Freescale Semiconductor
Application Note

Document Number: AN3075
Rev. 1, 06/2006

MPC8555E Security Quick Start Guide

by *Ahsan Kabir, Geoff Waters, and Dai Haruki*
NCSG/DSD Systems Engineering
Freescale Semiconductor

This application note provides a brief overview of the PowerQUICC™ III security (SEC) module, including initial configuration, followed by an example of how to build and run the Freescale Linux-based security driver on the MPC8555CDS platform.

Contents

1 SEC Basics

Most information in this section is generic and applies to all versions of the SEC implemented in PowerQUICC III devices, such as the MPC8555E, MPC8541E, MPC8548E, and so on.

The SEC is a crypto coprocessor, with bus mastering capabilities that connects to the main internal bus of the PowerQUICC. The e500 CPU core uses the SEC crypto acceleration by creating descriptors to step the SEC through the requested encryption and/or authentication operation. The e500 core launches these requests and interacts with the SEC through a memory-mapped interface.

freescale™ semiconductor

1.1 SEC Availability Check

Before configuring or operating SEC 2.0, verify that you can read the SEC identification register at CCSRBAR + 0x3_1020. The expected result is 0x0000_0000_0000_0040. If you cannot read the SEC ID, you may have a version of PowerQUICC that does not include an SEC. For export control reasons, the MPC8555 (and other PowerQUICC devices) are available in *E* and *non-E* versions, with the *E* version including the SEC.

If you can successfully read the SEC ID, the SEC is present and is ready for bus mastering operations. Any write to a crypto channel fetch FIFO address triggers a bus master request from the SEC. Avoid such writes until additional configuration is complete.

No additional configuration is required to enable SEC bus mastering or snooping of SEC transactions by the MPC8555E coherency module. Both are enabled by default and cannot be disabled. This is a difference between the SEC 2.0/MPC8555E and other SEC/PowerQUICC implementations.

1.2 Platform Configuration to Receive SEC IRQ

The SEC has a single $\overline{\text{IRQ}}$ signal to the MPC8555E platform. To enable the SEC to generate interrupt signals to the e500 core, you must ensure that this signal is properly reported to the MPC8555E platform. The SEC interrupt source is internal with respect to the PIC and is mapped as internal interrupt source number 29. To configure the MPC8555E PIC to enable SEC interrupts and route them to the CPU core, program the internal interrupt vector/priority register (IIVPR29) at offset 0x5_05A0 from CCSRBAR. Figure 1 shows the bit fields of this register.

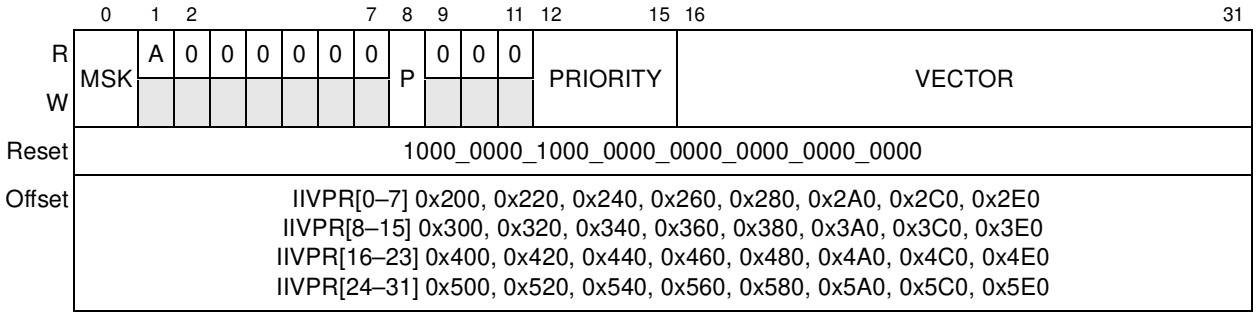

Figure 1. IIVPR Register Bit Fields

We configure the IIVPR bit fields as follows:

— MSK = 0, allowing interrupts from SEC to occur.

— A = 1 to indicate that an SEC interrupt is requested.

— PRIORITY = 1 to indicate that all internal interrupts are active-high. Clearing this bit disables the interrupt.

— VECTOR = assign any 16-bit unique integer value not shared with any other interrupt source.

The PIC routes SEC-generated interrupts to the e500 core as an external interrupt (IVOR4) if the PIC is properly initialized and the IIVPR is properly configured. Typically, IVOR4 is configured with a value of 0x500 and the IVPR is configured with a value of 0x0. The e500 core jumps to address 0x500 (0x500 + 0x0). The software handler in this location takes the following actions:

a) Reads the IACK register (at offset 0x600A0) to get the interrupt vector number.

An SEC handler is associated with this vector number during system initialization.

b) From 0x500, the SEC handler function is called.

When the SEC handler exits, it reads the EOI register (at offset 0x600B0) and ends the interrupt cycle.

1.3 SEC Configuration to Transmit Appropriate IRQs

All SEC low-level interrupts are disabled (masked) at reset and must be enabled (unmasked) by setting the corresponding bit in the controller interrupt mask register. The SEC interrupt hierarchy consists of low-level sources (EUs) and high-level sources (channels and controller) that are aggregated into a single interrupt source from the SEC to the MPC8555E:

- Execution unit interrupt propagation and control:
  - Any EU error not disabled in the individual EU interrupt control register (ICR) is reflected in the EU interrupt status register (ISR), and an interrupt signal is passed to the controller interrupt status register and channel pointer status register.
  - Any EU errors not disabled in the ICR are reflected in the HALT and IE bits in the EU status register.
  - If an EU error is disabled in the ICR, the EU does not treat it as an error and continues processing. The corresponding bit in the ISR is not set.
- Crypto-channel interrupt propagation and control:
  - Channels can generate 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 and cause the channel error bit to be set in the controller ISR.
  - Channel done interrupts can be disabled in the channel. The channel configuration register (CCCR) determines whether the channel signals DONE through an interrupt, a writeback of (0xFF) to the descriptor header in system memory, or neither.
- Controller interrupt propagation and control:
  - The controller ISR reflects the status of all SEC interrupt sources except 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 $\overline{\text{IRQ}}$ to the MPC8555E PIC.
  - The controller IMR can block the channel from signalling DONE via interrupts, but DONE notification via header writeback (0xFF) is totally controlled by the SEC crypto channel configuration registers (CCCR).

For initial debugging, enable channel DONE notification through interrupts after each descriptor by setting CCCR[CDIE] in each channel. The controller IMR should be set to enable (unmask) channel DONE and ERROR interrupts, as well as the bus interface internal timeout (ITO) error. EU interrupts should be left masked because any serious EU error triggers a channel error, making the direct interrupts from the EUs unnecessary except in debug mode. When these configuration steps are complete, you are ready to run the first SEC descriptor. Appendix A presents an example of preparing an SEC descriptor, including known good inputs and expected output. To perform this test, define pointers for the descriptor and each of the inputs and outputs defined by the descriptor, place the appropriate inputs at those pointer locations, and

then write a pointer to the descriptor to the fetch FIFO of any of the four crypto-channels. Upon receiving the DONE interrupt from the SEC, check the result produced by the SEC against the expected result shown in Appendix A. Section 2 describes a more complete evaluation of SEC driver testing, including performance testing in a Linux environment.

2 Testing Procedure

You can obtain both the Freescale security driver code (ID number SEC2DRIVERS) and the *SEC 2.x Reference Device Driver User's Guide* (SEC2XSWUG) at the Freescale web site listed on the back cover of this document. Before downloading the code, you must register at the site and provide the user name and password. Alternatively, there is a security driver patch in the `AN3075SW.zip` file that you can apply to the LTIB kernel package to see the sources for the driver. In our testing procedure, we use the patch file. This section describes how you can run the driver on MPC8555CDS platform. It discusses the tools to work with the Linux-based security driver. A variety of tools are available for building uboot, kernel, modules, file systems, and so on. Commonly used tools are PCS, ELDK, and Linux Target Image Builder (LTIB). This application note describes how to enable the SEC performance test module in LTIB environment. The commands/status printed on the screen are shown here in courier font. These commands can run on any Linux host (x86 machine) running RedHat or the Fedora core. Install the iso image (`MPC8555CDS_20060124-ltib.iso`) shipped in your Linux host (for example, an x86 machine running Fedora Core 4.0) as follows:

1. As root, mount the ISO image on your machine using one of the following commands:
   - `mount -o loop MPC8555CDS_20060124-ltib.iso /mnt/cdrom`
   - `mount -o loop MPC8555CDS_20060124-ltib.iso /media/cdrom`
2. As a non-root user, install the LTIB: `/mnt/cdrom/install`.
3. Input the desired LTIB installation path when prompted.

   The script installs LTIB into two different directories on your machine:

   — `/opt/freescale/pkgs` contains all packages including the Linux kernel, uboot and application packages.

   — `<install_path>/ltib` contains the main LTIB scripts and specification files for the MPC8555CDS BSP.
4. Configure LTIB using the following command on your Linux host:

   `#./ltib --configure`

   You are prompted to save the new configuration file, which brings up the menu again with several other options as shown in Figure 3. Be sure to make the following menu selections:

   a) Uncheck BUILD A BOOTLOADER.

   b) Check KERNEL (LINUX 2.6.11).

   c) Check PACKAGE SELECTION → PACKAGE LIST → MODUTILS (MODULE-INIT-TOOLS).

   d) Check TARGET IMAGE GENERATION → OPTIONS → CREATE A KERNEL THAT CAN BE BOOTED WITH UBOOT.

   e) Check TARGET IMAGE GENERATION → OPTIONS → CREATE A RAMDISK THAT CAN BE USED WITH UBOOT.

   When you save and exit from this menu configuration, LTIB gets the kernel tree, builds the kernel image, builds ramdisk, and deletes the source tree for the kernel.

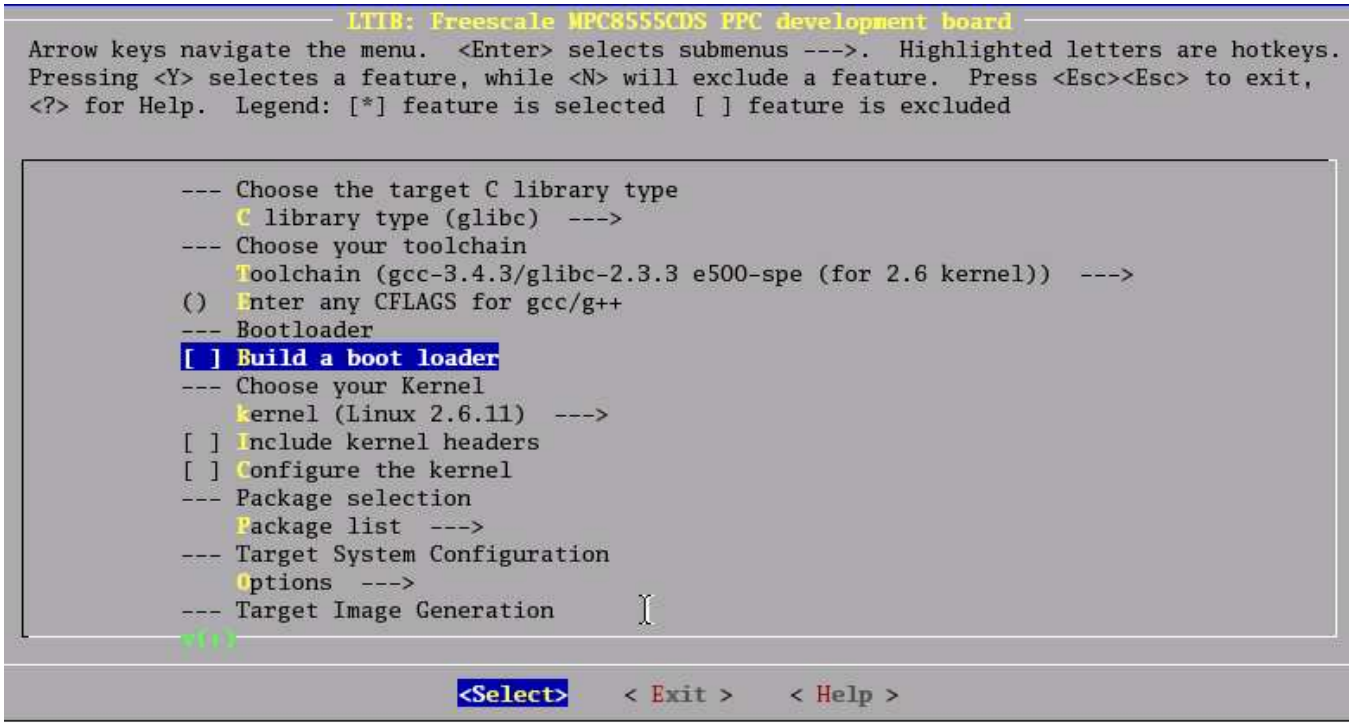

Figure 2. LTIB Menu Configuration, Package Selection

2.1 Obtain the Kernel Tree

The files in the `AN3075SW.zip` file available at the Freescale web site are as follows:

- `AN3075.pdf`: This application note.
- `MPC8555CDS_20060124-ltib.iso`: LTIB iso image that should be installed on your Linux host.
- `kernel-2.6.11-sec.patch`: The patch file you will need to apply to your LTIB kernel source tree.
- `.config`: Working kernel configuration file.
- `uImage.bin`: Tested functional kernel image for use with uboot.
- `ramdisk.bin`: Tested functional ramdisk image for use with uboot.

To add the security driver, you need the kernel tree. First, you must find the exact package name of the kernel.

2.1.1 Get the Kernel Package Name

Example 1 shows how you can get the kernel package name:

Example 1. Get Kernel Package Name

```
#./ltib -m listpkgs | grep kernel

iptables-1.2.11-1       iptables                  n    GPL        Tools for managing kernel
packet filtering capab

kernel-2.6.11-0         kernel-2.6.11-pq3         y    GPL        Linux kernel (core of the
Linux operating system
```

```
lkc-1.4-2               lkc                          n    GPL         Linux kernel configuration
language parser
module-init-tools-3.1-0. module-init-tools           y    GPL         Linux 2.6 kernel module
management utilities.
sysklogd-1.4.1-1        sysklogd                     n    GPL         System logging and kernel
message trapping daemo
```

2.1.2 Get the Kernel Source Tree

Example 1 shows that the name of the kernel package is `kernel-2.6.11-pq3`. Here is an example of how you can get the kernel package source tree.

Example 2. Get the Kernel Source Tree

```
#./ltib -m prep -p kernel-fsl-2.6.11-pq3
```

The code in Example 2 places the kernel tree in the `$(LIB_INSTALL)/rpm/BUILD/linux-2.6` directory, where `$LIB_INSTALL` is the directory in which LTIB is installed.

2.1.3 Apply the Patch

Applying the patch onto the existing kernel adds the sec2 driver code (the main driver code), the sec2 test code, and the sec2 performance code. Example 3 shows how to apply the patch. In this case, `$(UNZIP_DIR)` is the directory where you extracted the Freescale `.zip` file.

Example 3. Apply the Patch

```
#cd $(LIB_INSTALL)/rpm/BUILD/linux-2.6.10
#patch -p1 < $(UNZIP_DIR)/kernel-2.6.11-sec.patch
```

After the patch is applied, the sec2 code is located in the following directories:

- sec2 driver code $(LIB_INSTALL)/rpm/BUILD/linux-2.6.10/drivers/sec2 directory
- sec2 test code in $(LIB_INSTALL)/rpm/BUILD/linux-2.6.10/drivers/sec2x-test directory
- sec2 performance code in $(LIB_INSTALL)/rpm/BUILD/linux-2.6.10/sec2x-perf directory.

2.2 Build the Kernel

Build the kernel using LTIB with the kernel configuration option (-c), as illustrated in Example 4.

Example 4. Build the Kernel

```
#./ltib -m scbuild -p kernel-2.6.11-pq3 -c
```

The menu configuration window appears, as shown in Figure 3. Select the DEVICE DRIVERS menu entry. Under DEVICE DRIVERS in Figure 3, SEC2.X OPTIONS is the driver code that is compiled and linked with the kernel. The SEC2.X TEST OPTIONS and SEC2.X PERFORMANCE TESTING OPTIONS are selected as loadable modules, as shown in Figure 4, Figure 5, and Figure 6.

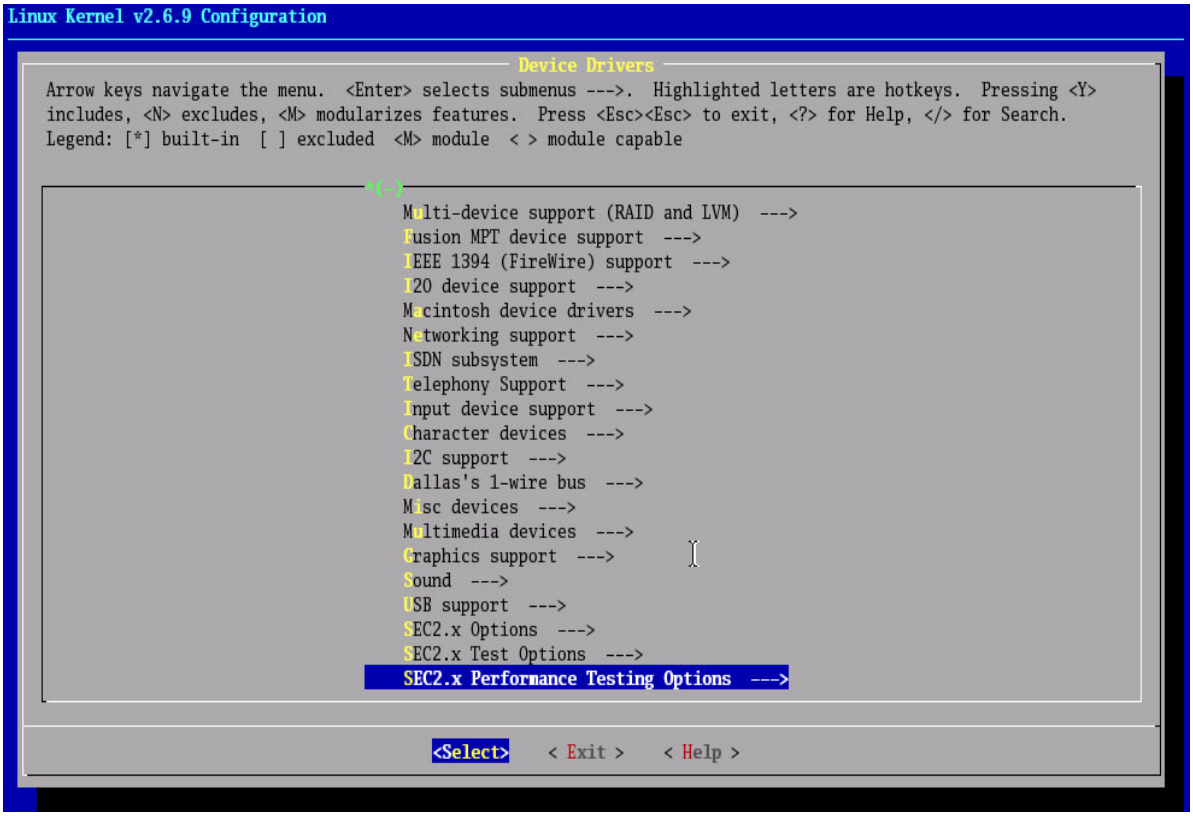

Figure 3. Kernel Build Menuconfig Showing the Security Options

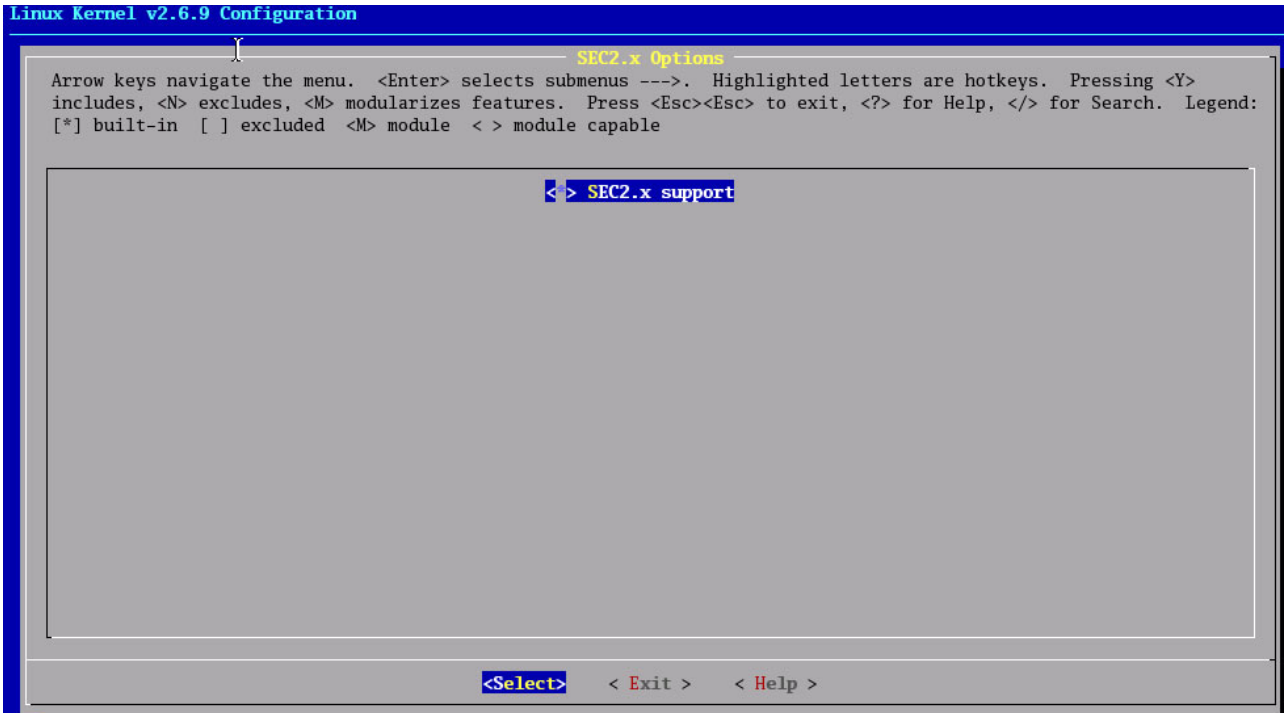

Figure 4. Driver Code Integrated in the Kernel Image

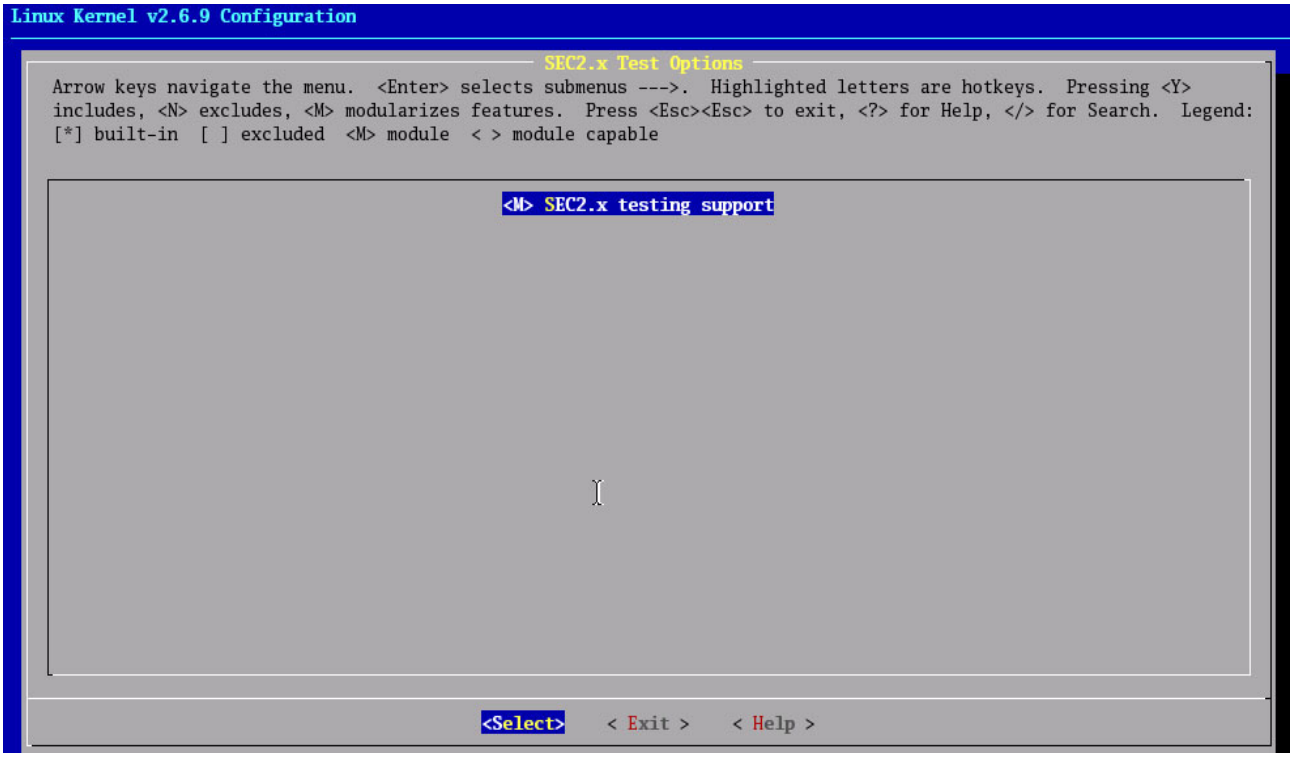

Figure 5. Driver Basic Test Code Configured as a Loadable Module

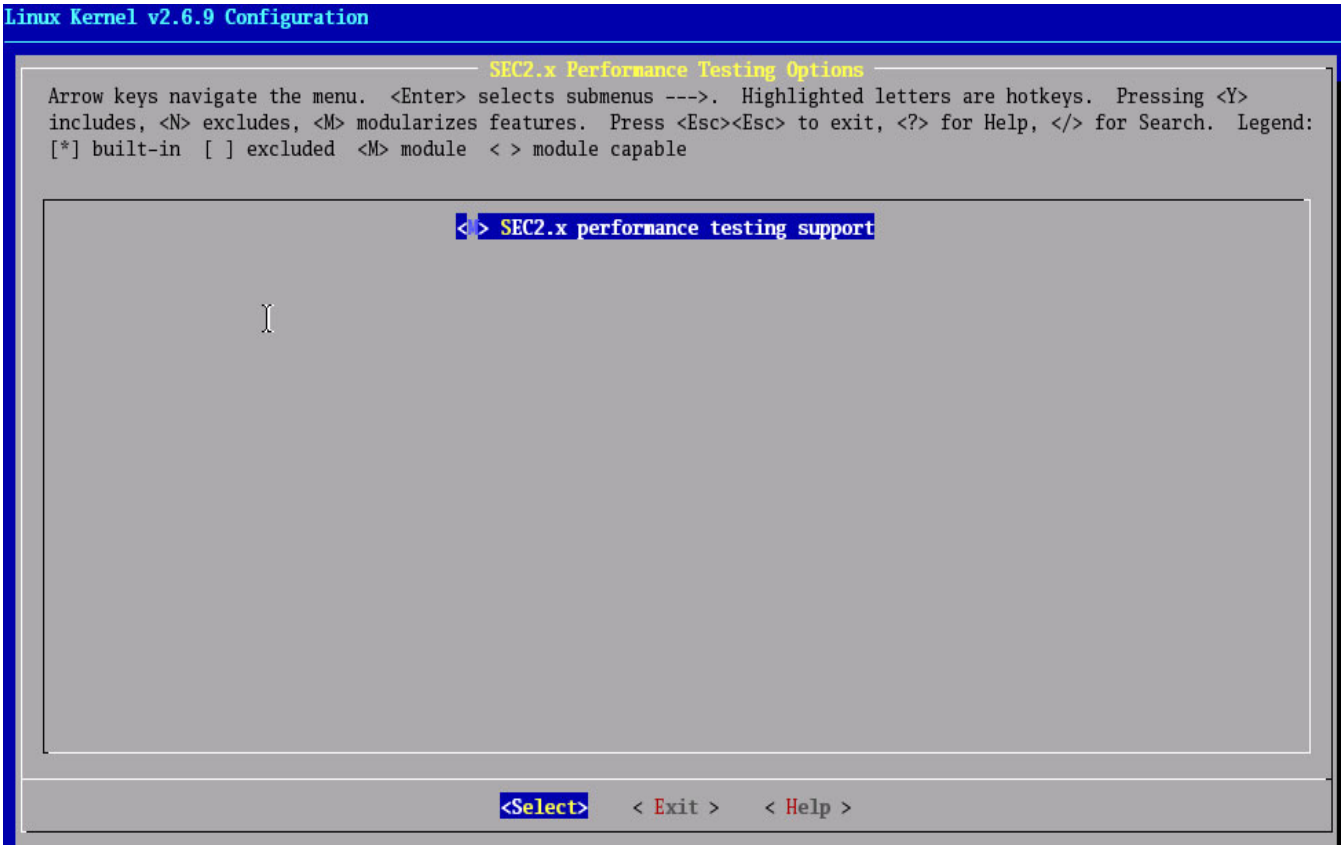

Figure 6. Driver Performance Test Configured as a Loadable Module

In addition to making the SEC-related configuration change, you must ensure that FREESCALE 85XX OPTIONS -> ARCADIA X3.1 BOARD support is disabled if you do not have the latest Arcadia X3.1 board for your MPC8555CDS platform. Consult the platform user's manual to determine the type of your motherboard. The other menu entries should be easy to configure when you are familiar with features that the MPC8555 processor and the MPC8555CDS platform can support. For example, the MPC8555CDS platform does not support the SCSI interface, parallel interface and so on. After you finish with menu configuration, save the changes and exit. At this point the kernel build starts.

2.3 Deploy the Kernel

After the kernel is built, use LTIB for deployment. The LTIB deployment function creates a kernel image (`vmlinux.gz.uboot`) in your `$(LTIB_INSTALL)/` directory that can run from uboot. It also creates a ramdisk image (`rootfs.ext2.gz.uboot`) in your `$(LTIB_INSTALL)/` directory. Issue the kernel deployment command as follows:

```
#./ltib -m scdeploy -p kernel-2.6.11-pq3
```

LTIB provide the ramdisk size information at the terminal when you run this command. The sample terminal information for ramdisk is shown in Example 5. This information is useful when you set up environment variables from uboot.

Example 5. ramdisk Terminal Information

```
Your ramdisk exceeds the old default size of 4096k, you may need to
set the command line argument for ramdisk_size in your bootloader
to at least 12320k. For instance, for uboot:
setenv bootargs root=/dev/ram rw ramdisk_size=12320
```

3 Execution and Interpretation of Results

The security driver test environment consists of an x86 machine running Fedora Core 4.0 and an MPC8555CDS system (see Figure 7). Serial cable from the MPC8555CDS is connected to the serial port of the Linux host. An Ethernet connection is also established between the host and the target. A minicom is used on the Linux host to see the target uboot and kernel status/error.

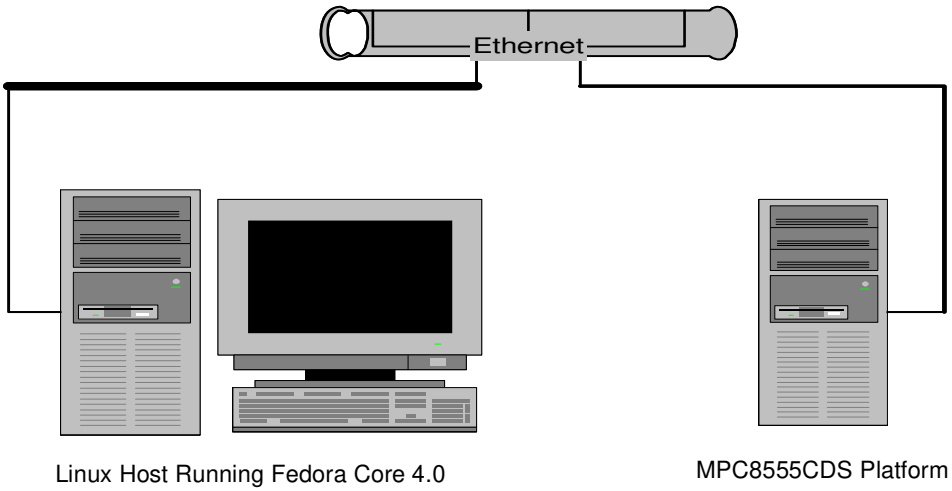

Figure 7. Driver Testbed

After LTIB deployment, copy the kernel image and ramdisk image from the `$(LIB_INSTALL)/` directory to the `/tftpboot/` directory, renaming `vmlinux.gz.uboot` as `uImage.bin` and `rootfs.ext2.gz.uboot` as `ramdisk.bin`. We then power on the host, invoke the minicom, and set the baud rate for the minicom to 115200. Next, we power on the MPC8555CDS platform. The host minicom terminal displays information on uboot bootup status. We stop the uboot from autobooting by pressing any key.

Example 6. Getting Started

```
U-Boot 1.1.3 (FSL pq3-20050505-0) (May 6 2005 - 13:40:57)


CPU:   8541, Version: 1.1, (0x807a0011)
Core:  E500, Version: 2.0, (0x80200020)
Clock Configuration:
         CPU: 528 MHz, CCB: 264 MHz,
         DDR: 132 MHz, LBC:  66 MHz
L1:      D-cache 32 kB enabled
         I-cache 32 kB enabled
Board: CDS Version 0x11, PCI Slot 3
CPU Board Revision 0.0 (0x0000)
         PCI1: 32 bit, 33 MHz, sync
         PCI2: 32 bit, 66 MHz, sync
I2C:   ready
DRAM:  Initializing
         SDRAM: 64 MB
         DDR: 256 MB
FLASH: 16 MB
L2 cache 256KB: enabled
In:    serial
Out:   serial
Err:   serial
Net:   ENET0: PHY is Cicada Cis8204 (fc446)
ENET1: PHY is Cicada Cis8204 (fc446)
ENET0, ENET1
Hit any key to stop autoboot:  0
=>
```

3.1 Set Up uboot Environment Variables

When you see the uboot prompt, set up the environment variables shown in Example 7. Some environment variables in the following list may differ on your testbed depending upon the IP address you assign to your host, target, and so on.

Example 7. Set Up uboot Environment Variables

```
=>set ipaddr 10.82.119.154
=>set serverip 10.82.119.151
=>set netmask 255.255.255.0
=>set bootfile uImage.bin
=>set ramdiskfile ramdisk.bin
=>set ramdisk_size ramdisk_size=YYYYY
=>set bootargs root=/dev/ram rw console=ttyS0,115200 \$ramdisk_size
=>set bootcmd tftp 100000 \$bootfile\; tftp 2000000 \$ramdiskfile\; bootm 1000000 2000000\;
=>save
```

YYYYY is ramdisk size reported by LTIB after deployment. To put $ and ; in a variable, use \ like \$. The save command saves all the environment variables that you modified into the on-board Flash memory of the MPC8555CDS system. The next time the system boots, it is not necessary to modify environment variables because they are stored in Flash memory.

3.2 Boot Linux

After you have modified the environment variables, you are ready to boot the Linux kernel. Type boot and press the Enter key, and the Linux kernel boots in response. The listing in Example 8 shows the boot status. Notice that it first downloads the kernel image from tftp server to the RAM area of the MPC8555CDS system. Next, it downloads the ramdisk image from tftp server to the RAM area of the MPC85555CDS system. When uboot has both the kernel image and ramdisk image, it proceeds to boot the Linux kernel. When booting completes, the Linux prompt appears.

Example 8. Linux Boot Status

```
=>boot

ENET0: No link.

Speed: 100, full duplex
Using ENET1 device
TFTP from server 10.82.119.151; our IP address is 10.82.119.154
Filename '301/uImage.bin'.
Load address: 0x1000000
Loading: #################################################################
         #################################################################
         #################################################################
         ##############################################
done
Bytes transferred = 1226188 (12b5cc hex)
Speed: 100, full duplex
Using ENET1 device
TFTP from server 10.82.119.151; our IP address is 10.82.119.154
Filename '301/ramdisk.bin'.
Load address: 0x2000000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
done
Bytes transferred = 2654634 (2881aa hex)
## Booting image at 01000000 ...
         Image Name:   Linux for mpc8555cds
         Image Type:   PowerPC Linux Kernel Image (gzip compressed)
         Data Size:    1226124 Bytes =  1.2 MB
         Load Address: 00000000
         Entry Point:  00000000
         Verifying Checksum ... OK
         Uncompressing Kernel Image ... OK
         ## Loading RAMDisk Image at 02000000 ...
```

```
Image Name:   uboot ext2 ramdisk rootfs
Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
         Data Size:    2654570 Bytes =  2.5 MB
         Load Address: 00000000
         Entry Point:  00000000
         Verifying Checksum ... OK
         Loading Ramdisk to 0fd27000, end 0ffaf16a ... OK
Memory CAM mapping: CAM0=256Mb, CAM1=0Mb, CAM2=0Mb residual: 0Mb
Linux version 2.6.9 (r9aahw@psys02.am.freescale.net) (gcc version 3.4.3) #6 Mon Dec 12 14:38:25
CST 2005
mpc85xx_cds_setup_arch
board frequency is 528000000
CDS Version = 11 in PCI slot 3
Built 1 zonelists
Kernel command line: root=/dev/ram rw console=ttyS1,115200 $ramdisk_size
         Entry Point:  00000000
         Verifying Checksum ... OK
         Loading Ramdisk to 0fd27000, end 0ffaf16a ... OK
Memory CAM mapping: CAM0=256Mb, CAM1=0Mb, CAM2=0Mb residual: 0Mb
Linux version 2.6.11 (r9aahw@psys02.am.freescale.net) (gcc version 3.4.3) #4 Tue Apr 4 14:38:25


CDT 2006
mpc85xx_cds_setup_arch
board frequency is 528000000
CDS Version = 11 in PCI slot 3
Built 1 zonelists
Kernel command line: root=/dev/ram rw console=ttyS1,115200 $ramdisk_size
OpenPIC Version 1.2 (1 CPUs and 44 IRQ sources) at fbf78000
PID hash table entries: 2048 (order: 11, 32768 bytes)
Console: colour dummy device 80x25
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
Memory: 254080k available (1892k kernel code, 500k data, 312k init, 0k highmem)
Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
checking if image is initramfs...it isn't (no cpio magic); looks like an initrd
Freeing initrd memory: 2592k freed
NET: Registered protocol family 16
PCI: Probing PCI hardware
Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
PCI: Via IRQ fixup for 0000:01:02.2, from 99 to 3
PCI: Via IRQ fixup for 0000:01:02.3, from 99 to 3
PCI: Via IRQ fixup for 0000:01:02.5, from 98 to 2
PCI: Via IRQ fixup for 0000:01:02.6, from 98 to 2
Generic RTC Driver v1.07
Macintosh non-volatile memory driver v1.1
serio: i8042 AUX port at 0x60,0x64 irq 12
serio: i8042 KBD port at 0x60,0x64 irq 1
Serial: 8250/16550 driver $Revision: 1.90 $ 6 ports, IRQ sharing disabled
ttyS0 at MMIO 0xe0004500 (irq = 90) is a 16550A
ttyS1 at MMIO 0xe0004600 (irq = 90) is a 16550A
RAMDISK driver initialized: 16 RAM disks of 131072K size 1024 blocksize
loop: loaded (max 8 devices)
Intel(R) PRO/1000 Network Driver - version 5.3.19-k2
Copyright (c) 1999-2004 Intel Corporation.
eth0: Gianfar Ethernet Controller Version 1.1, 00:e0:0c:00:00:fd
eth0: Running with NAPI enabled
```

```
eth0: 256/256 RX/TX BD ring size
eth1: Gianfar Ethernet Controller Version 1.1, 00:e0:0c:00:01:fd
eth1: Running with NAPI enabled
eth1: 256/256 RX/TX BD ring size
atkbd.c: keyboard reset failed on isa0060/serio1
atkbd.c: keyboard reset failed on isa0060/serio0
i2c /dev entries driver
NET: Registered protocol family 2
IP: routing cache hash table of 2048 buckets, 16Kbytes
TCP: Hash tables configured (established 16384 bind 32768)
arp_tables: (C) 2002 David S. Miller
NET: Registered protocol family 1
NET: Registered protocol family 17
RAMDISK: Compressed image found at block 0
VFS: Mounted root (ext2 filesystem).
Freeing unused kernel memory: 312k init
Setting the hostname to freescale
Mounting filesystems
Starting syslogd and klogd
Running depmod
Setting up networking on loopback device:
eth0: PHY is Cicada Cis8204 (fc446)

Setting up networking on eth0:
Adding static route for default gateway to 192.168.0.1:
Setting nameserver to 192.168.0.1 in /etc/resolv.conf:
Setting time from ntp server: ntp.cs.strath.ac.uk
ntp.cs.strath.ac.uk: Host name lookup failure
Starting inetd:
Starting the port mapper:
Starting the boa webserver:
/ #
```

3.3 Run the Security Performance Module

The security performance module includes tests that perform the following function 50,000 times:

- SEC Driver (Request Builder)
- Check for available channels and EUs
- Check request valid
- Reserve empty descriptor from pool
- Translate request into descriptor format
- Write descriptor pointer to SEC channel and wait while SEC takes the following actions:
  - Reads and decodes descriptor
  - Internally assigns resources
  - Fetches keys, context
  - Reads data
  - **Processes data**
  - Writes data
  - Writes context (optional)

  - Signals DONE through an interrupt
- Interrupt service routine
  - Check for errors, if none `normal completion` message sent to message queue
- Post request service routine:
  - Free channels, free descriptor
  - Check for next request

This test mimics the behavior of a security protocol stack that processes one packet at a time. In this test, the CPU waits while the SEC executes the descriptor. However, in a real application, the CPU can be performing other tasks not dependent on completion of the current packet/descriptor. A security protocol stack that begins processing the next packet and prepares and launches the descriptor for that packet before the first descriptor is done has higher performance.

Note that this test does not include packet Rx or Tx overhead or any security protocol stack overhead. The test begins with packet-like data already in memory and with a known request to the SEC (AES-ECB, 3DES-HMAC-SHA-1, or AES-HMAC-SHA-1). The test does not lock descriptors, keys, or other context in the e500 L2 cache, and it is representative of the caching behavior of a packet processing application.

The performance results shown in Example 12 were measured with the system configuration shown in Example 6, specifically, a 528 MHz e500 core, a 132 MHz DDR clock rate, and a 132 MHz SEC. Changing these frequencies has a nearly linear effect on the throughput results.

The directory structure of your embedded Linux should look like the one shown in Example 9.

Example 9. Linux Directory Structure

```
/ # ls
bin      home             lost+found       proc     sys         var
dev      lib              mnt              root     tmp
etc      linuxrc          opt              sbin     usr
```

Before you run the security performance module, ensure that the sec2 driver is part of the kernel. Expect to see sec2 as a result of the command shown in Example 10.

Example 10. sec2 Display

```
/ # cat /proc/devices
Character devices:
         1 mem
         2 pty
         3 ttyp
         4 /dev/vc/0
         4 tty
         4 ttyS
         5 /dev/tty
         5 /dev/console
         5 /dev/ptmx
         7 vcs
         10 misc
         13 input
         89 i2c
         128 ptm
```

```
        136 pts
        254 sec2

Block devices:
        1 ramdisk
        7 loop
```

After verifying that the sec2 driver is part of the kernel, you are ready to run the sec2 performance test. The listing in Example 11 shows how to verify that you have the right module file in the `sec2x-perf` directory.

Example 11. Verify the Module File in the sec2x-perf Directory

```
/ # cd lib/modules/2.6.11/kernel/drivers/
/lib/modules/2.6.9/kernel/drivers # ls
input       sec2x-perf  sec2x-test
/lib/modules/2.6.9/kernel/drivers # cd sec2x-perf/
/lib/modules/2.6.9/kernel/drivers/sec2x-perf # ls
sec2perfTest.ko
```

When you install the security performance module, `sec2perfTest.ko`, using insmod, the security performance test runs automatically. The perfTest routine in the `/sec2x-perf/perfTest.c` file executes during module initialization. This is the major routine that subsequently calls the aesPerf and ipsecPerf routines to run crypto operations and provide the performance numbers at the console. The listing in Example 12 shows how to obtain performance numbers for crypto operations.

Example 12. Obtain Performance Number for Crypto Operations

```
/ # cd lib/modules/2.6.11/kernel/drivers/
/lib/modules/2.6.11/kernel/drivers # ls -l
drwxr-xr-x    3 root     root         1024 Dec 16  2005 input
drwxr-xr-x    2 root     root         1024 Dec 16  2005 sec2x-perf
drwxr-xr-x    2 root     root         1024 Dec 16  2005 sec2x-test
/lib/modules/2.6.11/kernel/drivers # cd sec2x-perf/
/lib/modules/2.6.11/kernel/drivers/sec2x-perf # ls
sec2perfTest.ko
/lib/modules/2.6.11/kernel/drivers/sec2x-perf # insmod sec2perfTest.ko
sec2perfTest: module license 'Freescale Restricted' taints kernel.
AES ECB 64 byte packet, 38 Mbps (667369 uS)
AES ECB 128 byte packet, 80 Mbps (637237 uS)
AES ECB 256 byte packet, 141 Mbps (724620 uS)
AES ECB 512 byte packet, 255 Mbps (801156 uS)
AES ECB 1024 byte packet, 459 Mbps (891116 uS)
AES ECB 2048 byte packet, 769 Mbps (1063912 uS)
AES ECB 4096 byte packet, 1165 Mbps (1405636 uS)
IPsec 3DES CBC + SHA1 64 byte packet, 30 Mbps (837015 uS)
IPsec 3DES CBC + SHA1 128 byte packet, 60 Mbps (851264 uS)
IPsec 3DES CBC + SHA1 256 byte packet, 115 Mbps (887130 uS)
IPsec 3DES CBC + SHA1 512 byte packet, 212 Mbps (961593 uS)
IPsec 3DES CBC + SHA1 1024 byte packet, 491 Mbps (833772 uS)
IPsec 3DES CBC + SHA1 2048 byte packet, 583 Mbps (1404547 uS)
IPsec 3DES CBC + SHA1 4096 byte packet, 638 Mbps (2567492 uS)
IPsec AES CBC + SHA1 64 byte packet, 31 Mbps (824492 uS)
IPsec AES CBC + SHA1 128 byte packet, 60 Mbps (840953 uS)
IPsec AES CBC + SHA1 256 byte packet, 117 Mbps (874799 uS)
IPsec AES CBC + SHA1 512 byte packet, 216 Mbps (944608 uS)
IPsec AES CBC + SHA1 1024 byte packet, 378 Mbps (1083581 uS)
```

```
IPsec AES CBC + SHA1 2048 byte packet, 636 Mbps (1286452 uS)
IPsec AES CBC + SHA1 4096 byte packet, 699 Mbps (2340714 uS)
test elapsed time = 23.834859 seconds
accumulated time  = 22729458 microseconds
```

When you run the sec2 performance module, check your wristwatch from the time the module loads and starts until it finishes. The elapsed time number should be ± second from your wristwatch. Remove the modules using the rmmod command and then install the module again if you want to perform the test multiple times. The following line shows how to remove the module:

```
/lib/modules/2.6.11/kernel/drivers/sec2x-perf # rmmod sec2perfTest.ko
```

Appendix A Preparing a Security Descriptor

The *SEC 2.x Reference Device Driver User's Guide* (SEC2XSWUG) describes the driver architecture and layout. This application note explains how to build and run the Freescale security driver. Appendix A describes the basic steps involved in a crypto operation. The example shows how a security descriptor is prepared.

A.1 Descriptor Example

After you verify that the SEC ID register can be properly read and that the CCCR and controller IMRs are configured as recommended, the next step is to run the following golden descriptor. This descriptor and associated data can be used to verify that the SEC is working properly. The descriptor performs an HMAC-MD-5 operation.

```
encrypt_type : mdeu

begin_descriptor:
                          //   descriptor type = common_nonsnoop_no_afeu
                          //   authentication function = hmac-md5
                          //   direction = outbound
                          //   done notification = off
                          //   primary cha = mdeu

         31e00010         // Header word 1
         0                // Header word 2

         0                // 0 Length (unused)
         0                //   Extent (unused)
         0                //   Nil pointer 0

         0                // 1 Length of cipher context in = 0
         0                //   Extent (unused)
         0                //   Nil pointer 1

         3                // 2 Length of auth key = 3
         0                //   Extent (unused)
         @p2              //   Pointer to key

         3a5              // 3 Length of cipher data = 933
         0                //   Extent (unused)
         @p3              //   Pointer to cipher data
```

```
        0                // Nil length 4
        0                //   Extent (unused)
        0                //  Nil pointer 4

        10               // 5 Length of auth out = 16
        0                //   Extent (unused)
        @p5              //   Pointer to auth/context out

        0                // Nil length 6
        0                //   Extent (unused)
        0                //   Nil pointer 6

end_descriptor

begin_memory p2:                // auth key
        10771A0000000000
end_memory

begin_memory p3:                // cipher data in
        070694B844184E0E
        936BC56F86C2696E
        D4E4115902C7FD49
        BF3607E77BD5A21D
        BBE4904DBA485A39
        0B9490174A6230A2
        DDF4F52D450B2376
        900FAF7E6CE67DEC
        B27E84E492DF3BFD
        CB2EEC74D7402515
        951BE1B35CC3C295
        85F0152D875371AE
        294D29624F8468BF
        746078E0E5C44FF8
        BEC4C0DD3D245C53
        EF047B395C6F1927
        349DF070CB574B00
        39C93FBE8037182F
        9D7229EADE407463
        BD79D6F6AD0CBF79
        B758264D5ACAC010
        C1391D47BD2E0D56
        DD6C6B8052C1DA5D
        CCDF715387968447
        BF8393340E92BDA0
        353583C5DDAA5E65
        6AADFD5D5650372B
        8D28587B38DA803D
        FEE18D22E42CAD63
        1A0575083CF16CAD
        A50C014A87662865
        0C9301E8D4299C33
        CECBDE281ECFF722
        FC82F78A9CDFBCBA
        F2E1FD0F8C963C0B
        5EF2015EEAC612C5
        F4CF6BB638BF23B8
        680F9ECEFF6BBD9D
        15A6B7640B9D9290
        BF289FC224A08BEF
```

```
50B7950A5E974FC0
5428CBE86381D69D
9896C39B23647E1E
DBDECA83E62CE773
12B2693DB57016F5
C3DDA295DA0C335B
8F72D727E5723290
62870CC4588354A5
E03C877D27687C7B
3C2FEC0EFB386A23
6A7DA75439685939
85EB544E37020B6F
9C15A8E57A2B7894
001705337507D1FF
4A44D9DEEBE028CF
3B468F65BAD9C7B2
DDB143C9021659E4
D7D0CACBC40A9900
51E54DD408DB30FB
6046E11FBCBB20F2
7B0E58470D798CFC
89DCDBA5CE7CAFD6
60F0B9CCAB697D4F
40E81EB208747E73
71D9A7FF80118649
85FDE21FC2937D5D
26D0671667F44949
530A1F603B08A156
D5492EFDE89FC1B1
E4C2904ECAF6F3B5
00CC0EDCE2EBD3A4
40526554889E65E2
2ECF8160B4EA4161
3B43459033B61298
DCA44350D89401C5
4ED91FC0D1D4F38B
E40931947936581F
5DDDAFBE74168ABF
7FF92ABFAAC219C1
6C233E8256049FF0
70887E0016A6C35D
8B11928EF8E4F08B
B5B5B1069C31FB41
637CFC9FF371936B
23CDB185329B1387
DF8650A4B054433C
79C62AE36AB0C5F4
D0332B8A76D5A156
7D965D28F532048D
C0274DCB8C70A5F3
51034B80F0C34309
2DEDC61B7BE25753
9E946D05FE4B74E9
273A3BC908413185
3E508307F67D1BD7
37EC812DF38BAB2C
952EED2275FB7781
```

```
         3E9499A8641248EF
         2E563F6B70A38840
         5CF7792DEE6D4331
         F662401CEB1D4D16
         18E6B3CB0022EC34
         A6BF7A71222068E4
         DB9B9A30237D9F1B
         449DB1B392CE8F28
         57302FD3DA4EB73A
         1B9A47C1B881458A
         A8BB6547A9EE428C
         9E966438C2033726
         B4033E4A23FF209F
         72611BCC61A1D8E9
         A6089294C3641A55
         80CC6A7ABDCF8A16
         B789777293B7BBCA
         249CD1EDE05AAA2C
         71C8398E0250BE0A
         EC3B5F762E000000
end_memory

begin_memory exp_p5:                // authentication code out
         84C5EA35D0AF8926
         80B8BF81B4326E01
end_memory
```

If the HMAC value written out to p5 matches the expected result shown here, you are well on your way to accelerated cryptographic operations. If the output does not match the expected results or the descriptor generates an error, this application note has provided some solid avenues for investigation. Contact your local field applications engineer (FAE) or Freescale support at the address listed on the back cover of this document for answers to your questions on MPC8555E operation, including the SEC.

4 Revision History

Table 1 provides a revision history for this application note. Note that this revision history table reflects the changes to this application note template, but can also be used for the application note revision history.

Table 1. Document Revision History

Rev. Number	Date	Description
0	4/2006	Initial release.
1	6/2006	Corrected the name of the software file that accompanies this application note.

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
1-800-521-6274
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 3 5437 9125
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
1-800-441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN3075
Rev. 1
06/2006

freescale™
semiconductor