Keystone I 以太网子系统调试手册

Marvin Liang *China MCP team*

摘要

Keystone I 系列芯片(C6670，C6671，C6672，C6674，C6678， TCI6614)的以太网子系统可以划分为三个功能实体：内部 Switch（SGMII Serdes/SGMII/EMAC/CPSW）， PA(包加速器)，SA(安全加速器)。该子系统因为包含的子模块多，相对较为复杂，相关的四本用户手册内容组织并不是很容易理解，不少用户在开发过程中都遇到了一些技术难题。本手册总结实际应用中遇到的部分典型的技术问题和各种问题的调试方法，同时也提供以太网子系统的各种实用配置和建议。对于具备基本 Keystone I 以太网知识的用户，本手册可以作为用户手册的有力补充。 期望读者在初步阅读 user guide 和具备基本的 PA LLD/SA LLD 开发经验以后再查阅本文。

Document History

Version	Date	Author	Notes
1.0	May 2013	Marvin Liang	First release

目录

图

表

参考文献

1. *KeyStone Architecture Network Coprocessor (NETCP) User Guide* (SPRUGZ6)
2. *KeyStone Architecture Gigabit Ethernet (GbE) Switch Subsystem User Guide (*SPRUGV9A*)*
3. *KeyStone Architecture Packet Accelerator(PA) User Guide (*SPRUGS4*)*
4. *KeyStone Architecture Security Accelerator(SA) User Guide (*SPRUGY6*)*

1. 以太网子系统说明

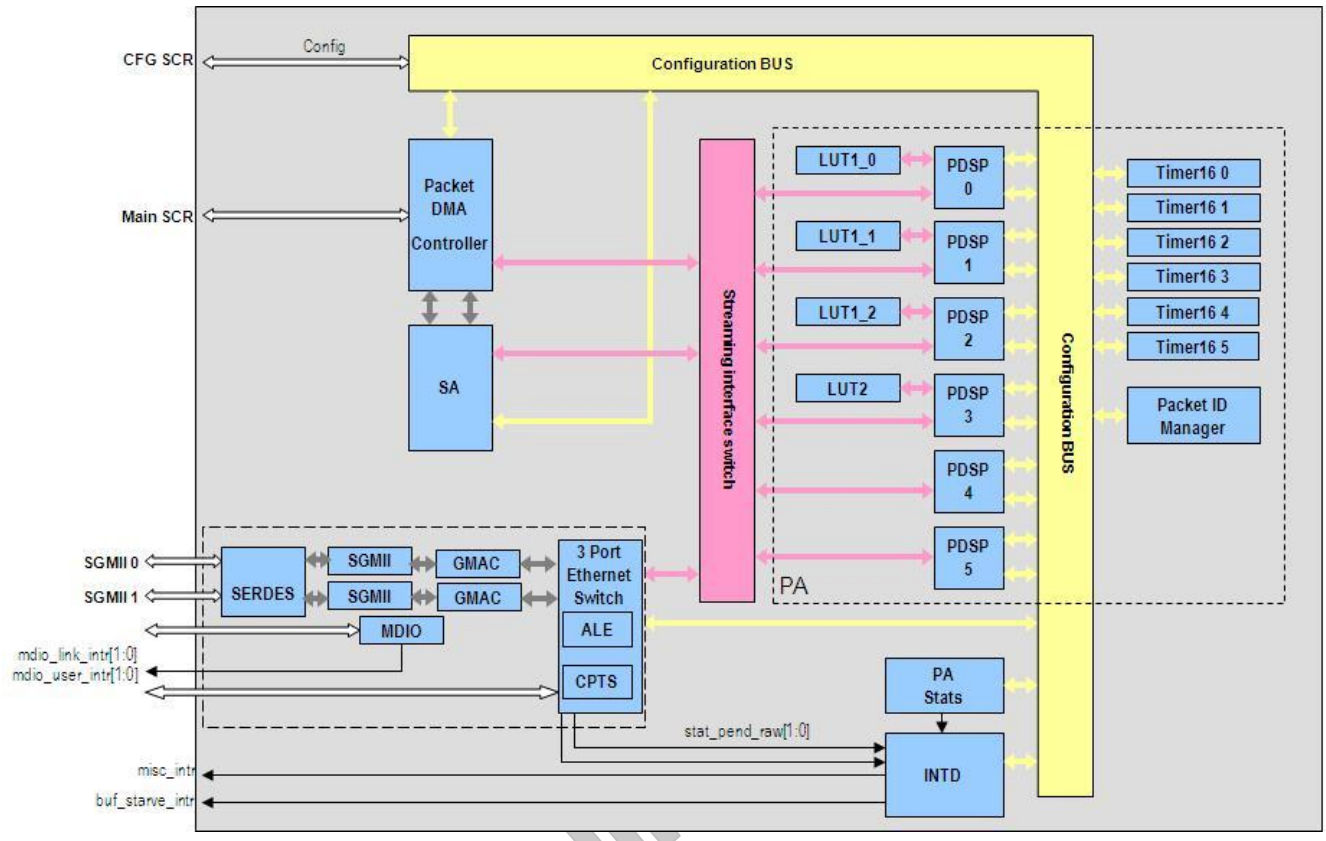

图 1　以太网子系统框图

在用户手册中，以太网子系统可以称为网络协处理器子系统(NETCP)，其主要构成模块有：

1. 以太网交换部分
   主要模块有 Serdes，SGMII， EMAC，三端口交换机等
2. 包加速器
   主要模块有 6 个 PDSP，3 个一级的查找表，1 个二级的查找表，对应的 Packet DMA 等
3. 安全加速器
   主要模块有 2 个 PDSP，加解密和鉴权引擎等

注意：

以太网子系统的整个初始化过程，一般推荐采用从内到外的次序：

PA PAKET DMA->PA PDSP->3 port switch/CPSW->EMAC->SGMII->SGMII Serdes

2. SGMII

2.1 SGMII SerDes配置

Serdes 的配置主要与供给的 Serdes 时钟相关，下面给出一个对应不同时钟频率的配置示例：
312.5MHz-> SGMII PLL Configuration Register 0x41，
250MHz-> SGMII PLL Configuration Register 0x51，
156.25MHz-> SGMII PLL Configuration Register 0x81
配置完毕以后，请注意检查 SGMII SerDes Status Register (SGMII_SERDES_STS)的最低 Lock 位是否为 1。

2.2 SGMII 与PHY的连接

当 SGMII 与 PHY 连接的时候，SGMII 一定是 Slave 模式，一般情况下同时使能自协商。

2.3 SGMII 与Switch的连接

对于支持 SGMII 标准的外部 switch 与 Keystone I 芯片连接的时候，可以一端配置为 master，另一端配置为 slave，同时使能自协商模式。但是对于只支持 1000BASE-X 的 switch 与 Keystone I 芯片连接的时候，应该将两端都设置为 master 模式，并禁止自协商，强制为 1000M 全双工。

2.4 MDIO接口与PHY的控制

通过 MDIO 控制寄存器 MDIO_CONTROL 可以对外接的 PHY 进行控制，按照用户手册的建议配置到 2.5MHz。需要注意的是：

1. MDIO 模块初始化后，应该先通过 MDIO 读取 PHY 的 ID 寄存器，确认该连接是否正常。
2. 读取 MDIO 模块的 PHY Alive Status Register (ALIVE)获取 PHY ID (注意 PHY ID 是 0~31)
3. 将第二步读到的 PHY ID 作为 PHY address 写到 MDIO User Access Register 0 (USERACCESS0)或者 MDIO User Access Register 1 (USERACCESS1)中访问对应的 PHY 寄存器

2.5 1000M半双工的问题说明

Keystone I 不支持 1000M 半双工的应用，当外接的 PHY 或者是 switch 支持 1000M 半双工时，可能会对 keystone I 的网络功能造成影响。特别是当某些 PHY 在复位过程中，可能会发起 1000M 半双工的请求，这种情况下，请在系统上电初始化时先通过 MDIO 接口关闭 PHY 的 1000M 半双工协商能力。

3. EMAC

3.1 VLAN aware模式和VLAN unaware模式

1. VLAN unaware 模式下所有的包不会被改变
2. VLAN aware 模式的配置步骤：
   A. 配置 GbE switch Control 寄存器(CPSW_CONTROL)的 VLAN_AWARE 比特
   B. 配置 ALE Control 寄存器(ALE_CONTROL)的 ALE_VLAN_AWARE 比特
   C. 配置 P0_PORT_VLAN， P1_PORT_VLAN 和 P2_PORT_VLAN 寄存器

3. VLAN aware 模式对不同包的不同策略:
   A. 对于 untagged 包，插入端口所配置的 VLAN 头
   B. 对于 priority tagged 包，根据 CPSW_CONTROL 中的

P0_PASS_PRI_TAGGED/P1_PASS_PRI_TAGGED/P2_PASS_PRI_TAGGED 配置，如果该位为 0(default)，则该端口上收到的 priority tagged 包中的 VLAN ID 会被替换为该端口配置的 VLAN ID。

C. 对于 VLAN tagged 包，包中的 VLAN 优先级会被替换为该端口配置的 VLAN 优先级，VLAN ID 不变。

3.2 EMAC的复位

EMAC 模块提供了一个软复位的寄存器，一般情况下在芯片上电复位启动后，不建议使用该寄存器对 EMAC 模块做单独的复位操作，该复位操作后可能会引起以太网包收发异常，请谨慎使用。

3.3 MAC地址的说明

对于 Keystone I 的芯片来说，一般可以配置 3 个 MAC 地址。以 C6678 为例，

MACID1 (0x02620110) 和MACID2 (0x02600114)

该 MAC 地址是出厂时，TI 写到芯片的 Efuse 里的，类似于一个芯片 ID，只读不可修改。如果芯片选择以太网 boot 方式，芯片会以该地址发出 bootp 的报文(Ethernet ready announcement)

MAC1_SA_LO （0x2090870）和 MAC1_SA_HI （0x2090874）

MAC2_SA_LO （0x20908A0）和 MAC2_SA_HI （0x20908A4）

以上两个 MAC 地址是分别配置给两个 EMAC 端口的，但该地址的作用仅限于使能以太网 Rx 流控以后，封装发到网络上的流控帧，并不按 MAC 地址过滤以太网包。

因为 Keystone 芯片有包加速器（PA），MAC 地址的过滤应该由配置 PA 来完成，所以可以认为 Keystone I 的 EMAC 模块工作在混杂模式（Promiscuous mode）。 换句话讲，所有的报文都会接收并到 PA 进行过滤，从业务层面，对于一颗 Keystone I 的芯片理论上最大可以配置 64 个 MAC 地址(LUT1-0 有 64 个表项)。

4. CPSW

4.1 ALE模块的注意事项

4.1.1 MAC地址的老化问题

因为 ALE 只有 1K 个表项，且硬件没有提供内部的 timer 来自动维护一个老化周期，所以应用程序需要自己维护一个 timer。 Time 第一次超时，应用程序对 ALE_CONTROL 寄存器的 AGE_OUT_NOW 比特置位，会导致所有 ageable 的 ALE Entry 被设置为 untouched； 第二次 timer 超时，再次设置 ALE_CONTROL 寄存器的 AGE_OUT_NOW 比特，在两次超时之间仍保持为 untouched 的表项将会被清除。如果在两次超时之间，有包交换并重新 touched 的表项将会被保留。

4.1.2 ALE Bypass

设置 ALE bypass 只是表明从网络外发到芯片的包，将不会通过查找 ALE 再交换，而是无条件发送到 HOST port。 而从 HOST port 发送到外部的网络的包仍然要通过 ALE 交换。

4.1.3 未知的单播，多播，广播包

1.在某一个端口收到的，ALE 中无法匹配的多播和广播包会被广播到另两个端口。

2. 在 HOST 端口收到的，ALE 中无法匹配的单播包，会被发送到两个 SGMII 端口。

3. 在某 SGMII 端口收到的，ALE 中无法匹配的单播包，只会被发送另一个 SGMII 端口。

4.2 以太网子系统环回配置及其应用

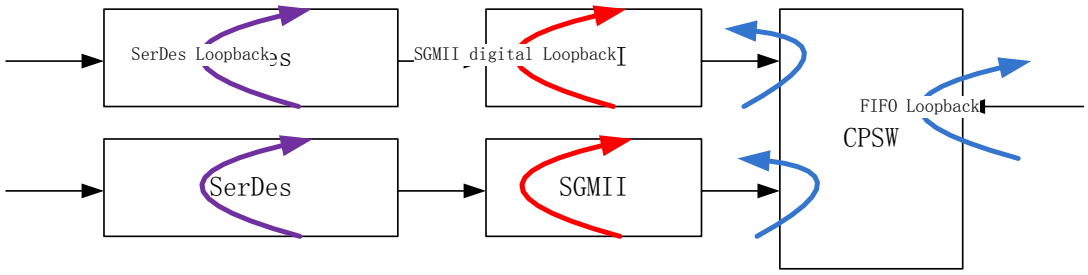

图 2 以太网子系统环回

以太网子系统提供如图的 SGMII Serdes 环回，SGMII digital 环回，CPSW FIFO 环回。注意环回的方向，SerDes 环回，SGMII digital 环回和 CPSW FIFO 环回都有内环的功能，也就是芯片内的 CPU(包括 DSP core， ARM core)向外发的包会被环回到芯片。因为发生环回的节点不一样，所以可以利用这几种环回来定位以太网配置问题。示例：

1. CPSWFIFO 环回成功，而 Serdes 环回和 digital 环回失败，基本可以判定 EMAC/SGMII 配置有问题
2. CPSWFIFO 环回和 digital 环回成功，而 Serdes 环回失败，基本可以判定 SGMII Serdes 配置有问题
3. CPSWFIFO 环回，digital 环回和 CPSW FIFO 环回都成功，说明 CPSW 以下层的配置在基本是正确的

   CPSW FIFO 环回还包含了外环的功能，也就是从外部网络（PHY/Switch）进入芯片的包会被环回到外部网络，芯片内的 CPU core 将无法收到这些包。这个功能可以用以判定 PHY/switch 与 keystone I 的连接配置是否正确。注意 CPSW FIFO 环回只能用于 debug， 使能以后不能动态的去使能，也就是说使能以后若想恢复正常状态，必须对整个 SoC 下电，重新上电复位。

4.3 以太网流控

Rx 流控的意义是 CPSW 检测到 FIFO 超过一定的门限以后，该端口主动向网络外发出以太网流控帧。该流控帧的发送量是 30pps。

Tx 流控的意义是 CPSW 检测到网络对端发出的流控帧，报告给 HOST(DSP/ARM)，由 HOST 软件来裁决减少发送的网络流量。

需要注意的是，如果 Rx FIFO 的占用情况一直满足 Rx 流控的检测条件，Rx flow control 帧会不断的发送到网络上，这种情况多出现在上层软件或者 PA/SA 出现某种异常的场景下。

4.4 CPSW的统计寄存器

1. 寄存器实际都是 R/W 属性，比如当你读到一个寄存器的值为 0x80，回写大于 0x80 的数字，会导致该寄存器的值清 0。 实际对该组寄存器的写操作是减法关系，如果该寄存器正在动态变化中，对它进行写操作并不会导致统计值的丢失。比如当你读到一个寄存器的值为 0x80，回写 0x80 的时刻该寄存器实际的值已经变为 0x90（因为该时刻有包的收发），则此时写 0x80 的结果是该寄存器的值变为 0x90-0x80=0x10。
2. CPSW 的统计寄存器有两组，其中 STATA 是对应于 HOST port， STATB 是两个 SGMII port 的和
3. 对于统计寄存器中的 Rx，Tx 分别都是站在 CPSW 模块的角度统计的结果，换句话说，统计 STATA 中的 Rx 表示的是 HOST port 从 DSP/ARM cores 收进 CPSW 的包，Tx 表示的是从 HOST port 发出

CPSW 到 DSP/ARM core 的包；而统计 STATB 中的 Rx 表示的是两个 SGMII 端口从网络收进来的包，Tx 表示的是从两个 SGMII 端口发送到外部网络的包。

简单的示例：

1) 如果一个包从网络外发送到 Keystone I 芯片，DSP/ARM core 没有收到
   A. 检查 STATB 的 Rx 寄存器
   B. 检查 STATA 的 TX 寄存器
2) 如果一个包从 DSP/ARM core 发送到网络外，网络上没有收到
   C. 检查 STATA 的 Rx 寄存器
   D. 检查 STATB 的 TX 寄存器

4. 一般情况下 RXSOFOVERRUNS/ RXMOFOVERRUNS/ RXDMAOVERRUNS 寄存器都为 0，在某些特定条件下，如以太网子系统 reset 过程中收到包，这些寄存器出现较小的值且不增加是正常的。但如果这些寄存器出现不断增加且收包流量并不大，这种情况一般说明以太网收包异常，通常异常是由于收包的软件模块异常或者 PA 子系统异常造成的。

4.5 802.1P VLAN QoS 的配置

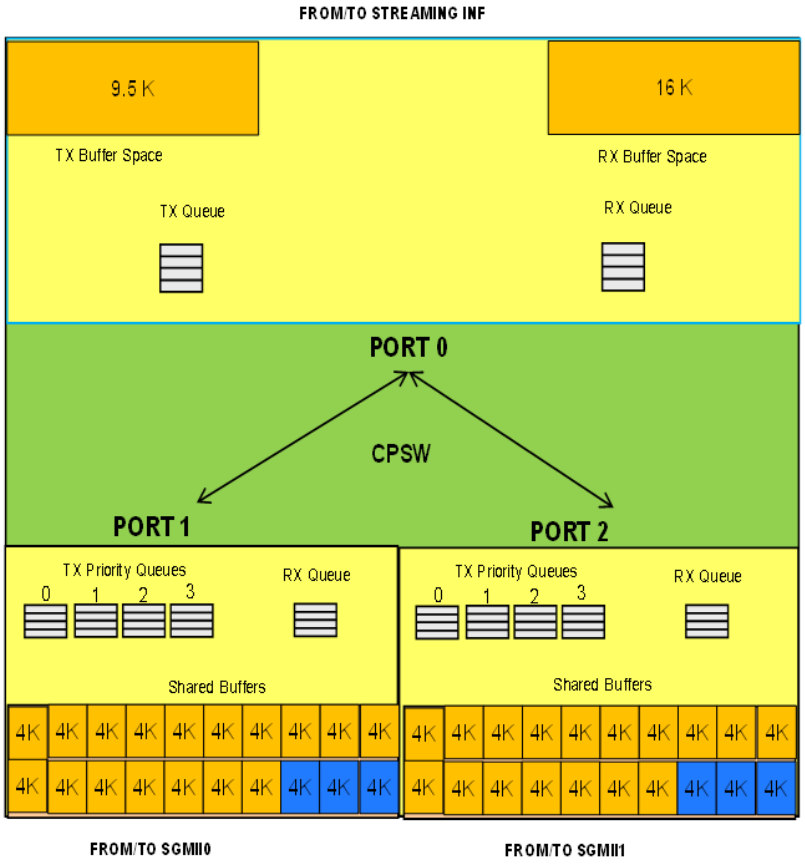

图 3 CPSW FIFO示意图

每个 CPSW EMAC 端口都有收发的 FIFO，收发的总 memory 数为 20 个 4K block。 Rx/Tx 的收发 block 数可以通过 P1_MAX_BLKS 和 P2_MAX_BLKS 寄存器配置。其中 Tx FIFO 分为四个 优先级队列（priority queue），优先级 0 到 3，其中 3 的优先级最高。配置过程为：

1. 配置 packet priority to header packet priority mapping register(P0_RX_PRI_MAP 和两个 EMAC 的 MAC_RX_PRI_MAP)，将实际的 VLAN 包的 8 个优先级映射到芯片内部包头的优先级
2. 配置 header priority to switch priority mapping register (P1_TX_PRI_MAP 和 P2_TX_PRI_MAP)，将内部包头的优先级映射到 4 个优先级队列

注意：

a. 对于untagged VLAN包，映射所要用的VLAN优先级将使用端口的VLAN优先级。在P0_PORT_VLAN，P1_PORT_VLAN和P2_PORT_VLAN寄存器中配置
b. 当只使用0~3四个优先级队列中的部分时，必须从优先级最高的队列3开始分配。比如只使用两个队列，则应该用3和2。
c. 端口0代表HOST port，端口0收到的包都会通过ALE交换后从两个EMAC端口发送出去，所以端口0只需要配置packet priority to header packet priority mapping register，而没有header priority to switch priority mapping register。

下面是三个VLAN映射的示例：

```
   //set port 0(HOST port) default vlan id 1 and priority 7(higest)
   CSL_CPSW_3GF_setPort0VlanReg (1, 0, 7);
   //set SGMII port 0 default vlan id 2 and priority 6
   CSL_CPSW_3GF_setPortVlanReg(0,2,0,6);
   //set SGMII port 1 default vlan id 3 and priority 5
   CSL_CPSW_3GF_setPortVlanReg(1,3,0,5);

   #ifdef _CPSW_WORK_MODE_1_
   //map all the packets which received by port 0 as header priority 7
   hCpsw3gfRegs->P0_RX_PRI_MAP_REG = 0x77777777;
   //map all the packets which received by SGMII port 0 as header priority 6
   hCpgmacSlRegs->SL_PORT[0].PRI_MAP_REG = 0x66666666;
   //map all the packets which received by SGMII port 1 as header priority 5
   hCpgmacSlRegs->SL_PORT[1].PRI_MAP_REG = 0x55555555;

   //SGMII port 0: map the packets which have header priority 7 to switch priority
queue 3, others to switch priority queue 2
   hCpsw3gfRegs->PORT_INFO_GROUP[0].P_TX_PRI_MAP_REG = 0x32222222;
   //SGMII port 1: map the packets which have header priority 7 to switch priority
queue 3, others to switch priority queue 2
   hCpsw3gfRegs->PORT_INFO_GROUP[1].P_TX_PRI_MAP_REG = 0x32222222;
   #endif

   #ifdef _CPSW_WORK_MODE_2_
   //map all the packets which received by port 0 as header priority 7 and 6
   hCpsw3gfRegs->P0_RX_PRI_MAP_REG = 0x77776666;
   //map all the packets which received by SGMII port 0 as header priority 5 and 4
   hCpgmacSlRegs->SL_PORT[0].PRI_MAP_REG = 0x55554444;
   //map all the packets which received by SGMII port 1 as header priority 3 and 2
   hCpgmacSlRegs->SL_PORT[1].PRI_MAP_REG = 0x33332222;

   //SGMII port 0: map the packets which have header priority 7 to switch priority
queue 3,header priority 6 to switch priority queue 2, 5->1;4,3,2,1->0
   hCpsw3gfRegs->PORT_INFO_GROUP[0].P_TX_PRI_MAP_REG = 0x32100000;
   //SGMII port 1: map the packets which have header priority 7 to switch priority
queue 3,header priority 6 to switch priority queue 2, 5->1;4,3,2,1->0
   hCpsw3gfRegs->PORT_INFO_GROUP[1].P_TX_PRI_MAP_REG = 0x32100000;
   #endif

   #ifdef _CPSW_WORK_MODE_3_
   //map all the packets which received by port 0 as header priority 7 6 and 5
   hCpsw3gfRegs->P0_RX_PRI_MAP_REG = 0x77766655;
   //map all the packets which received by SGMII port 0 as header priority 4,3 and 2
   hCpgmacSlRegs->SL_PORT[0].PRI_MAP_REG = 0x44433322;
   //map all the packets which received by SGMII port 1 as header priority 1 and 0
   hCpgmacSlRegs->SL_PORT[1].PRI_MAP_REG = 0x11110000;

   //SGMII port 0: map the packets which have header priority 7 to switch priority
queue 3,header priority 6 to switch priority queue 2, 5->1;4,3,2,1->0
   hCpsw3gfRegs->PORT_INFO_GROUP[0].P_TX_PRI_MAP_REG = 0x32100000;
   //SGMII port 1: map the packets which have header priority 7 to switch priority
queue 3,header priority 6 to switch priority queue 2, 5->1;4,3,2,1->0
   hCpsw3gfRegs->PORT_INFO_GROUP[1].P_TX_PRI_MAP_REG = 0x32100000;
   #endif
```

5. PA

5.1 PA PLL配置

根据 data sheet（参考 *SPRS671D TMS320TCI6614 Data Manual Figure 8-25 PASS PLL Block Diagram*），PA 子系统的时钟可以选择两个来源，一个是 PASS PLLOUT，另一个是从 SYSCLK1 固定的三分频。当芯片的主频配置为 1GHz 时，如果选择 SYSCLK1，则 PA 子系统工作频率是 333MHz，可以选择 PASS PLLOUT 方式，通过 PA PLL 配置子系统工作于 350MHz。 但是对于主频为 1.2GHz 的芯片，只能选择 PASS PLLOUT 方式，通过 PA PLL 配置子系统工作于 350MHz，不能支持 SYSCLK1 三分频到 400MHz。

5.2 PA的Packet DMA模块

1. PA 的 packet DMA 模块有 9 个 Tx 通道和 24 个 Rx channel，注意这些通道在硬件上都有固定的映射关系，请在使能 PA 子系统的时候，使能所有的通道。
2. PA 的 packet DMA 模块有 32 个 Rx flow。 实际应用中经常见到做法是所有用到 Rx Flow/FDQ 的地方都配置同一个 Rx Flow/FDQ，这样的做法虽然最简单，但是一旦出现问题则很容易导致该 FDQ 的描述符迅速耗尽，而且很难定位导致问题的点。
3. 一般来说在 PA 与其他的硬件模块（特别是 SA）数据交互过程都非常的快，而 PA 将分类好的数据交给 HOST queue，因为有软件的处理所以较慢。故而推荐分开灵活配置不同的 Rx Flow(FDQ): 硬件模块交互间的 FDQ 一般包含 16 个描述符/buffer 就足够了，而最后到 HOST queue 的 Rx Flow/FDQ 可以根据实际软件的处理速度来配置更多的个描述符/buffer。
4. 分开配置不同的 Rx Flow/FDQ 带来的另一个好处是，配置到 HOST queue 的 RX flow/FDQ 里的描述符耗尽，也不会影响到硬件的处理。而且因为 FDQ 根据不同的功能和包处理流程做了不同的配置，在定位某些网络处理问题的时候更加容易。比如在某一个 FDQ 里的描述符用尽时，基本可以确定是该级的相关模块处理的问题，更容易缩小问题定位的范围。

下面是两个的配置实例：

IPSec ESP 隧道模式

1. 当 PDSP1 根据 IPSec ESP 的 SPI 将包分类并路由到 SA 的时候，将需要一个 Rx Flow 的配置（在 LUT1-1 的配置），假定配置为 Rx Flow No.1 (FDQ No.1000，包含空闲描述符/buffer 16 个)。
2. 当 SA 对 IPSec ESP 报文解密完毕，并送到 PDSP2 做内层 IP 分类的时候，配置 Rx Flow No.2 (FDQ No.1001，包含空闲描述符/buffer 16 个)。
3. 当 PDSP3 做完 L4/L5 分类以后，将包送到 HOST queue 时配置 Rx Flow No.3 (FDQ No.1002，包含空闲描述符/buffer 8K 个)。

IPSec AH+ESP 隧道模式

1. 当 PDSP1 根据 IPSec AH 的 SPI 将包分类并路由到 SA 的时候，将需要一个 Rx Flow 的配置（在 LUT1-1 的配置），假定配置为 Rx Flow No.1 (FDQ No.1000，包含空闲描述符/buffer 16 个)。
2. 当 SA 对 IPSec AH 报文鉴权校验完毕，并送回 PDSP1 做 IPSec ESP SPI 分类的时候，配置 Rx Flow No.4 (FDQ No.1003，包含空闲描述符/buffer 16 个)。
3. 当 PDSP1 根据 IPSec ESP 的 SPI 将包分类并路由到 SA 的时候，配置 Rx Flow No.1 (FDQ No.1000，包含空闲描述符/buffer 16 个)。

4. 当 SA 对 IPSec ESP 报文解密完毕，并送到 PDSP2 做内层 IP 分类的时候，配置 Rx Flow No.2 (FDQ No.1001，包含空闲描述符/buffer 16 个)。
5. 当 PDSP3 做完 L4/L5 分类以后，将包送到 HOST queue 时配置 Rx Flow No.3 (FDQ No.1002，包含空闲描述符/buffer 8K 个)。

为了更好的定位在开发过程中遇到的网络问题，比如丢包，packet DMA 队列卡死，描述符丢失等等，推荐在应用软件侧做一些描述符的辅助定位手段：

1. 当描述符初始化以后，记录下所有的描述符首指针。
2. 查询/扫描 PA Packet DMA Tx 队列 No. 640~648 中描述符个数
3. 当发现队列 No. 640~648 队列中有包拥塞时（正常情况下该队列中的描述符个数不会超过 64 个），停止包的收发
4. 将在步骤 1 记录的描述符地址与 FDQ 残留的和队列 No. 640~648 队列中拥塞的做比较
5. 发现丢失的描述符，并检查其内容，特别是检查是否有描述符的 packet length 超过其链接的所有分片的 buffer length 总和的情况

5.3 PA Bypass

芯片可以配置为 bypass 整个 PA 子系统。PA bypass 的实质是忽略所有的 PA 模块（PDSP/LUT）的处理，从 SGMII 端口收到的包经过内部的 packet stream switch 直接通过 PA PKTDMA 模块传输到 HOST queue。其配置过程为：

1. 配置 CPSW Configuration Register (CPSW_CFG_REG)为 0x606（参考 sprugz6 Network Coprocessor (NETCP) User Guide）
2. 初始化并配置 PA Packet DMA 模块
3. 配置 PA Packet DMA 的 Rx flow No.22 和 Rx flow No.23，其中 Rx flow No.22 中的 destination queue 代表了从 SGMII0 收到的包所要放到的目的队列，Rx flow No.23 中的 destination queue 代表了从 SGMII1 收到的包所要放到的目的队列。

注意：PA bypass 以后，由于所有的包未经过 PA 子系统的过滤直接进入到 HOST 队列中，流量较大的情况下，很容易造成 FDQ 的描述符耗尽。而且容易造成软件协议栈 loading 过高，容易遭受网络攻击。另由于在芯片设计中，PA 可以直接与 SA 进行数据传输，特别是在 from network 方向上对加密的包进行解密以后在送到 HOST queue，而 PA bypass 以后，对于加密包的解密处理将增加 HOST 侧的 CPU 负荷。

5.4 PA的固件和Low level driver

PA 的固件有三个文件，位于 PDK 安装目录\pdk_C6678_x_x_x_xx\packages\ti\drv\pa\fw (x_x_x_xx 为版本号)，PDSP0~PDSP2 共用一个文件（classify1_bin.c），PDSP3 用一个文件（classify2_bin.c），PDSP4 和 PDSP5 共用一个文件（pam_bin.c）。PA 子系统运行于大端序，所以固件都是大端序。
注意固件版本和 LLD 版本一定要一致。固件的版本号可以从上述 C 文件的数组中读到，第三个 32 bit WORD 就是版本号。

```
const uint32_t c1[] =  {
     0x21007b00,
     0xbabe0001,
     0x01030007,
     0x24505084,
     0x108484c4,
     0x10e4e4e5,
…
```

如上例固件版本号为 1.3.0.7

PA LLD 的版本号可以从 PA LLD 安装目录的 paver.h 中得到(PA_LLD_VERSION_ID 和 PA_LLD_VERSION_STR)。

PA LLD 中的函数 Pa_downloadImage 是下载 PDSP 固件的，注意其中每个 PDSP 的常量表定义 pap_pdsp_const_reg_map，不同的版本这些值可能不同。对于某些有 ARM core 的 Keystone I 芯片，该常量表不是通过 PA LLD 函数来赋值而是 Linux kernel 代码直接赋值的，所以必须跟各个版本的固件对应。如果常量表和固件，LLD 的版本不匹配，可能会有某些功能异常。

5.5 PA LUT1表项增加的策略

LUT1 是线性查找表，也就说，配置的表项从下标 No. 0 一直找到 No.63。 因为不同的表项内容间可能有包含关系，比如某一项 A 只要求匹配目的 IP 地址，另一项 B 配置了同样的目的 IP 地址，同时也配置了源 IP 地址。这样其实是 B 的条件比 A 更加严格，那么严格的表项 B 应该比宽松的 A 下标小。比如配置 A 在 No.1， B 在 No.0， 只有这样 B 才能有被匹配的可能性。

注意：PA LLD 提供的函数如 Pa_addIp，Pa_addMac 等，都提供了不指定下标的方式添加表项，函数的参数为不指定下标的时候，PA LLD 的处理为从下标较大的开始添加。例如当某个 64 表项的表为空，而两次调用 Pa_addIp 添加表项并不指定表项位置的时候，第一次添加的表项在 No.63 的位置，第二次添加的表项在 No.62 位置。

5.6 PA相关的调式

5.6.1 Device simulator 辅助调试

由于 PA 子系统主要由硬件模块组成，提供的寄存器有限，对于复杂的功能来说，调试相对比较困难。但是 TI 的芯片 simulator 上对该子系统进行了充分的支持，在初期开发阶段，simulator 上调试 PA 更加方便。

下面是一个在 TCI6608 的 simulator 上使能 PA 调试和 Log 输出的例子：

1. CCS 安装路径\simulation_csp_ny\bin\configurations\tisim_tci6608_pv.cfg
   如果该行被注释，请解注释 INPUT5 log_file, pass.log;
2. 在软件初始化中加入下面的代码

```
typedef struct paLog_s
{
    /* PA trace levels */
    UInt paLut1TraceLevel;
    UInt paLut2TraceLevel;
    UInt paCdeTraceLevel;
    UInt rsvd1[5];

    /* System trace levels */
    UInt sysStreamTraceLevel;
    UInt sysMemIfTraceLevel;
    UInt rsvd2[6];
    /* SA trace levels */
    UInt rsvd3[8];
    /* Unused */
    UInt rsvd4[40];
} paLog_t;

#define PA_LOG_IF               0x02000100
#define PA_LOG_LEVEL_DISABLE    256

void enablePaSimLogging(void)
{
  paLog_t *plt = (paLog_t *)PA_LOG_IF;
  plt->paLut1TraceLevel = 0;
  plt->paLut2TraceLevel = 0;
  plt->paCdeTraceLevel  = 0;
  plt->sysStreamTraceLevel = 2;
  plt->sysMemIfTraceLevel  = 2;
}
```


3. 上两步的初始化过程将在 simulator 的安装路径(如 CCS 安装路径 \simulation_csp_ny\env\ccs\drivers\pass.log)下产生名为 Pass.log 的详细 log 文件 该文件包含 LUT 表项的添加过程，LUT 表项每一级的匹配过程等等详细的内容。

注意这个方法非常适合诊断某些包未按照预想的规则匹配/路由的情况，软件只需要将测试包发到 640 队列（忽略以太网子系统其他的模块的初始化过程）就可以开始测试。但是对于发包量比较大的测试，产生的 Log 文件可能过于庞大。

5.6.2 PA 子系统提供的系统统计

用 PA LLD 提供的 Pa_requestStats 和 Pa_formatStatsReply 函数可以提取 PA 子系统给出的统计计数器，这些统计计数器在系统运行过程中可以诊断部分的包匹配，包错误等问题。下面是对这些计数器的详细说明。

```
/*pa 统计结构体*/                    C1 统计是基于第一级分类模块的（PDSP0，PDSP1 and PDSP2）
typedef struct pa_c1_stats_s {
  UINT32 nPackets;                  进入到第一级分类模块的包总数
  UINT32 nIpv4Packets;              接收到的 IPv4 总数
  UINT32 nIpv4PacketsInner;         接收到的内层 IP 为 IPv4 的包总数(一般用于统计 IPSec 隧道模式中，
内层 IP 为 IPv4 的情况，即 PDSP2 接受到的 IPv4)
  UINT32 nIpv6Packets;              接收到的 IPv6 总数
  UINT32 nIpv6PacketsInner;         接收到的内层 IP 为 IPv6 的包总数(一般用于统计 IPSec 隧道模式中，
内层 IP 为 IPv6 的情况，即 PDSP2 接受到的 IPv6)
  UINT32 nCustomPackets;            match 自定义的一级分类包数(自定义的一级分类包
Pa_setCustomLUT1)
  UINT32 nSrioPackets;              从 SRIO 接口转发来的包数
  UINT32 nLlcSnapFail;              接收到的错误的 LLC/SNAP 帧数
  UINT32 nTableMatch;               接收到的 match LUT1-0，LUT1-1，LUT1-2 的包总数
  UINT32 nNoTableMatch;             接收到的没有 match LUT1-0，LUT1-1，LUT1-2 的包总数
  UINT32 nIpFrag;                   接收的 IP 分片包的总数
  UINT32 nIpDepthOverflow;          接收到的超过配置的 IP 层数的包数
  UINT32 nVlanDepthOverflow;        接收到的超过配置的 VLAN 层数的包数
  UINT32 nGreDepthOverflow;         接收到的超过配置的 GRE 层数的包数
  UINT32 nMplsPackets;              接收到的 MPLS 包数
  UINT32 nParseFail;                分析失败的包数
  UINT32 nInvalidIPv6Opt;           接收到的非法的 IPv6 option 的 IPv6 包数
  UINT32 nTxIpFrag;                 发送的 IP 分片数
  UINT32 nSilentDiscard;            被第一级分类模块丢弃的包数
  UINT32 nInvalidControl;           第一级分类模块非法的 PA 配置命令总数
  UINT32 nInvalidState;             PA 自检测错误并恢复的次数
  UINT32 nSystemFail;               PA 自检测错误并无法恢复，重启动次数


}PA_CLASSIFY1_STATS_S;
typedef struct pa_c2_stats_s  {   C2 统计是基于第二级分类模块的（PDSP3）
  UINT32 nPackets;                  进入到第二级分类模块的包总数
  UINT32 nUdp;                      接收到的 UDP 包总数
  UINT32 nTcp;                      接收到的 TCP 包总数
  UINT32 nCustom;                   match 自定义的二级分类包数(自定义的二级分类 Pa_setCustomLUT2)
  UINT32 nSilentDiscard;            被第二级分类模块丢弃的包数
  UINT32 nInvalidControl;           第二级分类模块收到的非法的 PA 配置命令总数
} PA_CLASSIFY2_STATS_S;
```


5.6.3 辅助调试的寄存器

利用PDSP所提供的一些辅助计数器/寄存器可以更容易的在运行时定位问题。比如下面的计数器

0x2000000 + 0x10*n PDSPn 收到的包数（包括配置命令和数据包）

0x2000008 + 0x10*n PDSPn收到的配置命令数

举例说明其用法：

1. 如果读到0x2000000为0xa，0x2000008为0x1，代表PDSP0收到了9个数据包和1个配置命令
2. 在第一步的基础上继续读到

   0x2000010为0x3，0x2000018为0x1， 代表PDSP1收到了2个数据包和1个配置命令
3. 根据前两步的结果，我们可以知道，有2个数据包在PDSP0收到后继续传给了PDSP1

下面是PDSP提供的辅助调试寄存器列表：

PDSP Name	PDSP Control	PDSP Status/PC	PDSP Instruction RAM	Debug register 32X4 byte	LUT table bitmap	LUT table Route info
PA PDSP0	0x2001000	0x2001004	0x2010000	0x2002000	0x2040000 (8 bytes)	0x2041000 (64bytes/entry)
PA PDSP1	0x2001100	0x2001104	0x2018000	0x2002100	0x2040100 (8 bytes)	0x2043000 (64 bytes/entry)
PA PDSP2	0x2001200	0x2001204	0x2020000	0x2002200	0x2040200 (8 bytes)	0x2045000 (64 bytes/entry)
PA PDSP3	0x2001300	0x2001304	0x2028000	0x2002300	0x2040300 (1024 bytes)	0x2047000 (64 bytes/entry)
PA PDSP4	0x2001400	0x2001404	0x2030000	0x2002400		
PA PDSP5	0x2001500	0x2001504	0x2038000	0x2002500		
SA PHP1 (IPSec)	0x20C1000	0x20C1004	0x20c4000	0x20C1400		
SA PHP2 (Air cipher& SRTP)	0x20C1100	0x20C1104	0x20c8000	0x20C1500		

表1 PDSP 辅助寄存器列表

PDSP control register: PDSP 寄存器，控制 PDSP 的运行状态。

PDSP status register: PDSP 状态寄存器，其实是 PDSP 固件的 PC 值，该值不恒定，如果该值一直不变化，可以确定是 PDSP 固件 crash。

PDSP instruction RAM: PDSP 指令空间，设置 PDSP control register 为 1，可以将 PDSP halt，同时查看 PDSP 的指令空间。在某些功能异常的时候，可以回读 PDSP 指令空间，特别是固件的版本号，看看是否与期望的一致。

PDSP debug register: PDSP 的调试寄存器。

LUT table bit map: 查找表的占用状态。每个查找表的表项在占用后，PDSP 都会对该表项对应的位域置位。这个位表中保存的是 PDSP/硬件所维护的查找表状态，应用软件可以将其回读来与自己所保存的比较，看看是否有不一致的情况发生。理论上软硬件应该是同步的。

LUT table route info: 根据 LUT table bit map 所查找到的对应表项下标，乘以 64 字节的偏移量，可以找到该表项匹配后对应的路由信息，该信息是由软件配置的。同样可以回读，看看是否与期望的一致。

5.6.4 PDSP的单步跟踪

在某些 PDSP 相关的问题定位过程中，可能需要对 PDSP 做单步跟踪。该方法主要适用于在 simulator 的方式下无法重现的功能问题，比如表项匹配失败失败，路由错误等等。获得的信息可以交由 TI 进行分析。

```
#include <ti/csl/cslr_device.h>
#include <ti/csl/cslr_pa_ss.h>

void mdebugHaltPdsp (Int pdspNum)
{
    CSL_Pa_ssRegs *passRegs = (CSL_Pa_ssRegs *)CSL_PA_SS_CFG_REGS;
    passRegs->PDSP_CTLSTAT[pdspNum].PDSP_CONTROL &= ~(CSL_PA_SS_PDSP_CONTROL_PDSP_ENABLE_MASK);
}

void mdebugRunPdsp (Int pdspNum)
{
    CSL_Pa_ssRegs *passRegs = (CSL_Pa_ssRegs *)CSL_PA_SS_CFG_REGS;
    passRegs->PDSP_CTLSTAT[pdspNum].PDSP_CONTROL |= (CSL_PA_SS_PDSP_CONTROL_PDSP_ENABLE_MASK);
}



#pragma DATA_SECTION (dbgPdspPC, ".testPkts")
#pragma DATA_SECTION (dbgPdspRegs, ".testPkts")
uint32_t dbgPdspPC[1000];
uint32_t dbgPdspRegs[1000][32];


/* Note: The corresponding PDSP should be disabled before this function is called */

void mdebugStepPdsp(Int pdspNum, Int numCycles)
{

    int i;
    CSL_Pa_ssRegs *passRegs = (CSL_Pa_ssRegs *)CSL_PA_SS_CFG_REGS;
    /* PASS base address, it can be virtual or physical address depending on where this code is
running */

    CSL_Pa_ssPdsp_ctlstatRegs *pdspRegs = &passRegs->PDSP_CTLSTAT[pdspNum];
    /* offset = 0x1000, 0x1100, 0x1200,etc */
    CSL_Pa_ssPdsp_debugRegs *pdspDbgRegs = &passRegs->PDSP_DEBUG[pdspNum];
    /* offset = 0x2000, 0x2100, 0x2200, etc */

    for (i = 0; i < numCycles; i++)
    {
        dbgPdspPC[i] = pdspRegs->PDSP_STATUS;  /* offset = 4 */
        memcpy(dbgPdspRegs[i], (void *) &pdspDbgRegs->PDSP_IGP[0], 32*sizeof(uint32_t));

        pdspRegs->PDSP_CONTROL = 0x10b;     /* single step */

        utilCycleDelay (50);    /* a simple deal loop */
    }
    pdspRegs->PDSP_CONTROL = 3; /* re-enable PDSP for normal operation */
}
```


调用示例：单步跟踪 PDSP1,600 步

```
mdebugStepPdsp(1, 600);
```

6. SA

SA 调试的过程中，请充分利用 SA 提供的相关统计信息：

6.1 系统统计

```
typedef struct {
  uint32_t  errNoMem;       /*由于内存不足（包括实例内存和安全上下文内存）而导致的丢包数*/
  uint32_t  errCtx;         /*找不到对应的安全上下文而导致的丢包数*/
  uint32_t  errEngine;      /*对应的硬件加解密引擎错误而导致的丢包数*/
  uint32_t  errProto;       /*被丢弃的错误包数，主要未能通过协议头检查的包*/
} Sa_ErrorSysStats_t;
```

注意：

描述符中提供了 PS_flag 字段，可以指定该描述符所发向的目的 SGMII 端口(direct packet mode)，但是当该描述符要先经过 SA 的时候，请清空该字段，否则 SA 对该包的处理导致安全上下文的错误(*errCtx*)。在经过 SA 处理后的包仍需要指定对应的 SGMII 端口的情况，配置该包从 SA 输出以后的路由到 645 队列/PDSP5，并用 PA 的 Next route 命令指定到以太网以及对应的 SGMII 端口号.

6.2 通道统计

6.2.1 SRTP

```
typedef struct {
  uint32_t  replayOld;      /*防重放检测到与以前序列号的包而丢弃的包数*/
  uint32_t  replayDup;      /*防重放检测到与以前相同的包而丢弃的包数*/
  uint32_t  authFail;       /* 鉴权失败而丢弃的包数 */
  uint32_t  pktEncHi;       /*成功加密的包数高 32bit*/
  uint32_t  pktEncLo;       /*成功加密的包数低 32bit*/
  uint32_t  pktDecHi;       /*成功解密的包数高 32bit*/
  uint32_t  pktDecLo;       /*成功解密的包数低 32bit*/
} Sa_SrtpSysStats_t;
```

6.2.2 IPSec

```
typedef struct {
  uint32_t  replayOld;      /*防重放检测到与以前序列号的包而丢弃的包数*/
  uint32_t  replayDup;      /*防重放检测到与以前相同的包而丢弃的包数**/
  uint32_t  authFail;       /*鉴权失败而丢弃的包数 */
  uint32_t  pktEncHi;       /*成功加密的包数高 32bit*/
  uint32_t  pktEncLo;       /*成功加密的包数低 32bit*/
  uint32_t  pktDecHi;       /*成功解密的包数高 32bit*/
  uint32_t  pktDecLo;       /*成功解密的包数低 32bit*/
} Sa_IpsecSysStats_t;
```

6.2.3 Air Cipher

```
typedef struct {
  uint32_t  authFail;       /*鉴权失败而丢弃的包数 */
  uint32_t  pktToAirHi;     /*成功加密的包数高 32bit */
  uint32_t  pktToAirLo;     /*成功加密的包数低 32bit */
  uint32_t  pktFromAirHi;   /* 成功解密的包数高 32bit */
  uint32_t  pktFromAirLo;   /*成功解密的包数低 32bit */
} Sa_AcSysStats_t;
```


注意：

取通道统计应该在调用完SA API Sa_chanReceiveData或Sa_chanSendData以后，而这两个API本身有部分加解密的初步检查作用，应用程序必须要对返回值进行判断。如果它们返回错误，则应该终止向SA发包的过程，并根据返回值来检查自身应用软件的错误。何时调用这两个API是需要注意的另一个问题，在IPsec/SRTP的场景下，TX侧代表to network方向，即加密方向，需要调用调用Sa_chanSendData；Rx侧代表from network方向，即解密方向，需要调用Sa_chanReceiveData。对于air cipher的场景, TX侧对于空中接口来说实际为from air方向, 即解密方向，需要调用调用Sa_chanReceiveData；Rx侧代表to air方向，即加密方向，需要调用Sa_chanSendData。

6.3 辅助寄存器

SA 的辅助寄存器主要用来定位 SA 相关的硬件错误，当然该硬件错误可能是由软件的非法操作导致的。下面是常用的辅助寄存器列表，表中标注了正常情况下所期望的值。特别请注意 CMD_STATUS 寄存器，在运行时间的 SA 异常情况，常能够反映到该寄存器中。其具体的含义请参考 SA 用户手册。

Register Name	Register Address	Expected value
EFUSE enable	0x20c0004	1
CMD_STATUS	0x20c0008	0xfef
BLKMGR_PA_BLKS	0x20c000C	4
PA_ENG_ID	0x20c0018	8
CDMA_ENG_ID	0x20c001C	0x10

表 2　SA 调试寄存器列表

7. 总结

本文主要总结了目前为止中国区用户在开发 Keystone I 系列芯片的时候，所遇到的以太网子系统相关的典型技术问题，以及阅读 user guide 时一些较为集中的疑问。很多用户在开发过程中容易忽略的问题本文都做了强调。PA 子系统部分的调试办法基本可以覆盖开发过程中所遇到的所有问题。文档中给出的各种寄存器配置以及示例程序，都在实际开发中有使用。