Separating Control Plane and Data Plane between GPP and AIOP on LS2085

This document introduces a method to separate control plane and data plane between GPP(ARM) and AIOP based on different L4 protocols implemented in the AIOP software. So far, in the current MC version, this scenario could not be implemented from WRIOP using DPDMUX, so it is a good choice for users to separate the traffic in AIOP.

1. Basic Concept of DPAA2 Objects

Management Complex(MC): GPPs discovering and using higher level, and familiar, resources called objects. These objects are created by MC firmware from low-level DPAA2 resources, such as FQs and buffer pools.

Wire Rate I/O Processor(WRIOP): In DPAA1, network interfaces are implemented within the Frame Manager. In DPAA2, these interfaces are implemented within WRIOP. The WRIOP combines the Ethernet MACs with packet parsing and classification logic to provide intelligent distribution and queuing decisions for incoming traffic.

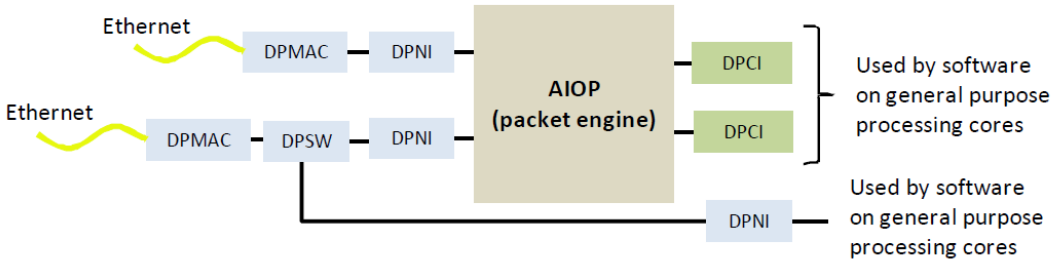
Advanced I/O Processor(AIOP): While the DPAA1 Frame Manager offers a level of I/O intelligence using microcode packages, the AIOP is a C-programmable packet processing engine. It is comprised of an array of small processors with hardware scheduling, and contains hardware acceleration for parsing, look-ups, and data movement.

DPNI: The DPNI object is the center of network interfaces. On ingress, it receives frames from a DPMAC or another object such as a DPSW, parses headers, determines the frame's traffic class, and enqueues the frame onto a frame queue selected based on the traffic class and other header values.

DPMAC: The DPMAC object represents an Ethernet MAC, a hardware device that connects to a PHY and allows physical transmission and reception of Ethernet frames.

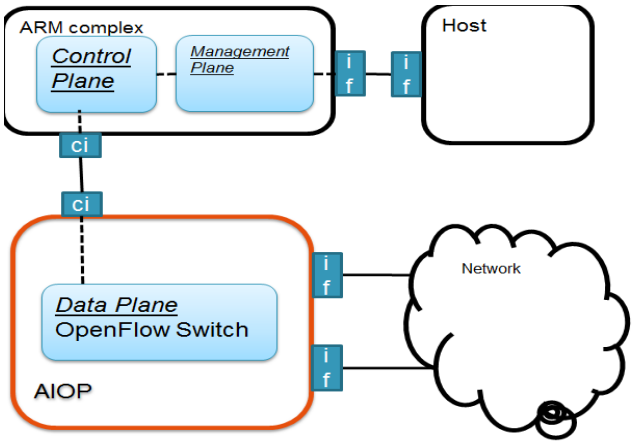
DPRC: A Datapath Resource Contrainer(DPRC) is a special object, used by the MC as a container to store all the resource and object information usable by the software partition the DPRC is assigned to.

The following figure describes AIOP interfaces structure by DPAA2 objects.



2. AIOP Application to Implement Control in ARM and Data Plane in AIOP

AIOP is built to make easy and efficient implementation of data plane, the following figure describes the whole structure to implement control panel in GPP(ARM) and data plane in AIOP.



This AIOP application demonstrate AIOP-GPP communication using a DPNI-DPNI connection. The application filters ICMP and ARP request.

This application uses two kinds of connections:

DPNI to DPNI connection: a network interface from AIOP that provides connection to another network interface on GPP.

DPNI to DPMAC connection: a network interface from AIOP connected to external traffic(physical wired connection).

The following figure describes the work flow of this application and AIOP APIs call sequence.

If the endpoint is DPNI, the call function app_dpni_rx_gpp will be invoked, it is the DPNI-DPNI connection with GPP, it will be the only interface that communicates with GPP.

If the endpoint is DPMAC, it will be the only interface that receives frames from the external network.

dpni_drv_get_dpni_id: Get DPNI ID for NI.

dpni_drv_get_connected_obj: Get the connected OBJ ID and type.

dpni_drv_register_rx_cb: Attaches a pointer to a call back function to a NI ID.

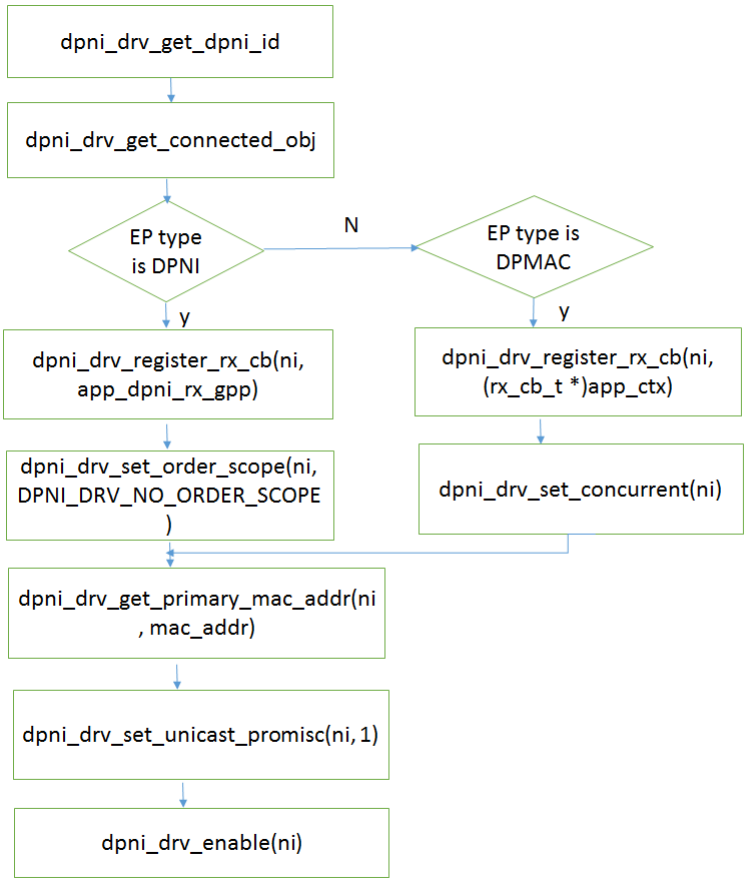
dpni_drv_set_order_scope: Set order scope source for the specified NI. GPP hasn't activated order preservation and order restoration, so set initial order scope to "No order scope".

dpni_drv_set_concurrent: set the initial ordering mode to concurrent for the given NI.

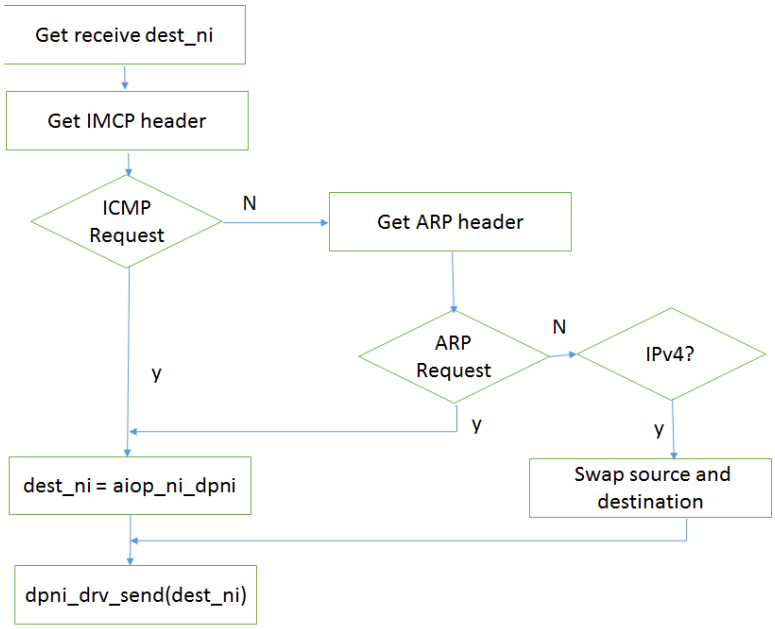
dpni_drv_get_primary_mac_addr: Get Primary MAC address of NI.

dpni_drv_set_unicast_promisc: Enable unicast promiscuous mode.

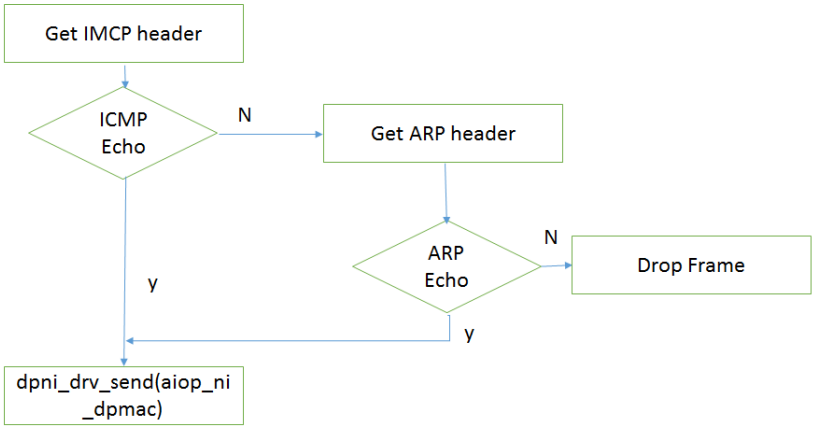
dpni_drv_enable: Enable NI to receive frames.



The work flow of app_dpni_rx_dpmac function is described as the following. This function performs frames processing received from DPMAC. Get IMCP and ARP headers, if this frame is ICMP or ARP request, send this packet to GPP DPNI. If this frame is a formal IPv4 packet, swapping the source and destination, then send it out.



The follow figure describes work flow of function app_dpni_rx_gpp, this function handles frames received from GPP via DPNI-DPNI connection. ARP reponse or ICMP echo reply, that come from GPP are sent back to the external network interface.



Define buffer pools allocation in the application in file apps.h as the following. Two AIOP buffer pools are reserved for frame data buffers of all AIOP DPNIs. One for frame data residing in DP-DDR and one for frame data residing in PEB. The buffer amount, size and alignment defined below applies to both pools and to all AIOP DPNIs. Each DPNI uses two Storage Profile IDs (one for DP-DDR and one for PEB) .

```
#define APP_DPNI_NUM_BUFS_IN_POOL      512    /**< Number of buffers */
#define APP_DPNI_BUF_SIZE_IN_POOL      2048   /**< Size of buffer */
#define APP_DPNI_BUF_ALIGN_IN_POOL     64     /**< Alignment of buffer */
#define APP_DPNI_SPID_COUNT            8      /**< Max number of SPIDs */
```

The script dynamic_aiop_root.sh invokes restool command to perform the following operations:

1. Create an AIOP DPRC

AIOP DPRC contains 3 DPBPs, 2 DPNIs

2. Create an AIOP Tool DPRC

AIOP Tool DPRC contains 1 DPAIOP and 1 DPMCP

DPNIs of AIOP DPRC will be connected according to:

1 DPNI <-----> DPMAC.1

1 DPNI <-----> 1 DPNI from Root Container, available in Linux

The DPNI-DPMAC connection is defined as the following:

log_info "Connecting $atc_DPNI1<------->$DPMAC1"

restool_cmd "dprc connect dprc.1 --endpoint1=$atc_DPNI1 --endpoint2=$DPMAC1" None None

The DPNI-DPNI connection is defined as the following:

log_info "Connecting $atc_DPNI2<------->$ROOTDPNI"

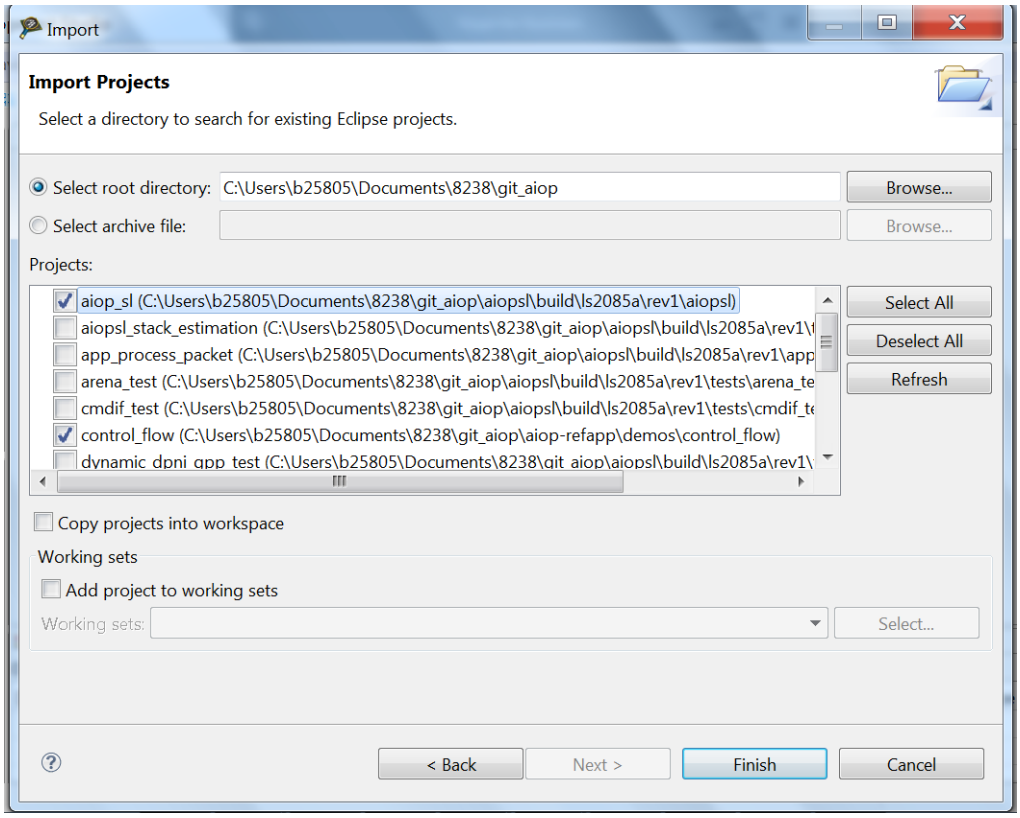
restool_cmd "dprc disconnect dprc.1 --endpoint=$ROOTDPNI" None None

restool_cmd "dprc connect dprc.1 --endpoint1=$atc_DPNI2 --endpoint2=$ROOTDPNI" None

3. Build AIOP Application Project with CodeWarrior

AIOP integrates e200 cores, it is required to build AIOP application images with CW4NET CodeWarrior for APP.

Import AIOP projects into CodeWarrior from File->Import->General->Existing Projects into Workspace, then build the project to generate ELF file from Project->Build Project.



4. Running AIOP Application Program on LS2085ARDB

Deploy images on the alternate bank and boot Kernel ITB image.

=> tftp 0x80000000 u-boot-ls2085ardb.bin; erase 0x584100000 +$filesize; cp.b 0x80000000 0x584100000 $filesize

=> tftp 0x80000000 PBL_0x2a_0x41_1800_600_1867_1600.bin; erase 0x584000000 +$filesize; cp.b 0x80000000 0x584000000 $filesize

=> tftp 0x80000000 mc.itb; erase 0x584300000 +$filesize; cp.b 0x80000000 0x584300000 $filesize

=> tftp 0x80000000 dpc-0x2a41.dtb; erase 0x584800000 +$filesize; cp.b 0x80000000 0x584800000 $filesize

=> tftp 0x80000000 dpl-eth.0x2A_0x41.dtb; erase 0x584700000 +$filesize; cp.b 0x80000000 0x584700000 $filesize

=>setenv mcinitcmd "fsl_mc start mc 0x580300000 0x580800000"

=> fsl_mc apply dpl 0x580700000

=> tftp 0xa0000000 kernel-ls2085ardb.itb

=> bootm 0xa0000000

Running the script dynamic_aiop_root.sh to allocate resources for the AIOP allocation.

root@ls2085ardb:/# ./usr/aiop/scripts/dynamic_aiop_root.sh

Creating AIOP Container

Assigned dpbp.1 to dprc.2

Assigned dpbp.2 to dprc.2

Assigned dpbp.3 to dprc.2

Assigned dpni.1 to dprc.2

Connecting dpni.1<------->dpmac.1

Assigned dpni.2 to dprc.2

Connecting dpni.2<------->dpni.0

AIOP Container dprc.2 created

----- Contents of AIOP Container: dprc.2 -----

dprc.2 contains 5 objects:

object label plugged-state

dpni.2 plugged

dpni.1 plugged

dpbp.3 plugged

dpbp.2 plugged

dpbp.1 plugged

==========

Creating AIOP Tool Container

Assigned dpaiop.0 to dprc.3

Assigned dpmcp.22 to dprc.3

AIOP Tool Container dprc.3 created

----- Contents of AIOP Tool Container: dprc.3 -----

dprc.3 contains 2 objects:

object label plugged-state

dpaiop.0 plugged

dpmcp.22 plugged

==========

Performing VFIO mapping for AIOP Tool Container (dprc.3)

Performing vfio [554.995826] vfio-fsl-mc dprc.3: Binding with vfio-fsl_mc driver

mapping for dprc.3

[555.004990] vfio-fsl-mc dpaiop.0: Binding with vfio-fsl_mc driver

[555.011592] vfio-fsl-mc dpmcp.22: Binding with vfio-fsl_mc driver

========== Summary ==========

AIOP Container: dprc.2

AIOP Tool Container: dprc.3

==========

root@ls2085ardb:/#

Running aiop_tool to load AIOP application from GPP.

root@ls2085ardb:/# aiop_tool load -g dprc.3 -f /usr/aiop/bin/aiop_control_flow.elf

AIOP Image (/usr/aiop/bin/aiop_control_flow.elf) loaded successfully.

root@ls2085ardb:/#

Configure the networking interfaces:

$ ip link set dev ni0 down

$ ip addr flush dev ni0

$ ip addr add 6.6.6.1/8 dev ni0

$ ip link set dev ni0 up

$ arp -s 6.6.6.10 <MAC_address_of_traffic_generator>

$ arp -n

Address HWtype HWaddress Flags Mask Iface

6.6.6.10 ether <MAC_address_of_traffic_generator> CM ni4

The following message will be printed for ICMP Frames.

RX on NI 0

MAC_SA: 00-00-00-00-00-02 MAC_DA: 02-00-c0-a8-48-01
IP_SRC: 6.6.6.10 IP_DST: 6.6.6.1

ICMP_TYPE: 8 ICMP_CODE: 0
RX on NI 0
MAC_SA: 00-00-00-00-00-02 MAC_DA: 02-00-c0-a8-48-01
ARP_OPCODE: 1 S_ADDR: 6.6.6.10 T_ADDR: 6.6.6.1
RX on NI 0
MAC_SA: 00-00-00-00-00-06 MAC_DA: 00-10-94-00-00-10
IP_SRC: 192.0.0.1 IP_DST: 192.85.1.2