STD-EA9-ETH-001

ExynosAuto 9 Ethernet

Ethernet IP concepts

Version 1.1

Technical Document

SAMSUNG ELECTRONICS

SAMSUNG

Important Notice

Samsung Electronics Co. Ltd. (“Samsung”) reserves the right to make changes to the information in this publication at any time without prior notice. All information provided is for reference purpose only. Samsung assumes no responsibility for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

This publication on its own does not convey any license, either express or implied, relating to any Samsung and/or third-party products, under the intellectual property rights of Samsung and/or any third parties.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

Customers are responsible for their own products and applications. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could reasonably be expected to create a situation where personal injury or death may occur. Customers acknowledge and agree that they are solely responsible to meet all other legal and regulatory requirements regarding their applications using Samsung products notwithstanding any information provided in this publication. Customer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim (including but not limited to personal injury or death) that may be associated with such unintended, unauthorized and/or illegal use.

Samsung Electronics Co., Ltd.
San #24 Nongseo-Dong, Giheung-Gu
Yongin-City, Gyeonggi-Do, Korea 446-711

Contact Us: *mobilesol.cs@samsung.com*

Home Page: *http://www.samsungsemi.com*

Revision History

Revision No.	Date	Description	Author(s)
0.1	2018.10.25	• Initial draft	Ilho Lee Jung-Sik Lee SHINSangki Wonsang Ryou
0.11	2018.10.29	• Change 3 words • SYS -> ServerOS • IVI -> GuestOS0 • AND -> GuestOS1	Jung-Sik Lee
1.0	2019.03	• 3 Rx Packet Flow Concept • 4 Audio Video Bridging • APPENDIX 3.3 Qdisc and Tx Queuing	Jung-Sik Lee Bumyong Lee
1.1	2019.04	• MAC-DA Filter slots: 32 -> 128 • AVB Node Type Table is updated	Jung-Sik Lee Bumyong Lee

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronyms	Descriptions
SFR	Special Function Register
SPI	Serial Peripheral Interface
GIC	Generic Interrupt Controller
MMU	Memory Management Unit
S2MPU	Stage 2 Memory Protection Unit
VM	Virtual Machine
VGEN	Virtual Machine Information Generator
VID	VGEN ID
VMID	Virtual Machine ID of ARM
DSP	Digital Signal Processor
SFI	Safety Island
ABOX	Audio BOX
CH	Channel
PCP	Priority Code Point in VLAN Tag
Qdisc	Queuing discipline
QoS	Quality of Service
AVB	Audio Video Bridging
AVTP	Audio Video Transport Protocol
gPTP	generalized Precision Time Protocol
SRP	Stream Reservation Protocol
FQTSS	Forwarding and Queueing for Time-Sensitive Streams
CBS	Credit-Based Shaper

1 Purpose of the Document

1.1 Purpose

This Document explains isolation/protection/Flow of DMA Channels in Ethernet Virtualization, in Samsung ExynosAuto V9. (This is supported from EVT1.)

- Interrupts: According to DMA CH assignment, SW can mask / unmask SPIs for each core or VM. It will be managed by GIC.

- SFR isolation: S2MPU or ARM MMU will isolate SFRs according to configuration.

- Memory protection: S2MPU will protect Queues for RX/TX of each DMA CH.

- Rx Flow Concept: You can map between Rx Queues and Rx DMA CHs.

2 DMA CH ISOLATION/PROTECTION

2.1 Overview of DMA CH protection

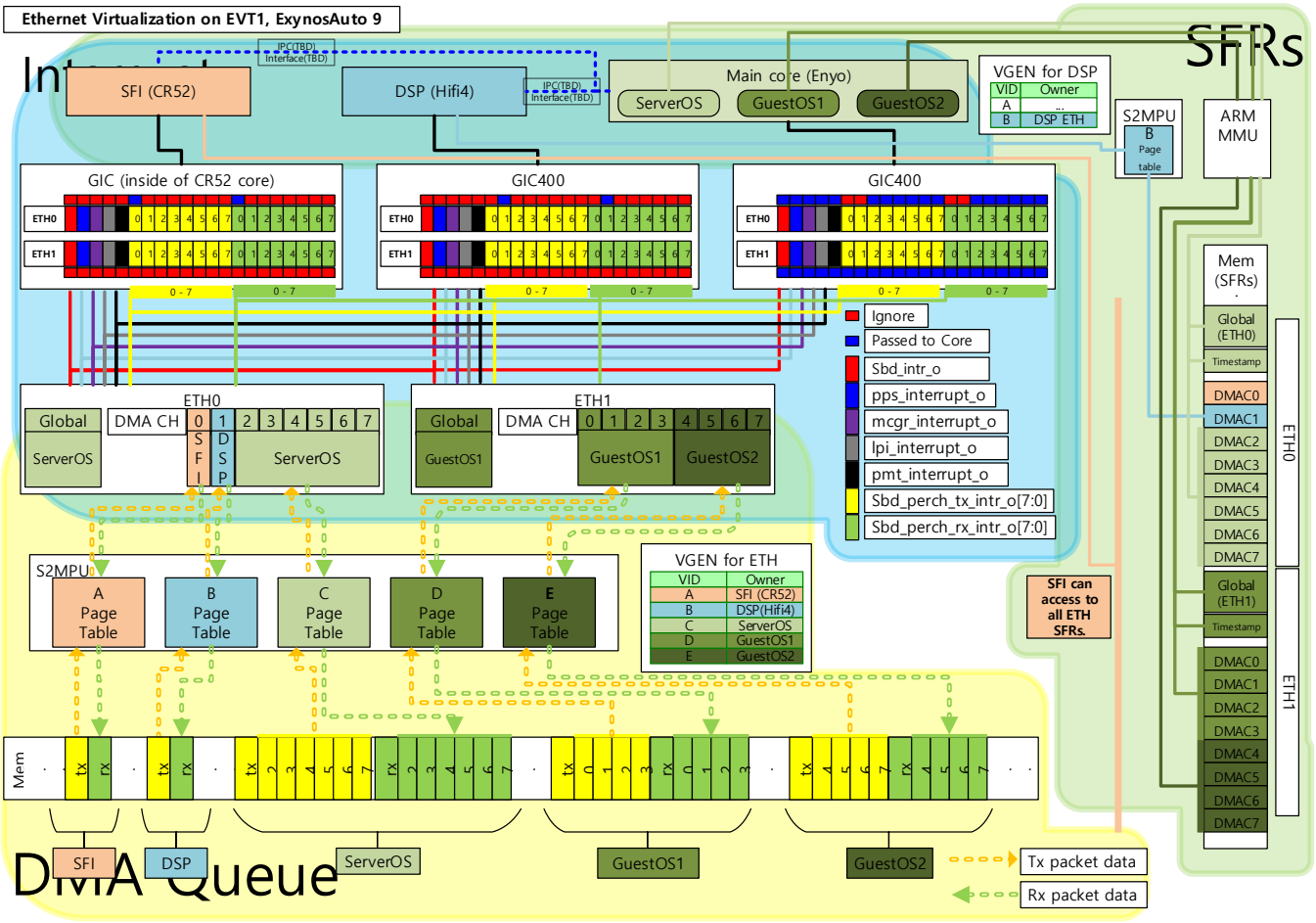


Figure 1: DMA CH Protection Concept Diagram

This diagram is an example. So your scenario need to be reviewed with Samsung to configure DMA CHs.

SW can configure masking / unmasking Ethernet interrupts to each GIC or GIC400 of each core.

RX/TX Queues, which will be allocated in memory for SFI, DSP, and VM (ServerOS, GuestOS1, GuestOS2) of Main core (Enyo), are protected by S2MPU.

Table 1: VID for RX/TX Queues

	SFI (VID:A)	DSP (VID:B)	ServerOS (VID:C)	GuestOS1 (VID:D)	GuestOS2 (VID:E)
Protection unit	N/A	Ethernet S2MPU			

In case of SFRs (Special Function Registers) of Ethernet CHs (Channels) dedicated to DSP, VM, and SFI

- SFRs of Ethernet CH assigned to DSP: These are protected by ARM MMU from a VM's possible malicious access to them.

- SFRs of Ethernet CH assigned to main cores for a VM: These are protected by S2MPU from DSP's possible malicious access to them.

- SFRs of Ethernet CH assigned to SFI: These are protected by DSP's S2MPU and ARM MMU for VMs.

Table 2: VID/VMID for SFRs

	SFI (VID:A)	DSP (VID:B)	ServerOS (VMID:0)	GuestOS1 (VMID:1)	GuestOS2 (VMID:2)
Protection unit	N/A	DSP S2MPU	ARM MMU		

2.2 Configuration of Interrupt in GIC

2.2.1 Mask/Unmask in GIC

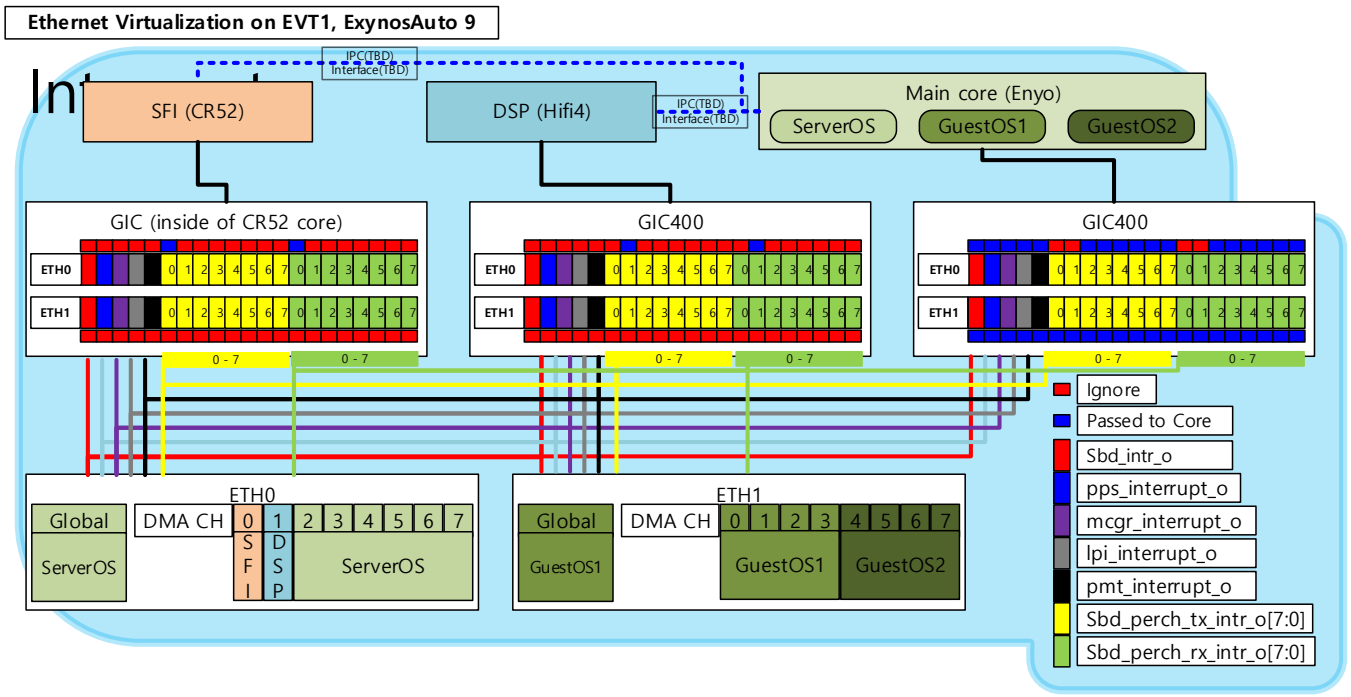


Figure 2: Interrupt Isolation Concept Diagram

According to customer's DMA CH assignment request, SW can mask/unmask each interrupt (SPI, Shared Peripheral Interrupt) of GICs for SFI, DSP, and VMs of Main core(Enyo, CA-76)SW can mask/unmask each interrupt for each VM (SPI, Shared Peripheral Interrupt) of GIC400.

- Mask: a SPI is ignored. DSP/ARM cores can't receive interrupt signals.
- Unmask: a SPI is passed to Cores. The cores will handle the signals.

Table 3: GIC configurations

	First configuration (Mask/Unmask SPI)	Change Mask/Unmask SPIs in runtime (This does not mean EHT DMA CH reassignment. This is Just mask/unmask of SPIs in GIC.)
SFI's GIC	During OS initiation	Possible to mask/unmask
DSP's GIC400	After loading HiFi4 DSP or ABOX GIC400 device driver probe	Possible to mask/unmask

Main core(Enyo)'s GIC400	By default, all of SPIs are disabled.	Possible to mask/unmask

2.3 Ethernet SFR (Special Function Register) Isolation

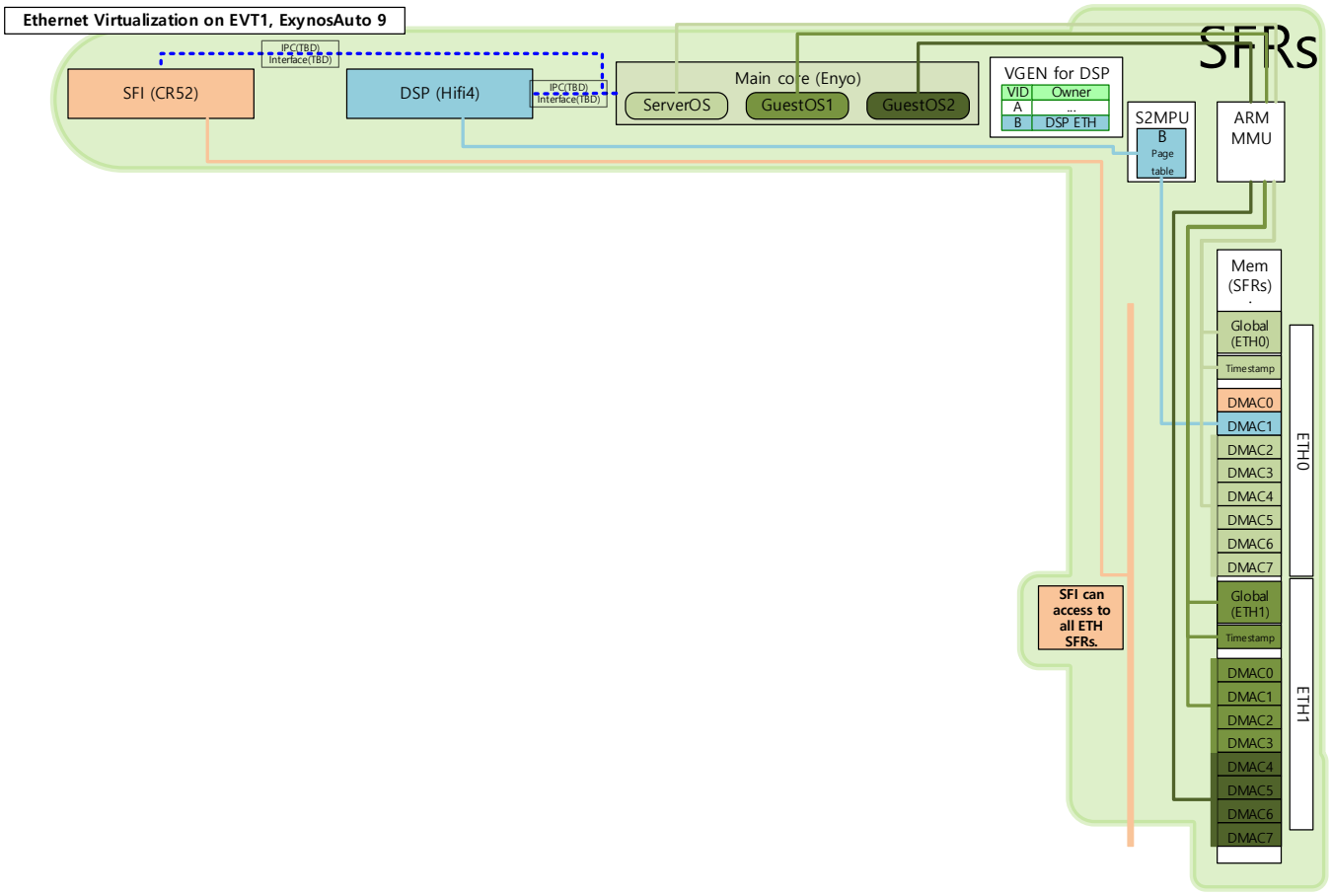


Figure 3: SFR Isolation Concept Diagram

2.3.1 SFRs of Ethernet CH assigned to DSP

This section explains how SFRs of Ethernet CH assigned to DSP are protected by ARM MMU from a possible malicious access of a VM to them.

- Each VM is identified by a VMID (Virtual Machine ID). The VMID of ServerOS is '0', and the VMID of GuestOS1 is '1'. A hypervisor sets up and maintains a 2nd-stage page table per VMID, which is fed into ARM MMU. There will be 3 2nd-stage page tables here – ServerOS, GuestOS1, and GuestOS2.

- The page table represents SFRs accessible to a VM. That is, this table will include information about SFRs of Ethernet CH assigned to the VM. When a VM would try to access SFRs of Ethernet CH allocated to DSP, a MMU translation abort takes place, because there is no information about SFRs of Ethernet CH allocated to DSP in the table.

2.3.2 SFRs of Ethernet CH assigned to a VM

This section explains how SFRs of Ethernet CH assigned to a VM are protected by a S2MPU from a possible malicious access of DSP.

- The Ethernet DMA CH dedicated to DSP is identified by a VID (VGEN ID) like VMID. This VID is unique in Ethernet S2MPU. Also a hypervisor sets up and maintains a page table for this VID fed into S2MPU. The page table contains information of memory accessible to DSP core. When the DSP would try to access SFRs of Ethernet CH allocated to VMs, a S2MPU protection abort takes place, because the table wouldn't have any information about SFRs of Ethernet CH allocated to those VMs.

2.3.3 SFRs of Ethernet CH assigned to SFI

These are protected by the S2MPU and ARM MMU above.

2.3.4 Exception

SFI can access all SFRs of 2 Ethernet IPs due to Safety traffic.

2.4 Rx/Tx Queue protection

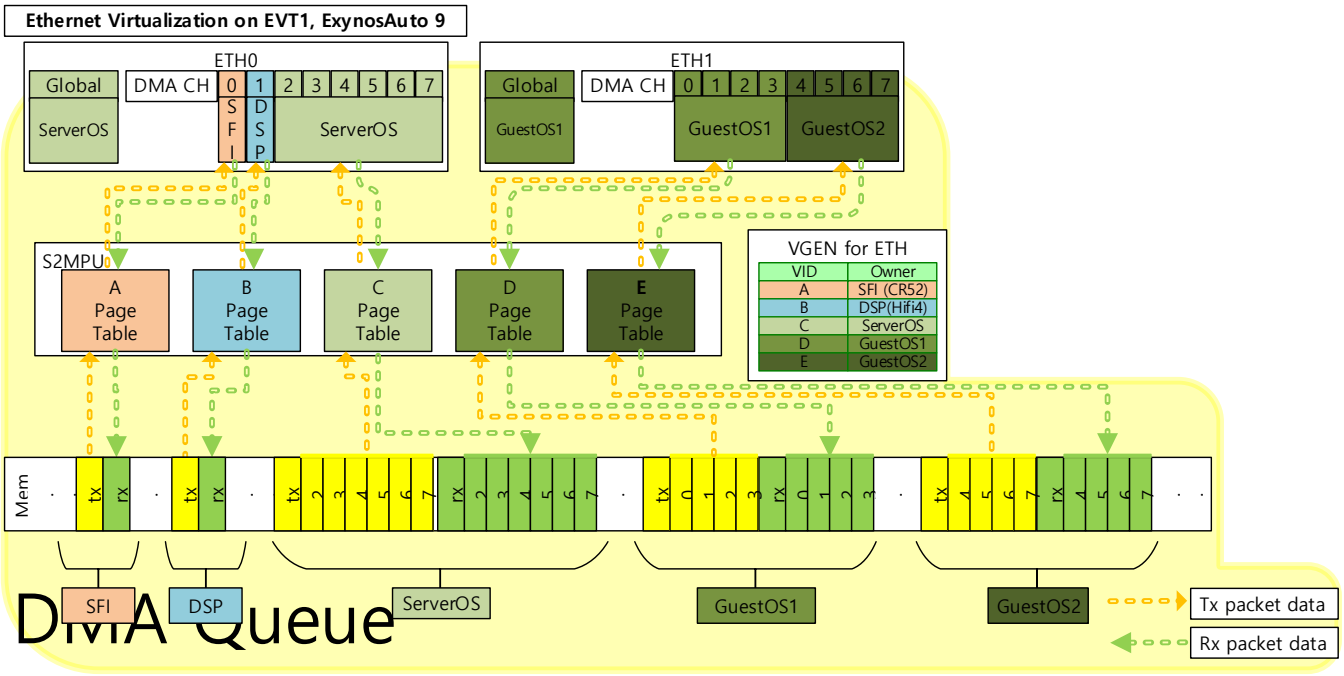


Figure 4: Rx/Tx Queues protection Concept Diagram

2.4.1 Isolation of Rx/Tx Queue of Ethernet CH dedicated to SFI/DSP/VMs (ServerOS, GuestOS1, and GuestOS2)

This section explains how Rx/Tx Queue of Ethernet CH dedicated to DSP/VMs is isolated.

- As seen above, SFI, DSP, ServerOS, GuestOS1 and GuestOS2 has its own Ethernet CHs assigned to each. A VID is assigned to SFI, DSP, ServerOS, GuestOS1 and GuestOS2 as well. The VID of SFI is ‘A’, and the VID of DSP is ‘B’.

- The VID is unique on the inside of Ethernet S2MPU, and hypervisor sets up a page table per VID and maintain them. In this case, hypervisor keeps 5 page tables in place - SFI, DSP, ServerOS, GuestOS1 and GuestOS2.

- The page table contains information of memory accessible to Rx/Tx DMA of Ethernet CHs.

- If R/W DMA of DSP’s Ethernet CH would try to access memory allocated to VMs, a S2MPU protection abort takes place. Also if R/W DMA of VM’s Ethernet CH would try to access memory assigned to DSP, a S2MPU protection abort occurs.

3 Rx Packet Flow Concept

3.1 Overview

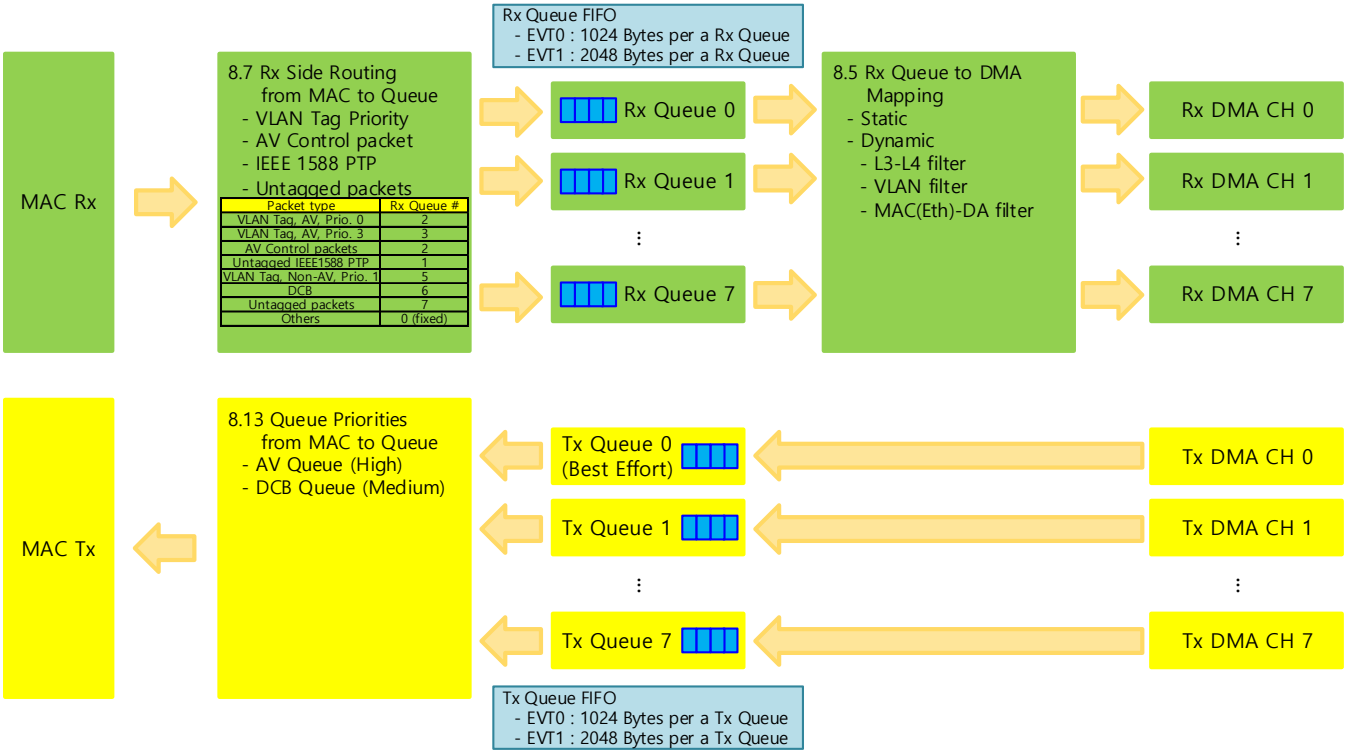


Figure 5: Rx/Tx Packet Flow Concept

In Rx side, there are 2 functions to control Rx Packet Flow.

- Routing from MAC to Queue: You can map Rx Queues with a packet type.

- In Figure 5, you can see one example.

  - You can route received packets, which have VLAN TAG with AV type and PCP (Priority Code Point) 0x000, to Rx Queue 2.

  - You can route received untagged IEEE 1588 PTP packets to Rx Queue 1.

  - If you do not set up the class for received packets, they go to Rx Queue 0 by default.

  - In case of VLAN Tagged IEEE1588 packets, you can route VLAN Tagged IEEE1588 packets to a specific Rx Queue according to configuration.

- You can route the VLAN Tagged IEEE1588 packets to 3 Queues as below.
  - Rx Queue assigned to untagged IEEE1588 packets
  - Rx Queue assigned to DCB packets a specific VLAN's PCP, in case of the VLAN Tagged IEEE1588 packets with same VLAN's PCP and EtherType non-0x22F0
  - Rx Queue assigned to AV packets with a specific VLAN's PCP, in case of the VLAN Tagged IEEE1588 packets with same VLAN's PCP and EtherType 0x22F0

- Mapping between Rx Queues and Rx DMA CHs
  - You can map a specific Rx Queue with a specific Rx DMA CH.

In Tx side, you can give priority for each Tx Queues.

- If you set up Tx Queue 1 as AV queue, the Queue will have high priority.
- If you set up Tx Queue 2 as DCB queue, the Queue will have medium priority.
- Tx Queue 0 is Best Effort Queue with low priority.
- Tx DMA CHs' have its own Tx Queue with same #. (Ex, Tx DMA CH 0 has Tx Queue 0.)

3.2 Mapping between Rx Queues and Rx DMA CHs

Figure 6: Mapping between Rx Queues and Rx DMA CHs

Each Rx Queue can select one of method for mapping to Rx DMA CH.

- Static Mapping

- Dynamic Mapping

3.2.1 Static Mapping

You can route all packets from a specific Queue to specific Rx DMA CH.

- For example, Rx Queue 0's all received packets can go to Rx DMA CH 0. Or you can send all Rx Queue 1's packets to Rx DMA CH 2.

- If you set up Static Mapping in an Rx queue, you cannot use Dynamic Mapping in the Rx Queue.

3.2.2 Dynamic Mapping

For Dynamic Mapping, Synopsys IP supports RxPacketFilter.

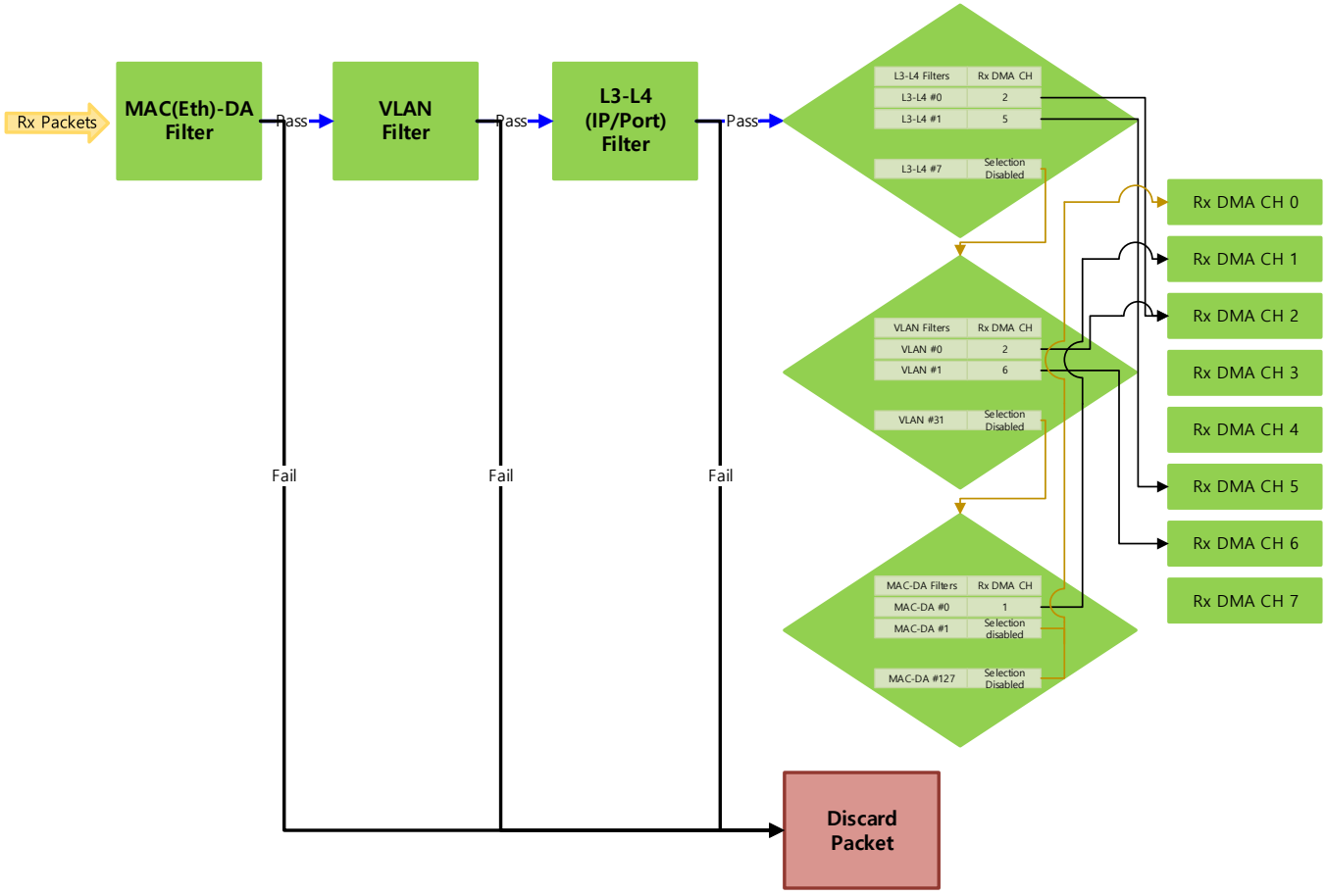


Figure 7: RxPacketFilter

If you select Dynamic Mapping in an Rx Queue, the Rx Queue's packets will be classified by RxPacketFilter configuration.

In RxPacketFilter, there are 2 actions for filtering and mapping.

Filtering: Synopsys IP supports 3 kinds of filters to classify packets.

Table 4: Filter type

Filter type	Criteria	Max filter slot	Comments
MAC-DA (Eth-DA)	MAC address	128	There are 3 sections. - 0~31, 32~63, 64~127 - 1~31 slots are support MAC SA (Source Address), Byte Masking, and Packet duplication. (0 slot is for default MAC address.)
VLAN	VLAN Tag	32	You can use VLAN tag ID (12bit or 16bit)
L3-L4	IP address or Port	8	

- If receive packet's MAC address, VLAN Tag, IP address, and Port can be compared with 3 types' filters according to Figure7: RxPacketFilter diagram.

  - Filters' order is MAC-DA -> VLAN -> L3-L4.

- If the received packet is failed to pass any filter, the packet is discarded.

- If the received packet is passed, then Rx DMA selection is working.

Rx DMA Selection:

- Rx DMA Selection's order is opposite from Filter's one.

  - L3-L4 -> VLAN -> MAC-DA

- If the passed packet's info is matched with one of L3-L4 filters' value and the filter's DMA Selection is enabled, the packet goes to the DMA CH which is selected by the filter.

- If the matched filter's DMA Selection is disabled, the packet's info is compared with VLAN's filters.

- If the packet's info is matched with one of VLAN filters' value and the filter's DMA Selection is enabled, the packet goes to the DMA CH which is selected by the filter.

- If the matched filter's DMA Selection is disabled, the packet's info is compared with MAC-DA's filters.

- If the packet's info is matched with one of MAC-DA filters' value and the filter's DMA Selection is enabled, the packet goes to the DMA CH which is selected by the filter.

- If the matched filter's DMA Selection is disabled, the packet goes to Rx DMA CH 0.

4 Audio Video Bridging (AVB)

4.1 AVB on ExynosAuto 9

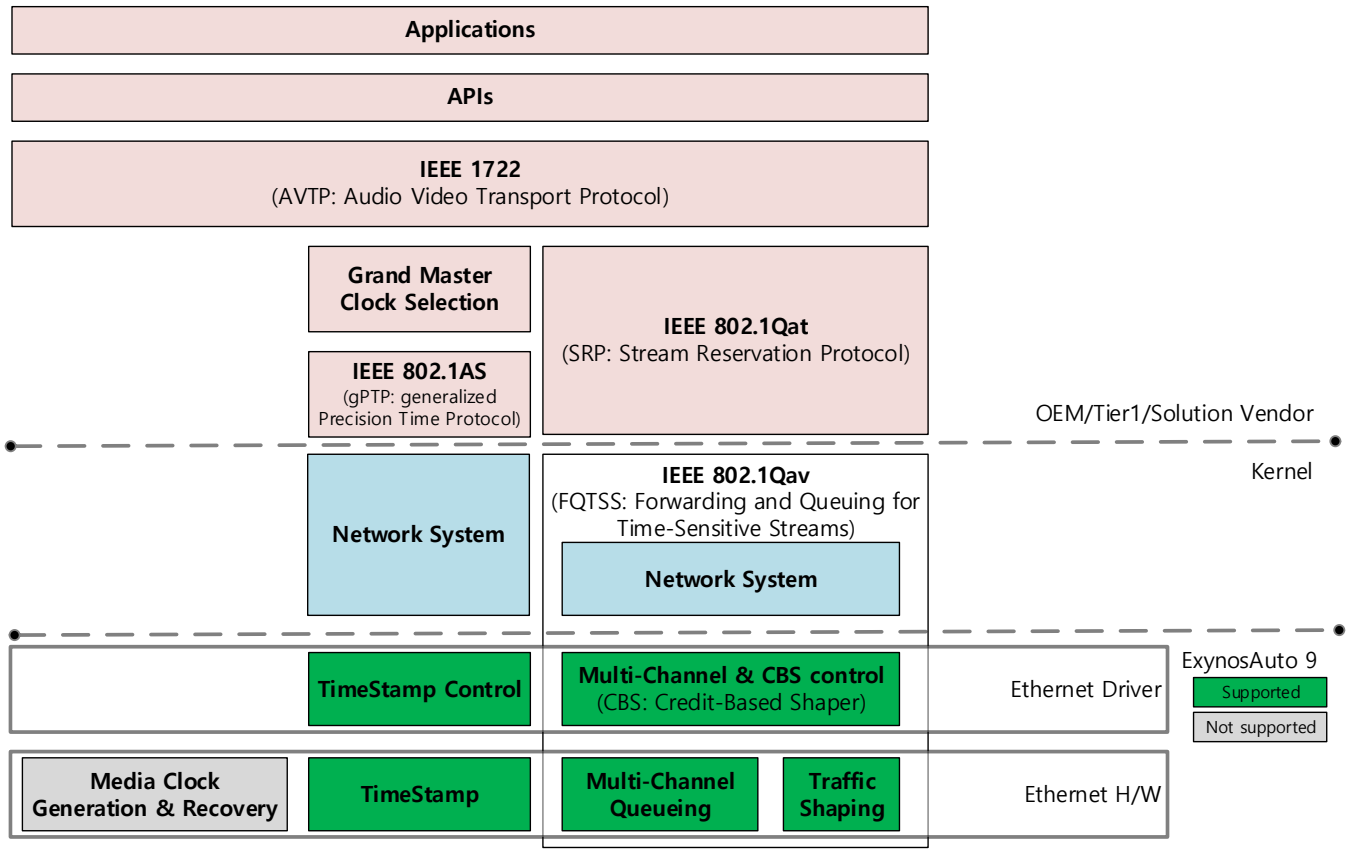


Figure 8: AVB Diagram with ExynosAuto 9

IEEE 802.1BA: IEEE Standard for Local and Metropolitan Area Networks---Audio Video Bridging (AVB) Systems

- This standard defines profiles that select features, options, configurations, defaults, protocols and procedures of bridges, stations and LANs that are necessary to build networks that are capable of transporting time sensitive audio and/or video data streams

IEEE 1722: AVB Transport Protocol

- Encapsulation for Audio Video Streams

- Need to implement by SW almost

- SoC does not have HW logic to generate and recover Media Clock between Ethernet IP and Clock for Audio and System. So Media Clock Generation and Recovery are not supported on our SoC.

IEEE 802.Qat: Stream Reservation Protocol

- To register/unregister Talker and Listener of Media Stream,
- To manage Bandwidth of Media stream
- Need to implement by SW almost

IEEE 802.1AS: gPTP or PTP

- To synchronize time for Media stream
- SoC supports HW timestamp in Ethernet IP.

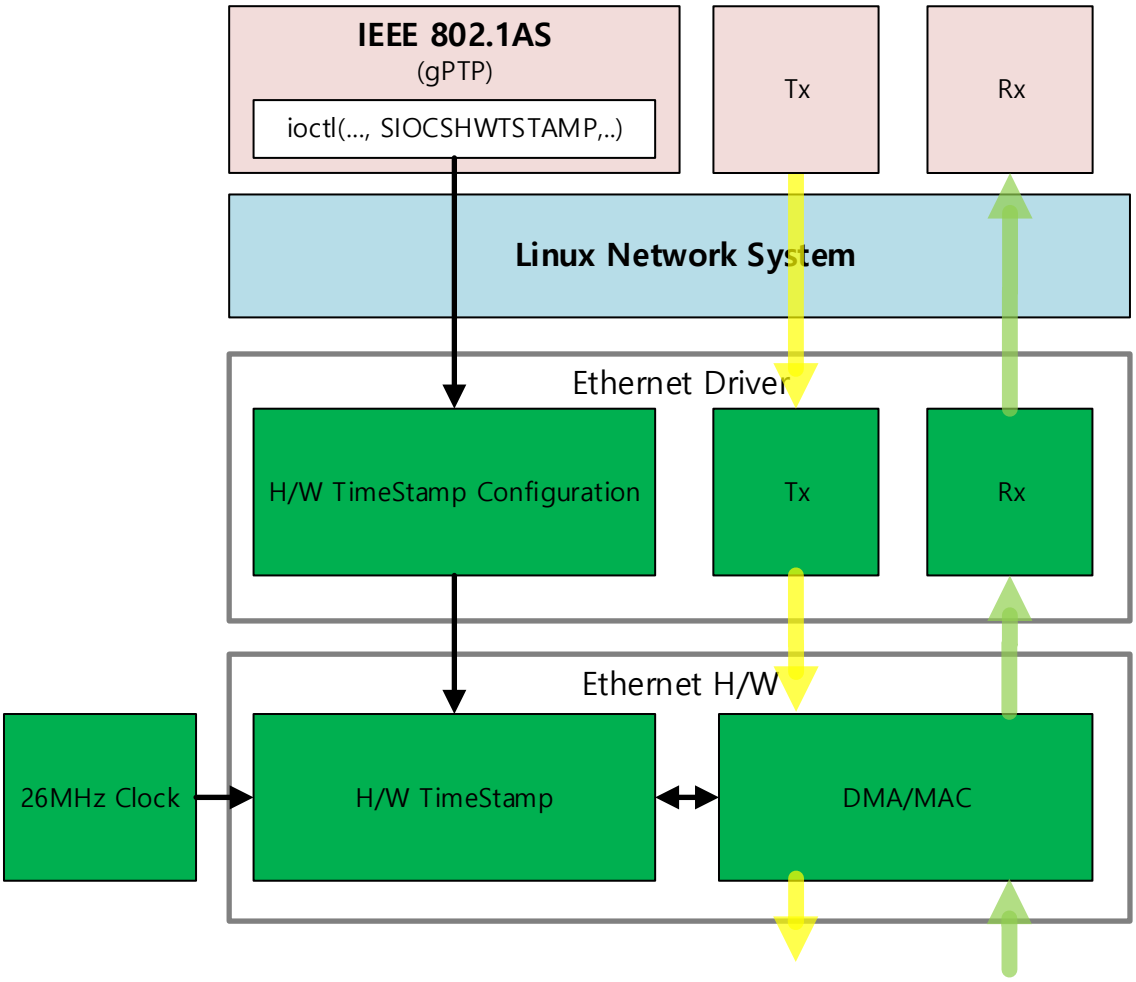


Figure 9: HW Timestamp in ExynosAuto 9

Application can set up H/W Timestamp by using ioctl with SIOCSHWTSTAMP. If HW Timestamp is enabled, Ethernet MAC chip record timestamp for PTP packets. And the application can get the Timestamps to calculated time difference between Master and Slave in AVB Network to synchronize the time between Master and Slave.

IEEE 802.1Qav: Traffic shaping for AV streams

- Forwarding and Queuing Time-sensitive stream
- SoC supports Queuing and Traffic Shaping for reserved Traffic by Credit-Based Shaper.

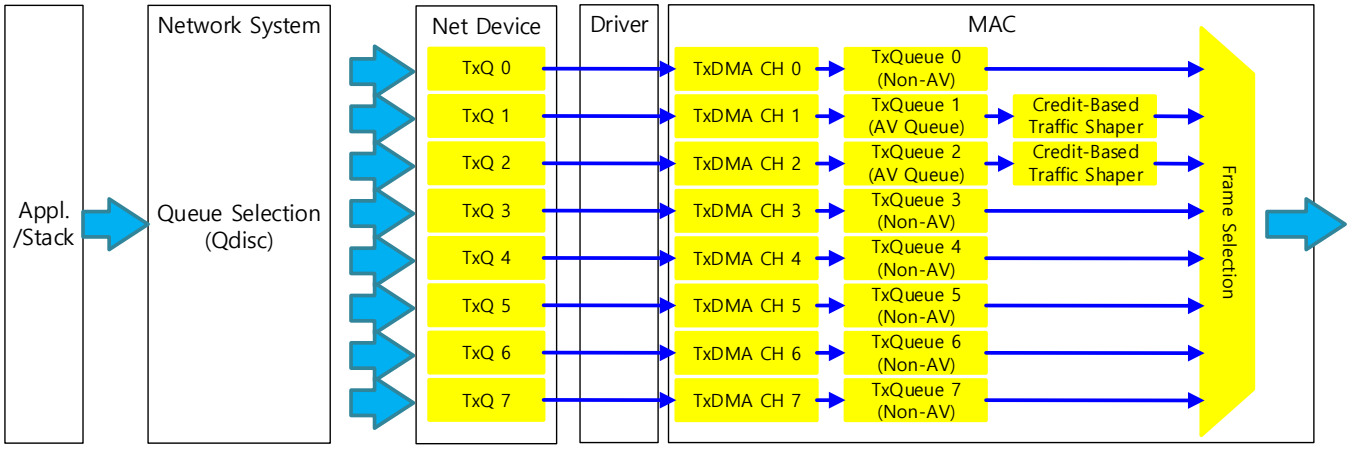


Figure 10: Credit-Based Shaper in ExynosAuto 9

In Linux, you can enable Credit-Based Shaper for AV Queues. EA9's Ethernet IP can set up AV queues in TxQueue 1 to 7. And you can enable CBS in these AV queues.

You can configure CBS parameters through ioctl with SIOCDEVPRIVATE. (You can see more explanation in APPENDIX Ch. 5.3.3.)

4.2 **Possibility on ExynosAuto 9**

Table 5: AVB Node Type

Type	Supported on EA9	Comments
Audio Talker	TBD	If stream sync or Media clock is synced by SW of OEM/Tier1/AVB solution, You can use Audio Talker/Listener on AVB stack. HW media clock generation and Recovery: Not supported - EVT1: TBD
Audio Listener	TBD	If stream sync or Media clock is synced by SW of OEM/Tier1/AVB solution, You can use Audio Talker/Listener on AVB stack. HW media clock generation and Recovery: Not supported - EVT1: TBD
Video Talker	Supported	Stream sync is controlled by AVB stack and player.
Video Listener	Supported	Stream sync is controlled by AVB stack and player.

5 APPENDIX

5.1 Register Maps

SFR map for DMA channel access (EVT0)

- 128 byte offset between EQOS_DMA_CH* DMA channel registers
- 64byte offset between EQOS_MTL_Q* Queue Transmit Operation Mode registers

BLK_FSYS2	
UFS_EMBD0	0x17E6_0000
UFS_EMBD0	0x17E5_0000
UFS_EMBD0	0x17E4_0000
UFS_EMBD0	0x17E3_0000
UFS_EMBD0	0x17E2_0000
UFS_EMBD0	0x17E1_0000
UFS_EMBD0	0x17E0_0000
UFS_EMBD0_VGEN	0x17DF_0000
UFS_EMBD1_FMP	0x17DE_0000
UFS_EMBD0_VGEN	0x17DD_0000
UFS_EMBD0_FMP	0x17DC_0000
	0x17DB_0000
SerialFLASH	0x17DA_0000
ETHERNET1	0x17D9_0000
ETHERNET0	0x17D8_0000
	0x17D7_0000
	0x17D6_0000
	0x17D5_0000
	0x17D4_0000

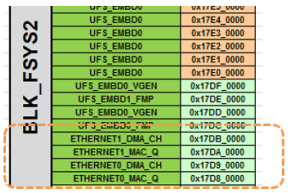
DMA registers	Offset
EQOS_MAC Registers	0x0000
EQOS_MTL Registers	0x0c00
EQOS_MTL_Q0~7 Registers	0x0d00
EQOS_DMA Registers	0x1000
EQOS_DMA_CH0 Registers	0x1100
EQOS_DMA_CH1 Registers	0x1180
EQOS_DMA_CH2 Registers	0x1200
EQOS_DMA_CH3 Registers	0x1280
EQOS_DMA_CH4 Registers	0x1300
EQOS_DMA_CH5 Registers	0x1380
EQOS_DMA_CH6 Registers	0x1400
EQOS_DMA_CH7 Registers	0x1480

128 byte offset

Figure 11: Register Map in EVT0

SFR map for DMA channel access (EVT1) 2018.10.17

- 4KB offset between EQOS_DMA_CH* DMA channel registers and EQOS_MTL_Q* Queue Transmit Operation Mode registers



Ex) Ethernet0

SFR address in bus view

0x17D8_0000

MAC and MTL registers	Offset address
EQOS_MAC Registers	0x0000
EQOS_Timestamp Registers	0x1000
EQOS_MTL Registers	0x2000
EQOS_MTL_Q0	0x8000
EQOS_MTL_Q1	0x9000
EQOS_MTL_Q2	0xA000
EQOS_MTL_Q3	0xB000
EQOS_MTL_Q4	0xC000
EQOS_MTL_Q5	0xD000
EQOS_MTL_Q6	0xE000
EQOS_MTL_Q7	0xF000

64KB

128KB

0x17D9_0000

DMA registers	Offset
EQOS_DMA Registers	0x1000
EQOS_DMA_CH0	0x8000
EQOS_DMA_CH1	0x9000
EQOS_DMA_CH2	0xA000
EQOS_DMA_CH3	0xB000
EQOS_DMA_CH4	0xC000
EQOS_DMA_CH5	0xD000
EQOS_DMA_CH6	0xE000
EQOS_DMA_CH7	0xF000

64KB

Ethernet0 (Total 128K)
- 0x17D8_0000 ~ 0x17D8_1FFF
  - MAC/MTL Mode set common registers and Q registers * 8
- 0x17D9_0000 ~ 0x17D9_FFFF
  - DMA configuration, 4K offset for TX and RX DMA channel register * 8

Ethernet1 (Total 128K)
- 0x17DA_0000 ~ 0x17DA_1FFF
  - MAC/MTL Mode set common registers and Q registers * 8
- 0x17DB_0000 ~ 0x17DB_FFFF
  - DMA configuration, 4K offset for TX and RX DMA channel register * 8

Figure 12: Register Map in EVT1

SFR map for DMA channel access (EVT1) 2018.10.17

Implementation of Static remap function in bus – Ethernet0

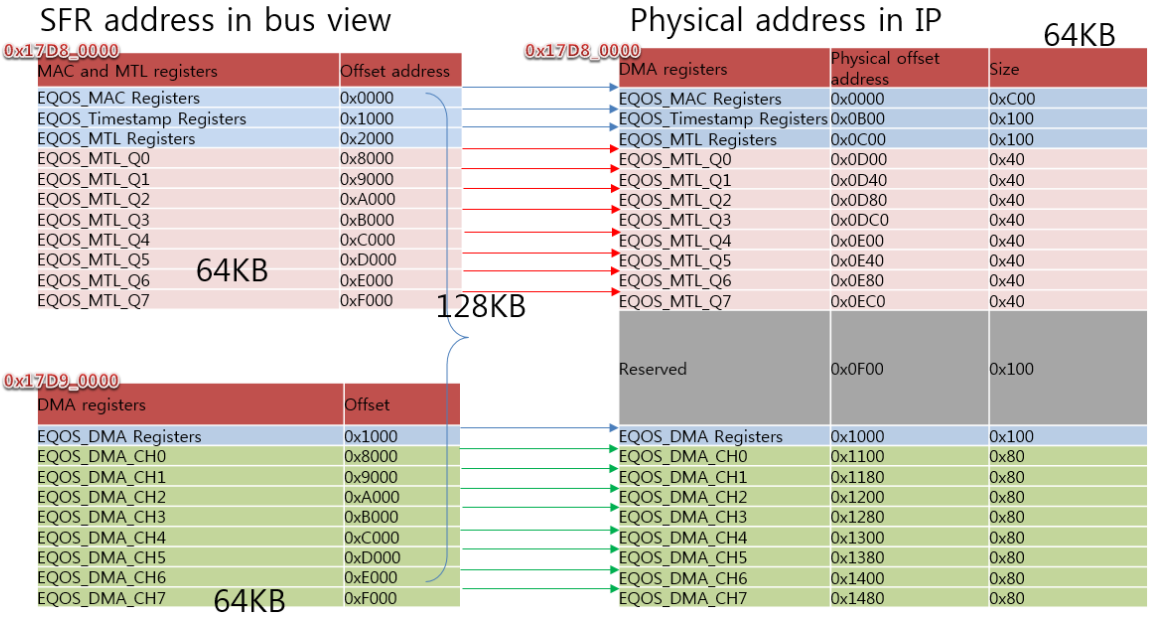


Figure 13: Register Map in EVT1, Ethernet 0

SFR map for DMA channel access (EVT1) 2018.10.17

Implementation of Static remap function in bus – Ethernet 1

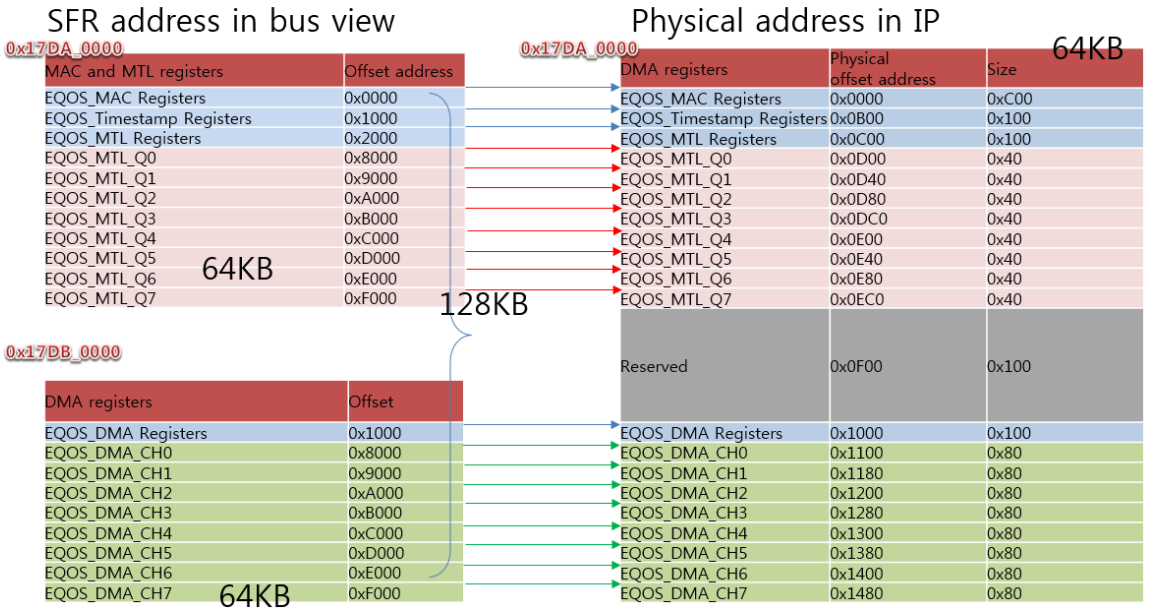


Figure 14: Register Map in EVT1, Ethernet 1

5.2 Diagrams about Interrupts

DMA Interrupt Design in EVT0

- Ethernet interrupts are connected to main GIC400 only
- No interrupt connection to other hosts
- sbd_perch_tx_intr_o[7:0] bits are merged to 1 bit
- sbd_perch_rx_intr_o[7:0] bits are merged to 1 bit

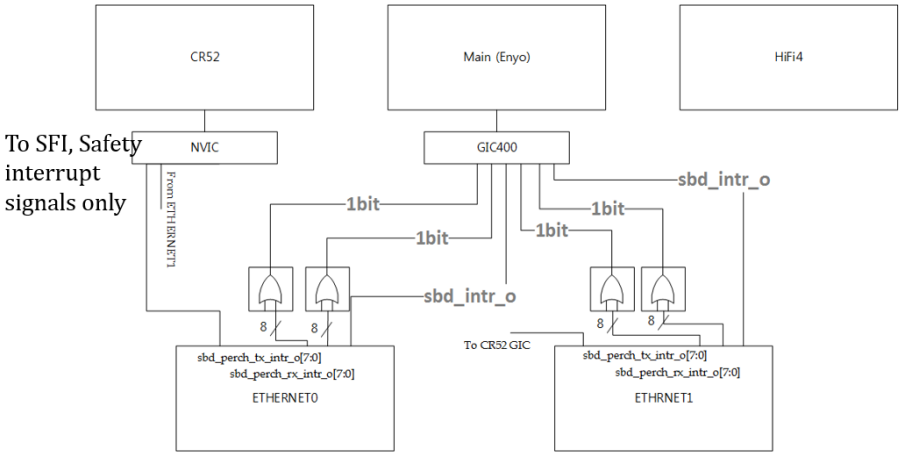
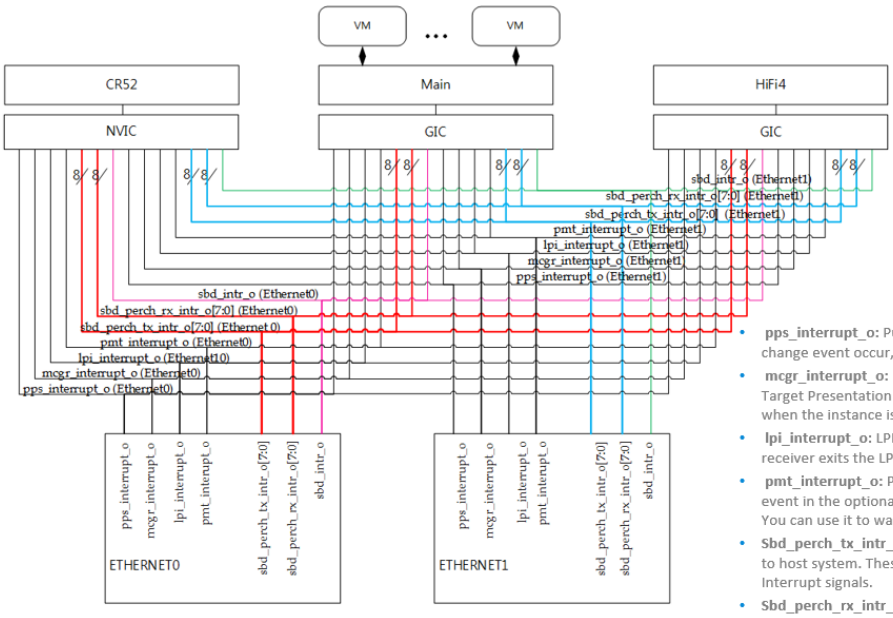


Figure 15: Interrupt lines in EVT0

Interrupts (All in EVT1)



- **pps_interrupt_o:** Pulse Per Second Interrupt. When pps value change event occur, this interrupt gives notification to the hosts.
- **mcgr_interrupt_o:** MCGR DMA Read Request Used to Request a Target Presentation time value for one of the comparator modules when the instance is running in recovery mode.
- **lpi_interrupt_o:** LPI Interrupt. This signal is high when the MAC receiver exits the LPI state.
- **pmt_interrupt_o:** PMT Interrupt. This signal indicates an interrupt event in the optional PMT module of the DWC_ether_qos core. You can use it to wake the system from the Sleep mode.
- **Sbd_perch_tx_intr_o[7:0] :** Per Channel Transmit Interrupt signal to host system. These are per channel Transmit Completion Interrupt signals.
- **Sbd_perch_rx_intr_o[7:0] :** Per Channel Receive Interrupt signal to host system. These are per channel Receive Completion Interrupt signals,
- **Sbd_intr_o:** Subsystem Interrupt. When this signal is high, it indicates an interrupt event to the application from one or multiple DMA Channels.

Figure 16: Interrupt lines in EVT1

5.3 **Qdisc (Queuing discipline) and Tx Queueing**

You can use Linux Qdisc with our 8 DMA CHs to manage queues in Kernel. There is no change in our driver to support Qdisc. If you map your Net device's queues with 8 DMA CHs and use Qdisc, you can manage your queues in Net device more efficiently.

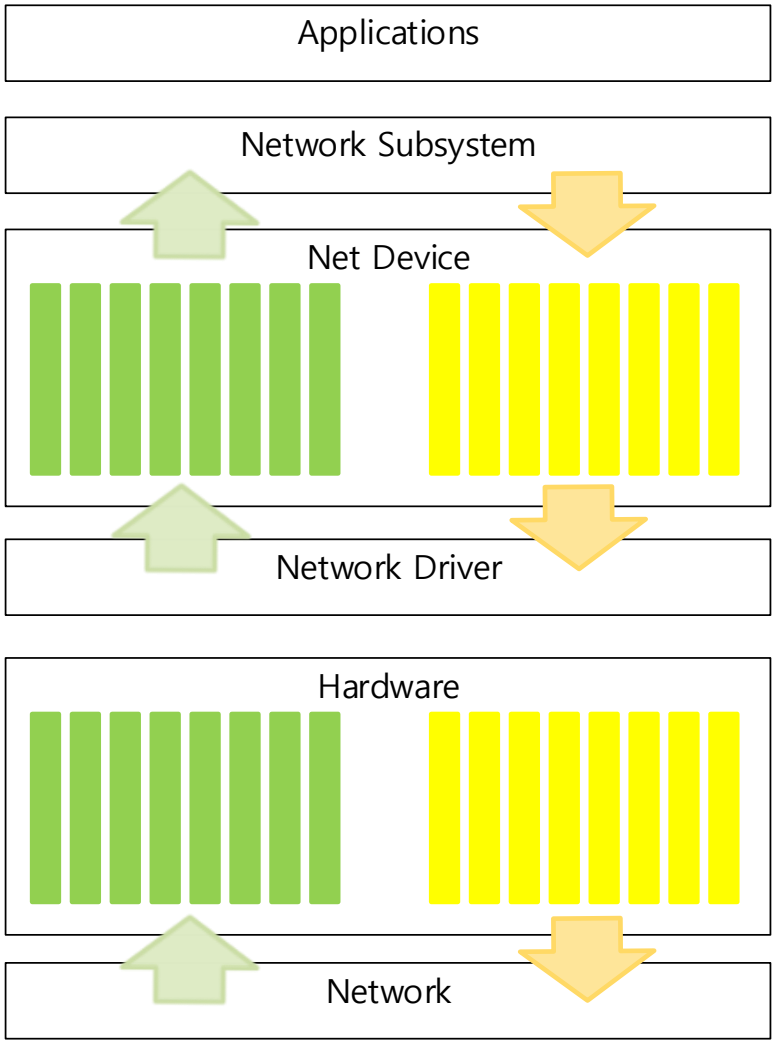


Figure 17: Linux Network Multi-Queue

Linux supports Multi-Queue in Net device to support Multi-Channel in Network. You can register a Net device as Multi-Queue, when you register Network Driver to Kernel, If you do this, Multi-Queue is created in Net device. Network Driver will send Data of a specific Queue of Net device to Network through mapped Tx DMA CH.

Multi-Channel can be configured in Device Tree by Ethernet driver.

Linux supports Qdisc (Queueing discipline) to support Traffic Control and QoS Control before sending Data to Net device in Kernel space. Qdisc are queueing Packets and support Traffic shaping and Queue Scheduling for QoS.

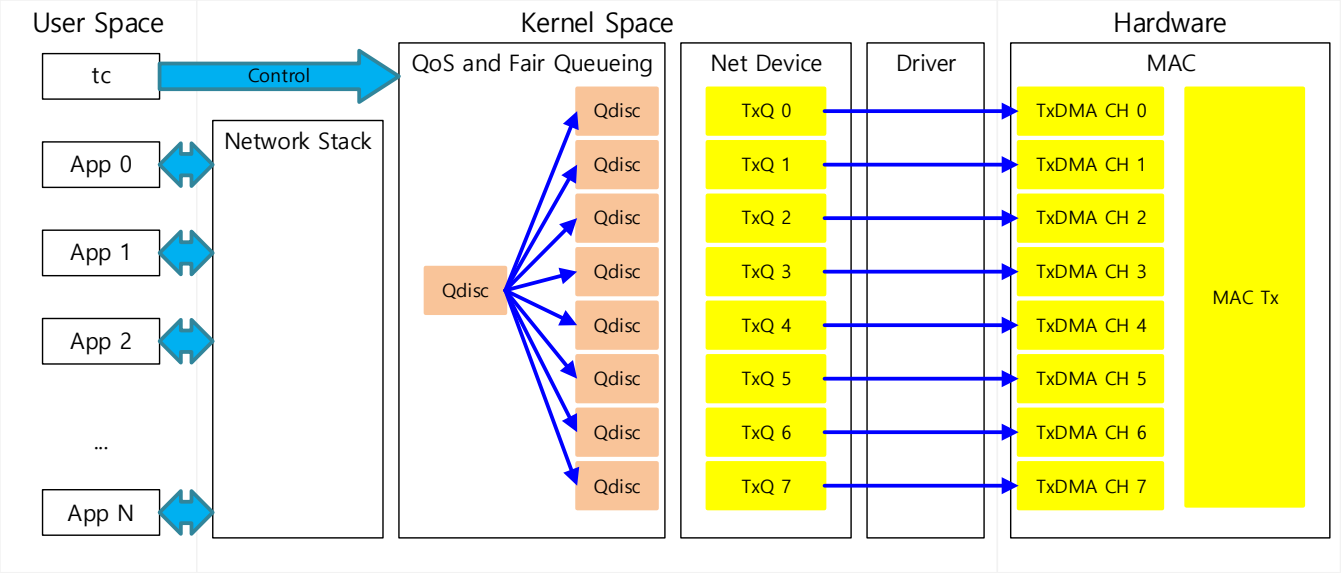


Figure 18: Tx Packet Flow with Qdisc and Multi-Channel

Qdisc is configured by tc command in User space. Packets, which are sent by Applications, are sent to Queues of Net Device through Qdisc, and then the Packets go to mapped Tx DMA CHs, which are assigned to specific Tx Queues of Net Device.

multiq and mqprio among several Qdisc types can support HW multi-Channel.

5.3.1 multiq

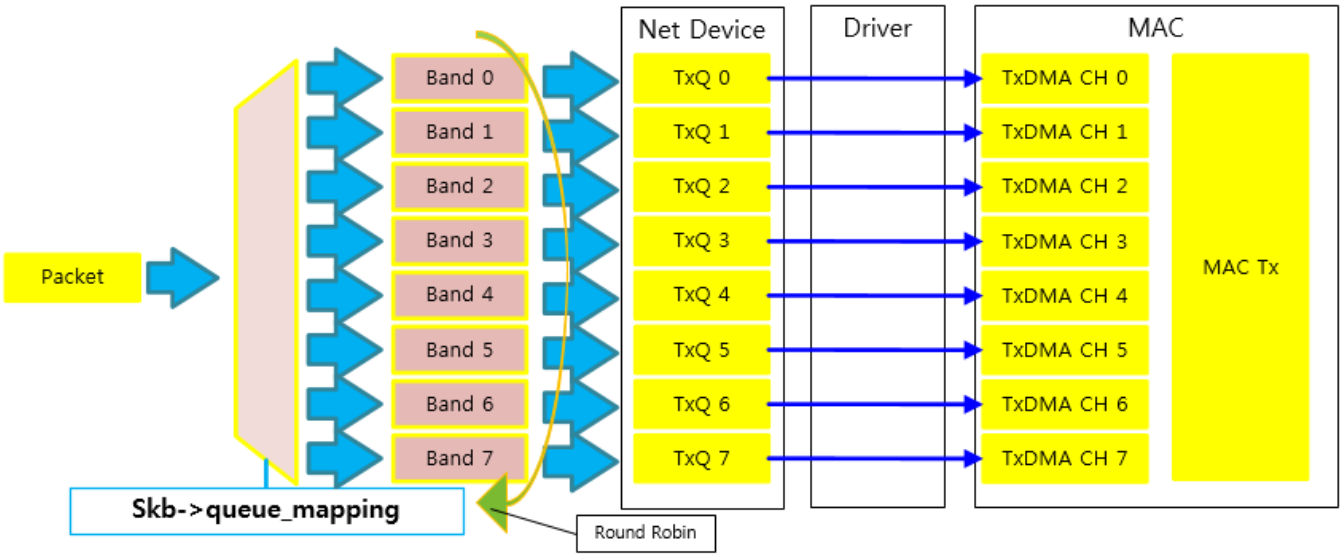


Figure 19: Qdisc - multiq

multiq maps HW Tx Channels and Bands as 1 on 1. Bands' scheduling is Round robin, and there is no priority between Bands.

A packet, which is sent to Qdisc, goes to a specific Band assigned by skb->queue_mapping.

```
# tc qdisc add dev eth0 root handle 1: multiq

# tc filter add dev eth0 parent 1: protocol ip prio 1 u32 ₩
    match ip src 192.168.100.2 ₩
    action skbedit queue_mapping 1
```

Figure 20: Qdisc - example for multiq: tc command

Here is one example for multiq by tc command.

If you enter this tc command, you can configure 8 Bands like below.

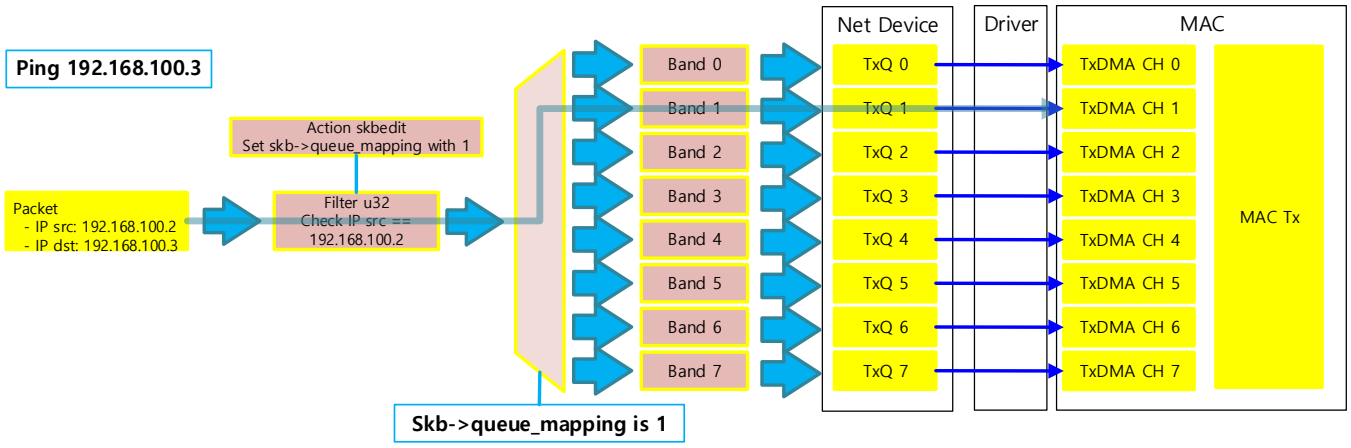


Figure 21: Qdisc - example for multiq

A packet (IP src: 192.168.100.2), which is sent to filter u32, skb->queue_mapping is set 1 by action skbedit, and then the packet goes to Network through Band 1, TxQ 1, and Tx DMA CH 1.

5.3.2 mqprio

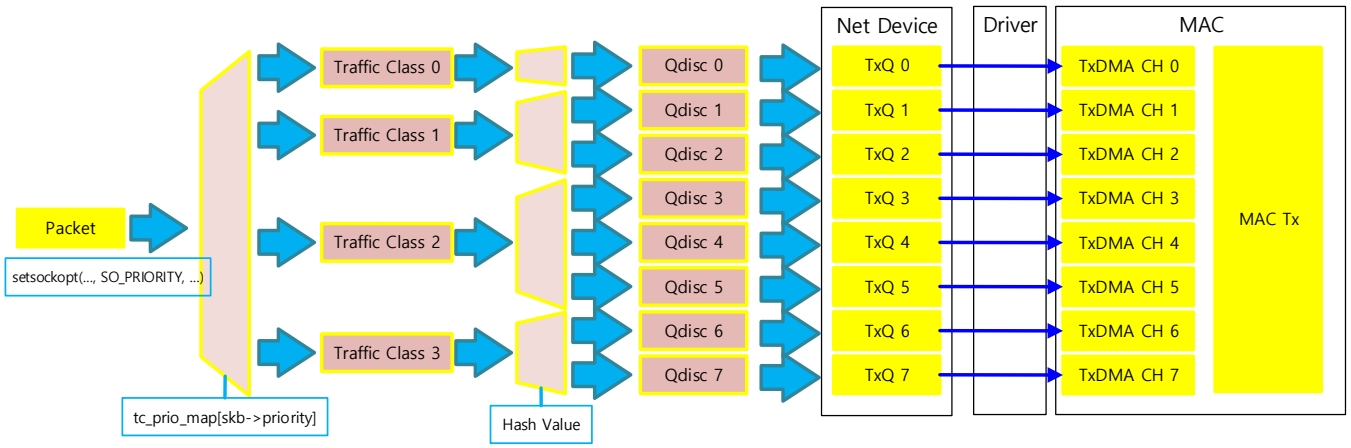


Figure 22: Qdisc - mqprio

Traffic Classes are mapped to Priorities (from 0 to 15), Data is sent by Multi-Channel assigned to each Traffic Queue. A Traffic Class can be mapped to several Channels. Priorities (from 0 to 15) have their own Traffic Class, and a packet's priority is set by setsockopt API.

```
# tc qdisc add dev eth0 handle 100: parent root mqprio
num_tc 4 ₩
         map 2 2 1 3 0 1 3 2 2 2 2 2 2 2 2 2 ₩
         queues 1@0 2@1 3@3 2@6₩
         hw 0
```

Figure 23: Qdisc - example for mqprio: tc command

Here is one example for mqprio by tc command.

If you enter this tc command, you can configure 4 Traffic classes and 8 Qdisc like below.

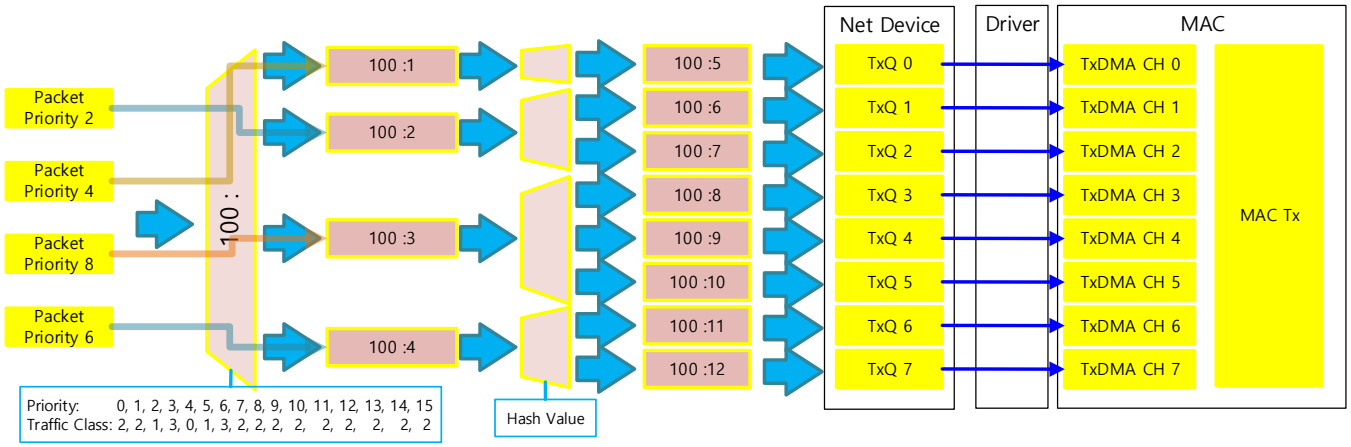


Figure 24: Qdisc - example for mqprio

Through above tc command, you can make 4 Traffic classes. 100:1(Traffic class 0) has 1 Qdisc (100:5), 100:2(Traffic class 1) has 2 Qdisc (100:6 and 100:7), 100:3(Traffic class 2) has 3 Qdisc (100:8, 100:9, and 100:10), and 100:4(Traffic class 3) has 2 Qdisc (100:11 and 100:12). 16 Priorities have their own traffic class by "map" parameter.

If a packet's priority is 2, the packet goes to Network through Traffic class 1, which is assigned to priority 2, and 100:6 or 100:7.

5.3.3 Credit-Based Shaper

Synopsys Ethernet IP can set AV Queue in Tx Queue 1 to 7 to support 802.11Qav. AV Queue can control Bandwidth by using Credit-Based Shaper Algorithm.

In Linux-4.15 kernel, Qdisc can configure Credit-Based Shaper.

In Linux-4.14.70 kernel, we need to use ioctl which is supported in Ethernet driver.

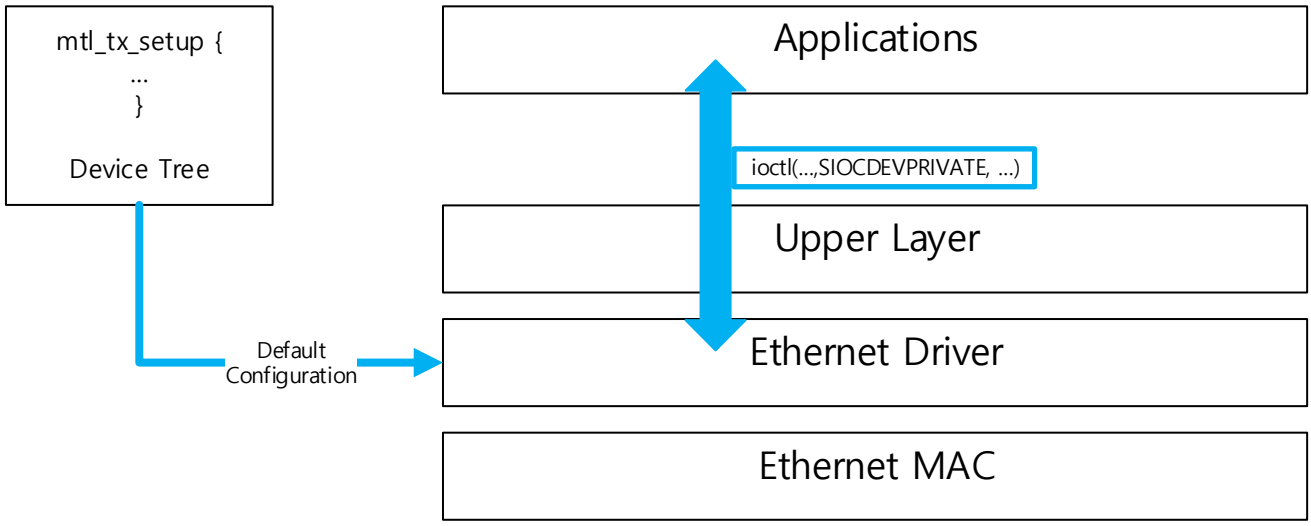


Figure 25: SIOCDEVPRIVATE for Credit-Based Shaper

You can configure send_slope, low_slope, high_credit, and low_credit through ioctl (SIOCDEVPRIVATE).

<End of Document>