BMW IDCEvo :: AVB Traffic Shaping

Version	State	Date	Remarks
1.0	Draft	04 Jun 2024	Exploring and collecting info.

CBS formula for DWC GMAC

idleSlope and sendSlope

Example: -

Mode = 100Mbps

portTransmitRate = 100 Mbps

idleSlope = 70 Mbps (assuming 70% bandwidth reserved for a higher priority traffic class)

sendSlope = 30 Mbps

Cycle time for various speed modes: -

1 cycle time for 100 Mbps = 40 ns
1 cycle time for 1000 Mbps = 8 ns

1 cycle time for 2500 Mbps = 3.2 ns

Note:

How to calculate "Cycle Time/Calculation interval" for a data rate?

Example: 100Mbps link

Bit width = 1/100 M = 1/(100 x 10^6) = 10ns

The maximum base frequency for a 100Mbps signal will be produced when the input bitstream is 010101010101...01010. As each bit lasts 10ns then the period between consecutive levels is 20ns.

With a 3-level coding the maximum base frequency will occur with the code + - + - + - + - ....+ - + - + -. If this code is not included and the maximum transition is made as

+ - 0 + - 0 + - 0 + - 0 + - 0 + - 0 + ..., then the pattern "+ - 0 +" will repeat as 4 bits i.e. 4 x 10ns = 40ns.

Thus, Cycle Time for 100Mbps link = 40ns

Measurement Interval for various AVB class

AVB Class	Measurement Interval
Class A	125 uS
Class B	250 uS
Class C	1333 uS

CBS parameter calculation

Credit accumulated per cycle (=40ns)

1s accumulates credits in bits = 70Mbits

40ns ..... = (70x 10^6) x (40 x 10 ^-9) = 2.8 bits

So, credits accumulated in 1 cycle (40ns) = 2.8 bits

Credit drained per cycle (=40ns)

1s drains credits in bits = 30Mbits

40ns ..... = (30 x 10^6) x (40 x 10^-9) = 1.2 bits

So, credits drained in 1 cycle (40ns) = 1.2 bits

idleSlopeCredit and sendSlopeCredit values

Formula: -

I. bandwidthFraction

bandwidthFraction = idleSlope / portTransmitRate

Ex . - If 100Mbps is portTransmitRate and idleSlope is 50Mbps, then

bandwidthFraction = 50/100 = 0.5

II. sendSlope

sendSlope = (idleSlope – portTransmitRate)

III. loCredit

loCredit = maxFrameSize x (sendSlope/portTransmitRate)

IV. hiCredit

hiCredit = maxInterferenceSize x (idleSlope/portTransmitRate)

hiCredit

maxInterferenceSize is simply the size of one maximum sized Ethernet frame, which is 2000 octets, meaning 2000 x 8 bits = 16000 bits. In the case of a 100 Mb/s Ethernet

connection and a traffic class (queue) with 50% bandwidth reservation,

hiCredit = 16000 bits x (50 Mb/s /100 Mb/s) = 16000 x 0.5 = 8000 bits.

hiCredit = 8000 bits

loCredit

In order to calculate loCredit we need to find the maxFrameSize, which is the maximum number of bits that can be transmitted in a frame for a stream (traffic class).

This value is defined through the SRP protocol, which uses a traffic specification (TSpec) for each stream.

In addition to defining the maximum number of bits per frame, the TSpec also defines a maximum number of frames which can be transmitted (maxIntervalFrames).

Both the maxFrameSize and the maxIntervalFrames are measured over a **class measurement interval** which applies at the source (talker) of the stream.

For class A, measurement interval is 125 s and for class B it is 250 s.

Example: -

link speed = 100 Mb/s and a traffic class with 50% bandwidth reservation.

If we are using traffic class A, then maxFrameSize will be 50% of the maximum number of bits I can transmit during 125 s over a 100 Mb/s link. Which is:

100 000 000 b/s x 0,000125 s x 0.5 = 6250 bits. However, this number needs to be an integral number of octets, as 6250 is not, the number is rounded down to 6248.

As we know that idleSlope = bandwidth reserved of the class (queue), 50Mbps

So, sendSlope = idleSlope - portTransmitRate = (50 - 100) Mbps = -50Mbps

Formula: -

loCredit = maxFrameSize x (sendSlope/portTransmitRate)

loCredit = 6248 x (-50/100) = -3124 bits.

CBS formula for DWC XGMAC

Reference: DWC_xgmac_databook_WM-7543.pdf

Programming example for idleSlope, sendSlope, hiCredit, and loCredit
Consider a case where 3 TCs are in use (TC0, TC1, and TC2) and of which:

TC2 [AV enabled] has 45% bandwidth allocated
TC1 [AV enabled] has 35% bandwidth allocated and the remaining bandwidth is assigned to TC0
TC0 [Best Effort] gets the left over bandwidth of 20%

Consider 1500 bits for hiCredit and 800 bits for loCredit programming

TC2 Programming
idleSlope: (45/100) * 32 * 1024 = 14746 (45 is the bandwidth, 32 since MAC uses a 32 bit interface, and 1024 is the scaling factor)
sendSlope: ((100-45)/100) * 32 * 1024 = 18022 (100- bandwidth must be used to compute sendSlope)
hiCredit: ((100-45)/100) * 1500 * 1024 = 844800
loCredit: ((100-45)/100) * -800 * 1024 = -446600

TC1 Programming
idleSlope: (35/100) * 32 * 1024 = 11469
sendSlope: ((100-35)/100) * 32 * 1024 = 21299
hiCredit: ((100-35)/100) * 1500 * 1024 = 998400
loCredit: ((100-35)/100) * -800 * 1024 = -532480
The 2's complement value must be programmed for loCredit field because it is always a negative value.

XGMAC hardware register details for sendSlope and idleSlope values:

I. Hw Register: MTL_TC(#i)_Quantum_Weight (for i = 0; i <= DWCXG_NUM_TC-1)
(Ref - Pg 1234 of XGMAC dwc doc.)

Bits	Name	Memory Access	Description
20:0	QW	R/W	idleSlopeCredit, Quantum or Weights. ■ **idleSlopeCredit** ❑ When Audio Video Bridging feature is enabled, this field contains the idleSlopeCredit value required for the credit-based shaper algorithm for Queue 1. This is the rate of change of credit in bits per cycle (3.2 ns for 2.5/10 Gbps; 8 ns for 1000 Mbps) when the credit is increasing. The software must program this field with computed credit in bits per cycle scaled by 1024. The maximum value is portTransmitRate, that is, 0x2000 in 1000/2500 Mbps mode and 0x8000 in 10 Gbps mode. Bits[20:16] must be written to zero.

II. Hw register: MTL_TC(#i)_SendSlopeCredit (for i = 1; i <= DWCXG_NUM_TC-1)

(Ref - Pg 1236 of XGMAC dwc doc.)

15:0	SSC	R/W	sendSlopeCredit Value. ■ When Audio Video Bridging operation is enabled, this field contains the sendSlopeCredit value required for credit-based shaper algorithm for TC<n>. The software must program this field with computed credit in bits per cycle scaled by 1024. This field must be programmed with absolute sendSlopeCredit value. The credit-based shaper logic subtracts it from the accumulated credit when TC<n> is selected for transmission. See the application note for more details on programming this value. **Value After Reset:** 0x0 **Exists:** Always

III. Hw register: MTL_TC(#i)_HiCredit (for i = 1; i <= DWCXG_NUM_TC-1)

(Ref - Pg 1237 of XGMAC dwc doc.)

28:0	HC	R/W	hiCredit Value. ■ When the Audio Video Bridging feature is enabled, this field contains the hiCredit value required for the credit-based shaper algorithm. This is the maximum value that can be accumulated in the credit parameter. This is specified in bits scaled by 1024. The maximum value is maxInterfer-enceSize, that is, best-effort maximum packet size (16384 bytes or 131072 bits). The value to be specified is 131072 * 1024 = 134217728 or 0x0800_0000 **Value After Reset:** 0x0 **Exists:** Always

IV. Hw register: MTL_TC(#i)_LoCredit (for i = 1; i <= DWCXG_NUM_TC-1)

(Ref - Pg 1238 of XGMAC dwc doc.)

28:0	LC	R/W	loCredit Value. ■ When Audio Video Bridging operation is enabled, this field contains the loCredit value required for the credit-based shaper algorithm. This is the minimum value that can be accumulated in the credit parameter. This is specified in bits scaled by 1024. The maximum value is maxPacket-Size transmitted from this queue, that is, 16384 * 8 * 1024 = 134217728 or 0x0800_0000. Because it is a negative value, the programmed value is twos complement of the value, that is, 0x1800_0000. **Value After Reset:** 0x0 **Exists:** Always

CBS setting using EMAC driver

I. **Reading of dts**

```
File: - drivers/net/ethernet/stmicro/stmmac/stmmac_platform.c

static int stmmac_mtl_setup(struct platform_device *pdev, struct plat_stmmacenet_data *plat) {

        //....

       else if (of_property_read_bool(q_node,  "snps,avb-algorithm")) {
                    plat->tx_queues_cfg[queue].mode_to_use = MTL_QUEUE_AVB;

                    /* Credit Base Shaper parameters used by AVB */
                    if (of_property_read_u32(q_node, "snps,send_slope",
                          &plat->tx_queues_cfg[queue].send_slope))
                          plat->tx_queues_cfg[queue].send_slope = 0x0;
                    if (of_property_read_u32(q_node, "snps,idle_slope",
                          &plat->tx_queues_cfg[queue].idle_slope))
                          plat->tx_queues_cfg[queue].idle_slope = 0x0;
                    if (of_property_read_u32(q_node, "snps,high_credit",
                          &plat->tx_queues_cfg[queue].high_credit))
                          plat->tx_queues_cfg[queue].high_credit = 0x0;
                    if (of_property_read_u32(q_node, "snps,low_credit",
                          &plat->tx_queues_cfg[queue].low_credit))
                          plat->tx_queues_cfg[queue].low_credit = 0x0;
       }

      //....

}
```

II. **Configuring DWC IP hardware registers for configuring CBS parameters read from DTS**

```
File: - drivers/net/ethernet/stmicro/stmmac/dwmac-sxgmac.c
-----------------------------------------------------------------
static int dwmac_sxgmac_probe(struct platform_device *pdev)
{
   //....
   if (pdev->dev.of_node) {
       plat_dat = stmmac_probe_config_dt(pdev, stmmac_res.mac);
       if (IS_ERR_OR_NULL(plat_dat)) {
              dev_err(&pdev->dev, "dt configuration failed\n");
              return PTR_ERR(plat_dat);
       }
   }
   //....
}

struct plat_stmmacenet_data *
stmmac_probe_config_dt(struct platform_device *pdev, u8 *mac)
{
```

```
    //....
    rc = stmmac_mtl_setup(pdev, plat);
    if (rc) {
            stmmac_remove_config_dt(pdev, plat);
            return ERR_PTR(rc);
    }

    //....
    return plat;
}

static const struct stmmac_hwif_entry {
        bool gmac;
        bool gmac4;
        bool xgmac;
        u32 min_id;
        u32 dev_id;
        const struct stmmac_regs_off regs;
        const void *desc;
        const void *dma;
        const void *mac;
        const void *hwtimestamp;
        const void *mode;
        const void *tc;
        const void *mmc;
        int (*setup)(struct stmmac_priv *priv);
        int (*quirks)(struct stmmac_priv *priv);
} stmmac_hw[] = {

    //....
     {
                //....
                .min_id = DWXGMAC_CORE_2_10,
                .dev_id = DWXGMAC_ID,
                .regs = {
                        .ptp_off = PTP_XGMAC_OFFSET,
                        .mmc_off = MMC_XGMAC_OFFSET,
                },
                .desc = &dwxgmac210_desc_ops,
                .dma = &dwxgmac210_dma_ops,
                .mac = &dwxgmac210_ops,
                //....
        },

    //....
}


const struct stmmac_ops dwxgmac210_ops = {
    //....
    .config_cbs = dwxgmac2_config_cbs,
    .dump_regs = dwxgmac2_dump_regs,
    //....
};

File: - dwxgmac2_core.c
static void dwxgmac2_config_cbs(struct mac_device_info *hw,
                                u32 send_slope, u32 idle_slope,
                                u32 high_credit, u32 low_credit, u32 queue)
{
        void __iomem *ioaddr = hw->pcsr;
        u32 value;

        if (queue >= hw->num_tc)
                return;

        writel(send_slope, ioaddr + XGMAC_MTL_TCx_SENDSLOPE(queue));
        writel(idle_slope, ioaddr + XGMAC_MTL_TCx_QUANTUM_WEIGHT(queue));
        writel(high_credit, ioaddr + XGMAC_MTL_TCx_HICREDIT(queue));
        writel(low_credit, ioaddr + XGMAC_MTL_TCx_LOCREDIT(queue));

        value = readl(ioaddr + XGMAC_MTL_TCx_ETS_CONTROL(queue));
        value &= ~XGMAC_TSA;
        value |= XGMAC_CC | XGMAC_CBS;
        writel(value, ioaddr + XGMAC_MTL_TCx_ETS_CONTROL(queue));
}
```

IDCEvo AVB Bandwidth Requirements

There is two Queues allocated for AVB traffic viz. Queue 5 and Queue 6.

Queue 5: -

Queue 5 carries video traffic of AVB Class B.

Maximum Data rate in class B = 1536 bytes/packet x 4000 packest/seconds = 6144000 bytes = 6144000 x 8 = 49152000 bps = 48 Mbps

Since there are 4 video Tx stream, so total data rate = 4 x 48 = 192 Mbps.

For portTransmitRate = 1Gbps: -

Bandwidth allocated to AVB in Queue 5 = 192 Mbps or 19.2% of portTransmitRate.

Therefore, **idleSlope = 192 bits** (to verify)

sendSlope = (idleSlope – portTransmitRate) = 192 - 1000 = **-808 bits** (to verify)

loCredit: -

link speed = 1000 Mb/s and a traffic class B with 19.2% of bandwidth reservation.

For traffic class B, the maxFrameSize will be number of bits that can be transmitted during 250 s over a 1000 Mb/s link. Which is:

1000 000 000 b/s x 250 us x 0.192 = 48000 bits. Also, this number is integral number of octets.

Formula: -

loCredit = maxFrameSize x (sendSlope/portTransmitRate)

loCredit = 48000 x (-808/1000) = -38784 bits = 0xFFF81CC0

(As, 38784 = 0x9780. So, -33600 = 2's complement of 0x00009780 = 0xFFFF6880)

loCredit = 0xFFFF6880

hiCredit: -

Formula: hiCredit = maxInterferenceSize x (idleSlope/portTransmitRate)

maxInterferenceSize is simply the size of one maximum sized Ethernet frame, which is 1536 octets or 1600 bytes, meaning 1600 x 8 bits = 12800 bits. In the case of a 1000 Mb/s Ethernet

connection and a traffic class (queue) with 19.2% bandwidth reservation,

hiCredit = 12800 bits x (192 Mb/s / 1000 Mb/s) = 2458 bits.

hiCredit = 2458 bits

Queue 6: -

Queue 6 carries audio traffic of AVB Class C.

Maximum data rate in Queue 6 = 25Mbps or 2.5 % of 1000 Mbps bandwidth

For class C, measurement interval is 1.3333 ms

CBS parameter calculation suggested by Samsung

Formula:

idleSlope = (bitrate(bps)/10^9)*B*S
sendSlope = B*S-idleSlope

hiCredit = F*N*(idleSlope/B)
loCredit = 2's complement of (F_max*S*(sendSlope/B))

Here, Calculation interval: 40ns(100Mbps), 8ns(1Gbps)
- Bit per calculation interval (B): 4bit(100Mbps), 8bit(1Gbps, 2.5Gbps), 32bit(5Gbps, 10Gbps)
- Scale value (S): 1024
- Max. frame size of stream by bit (F): 1500*8
- Max. frame size of system by bit (Fmax): 1500*8
- Max. number of burst frame (N)

CBS parameter calculation for 100Mbps (BMW RSE/CDE) variants

For Queue-5 having bandwidth of 192Mbps

Currently, RSE and CDE variant does not support video use case, so Queue-05 has been set to support DCB

algorithm.

For Queue-6 having bandwidth of 25Mbps

idle_slope = (25/100) * 4 * 1024 = 1024 = 0x400

send_slope = 4*1024 - 1024 = 3072 = 0xC00

hi_credit = 1500*8*(1024/4) = 3,072,000 = 0x2ee000

lo_credit = 2's complement of (1500*8*1024*(3072/4)) = 2's comp of (9,563,136,000) =
= 2's comp of (0x232800000)
= 0xFDCD800000
= 0xCD800000

CBS parameter calculation for RGMII (1Gbps)

For Queue-5 having bandwidth of 192Mbps

N=3
B=8

S = 1024

F_max =1500*8 bits

idle_slope = ((192*10^6) / 10^9) * 8 * 1024 = 0.192*8*1024 = 0x625

send_slope = 8*1024 - 0.192*8*1024 = (1-0.192)*8*1024 = 0x19db

high_credit = 1500*8 *3 * (0.192 * 8 * 1024 / 8) = 0x6c0000

lo_credit = 2's complement of 1500 * 8 * 1024 * (1 - 0.192) * 8 * 1024 / 8

= 2's complement of 10,166,992,896 = 0xa2000000 //Take only 4 bytes

Note: - While verifying set bandwidth of 192Mbps using IPerf, I have tweaked CBS param. for idle_slope from 0x625 (theoretical value)

to 0x640.

For Queue-6 having bandwidth of 25Mbps

idle_slope = 0.025 * 8 * 1024 = 205 = 0xcd

send_slope = 8*1024 - 0.025*8*1024 = 7987 = 0x1f33

high_credit = 1500*8*3*(0.025*8*1024 / 8) = 0xe1000

lo_credit = 2's complement of 1500 * 8 * 1024 * (1 - 0.024) * 8 * 1024 / 8

= 2's complement of 12,268,339,200 or 0x02 db40 0000

= 0xfd22c00000 = 0x22c00000 //Take only 4 bytes

Note on tweak/fine tuning: - While verifying set bandwidth of 25Mbps using IPerf, I have tweaked CBS param. for **idle_slope** from 0xcd(theoretical value) to 0xd0.

CBS parameter calculation for 10Gbps USXGMII

For Queue-5 having video bandwidth of 192 Mbps: -

BW fraction = 0.192Gbps / 10 Gbps = 0.0192

idle_slope = BW fraction * B * S = 0.0192*32*1024 = 629.15 = 0x275

send_slope = B*S - idle_slope = 32*1024 - 629.15 = 32,138.85 = 0x7D8A

hi_credit = 0x800000

lo_credit = 0x18000000

For Queue-6 having video bandwidth of 25 Mbps: -

BW fraction = 0.025Gbps / 10 Gbps = 0.0025

idle_slope = BW fraction * B * S = 0.0025*32*1024 = 82 = 0x52

send_slope = B*S - idle_slope = 32*1024 - 82 = 32686 = 0x7FAE

hi_credit = 0x800000

lo_credit = 0x18000000

Note : -

As per Samsung, In case of 10Gbps USXGMII, just set fix max value for hiCredit and loCredit, max hiCredit is 0x8000000, max loCredit is 0x18000000. And test result also shows no issue with these values.

Qdisc settings to route traffic to a Queue

I. Kernel config macros needed to enable qdisc to work with "tc" command: -

```
CONFIG_NET_SCHED=y
CONFIG_NET_SCH_MULTIQ=y
CONFIG_NET_SCH_MQPRIO=y
CONFIG_NET_CLS_U32=y
CONFIG_NET_CLS_FLOW=y
CONFIG_NET_CLS_FLOWER=y
CONFIG_NET_CLS_ACT=y
CONFIG_NET_ACT_SKBEDIT=y
```

II. "tc" command to route an IP address traffic to a Queue: -

```
# ifconfig eth0 192.168.2.1

# tc qdisc add dev eth0 root handle 1: multiq

# tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip src 192.168.2.1 action skbedit queue_mapping 3

# tc -s filter show dev eth0

O/p -
filter parent 1: protocol ip pref 1 u32 chain 0
filter parent 1: protocol ip pref 1 u32 chain 0 fh 800: ht divisor 1
filter parent 1: protocol ip pref 1 u32 chain 0 fh 800::800 order 2048 key ht 800 bkt 0 terminal flowid ??? not_in_hw
  match c0a80201/ffffffff at 12
        action order 1:  skbedit queue_mapping 3 pipe
         index 1 ref 1 bind 1 installed 21 sec used 21 sec
        Action statistics:
        Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
        backlog 0b 0p requeues 0



{ Note - c0a80201 is hex code of IP address 192.168.2.1
```

In above command, we have bound the IP address "192.168.2.1" to Queue - 03.

Hardware register read verification for CBS settings for different Queues

GMAC0 Base Address = 0x16CC 8000

MTL Offset = 0x1000

Hardware address of ith "MTL_TC(#i)_SendSlopeCredit" register = 0x16CC9000 + (0x111C + 0x80*i),

where i = 1 to 16.

Queue	Physical address of Hardware Register : MTL_TC(#i)_SendSlopeCredit
Queue - 1	0x16CC A19C
Queue - 2	0x16CC A21C
Queue - 3	0x16CC A29C
Queue - 4	0x16CC A31C
Queue - 5	0x16CC A39C
Queue - 6	0x16CC A41C

Verify Hw register as set using DTS: -

Run devmem command on Node0

devmem2 0x16cca29c

O/p -
/dev/mem opened.
Memory mapped at address 0x7f8a492000.
Read at address 0x16CCA29C (0x7f8a49229c): 0x00001F3C

(Here, 0x1f3c is sendSlope value set for Queue-3.)

Testing CBS Traffic Shaping

Method 1: By keeping only single DMA channel to route all data to a Queue

Step I. Keeping only single DMA channel for Android

File: - and_ivi/exynosautov920-evt0-idcevo-sp25-phy-eth.dtsi

Code change: -

```
&gmac0 {
        status = "okay";
        mac-address = [38455420d007];
-       snps,ndev-dma-ch-map = <3 8 9 10 11>;
+        //snps,ndev-dma-ch-map = <3 8 9 10 11>;
+        snps,ndev-dma-ch-map = <3>;
        sxgmac,hpt;
        sxgmac,hpt-fe;
        sxgmac,vlink-compatible = "vsxgmac1";
```

Step II. Modify CSB parameter for TX Queue-03 in SYS/Node0 dts

File: - arch/arm64/boot/dts/exynos/exynosautov920-eth.dtsi

```
queue3 {
-                   snps,weight = <0x1>;
+                    /*snps,weight = <0x1>;
                     snps,dcb-algorithm;
-                    snps,priority = <0x08>;
+                    snps,priority = <0x08>;*/
+                    snps,avb-algorithm;
+                    snps,send_slope = <0x7d8a>;
+                    snps,idle_slope = <0x275>;
+                    snps,high_credit = <0x800000>;
+                    snps,low_credit = <0x18000000>;
};
```

Step III. Verifying DMA used as set above in Android

File: - drivers/net/ethernet/stmicro/stmmac/stmmac_main.c

Code change: -

```
--- a/drivers/net/ethernet/stmicro/stmmac/stmmac_main.c
+++ b/drivers/net/ethernet/stmicro/stmmac/stmmac_main.c
@@ -4703,6 +4703,8 @@ static netdev_tx_t stmmac_xmit(struct sk_buff *skb, struct net_device *dev)

        stmmac_set_tx_owner(priv, first);

+        pr_info("CBS_DBG: end of seqos_xmit- tx_packet(%d) DMA_CH(%d)\n", skb->len, queue);
+
        netdev_tx_sent_queue(netdev_get_tx_queue(dev, queue), skb->len);

        stmmac_enable_dma_transmission(priv, priv->ioaddr);
```

<u>Verification: -</u>

Ping PC from target

ping -s 200 <IP addr. of Desktop PC>

Ex. - ping -s 200 192.168.2.5

Watch kernel log

dmesg | grep CSB_DBG

O/p-

```
console:/ # dmesg | grep CBS
[  164.041887] CBS_DBG: end of seqos_xmit- tx_packet(242) DMA_CH(0)
[  165.043095] CBS_DBG: end of seqos_xmit- tx_packet(242) DMA_CH(0)
[  166.044152] CBS_DBG: end of seqos_xmit- tx_packet(242) DMA_CH(0)
[  167.045290] CBS_DBG: end of seqos_xmit- tx_packet(242) DMA_CH(0)
[  168.046514] CBS_DBG: end of seqos_xmit- tx_packet(242) DMA_CH(0)
[  169.047736] CBS_DBG: end of seqos_xmit- tx_packet(242) DMA_CH(0)
```

Note: Here in kernel log, DMA_CH<0> corresponds to DMA channel number - 03 , as set in DTS.

Step III. Iperf test

On Android console: -

```
## iperf on target's console working as server: -

export TMPDIR=/data/local/tmp
export TMPDIR=/data/local/tmp
export TEMP=/data/local/tmp
cd /data/local/tmp

iperf3 -s

O/p -

-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.2.6, port 57622
[  5] local 192.168.2.2 port 5201 connected to 192.168.2.6 port 57623
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  21.3 MBytes   178 Mbits/sec  105   98.4 KBytes
[  5]   1.00-2.00   sec  21.0 MBytes   176 Mbits/sec   17    137 KBytes
[  5]   2.00-3.00   sec  21.3 MBytes   179 Mbits/sec   48   91.2 KBytes
[  5]   3.00-4.00   sec  21.3 MBytes   179 Mbits/sec   58    103 KBytes
[  5]   4.00-5.00   sec  20.3 MBytes   170 Mbits/sec   15    128 KBytes
[  5]   5.00-6.00   sec  20.6 MBytes   173 Mbits/sec   60    137 KBytes
[  5]   6.00-7.00   sec  20.6 MBytes   173 Mbits/sec   92    101 KBytes
[  5]   7.00-8.00   sec  21.7 MBytes   182 Mbits/sec    2    140 KBytes
[  5]   8.00-9.00   sec  21.0 MBytes   176 Mbits/sec    0    153 KBytes
[  5]   9.00-10.00  sec  21.3 MBytes   179 Mbits/sec   18    141 KBytes
[  5]  10.00-10.04  sec   753 KBytes   142 Mbits/sec    0    141 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.04  sec   211 MBytes   176 Mbits/sec  415             sender
```

On Desktop PC connected as iperf client: -

```
## iperf testing on Desktop PC

iperf3.exe -c 192.168.2.2 -R
Connecting to host 192.168.2.2, port 5201
Reverse mode, remote host 192.168.2.2 is sending
[  4] local 192.168.2.6 port 57623 connected to 192.168.2.2 port 5201
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-1.00   sec  21.3 MBytes   178 Mbits/sec
[  4]   1.00-2.00   sec  21.2 MBytes   178 Mbits/sec
[  4]   2.00-3.00   sec  21.1 MBytes   177 Mbits/sec
[  4]   3.00-4.00   sec  21.3 MBytes   179 Mbits/sec
[  4]   4.00-5.01   sec  20.0 MBytes   167 Mbits/sec
[  4]   5.01-6.00   sec  20.8 MBytes   175 Mbits/sec
[  4]   6.00-7.00   sec  20.6 MBytes   173 Mbits/sec
[  4]   7.00-8.00   sec  21.5 MBytes   181 Mbits/sec
[  4]   8.00-9.00   sec  21.1 MBytes   177 Mbits/sec
[  4]   9.00-10.00  sec  21.3 MBytes   179 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec   211 MBytes   177 Mbits/sec  415             sender
[  4]   0.00-10.00  sec   211 MBytes   177 Mbits/sec                  receiver

iperf Done.
```

Method 2 : Using "tc" command to route all data to a Queue

Step I. Using "tc" command bind an IP address to a Queue

(Do not disble DMA channels as in Method - 1, Step - I, shown above).

```
# ifconfig eth0 192.168.2.2 up
# tc qdisc add dev eth0 root handle 1: multiq
# tc filter add dev eth0 parent 1: protocol ip prio 1 u32 match ip src 192.168.2.2 action skbedit queue_mapping 0
# tc -s filter show dev eth0
```

Note: In above command, 0 highlighted with violet color is Queue number. Its mapping with DTS's Queue number and EMAC driver's Queue number is shown below.

Queue number associated Android, as per DTS	Queue number interpreted by EMAC driver
Queue - 3	0
Queue - 8	1
Queue - 9	2
Queue - 10	3
Queue - 11	4

Step II. Modify CSB parameter for TX Queue-03 in SYS/Node0 dts

Same as shown in Method - I, Step II.

Step III. Iperf test

(Same as Method - 1, Step - III, mentioned above.)

Verification:

```
# dmesg | grep CBS

[  180.965954] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
[  180.982390] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
[  180.999596] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
[  181.021361] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
[  181.056783] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
[  181.068963] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
[  181.084153] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
[  181.118585] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
[  181.137811] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
[  181.158075] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
[  181.174544] CBS_DBG: end of seqos_xmit- tx_packet(1514) DMA_CH(0)
```

Debug patch used (for reference): -

https://androidhub.harman.com/c/Android/kernel/+/474333

https://androidhub.harman.com/c/and_ivi_dts-idcevo-la/+/474475

https://gerrit1.harman.com/c/GTC_Foundation_Software/Development/Kernel/BSP/Linux/Samsung_KITT_EA/+/308511

Method 3 : Verify Hardware register setting using Devmem uility

Step I. Run "devmem" tool to verify CBS register settings for Queue-06 as below.

Hardware register Name	Hardware register address for Queue – 06 related to CBS	Values to write/read
idleSlope Queue - 06	0x16cca418	0xd0
sendSlope Queue - 06	0x16cca41c	0x1f3c
hiCredit Queue - 06	0x16cca420	0xd8000
loCredit Queue - 06	0x16cca424	0x04000000

Note: -

Method - 03 only verifies Hardware register settings for CBS parameters done by DTS change. For actual functionalities Method-1 or Method-2 should be followed.

Network and AVB DMA/Queue assignment for Node0-Linux and Android

For Node0-Linux DTS

```
&gmac0 {
   //...
   snps,ndev-dma-ch-map = <0 12 13 14 15>;
   snps,avb-dma-ch-map = <4 5>;
   //...
   sxgmac,dsp-ch-map = <1 6 7>;
   //...
};
```

Network queues assigned to VM2:
0, 12, 13, 14, 15

Network Queues used by AVB(set in VM2 dts): -
4, 5

Network Queues used by DSP(set in VM2 dts): -
1, 6, 7

AVB Queue 5 is used for Video @192Mbps. But Queue-4 is unused.

DSP Queue 6 is used for Audio @25Mbps. Queue - 1 is unused and Queue-7 is for AVB PTP.

For Android DTS

```
&gmac0 {
   //...
   snps,ndev-dma-ch-map = <3 8 9 10 11>;
   //...
};
```

Network queues assigned to VM3:
    3, 8, 9, 10, 11

Change Tx Queue to follow AVB or DCB algorithm

Gerrit: -

https://gerrit1.harman.com/c/GTC_Foundation_Software/Integration/Elina/Project/Samsung_KITT/automotive/EA/linux_sys_dts-idcevo-la/+/304587

I. Verify CBS in RSE/CDE variant: -

Important code/dts change: -

```
&mtl_tx_setup {
   queue5 {
      snps,weight = <0x1>;
      snps,dcb-algorithm;
      snps,priority = <0x20>;
   };
   queue6 {
      snps,send_slope = <0xc00>;
      snps,idle_slope = <0x400>;
      snps,high_credit = <0x2ee000>;
      snps,low_credit = <0xcd800000>;
   };
};
```

Patch used for Debug print for verification: -

```
--- a/drivers/net/ethernet/stmicro/stmmac/stmmac_main.c
+++ b/drivers/net/ethernet/stmicro/stmmac/stmmac_main.c
@@ -3162,8 +3162,12 @@ static void stmmac_configure_cbs(struct stmmac_priv *priv)
        /* queue 0 is reserved for legacy traffic */
        for (queue = 1; queue < tx_queues_count; queue++) {
                mode_to_use = priv->plat->tx_queues_cfg[queue].mode_to_use;
-               if (mode_to_use == MTL_QUEUE_DCB)
+               if (mode_to_use == MTL_QUEUE_DCB) {
+                       pr_alert("CBS_DBG: func - %s, Queue - %d is DCB\n", __func__, queue);
                        continue;
+               }
+
+               pr_alert("CBS_DBG: Func - %s, Queue - %d is AVB\n", __func__, queue);

                stmmac_config_cbs(priv, priv->hw,
                                priv->plat->tx_queues_cfg[queue].send_slope,
```

Debug prints observed: -

```
root@idcevo-hv-v920:~# dmesg | grep CBS
[    3.663956]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 1 is DCB
[    3.665564]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 2 is DCB
[    3.666769]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 3 is DCB
[    3.666952]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 4 is DCB
[    3.692125]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 5 is DCB
[    3.692370]SYS[  T408] CBS_DBG: Func - stmmac_configure_cbs, Queue - 6 is AVB
[    3.715838]SYS[  T408] CBS_DBG: func - dwxgmac2_config_cbs, CBS Queue - 6
[    3.716713]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 7 is DCB
[    3.719603]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 8 is DCB
[    3.738552]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 9 is DCB
[    3.738563]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 10 is DCB
[    3.738569]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 11 is DCB
[    3.738574]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 12 is DCB
[    3.738579]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 13 is DCB
[    3.738584]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 14 is DCB
[    3.738587]SYS[  T408] CBS_DBG: func - stmmac_configure_cbs, Queue - 15 is DCB
```

=> From above debug log, we can see that we are able to make/override a Tx Queue in DCB mode using DTBO equivalent DTS file, which was made AVB mode using SOC dts file.

Verification: -

For verification, Queue-03 in Android was set as below.

```
&mtl_tx_setup {
        queue3 {
                snps,send_slope = <0xc00>;
                snps,idle_slope = <0x400>;
                snps,high_credit = <0x2ee000>;
                snps,low_credit = <0xcd800000>;
        };

//...

};
```

Test results on test PC: -

```
C:\Third_Party_Softwares\iperf-3.1.3-win64\iperf-3.1.3-win64>iperf3.exe -c 192.168.2.2 -R
Connecting to host 192.168.2.2, port 5201
Reverse mode, remote host 192.168.2.2 is sending
[  4] local 192.168.2.6 port 58516 connected to 192.168.2.2 port 5201
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-1.00   sec  2.83 MBytes  23.8 Mbits/sec
[  4]   1.00-2.00   sec  2.74 MBytes  23.0 Mbits/sec
[  4]   2.00-3.00   sec  2.74 MBytes  23.0 Mbits/sec
[  4]   3.00-4.00   sec  2.74 MBytes  23.0 Mbits/sec
[  4]   4.00-5.00   sec  2.74 MBytes  23.0 Mbits/sec
[  4]   5.00-6.00   sec  2.74 MBytes  23.0 Mbits/sec
[  4]   6.00-7.00   sec  2.74 MBytes  23.0 Mbits/sec
[  4]   7.00-8.00   sec  2.74 MBytes  23.0 Mbits/sec
[  4]   8.00-9.00   sec  2.74 MBytes  23.0 Mbits/sec
[  4]   9.00-10.00  sec  2.74 MBytes  23.0 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  28.2 MBytes  23.6 Mbits/sec    0             sender
[  4]   0.00-10.00  sec  27.6 MBytes  23.2 Mbits/sec                  receiver

iperf Done.
```

II. Verify CBS in 10Gbps Switch variant: -

Important code/dts change: -

```
&mtl_tx_setup {
    queue5 {
        snps,send_slope = <0x7d8a>;
        snps,idle_slope = <0x275>;
        snps,high_credit = <0x800000>;
        snps,low_credit = <0x18000000>;
    };
    queue6 {
        snps,send_slope = <0x7fae>;
        snps,idle_slope = <0x52>;
        snps,high_credit = <0x800000>;
        snps,low_credit = <0x18000000>;
    };
};
```

For verification Queue-03 in Android DTS setting was changed with CBS param for Queue-05 and Queue-06 respectively for 192Mbps and 25Mbps bandwidth allocation

one by one.

Test results are shown here.

CBS param verification for 192 Mbps in 10Gbps EMAC variant: -

On target: -

```
# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.2.6, port 54398
[  5] local 192.168.2.2 port 5201 connected to 192.168.2.6 port 54399
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  22.1 MBytes   185 Mbits/sec    0    212 KBytes
[  5]   1.00-2.00   sec  21.9 MBytes   183 Mbits/sec    0    212 KBytes
[  5]   2.00-3.00   sec  21.9 MBytes   183 Mbits/sec    0    212 KBytes
[  5]   3.00-4.00   sec  21.4 MBytes   180 Mbits/sec    0    212 KBytes
[  5]   4.00-5.00   sec  21.9 MBytes   183 Mbits/sec    0    212 KBytes
[  5]   5.00-6.00   sec  21.4 MBytes   180 Mbits/sec    0    212 KBytes
[  5]   6.00-7.00   sec  21.9 MBytes   183 Mbits/sec    0    212 KBytes
[  5]   7.00-8.00   sec  21.9 MBytes   183 Mbits/sec    0    212 KBytes
[  5]   8.00-9.00   sec  21.4 MBytes   180 Mbits/sec    0    212 KBytes
[  5]   9.00-10.00  sec  21.9 MBytes   183 Mbits/sec    0    212 KBytes
[  5]  10.00-10.04  sec   878 KBytes   188 Mbits/sec    0    212 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.04  sec   218 MBytes   183 Mbits/sec    0             sender
```

CBS param verification for 25 Mbps in 10Gbps EMAC variant: -

```
# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------

Accepted connection from 192.168.2.6, port 55778
[  5] local 192.168.2.2 port 5201 connected to 192.168.2.6 port 55779
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  3.57 MBytes  29.9 Mbits/sec    0   34.2 KBytes
[  5]   1.00-2.00   sec  2.94 MBytes  24.7 Mbits/sec    0   34.2 KBytes
[  5]   2.00-3.00   sec  2.94 MBytes  24.7 Mbits/sec    0   34.2 KBytes
[  5]   3.00-4.00   sec  2.94 MBytes  24.7 Mbits/sec    0   44.2 KBytes
[  5]   4.00-5.00   sec  2.57 MBytes  21.6 Mbits/sec    0   44.2 KBytes
[  5]   5.00-6.00   sec  2.94 MBytes  24.7 Mbits/sec    0   44.2 KBytes
[  5]   6.00-7.00   sec  2.94 MBytes  24.7 Mbits/sec    0   44.2 KBytes
[  5]   7.00-8.00   sec  2.94 MBytes  24.7 Mbits/sec    0   44.2 KBytes
[  5]   8.00-9.00   sec  2.57 MBytes  21.6 Mbits/sec    0   44.2 KBytes
[  5]   9.00-10.00  sec  2.94 MBytes  24.7 Mbits/sec    0   44.2 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.04  sec  29.3 MBytes  24.5 Mbits/sec    0             sender
```

Gerrits Pushed

Sl. No.	Gerrit Link	Description
1.	https://gerrit1.harman.com/c/GTC_Foundation_Software/Development/Kernel/BSP/Linux/Samsung_KITT_EA/+/283959	Repo - Samsung_KITT_EA (Node 0 Linux)
2.	https://gerrit1.harman.com/c/GTC_Foundation_Software/Integration/Elina/Project/Samsung_KITT/automotive/EA/linux_sys_dts-idcevo-la/+/304587	Repo - linux_sys_dts-idcevo-la (Node 0 DTS)

Reference and related documents

Sl. No.	Location	Abstract
1.	https://www.duo.uio.no/bitstream/handle/10852/87168/5/Analysis_of_bandwidth_reservation_strategy_for_AVB_classes_in_Ethernet_TSN_switch.pdf	General CBS algorithm and formula
2.	IDCEVO-EthernetB...24-1344-1778.pdf	IDCEvo Bandwicth Estimation for AVB
3.	EMAC TX DMA channel selection with qdisc	Taken reference from AUDI HCP3
4.	https://elvis.harman.com/cgi-bin/ticket?TID=3572310	Ticket to Samsung