Freescale Semiconductor
Application Note

AN3062
Rev. 0, 01/2006

MPC885 Family SEC Lite Programming

by Jay Azurin and Geoff Waters
DSD Applications and Systems Engineering
Freescale Semiconductor, Inc.
Austin, TX

This application note and the accompanying software in the `AN3062SW.zip` file assist users in programming the MPC885 Integrated Security Engine (SEC Lite). The software `.zip` file contains a simple SEC driver, information on initializing the MPC885 SEC Lite, and working software as a starting-point for design. In conjunction with the SEC driver, there is an Ethernet driver to demonstrate encryption and decryption of an Ethernet frame payload.

This application note assumes familiarity with the following manuals, which are available at the web site listed on the back cover of this document:

- *MPC885 PowerQUICC™ Reference Manual* (MPC885RM)
- *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture* (MPCFPE32B)

Contents

1 Basics of SEC Lite

To write software drivers for the SEC Lite, you need to know about the ID register, data coherency and caches, and interrupts.

1.1 ID Register

The SEC Lite acts as a memory-mapped peripheral within the MPC885/875. Before you configure the SEC Lite, verify that you can read the SEC identification register at (IMMR [14–15] = 10) + 1020. The expected result is 0x2000_0000.

1.2 Data Coherency and Caches

The SEC Lite has its own descriptor-driven DMA controller, and it acts as a master within the MPC875/885 to fetch items for encryption, including keys and data. SEC descriptors can be directly written to the SEC crypto-channel descriptor buffer or into memory. When descriptors are written to memory, the SEC is launched by writing a descriptor pointer to the crypto-channel fetch register. The SEC then drives the pointers to the MPC875/885 memory controller, and the memory controller returns data from the requested memory location.

The MPC875/885 caches do not implement hardware snooping to enforce coherency. Therefore, in some cases, there are channel errors because the SEC descriptor created by the CPU cannot be found at the memory location written to the SEC fetch register. EU errors occur when the descriptor is directly written to the channel descriptor buffer and the keys or data read by the SEC are incorrect. Both channel errors and EU errors can occur when the CPU generates or modifies data in its cache, but the data is not pushed to main memory before the SEC fetches it.

The simplest way to avoid these errors is to write the data needed by the security core to non-cached memory space. Otherwise, software must ensure that the data cache is flushed before it writes to the SEC channel fetch register. For details on flushing the caches, consult the application note AN3066, *MPC8XX Performance Driven Optimization of Caches and MMUs*.

Write-through mode can also be used so that store operations always update memory. Write-through mode is used when external memory and internal cache images must always agree, which is the case for security.

If you are uncertain whether initial SEC difficulties are related to cache coherency, disable the data cache by clearing the DEN bit in the MPC875/885 data cache control register (DC_CST) to 0. For a full description of this register, see the section on data cache control registers in the *MPC885 Reference Manual*. If disabling the data cache eliminates the problem, then the problem is coherency-related.

1.3 Interrupts

As Figure 1 shows, the SIU receives interrupts from internal sources, such as the FECs, SEC Lite, external pins, $\overline{\text{IRQ}}$[0–7], and other modules. The SIU has eight interrupt levels to set the priority of the SIU interrupt sources (see the chapter on the SIU in the *MCP885 Reference Manual*). The interrupt level associated with the SEC Lite interrupt is programmable via the CPTR[SEC_INT] register field, as described in the chapter on the SEC Lite Master/Slave Interface Module in the *MPC885 Reference Manual*.

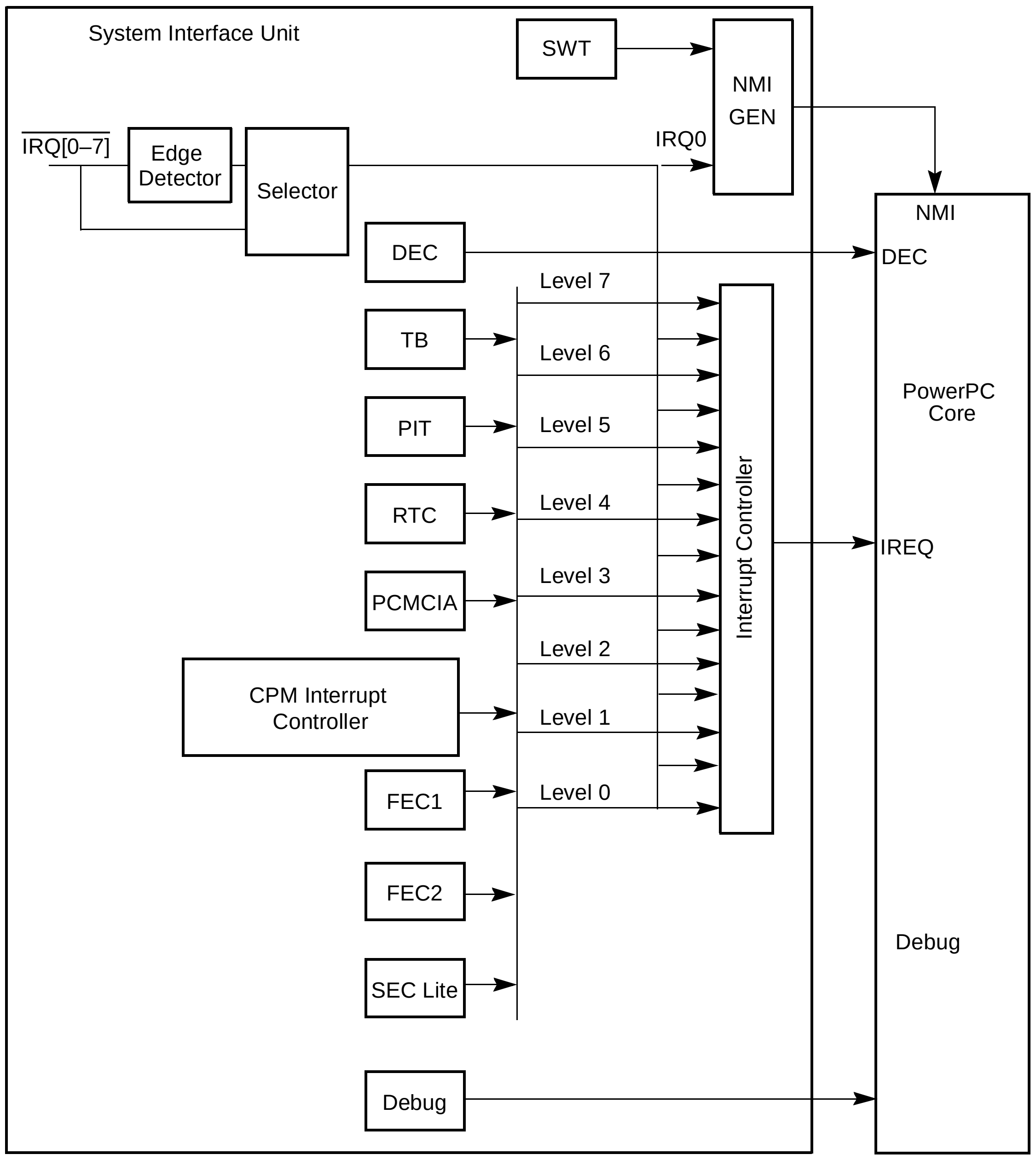

Figure 1. MPC885 Family Interrupt Structure

To ensure that the interrupt request from the SEC is properly reported to the MPC875/885 platform, perform the following steps:

1. In the SIU, set SIMASK so the SEC interrupt appears in the SIPEND register.
2. In the CPM, set the SEC_INT level in the CPTR register to correspond to the location of the interrupt in the interrupt hierarchy. Program this field with the default value of zero unless there are interrupt latency issues, in which case the SEC Lite interrupt level must be lower than the FECs or any peripheral of the CPM. (level 0 represents highest priority).
3. Ensure that there is an exception handler at offset 0x500 to handle the interrupt produced by the SEC Lite.

The SEC interrupt hierarchy consists of low-level sources (EUs), and high-level sources (channel and controller) that are aggregated to a single interrupt source from the SEC to the MPC875/885:

- Execution unit interrupt propagation and control:
  - Any EU error not specifically disabled via the EU interrupt control register (ICR) is reflected in the EU interrupt status register (ISR), and an interrupt signal is passed to the controller ISR and channel pointer status register.
  - Any EU errors not disabled via the EU interrupt control register (ICR) are reflected in the Halt and IE bits in the EU status register.
  - If an EU error is disabled via the EU ICR, the EU does not treat it as an error, and it continues processing. The corresponding bit in the EU ISR is not set.
- Crypto-channel interrupt propagation and control:
  - The channel generates interrupts for one of two reasons, ERROR and DONE.
  - Channel error interrupts cannot be disabled in the channel. Errors detected by the channel are reported in the channel pointer status register ERROR field. These errors cause the Channel Error bit to be set in the controller ISR.
  - Channel DONE interrupts can be disabled in the channel. The setting of the channel configuration register (CCCR) determines whether the channel signals DONE via an interrupt, a writeback of (0xFF) to the descriptor header in system memory, or neither.
- Controller interrupt propagation and control:
  - The controller interrupt status register reflects the status of all SEC interrupt sources, except for those disabled at a lower level. Recall that EU errors disabled in the EU ICR do not reach the controller ISR.
  - The controller interrupt mask register (IMR) controls which interrupt sources (as reported by the controller ISR) are allowed to assert the SEC IRQ to the MPC875/885 platform.
  - The controller IMR can block the channel from signalling DONE via an interrupt, but the CCCR totally controls the DONE notification via header writeback (0xFF).

For initial debug, DONE notification via interrupts should be enabled via the CCCR by setting NT = 1 and CDIE = 1. With all low-level EU interrupts enabled in the EU and the two top-level EU interrupts (ERROR and DONE) masked in the controller, the only interrupt sources remaining are the channel ERROR, DONE, and TEA. If the target of an SEC transaction signals TEA, the PQ1 SIU should also detect the TEA. Therefore, this is a somewhat redundant location to report the TEA, but it is potentially useful in determining why SEC-initiated transactions trigger a TEA. The controller master error address register stores the address from which the SEC was driving when the TEA was received.

2 Hardware Preparation

Two CodeWarrior™ projects are provided: `DUET_SEC_demo_875ADDER.mcp` and `DUET_SEC_demo_885ADS.mcp`. The project selected must correspond to the target board in use. For example, `DUET_SEC_demo_875ADDER.mcp` should be opened for use with the MPC875ADDER board. Also, either MPC885ADS or MPC875ADDER must be defined in the `fec.h` file, depending on the target board. Connecting the serial cable provided with the MPC875ADDER board from SK1 port to a serial port in the computer displays the terminal output of the demonstration programs. For the MPC885ADS board, the serial cable should be connected to the lower serial port on the ADS.

Following are the terminal settings for the host PC:

- Baud rate: 57600
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow control: None

3 Software Overview

The software encrypts and decrypts a message by calling simple routines. The message is displayed on the the terminal screen, and the user determines whether the message is correctly decrypted. To encrypt and decrypt an Ethernet frame payload, the software links the fast Ethernet controllers (FEC) and the MPC885 SEC Lite using simple API calls. The SEC driver contains simple routines used in this software example. The DPDinit() routine, which is the heart of the driver, sets up the data packet descriptor to encrypt or decrypt a buffer. The options selected for this DPD are predefined in the code.

When it receives Ethernet frames, the FEC driver interrupt handler calls a decrypt routine. This handler verifies whether the reception completes without errors. Each Ethernet frame is contained in a single data buffer linked to a single buffer descriptor (BD). Eight transmit and eight receive buffers are established. The size of the transmit buffer is programmed using the TX_BUFFER_SIZE definition. The 15-byte minimum allows for the 14-byte Ethernet header and one byte of data. For small frames, the FEC automatically pads the frame up to the minimum size of 64 bytes. Each Ethernet frame is received in a single buffer. The receive buffer size is programmable via the RX_BUFFER_SIZE definition. RX_BUFFER_SIZE must be divisible by 16, and receive buffers must begin on an address aligned on a 16-byte boundary. To guarantee that each transmitted frame is received in a single receive buffer, RX_BUFFER_SIZE must be greater than or equal to (TX_BUFFER_SIZE + 4).

The example sets TX_BUFFER_SIZE to 1514, which yields a maximum-size Ethernet frame of 1518 bytes after the FEC automatically calculates the 32-bit frame check sequence (also known as CRC-32) and appends it to the transmit frame. Correspondingly, RX_BUFFER_SIZE is the next size larger that is evenly divisible by 16 (1520, for example). Note that the FEC allows a single frame to span multiple buffers, both on receive and transmit. However, our example restricts the transmit and receive frames to a single buffer for simplicity.

The software includes demonstration programs to illustrate the flexibility in programming and use of the FECs in conjunction with the SEC Lite. These demonstration programs are not compliant to any specific security protocol, but are representative of the IPSec payload encryption and authentication. The demonstration programs use an API defined in the `fec.c` and `sec.c` files. These APIs can be used as low-level Ethernet and SEC drivers so that users can build their own demonstration programs. However, the code is not fully optimized and is presented here only for demonstration. All demonstration programs are interrupt driven, but the software can easily be converted to work in polling mode. To use polling mode, the interrupt handler code (processInterrupt()) is placed into the main routine within a software loop. The remainder of this section is an overview of the demonstration programs.

3.1 Decrypt the Message demonstration (Exercise 1)

To access the demonstration for decrypting a message, select the target called **EXERCISE 1** in the CodeWarrior project. The `encrypted_message.h` file contains an encrypted message that is decrypted with the keys in the `sec_app.h` file and the SEC API function decrypt(). The demonstration proceeds as follows:

1. Print out the encrypted message to the terminal.
2. Call the decrypt() API function to decrypt the encrypted message.
3. Print out the decrypted message on the terminal

3.2 Ethernet Frame Exchange demonstration (Exercise 2)

To access the demonstration for Ethernet frame exchange, select the **EXERCISE 2** target in the CodeWarrior project. The FECs send Ethernet frames to each other. The status of the demonstration is printed out on the terminal. This demonstration requires a crossover Ethernet cable between the two Ethernet ports on the board. Also, the ECHO_MODE and CUSTOM_HANDLER parameters in `fec.c` must be set to OFF.

3.3 Secure Ethernet demonstration (Exercise 3)

To access the secure Ethernet demonstration, select the **EXERCISE 3** target in the CodeWarrior project. Both the SEC API and the FEC API are used. An encrypted message is sent as part of the Ethernet frame data. The receiving FEC decrypts the message in the interrupt handler by calling the decrypt() function. The interrupt handler also prints out the decrypted message on the terminal. In the main routine, FEC1 and FEC2 exchange secure Ethernet frames. This test requires a crossover Ethernet cable between the two Ethernet ports on the board. Also, the ECHO_MODE parameter must be set to OFF and the CUSTOM_HANDLER parameter must be set to ON in `fec.c`. This is a generic demonstration of the use of the SEC in conjunction with the FECs to encrypt and authenticate packetized data.

4 sec.c Functions

Following are brief descriptions of the SEC Lite driver functions in `sec.c`:

• *DPDinit()*. Initializes the data packet descriptor for the SEC Lite. It takes as input pointers to the data in and data-out buffers, length of the data buffer, and a parameter to indicate either encryption or decryption of the data block. The routine also writes to the fetch register to trigger encryption/decryption. Before the routine completes, it polls on the CHADONE bit in the SEC_ISR register. By default, this function programs the data packet descriptor settings for the SEC Lite as follows:
  — DEU: 3DES
  — MDEU: HMAC
  — Initialization bit (INT set)
  — Message digest algorithm: SHA-1
  — Padding automatic message padding
  — HMAC snoop enabled

  The `sec_app.h` file contains the HMAC key, data, and SDES key for the encryption/decryption operations.

- *encrypt()*. Calls the DPDinit() routine with the encryption parameter set.
- *decrypt()*. Calls the DPDinit() routine with the decryption parameter set.

5 fec.c Functions

Following are brief descriptions of the FEC driver functions in `fec.c`:

- *GetIMMR()*. Returns the value of the IMMR[ISB] field. It determines the internal memory map base address.
- *InterruptHandler()*. Tests whether the interrupt is an external interrupt and whether the interrupt level corresponds to the FECs interrupt level. If everything checks OK, the function calls ProcessInterrupt() to handle the interrupt if a receive event occurs on either FEC.
- *FECInit()*. Initializes either FEC1 or FEC2, depending on the value of the FECnum parameter. Also, the FEC can be configured in loopback mode depending on the loop_mode parameter. The default initialization for the FECs is 100 Mbps full duplex in MII mode.
- *Delay()*. Implements a simple software delay.
- *EnableEE() and DisableEE()*. Enables or disables external interrupts by writing to special-purpose registers EID and EIE, respectively. A write to these bits changes the state of MSR[EE].
- *InterruptInit()*. Copies the external interrupt code to the vector defined for the external interrupt handler (0x500).
- *LoadTxBuffer()*. Prepares the transmit buffers with the appropriate destination address, source address, and type/length fields required for an Ethernet frame.
- *InitBDs()*. Initializes BD rings to point Rx BDs to the Rx buffer pool and Tx BDs to the Tx buffer pool. This function also initializes the buffer descriptor control and data length fields. It ensures that transmit and receive functions are disabled before BDs are initialized. This function defines a number of buffers for an Rx and Tx buffer pool, but it does not attempt to manage memory. It uses the first half of the BD pool for Rx and the second half for Tx.
- *FECsend()*. Sends a frame to a destination indicated by the destination parameter. In our application, the destination and source are limited to FEC1 and FEC2. Another parameter of this function is the pattern of the transmit buffer, which is predetermined as shown in Table 1:

Table 1. Predefined Tx Buffer Patterns for FECSend()

Parameter	Pattern
FIVES	0x55
ALLAs	0xAA
ALLOs	0x00
ALLFs	0xFF
INCWALKINGONES	0x01020408...800102...
DECWALKINGONES	0x80402010...018040...
INCFROMZERO	0x01020304...FF0102...
DECFROMFFs	0xFFFDFEFC...00FFFD...

- *resetPHY_MPC875ADDER()*. Resets the AM79C874 NetPHY-1LP Ethernet PHY on the MPC872 ADDER board.
- *enablePHY_MPC885ADS()*. Enables the Ethernet PHYs on the MPC885ADS board by writing to the BSCR5 register.
- *ProcessInterrupt()*. Processes the interrupts generated by receive events from the FECs. The main purpose of this handler is to check the RxBDs for errors and update the global counters. This function can be configured for Echo mode by turning on the ECHO_ON flag in `fec.c`. When this feature is enabled, the FEC echoes any frame it receives back to the source of the frame.
- *printStatus()*. Prints out the contents of the global counters.
- *blinkLED_MPC875ADDER() and blinkLED_MPC885ADS()*. Toggles a debug LED on the MPC875ADDER and MPC885ADS boards.
- *FECGracefulTxStop() and FECGracefulTxResume()*. Gracefully stops the transmitter. These functions are used in the printStatus() routine to print out accurate data.

6 Development Environment

The following development tools were used:

- CodeWarrior 8.1 compiler and debugger.
- WireTAP 8xx (a BDM debugger)
- MPC875ADDER development board (also tested on the MPC885ADS board)
- Windows 2000 development platform

NOTE

These development tools are not an expressed or implied endorsement and are not meant to communicate preference of one manufacturer’s product over another.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

Document Number: AN3062
Rev. 0
01/2006