Freescale Semiconductor
Application Note

Document Number: AN2912
Rev. 1, 04/2005

SEC Lite Descriptor Programmer's Guide

by *Geoffrey Waters*
Security Applications
Freescale Semiconductor, Inc.
Austin, TX

This application note is offered as a supplement to the *MPC885 PowerQUICC™ Family Reference Manual* and the *SEC 1.0 Reference Device Driver User's Guide*, to assist the user in understanding and creating descriptors in the event the user has more specific requirements than those covered by the *SEC 1.0 Reference Device Driver User's Guide*. This application note assumes the reader is already basically familiar with the SEC Lite architecture as explained in the *MPC885 PowerQUICC Family Reference Manual*. The *SEC 1.0 Reference Device Driver User's Guide* covers both the SEC and SEC Lite and includes a superset of the functionality supported by the SEC Lite. This application note will also be helpful in explaining which SEC 1.0 driver functions are not supported in the SEC Lite.

Contents

1 SEC Lite Data Packet Descriptor Overview

The SEC Lite has DMA capability to off-load data movement and encryption operations from the MPC885 CPU core. As the system controller, the CPU of the MPC885 maintains a record of current secure sessions and the corresponding keys and contexts of those sessions. Once the MPC885 CPU core determines a security operation is

freescale™ semiconductor

required, it can either directly write keys, context, and data to the SEC Lite (SEC Lite in target mode), or the CPU can create a data packet descriptor to guide the SEC Lite through the security operation, with the SEC Lite acting as an internal bus master. The descriptor can be created in main memory, or written directly to the data packet descriptor buffer in the SEC Lite crypto-channel.

2 Descriptor Structure

The SEC Lite data packet descriptors are conceptually similar to descriptors used by most devices with DMA capability. The descriptors are fixed length (64 bytes) and consist of 16 32-bit fields. See Figure 1 for a data packet descriptor. Descriptors begin with a header, which describes the security operations to be performed and the mode to which the execution unit(s) will be set while performing the operation.

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

	0 — 31	32 — 48	48 — 63
Header dword	Header	-	Length 1
Pointer Dword 1	Pointer 1	-	Length 2
Pointer Dword 2	Pointer 2	-	Length 3
Pointer Dword 3	Pointer 3	-	Length 4
Pointer Dword 4	Pointer 4	-	Length 5
Pointer Dword 5	Pointer 5	-	Length 6
Pointer Dword 6	Pointer 6	-	Length 7
Pointer Dword 7	Pointer 7	Next-Descriptor Pointer	

Figure 1. Descriptor Format

3 Descriptor Header

Descriptors are created by the host to guide the SEC Lite 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 SEC 1.0 device drivers allow the host to create proper headers for each cryptographic operation.

Figure 2 shows the descriptor header.

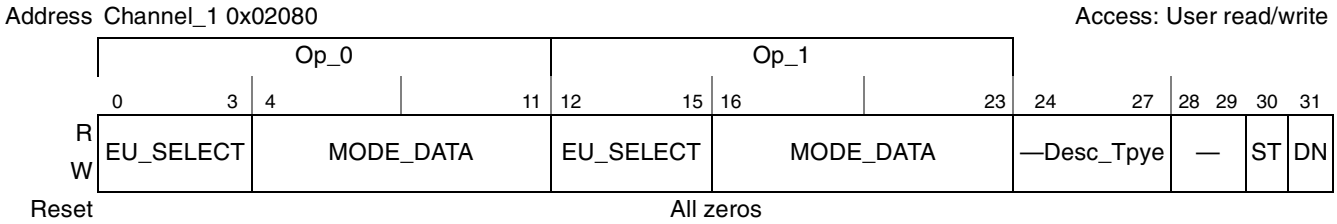

Figure 2. Descriptor Header

Table 1 defines the descriptor header fields.

Table 1. Descriptor Header Field Descriptions

Bits	Name	Description
0–11	Op_0	Op_0 contains two fields, EU_SELECT and MODE_DATA. Figure 3 shows the field detail. EU_SELECT[0–3]—Programs the channel to select a primary EU of a given type. Table 2 lists the possible EU_SELECT values. MODE_DATA[4–11]—Programs the primary EU mode data. The mode data is specific to the chosen EU. This data is passed directly to bits 0–7 of the specified EU mode register.
12–23	Op_1	Op_1 contains two fields, EU_SELECT and MODE_DATA. Figure 3 shows the field detail. EU_SELECT[12–15]—Programs the channel to select a secondary EU of a given type. Table 2 lists the possible EU_SELECT values. MODE_DATA[16–23]—Programs the secondary EU mode data. The mode data is specific to the chosen EU. This data is passed directly to bits 0–7 of the specified EU mode register. The MDEU is the only valid secondary EU. Values for Op_1 EU_SELECT other than 0011 (MDEU) or 0000 (No EU selected) will result in an unrecognized header error condition. Selecting 0011 (MDEU) for both primary and secondary EU will also create an error condition.
24–27	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 7 lists the valid types of descriptors.
28–29	—	Reserved—Set to zero.

Table 1. Descriptor Header Field Descriptions (continued)

Bits	Name	Description
30	ST	Snoop type—Selects which of the two types of available snoop modes applies to the descriptor. See Figure 7 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.
31	DN	Done notification flag Setting this bit indicates whether to perform notification upon completion of this descriptor. The notification can take the form of an interrupt, modified header writeback, or both, depending upon the state of the INTERRUPT_ENABLE and WRITEBACK_ENABLE control bits in the crypto-channel configuration register. 0 Do not signal DONE upon completion of this descriptor (unless globally programmed to do so via the crypto-channel configuration register.) 1 Signal DONE upon completion of this descriptor **Note:** The SEC can be programmed to perform DONE notification upon completion of each descriptor, upon completion of any descriptor, or completion of the final descriptor in a chain. This bit provides for the second case. When the crypto-channel requests a write of the descriptor header back to system memory, the most significant byte (big endian) of the header is always read as set to 0xFF, and the remaining 24 bits are not changed.

Figure 3 shows the two fields of Op_*n*.

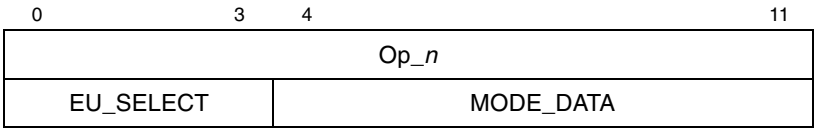

Figure 3. Op_*n* fields

A valid Op_0 EU_SELECT value must be selected or an ‘unrecognized header error’ will be signaled by the channel. Valid Op_0 EU_Select choices are 0010 (DEU), 0110 (AESU), and 0011 (MDEU). The MDEU is the only valid selection for Op_1, and when Op_1 is in use, the DEU or AESU must be selected as Op_0. 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	Reserved
0010	DEU (DES execution unit)
0011	MDEU (message digest execution unit)
0100	Reserved
0101	Reserved

Table 2. EU_Select Values (continued)

Value	EU Select
0110	AESU (AES execution unit)
Others	Reserved EU

4 Execution Unit MODE_DATA

The SEC Lite execution units are programmed via the descriptor header. The MODE_DATA portion of the Op_*n* field in the descriptor header is written to bits 0–7 of the mode register in the execution unit selected by the EU_SELECT field in Op_*n*. A complete explanation of the execution unit registers can be found in the *MPC885 PowerQUICC Family Reference Manual*. However, the mode register for each EU is provided in this section for convenience.

4.1 DEU Mode Register

Shown in Figure 4, the mode register is used to control operation of the DEU.

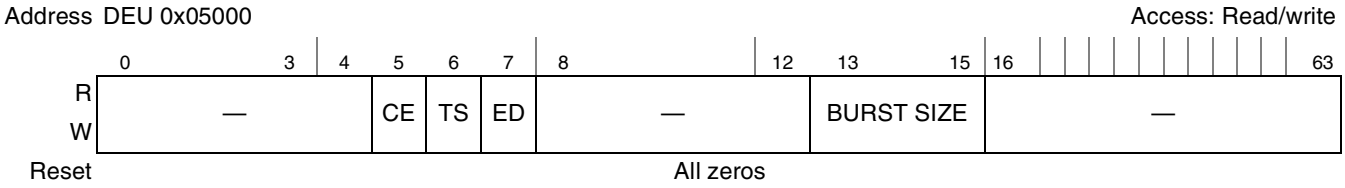

Figure 4. DEU Mode Register

Table 3 describes DEU mode register fields.

Table 3. DEU Mode Register Bit Definitions

Bits	Name	Description
0–4	—	Reserved
5	CE	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
6	TS	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
7	ED	Encrypt/decrypt. If set, DEU operates the encryption algorithm; if not set, DEU operates the decryption algorithm. 0 Perform decryption 1 Perform encryption
8–12	—	Reserved

Table 3. DEU Mode Register Bit Definitions (continued)

Bits	Name	Description
13–15	BURST SIZE	The SEC Lite 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.
16–63	—	Reserved

4.2 MDEU Mode Register

The MDEU mode register, displayed in Figure 5, is used to control operation of the MDEU.

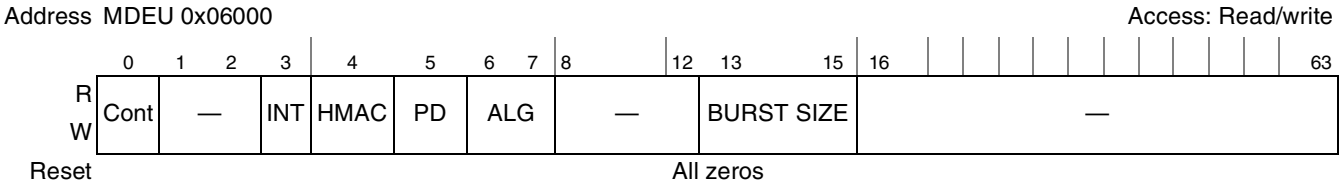

Figure 5. MDEU Mode Register

Table 4 describes MDEU mode register signals.

Table 4. MDEU Mode Register Bit Definitions

Bits	Signal	Description
0–2	—	Reserved—must be set to zero.
3	INT	Initialization bit. Causes an algorithm-specific initialization of the digest registers. In the SEC Lite, must always be set to 1. 0 Do not initialize. 1 Initialize the selected algorithm's starting registers.
4	HMAC	Identifies the hash operation to execute 0 Perform standard hash 1 Perform HMAC operation. This requires a key and key-length information.
5	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.
6–7	ALG	Message digest algorithm selection 00 SHA-160 algorithm (full name for SHA-1) 01 SHA-256 algorithm 10 MD5 algorithm 11 Reserved
8–12	—	Reserved

Table 4. MDEU Mode Register Bit Definitions (continued)

Bits	Signal	Description
13–15	BURST SIZE	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.
16–63	—	Reserved

4.3 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 descriptor (the MDEU acting as sole or secondary EU), the mode register bit settings in Table 5 should be used:

Table 5. Mode Register—HMAC Generated by Single Descriptor

Bits	Field	Value
3	Init	1(on)
4	HMAC	1 (on)
5	PD	1 (on)

4.4 AESU Mode Register

The AESU mode register, shown in Figure 6, is 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 SEC Lite 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 AESU 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.

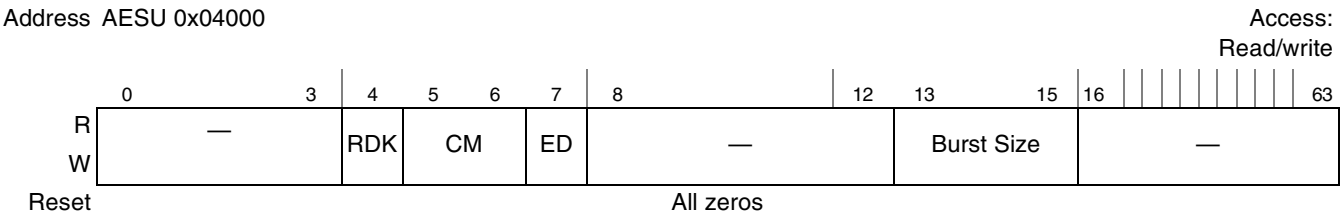

Figure 6. AESU Mode Register

Table 6 describes AESU mode register signals.

Table 6. AESU Mode Register Signals

Bits	Signal	Description
0–3	—	Reserved
4	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.
5–6	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
7	ED	Encrypt/Decrypt. If set, AESU operates the encryption algorithm; if not set, the AESU operates the decryption algorithm. Note: This bit is ignored if CM is set to 11 (CTR Mode). 0 Perform decryption. 1 Perform encryption.
8–12	—	Reserved
13–15	BURST SIZE	The SEC Lite 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 the FIFO. 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.
16–63	—	Reserved

NOTE: Restore Decrypt Key (RDK)

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 ~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 AESU mode must be decrypt. See the “SEC Lite Descriptors” chapter in the *MPC885 PowerQUICC Family Reference Manual* for more information. The descriptor will cause the SEC Lite to write the contents of the context registers and the 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 need for simultaneous HMAC generation and so forth), and set the restore decrypt key mode bit. Load the context registers and the expanded decrypt key with the previously saved key and context data from the first message. The key size is written as before (16, 24, or 32 bytes).

5 Descriptor Fields

The SEC Lite accepts seven fixed-format descriptors. The Desc_Type field in the descriptor header 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 Desc_Type) is shown in Table 8.

Table 7 shows the permissible values for the Desc_Type field in the descriptor header.

NOTE

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 7. Descriptor Types

Value	Descriptor Type	Notes
0000	aesu_ctr_nonsnoop	AESU CTR non-snooping
0001	common_nonsnoop_no_afeu	Common, non-snooping, 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	Reserved	Reserved in SEC Lite
0110	Reserved	Reserved in SEC Lite
0111	Reserved	Reserved in SEC Lite
1000	Reserved	Reserved in SEC Lite
1001	Reserved	Reserved in SEC Lite
1010	Reserved	Reserved in SEC Lite
1011	Reserved	Reserved in SEC Lite
1100	hmac_snoop_aesu_ctr	AESU CTR hmac snooping
1101	non_hmac_snoop_aesu_ctr	AESU CTR non-hmac snooping
1110	Reserved	Reserved in SEC Lite
1111	Reserved	Reserved in SEC Lite

Table 8 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: Some outputs are optional.

Table 8. Descriptor Length/Pointer Mapping

Descriptor Type	Pointer Dword1	Pointer Dword2	Pointer Dword 3	Pointer Dword4	Pointer Dword 5	Pointer Dword 6	Pointer Dword 7
0000 aesu_ctr_ nosnoop	null	Cipher IV	Cipher Key	In FIFO	Out FIFO	Cipher IV Out	null
0001 common_ nosnoop	null	Cipher IV	Cipher Key	In FIFO	Out FIFO	Cipher IV Out	null
0010 hmac_snoop _no_afeu	HMAC Key	HMAC Data	Cipher Key	Cipher IV	In FIFO	Out FIFO	HMAC Out
0011 Reserved	Reserved						
0100 Reserved	Reserved						
1100 hmac_snoop_ aesu_ctr	HMAC Key	HMAC Data	AES Key	AES Ctx	In FIFO	Out FIFO	HMAC Out
1101 Reserved	Reserved						
others	Reserved						

5.1 Descriptor Type 0001

Descriptor type 0001 is used for a wide variety of functions, most of which don't require all the length/pointer fields to be used. Two non-obvious uses of this descriptor type are highlighted in Table 9.

Table 9. Descriptor Type 0001 Length/Pointer Mapping

Descriptor Type	Pointer Dword1	Pointer Dword2	Pointer Dword 3	Pointer Dword4	Pointer Dword 5	Pointer Dword 6	Pointer Dword 7	Use
0001 common_ nosnoop	null	null	null	In FIFO	null	Hash out	null	Hash Only
0001 common_ nosnoop	null	null	HMAC Key	In FIFO	null	HMAC out	null	HMAC Only

For HMAC-only operations, the HMAC key should be loaded, followed by the data. The HMAC itself is written out via pointer DWORD 6.

5.2 Snoop Type Bit

As described in Table 1, bit 30 of the descriptor 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 as IPSec. Upon transmission of 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.

Upon 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.

Figure 7 provides an overview of the snooping concept.

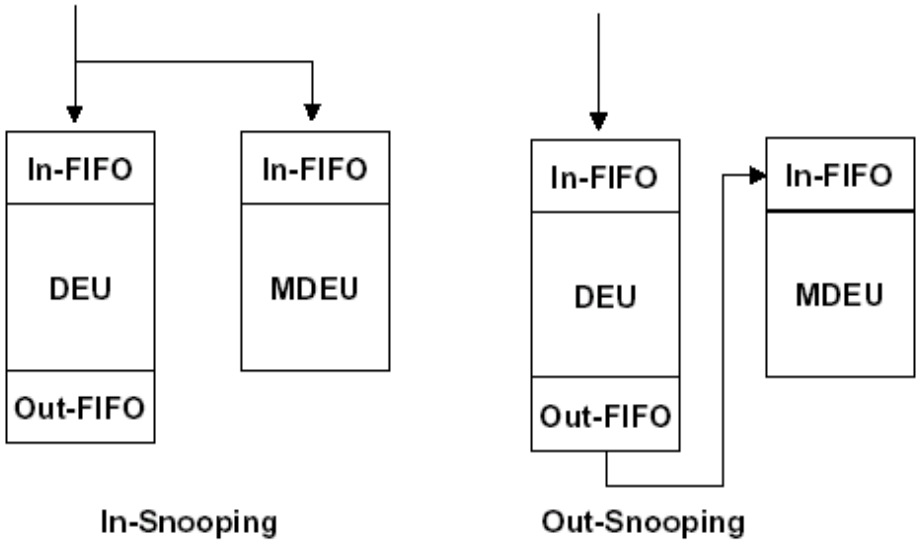

Figure 7. Snooping

5.3 Notification Bit

The notification bit (DN) in the SEC Lite descriptor header acts as a manual override to the crypto-channel configuration register's NOTIFICATION_TYPE bit. The NOTIFICATION_TYPE bit determines whether the SEC Lite 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 DN in the descriptor header is unnecessary and redundant if NOTIFICATION_TYPE is set to end-of-descriptor, but if NOTIFICATION_TYPE is set to end-of-chain, setting DN 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 upon the state of the INTERRUPT_ENABLE and WRITEBACK_ENABLE control bits in the crypto-channel configuration register.

When the channel signals DONE by using header writeback, the most significant byte of the original header (at its original location in system memory) always reads as set to 0xFF and the remaining 24 bits are not modified.

5.4 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. Data field length gives the length of the block in bytes. The maximum allowable number of bytes is 32 Kbytes. A value of zero loaded into data field length indicates that this length/pointer pair should be skipped and processing should continue with the next pair.

Figure 8 shows the descriptor length field.

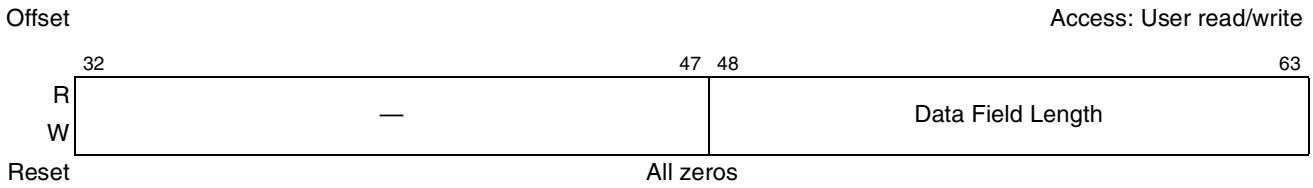

Figure 8. Descriptor Length Field *n*

Table 10 shows the descriptor length field *n* field descriptions.

Table 10. Descriptor Length Field *n* Field Descriptions

Bits	Name	Reset Value	Description
32–47	—	0	Reserved, set to zero
48–63	Data Field Length	0	The maximum length to which this field can be set is 32 Kbytes. 0x0000 = 0 bytes 0x0001 = 1 byte • • • 0x8000_0xFFFF = 32 Kbytes

Figure 9 shows the descriptor pointer field.

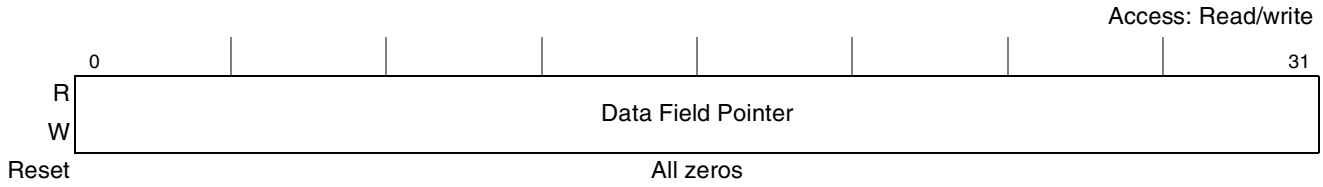

Figure 9. Descriptor Pointer Field

Table 11 shows the descriptor pointer field mapping.

Table 11. Descriptor Pointer Field Mapping

Bits	Name	Reset Value	Description
0–31	Data Field Pointer	0	Contains the address, in the MPC885's global memory space, of the first byte of the data packet for either read or writeback. Transfers from the 8xx bus with data field pointer set to zero will be skipped.

5.5 Descriptor Chaining

Following the length/pointer pairs is the next-descriptor pointer field, which contains the pointer to the next descriptor in memory. Upon completion of processing of the current descriptor, this value, if non-zero, is used to request an 8xx burst read of the next data packet descriptor. This automatic load of the next descriptor is referred to as descriptor chaining.

Figure 10 displays the next descriptor pointer field.

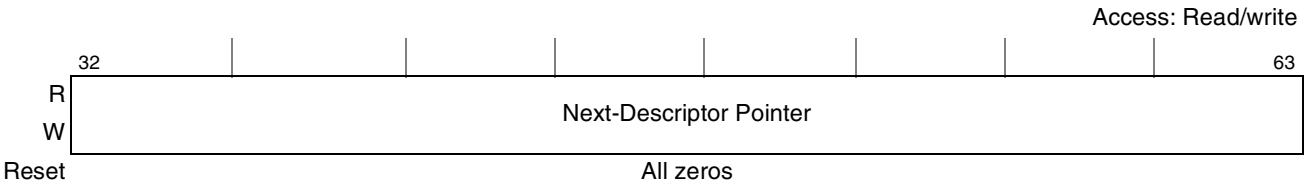

Figure 10. Next-Descriptor Pointer Field

Table 12 describes the descriptor pointer field mapping.

Table 12. Next-Descriptor Pointer Field

Bits	Name	Reset Value	Description
0–31	Next-Descriptor Pointer	0	Contains the address, in 8xx address space, of the next descriptor to be fetched if descriptor chaining is enabled

Descriptor chaining provides a measure of decoupling between host CPU activities and the status of the SEC Lite. Rather than waiting for the SEC Lite to signal DONE and arbitrating for the 8xx 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 SEC Lite by filling the next-descriptor pointer 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 configuration register.

See the "Crypto-Channel Configuration Register (CCCR)" section in the *MPC885 PowerQUICC Family Reference Manual* for additional information on how the SEC Lite can be programmed to signal and act upon completion of a descriptor.

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

Figure 11 shows a conceptual chain, or linked list, of descriptors.

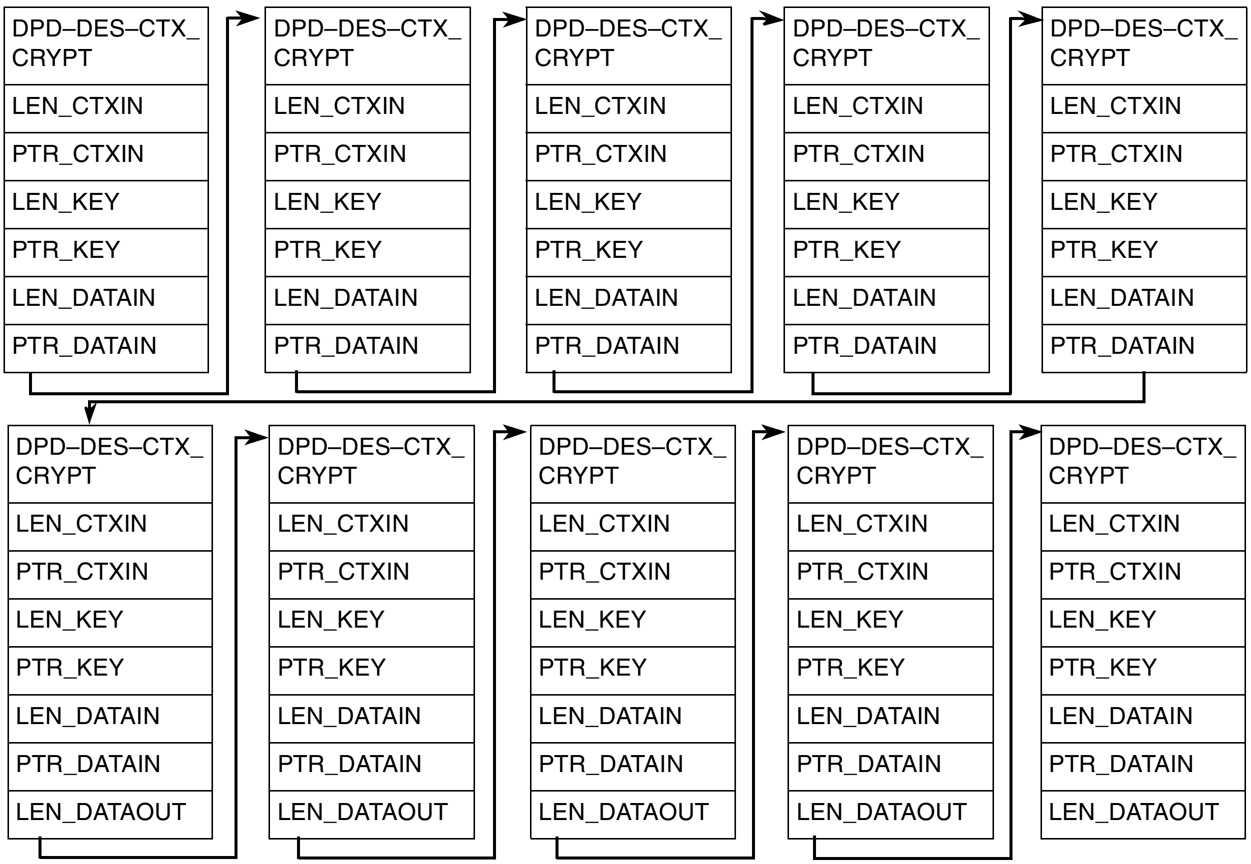

Figure 11. Chain of Descriptors

5.6 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 do not require context; others may only need to fetch a small, contiguous block of data). Therefore, when processing data packet descriptors, the SEC Lite will skip entirely any pointer that has an associated length of zero.

5.7 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 the key and context for the next descriptor is different from the key and context for the current descriptor.

The descriptor shown in Table 13 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.) Upon completion of the descriptor, the DEU is automatically cleared and released.

An actual 3DES-CBC encrypt descriptor is provided in the Appendix.

Table 13. Representative Descriptor DPD_Type 0001_3DES-CBC Encrypt

Field	Value / Type	Description
Descriptor Header	0x2070_0010	DPD_Type 0001_3DES-CBC encrypt
Length 1	Length	null
Pointer 1	Pointer	null
Length 2	Length	Number of bytes of IV to be written to DEU IV register (always 8)
Pointer 2	Pointer	8xx address of IV
Length 3	Length	Number of bytes of key to be written to DEU key register (must be 16 or 24)
Pointer 3	Pointer	8xx address of key
Length 4	Length	Number of bytes to be ciphered
Pointer 4	Pointer	8xx address of data to be ciphered
Length 5	Length	Bytes to be written (should be equal to length of data in)
Pointer 5	Pointer	8xx address where ciphered data is to be written
Length 6	Length	(Optional) Number of bytes of IV to be written to 8xx memory space (always 8)
Pointer 6	Pointer	(Optional) 8xx address where IV is to be written
Length 7	null	null
Pointer 7	null	null
Next-Descriptor Pointer	Pointer	Pointer to next descriptor

6 Additional Examples

Provided below are descriptor examples of some common cryptographic transforms.

6.1 Dynamically Assigned DES-HMAC-MD-5 Encrypt (Outbound IPSec ESP)

Table 14. Representative Descriptor DPD_Type 0010_DES-ECB-HMAC-MD-5 Encrypt

Field	Value / Type	Description
Descriptor Header	0x2003_1E22	DES-ECB-HMAC-MD-5 encrypt
Length 1	Length	Number of bytes of HMAC key to be written to MDEU key register
Pointer 1	Pointer	8xx address of HMAC Key

Table 14. Representative Descriptor DPD_Type 0010_DES-ECB-HMAC-MD-5 Encrypt (continued)

Field	Value / Type	Description
Length 2	Length	Number of bytes to be HMACed but not ciphered
Pointer 2	Pointer	8xx address of data to be HMACed
Length 3	Length	Number of bytes of key to be written to DEU key register (must be 8 for single DES
Pointer 3	Pointer	8xx address of key
Length 4	Length	Number of bytes of IV to be written to DEU IV register (0 for ECB mode)
Pointer 4	Pointer	8xx address of IV
Length 5	Length	Number of bytes of plaintext to be encrypted **Note:** For this descriptor type, the MDEU will also process this data, so that the total data processed through the HMAC function will be Length 2 + Length 5
Pointer 5	Pointer	Address of plaintext to be encrypted Note: This address must be the first address after Pointer 2 + Length 2, so that data being ciphered is contiguous to the data being HMACed only.
Length 6	Length	Number of bytes of plaintext to be written out to memory (should be equal to length of data in)
Pointer 6	Pointer	8xx address where plaintext is to be written
Length 7	Length	Number of bytes of HMAC to be written to 8xx memory space (always 16 for HMAC-MD-5)
Pointer 7	Pointer	8xx address where HMAC is to be written
Next-Descriptor Pointer	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 configures the DEU to operate in DES–ECB–encrypt mode. The Op_1 MODE_DATA configures the MDEU to operate in HMAC-MD-5 mode.

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 DES key is loaded next, followed by the DES IV, which for ECB is null. The data to be encrypted and HMACed is only brought into the SEC a single time, with the DEU and MDEU selectively reading the portions of the data stream corresponding to their data of interest.

Plaintext is brought into the DEU input FIFO, with the MDEU snooping the DUEs out FIFO for the encrypted 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 HMACed has been processed through the MDEU, the descriptor will cause the MDEU to write the HMAC to the indicated area in 8xx memory. The SEC Lite will write the entire 16-byte HMAC-MD-5 to 8xx memory, and the host will append the most significant 12 bytes of the HMAC generated by the SEC Lite to the packet prior to transmission.

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 14.

An actual DES-ECB-HMAC-MD-5 encrypt descriptor is provided in the Appendix.

6.2 Dynamically Assigned AES-HMAC-SHA-256 Decrypt (In-bound IPSec ESP)

Table 15 shows a representative descriptor DPD_Type 0010_AES-CBC-HMAC-SHA-256 decrypt.

Table 15. Representative Descriptor DPD_Type 0010_AES-CBC-HMAC-SHA-256 decrypt

Field	Value / Type	Description
Descriptor Header	0x6023_1D 22	DPD_Type 0010_AES-CBC-HMAC-SHA-256 decrypt
Length 1	Length	Number of bytes of HMAC key to be written to MDEU key register
Pointer 1	Pointer	8xx address of HMAC key
Length 2	Length	Number of bytes to be HMACed but not ciphered
Pointer 2	Pointer	8xx address of data to be HMACed
Length 3	Length	Number of bytes of key to be written to AESU key register (must be 16, 24, or 32)
Pointer 3	Pointer	8xx address of key
Length 4	Length	Number of bytes of IV to be written to AESU IV register (always 16)
Pointer 4	Pointer	8xx address of IV
Length 5	Length	Number of bytes of ciphertext to be decrypted Note: For this descriptor type, the MDEU will also process this data, so that the total data processed through the HMAC function will be Length 2 + Length 5
Pointer 5	Pointer	8xx address of ciphertext to be decrypted Note: This address must be the first address after Pointer 2 + Length 2, so that data being ciphered is contiguous to the data being HMACed only.
Length 6	Length	Number of bytes of plaintext to be written out to memory (should be equal to Length 5)
Pointer 6	Pointer	8xx address where plaintext is to be written
Length 7	Length	Number of bytes of HMAC to be written to 8xx memory space (always 32 for SHA-256)
Pointer 7	Pointer	8xx address where HMAC is to be written
Next-Descriptor Pointer	Pointer	Pointer to next descriptor

The descriptor header encodes the information required to select the AESU for Op_0, and the MDEU for Op_1. The Op_0 mode data configured the AESU to operate in AES, CBC, decrypt mode, without restore decrypt key. The Op_1 mode data configured the MDEU to operate in HMAC-SHA-256 mode.

The descriptor header also encodes the descriptor type 0010_0, 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 AES key is loaded next, followed by the AES

IV. The data to be decrypted and HMACed is only brought into the SEC a single time, with the AESU and MDEU selectively reading the portions of the data stream corresponding to their data of interest.

Ciphertext is brought into the AESU 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 AESU output FIFO, and this data is written back to system memory as needed. When the final byte of data to be HMACed has been processed through the MDEU, the descriptor will cause the MDEU to write the HMAC to the indicated area in memory. The SEC will write the entire HMAC (32 bytes for SHA-256) to memory, and the CPU will compare the most significant 12 bytes of the HMAC generated by the SEC with the HMAC which was received with the in-bound packet. If the HMACs match, the packet integrity check passes.

An actual AES-CBC-HMAC-SHA-256 decrypt descriptor is provided in the Appendix.

6.3 SSLv3.1/TLS1.0 Processing

The SEC Lite is capable of accelerating SSL 3.1 and TLS 1.0 record layer processing, performing both the bulk encryption and the HMAC operation specified in RFC2104. The SEC can also accelerate the encryption used in pre-standard SSL (2.0 and 3.0). However, the MDEU does not calculate the version of HMAC used in pre-standard versions of SSL.

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 of 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 SEC snooping mechanisms which work for IPSec.

Performing TLS record layer encryption and authentication with the SEC requires two descriptors. For out-bound 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 CPU.

The following examples and explanations cover TLS outbound and inbound processing using dynamic assignment.

6.3.1 Outbound TLS Descriptor 1

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

Table 16. Outbound TLS Descriptor 1

Field	Value / Type	Description
Header	0x31E0_0010	DPD_Type 0001_0 HMAC_MD-5
Length 1	Length	null
Pointer 1	Pointer	null
Length 2	Length	null
Pointer 2	Pointer	null

Table 16. Outbound TLS Descriptor 1 (continued)

Field	Value / Type	Description
Length 3	Length	Number of bytes of HMAC key to be written to MDEU Key register
Pointer 3	Pointer	Address of HMAC key
Length 4	Length	Number of bytes of data to be written to MDEU Input FIFO
Pointer 4	Pointer	Address of data
Length 5	Length	null
Pointer 5	Pointer	null
Length 6	Length	Number of bytes of HMAC to be written out to memory (always 16 bytes for MD-5)
Pointer 6	Pointer	Address where HMAC is to be written
Length 7	Length	null
Pointer 7	Pointer	null
Next-Descriptor Pointer	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 auto-padding if the data size is not evenly divisible by 512 bits, and calculate an HMAC-MD-5.

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.

6.3.2 Out-bound 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.

Table 17. Outbound TLS Descriptor 2

Field	Value/ Type	Description
Header	0x1000_0050	AFEU, new key, don't dump context, perform permute
Length 1	Length	null
Pointer 1	Pointer	null
Length 2	Length	null
Pointer 2	Pointer	null
Length 3	Length	Length of ARC-4 key
Pointer 3	Pointer	Pointer to ARC-4 Key
Length 4	Length	Length of data to be read and permuted
Pointer 4	Pointer	Pointer to data in memory

Table 17. Outbound TLS Descriptor 2 (continued)

Field	Value/ Type	Description
Length 5	Length	Length of data to be written after permutation
Pointer 5	Pointer	Pointer to memory buffer for writeback
Length 6	Length	null
Pointer 6	Pointer	null
Length 7	Length	null
Pointer 7	Pointer	null
Next-Descriptor Pointer	Pointer	Pointer to next descriptor

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

6.3.3 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.

Table 18. Inbound TLS Descriptor 1

Field	Value/ Type	Description
Header	0x1000_0050	AFEU, new key, don't dump context, perform permute
Length 1	Length	null
Pointer 1	Pointer	null
Length 2	Length	null
Pointer 2	Pointer	null
Length 3	Length	Length of ARC-4 key
Pointer 3	Pointer	Pointer to ARC-4 Key
Length 4	Length	Length of data to be read and permuted
Pointer 4	Pointer	Pointer to data in memory
Length 5	Length	Length of data to be written after permutation
Pointer 5	Pointer	Pointer to memory buffer for writeback
Length 6	Length	null
Pointer 6	Pointer	null
Length 7	Length	null
Pointer 7	Pointer	null
Next-Descriptor Pointer	Pointer	Pointer to next descriptor

NOTE

ARC-4 does not have a concept of encrypt vs. decrypt. As a stream cipher, ARC-4 generates a key stream which is XORed with the input data. If the input data is plaintext, the output is ciphertext. If the input data is ciphertext (which was previously XORed 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.

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 descriptor 2 and link it to inbound descriptor 1.

Alternatively, the SEC could signal DONE at the conclusion of inbound descriptor 1 to allow the CPU 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, although doing so could save one DONE interrupt from the SEC Lite.

6.3.4 Inbound TLS Descriptor 2

The second descriptor performs the HMAC of the record header and the record payload. In the example shown in Table 19, the HMAC is generated using the MD-5 algorithm.

Table 19. Inbound TLS Descriptor 2

Field	Value/ Type	Description
Header	0x31E0_0010	MDEU, HMAC, MD-5, autopad
Length 1	Length	null
Pointer 1	Pointer	null
Length 2	Length	null
Pointer 2	Pointer	null
Length 3	Length	Length of MD-5 key
Pointer 3	Pointer	Pointer to MD-5 Key
Length 4	Length	Length of data to be read and permuted
Pointer 4	Pointer	Pointer to data in memory
Length 5	Length	null
Pointer 5	Pointer	null
Length 6	Length	Length of HMAC to be written to memory (16 bytes for MD-5)
Pointer 6	Pointer	Pointer to memory location for HMAC write
Length 7	Length	null

Table 19. Inbound TLS Descriptor 2 (continued)

Field	Value/ Type	Description
Pointer 7	Pointer	null
Next-Descriptor Pointer	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 auto-padding 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 CPU 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.

6.3.5 AES-CCM for 802.11i

The SEC Lite is capable of performing AES-CCM operations, as used in 802.11i MAC layer security processing. AES CCM mode combines counter (AES-CTR) mode privacy with cipher block chaining (AES-CBC) for generation of an MIC (message integrity code).

Performing AES-CCM encryption and integrity checking with the SEC Lite requires two descriptors. For outbound 802.11i frames, one descriptor is used to calculate the MIC, and a second is used to encrypt the frame. For inbound frames, the first descriptor decrypts the record, while the second descriptor calculates the MIC for validation by the CPU.

The examples and explanations of Table 20 and Table 21 cover AES-CCM outbound processing using dynamic assignment. Actual AES-CCM descriptors are shown in the Appendix.

Table 20. Representative Descriptor DPD_Type 0001_1_AES-CBC Encrypt

Field	Value / Type	Description
Header	0x6030_0010	DPD_Type 0001_AES_CBC_Encrypt
Length 1	Length	null
Pointer 1	Pointer	null
Length 2	Length	Number of bytes of AES context to be written to AESU context registers (always 56)
Pointer 2	Pointer	Address of context
Length 3	Length	Number of bytes of key to be written to AESU key register (must be 16)
Pointer 3	Pointer	Address of key
Length 4	Length	Number of bytes to be MICed only
Pointer 4	Pointer	Address of data to be MICed only
Length 5	Length	Number of bytes to be MICed and ciphered

Table 20. Representative Descriptor DPD_Type 0001_1_AES-CBC Encrypt (continued)

Field	Value / Type	Description
Pointer 5	Pointer	Address of data to be MICed and ciphered
Length 6	Length	Number of bytes of ciphered data to be written
Pointer 6	Pointer	Address where ciphered data should be written
Length 7	Length	Number of bytes of AES context to be written to memory
Pointer 7	Pointer	Address for writeback of context
Next-Descriptor Pointer	Pointer	Pointer to next descriptor

The descriptor header encodes the information required to select the AESU for Op_0, and no EU for Op_1. The Op_0 mode data configured the AESU to operate in AES-CBC, encrypt mode.

The descriptor header also encodes the descriptor type 0001, which defines the input and output ordering for “AES-CBC”. The AES IV is loaded first, followed by the length and pointer to the AES key. This is followed by a length and pointer to the data which will ciphered.

Since AES CBC is traditionally used as a symmetric cipher mode, the output will be the ciphertext of the entire input. The MIC value will be the least significant 64 bits (bits 0-63) of the last 128 bit block. This truncated MIC value must then be prepended to the end of the data before counter mode encryption with descriptor 2.

Table 21. Representative Descriptor DPD_Type 0001_AES-CTR Encrypt

Field	Value / Type	Description
Header	0x6070_0010	DPD_Type 0001_AES_CTR_Encrypt
Length 1	Length	null
Pointer 1	Pointer	null
Length 2	Length	Number of bytes of AES context to be written to AESU context registers (always 56)
Pointer 2	Pointer	Address of context
Length 3	Length	Number of bytes of key to be written to AESU key register (must be 16)
Pointer 3	Pointer	Address of key
Length 4	Length	Number of bytes to be MIC’d only
Pointer 4	Pointer	Address of data to be MIC’d only
Length 5	Length	Number of bytes to be MIC’d and ciphered
Pointer 5	Pointer	Address of data to be MIC’d and ciphered
Length 6	Length	Number of bytes of ciphered data to be written
Pointer 6	Pointer	Address where ciphered data should be written
Length 7	Length	Number of bytes of AES context to be written to memory

Table 21. Representative Descriptor DPD_Type 0001_AES-CTR Encrypt (continued)

Field	Value / Type	Description
Pointer 7	Pointer	Address for writeback of context
Next-Descriptor Pointer	Pointer	Pointer to next descriptor

7 Conclusion

The SEC Lite is capable of accelerating a wide range of common cryptographic operations. Users can take advantage of the *SEC 1.0 Reference Device Driver* provided by Freescale; or with the understanding of SEC descriptor construction provided by this application note, users can create their own device driver or optimized cryptographic macro routines.

8 Revision History

Table 22 provides a revision history for this application note.

Table 22. Document Revision History

Revision Number	Date	Substantive Change(s)
0	032005	Initial release.
1	04/2005	Removed all references to static mode (sections modified: 4.2, 4.3, 5.1, 5.4, 6.1, and 6.2).

Appendix A

The following are real examples of the descriptors shown in several of the tables. Not all the tables have a matching example descriptor, a few were redundant.

A.1 3DES_CBC_ENC

```
encrypt_type: DEU_1

begin_descriptor:
    20700010      // SEC 1.0 Header (type Common 0001)
    0                              // Adr0 Size Zero
    0                              // NIL Pointer Adr0
    8                              // Adr1 Size
    @q1                            // Pointer Adr1
    18                             // Adr2 Size
    @q2                            // Pointer Adr2
    800                            // Adr3 Size
```

```
    @p3                             // Pointer Adr3
    800                             // Adr4 Size
    @p4                             // Pointer Adr4
    0                               // Adr5 Size Zero
    0                               // NIL Pointer Adr5
    0                               // Adr6 Size Zero
    0                               // NIL Pointer Adr6
    0                               // NIL Pointer to next descriptor
end_descriptor

begin_memory q1:
    22d40d656dfd6d27// Context 0
end_memory

begin_memory q2:
         3d0757674a49e93e  // key0 parity corrected
         684004a10d737689  // key1 parity corrected
         195734c8d9987cb3  // key2 parity corrected
end_memory

begin_memory p3:
    f3d27d82db534079// Data 0
    38953e2e5c48a71b// Data 1
    fec2443ca2bf03dc// Data 2
    93ba5fe8025983bf// Data 3
    8aa74c8d858f5256// Data 4
    c1fc00782b5dfc00// Data 5
    35e3780fbf29c378// Data 6
    2a5c525773a21bbb// Data 7
    51f6150b4fdf0815// Data 8
    ecc493604cad5aec// Data 9
    0745621ce90b2092// Data 10
    fa284f1eae9d3f41// Data 11
    f342286f07d358e6// Data 12
    eb551f2f76bd8ee8// Data 13
```

```
99362064e9a3fb59// Data 14
1d3219f3254d6862// Data 15
12767e485593a1a5// Data 16
8480614af57a0bb3// Data 17
7b4e666fe5c45e24// Data 18
f9096a57f25ad56d// Data 19
380478ebf9700794// Data 20
fa8fb2ecbf2b8e66// Data 21
c3ca532197ba0c21// Data 22
abb03610a5e70f45// Data 23
29d800057a90ea8d// Data 24
0541e0a8f2be7bc2// Data 25
20629a0123294ed7// Data 26
a7d4089e5aad2667// Data 27
5f7576ecc87430b2// Data 28
e511239a7498aeef// Data 29
c7454911447cc743// Data 30
ea8ae4a890aeb1a7// Data 31
8dd4d44c4a13760e// Data 32
64bd4a54a2a86a8d// Data 33
392a16ca2f47c00c// Data 34
e84063ad6395e291// Data 35
3c5a6b4be0ee93c4// Data 36
b0919b0332429db4// Data 37
9c4f3d1f6fe7ed20// Data 38
bd397a6653933f0a// Data 39
31e023c7221c15ad// Data 40
947cc64981559935// Data 41
9d3c338757207626// Data 42
c0d4a3517a41da6d// Data 43
e51b8ee5effde2fb// Data 44
50b7321364383c39// Data 45
345f5505834c3e0c// Data 46
2b5301566ad34ab0// Data 47
2bc445619fb9889a// Data 48
```

```
ac22be322ad6b95f// Data 49
2f4e291237f03f79// Data 50
444610a32cd0af3f// Data 51
6d7efa9332c0edcc// Data 52
d90861b399881dc0// Data 53
952aeccf1e975b5a// Data 54
27afe321418d82fe// Data 55
6e0474162e1a7f5a// Data 56
9f492a2afc499b3c// Data 57
05c832665bd6ac56// Data 58
491df44ee561125a// Data 59
9afa152154554488// Data 60
203d40a02c610fcb// Data 61
b44839289b09057c// Data 62
b6a457d277bfd846// Data 63
d149d7903b6bcb2f// Data 64
439583da63dcde2f// Data 65
2270865e02d29f31// Data 66
d3f0772d2212e32f// Data 67
80561b99b3dbd982// Data 68
3acb19c39ae2d48f// Data 69
966f660979af1da5// Data 70
ce077725779adce9// Data 71
8719b8d943ae0d7c// Data 72
cc9fe22f0c8f458b// Data 73
8ddc6ea3f612b95d// Data 74
0966e32b70f1c4f2// Data 75
340822a6ec524d3e// Data 76
1b6a2408702a1d6c// Data 77
2e06cb4e34993457// Data 78
7241ac04bf4b514c// Data 79
761c40ace3ff1693// Data 80
0ffc9f024380afb3// Data 81
1994135276b6c92b// Data 82
ad256e224c2f8435// Data 83
```

```
aac784325a065fed// Data 84
c9a09a384b27a9b7// Data 85
73d572c167e519e7// Data 86
98181d5abac005b1// Data 87
92f6d7f5888f96e7// Data 88
1144d43ea6fd9e1a// Data 89
dc5fbd252dda7000// Data 90
d6b534fd02c9d53c// Data 91
b91b8a297ce8b1e3// Data 92
3823b32068384af2// Data 93
6961d15e61c28fa5// Data 94
4412f5e1e63aab9a// Data 95
e031868dae5b4ecb// Data 96
d72de8afa0d6f2f2// Data 97
c9a295f6698f904d// Data 98
77cbbc7750ee7858// Data 99
83a94c56ee3634a2// Data 100
5d8bb9c9534d8260// Data 101
145214eece4f46f5// Data 102
6bd68b85e0d7e34b// Data 103
6571a541b292f082// Data 104
652ee62fcf5a6ccd// Data 105
e716f4a0c6e4c945// Data 106
95effe8b91ad1dee// Data 107
ff191392fcff888d// Data 108
3d5fd5eeaa835f53// Data 109
693e623d0b8afe3b// Data 110
02fc083004949d26// Data 111
056e7e0931545a8e// Data 112
9ffe35a96096bdd0// Data 113
2960031fb106ec5c// Data 114
36a844f34335845f// Data 115
00e228da43425ed7// Data 116
cc455e6ac809534e// Data 117
5dcb133684a368b7// Data 118
```

```
1e19b6acefa27a8d// Data 119
c59f29b2a8cf1033// Data 120
8c23fee72faa867c// Data 121
94171a2cb715aae3// Data 122
3eccd7aae295700e// Data 123
bcd9c3e6404d77f3// Data 124
1fe36e293aef3074// Data 125
f57242c3d7b855ee// Data 126
92c5ccd5ee2f2cc4// Data 127
ba9aff1583e200d2// Data 128
2f556bc468b5b615// Data 129
1ec242a3353fa34e// Data 130
b1b1ff8bfcfbaf3e// Data 131
18661e93601fa333// Data 132
0bd14df47e53d558// Data 133
250846ae7fd76612// Data 134
85a674959d2a723e// Data 135
b916191e5febddd0// Data 136
7d29d33bbac5e00d// Data 137
ba09e46ec774c8d0// Data 138
953fd651ec66c441// Data 139
f73f51db30c683fa// Data 140
a5a644b6f153d33c// Data 141
f30458d4da307b96// Data 142
eb6317b76ad02e4d// Data 143
c6bbdf8889d9468d// Data 144
6b37600780fae396// Data 145
7e902968dae1b9fb// Data 146
eeddb2073004afac// Data 147
8ffcd5792c83f9ab// Data 148
e1078d5a2e1ffc63// Data 149
520da21e74d235ec// Data 150
9983cc26d132e053// Data 151
f6fe6b4c9edb0c4e// Data 152
fe067a1247716144// Data 153
```

```
3ef1596cebb629f1// Data 154
9121cf6665f5f1bf// Data 155
9523c257b0628598// Data 156
c48d5f9ba2a50f60// Data 157
49aa6b56dada5638// Data 158
b2bcddad6e7e6aa4// Data 159
3e860447fa10e165// Data 160
2b4c1a489c592a59// Data 161
10c09d5576f2e736// Data 162
92d2535af6476abf// Data 163
3d7a62505b742715// Data 164
43bdf354eb14af66// Data 165
ca81fd697a68ad0b// Data 166
3d8663369fa38ba3// Data 167
83fae4323a87a464// Data 168
f6a0a962dcb2e038// Data 169
f7a53e0df9c2a31d// Data 170
f86699be522adb0d// Data 171
6c79fd4098e76094// Data 172
634e838f45aaaae7// Data 173
cc4a7f21bf8bcc9a// Data 174
0e97d7abc0fee4b1// Data 175
d9f26428db7de77f// Data 176
5499505d83ba5224// Data 177
e936654ff05720f8// Data 178
b595c85d04abd62c// Data 179
17c69b00e4c91168// Data 180
18d857e76d57b327// Data 181
538a3c6837618355// Data 182
298650b99e604723// Data 183
26039cc53ad37b18// Data 184
59dd57e13f603190// Data 185
da0b7a06d79e2d39// Data 186
cfe51a2089d94a40// Data 187
5219a68ccd3afbd3// Data 188
```

```
57e09777bf4a9997// Data 189
64d95c196d29a391// Data 190
94e918697627645a// Data 191
7dd4c502e3d60082// Data 192
dbe3e61b4be24fde// Data 193
b0ba871c2cc64c14// Data 194
2b0dd7e35fe4d8db// Data 195
03d7578eae58bf68// Data 196
263e9de9a9900e9b// Data 197
f3de270fb0d2dcc0// Data 198
52675886d5420ca9// Data 199
c40c37cb63966630// Data 200
ff83589536503b51// Data 201
aeeae35d5cff6d1c// Data 202
ce842891f2ea3230// Data 203
3dd7d6f014311d4b// Data 204
77398039c768b366// Data 205
f62ee715bb8ad025// Data 206
6a8859ec37fee073// Data 207
10231cdb27214074// Data 208
8832d46a9ac8e86b// Data 209
8a32c8a2f2473073// Data 210
bb0066f5ce0927b1// Data 211
d7816baf8d05d6fd// Data 212
52b69b3185c37779// Data 213
6052b325dc715023// Data 214
bff15274293ca00c// Data 215
55debc1bacb65ca2// Data 216
ac979454d668e550// Data 217
58376ee87dcc78ab// Data 218
e918b9e9514179c1// Data 219
e49b507475c3bac3// Data 220
b92e26cc52edfd29// Data 221
5710f60b8f943b1b// Data 222
465a8d0a073a2cd5// Data 223
```

```
    77b430685e446d1b// Data 224
    404aef3e66300393// Data 225
    0d821fe848aed3f9// Data 226
    4b636a4727c1d9a3// Data 227
    69accd3b58a95c0d// Data 228
    923649b5c8c7f4c7// Data 229
    a0eed69051240352// Data 230
    0539337860170731// Data 231
    9d5de1abca091343// Data 232
    68d1f0e3d77f0375// Data 233
    bfd9e28df0fb14bd// Data 234
    f78893fbbcb6f7ca// Data 235
    5c4a0cddb4c59a2f// Data 236
    e2854de556ae2dc7// Data 237
    6f838fb0bf3ea849// Data 238
    0f358c99bee01073// Data 239
    995e07af7d044f6c// Data 240
    f9f20e5e9522965b// Data 241
    40a241a3f2c1fa6c// Data 242
    d10eabc07f209489// Data 243
    a01c44360537276c// Data 244
    bda27b4968714a6a// Data 245
    8976474635e1bc34// Data 246
    4af5d25870b609ff// Data 247
    726fbc4fc465a0fa// Data 248
    f845524a2d798d96// Data 249
    b7954413bb412748// Data 250
    eeaabcafdb3e8021// Data 251
    5bbacf11b91db247// Data 252
    e935eb8b9a702b15// Data 253
    025d671b304c0f95// Data 254
    aa103c8f21b79115// Data 255
end_memory

begin_memory exp_p4:
```

```
ddb3d58eed57672f// Expected Data 0
bedba50de12a93fe// Expected Data 1
7c89c862a9c42c90// Expected Data 2
1ad291f0121852ac// Expected Data 3
f235614d39efb705// Expected Data 4
4a9d82d2eac38911// Expected Data 5
1f9c58cbdacd8dcc// Expected Data 6
ea97c030d5cc1ef3// Expected Data 7
2f48c3f8561fb4ef// Expected Data 8
7f4754c7b0d321fd// Expected Data 9
463a0c6057752b89// Expected Data 10
f16aa6db6df51d02// Expected Data 11
f225f75a2271198a// Expected Data 12
71e375db7d5c09d8// Expected Data 13
704d272b386422e3// Expected Data 14
520ac3e481021d63// Expected Data 15
9c876c05e6a60f54// Expected Data 16
861d691665975a0c// Expected Data 17
14f68be967077a62// Expected Data 18
276427cce070ffe5// Expected Data 19
3b9afbff2514c049// Expected Data 20
6c83c80514629a4c// Expected Data 21
cd5e5770481569lf// Expected Data 22
e357bde509f29672// Expected Data 23
6796ab6c2992a691// Expected Data 24
98f055b677eb0d11// Expected Data 25
3092ced39b67e8ba// Expected Data 26
d8f903ba5f4aa767// Expected Data 27
8402990b12f690ec// Expected Data 28
7121128f46a064b0// Expected Data 29
6e2b88771b963dbb// Expected Data 30
fb563561b1472d7d// Expected Data 31
b9eebe8d3c763608// Expected Data 32
91ec49190ac6aacb// Expected Data 33
e3bc57eb078fe4b4// Expected Data 34
```

```
38f52d93b5596e9d// Expected Data 35
5e942f7fddef6e7c// Expected Data 36
c6af96213d94ff50// Expected Data 37
bfdc96e5f064f0f0// Expected Data 38
bfd1f662ae62fd44// Expected Data 39
282fc941854e17ea// Expected Data 40
b4d292994cc5d11c// Expected Data 41
1a671fd0bd7bcbca// Expected Data 42
c7fe7d9efd06b60d// Expected Data 43
8eaf8cd9a58937ac// Expected Data 44
58ce53ef8a440bc7// Expected Data 45
51975f64f22823b0// Expected Data 46
b2a01233ddd0b494// Expected Data 47
78293c754f61667b// Expected Data 48
f1869d5e27ad6bd6// Expected Data 49
6ed09d50b5c3417f// Expected Data 50
fe7455791276a433// Expected Data 51
66fa1f5938e3e367// Expected Data 52
c40b91ba7c29470a// Expected Data 53
ec704a88b0c263b0// Expected Data 54
b7724be90a029d2f// Expected Data 55
a474401bfc2814a9// Expected Data 56
96566e9edbde6b3d// Expected Data 57
57d6e78c10360b30// Expected Data 58
7cc951fbb13573f3// Expected Data 59
6c8c2d82e106af4b// Expected Data 60
8ac71e3ecacc2d77// Expected Data 61
9affd9323984deba// Expected Data 62
b1fd10e4741ec22c// Expected Data 63
3489b97a7b6f6d1a// Expected Data 64
9c4399f72fd22cdf// Expected Data 65
61e265e0eb7742d1// Expected Data 66
d664d7d5d51a8da8// Expected Data 67
a3ba10a66cb4de1b// Expected Data 68
12523f6ef7a7de7f// Expected Data 69
```

```
1397ef082f86873c// Expected Data 70
080b52278ba44149// Expected Data 71
7724853f85fbb8f5// Expected Data 72
c78acc6770ad9a37// Expected Data 73
7c65e7ad64dd4b84// Expected Data 74
9cace126cd7325a7// Expected Data 75
70aa181e1163e444// Expected Data 76
513153fa3e8c9261// Expected Data 77
4840a613a4cba6ca// Expected Data 78
a63097a584ce9a6d// Expected Data 79
3147cb90456969c3// Expected Data 80
ea617a6b646f5ffe// Expected Data 81
07391538520dc518// Expected Data 82
46eda68b8daf714d// Expected Data 83
fd9adfb932770d8e// Expected Data 84
68299a8e662eba3c// Expected Data 85
254f074b6e17003a// Expected Data 86
3baad8506da0b3f2// Expected Data 87
f365ed839dacaace// Expected Data 88
ddc37e2bed290aae// Expected Data 89
2f5cc16127182c6b// Expected Data 90
624a5fe06698ad72// Expected Data 91
1a1bad307f86d794// Expected Data 92
d50d192cbfd9df21// Expected Data 93
47546950a878b30e// Expected Data 94
3ef773f830f32188// Expected Data 95
c9355882ab7e1453// Expected Data 96
ec10251d3e0d338d// Expected Data 97
e455a7ed37e47f59// Expected Data 98
e8683331ec7d5f38// Expected Data 99
c07595fbd5a45f4b// Expected Data 100
b6a081826e5e53a0// Expected Data 101
667c4caa8baa9e43// Expected Data 102
d929228ea4fee837// Expected Data 103
352523ffb50997d5// Expected Data 104
```

```
1e1868f69909058c// Expected Data 105
28598a55fe7f72c9// Expected Data 106
30193c8f8b6a31d5// Expected Data 107
03c634877144ae9d// Expected Data 108
e0910e9ddcdc2883// Expected Data 109
172b8c7d56a8bedb// Expected Data 110
9e89458c144b8abd// Expected Data 111
cef632dd3c9667e5// Expected Data 112
5aab1fa6883194c2// Expected Data 113
f1d9e11dcecb776b// Expected Data 114
c58c959ae0ce0a5e// Expected Data 115
404cc05cee67ffd8// Expected Data 116
13978b8bee3125ff// Expected Data 117
1cf93bda49749da9// Expected Data 118
74c00d5ce0735074// Expected Data 119
fcb8d0ec5d23f4c5// Expected Data 120
bd738e077960af56// Expected Data 121
ea74bbd0aea95e3e// Expected Data 122
23fb95a7de1d51d6// Expected Data 123
e6ecebce42bcba71// Expected Data 124
50b4831f50a528d5// Expected Data 125
ebe51f00ab8d6485// Expected Data 126
b7316cd4cb3905fa// Expected Data 127
e613a6af0f0d82bc// Expected Data 128
41091b3fb4fc9247// Expected Data 129
be02151eb0580171// Expected Data 130
24561fa036ebb32c// Expected Data 131
b0371a2e44bcb6b1// Expected Data 132
9feb546ecbc347d3// Expected Data 133
c8527a208f187fa4// Expected Data 134
21810a033957d74a// Expected Data 135
68bc17df79305bf8// Expected Data 136
9ff69545b487c946// Expected Data 137
6dc10c7600000a63// Expected Data 138
c2bdd4a09a676886// Expected Data 139
```

```
ba44735cdabe5da6// Expected Data 140
0645fce135dc58d5// Expected Data 141
942790831963480f// Expected Data 142
8cfc9c09dacc51f4// Expected Data 143
45638ad197da2b24// Expected Data 144
d22095da28ca42ae// Expected Data 145
aac9db4bd6b86550// Expected Data 146
3686fd4eb5140d86// Expected Data 147
177053d4b49325aa// Expected Data 148
5b325b52225551f7// Expected Data 149
873a272e7ae00c63// Expected Data 150
87835084e8e0cf50// Expected Data 151
4d48390b23de1153// Expected Data 152
c1199ab1f1373b99// Expected Data 153
5822b8784a00c305// Expected Data 154
64f0082f5b6c2bb8// Expected Data 155
ae0a1a64d9247707// Expected Data 156
d58fca3adea8f386// Expected Data 157
d3513bcb912af9f6// Expected Data 158
8d387956c7cd638a// Expected Data 159
2075024a7ae44ced// Expected Data 160
c358b666b4c10dc4// Expected Data 161
db0ee40fc9af7285// Expected Data 162
7cc77214cc922c89// Expected Data 163
b543923d6dfc8c2b// Expected Data 164
06abbc1238d54c2b// Expected Data 165
6d589c96ae01dbf9// Expected Data 166
e1af6e8a9f782272// Expected Data 167
3e8578d843dfac06// Expected Data 168
8dab677c8046e21c// Expected Data 169
f9c40ed0ccb12d48// Expected Data 170
d4cdb247a2111c1e// Expected Data 171
990cb7edd75062c7// Expected Data 172
86d1a59d4e5d0138// Expected Data 173
65477403459e3378// Expected Data 174
```

```
dbe83891ab4bcbc5// Expected Data 175
b46704a186791607// Expected Data 176
f5b9070cc501e110// Expected Data 177
f94212fd039c5ffc// Expected Data 178
1a4494c8cdc8f159// Expected Data 179
d58c9faf7053ad27// Expected Data 180
f70f81ddc3bcaf48// Expected Data 181
068cbdf176dbca7b// Expected Data 182
7789b2925f251082// Expected Data 183
e75f0b4d2c09d80a// Expected Data 184
3ebe2958239eb314// Expected Data 185
c7d60b0e73ce508e// Expected Data 186
56e35588d6a44c18// Expected Data 187
1daffbd57a53a596// Expected Data 188
f2f7102d3be7688a// Expected Data 189
700e8e8deeb66748// Expected Data 190
b4ff2e8f92b69eff// Expected Data 191
63c2496e27872e19// Expected Data 192
5b20fb3637c92100// Expected Data 193
1214f5800cd4a20e// Expected Data 194
a20529540b73ca16// Expected Data 195
2457088356006378// Expected Data 196
47cea2f88f29e5c3// Expected Data 197
e2fe6f9ac843497a// Expected Data 198
03b66cbe7e42875e// Expected Data 199
d15f5e0685274672// Expected Data 200
9f2ef338b1e9fbd6// Expected Data 201
14f4615e325cd5ac// Expected Data 202
506f22c0b2904fe8// Expected Data 203
f7911b9bdbdcc41b// Expected Data 204
4424c9630875a6dd// Expected Data 205
f18162e698452549// Expected Data 206
2ec46d76ad56a451// Expected Data 207
5791b4a5061c474f// Expected Data 208
a5dc1059c3433af2// Expected Data 209
```

```
f7bdfc632eff5d40// Expected Data 210
e5ec6de2ad9d9ae1// Expected Data 211
3ad21a6fa3c848d6// Expected Data 212
caec7dfe7cc84409// Expected Data 213
2193d2e012458301// Expected Data 214
6f17acc69a71d8f0// Expected Data 215
607af63c5a4c4609// Expected Data 216
213c9865e1079e8d// Expected Data 217
7b02cd51347c150c// Expected Data 218
4a190672e7f79f86// Expected Data 219
d9b25429c2db0cbd// Expected Data 220
1c12c53db0252814// Expected Data 221
b97340368779c2d5// Expected Data 222
f3f734635ac2087c// Expected Data 223
e284a41c2a8265bc// Expected Data 224
2ba058f71b4f36d3// Expected Data 225
e2aa63b6705f931d// Expected Data 226
fecf335b744f124c// Expected Data 227
ba40ff0c73f3fe99// Expected Data 228
b911cb024bb88890// Expected Data 229
8e57c90efe4f6276// Expected Data 230
7f5b9b1111ae1e69// Expected Data 231
b7fc6625827c2ccc// Expected Data 232
a6184c275019dabd// Expected Data 233
c72943f29e0f1a1d// Expected Data 234
0ca21a15a8823556// Expected Data 235
a4a3ac4da5db101d// Expected Data 236
bc0fae8c9467e180// Expected Data 237
93f210bb513417a6// Expected Data 238
fbb36a1068527ef7// Expected Data 239
6c269f1f21a92a08// Expected Data 240
a41ca772ff76366c// Expected Data 241
9702d41c145e43f3// Expected Data 242
8628336fef7bd4fd// Expected Data 243
3628ecb2f8c45efb// Expected Data 244
```

```
    bb20a4cf29f63d47// Expected Data 245
    57e81639f800026c// Expected Data 246
    7439103d5578d9a2// Expected Data 247
    dc0d178ecb2ab397// Expected Data 248
    14d3d911142bfe7d// Expected Data 249
    ffc10a2f366e086c// Expected Data 250
    528cc486e8846501// Expected Data 251
    cfe18b1c60f1c617// Expected Data 252
    3098b0a4157eb14d// Expected Data 253
    463ee22f68e09774// Expected Data 254
    cd5b07b6b7a62c10// Expected Data 255
end_memory
```

A.2 DES_ECB_HMAC_MD5_Encrypt

```
encrypt_type : DEU & MDEU


begin_descriptor:
    20031e22      // Header (type 0010)
    8                    // 8 bytes key for ipad
    @q0                  // pointer to key (address 0)
    20                   // 32 bytes hash-only input
    @p1                  // pointer to hash-only input (address 100)
    8                    // 8 bytes key for sdes
    @q2                  // pointer to sdes key (address 0)
         0                    // no IV in ECB
         0                    // no IV in ECB
    3e0                                    // snooped data size
    @p4                                    // Pointer to snooped data
    3e0                                    // encrypted data size
    @p5                                    // Pointer to encrypted data
    10                                     // HMAC output size
    @q6                                    // Pointer to HMAC output
    0                                      // NIL Pointer to next descriptor
end_descriptor
```

```
begin_memory q0:
    0101010101010101// HMAC Key
end_memory

begin_memory p1:
    00d931dde5a5f895// Data 0
    195601a51c127fdd// Data 1
    ea34384f1053862e// Data 2
    4f1dd86cff88d34b// Data 3
end_memory

begin_memory q2:
    0101010101010101// SDES Key
end_memory

begin_memory p4:
    5614fbb267e7b920// Data 4
    ef3871d780935755// Data 5
    2f5104affadec56c// Data 6
    6072d8a59b279f0d// Data 7
    0a5abd71021b03d9// Data 8
    51d93d7cb3504242// Data 9
    e5219acd7e1b06b8// Data 10
    28bd656b280f5df1// Data 11
    a1eb5d6e8dccd0ad// Data 12
    d163ad5227f8d5e6// Data 13
    5e1a593fbde3bfec// Data 14
    cd65d179438356f3// Data 15
    a97f03202f989f2b// Data 16
    e60b6fa168e09d88// Data 17
    98126a845d279ee1// Data 18
    ec1ad723d58e9a32// Data 19
    973cd25725e2fce7// Data 20
    5dd6f27f81f5a912// Data 21
```

```
199cdc38adc384a4// Data 22
72adf81e8a0ae0fb// Data 23
6313520794070d75// Data 24
af2f4c729cedfe64// Data 25
602b8e323b262bf0// Data 26
52b8109a5a55649d// Data 27
d75552ed0bff06d1// Data 28
a5648c136b2c65e1// Data 29
468fec86115828e4// Data 30
361a2de2edf5b5ae// Data 31
5c0cec8a56d743e9// Data 32
4bb0546f27c898df// Data 33
6f696c0f68e460b1// Data 34
b84a9c7db05207fa// Data 35
0285bc6d032b3aca// Data 36
cf9bb57b5105095e// Data 37
2607d9913beb4e81// Data 38
9f9c913215db494d// Data 39
2106cff8c35feb25// Data 40
6f1c0d62c0206aab// Data 41
ca2cf998bc0de979// Data 42
0ebb7280ddce6e86// Data 43
a8643e2f6f53548b// Data 44
6bb8955597d351ea// Data 45
31de4245acc6ffca// Data 46
6c7990df2d5ad48d// Data 47
d0c20e885ed52910// Data 48
a9be9d6323cb865d// Data 49
5f0c7cae53a81c1d// Data 50
28328f24292333ce// Data 51
42b94fe2abd10584// Data 52
0742ca9080d743e6// Data 53
238a7437991b2248// Data 54
54fdfa61bd0b7cdd// Data 55
c4b50d571a29bc2f// Data 56
```

```
126ee2e4d7307ce0// Data 57
a1908e8e25e25309// Data 58
eef2ebcec41b715b// Data 59
f6859e6d1e3f08cc// Data 60
fe2d0dd56788fdd2// Data 61
8f7092ce22eae706// Data 62
d64bba4ab4406b16// Data 63
0000000000000080// Data 64
0000000000000040// Data 65
0000000000000020// Data 66
0000000000000010// Data 67
0000000000000008// Data 68
0000000000000004// Data 69
0000000000000002// Data 70
0000000000000001// Data 71
0000000000008000// Data 72
0000000000004000// Data 73
0000000000002000// Data 74
0000000000001000// Data 75
0000000000000800// Data 76
0000000000000400// Data 77
0000000000000200// Data 78
0000000000000100// Data 79
0000000000800000// Data 80
0000000000400000// Data 81
0000000000200000// Data 82
0000000000100000// Data 83
0000000000080000// Data 84
0000000000040000// Data 85
0000000000020000// Data 86
0000000000010000// Data 87
0000000080000000// Data 88
0000000040000000// Data 89
0000000020000000// Data 90
0000000010000000// Data 91
```

```
0000000008000000// Data 92
0000000004000000// Data 93
0000000002000000// Data 94
0000000001000000// Data 95
0000008000000000// Data 96
0000004000000000// Data 97
0000002000000000// Data 98
0000001000000000// Data 99
0000000800000000// Data 100
0000000400000000// Data 101
0000000200000000// Data 102
0000000100000000// Data 103
0000800000000000// Data 104
0000400000000000// Data 105
0000200000000000// Data 106
0000100000000000// Data 107
0000080000000000// Data 108
0000040000000000// Data 109
0000020000000000// Data 110
0000010000000000// Data 111
0080000000000000// Data 112
0040000000000000// Data 113
0020000000000000// Data 114
0010000000000000// Data 115
0008000000000000// Data 116
0004000000000000// Data 117
0002000000000000// Data 118
0001000000000000// Data 119
8000000000000000// Data 120
4000000000000000// Data 121
2000000000000000// Data 122
1000000000000000// Data 123
0800000000000000// Data 124
0400000000000000// Data 125
0200000000000000// Data 126
```

```
    0100000000000000// Data 127
end_memory

begin_memory exp_p5:
    0000000000000008// Expected Data 4
    0000000000000004// Expected Data 5
    0000000000000002// Expected Data 6
    0000000000000001// Expected Data 7
    0000000000008000// Expected Data 8
    0000000000004000// Expected Data 9
    0000000000002000// Expected Data 10
    0000000000001000// Expected Data 11
    0000000000000800// Expected Data 12
    0000000000000400// Expected Data 13
    0000000000000200// Expected Data 14
    0000000000000100// Expected Data 15
    0000000000800000// Expected Data 16
    0000000000400000// Expected Data 17
    0000000000200000// Expected Data 18
    0000000000100000// Expected Data 19
    0000000000080000// Expected Data 20
    0000000000040000// Expected Data 21
    0000000000020000// Expected Data 22
    0000000000010000// Expected Data 23
    0000000080000000// Expected Data 24
    0000000040000000// Expected Data 25
    0000000020000000// Expected Data 26
    0000000010000000// Expected Data 27
    0000000008000000// Expected Data 28
    0000000004000000// Expected Data 29
    0000000002000000// Expected Data 30
    0000000001000000// Expected Data 31
    0000008000000000// Expected Data 32
    0000004000000000// Expected Data 33
    0000002000000000// Expected Data 34
```

```
0000001000000000// Expected Data 35
0000000800000000// Expected Data 36
0000000400000000// Expected Data 37
0000000200000000// Expected Data 38
0000000100000000// Expected Data 39
0000800000000000// Expected Data 40
0000400000000000// Expected Data 41
0000200000000000// Expected Data 42
0000100000000000// Expected Data 43
0000080000000000// Expected Data 44
0000040000000000// Expected Data 45
0000020000000000// Expected Data 46
0000010000000000// Expected Data 47
0080000000000000// Expected Data 48
0040000000000000// Expected Data 49
0020000000000000// Expected Data 50
0010000000000000// Expected Data 51
0008000000000000// Expected Data 52
0004000000000000// Expected Data 53
0002000000000000// Expected Data 54
0001000000000000// Expected Data 55
8000000000000000// Expected Data 56
4000000000000000// Expected Data 57
2000000000000000// Expected Data 58
1000000000000000// Expected Data 59
0800000000000000// Expected Data 60
0400000000000000// Expected Data 61
0200000000000000// Expected Data 62
0100000000000000// Expected Data 63
00d931dde5a5f895// Expected Data 64
195601a51c127fdd// Expected Data 65
ea34384f1053862e// Expected Data 66
4f1dd86cff88d34b// Expected Data 67
5614fbb267e7b920// Expected Data 68
ef3871d780935755// Expected Data 69
```

```
2f5104affadec56c// Expected Data 70
6072d8a59b279f0d// Expected Data 71
0a5abd71021b03d9// Expected Data 72
51d93d7cb3504242// Expected Data 73
e5219acd7e1b06b8// Expected Data 74
28bd656b280f5df1// Expected Data 75
a1eb5d6e8dccd0ad// Expected Data 76
d163ad5227f8d5e6// Expected Data 77
5e1a593fbde3bfec// Expected Data 78
cd65d179438356f3// Expected Data 79
a97f03202f989f2b// Expected Data 80
e60b6fa168e09d88// Expected Data 81
98126a845d279ee1// Expected Data 82
ec1ad723d58e9a32// Expected Data 83
973cd25725e2fce7// Expected Data 84
5dd6f27f81f5a912// Expected Data 85
199cdc38adc384a4// Expected Data 86
72adf81e8a0ae0fb// Expected Data 87
6313520794070d75// Expected Data 88
af2f4c729cedfe64// Expected Data 89
602b8e323b262bf0// Expected Data 90
52b8109a5a55649d// Expected Data 91
d75552ed0bff06d1// Expected Data 92
a5648c136b2c65e1// Expected Data 93
468fec86115828e4// Expected Data 94
361a2de2edf5b5ae// Expected Data 95
5c0cec8a56d743e9// Expected Data 96
4bb0546f27c898df// Expected Data 97
6f696c0f68e460b1// Expected Data 98
b84a9c7db05207fa// Expected Data 99
0285bc6d032b3aca// Expected Data 100
cf9bb57b5105095e// Expected Data 101
2607d9913beb4e81// Expected Data 102
9f9c913215db494d// Expected Data 103
2106cff8c35feb25// Expected Data 104
```

```
    6f1c0d62c0206aab// Expected Data 105
    ca2cf998bc0de979// Expected Data 106
    0ebb7280ddce6e86// Expected Data 107
    a8643e2f6f53548b// Expected Data 108
    6bb8955597d351ea// Expected Data 109
    31de4245acc6ffca// Expected Data 110
    6c7990df2d5ad48d// Expected Data 111
    d0c20e885ed52910// Expected Data 112
    a9be9d6323cb865d// Expected Data 113
    5f0c7cae53a81c1d// Expected Data 114
    28328f24292333ce// Expected Data 115
    42b94fe2abd10584// Expected Data 116
    0742ca9080d743e6// Expected Data 117
    238a7437991b2248// Expected Data 118
    54fdfa61bd0b7cdd// Expected Data 119
    c4b50d571a29bc2f// Expected Data 120
    126ee2e4d7307ce0// Expected Data 121
    a1908e8e25e25309// Expected Data 122
    eef2ebcec41b715b// Expected Data 123
    f6859e6d1e3f08cc// Expected Data 124
    fe2d0dd56788fdd2// Expected Data 125
    8f7092ce22eae706// Expected Data 126
    d64bba4ab4406b16// Expected Data 127
end_memory

begin_memory exp_q6:
    0341d84dbd50198f// expected hmac 0
    bfa5d9977ccffa56// expected hmac 1
end_memory
```

A.3 AES_CBC_HMAC_SHA256_Decrypt

```
encrypt_type : AESU & MDEU
```

```
begin_descriptor:
    60231d22      // Header (type 0010)
    10                  // 16 bytes key for HMAC
    @p0                 // pointer to key (address 0)
    10                  // 16 bytes hash-only input
    @p1                 // pointer to hash-only input (address 100)
    10                  // 16 bytes key for aes
    @q2                 // pointer to aes key (address 0)
        10                  // iv
        @q3                 // iv
    70                                      // snooped data size
    @p4                                     // Pointer to snooped data
    70                                      // encrypted data size
    @p5                                     // Pointer to encrypted data
    28                                      // HMAC output size
    @q6                                     // Pointer to HMAC output
    0                                       // NIL Pointer to next descriptor
end_descriptor

begin_memory q0:
        1010101010101010
        0101010101010101
end_memory

begin_memory p1:
        ec48626ea2128b3d
        e522ff8ad2c12729
end_memory

begin_memory q2:
        1010101010101010
        0101010101010101
end_memory

begin_memory q3:
```

```
        ec48626ea2128b3d
        e522ff8ad2c12729
end_memory

begin_memory p4:
        f8257cdc5f9039ec
        64a2502cd5c474a2
        0b0f02d6f1e6de6a
        4a51720efc1e03d3
        9c2993dd2aa76c0e
        2fbafffb45416f39
        cee237ea45be7de8
        6f2123e3cbbccd90
        aa16e4d7f0d1f927
        e815c160651486fb
        97db907d2b937a30
        24d4da494aa3f402
        afc0402298039c46
        2e2b5968e8cc80cb
end_memory

begin_memory exp_p5:
        c0998e7669a4e826
        ca317de4aa223532
        b21e152becde2c01
        4e1535870c61c41f
        b91c9448e81ae13f
        645628085b45c2d0
        733fdb12b9ac0353
        43c1e34d49cdf356
        99b639ab483a3e9a
        2f8ce9047e291fa6
        6fe651a05838aec2
        10a10cf1134101f8
        e2eb95fee50e184e
```

```
        fa1d3be3eb0f797d
end_memory

begin_memory exp_q6:
        f2fa4b462458b974
    6193ac7c0566aa07
    a477157327be5089
    d8a10aa1c99db722
        0000000000000080
end_memory
```

A.4 HMAC-MD-5

Note: This HMAC-MD-5 descriptor example does not work in combination with the ARC-4 descriptor in Section A.5, “ARC-4 Encrypt” to perform a TLS operation.

```
encrypt_type: MDEU

begin_descriptor:
    31E00010      // Header (type 0001)
    0                              // Adr0 Size Zero
    0                              // NIL Pointer Adr0
    20                             // Adr1 Size 20
    @q1                            // pseudo-reset context Pointer Adr1
    10                             // Adr2 Size
    @q2                            // Pointer Adr2
    8                              // Adr3 Size
    @p3                            // Pointer Adr3
    0                              // Adr4 Size Zero
    0                              // NIL Pointer Adr4
    10                             // Adr5 Size
    @q5                            // Pointer Adr5
    0                              // Adr6 Size Zero
    0                              // NIL Pointer Adr6
    0                              // NIL Pointer to next descriptor
```

```
end_descriptor

begin_memory q1:
    efcdab8967452301// B,A
    1032547698badcfe// C,D
    00000000c3d2e1f0// E
    0000000000000000// counter
end_memory

begin_memory q2:
    0b0b0b0b0b0b0b0b// Key 0
    0b0b0b0b0b0b0b0b// Key 1
end_memory

begin_memory p3:
    6572656854206948// Data 0
end_memory

begin_memory exp_q5:
    1cbb38367a729492// null 0
    9dfc8b15f88ef413// null 1
end_memory
```

A.5 ARC-4 Encrypt

Note: This ARC-4 descriptor example does not work in combination with the HMAC-MD-5 descriptor in Section A.4, “HMAC-MD-5” to perform a TLS operation.

```
encrypt_type : AFEU

begin_descriptor:
    10000050      // Header (type 0101)
    0                               // Adr0 Size Zero
    0                               // NIL Pointer Adr0
    0                               // Adr1 Size Zero
    0                               // NIL Pointer Adr1
```

```
    A                           // key size = 10d bytes
    @q2                         // pointer p1 to key
    56                          // data size = 86d (0x56) bytes
    @p3                         // pointer p2 to data in
    56                          // data size = 86d (0x56) bytes
    @p4                         // pointer p3 to data out
    0                              // Adr5 Size Zero
    0                              // NIL Pointer Adr5
    0                              // Adr6 Size Zero
    0                              // NIL Pointer Adr6
    0                              // NIL Pointer to next descriptor
end_descriptor

begin_memory q2:
    92a3f1e14fabe563            // key 0
    000000000000495d            // key 1
end_memory

begin_memory p3:
    c2b2cd6cbf84c3fe            // data 0
    25c6bd503eb96a58            // data 1
    03857688d05f2982            // data 2
    d75a5904ee6df109            // data 3
    14264f01a8ba2f28            // data 4
    f939d1c995425a5e            // data 5
    62afcddda6cf755d            // data 6
    55287072d00c5d62            // data 7
    8ae5c6559ee505e2            // data 8
    cb444c309e6089a5            // data 9
    00004d2bb4b82c31            // data 10
end_memory

begin_memory exp_p4:
    94b01d17ebc854de            // expected data 0
    290ba6578fd2a606            // expected data 1
```

```
    22286e3472a039b5            // expected data 2
    6cff1075caabe379            // expected data 3
    c2e4a9607cfbd5ec            // expected data 4
    e48eae9921a35221            // expected data 5
    54178811ac9dcd8c            // expected data 6
    f45ead3216c42352            // expected data 7
    92919a1e612dfc1e            // expected data 8
    fcd1cc7028da007c            // expected data 9
    0000ff5ba1e3f93b            // expected data 10
end_memory
```

A.6 2 Descriptor AES-CCM Encrypt

```
//2-pass CCM encryption descriptor chain

encrypt_type : AESU

begin_descriptor:
    60300010      // Header (type -- 0001)   Configure for AES-CBC Encrypt
    0                   // Adr0 Size Zero
    0                   // NIL Pointer Adr0
    10                  // 16 bytes CTX input
    @p1                 // pointer to CTX input
    10                  // 16 bytes key
    @q2                 // pointer to key
    40                  // 64 bytes fifo input
    @p3                 // pointer to fifo input
    40                  // 64 bytes fifo output
    @p4                 // pointer to data to hash and decrypting
    0                   // Adr5 Size Zero
    0                   // NIL Pointer Adr5
    0                             // Adr6 Size Zero
    0                             // NIL Pointer Adr6
    0                             // NIL Pointer to next descriptor
end_descriptor
```

```
begin_memory p1:  // CBC IV input
         0102030400000059
         1800a5a4a3a2a1a0
end_memory

begin_memory q2:  // aesa input key
         c7c6c5c4c3c2c1c0
         cfcecdcccbcac9c8
end_memory

begin_memory q3:  // aesa fifo input
         0000000000000000        // 16 bytes of ZEROs
         0000000000000000
         0504030201000800        //Header length = 0008 (first 2 least significant bytes)
         0000000000000706        // followed by 8-byte header (padded to 16 bytes)
         0f0e0d0c0b0a0908        // followed 24 byte of payload data
         1716151413121110
         1f1e1d1c1b1a1918
         0000000000000000
end_memory

begin_memory exp_p4:// aesa fifo output
         39e203cad354c2f0        // Intermediate data (discard)
         779e394ca824bd70
         404aea28868bde48
         8c4abf95c242aa00
         4fc22ba6bcff890f
         33aa9687165f2113
         f36c92866a05b9f7        //MAC Tag bytes 1-8
         11aaef99c43d16fb//MAC Tag bytes 9-16 (discard)
end_memory

====================================
encrypt_type : AESU
```

```
begin_descriptor:
    60700010      // Header (type -- 0001)  Configure for AES-CTR Encrypt
    0                   // Adr0 Size Zero
    0                   // NIL Pointer Adr0
    18                  // 24 bytes CTX input
    @p1                 // pointer to CTX input
    0                   // Adr2 Size Zero
    0                   // NIL Pointer Adr2
    30                  // 48 bytes fifo input
    @p3                 // pointer to fifo input
    30                  // 48 bytes fifo output
    @p4                 // pointer to fifo output
    0                   // Adr5 Size Zero
    0                   // NIL Pointer Adr5
    0                             // Adr6 Size Zero
    0                             // NIL Pointer Adr6
    0                             // NIL Pointer to next descriptor
end_descriptor

begin_memory p1:  // CTR input
         0102030400000001        // Initial Counter value
         0000a5a4a3a2a1a0
         0000000000000080        // Modulus size of 2^128
end_memory

begin_memory q3:  // aesa fifo input
         f36c92866a05b9f7        // MAC Tag bytes 1-8
         0000000000000000        // Zero pad
         0f0e0d0c0b0a0908        // 24 bytes of payload data (padded)
         1716151413121110
         1f1e1d1c1b1a1918
         0000000000000000
end_memory
```

```
begin_memory exp_p4:// aesa fifo output
        160940106ed591a0        // Encrypted MAC Tag bytes 1-8
        xxxxxxxxxxxxxxxx        // Encrypted MAC Tag bytes 9-16 (don't care - discard)
        cff835e1361ac972        // 24 bytes of encrypted payload data
        e3875c0894a81c29
        3b3ae4c939c415cc
        xxxxxxxxxxxxxxxx// encrypted Zero-pad result (don't care - discard)
end_memory
```

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 2666 8080
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN2912
Rev. 1
04/2005

freescale™ semiconductor