. For outbound records, one descriptor is used to calculate the HMAC, and a second is used to encrypt the record, HMAC, and padding. For inbound records, the first descriptor decrypts the record, while the second descriptor is used to recalculate the HMAC for validation by the host. With some planning, the user may create the outbound descriptors and launch them as a chain, leaving the MPC194 to complete the full HMAC/encrypt operation before signaling DONE. Placing the output from descriptor 1 into the MPC184 on-chip gpRAM, then fetching that data is input for descriptor 2 can provide additional bus bandwidth savings, and improved system performance. It is anticipated that for inbound records, the MPC184 will signal DONE after decryption, so that the host can determine the location of the HMAC before setting up the HMAC validation descriptor.

The following sections provide examples and explanations covering TLS outbound and inbound processing using dynamic assignment.

10.1 Outbound TLS Descriptor 1

The first descriptor performs the HMAC of the record header and the record payload, as shown in Table 28. In the example shown, the HMAC is generated using the MD-5 algorithm.

Table 28. Outbound TLS Descriptor 1

Field	Value/Type	Description
Header	0x31E0_0010	DPD_Type 0001_HMAC_MD-5
LEN_1	Length	Null
PTR_1	Pointer	Null
LEN_2	Length	Null
PTR_2	Pointer	Null
LEN_3	Length	Number of bytes of HMAC key to be written to MDEU key register
PTR_3	Pointer	PCI address of HMAC key
LEN_4	Length	Number of bytes of data to be written to MDEU input FIFO
PTR_4	Pointer	PCI address of data
LEN_5	Length	Null

Table 28. Outbound TLS Descriptor 1 (continued)

Field	Value/Type	Description
PTR_5	Pointer	Null
LEN_6	Length	Number of bytes of HMAC to be written out to memory (always 16 MD-5)
PTR_6	Pointer	PCI address where HMAC is to be written
LEN_7	Length	Null
PTR_7	Pointer	Null
PTR_NEXT	Pointer	Pointer to next descriptor

The primary EU is the MDEU, with its mode bits set to cause the MDEU to initialize its context registers, perform autopadding if the data size is not evenly divisible by 512 bits, and calculate an HMAC-MD-5. The descriptor header doesn't designate a secondary EU, so the setting of the snoop type bit is ignored.

At the conclusion of outbound TLS descriptor 1, the crypto-channel has calculated the HMAC, placed it in memory, and has reset and released the MDEU.

10.2 Outbound TLS Descriptor 2

The second descriptor performs the encryption of the record, HMAC, pad length, and any padding generated to disguise the size of the TLS record, as shown in Table 29.

Table 29. Outbound TLS Descriptor 2

Field	Value/Type	Description
Type 0101 common_nonsnoop_afeu	0x1000_0010	AFEU, new key, don't dump context, perform permute
LEN_1	Length	Null
PTR_1	Pointer	Null
LEN_2	Length	Null
PTR_2	Pointer	Null
LEN_3	Length	Length of ARC-4 key
PTR_3	Pointer	Pointer to ARC-4 Key
LEN_4	Length	Length of data to be read and permuted
PTR_4	Pointer	Pointer to data in memory
LEN_5	Length	Length of data to be written after permutation
PTR_5	Pointer	Pointer to memory buffer for write back
LEN_6	Length	Null
PTR_6	Pointer	Null
LEN_7	Length	Null
PTR_7	Pointer	Null
PTR_NEXT	Pointer	Null or pointer to unrelated next descriptor

Not surprisingly, inbound TLS processing reverses the order of operations of outbound processing.

10.3 Inbound TLS Descriptor 1

The first descriptor performs the decryption of the record, HMAC, pad length, and any padding generated to disguise the size of the TLS record, as shown in Table 30.

Table 30. Inbound TLS Descriptor 1

Field	Value/Type	Description
Type 0101 common_nonsnoop_afeu	0x1000_0010	AFEU, new key, don’t dump context, perform permute
LEN_1	Length	Place holder
PTR_1	Pointer	Place holder
LEN_2	Length	Place holder
PTR_2	Pointer	Place holder
LEN_3	Length	Length of ARC-4 key
PTR_3	Pointer	Pointer to ARC-4 Key
LEN_4	Length	Length of data to be read and permuted
PTR_4	Pointer	Pointer to data in memory
LEN_5	Length	Length of data to be written after permutation
PTR_5	Pointer	Pointer to memory buffer for writeback
LEN_6	Length	Null
PTR_6	Pointer	Null
LEN_7	Length	Null
PTR_7	Pointer	Null
PTR_NEXT	Pointer	Null or pointer to unrelated next descriptor

Note that ARC-4 does not have a concept of encrypt vs. decrypt. As a stream cipher, ARC-4 generates a key stream which is XOR’d with the input data. If the input data is plaintext, the output is ciphertext. If the input data is ciphertext (which was previously XOR’d with the same key), the result is plaintext.

The primary EU is the AFEU, with its mode bits set to cause the AFEU to load the key and initialize the AFEU S-box for data permutation.

The descriptor header doesn’t designate a secondary EU, so the setting of the snoop type bit is ignored.

At the conclusion of inbound TLS descriptor 1, the AFEU has decrypted the TLS record so that the payload and HMAC are readable. The negotiation of the TLS session should provide the receiver with enough information about the session parameters (hash algorithm for HMAC, whether padding is in use) to create inbound descriptors 2 along with descriptor 1. If so, the next descriptor pointer field should point to descriptor 2.

Alternatively, the MPC184 could signal DONE at the conclusion of inbound descriptor 1 to allow the host to inspect the decrypted record, and generate the descriptor necessary to validate the HMAC. If this is the case, inbound descriptor 2 does not need to be linked to inbound descriptor 1, and could even be processed by a different crypto-channel.

10.4 Inbound TLS Descriptor 2

The second descriptor performs the HMAC of the record header and the record payload. In the example shown in Table 31, the HMAC is generated using the MD-5 algorithm.

Table 31. Inbound TLS Descriptor 2

Field	Value/Type	Description
Type 0001 common_nonsnoop_non_ afeu	0x31E0_0010	MDEU, HMAC, MD-5, autopad
LEN_1	Length	Null
PTR_1	Pointer	Null
LEN_2	Length	Null
PTR_2	Pointer	Null
LEN_3	Length	Length of MD-5 key
PTR_3	Pointer	Pointer to MD-5 key
LEN_4	Length	Length of data to be read and permuted
PTR_4	Pointer	Pointer to data in memory
LEN_5	Length	Null
PTR_5	Pointer	Null
LEN_6	Length	Length of HMAC to be written to memory (16 bytes for MD-5)
PTR_6	Pointer	Pointer to memory location for HMAC write (must be modulo-8)
LEN_7	Length	Null
PTR_7	Pointer	Null
PTR_NEXT	Pointer	Null or pointer to unrelated next descriptor

The primary EU is the MDEU, with its mode bits set to cause the MDEU to initialize its context registers, perform autopadding if the data size is not evenly divisible by 512 bits, and calculate an HMAC-MD-5.

The descriptor header does not designate a secondary EU, so the setting of the snoop type bit is ignored.

At the conclusion of inbound TLS Descriptor 2, the crypto-channel has calculated the HMAC, placed it in memory, and has reset and released the MDEU. The host can compare the HMAC generated by inbound TLS descriptor 2 with the HMAC that came as part of the record. If the HMACs match, the record is known to have arrived unmodified, and can be passed to the application layer.

The next descriptor pointer field can also be null, or point to an unrelated dynamic descriptor.

11 Conclusion

The MPC184 device driver will generate most of the descriptors described in this application note; however, the drivers are general purpose in structure, and may provide more options than certain applications require. By providing the user with greater detail and specific examples of descriptor programming, the user may choose to implement an application-specific minimal driver with higher performance and a smaller memory footprint.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-480-768-2130
(800) 521-6274

JAPAN:

Motorola Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu Minato-ku
Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre, 2 Dai King Street
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

(800) 521-6274

HOME PAGE:

www.motorola.com/semiconductors

MOTOROLA

AN2492/D