Freescale Semiconductor
Application Note

Document Number:AN4785
Rev 0, 08/2013

Modifying DPAA Offloading Applications

Contents

1 Introduction

This document supplies several methods for modifying the QorIQ USDPAA offloading applications using the DPA offloading driver API.

The document explores four target use cases providing detailed and interactive examples. These include:

- USDPAA classifier demo application
- USDPAA IP fragmentation demo application
- USDPAA IP reassembly demo application
- USDPAA IPSec offloading application

Readers should be familiar with either the P4080, B4860 or B4420 *QorIQ Integrated Multicore Communication Processor Family Reference Manual*. Additionally, information regarding the QorIQ Software Design Kit (SDK) may be found online at `/www.freescale.com/infocenter/`

The following table provides a short summary of each example.

Table 1. Summary of USDPAA offloading applications

Application	Description
USDPAA classifier demo application	Provides an example on how to use basic frame manager features such as parse, classify, distribution and header manipulation.

Table continues on the next page...

Table 1. Summary of USDPAA offloading applications (continued)

Application	Description
USDPAA IP fragmentation demo application	Illustrates how to configure basic IP fragmentation provided by the Frame Manager. For frame manager version 3 devices the reassembly application also demonstrates the Virtual Storage Profile configuration and selection for reassembled flows.
USDPAA IP reassembly demo application	Explains how to configure basic IP reassembly provided by the Frame Manager. For Frame Manager version 3 devices the reassembly application also demonstrates the Virtual Storage Profile configuration and selection for reassembled flows.
USDPAA IPSec offloading application	Reveals how to use standard Linux tools to configure IPSec offloading using the DPA offloading driver.

2 Altering the classifier_demo application

This application demonstrates how packet classification can be offloaded to the frame manager by using the DPA offloading driver. It shows several functionalities of the classifier and stats components.

2.1 Overview

The application performs the following operations on an Exact Match table that has its entries managed by key:

- Create an Exact Match Table
- Flush an Exact Match Table
- Free an Exact Match Table
- Insert entry in an Exact Match Table
- Import header manipulations created by `fmclib`
- Set up header manipulation on a table entry
- Modify the header manipulation associated with a table entry
- Modify entry by Key in an Exact Match Table
- Delete entry by Key in an Exact Match Table
- Get Statistics by Key in an Exact Match Table using DPA Classifier API
- Create a DPA Stats class counter for the entire Exact Match Table
- Retrieve synchronously and asynchronously the DPA Stats counter

The application has four stages and each stage is meant to validate different functionalities. The user can switch from one stage to another by typing the command “next_stage” in the USDPAA PPAM command line. When a frame is received, if the table lookup process results in a HIT condition, the frame is be sent to a specific frame queue. Once the frame is in a frame queue, the CPU dequeues it, displays information related to its characteristics and information about the queue (`fqid`) it was received on, and sends it back to be transmitted on the interface it was received from. The tested Exact Match table is managed by key, and each operation that is performed on the table uses the lookup key to identify the entry to work on.

A header manipulation operations chain is attached to each of the entries which are inserted in the table. The header manipulations are defined in the PCD configuration files provided with the application and are imported by the DPA Classifier. Each header manipulations chain consists of a forwarding header manipulation (L2 header update) followed by an L3 header update operation. The values in each header manipulation chain are updated at runtime as the application progresses through its different stages.

During each stage the user has the possibility of performing a number of actions on the DPA Stats class counter by entering the following commands in the USDPAA PPAM command line:

Table 2. DPA Stats class counter actions

Command Line	Description
get_stats	retrieve the statistics for the created counters in an asynchronous mode and print the returned values.
get_stats_sync	retrieve the statistics for the created counters in a synchronous mode and print the returned values.
reset_stats	reset the statistics for the created counters.

In order to have a successful operation, the user must verify that the returned values of the counters statistics matches the number of sent frames for each stage of the test.

Classifier demo is a USDPAA application that requires the following resources:

- `classif_demo_config.xml` - config file
- `classif_demo_policy.xml` - policy file

This user space application initializes the classification schemes using the FMC library, and afterwards it manages the DPA Classifier table.

The application receives traffic on the configured Ethernet port. Lookups are performed in the DPA Classifier table. If the result is Hit, the frame is sent to a specific frame queue on a software portal. A CPU dequeues the frame, displays information about the packet and about the queue it was received on, and sends the packet back to be transmitted on the interface from which it was received. If the result is Miss, the frame is silently dropped.

NOTE

Classifier demo application does not use the cores or the IP stack for performing classification (look-up actions) or header manipulation operation (protocol operations). It uses the frame manager features for traffic classification and header manipulation.

2.2 Running classifier_demo

Classifier_demo application supports the following platforms:

- P4080
- B4860
- B4420

To run the application, you may need to use a specific device tree file that comes with the Linux DPA offloading driver. The following shows you how to produce this device tree file for your platform.

2.2.1 Application environment specifications

This application uses the setup pictured bellow with the following connections:

Table 3. Port Connections

SoC	Traffic Port
P4080	fm1-gb0
B4860	fm0-mac5
B4420	fm0-mac3

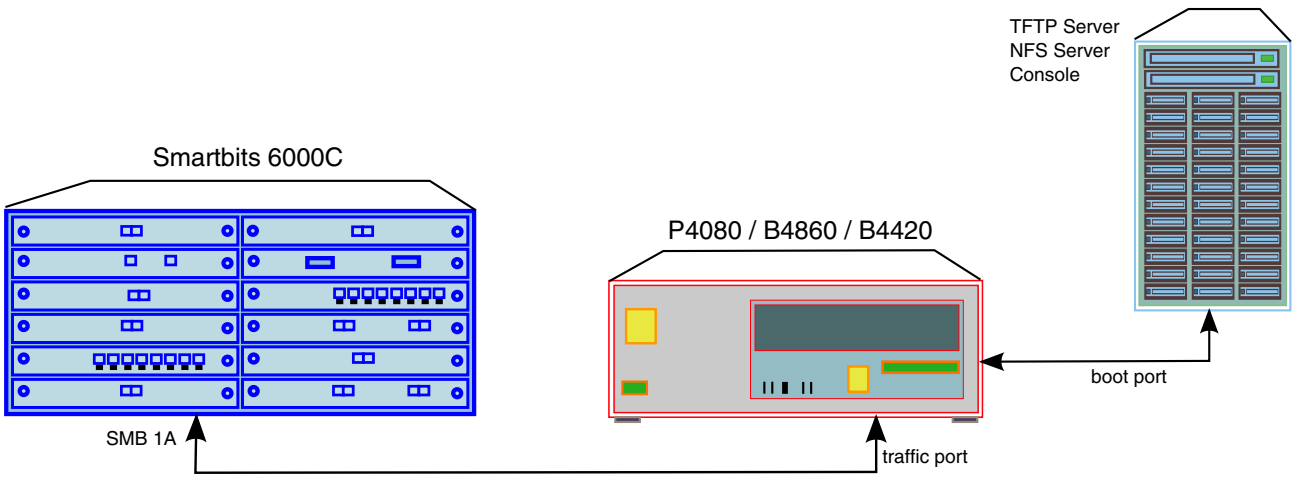

Figure 1. Use Case Set-up

2.2.2 Application start-up and configuration

Use the steps described later in **Compiling the Device Tree for USDPAA Applications** to generate a device tree binary file. Boot the board with the compiled kernel, `arch/powerpc/boot/uImage`, and the `DTB` file.

To reserve memory for USDPAA and enable the scatter-gather support in the DPAA Ethernet driver add the following to the boot arguments:

```
setenv othbootargs "usdpaa_mem=256M fsl_fm_max_frm=9600"
```

The following commands assume that the application runs on a B4860QDS board. When running on a different supported platform, please use the proper configuration file and the parameters listed in the section above. Run these commands to set up the USDPAA network configuration and PCD resources used by the application:

```
export DEF_CFG_PATH=/usr/etc/classifier_demo_config-b4860.xml
export DEF_PCD_PATH=/usr/etc/classifier_demo_policy.xml
export DEF_PDL_PATH=/etc/fmc/config/hxs_pdl_v3.xml
```

First ensure that the following device files have been created:

- `/dev/dpa_classifier`
- `/dev/dpa_stats`

The `classifier_demo` can be run with the following command:

```
/usr/bin/classifier_demo -f 0 -t 5
```

The following arguments specify the frame manager ports used by the application:

- -f [FMan index]
- -t [Traffic Port index]

The output displayed by the application would be something similar with the following:

```
Found /fsl,dpaa/dpa-fman0-oh@2, Tx Channel = 80a, FMAN = 0, Port ID = 1
Found /fsl,dpaa/dpa-fman0-oh@3, Tx Channel = 80b, FMAN = 0, Port ID = 2
Found /fsl,dpaa/dpa-fman0-oh@4, Tx Channel = 80c, FMAN = 0, Port ID = 3
Found /fsl,dpaa/ethernet@2, Tx Channel = 804, FMAN = 0, Port ID = 2
Found /fsl,dpaa/ethernet@5, Tx Channel = 807, FMAN = 0, Port ID = 5
Configuring for 1 network interface
Allocated DMA region size 0x1000000
dpa_classifier_demo: using the following config file:
/usr/etc/classifier_demo_config-b4860.xml
dpa_classifier_demo: using the following PCD file:
/usr/etc/classifier_demo_policy.xml
dpa_classifier_demo: using the following PDL file:
/etc/fmc/config/hxs_pdl_v3.xml
dpa_classifier_demo is assuming FMan:0 and port:5
Header manipulations:
        1) Forwarding 0x0x12e50448, IPSA update 0x0x12e50ad8
        2) Forwarding 0x0x12e4f498, IPSA update 0x0x12e4fe78
        3) Forwarding 0x0x12e50190, IPSA update 0x0x12e50148
        4) Forwarding 0x0x12e501e0, IPSA update 0x0x12e501b8
        5) Forwarding 0x0x12e50080, IPSA update 0x0x12e50068
        6) Forwarding 0x0x12e500f0, IPSA update 0x0x12e500c8
        7) Forwarding 0x0x12e4fc08, IPSA update 0x0x12e50118
        8) Forwarding 0x0x12e4fca0, IPSA update 0x0x12e4fc20
        9) Forwarding 0x0x12e4f9c8, IPSA update 0x0x12e4fcc8
        10) Forwarding 0x0x12e50aa0, IPSA update 0x0x12e50a88
        11) Forwarding 0x0x12e4ff90, IPSA update 0x0x12e503b8
        12) Forwarding 0x0x12e50340, IPSA update 0x0x12e50318
        13) Forwarding 0x0x12e50478, IPSA update 0x0x12e50460
        14) Forwarding 0x0x12e502e8, IPSA update 0x0x12e504b0
        15) Forwarding 0x0x12e50250, IPSA update 0x0x12e50300
        16) Forwarding 0x0x12e50208, IPSA update 0x0x12e4ff28
        17) Forwarding 0x0x12e4fe40, IPSA update 0x0x12e4ffe8
        18) Forwarding 0x0x12e4fe90, IPSA update 0x0x12e4fef8
        19) Forwarding 0x0x12e4fec0, IPSA update 0x0x12e4fea8
        20) Forwarding 0x0x12e50130, IPSA update 0x0x12e4f4b0

Successfully CREATED DPA Classifier Exact Match table (td=0).
DPA Stats library successfully initialized

Successfully Initialized DPA Stats instance: 0
Successfully created DPA Stats counter: 0
Successfully populated DPA Classifier table (20 entries)
Stage #1 is ready - Created & populated table.
When ready to go to the next test stage, type "next_stage"
Released 0 bufs to BPID 7
Released 0 bufs to BPID 8
Released 8192 bufs to BPID 9
Thread uid:0 alive (on cpu 1)
```

2.2.3 Running the application

At this point the user can start sending traffic with the traffic generator.

A DPA Stats class counter is created for the newly DPA Classifier Exact Match Table. The number of class members is configured to match the number of entries the table is populated with. During creation, the class counter is provided only with invalid keys, as such the returned values of the counters statistics is 0 for every member of the class. The class counter provides for every valid key, the following statistics: number of bytes and number of frames.

Stage 1: The application creates a 24 entry Exact Match table managed by key. The table is pre-filled with 4 static entries (marked with the word static in the table below) directly from the XML file. The application also populates at runtime the table with 20 more entries. The classification table content is therefore the following:

Table 4. Stage 1 Classification Table

Flow ID	IP Source	IP Dest.	Protocol (Id)	Dest FQ Id
0	192.168.1.1	192.168.10.10	TCP (6)	6020
1	192.168.1.1	192.168.10.10	UDP (17)	6021
2	192.168.1.1	192.168.10.11	TCP (6)	6022
3	192.168.1.1	192.168.10.11	UDP (17)	6023
4	192.168.1.2	192.168.10.10	TCP (6)	6024
5	192.168.1.2	192.168.10.10	UDP (17)	6025
6	192.168.1.2	192.168.10.11	TCP (6)	6026
7	192.168.1.2	192.168.10.11	UDP (17)	6027
8	192.168.1.3	192.168.10.10	TCP (6)	6028
9	192.168.1.3	192.168.10.10	UDP (17)	6029
10	192.168.1.3	192.168.10.11	TCP (6)	6030
11	192.168.1.3	192.168.10.11	UDP (17)	6031
12	192.168.1.4	192.168.10.10	TCP (6)	6032
13	192.168.1.4	192.168.10.10	UDP (17)	6033
14	192.168.1.4	192.168.10.11	TCP (6)	6034
15	192.168.1.4	192.168.10.11	UDP (17)	6035
16	192.168.1.5	192.168.10.10	TCP (6)	6036
17	192.168.1.5	192.168.10.10	UDP (17)	6037
18	192.168.1.5	192.168.10.11	TCP (6)	6038
19	192.168.1.5	192.168.10.11	UDP (17)	6039
20 (static)	192.168.1.6	192.168.10.10	TCP (6)	6016
21 (static)	192.168.1.6	192.168.10.10	UDP (17)	6017
22 (static)	192.168.1.6	192.168.10.11	TCP (6)	6018
23 (static)	192.168.1.6	192.168.10.11	UDP (17)	6019

The offload engine is performing forwarding (L2) header manipulation followed by L3 header update (IPSA replace) **only on the traffic that hits the classifier entries added at runtime** (flows 5 to 24 in the table above). It replaces the MACSA with the value (0x00:10:63:88:88:xx), MACDA with the value (0x00:20:63:aa:aa:(xx)) where xx is the flow Id. The IP source address is replaced with the value 17.34.51.(yy) (0x11, 0x22, 0x33, 0xyy), where the yy is the flow Id. Additionally, the application is replacing the IPDA of all flows (static flows included) with the value 192.120.(zz.zz), where zzzz is the destination queue id represented on 16 bits. During all test stages, you can check in the application output that these are the MAC addresses and IP SA of the frames received in frame queues associated with these flows (i.e. instead of the original MAC addresses and IP SA of the frames you are generating).

As soon as each key is inserted by the application in the DPA Classifier table, the same key is also added in the class counter by modifying the corresponding member position and updating the key. The pre-filled entries (i.e. entries marked static) are **not** taken into account.

Stage 2: In this stage the application removes table entries 0…9 (first 10 application entries) and also static table entry 20. Also, IPSA header manipulations are updated and are now replacing the IPSA with the value 172.10.10.(xx), where xx is the flow Id.

For each remove of an entry from the range 0…9 from the DPA Classifier Table, the key is also removed from the class counter by modifying the corresponding member position and invalidating the key. The returned counter statistics needs to match the number of sent frames and number of sent bytes.

Stage 3: In this stage the application modifies the lookup keys for table entries 15…19, and for the static entries 22…23. The application modifies the forwarding (L2 replace) header manipulations this time. The offloading engine replaces `MACSA` with the value `00:(xx):12:13:14:15`, where `xx` is the flow Id. The new lookup keys for the modified flows in the classification table are as following:

Table 5. Updated Classification Entries in Stage 3

Flow ID	New IP Source	New IP Dest.	New Protocol (Id)	Dest FQ Id
22 (static)	`192.168.1.18`	`192.168.10.11`	TCP (6)	6018
23 (static)	`192.168.1.18`	`192.168.10.11`	UDP (17)	6019
15	`192.168.33.2`	`192.168.10.10`	TCP (6)	6035
16	`192.168.33.1`	`192.168.10.11`	UDP (17)	6036
17	`192.168.33.1`	`192.168.10.11`	TCP (6)	6037
18	`192.168.33.1`	`192.168.10.10`	UDP (17)	6038
19	`192.168.33.1`	`192.168.10.10`	TCP (6)	6039

For each update of a key for the entries 15…19, the same key is also updated in the class counter by modifying the corresponding member position and setting the new key. The returned counter needs to match the number of sent frames and bytes for flows 10…19.

Stage 4: The application flushes out the Exact Match table and deletes it.

3 Adapting the fragmentation_demo application

The application demonstrates how IP fragmentation can be offloaded to the FMan by using the direct XML configuration and the DPA offloading driver.

3.1 Overview

The IP fragmentation function works strictly on offline ports and performs fragmentation of IPv4 and IPv6 packets into IP fragments according to RFC 791 and RFC 2460. The DPA Stats component usage is highlighted in the same application by creating single and class counters for the IP Fragmentation packet header manipulation.

The application initializes the IP fragmentation and the Parse-Classify-Police-Distribute (PCD) description through the use of the FMC Library.

Traffic is received on a backhaul Rx port (BP) by USDPAA fragmentation_demo application. IP packets that pass the Parse-Classify and Distribute descriptor configured on the Rx port are received in the application. The application swaps the Ethernet MAC addresses and enqueue the traffic in the offline port. This port performs a classification based on VLAN and, depending on the value, it performs IP fragmentation if the frame size is larger than a pre-configured value. Afterwards it forwards the obtained fragments back to the Tx queue of the BP port.

The user has the possibility of performing a number of actions on the DPA Stats counters by typing the following commands in the USDPAA PPAM command line:

Table 6. USDPAA PPAM commands

Command Line	Description
get_stats	retrieve the statistics for the created counters in an asynchronous mode and print the returned values.
get_stats_sync	retrieve the statistics for the created counters in a synchronous mode and print the returned values.
reset_stats	reset the statistics for the created counters.

To have a successful operation, you should verify that the returned values of the counters statistics matches the number of sent frames.

NOTE

The fragmentation demo application is different from the fragmentation performed in the Linux IP stack because the demo does not use the cores or the IP stack for performing fragmentation operation. It uses the frame manager for fragmentation processing.

3.2 Running fragmentation_demo

fragmentation_demo application supports the following platforms:

- P4080
- B4860
- B4420

In order to run it you may need to use a specific device tree file that comes with the DPA offloading driver. Please read below about how to produce this device tree file for your platform.

3.2.1 Application environment specifications

The environment is the same as described for the classifier_demo application. Please see **Classifier_demo: Application environment specifications.**

3.2.2 Application start-up and configuration

Use the steps described later in **Appendix A: Compiling the Device Tree for IP Offloading** to generate a device tree binary file. Boot the board with the compiled kernel, `arch/powerpc/boot/uImage`, and the `DTB` file.

To reserve memory for USDPAA and enable the scatter-gather support in the DPAA Ethernet driver add the following to the boot arguments:

```
setenv othbootargs "usdpaa_mem=256M fsl_fm_max_frm=9600"
```

The following commands assume that the application runs on a B4860QDS board. When running on a different supported platform, please use the proper parameters listed in **Running Classifier_demo: Application environment specifications**. For setting up the USDPAA network configuration and PCD resources used by the application run the following commands:

```
export DEF_CFG_PATH=/usr/etc/fragmentation_demo_config-b4860.xml
export DEF_PCD_PATH=/usr/etc/fragmentation_demo_policy.xml
export DEF_PDL_PATH=/etc/fmc/config/hxs_pdl_v3.xml
```

First ensure that the following device files have been created:

- `/dev/dpa_stats`

The `fragmentation_demo` can be run with the following command:

```
/usr/bin/fragmentation_demo –f 0 –t 5 –o 2
```

The following arguments specify the FMan ports used by the application:

- `-f [FMan index]`
- `-o [Offline Port index]`
- `-t [BP Rx port index]`

The output displayed by the application would be something similar with the following:

```
Found /fsl,dpaa/dpa-fman0-oh@2, Tx Channel = 80a, FMAN = 0, Port ID = 1
Found /fsl,dpaa/ethernet@0, Tx Channel = 802, FMAN = 0, Port ID = 0
Found /fsl,dpaa/ethernet@1, Tx Channel = 803, FMAN = 0, Port ID = 1
Found /fsl,dpaa/ethernet@2, Tx Channel = 804, FMAN = 0, Port ID = 2
Found /fsl,dpaa/ethernet@5, Tx Channel = 807, FMAN = 0, Port ID = 5
Configuring for 2 network interfaces
Allocated DMA region size 0x1000000
Released 0 bufs to BPID 4
fragmentation_demo is assuming FMan:0 and eth:5 and offline port:1
Released 0 bufs to BPID 7
Released 0 bufs to BPID 8
Warn: drained 8192 bufs from BPID 9
Released 8192 bufs to BPID 9
Thread uid:0 alive (on cpu 1)
fragmentation_demo >
```

3.2.3 Running the application

The user should send IPv4 or IPv6 frames generated by an external traffic generator. The following example of IPv4 frames was used to validate the IPv4 Fragmentation:

Table 7. fragmentation_demo IPv4 Input Traffic

MAC SA	VLAN TCI	IP Source	Protocol	Size	IP Fragment Offset
00:00:00:00:00:29	0x2614	192.168.1.1	UDP	1280	0
00:00:00:00:00:2a	0x2615	192.168.1.2	UDP	1280	0
00:00:00:00:00:2b	0x2616	192.168.1.3	UDP	1280	0
00:00:00:00:00:2c	0x2617	192.168.1.4	UDP	1280	0

IPv4 frames that have the VLAN TCI value equal to `0x2614` or `0x2616` are fragmented into fragments of maximum 256 bytes in size, while frames that have the VLAN TCI value equal to `0x2615` or `0x2617` are fragmented into fragments of maximum 512 bytes in size.

IPv6 frames that have the VLAN TCI value equal to `0x6800` or `0x6802` are fragmented into fragments of maximum 256 bytes in size, while frames that have the VLAN TCI value equal to `0x6801` or `0x6803` are fragmented into fragments of maximum 512 bytes in size.

The following example of IPv6 frames was used to validate the IPv6 fragmentation:

Table 8. fragmentation_demo IPv6 input traffic

MAC SA	VLAN TCI	IP Source	Protocol	Size
00:00:00:00:00:2d	0x6800	3FFE:1944:0400:000A:0000:00BC:2500:0D01	UDP	1024
00:00:00:00:00:2e	0x6801	3FFE:1944:0400:000A:0000:00BC:2500:0D02	UDP	1024
00:00:00:00:00:2f	0x6802	3FFE:1944:0400:000A:0000:00BC:2500:0D03	UDP	1024
00:00:00:00:00:30	0x6803	3FFE:1944:0400:000A:0000:00BC:2500:0D04	UDP	1024

After sending the traffic, the user can see the statistics values for the IP fragmentation process. The counter “OH_FRAG1” stands for a single counter of type Fragmentation with MTU of 256 bytes, the counter “OH_FRAG2” stands for single counter of type Fragmentation with MTU of 512 bytes and both objects are also grouped in a class counter of type Fragmentation.

```
STATISTICS:        FRAG_TOTAL_FRAMES  FRAG_FRAMES  FRAG_GEN_FRAGS
OH_FRAG1         :                 0            0               0
OH_FRAG2         :                 0            0               0
CLS_MBR_OH_FRAG1:                  0            0               0
CLS_MBR_OH_FRAG1:                  0            0               0
```

4 Manipulating the reassembly_demo application

This application demonstrates how IP reassembly can be offloaded to the FMan by using the direct XML configuration and the DPA offloading driver.

4.1 Overview

This application demonstrates how IP reassembly can be offloaded to the FMan by using the direct XML configuration and the DPA offloading driver. The IP Reassembly feature can be configured on an RX port or an Offline port and performs reassembly on detected IPv4 and IPv6 protocol fragments. For B4 platforms, the application also provides Virtual Storage Profile support. The DPA Stats component usage is highlighted in the same application by creating single counters for Ethernet and Reassembly, and both single and class counters for Classification Nodes.

The application initializes the Reassembly functionality and the Parse-Classify-Police-Distribute (PCD) description through the use of the FMC Library.

Traffic is received on a backhaul Rx port (BP) by USDPAA reassembly_demo application. IP fragments are detected and reassembled before arriving into the application. The application swaps Ethernet MAC addresses and enqueues the traffic back to the Tx queue of the same port.

The user has the possibility of performing a number of actions on the DPA Stats counters by typing the following commands in the USDPAA PPAM command line:

Table 9. DPA Stats class counter actions

Command Line	Description
get_stats	retrieve the statistics for the created counters in an asynchronous mode and print the returned values.
get_stats_sync	retrieve the statistics for the created counters in a synchronous mode and print the returned values.
reset_stats	reset the statistics for the created counters.

To have a successful operation, the user must verify that the returned values of the counters statistics matches the number of sent frames for each stage of the test.

4.1.1 Virtual storage profiles

The virtual storage profile is only available on frame manager v3. The configuration is transparent to the user. Three storage profiles are defined in the policy file `reassembly_demo_policy-v3.xml`:

```
<vsp name="Default_VSP" base="0"/>
<vsp name="IPv4_Reass_VSP" base="1"/>
<vsp name="IPv6_Reass_VSP" base="2"/>
```

The first storage profile, Default_VSP is also the default storage profile of the port. The second and the third storage profiles are used for IPv4 and IPv6 flows. Each of the storage profiles described above has a set buffer pool associated with it (up to four buffer pools can be defined per vsp)

The buffer pools are dynamically allocated and initialized by reassembly_demo.

When an IPv4 flow is reassembled / classified on inbound port, the IPv4_Reass_VSP is used and the buffers are selected from one of the buffer pools available for this VSP. When an IPv6 flow is reassembled/classified on inbound port, the IPv6_Reass_VSP with buffers from one of the buffer pools available for this VSP is used. For any other flows the Default_VSP is selected.

The VSP selection mechanism described in this paragraph is totally transparent to the user and doesn't affect the use case behavior in any way.

In order to test the non-consistent storage profile functionality the user needs to take into consideration that if a fragment that enters the reassembly distribution is not the first fragment, de default VSP of the port is selected. Otherwise according to the classification, if the fist fragment has the L4 destination port equal to a specific value, the IPv4 VSP is selected. If the reassembled frame has fragments in buffers from different buffer pools, then there is a NCSP event triggered and the frame is enqueued to the NCSP queue. For testing this particular case the user has to take care sending the first fragment of the IPv4 frame (the one with the Fragment Offset flag with value 0) after any other fragment (Ex: fragment3, fragment2, fragment1, fragment4). For the moment the NCSP could be tested only with IPv4 frames.

4.1.2 Differences from Linux IP stack

Reassembly_demo application does not use the cores or the IP stack for performing reassembly operation. It uses the frame manager for processing the reassembly.

4.2 Running reasembly_demo

reasembly_demo application supports the following platforms:

- P4080
- B4860
- B4420

In order to run it you may need to use a specific device tree file that comes with the DPA offloading driver. Please read below about how to produce this device tree file for your platform.

4.2.1 Application environment specifications

The environment is the same as described for the classifier_demo application. Please see **Classifier_demo: Application environment specifications.**

4.2.2 Application start-up and configuration

Use the steps described later in **Appendix A: Compiling the Device Tree for IP Reassembly** to generate a device tree binary file. Boot the board with the compiled kernel, `arch/powerpc/boot/uImage`, and the `DTB` file.

To reserve memory for USDPAA and enable the scatter-gather support in the DPAA Ethernet driver add the following to the boot arguments:

```
setenv othbootargs "usdpaa_mem=256M fsl_fm_max_frm=9600"
```

The following commands assume that the application runs on a B4860QDS board. When running on a different supported platform, please use the proper parameters listed in **Running Classifier_demo: Application environment specifications**. For setting up the USDPAA network configuration and PCD resources used by the application run the following commands:

```
export DEF_CFG_PATH=/usr/etc/reassembly_demo_config-b4860.xml
export DEF_PCD_PATH=/usr/etc/reassembly_demo_policy.xml
export DEF_PDL_PATH=/etc/fmc/config/hxs_pdl_v3.xml
```

First ensure that the following device files have been created:

- `/dev/dpa_stats`

The `reassembly_demo` can be run with the following command:

```
/usr/bin/reassembly_demo -f 0 -t 5
```

The following arguments specify the FMan ports used by the application:

- `-f [FMan index]`
- `-t [Traffic Port index]`

The output displayed by the application would be something similar with the following:

```
Loading configuration
Found /fsl,dpaa/dpa-fman0-oh@2, Tx Channel = 80a, FMAN = 0, Port ID = 1
Found /fsl,dpaa/dpa-fman0-oh@3, Tx Channel = 80b, FMAN = 0, Port ID = 2
Found /fsl,dpaa/dpa-fman0-oh@4, Tx Channel = 80c, FMAN = 0, Port ID = 3
Found /fsl,dpaa/ethernet@0, Tx Channel = 802, FMAN = 0, Port ID = 0
Found /fsl,dpaa/ethernet@1, Tx Channel = 803, FMAN = 0, Port ID = 1
Found /fsl,dpaa/ethernet@2, Tx Channel = 804, FMAN = 0, Port ID = 2
Found /fsl,dpaa/ethernet@5, Tx Channel = 807, FMAN = 0, Port ID = 5
Configuring for 1 network interface
```

```
Allocated DMA region size 0x1000000
Released 4096 bufs to BPID 1
Released 4096 bufs to BPID 2
Released 8192 bufs to BPID 3
reassembly_demo is assuming FMan:0 and port:5
Released 0 bufs to BPID 7
Released 0 bufs to BPID 8
Released 8192 bufs to BPID 9
Thread uid:0 alive (on cpu 1)
reassembly_demo >
```

4.2.3 Running the application

The user should send IPv4 or IPv6 frames generated by an external traffic generator. The following example of IPv4 and IPv6 frames can be used to validate the IP reassembly:

Table 10. reassembly_demo IPv4 & IPv6 Input Traffic

VLAN TCI	IP Source	Protocol	Size (w/ CRC)	ID field	Flags	Fragment Offset
0x0001	192.168.0.1	UDP	298	5	MF	0
0x0001	192.168.0.1	IPv4	298	5	MF	256 (32 x 8)
0x0001	192.168.0.1	IPv4	298	5	MF	512 (64 x 8)
0x0001	192.168.0.1	IPv4	298	5	0	768 (96 x 8)
0x0002	192.168.0.2	UDP	298	5	MF	0
0x0002	192.168.0.2	IPv4	298	5	MF	256 (32 x 8)
0x0002	192.168.0.2	IPv4	298	5	MF	512 (64 x 8)
0x0002	192.168.0.2	IPv4	298	5	0	768 (96 x 8)
0x0003	192.168.0.3	UDP	298	5	MF	0
0x0003	192.168.0.3	IPv4	298	5	MF	256 (32 x 8)
0x0003	192.168.0.3	IPv4	298	5	MF	512 (64 x 8)
0x0003	192.168.0.3	IPv4	298	5	0	768 (96 x 8)
0x0004	192.168.0.4	UDP	298	5	MF	0
0x0004	192.168.0.4	IPv4	298	5	MF	256 (32 x 8)
0x0004	192.168.0.4	IPv4	298	5	MF	512 (64 x 8)
0x0004	192.168.0.4	IPv4	298	5	0	768 (96 x 8)
0x0001	3FFE:1944:0100:000A:0000:00BC:2500:0D0B	UDP	414	5	MF	0
0x0001	3FFE:1944:0100:000A:0000:00BC:2500:0D0B	IPv6	414	5	MF	344
0x0001	3FFE:1944:0100:000A:0000:00BC:2500:0D0B	IPv6	414	5	MF	688
0x0001	3FFE:1944:0100:000A:0000:00BC:2500:0D0B	IPv6	414	5	0	1032
0x0002	3FFE:1944:0200:000A:0000:00BC:2500:0D0B	UDP	414	5	MF	0
0x0002	3FFE:1944:0200:000A:0000:00BC:2500:0D0B	IPv6	414	5	MF	344

Table continues on the next page...

Table 10. reassembly_demo IPv4 & IPv6 Input Traffic (continued)

VLAN TCI	IP Source	Protocol	Size (w/ CRC)	ID field	Flags	Fragment Offset
0x0002	3FFE:1944:0200:000A: 0000:00BC:2500:0D0B	IPv6	414	5	MF	688
0x0002	3FFE:1944:0200:000A: 0000:00BC:2500:0D0B	IPv6	414	5	0	1032
0x0003	3FFE:1944:0300:000A: 0000:00BC:2500:0D0B	UDP	414	5	MF	0
0x0003	3FFE:1944:0300:000A: 0000:00BC:2500:0D0B	IPv6	414	5	MF	344
0x0003	3FFE:1944:0300:000A: 0000:00BC:2500:0D0B	IPv6	414	5	MF	688
0x0003	3FFE:1944:0300:000A: 0000:00BC:2500:0D0B	IPv6	414	5	0	1032
0x0004	3FFE:1944:0400:000A: 0000:00BC:2500:0D0B	UDP	414	5	MF	0
0x0004	3FFE:1944:0400:000A: 0000:00BC:2500:0D0B	IPv6	414	5	MF	344
0x0004	3FFE:1944:0400:000A: 0000:00BC:2500:0D0B	IPv6	414	5	MF	688
0x0004	3FFE:1944:0400:000A: 0000:00BC:2500:0D0B	IPv6	414	5	0	1032

The IP Reassembly feature is considered to be working properly if for the provided example of IP fragments are reassembled in the following frames:

Table 11. Expected Reassembled Frames

IP Source	Protocol	Size	Flags	Fragment Offset
192.168.0.1	IPv4/UDP	1066	0	0
192.168.0.2	IPv4/UDP	1066	0	0
192.168.0.3	IPv4/UDP	1066	0	0
192.168.0.4	IPv4/UDP	1066	0	0
3FFE: 1944:0100:0 00A: 0000:00BC: 2500:0D0B	IPv6/UDP	1438	0	0
3FFE: 1944:0100:0 00A: 0000:00BC: 2500:0D0B	IPv6/UDP	1438	0	0
3FFE: 1944:0100:0 00A: 0000:00BC: 2500:0D0Bf	IPv6/UDP	1438	0	0

Table continues on the next page...

Table 11. Expected Reassembled Frames (continued)

IP Source	Protocol	Size	Flags	Fragment Offset
3FFE: 1944:0100:0 00A: 0000:00BC: 2500:0D0B	IPv6/UDP	1438	0	0

After sending the traffic, the user can validate the statistics values for the IP reassembly process on the Ethernet interface used to send and receive the traffic and the Classification Node:

```
reassembly_demo > get_stats_sync

REASS      : TIMEOUT RFD_POOL_BUSY INT_BUFF_BUSY EXT_BUFF_BUSY SG_FRAGS DMA_SEM NCSP
                   0         0              0              0           0       0     0
REASS_IPV4 : FRAMES FRAGS_VALID FRAGS_TOTAL FRAGS_MALFORMED FRAGS_DISCARDED AUTOLEARN_BUSY
EXCEED_16FRAGS
                   0         0            0           0                0
0                 0
REASS_IPV6 : FRAMES FRAGS_VALID FRAGS_TOTAL FRAGS_MALFORMED FRAGS_DISCARDED AUTOLEARN_BUSY
EXCEED_16FRAGS
                   0         0            0           0                0
0                 0
ETH        : DROP_PKTS  BYTES  PKTS BC_PKTS MC_PKTS CRC_ALIGN_ERR UNDERSIZE_PKTS
OVERSIZE_PKTS
                   0           0    0     0      0           0               0            0
ETH        : FRAGMENTS JABBERS 64BYTE_PKTS 65_127BYTE_PKTS 128_255BYTE_PKTS 256_511BYTE_PKTS
512_1023BYTE_PKTS 1024_1518BYTE_PKTS
                   0         0           0             0                0
0                 0                   0
ETH        : OUT_PKTS OUT_DROP_PKTS OUT_BYTES IN_ERRORS OUT_ERRORS IN_UNICAST_PKTS
OUT_UNICAST_PKTS
                   0          0            0          0           0            0             0
CNT_CLASSIF: IPv6_KEY0 IPv6_KEY1 IPv6_KEY2 IPv6_KEY3 MISS
                   0          0          0          0       0
CLS_CLASSIF: IPv6_KEY0 IPv6_KEY1 IPv6_KEY2 IPv6_KEY3 MISS
                   0          0          0          0       0
CNT_CLASSIF: IPv4_KEY0 IPv4_KEY1 IPv4_KEY2 IPv4_KEY3 MISS
                   0          0          0          0       0
CLS_CLASSIF: IPv4_KEY0 IPv4_KEY1 IPv4_KEY2 IPv4_KEY3 MISS
                   0          0          0          0       0
```

5 Customizing the ipsec_offload application

USDPAA ipsec_offload is a multi-threaded application which applies a subset of IPsec transformations to the IPv4/IPv6 traffic.

5.1 IPSec_offload application overview

The IPv4/IPv6 traffic is received on two 1G ports designated as inbound port, where encrypted/tunneled traffic is expected and outbound port, where clear-text traffic is expected. This task is accomplished using the DPA offload driver which connects and configures DPAA traffic processing accelerators (Fman, SEC) to completely offload IPSec processing based on common Linux configuration tools (setkey, ip xfrm, arp, ip neigh). The following features are currently supported:

- ESP tunnel mode.
- TOS/ECN/DSCP propagation.

5.1.1 IPSec_offload outbound flows

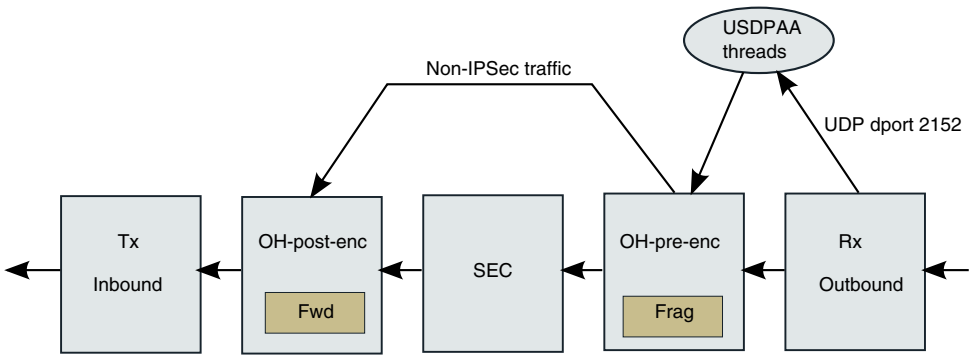

Figure 2. Outbound processing

The clear text traffic is received on the **outbound Rx** and classified based on UDP destination port. Frames matching UDP destination port 2152 are received by USDPAA frame processing threads.The rest of the frames are sent directly to **pre-encryption OH** port. On this port the traffic is classified according to configured IPSec traffic selectors. Frames for which IPsec policy is applicable are sent to SEC for security processing. Optionally, this port performs IP fragmentation for frames matching IPsec policy if frame size is larger than a configured value (`mtu_pre_enc`). Traffic not matching IPsec policy bypass security processing and are enqueued directly to post-encryption OH port. The SEC output frames are enqueued to **post-encryption OH** port. On this port the Ethernet source and destination addresses are updated with Tx port MAC address and next hop MAC address respectively.

5.1.2 IPSec_offload inbound flows

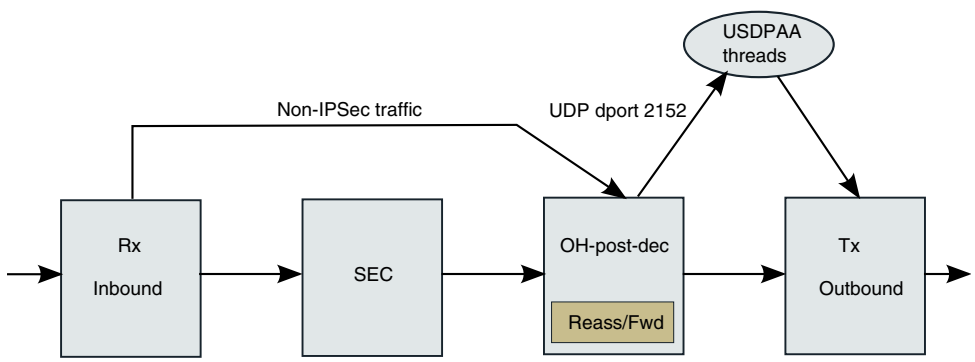

Figure 3. Inbound processing

IPsec traffic (IPv4/IPv6 ESP and UDP-encap ESP) received on the inbound port is classified according to offloaded security associations. Frames for which a security association is found are sent to the appropriate SEC engine input queues for decryption/decapsulation; traffic for which a security association is not found is dropped. Non-IPsec traffic is directly sent to **inbound** OH port. Decrypted and non-IPsec traffic is reassembled if fragmented, optionally passes inbound policy verification and further a post IPsec classification which sends frames matching UDP destination port 2152 to USDPAA frame processing threads. Frames not matching UDP destination port 2152 have their Ethernet source and destination MAC addresses updated with Tx port MAC address and next hop MAC address respectively.

5.1.3 Differences from existing ipsecfwd application

This USDPAA application allows complete offload of IPSec data paths, thus relieving the cores from the tasks of submitting/ receiving frames to/from the SEC engine prior or after tunnel payload processing nd from frame forwarding after IPsec processing. Core intervention is performed only for tasks related to application specific processing. Also, the application supports standard Linux tools for configuration.

5.2 Running the Application

The procedures for configuring the environment, starting up and configuring the Ipsec_offload application are discussed.

5.2.1 Application environment specifications

The application can run in a real network setup configuration and inter-operate with other IPSec gateways. There is also an option to put the inbound port in loopback mode and return IPSec traffic to the inbound port. The details on how to run the application refers to this mode.

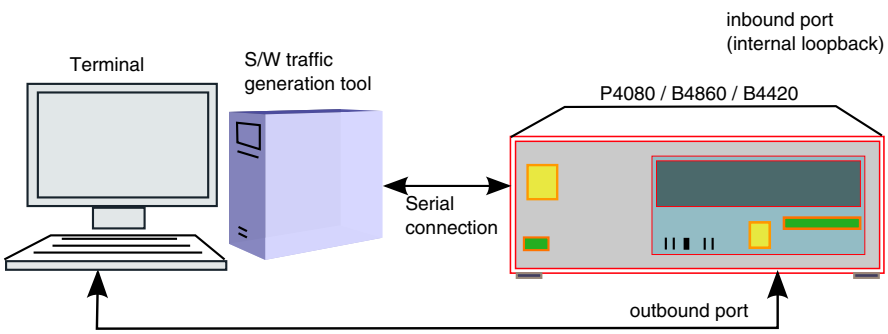

Figure 4. Use case set-up in loopback mode

Table 12. Port connections

SoC	Inbound Port	Outbound Port
P4080	`fm1-gb1`	`fm1-gb0`
B4860	`fm0-mac4`	`fm0-mac5`
B4420	`fm0-mac2`	`fm0-mac3`

5.2.2 Application Start-up and Configuration

5.2.2.1 Application Start-up

Use the steps described in Appendix A **Compiling the Device Tree for IP Offloading** to generate a device tree binary file. Boot the board with the compiled kernel (`arch/powerpc/boot/uImage`) and the DTB file.

To reserve memory for USDPAA and enable the scatter-gather support in the DPAA Ethernet driver add the following to the boot arguments:

```
setenv othbootargs "usdpaa_mem=256M fsl_fm_max_frm=9600"
```

The following commands assume that the application runs on a B4860QDS board. When running on a different supported platform, please use the proper parameters listed above in **Application Environment Specifications**. For setting up the USDPAA network configuration and PCD resources used by the application run the following commands:

```
export DEF_CFG_PATH="/usr/etc/ipsec_offload_config_b4860.xml"
export DEF_PCD_PATH="/usr/etc/ipsec_offload_pcd_b4860.xml"
export DEF_SWP_PATH="/usr/etc/ipsec_offload_swp.xml"
export DEF_PDL_PATH="/etc/fmc/config/hxs_pdl_v3.xml"
```

Start the application with the following commands:

```
/usr/bin/ipsec_offload \
-c /usr/etc/ipsec_offload_config_b4860.xml \
-p /usr/etc/ipsec_offload_policy.xml \
-s /usr/etc/ipsec_offload_swp.xml \
--fm 0 \
--ob_eth 5 --ib_eth 4 \
--ib-oh 1 --ob-oh-pre 2 --ob-oh-post 3 \
--max-sa 32 --vif eth2 --vof eth3 --mtu-pre-enc 500 --ib-loop -n &
```

The following arguments specify the FMan ports used by the application:

- --fm – Fman index
- --ob_eth – outbound Ethernet port index
- --ib_eth – inbound Ethernet port index
- --ib-oh – inbound Offline port index (post decryption)
- --ob-oh-pre – outbound pre encryption Offline port index
- --ob-oh-post – outbound post encryption Offline port index

The following arguments specify two Linux virtual interfaces associated with inbound and outbound Ethernet ports. These interfaces are used by Linux to receive and send local traffic - IP traffic matching Linux IP addresses and non-IP traffic like ARP:

- --vif – virtual inbound interface index
- --vof – virtual outbound interface index

The following argument configures MTU for pre encryption fragmentation. Frames larger than this value will be fragmented prior encryption:

- --mtu-pre-enc – pre encryption MTU

The following argument configures the inbound port in loop mode:

- --ib-loop.

5.2.2.2 Application configuration for IPsec

IPSec can be configured for offloading using setkey tool. To add an SA and two corresponding SPs add the following lines in a setkey.conf file:

```
flush;
spdflush;
add 192.168.0.100 192.168.0.200 esp 0x201
        -E 3des-cbc "abcdefghipqrstuvwxyabcde"
        -A hmac-sha1 "abcdefghipqrstuvwxya";
spdadd 172.16.0.1/32[any] 172.17.0.1/32[any] udp
        -P out ipsec esp/tunnel/192.168.0.100-192.168.0.200/require;
```

```
spdadd 172.17.0.1/32[any] 172.16.0.1/32[any] udp
        -P in ipsec esp/tunnel/192.168.0.100-192.168.0.200/require;
```

Now run the command:

```
setkey –f setkey.conf
```

These commands create an ESP tunnel with the following endpoints: IP src 192.168.0.100 – IP dst 192.168.0.200, SPI 0x201, 3DES-CBC encryption and HMAC-SHA1 authentication. The UDP traffic coming from 172.16.0.1 going to 172.17.0.1 will be tunneled using the defined tunnel.

To configure Linux interfaces, neighboring and routing for the loopback setup run the following commands:

```
#enable IP forwarding
echo "1" > /proc/sys/net/ipv4/ip_forward
#inbound interface
ifconfig eth2 192.168.100.1 netmask 255.255.255.0 up
echo "1" > /proc/sys/net/ipv4/conf/eth2/disable_policy
echo "1" > /proc/sys/net/ipv4/conf/eth2/disable_xfrm
echo "1" > /proc/sys/net/ipv4/conf/eth2/accept_local
#outbound interface
ifconfig eth3 172.16.0.254 netmask 255.255.0.0 up
echo "1" > /proc/sys/net/ipv4/conf/eth3/disable_policy
echo "1" > /proc/sys/net/ipv4/conf/eth3/disable_xfrm
echo "1" > /proc/sys/net/ipv4/conf/eth3/accept_local
#route to tunnel destination
route add -net 192.168.200.0/24 gw 192.168.100.254 dev eth2
#route to tunneled traffic destination - outbound interface in loop mode
route add -net 172.17.0.0/16 gw 172.16.0.1 dev eth3
#static neigh entry for loopback mode
ip neigh add 192.168.100.254 lladdr <inbound_port_mac_addres> dev eth2
```

5.2.3 Running Traffic

Assuming that the traffic is generated with a directly connected Linux box, you need to configure the Linux box to output frames for `172.16.0.1` on the connected interface (ethX):

```
ifconfig <ethX> 172.16.0.2 netmask 255.255.0.0 up
```

You can use the hping tool to generate UDP packets and tcpdump to capture traffic on interface ethX. Each sent frame should be returned by the application.

```
hping -2 –s 2152 –p 2152 –c 1 172.17.0.1
```

5.3 ipsec_offload application flow

The application follows PPAC/PPAM design. PPAM is responsible for initialization and for forwarding received frames (UDP destination port 2152) between outbound port and outbound/inbound offline ports. The following tasks are part of the initialization:

- Get command line configuration parameters
- Allocate memory and create buffer pools for additional required buffer pools (i.e – IP fragmentation, IP reassembly)
- Compile and apply PCD model for the application

- Init IPsec offloading component
- Start XFRM and neighbouring notification processing
- Mappings required by USDPAA to forward frames between ports

Offloading work is performed by XFRM and neighbour notifications processing threads. Each thread listens on a specific Netlink socket and receives messages containing the event and associated information. This information is used to configure IPsec offloading datapath.

6 References

1. [P4080/B4860/B4420] Integrated Multicore Communication Processor Family Reference Manual (pdf)
2. USDPAA PPAC User Guide (infocenter)
3. QMan/BMan API Reference (infocenter)

7 Appendix A – Preparing DPA Offloading DTB Files

Detail reference for compiling the device tree for these three applications:

1. USDPAA
2. IP Offloading
3. IP Reassembly

For these three SoC products:

1. P4080
2. P4860
3. P4420

7.1 Compiling the device tree for USDPAA applications

7.1.1 Compiling the device tree for P4080

Go to your Linux kernel source code directory and run the following commands:

```
cp drivers/staging/fsl_dpa_offload/dts/p4080-usdpaa.dts
   arch/powerpc/boot/dts

scripts/dtc/dtc -f -b 0 -p 1024 -I dts -O dtb -o p4080ds-usdpaa.dtb
   arch/powerpc/boot/dts/p4080ds-usdpaa.dts
```

The DTB file `p4080ds-usdpaa.dtb` will be built in the current directory.

7.1.2 Compiling the device tree for B4860

Go to your Linux kernel source code directory and run the following commands:

```
cp drivers/staging/fsl_dpa_offload/usecases/dts/b4860qds-usdpaa.dts
   arch/powerpc/boot/dts

scripts/dtc/dtc –f –b 0 –p 1024 –I dts –O dtb –o b4860qds-usdpaa.dtb
   arch/powerpc/boot/dts/b4860qds-usdpaa.dts
```

The DTB file `b4860qds-usdpaa.dtb` will be built in the current directory.

7.1.3 Compiling the device tree for B4420

Go to your Linux kernel source code directory and run the following commands:

```
cp drivers/staging/fsl_dpa_offload/usecases/dts/b4420qds-usdpaa.dts
   arch/powerpc/boot/dts

scripts/dtc/dtc –f –b 0 –p 1024 –I dts –O dtb –o b4420qds-usdpaa.dtb
   arch/powerpc/boot/dts/b4420qds-usdpaa.dts
```

The DTB file `b4420qds-usdpaa.dtb` will be built in the current directory.

7.2 Compiling the device tree for IP offloading

7.2.1 Compiling the device tree for P4080

Go to your Linux kernel source code directory and run the following commands:

```
cp drivers/staging/fsl_dpa_offload/dts/p4080ds-usdpaa.dts
arch/powerpc/boot/dts/

cp drivers/staging/fsl_dpa_offload/dts/p4080si-post.dtsi
arch/powerpc/boot/dts/fsl

cp drivers/staging/fsl_dpa_offload/dts/p4080si-pre.dtsi
arch/powerpc/boot/dts/fsl

cp drivers/staging/fsl_dpa_offload/dts/p4080si-chosen-offld.dtsi
arch/powerpc/boot/dts/fsl/p4080ds-chosen.dtsi

scripts/dtc/dtc –f –b 0 –p 1024 –I dts –O dtb –o p4080ds-usdpaa.dtb
arch/powerpc/boot/dts/p4080ds-usdpaa.dts
```

The DTB file `p4080ds-usdpaa.dtb` will be built in the current directory.

7.2.2 Compiling the device tree for B4860

Go to your Linux kernel source code directory and run the following commands:

```
cp drivers/staging/fsl_dpa_offload/dts/b4860qds-usdpaa.dts
 arch/powerpc/boot/dts

cp drivers/staging/fsl_dpa_offload/dts/b4860si-pre.dtsi
 arch/powerpc/boot/dts/fsl

cp drivers/staging/fsl_dpa_offload/dts/b4860si-chosen-offld.dtsi
```

```
 arch/powerpc/boot/dts/fsl/b4860si-chosen.dtsi

scripts/dtc/dtc -f -b 0 -p 1024 -I dts -O dtb -o
 b4860qds-usdpaa.dtb
arch/powerpc/boot/dts/b4860qds-usdpaa.dts
```

The DTB file `b4860qds-usdpaa.dtb` will be built in the current directory.

7.2.3 Compiling the device tree for B4420

Go to your Linux kernel source code directory and run the following commands:

```
cp drivers/staging/fsl_dpa_offload/dts/b4420qds-usdpaa.dts
 arch/powerpc/boot/dts

cp drivers/staging/fsl_dpa_offload/dts/b4420si-pre.dtsi
 arch/powerpc/boot/dts/fsl

cp drivers/staging/fsl_dpa_offload/dts/b4420si-chosen-offld.dtsi
 arch/powerpc/boot/dts/fsl/b4420si-chosen.dtsi

scripts/dtc/dtc -f -b 0 -p 1024 -I dts -O dtb -o
 b4420qds-usdpaa.dtb
arch/powerpc/boot/dts/b4420qds-usdpaa.dts
```

The DTB file `b4420qds-usdpaa.dtb` will be built in the current directory.

7.3 Compiling the device tree for IP reassembly

7.3.1 Compiling the device tree for P4080

On P4080 there is no specific device tree building process for IP reassembly. You can use the one described in **Compiling the Device Tree for USDPAA Applications**.

7.3.2 Compiling the device tree for B4860

Go to your Linux kernel source code directory.

```
cp drivers/staging/fsl_dpa_offload/dts/b4860si-pre.dtsi
 arch/powerpc/boot/dts/fsl

cp drivers/staging/fsl_dpa_offload/dts/b4860si-chosen-reass.dtsi
 arch/powerpc/boot/dts/fsl/b4860si-chosen.dtsi

scripts/dtc/dtc -f -b 0 -p 1024 -I dts -O dtb -o b4860qds-usdpaa.dtb
arch/powerpc/boot/dts/b4860qds-usdpaa.dts
```

The DTB file `b4860qds-usdpaa.dtb` will be built in the current directory.

7.3.3 Compiling the device tree for B4420

Go to your Linux kernel source code directory.

```
cp drivers/staging/fsl_dpa_offload/dts/b4420si-pre.dtsi
 arch/powerpc/boot/dts/fsl

cp drivers/staging/fsl_dpa_offload/dts/b4420si-chosen-reass.dtsi
 arch/powerpc/boot/dts/fsl/b4420si-chosen.dtsi

scripts/dtc/dtc –f –b 0 –p 1024 –I dts –O dtb –o b4420qds-usdpaa.dtb
arch/powerpc/boot/dts/b4420qds-usdpaa.dts
```

The DTB file `b4420qds-usdpaa.dtb` will be built in the current directory.

8 Revision history

This table summarizes revisions to this document.

Table 13. Revision history

Revision	Date	Description
0	08/2013	Initial public release.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4785
Revision 0, 08/2013