S32G PFE Master/Slave Simple Demo Building

by John Li (nxa08200)

本文说明在S32G3 RDB3板上实现M核PFE Master, A核Slave 简易方式Demo的搭建过程。

软件版本为 RTD4.0.0+PFE MCAL1.0.0(已经内置 PFE FW 1.4.0)+(PFE FW 1.4.0 可选)+BSP34(PFE Linux Driver 1.1.0)。

历史	说明	作者
V1	● 创建本文	John.Li

目录

1 需要的软件与工具

软件/工具	名称	说明
PFE MCAL 软件	PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2.zip	内部已经包含了 FW 1.4.0 \example_application\pfe_firmware\s32g_pfe_class.c/h
PFE FW	PFE-FW_S32G_1.4.0.zip	可选: PFE-FW_S32G_1.4.0\ s32g_pfe_class.c/h/fw s32g_pfe_util.c/h/fw PFE_Firmware_S32G_UserManual.pdf PFE-FW_S32G_1.4.0_ReleaseNotes.pdf
AutoSar MCAL 基础软件	SW32_RTD_4.4_4.0.0_D2210.exe	Modules configurations were developed and tested using the Tresos Configuration Tool version "EB tresos Studio 27.1.0 b200625-0900"
Linux BSP	BSP34(默认所含 PFE 版本为 1.1.0)	

注意：版本匹配：

- \ PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2 \ PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2_ReleaseNotes.txt

  This package shall be integrated with following version of NXP AUTOSAR MCAL:

  - SW32_RTD_4.4_4.0.0_D2210

  This package uses PFE firmware:

  - s32g_pfe_class firmware version 1.4.0

  ==========

  Compatibility

  ==========

  - Compatible firmware: PFE-FW RTM 1.4.0
  - Compatible MCAL Master driver: PFE LINUX RTM 1.1.0 (for Master-Slave Scenarios)
  - Compatible RTD: SW32_RTD_4.4_4.0.0_D2210

2 Demo 说明

2.1 RDB3 PFE 网络拓扑

S32G3 芯片手册中 serdes 说明：

Table 404. SerDes_0 working modes

Lane Configuration[1]	SerDes Subsystem Mode[2]	PHY lane 0	PHY lane 1	Lane 0 speed	Lane 1 speed	PHY reference clock (MHz)
0	Mode 0	PCIe0_x2 lane 0	PCIe0_x2 lane 1	Gen 3/2/1		100
1	Mode 1	PCIe0_x1 lane 0	SGMII (GMAC0)	Gen 3/2/1	1.25 Gbit/s	100

Table continues on the next page...

Table 404. SerDes_0 working modes (continued)

Lane Configuration[1]	SerDes Subsystem Mode[2]	PHY lane 0	PHY lane 1	Lane 0 speed	Lane 1 speed	PHY reference clock (MHz)
2	Mode 2	PCIe0_x1 lane 0	SGMII (PFEMAC2)	Gen 3/2/1	1.25 Gbit/s	100
3	Mode 3	SGMII (GMAC0)	SGMII (PFEMAC2)	1.25 Gbit/s		100 or 125[3]
4	Mode 2[4]	PCIe0_x1 lane 0	SGMII (PFEMAC2)	Gen 2/1	3.125 Gbit/s	100

Table 405. SerDes_1 working modes

Lane Configuration[1]	SerDes Subsystem Mode[2]	PHY lane 0	PHY lane 1	Lane 0 speed	Lane 1 speed	PHY reference clock (MHz)
0	Mode 0	PCIe1_x2 lane 0	PCIe1_x2 lane 1	Gen 3/2/1		100
1	Mode 1	PCIe1_x1 lane 0	SGMII (PFE MAC0)	Gen 3/2/1	1.25 Gbit/s	100
2	Mode 2	PCIe1_x1 lane 0	SGMII (PFE MAC1)	Gen 3/2/1	1.25 Gbit/s	100
3	Mode 3	SGMII (PFE MAC0)	SGMII (PFE MAC1)	1.25 Gbit/s		100 or 125[3]
4	Mode 4	SGMII (PFE MAC0)	SGMII (PFE MAC1)	3.125	1.25	100 or 125[3]
5	Mode 4	SGMII (PFE MAC0)	SGMII (PFE MAC1)	1.25	3.125	100 or 125[3]
6	Mode 4	SGMII (PFE MAC0)	SGMII (PFE MAC1)	3.125 Gbit/s		125
7	Mode 2[4]	PCIe1_x1 lane 0	SGMII (PFE MAC1)	Gen 2/1	3.125 Gbit/s	100

1. For lane configuration settings refer to the S32G3SERDESRM.
2. For details of SS_RW_REG_0[SUBSYS_MODE], see S32G3SERDESRM.
3. The SerDes subsystem's default setting is 125 MHz.
4. The max PCIe speed is limited to Gen 2.

然后在 RDB3 上设计中，仅考虑从 Serdes_1 出 EMAC0/1 的情况，则如 Linux BSP 手册说明：

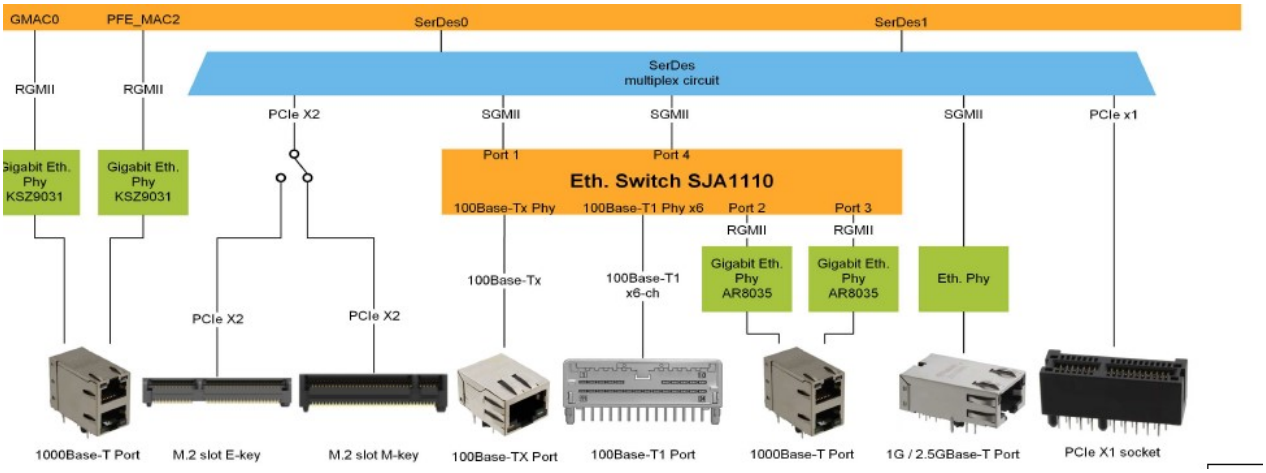
S32G399A RDB3 default port mapping in U-Boot:

SoC port	interface	board port	note
PFE_EMAC_0	SGMII(2.5G) to switch SJA1110A	P2, P4	–
PFE_EMAC_1	SGMII(2.5G)	P5	–
PFE_EMAC_2	RGMII	P3 top	–
GMAC	RGMII	P3 bottom	–
hwconfig	'pcie0:...;pcie1:mode=sgmii,clock=ext,fmhz=125,xpcs_mode=2G5'		

S32G399A RDB3 default port mapping in Linux:

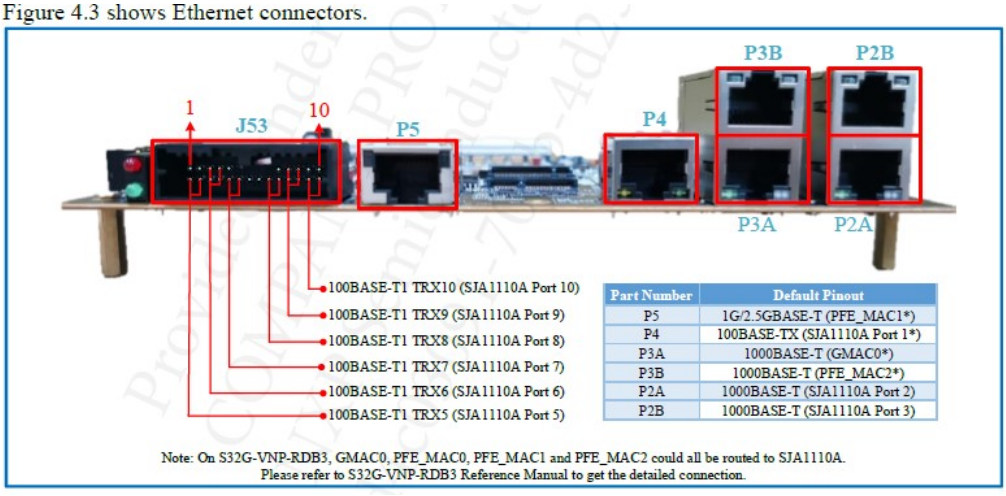
SoC port	interface	board port	Linux name	note
PFE_EMAC_0	SGMII(2.5G) to switch SJA1110A	P2, P4	`pfe0`	–
PFE_EMAC_1	SGMII(2.5G)	P5	`pfe1`	–
PFE_EMAC_2	RGMII	P3 top	`pfe2`	–
GMAC	RGMII	P3 bottom	`eth0`	–

对应 RDB3 硬件设计：



及 RDB3 的 user guide 说明：

Figure 4.3 shows Ethernet connectors.



然后对应 MCAL 的说明：

EMAC0 is connected to the SGMII (SERDES PCIe) PHY on the processor board. This is the default configuration

EMAC0 is also connected to Ethernet switch sja1105p on platform board, which is connected to ports: 2x BroadReach, RGMII-ADD-2 and RGMII-ADD-3.

To enable this configuration, update the following:

-in the PFE configuration

change EthCtrlMacLayerSubType for EthCtrlConfig_0 from SERIAL to REDUCED

-in the Mcu configuration:

change McuGENCTRL1 Source from SERDES_1_XPCS_0_TX to PFEMAC0_TX_DIV_CLK

change McuCgm2ClockMux4 Source from SERDES_1_XPCS_0_CDR to PFE_MAC_0_EXT_RX_CLK

Refer to chapter [23.7.2.3.1 PFE_MAC_0 clocking overview] in S32G2_RM for more information

If sja1105p switch driver is not integrated to application, then transmission and reception on EMAC0 will not work in this configuration.

所以注意以上描述非针对 RDB3 的说明，实际上在 RDB3 上 EMAC0 使用 SGMII 接口连接到 SJA1110 上。

EMAC1 is connected to RGMII port on processor board.

EMAC1 在 RDB3 上，可以配置为 SGMII，直接从 P5 出来，也可以配置为 RGMII，从 GMAC0 所使用的 P3A 口出来。

注意 MCAL 的 Master 工程默认就是使用如此设计，三个 ethctrl 设备分别为：EMAC0-SGMII, EMAC1-RGMII, AUX。接口。

另外注意在 ATF 初始化时，将 EMAC0/1/2 的 IOMUX 的 RGMII 接口，已经预先配置好。

所以 Linux 的软件设置和 MCAL PFE master example 工程是有一定冲突的，需要修改。如果不考虑 SJA1110 的影响，则 EMAC1 如果是配置成 SGMII 接口(Linux 默认)，则是从 P5 口接出，如果是配置成 RGMII 接口(MCAL 默认，Linux 需要修改)，则是从 P3A 接出。

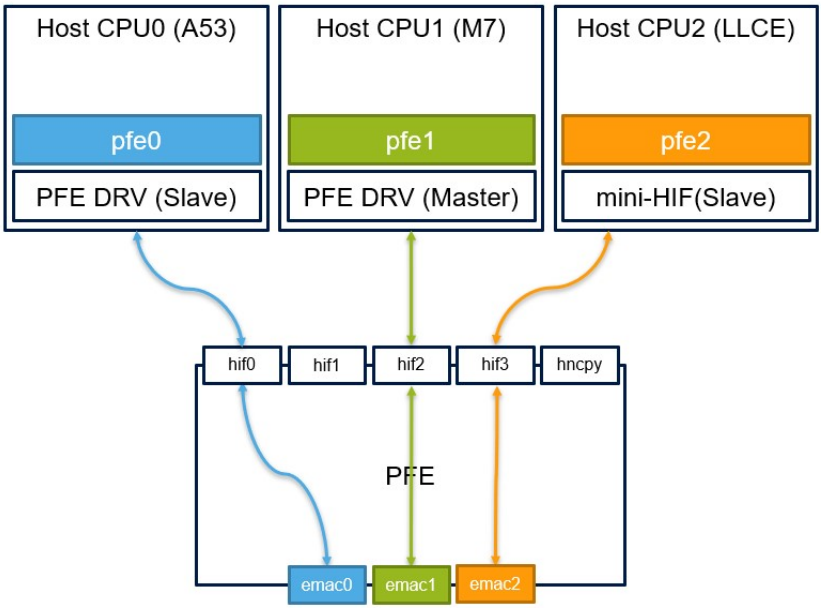
2.2 Master/Slave Demo 说明

PFE 驱动程序可以在同一系统内运行在多实例模式，多个实例可以通过 HIF 接口连接到 PFE 上。预期用例是在一个主机内运行一个主实例，以及一个或多个来自可以访问 PFE 的其他主机的从实例，甚至来自运行主实例的主机操作系统之外的其他操作系统 (OS)。

“主”实例实现完整的 PFE 平台驱动程序并执行一般 PFE 硬件初始化和启动。 主驱动程序可以访问完整的 PFE 寄存器空间。

“从属”实例仅实现部分平台驱动程序功能。

从实例不应该在主实例正常工作之前开始初始化。



2.3 本简易 Demo 说明

在真实的产品中，一般会使用一个基于 M7_0 核的 bootloader 来启动 M 和 A 核，这个 bootloader 负责所有 M 核和 A 核资源的初始化，解决 M 核和 A 核的资源冲突，并且启动 M 和 A 核。所以理论上运行 M PFE Master Mcal 驱动加 A PFE Slave Linux 驱动也是需要一个 bootloader 的。参考文档《S32G_Bootloader_V*》，Johnli，可以在公开 community 上搜索获得。

本文讨论一种简易的办法，就是：

- S32G3 RDB3 板子配置为 SDcard 启动，插入 SDcard，里面放有 PFE SLAVE 驱动的 Linux 镜像。
- 上电启动后运行 PFE Master 工程的 lauterbach 调试脚本：run_main_G3_REV1_1.cmm，这个脚本会重启整个 S32G3。
- 然后在脚本中用 wait 10S 的操作，这个时候 Linux 已经启动，并且使用 Uboot 的代码调用 ATF 来完成 PFE 相关 pre-init, partition reset 和时钟与管脚初始化(如上分析， EMAC0~2 的 RGMII IOMUX 已经配置好)，然后 Slave 驱动会等待一段时间，等 MCAL Master 驱动加载，继续运行 PFE Master MCAL 代码后，Linux 端 Slave 驱动也加载正确。然后就可以测试整个 M Master/A Slave Demo。

总结：以上办法实际上是把 bootloader 应该做的 PFE 相关硬件初始化工作由 Linux 来完成，以便快速搭建 Demo，这样客户在做真实的产品开发时，可以做为一个 NXP release 的标准参考。

3 M 核 Master 驱动编译说明

3.1 安装 RTD_MCAL 驱动

双击 SW32_RTD_4.4_4.0.0_D2210.exe 安装。安装向导中会要求提供 EB Tresos Studio 的路径（C:\EB\tresos\）。

RTD正确安装后，会在安装路径（默认：C:\NXP\SW32_RTD_4.4_4.0.0）下看到源码以及相关文档。并且对应的EB Tresos Studio安装路径下的links文件夹中会看到.link的文件，这个文件表明EB Tresos Studio已经和RTD关联起来。之后打开EB Tresos Studio新建或导入RTD配置工程时，它会关联到RTD路径下。如果安装时没有提供EB路径，也可以手动在EB相应的路径下自行新建后缀为.link的文件，写入刚才RTD的安装路径。

C:\EB\tresos\links\SW32_RTD_4.4_4.0.0.link

path=C:/NXP/SW32_RTD_4.4_4.0.0

注意其它的.link 文件要修改为备份名，比如说修改为 SW32G_RTD_4.4_3.0.2.link.bak，以防止冲突。MCAL 的详细情况，可以参考文档《S32G_RTD_MCAL_V*.pdf》。

3.2 安装 PFE_MCAL 驱动

将\pfe\Driver PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2 \eclipse\plugins\ Eth_43_PFE_TS_T40D11M10I0R0拷贝到C:\SW32_RTD_4.4_4.0.0 \eclipse\plugins。

3.3 编译 PFE master 工程

文档PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2 \example_application\readme.txt说明了PFE工程的相关情况，建议阅读。

打开EB Tresos 27.1.0，File->Import…->General->Existing Projects into Workspace:->Next Select root directory->Browse…->PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2 \example_application\MasterProject_G3_REV1_1

从而打开工程MasterProject_G3_REV1_1。(勾选 Copy projects into workspace)。

在工程MasterProject_G3右击，选择Properties->Resource->Location: C:\EB\tresos\workspace\MasterProject_G3_REV1_1。此为工程所在目录

Configuration Porject->Code Generation: Default Generation Path= ..\generate_G3_REV1_1。此为生成代码的地方， 所以：

在工程MasterProject_G3_REV1_1右击，选择generate project，使用EB生成代码后，将C:\EB\tresos\workspace\generate_G3_REV1_1*拷贝到PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2 \example_application

修改编译相关文件：PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2\example_application\ project_parameters.mk

GCC_DIR ?= C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2/gcc-9.2-arm32-eabi #编译器位置

TRESOS_DIR ?= C:/EB/tresos #指向EB安装 目录

PLUGINS_DIR ?= C:/NXP/SW32_RTD_4.4_4.0.0/eclipse/plugins #RTD plugins安装目录

然后修改Makefile:

HW ?= S32G3 #本来为G2，修改为G3

DEBUG ?= FALSE #默认不修改，可以使用编译参数

COMPILER ?= GCC #默认不修改，可以使用编译参数

Cygwin中编译MasterProject_G3的命令：

```
cd …/PFE-DRV_S32G_M7_MCAL_BETA_1.0.0_QLP2/example_application
$ make HW=S32G3_REV1_1 clean
$ make HW=S32G3_REV1_1
```

输出镜像为：

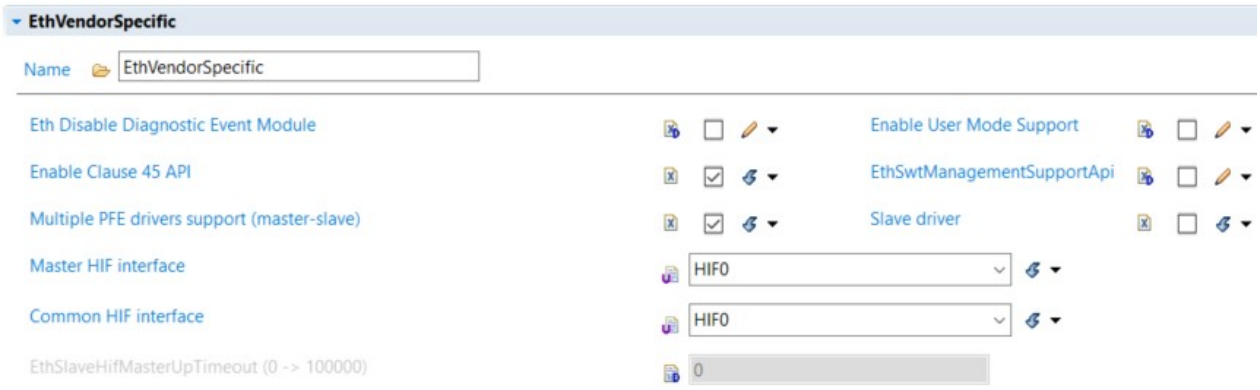
```
-o output_G3_REV1_1/main_G3_REV1_1.elf
```

3.4 PFE master 工程说明

1. EB 配置

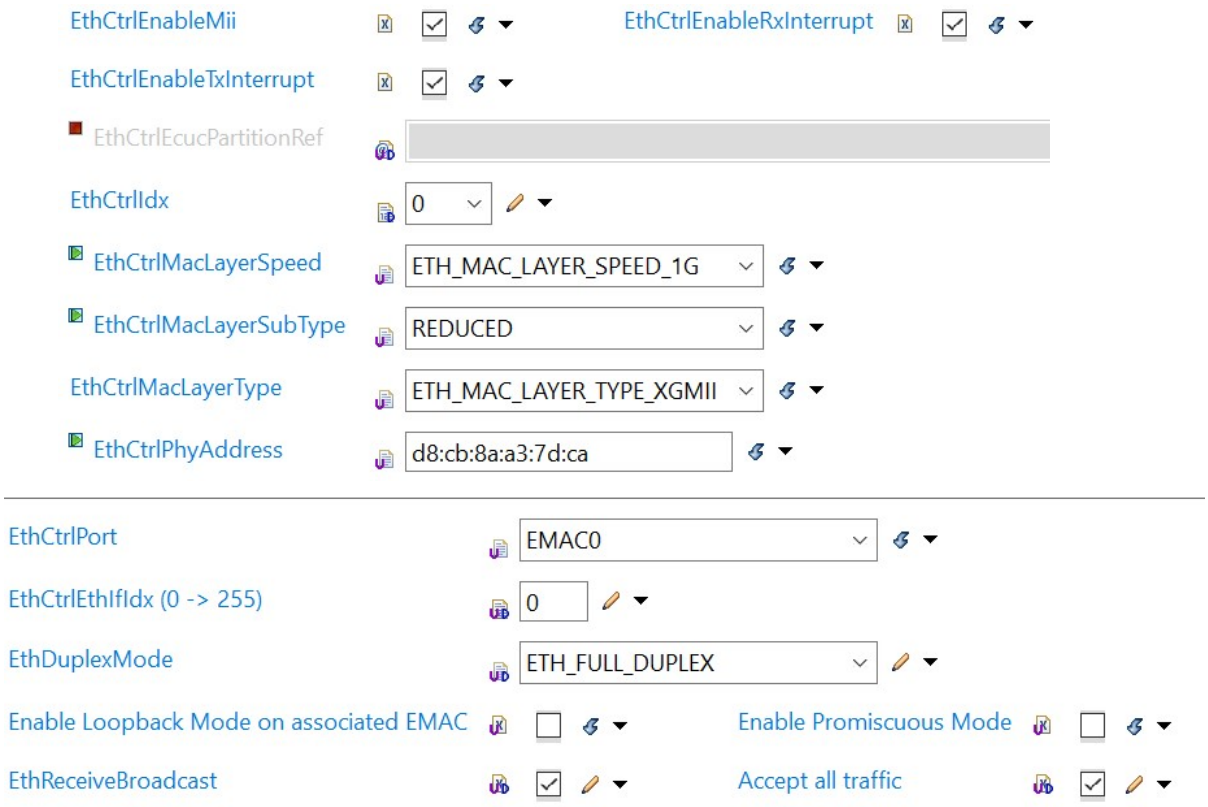
   MasterProject_G3_REV1_1->someId(…)->Eth_43_PFE->Eth_1(…)->General:

- Master HIF=HIF0， //Specifies HIF interface used by master PFE driver. This option is valid only if option 'Slave driver' is set
- Common HIF interface=HIF0 //Defines the HIF interface which this driver will use in case the "Use multiple HIF interfaces" option is disabled
- Multiple PFE drivers support (master-slave)=checked // When ON then role of this driver can be selected: master or slave and other driver(s) (Linux/QNX/another MCAL driver) can be used on the same system simultaneously
- EthGlobalTimeSupport =checked: Enables/disables the GlobalTime APIs used amongst others by Global Time Synchronization over Ethernet
- Slave drive=unchecked: //This option depends on option "Multiple PFE drivers support (master-slave)" being enabled. When ON then this driver requires another driver to be simultaneously running on the device. The other driver shall be configured as master. When OFF then this driver is a master driver that can either run alone or it can support one or more slave drivers running on the same device.
- Enable errata ERR051211 workaround=checked //The PFE MCAL driver implements workaround for ERR051211. If the workaround is enabled, the driver will automatically allocate 72 extra Rx buffers and Rx buffer descriptors. The number 72 a fixed value and there is no configuration option to change it.
- Enable FCI, routing and bridging support=checked //If FCI API is used, it needs to be enabled
- VarEthBmu2BufCnt=1024 //BMU2 buffer number, internal PFE memory buffers. Size of this memory region “pfe_bmu_mem” can be calculated as “VarEthBmu2BufCnt * 2048”. This memory region must reside within “0x00020000 - 0xBFFFFFFF” and must be aligned to its size.



MasterProject_G3_REV1_1->someId(…)->Eth_43_PFE->Eth_1(…)->EthCtrlConfig-> EthCtrlConfig_0->General: (由于 eMAC0 需要外部 SJA1110 测试，所以本 Demo 不用这个口测试，理论上可以删除掉)。

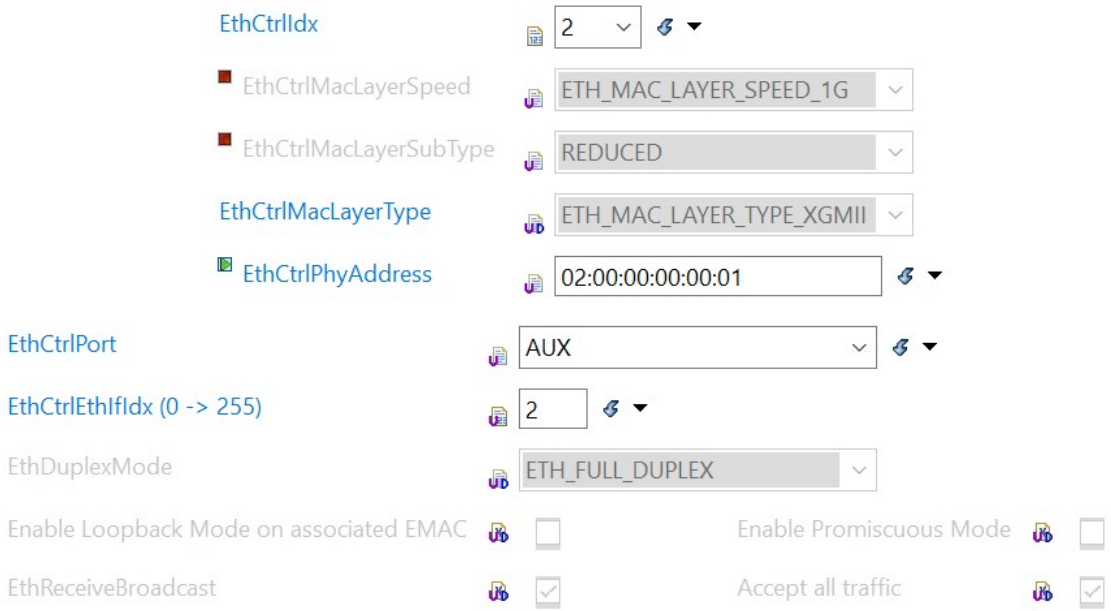
- Accept all traffic=checked: This parameter is for the case where the HIF associated to this instance and the EMAC corresponding to this controller are part of a bridge domain. If enabled, all ingress traffic received on the corresponding EMAC will be received by this controller. If disabled, only management traffic from the EMAC will be received by this controller. All other traffic will be received on the AUX controller. It is recommended to disable this parameter when the corresponding EMAC is part of bridge domain.



相关联 EthConfigEgressFiFo/IngressFiFo/ConfigSchedule 需要配置,建议对不同的 EthCtrlConfig 的 Egress/Ingress 配置采用不同的名称。

EthCtrlConfig_1 关联 EMAC1，与上相似，使用 RGMII 接口，本 Demo 使用此接口。

EthCtrlConfig_2 关联 AUX 接口，如下：(注意，如果需要 M/A 核可以互相连接网络，则需要在 M 核也配置 AUX 接口，然后将此 AUX 和 Linux 端的 aux，以及外出的 emac 加到一个网桥中)。



2. 发包代码说明：

Master Demo 代码提供了一个 ARP 广播包：

```
/* ARP packet */
uint8 tst_frm2_data[] =
{
  0x00U, 0x01U, /* HW type */
  0x08U, 0x00U, /* Protocol type */
  0x06U, /* HW size */
  0x04U, /* Protocol size */
  0x00U, 0x01U, /* Opcode: ARP request */
  0xd8U, 0xcbU, 0x8aU, 0xa3U, 0x7dU, 0xcfU, /* Sender MAC */
  0x00U, 0x00U, 0x00U, 0x00U, /* Sender IP */
  0x00U, 0x00U, 0x00U, 0x00U, 0x00U, 0x00U, /* Target MAC */
  0x00U, 0x00U, 0x00U, 0x00U, /* Target IP */
};
```

Main

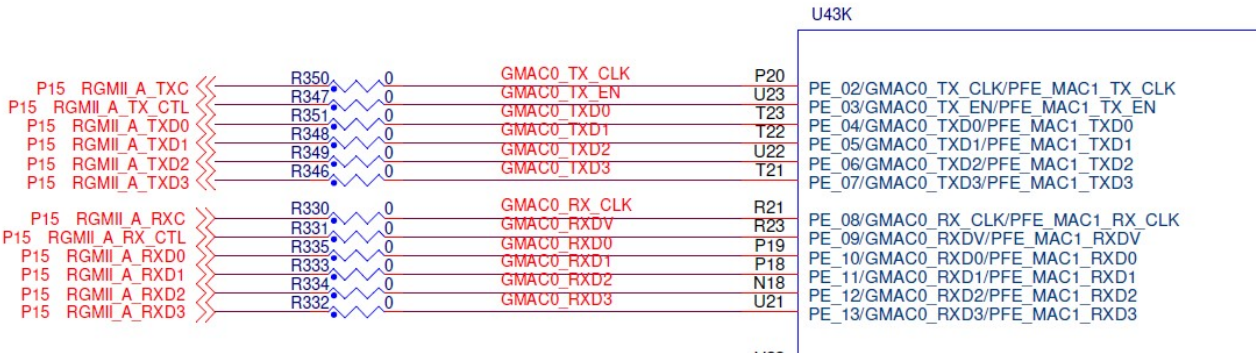
```
|-> SampleAppTask2
|   |->所以以下 for 循环轮流在 EMAC0/1/AUX 上发送广播包
/* Send a test frame from all the PFE controllers */
   for (CtrlIndex = 0U; CtrlIndex < ETH_43_PFE_NUM_CONTROLLER_CFG; CtrlIndex++)
   {
       tempRet |= Tst_Pfe_SendDummyFrame(CtrlIndex, broadcastMac);
|   |   |->
/* Write frame data */
       for (Idx = 0U; Idx < TST_FRM_LENGTH; Idx++)
       {
           bufPtr[Idx] = tst_frm2_data[Idx];
       }

       LengthInBytes = TST_FRM_LENGTH;
       tempRet = Eth_43_PFE_Transmit(srcCtrlIndex, BufferIndex, TST_FRM_TYPE, TRUE, LengthInBytes,
dstMacAddr);
```

3.5 RDB 板的 RGMII 接口支持说明

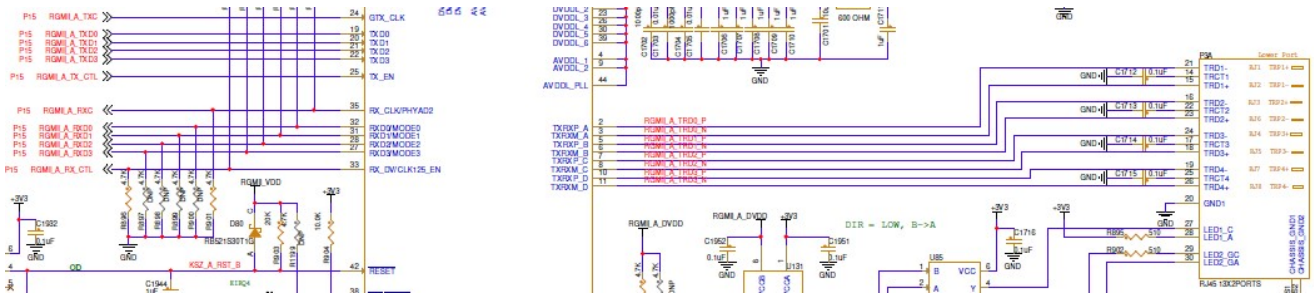
Master 工程本身的 EthCtrl0 连接的 EMAC0 为 SGMII 接口，EthCtrl1 连接的 EMAC1 为 RGMII 接口。

1. 硬件连接

在 RDB3 板上， PFE_EMAC1 上，如下设计：



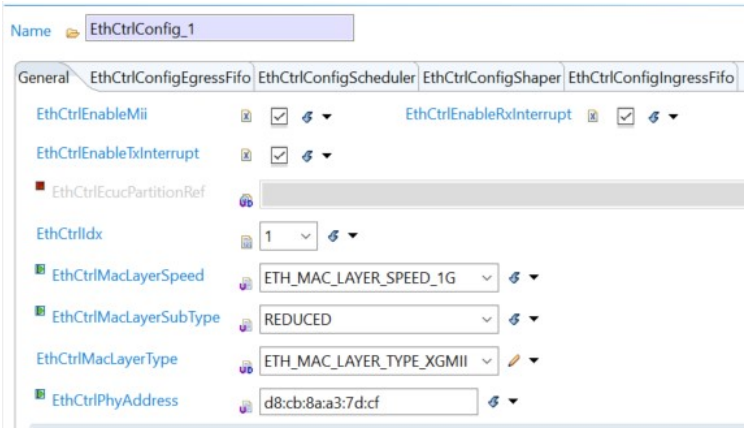
外部通过 KSZ9031 PHY 连接到 P3A RJ45 接口上：



PFE_MAC1 复用了 GMAC0 的 IO。

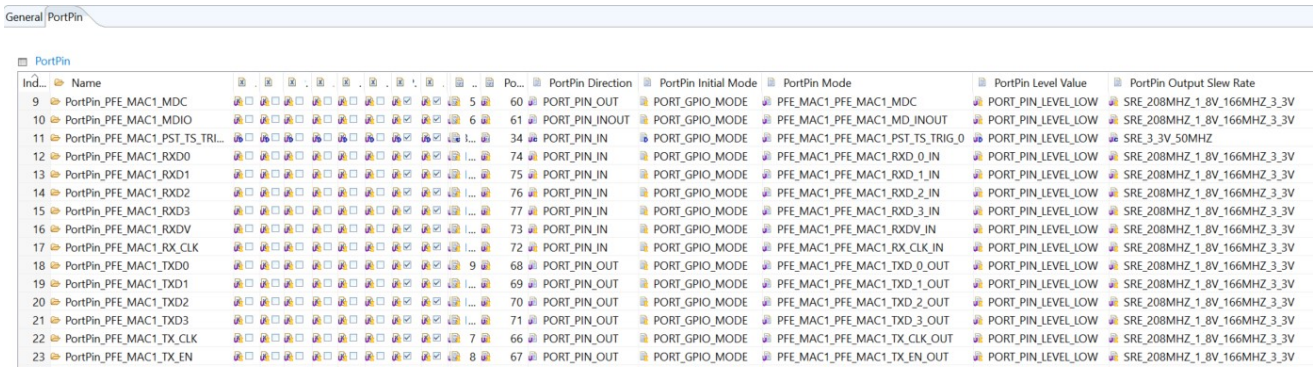
2. 软件说明

所以默认 PFE_EMAC1 在 PFE Master 工程中是设置为 RGMII 接口，如下：
MasterProject_G3->SomeId->Eth_43_PFE->Eth_1->EthCtrlConfig->EthCtrlConfig_1



PORT 模块已经预先配置好了，如下将 GMAC0 的 IOMUX 设置为了 PFE_EMAC1(此处可以不用考虑，因为本 Demo 使用 Linux 的 ATF 来初始化 IOMUX):

MasterProject_G3_REV1_1->SomeId->Port->Port->PortContainer->PortContainer_0->PortPin



注意发送函数是循环在三个 EthCtrl 接口上发送的，所以如果之前的 eMAC 初始化失败，会影响到后面正常的 eMAC 接口, 所以如果是客户自己的板子，要注意 EMAC0 是否有时钟，是否能正确初始化。

还有就是 MCU clock 设置中，PFE_EMAC1 的 TX/RX 源要确认一下，不能是 serdas 源，如下：

MasterProject_G3_REV1_1->SomeId->Mcu->Mcu->McuClockSettingConfig-> McuClockSettingConfig _0:

->McuCgm2ClockMux2:

- CGM2 Clock Mux2 Source = PERIPH_PLL_PHI5_CLK //不是 SERDES_1_XPCS_1_TX
- Clock Mux2 Divider0 Frequency (PFE_MAC_1_TX_CLK, GMAC_1_TX_CLK, REC_CLK) (dynamic range)= 1.25E8 //时钟不变

->McuCgm2ClockMux5:

- CGM2 Clock Mux5 Source = PFE_MAC_1_EXT_RX_CLK//不是 SERDES_1_XPCS_1_CDR
- Clock Mux5 Frequency (PFE_MAC_1_RX_CLK, SEQ_CLK) (dynamic range) = 1.25E8 //时钟不变

而在 Linux 端的 ATF 代码中配置了 PFE_MAC1 和 GMAC0 的 IOMUX，在配置为 Uboot 初始化网口为 EMAC1 的情况下会初始化为 PFE_MAC1 的 IOMUX，相关其它修改参考以下 Linux 端修改说明 。

3.6 为配合 Linux 做 PFE 初始化，去掉 MCU/PORT 初始化

修改\PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2\example_application\Makefile:

```
# The CONFIGURE_SOC should be equal to FALSE when SLAVE_DRIVER=TRUE
SLAVE_DRIVER ?= FALSE
ifeq ($(SLAVE_DRIVER), TRUE)
  CONFIGURE_SOC ?= FALSE
else
  CONFIGURE_SOC ?= FALSE #TRUE 就算是Master工程，也把初始化去掉。
endif
```

代码说明如下：

```
Std_ReturnType SampleAppInitTask(void)
{
/*********************************************************************************/
/* PORT                                                                          */
/*********************************************************************************/
#ifdef CONFIGURE_SOC
  Port_Init(NULL_PTR);
#endif /* CONFIGURE_SOC */

/*********************************************************************************/
/* MCU                                                                           */
/*********************************************************************************/
```

```
#if defined(CONFIGURE_SOC)
  Mcu_Init(NULL_PTR);
  Mcu_InitClock(McuClockSettingConfig_0);
  while ( MCU_PLL_LOCKED != Mcu_GetPllStatus() ){}
  Mcu_DistributePllClock();
  Mcu_SetMode(McuModeSettingConf_0);
#endif /* CONFIGURE_SOC */
```

3.7 为配合 Linux 做 PFE 初始化，去掉 Eth_43_PFE_PreInit(NULL_PTR);

Eth_43_PFE_PreInit 要求在 PFE CLOCK 初始化之前配置，而本 Demo 是使用 Linux 的 uboot 来配置，所以在 MCAL 侧建议去掉：(实际上因为 Uboot 配置为 emac0/1/2=sgmii,sgmii,rgmii，所以如果 Mcal 的配置与 Uboot 的不冲突，则不去掉代码执行也没有问题)：

…\PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2\example_application\src\sample_app_ethswt_initialization.c:

```
Std_ReturnType SampleAppInitTask(void)
{
/*************************************************************************/
/* PFE interfaces initialization                                         */
/*************************************************************************/
  //  Eth_43_PFE_PreInit(NULL_PTR);
```

3.8 Lauterbach 脚本修改

本 Demo 需要等待 Linux 运行来初始化 PFE 的 partition reset 和 Clock，所以需要 MCAL sample 晚一点执行，修改如下：

…\PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2\example_application\run_main_G3_REV1_1.cmm

```
; in case of linux slave, uncomment the following line to give some time to linux to boot
 wait 10s
```

注意脚本执行后，lauterbach 会停止在 main 函数的入口处，需要尽快执行，不然 Linux 这边的 slave 驱动在等待一定时间 M master 驱动没有加载后，会报加载失败。

可以重新 insmod 一下：

```
lsmod
Module                  Size  Used by
pfeng_slave           208896  0
sja1110                20480  0
```

```
rmmod /lib/modules/5.10.120-rt70+g0b76731696c1/kernel/drivers/net/ethernet/nxp/pfe/pfeng-slave.ko
lsmod
Module                  Size  Used by
sja1110                20480  0
insmod /lib/modules/5.10.120-rt70+g0b76731696c1/kernel/drivers/net/ethernet/nxp/pfe/pfeng-slave.ko
root@s32g399ardb3:~# lsmod
Module                  Size  Used by
pfeng_slave           208896  0
sja1110                20480  0
```

4 A 核 Slave 驱动编译

关于 Linux 端的编译等相关背景知识，请参考 BSP 文档《S32G3_LinuxBSP_34.0_User_Manual.pdf》，或者《S32G_Kernel_BSP32_V*.pdf》，《S32G_Uboot_BSP*.pdf》，《S32G_ATF_BSP32_V*.pdf》(Johnli，中文版，在 publick community 上搜索，版本比较老)。

4.1 修改 DTS 为 Aux 接口

\BSP34\bsp34.tar\arch\arm64\boot\dts\freescale\s32g-pfe-slave.dtsi

```
        soc {
                pfe_slave: pfe_slave@46000020 {…
                    pfesl_netif0: ethernet@100 {
                     -    status = "okay";
                     +  status = "disabled";
                     }
                    pfesl_netif1: ethernet@101 {
                     -    status = "okay";
                     +  status = "disabled";
                     }
                    pfesl_netif2: ethernet@102 {
                     -    status = "okay";
                     +  status = "disabled";
                     }
                    pfesl_aux0: ethernet@101 {
                     -    status = " disabled ";
                     +  status = " okay ";
                     }
```

\BSP34 \arch\arm64\boot\dts\freescale\s32g399a-rdb3.dts

```
#include "s32g399a-rdb3.dtsi"
+#include "s32g-pfe-slave.dtsi"
```

4.2 修改 ATF 默认使用的 PIT

PFE 的 MCAL 代码的：

```
/*********************************************************************************/
/* GPT                                                                           */
/*********************************************************************************/
  SampleAppGptInit();
Std_ReturnType SampleAppGptInit(void)
{
  Gpt_Init(NULL_PTR);

  /* Enable interrupt and start PIT timer channel used for switching tasks */
  Gpt_EnableNotification(SWITCH_TASK_GPT_CHANNEL);
  Gpt_StartTimer(SWITCH_TASK_GPT_CHANNEL, 400000UL);

  return(E_OK);
}
```

默认是使用 PIT0，(请参考 EB 的 GPT 配置)，而 BSP34 的 ATF 代码也使用了 PIT0，会导致冲突，所以需要将其修改为 PIT1，patch 如下：

```
diff --git a/fdts/s32cc.dtsi b/fdts/s32cc.dtsi
index f07e92e92..bb7b837e2 100644
--- a/fdts/s32cc.dtsi
+++ b/fdts/s32cc.dtsi
@@ -20,7 +20,7 @@

        chosen {
                stdout-path = "serial0:115200n8";
-               tick-timer = &pit0;
+               tick-timer = &pit1;
        };

  pit0: pit@40188000 {
                        compatible = "nxp,s32cc-pit";
```

```
                reg = <0x0 0x40188000 0x0 0x3000>;
                interrupts = <GIC_SPI 53 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&clks S32GEN1_SCMI_CLK_PIT_MODULE>;
                clock-names = "pit";
-               status = "okay";
+               status = "disabled";
        };
```

4.3 Yocto 的编译方法

在编译 Yocto 工程时，在执行完：

```
source nxp-setup-alb.sh -m s32g399ardb3
```

的 conf/local.conf 中增加：

```
DISTRO_FEATURES_append += " pfe-slave"
```

然后重新编译 Yocto 工程。编译完成默认生成的 pfe slave 驱动模块在：

/lib/modules/5.10.120-rt70+g0b76731696c1/kernel/drivers/net/ethernet/nxp/pfe/pfeng-slave.ko

注意：

在本文撰写的过程中，NXP 正在进行 git 服务器从 codeaurora 到 github 的迁移，所以 BSP34 的 Yocto 编译会失败，需要等待 BSP34.1 的 release 来修正 manifest 中的 git 服务器地址。

4.4 Standlone 编译方法

如下单独编译 Linux Slave 驱动的办法：

```
git clone https://github.com/nxp-auto-linux/pfeng/
cd pfeng/
git status
On branch master
Your branch is up to date with 'origin/master'.
git tag
BLN_PFE-DRV_S32G_A53_LNX_1.2.0
BLN_PFE-DRV_S32G_A53_LNX_1.3.0
BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.0
BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.2
BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.2_HF1
BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.2_RC1
```

BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.3

BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.4

BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.4_CD1_FORD

BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.5

BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.6

BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.7

BLN_PFE-DRV_S32G_A53_LNX_RTM_1.0.0

PFE_S32G_A53_LNX_BETA_0.9.1_RC1

PFE_S32G_A53_LNX_EAR_0.8.0

PFE_S32G_A53_LNX_pre-EAR_0.4.1

PFE_S32G_A53_LNX_pre-EAR_0.4.3

TEST-PFE_S32G_A53_LNX_BETA_0.9.0

git checkout BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.7

Note: checking out 'BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.7'.

git status

HEAD detached at BLN_PFE-DRV_S32G_A53_LNX_BETA_0.9.7

nothing to commit, working tree clean

参考文档 pfeng/doc/PFE_S32G_A53_LNX_UserManual.pdf

3.1.2 Variant 2: Standalone build

For how to compiling PFE driver for standalone.

关于如何编译 slave 驱动：

make PFE_CFG_MULTI_INSTANCE_SUPPORT=1 PFE_CFG_PFE_MASTER=0 KERNELDIR=<path to kernel> PLATFORM=aarch64-linux-gnu all

然后启动后，将编译生成 的 pfe-slave.ko 放到 /lib/modules/5.10.120-rt70+g0b76731696c1/kernel/drivers/net/ethernet/nxp/pfe/中，将之前的 pfe.ko 删除掉。之后，启动板子，进入：

/lib/modules/5.10.120-rt70+g0b76731696c1/kernel/drivers/net/ethernet/nxp/pfe

执行命令：

depmod -a

之后检查：

pwd

/lib/modules/5.10.120-rt70+g0b76731696c1

vi modules.alias

alias of:N*T*Cnxp,s32g-pfe-slaveC* pfeng_slave

alias of:N*T*Cnxp,s32g-pfe-slave pfeng_slave

注意：

在本文撰写的过程中，NXP 在进行 git 服务器从 codeaurora 到 github 的迁移时，丢失掉了 BLN_PFE-DRV_S32G_A53_LNX_RTM_1.1.0 的 tag，此问题已经上报。

4.5 Uboot 启动配置

目前默认 uboot 配置为：

=> pri

ethact=eth_pfeng //说明 Uboot 默认使用 PFE EMAC，不是 GMAC

hwconfig=pcie0:mode=rc,clock=ext;pcie1:mode=sgmii,clock=ext,fmhz=125,xpcs_mode=2G5 //xpcs_mode=both 会导致本 demo 失败，未查。

pfe1addr=00:01:be:be:ef:22

pfe2addr=00:01:be:be:ef:33

pfeaddr=00:01:be:be:ef:11

pfeng_mode=enable,sgmii,sgmii,sgmii //默认 emac1 配置为 sgmii

pfengemac=0 //默认 pfe init 只针对 emac0，但是默认状态下所有 emac 接口为 sgmii

Uboot 代码说明：

\drivers\net\pfeng\pfeng_cmd.c

```
static u32 emac_intf[PFENG_EMACS_COUNT] = {
#if CONFIG_IS_ENABLED(NXP_S32GRDB_BOARD)
    PHY_INTERFACE_MODE_SGMII, /* SJA1110A */
    PHY_INTERFACE_MODE_SGMII, /* ARQ107/ARQ113 */
    PHY_INTERFACE_MODE_RGMII  /* KSZ9031 */
pfeng_set_emacs_from_env
int pfeng_set_emacs_from_env(char *env_mode)
{…
    for (i = 0; i < PFENG_EMACS_COUNT; i++)
            if (intf[i] > -1)
                    pfeng_cfg_emac_set_interface(i, intf[i]);
```

```
    /* set INTF_SEL */
    writel((pfeng_intf_to_s32g(emac_intf[2]) << 8) |
                (pfeng_intf_to_s32g(emac_intf[1]) << 4) |
                (pfeng_intf_to_s32g(emac_intf[0])),
                S32G_PFE_EMACS_INTF_SEL);


    return 0;
}
```

所以 Uboot 会默认将 emac0/emac1/emac2 根据 pfengemac 变量，初始化为 sgmii/sgmii/rgmii：相当于 Mcal 的 Eth_43_PFE_PreInit(NULL_PTR)。

- 如果不考虑外部 switcher，则默认情况下外部的 emac 配置为 emac1-sgmii-p5。注意。

Eth_43_PFE_PreInit 默认情况下是配置为 SGMII 接口的，所以就算 pfengemac=0 配置不正确，也不影响，可以修改为 pfengemac=1。

- 如果 Uboot 修改为：

```
pfeng_mode=enable,sgmii,sgmii,rgmii
pfengemac=2
```

则外部的 emac 配置为 emac2-rgmii-p3b。但是需要在 MCAL 中增加 emac2 相关 EthCtrl，在 uboot 中配置 serdes0，本 Demo 不做考虑。

- 如果 Uboot 修改为：

```
pfeng_mode=enable,none,rgmii,none      //不将 emac0/2 关掉会导致 demo 不工作，原因未查。
pfengemac=1
static u32 emac_intf[PFENG_EMACS_COUNT] = {
#if CONFIG_IS_ENABLED(NXP_S32GRDB_BOARD)
    PHY_INTERFACE_MODE_SGMII, /* SJA1110A */
    PHY_INTERFACE_MODE_RGMII  /* KSZ9031 */    //PHY_INTERFACE_MODE_SGMII, /*
ARQ107/ARQ113 */
    PHY_INTERFACE_MODE_RGMII  /* KSZ9031 */
```

则外部的 emac 配置为 emac1-rgmii-p3a。uboot 中检查 pfe 状态为：

```
=>    pfeng info
PFE mode: enable
 emac0: none emac1: rgmii emac2: none
 fw: 's32g_pfe_class.fw' on mmc@0:1
```

4.6 准备 Linux 镜像

本文采用下载默认 Linux Demo 镜像，替换掉之中的 DTB/PFE 驱动/Uboot 配置的方法如下：

1. 烧写 Demo 镜像

```
sudo dd if= fsl-image-base-s32g274ardb2.sdcard of=/dev/sd<partition> bs=1M conv=fsync
```

2. Standalone 编译好 PFE slave 驱动模块后，放入到默认目录，并将其中的 master 驱动模块去掉

```
pwd
/media/vmuser/fsl-image-auto-s/lib/modules/5.10.120-rt70+g0b76731696c1/kernel/drivers/net/ethernet/nxp/pfe
sudo mv pfeng.ko pfeng.ko.back
sudo cp /mnt/hgfs/share_folder/bsp34/p
feng-slave.ko .
sudo depmod -a
```

3. Standalone 编译 DTB 后，替换掉原生的 DTB

```
pwd
/media/vmuser/boot_s32g39
mv s32g399a-rdb3.dtb s32g399a-rdb3.dtb.bak
cp /mnt/hgfs/share_folder/bsp34/s32g399a-rdb3.dtb .
```

4. 启动后修改 uboot 参数：
如之前说明。

5 M Master/A Slave Demo 测试

以 VLAN L2 网桥模式为例：此处考虑两个 emac 的测试，分别为：

- emac1-sgmii-p5(默认配置)。
- emac1-rgmii-p3a。

5.1 PFE_EMAC1(SGMII)测试过程

PFE Master 发包测试如下：

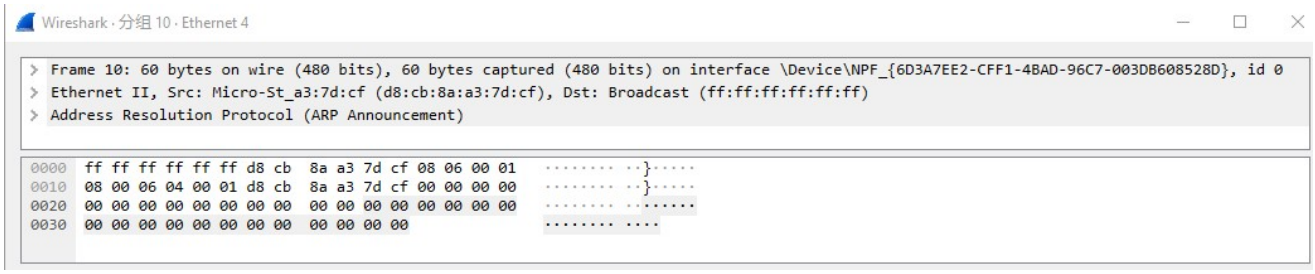
1. 将 RDB3 板子设置为 Sdcard 启动模式，上电后 Linux 调试串口会打印出 Linux 启动消息
2. 运行 lauterbach Trace32，打开脚本: …\
PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2\example_application\

run_slave_app_G3_REV1_1.cmm, 点击 Do 运行，这个时候 S32G3 会重启，然后 Linux 重新运行，10S 后 MCAL 程序会停在 main 函数入口,立即点击 Go 运行程序，要不然 Linux 端 Slave 驱动加载会超时失败。正常情况下 Linux 端打印为：

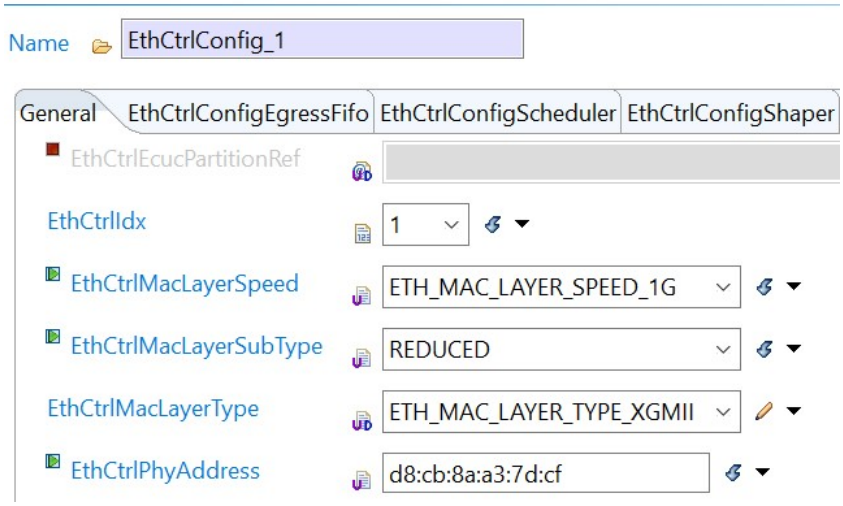
```
root@s32g399ardb3:~# dmesg |grep pfe
[    0.000000] OF: reserved mem: initialized node pfebufs@83200000, compatible id shared-dma-pool
[    6.465203] pfeng-slave 46000000.pfe_slave: PFEng ethernet driver loading ...
[    6.465224] pfeng-slave 46000000.pfe_slave: Version: RTM 1.1.0
[    6.465228] pfeng-slave 46000000.pfe_slave: Driver commit hash: M4_DRIVER_COMMIT_HASH
[    6.465233] pfeng-slave 46000000.pfe_slave: Multi instance support: SLAVE/mdetect=on
[    6.465237] pfeng-slave 46000000.pfe_slave: Compiled by: 10.2.0
[    6.465507] pfeng-slave 46000000.pfe_slave: Wait for PFE controller UP ...
[    6.465531] pfeng-slave 46000000.pfe_slave: PFE controller UP detected
[    6.465538] pfeng-slave 46000000.pfe_slave: Cbus addr 0x46000000 size 0x1000000
[    6.465574] pfeng-slave 46000000.pfe_slave: irq 'hif3' : 76
[    6.465581] pfeng-slave 46000000.pfe_slave: HIF channels mask: 0x0008
[    6.465587] pfeng-slave 46000000.pfe_slave: IHC channel: 3
[    6.465592] pfeng-slave 46000000.pfe_slave: MASTER IHC channel: 0
[    6.465603] pfeng-slave 46000000.pfe_slave: netif name: aux0sl
[    6.465609] pfeng-slave 46000000.pfe_slave: DT mac addr: 00:04:9f:be:ff:80
[    6.465616] pfeng-slave 46000000.pfe_slave: netif(aux0sl) mode: aux
[    6.465624] pfeng-slave 46000000.pfe_slave: netif(aux0sl) HIFs: count 1 map 08
[    6.472892] pfeng-slave 46000000.pfe_slave: PFE CBUS p0x(____ptrval____) mapped @
v0xffffffc015000000
[    6.472908] pfeng-slave 46000000.pfe_slave: HW version 0x101
[    6.472913] pfeng-slave 46000000.pfe_slave: Wait for Master UP ...
[    6.472918] pfeng-slave 46000000.pfe_slave: Detected Master UP
[    6.473028] pfeng-slave 46000000.pfe_slave: HIF0 not configured, skipped
[    6.473034] pfeng-slave 46000000.pfe_slave: HIF1 not configured, skipped
[    6.473039] pfeng-slave 46000000.pfe_slave: HIF2 not configured, skipped
[    6.473261] pfeng-slave 46000000.pfe_slave: HIF3 enabled
[    6.473271] pfeng-slave 46000000.pfe_slave: HIF3 started
[    6.473286] pfeng-slave 46000000.pfe_slave: IDEX-slave
[    6.473298] pfeng-slave 46000000.pfe_slave: IHC client registered
[    6.473303] pfeng-slave 46000000.pfe_slave: IDEX RPC installed
[    6.473744] pfeng-slave 46000000.pfe_slave aux0sl: checksum offload not possible for AUX interface
```

```
[    6.473997] pfeng-slave 46000000.pfe_slave aux0sl: registered
[    6.474021] pfeng-slave 46000000.pfe_slave aux0sl: AUX subscribe to HIF3
[    6.475356] pfeng-slave 46000000.pfe_slave aux0sl: checksum offload not possible for AUX interface
[    6.484741] pfeng-slave 46000000.pfe_slave aux0sl: Enable HIF3
```

3. 将 S32G3 RDB3 的 P5 接口连接到 PC 上，打开 wireshark,选择调试连接的网口。
4. 在 trace32 上点击 Go 运行程序，然后就可以看到 wireshark 抓到的包如下：



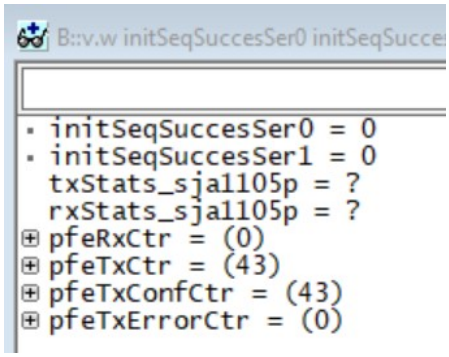
可见源 MAC 地址为 EMAC1 配置的：



5. Lauterbach 脚本 run_main_G3.cmm 默认打开的观察变量：

```
v.w initSeqSuccesSer0 initSeqSuccesSer1 txStats_sja1105p rxStats_sja1105p pfeRxCtr pfeTxCtr pfeTxConfCtr pfeTxErrorCtr
```

检查如下：pfeTxCtrt 和 pfeTxConfCtr 在不断计数：



6. Linux 端在 ipconfig up 好 AUX 接口后，

```
root@s32g399ardb3:~# ifconfig aux0sl up
root@s32g399ardb3:~# ifconfig
aux0sl: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::204:9fff:febe:ff80  prefixlen 64  scopeid 0x20<link>
        ether 00:04:9f:be:ff:80  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 1062 (1.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device memory 0x46000000-46ffffff
```

7. 运行 fci 命令配置好网桥，将 MCAL 的 HIF0，Linux 的 HIF3 和 EMAC1 加入到一个网桥中。

```
libfci_cli bd-update --vlan=1 --uh=FORWARD --um=FLOOD --mh=FORWARD --mm=FLOOD
libfci_cli bd-insif --vlan=1 -i emac1 --tag=OFF
libfci_cli bd-insif --vlan=1 -i hif3 --tag=OFF
libfci_cli bd-insif --vlan=1 -i hif0 --tag=OFF
libfci_cli phyif-update -i emac1 --mode=VLAN_BRIDGE -E --promisc ON
libfci_cli phyif-update -i hif3 --mode=VLAN_BRIDGE -E --promisc ON
libfci_cli phyif-update -i hif0 --mode=VLAN_BRIDGE -E --promisc ON
```

8. 可以使用 TCPdump 查看收到了从 MCAL 发送过来的广播包

```
tcpdump -i aux0sl
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on aux0sl, link-type EN10MB (Ethernet), capture size 262144 bytes
17:30:07.956179 ARP, Request who-has 0.0.0.0 tell 0.0.0.0, length 28
```

9. PC 端 ping S32G Linux 或相反

ifconfig aux0sl 192.168.0.2

然后 PC 配置为同一网端，然后可以从 PC 或 S32G3 Linux 端互相 ping。

5.2 PFE_EMAC1(RGMII)测试过程

PFE_EMAC1-RGMII-P3A 的测试过程与上类似，只是注意一下：

- 需要修改 Linux 端 uboot 代码。

如之前说明，修改 Uboot 源代码及 Uboot 配置参数。

- 接口使用 P3A。

6 M Master/A Slave Demo FCI 配置说明

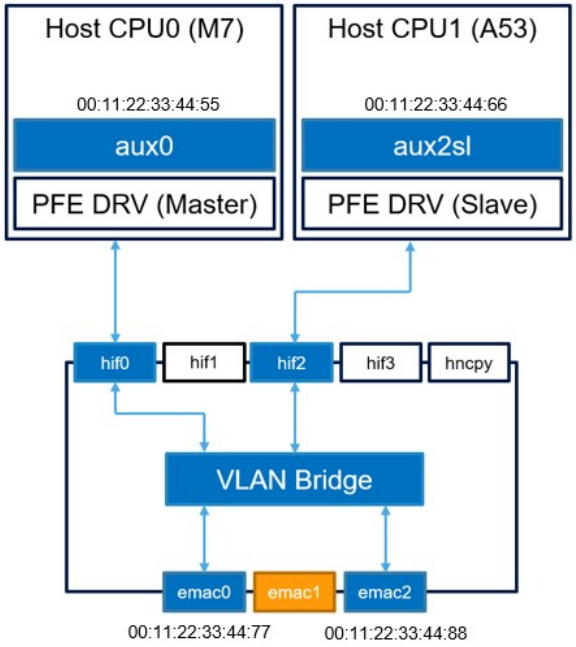
以下以 M Master/A Slave 为例说明 FCI 配置，考虑：

6.1 VLAN L2 网桥

- 以太网交换机。
- 可以使用 VLAN 将接口分配给网桥域
- 使用 MAC 地址来路由流量
- 可以采用自学习或静态分配 MAC 地址的 L2 Bridge
- 可识别多种 BD 类型：

1. 默认 BD：此网桥域的出厂默认 VLAN ID 为 1。此域用于处理入口帧，这些入口帧要么是具有等于默认 BD 的 VLAN ID 的 VLAN ID，或是没有 VLAN ID 的以太网帧。
2. 回退 BD：此域用于处理具有未知 VLAN 标记的入口帧。Unknown VLAN tag 表示 VLAN tag 不匹配任何现有的标准 BD 或默认 BD。
3. Standard BD：标准的用户自定义网桥域。 由 VLAN 感知网桥使用。 这些 BD 处理入口帧，这些帧具有与 BD 的 VLAN ID 相匹配的 VLAN 标记。

例子： 1 个网桥域，VLAN_ID=10，VLAN 10 中的物理接口：hif0、hif2、emac0、emac2。



FCI 命令如下：

方法 1：

新建一个 VLAN 桥接域，VLAN ID 为 10

libfci_cli bd-add –vlan 10

为网桥域配置单播/多播命中/未命中操作

libfci cli bd-update --vlan 10 --ucast-hit FORWARD --ucast-miss FLOOD --mcast-hit FORWARD --mcast-miss FLOOD

将 EMAC 和 HIF 添加为禁用 VLAN 标记的 VLAN 桥接域的成员

libfci_cli bd-insif --vlan 10 --i emac0 --tag OFF

libfci_cli bd-insif --vlan 10 --i emac2 --tag OFF

libfci_cli bd-insif --vlan 10 --i hif0 --tag OFF

libfci_cli bd-insif --vlan 10 --i hif2 --tag OFF

根据“VLAN_BRIDGE”模式配置物理接口，启用流量混杂，阻塞状态模式“NORMAL”（启用 MAC 地址学习）

libfci_cli phyif-update --i emac0 -E --promisc ON --mode VLAN_BRIDGE --bs NORMAL

libfci_cli phyif-update --i emac2 -E --promisc ON --mode VLAN_BRIDGE --bs NORMAL

libfci_cli phyif-update --i hif0 -E --promisc ON --mode VLAN_BRIDGE --bs NORMAL

libfci_cli phyif-update --i hif2 -E --promisc ON --mode VLAN_BRIDGE --bs NORMAL

方法 2：

新建一个 VLAN 桥接域，VLAN ID 为 10

```
libfci_cli bd-add –vlan 10
```

为网桥域配置单播/多播命中/未命中操作

```
libfci_cli bd-update --vlan 10 --ucast-hit FORWARD --ucast-miss FLOOD --mcast-hit FORWARD --mcast-miss FLOOD
```

将 EMAC 和 HIF 添加为禁用 VLAN 标记的 VLAN 桥接域的成员

```
libfci_cli bd-insif --vlan 10 --i emac0 --tag OFF
libfci_cli bd-insif --vlan 10 --i emac2 --tag OFF
libfci_cli bd-insif --vlan 10 --i hif0 --tag OFF
libfci_cli bd-insif --vlan 10 --i hif2 --tag OFF
```

创建新的静态 MAC 表条目并设置出口端口

```
libfci_cli bd-stent-add –vlan 10 –mac 00:11:22:33:44:55
libfci_cli bd-stent-update –vlan 10 –mac 00:11:22:33:44:55 –egress hif0
libfci_cli bd-stent-add –vlan 10 –mac 00:11:22:33:44:66
libfci_cli bd-stent-update –vlan 10 –mac 00:11:22:33:44:66 –egress hif2
libfci_cli bd-stent-add –vlan 10 –mac 00:11:22:33:44:77
libfci_cli bd-stent-update –vlan 10 –mac 00:11:22:33:44:77 –egress emac0
libfci_cli bd-stent-add –vlan 10 –mac 00:11:22:33:44:88
libfci_cli bd-stent-update –vlan 10 –mac 00:11:22:33:44:88 –egress emac2
```

根据“VLAN_BRIDGE”模式配置物理接口，启用流量混杂，阻塞状态模式“FW_ONLY”（禁用 MAC 地址学习）

```
libfci_cli phyif-update --i emac0 -E --promisc ON --mode VLAN_BRIDGE --bs FW_ONLY
libfci_cli phyif-update --i emac2 -E --promisc ON --mode VLAN_BRIDGE --bs FW_ONLY
libfci_cli phyif-update --i hif0 -E --promisc ON --mode VLAN_BRIDGE --bs FW_ONLY
libfci_cli phyif-update --i hif2 -E --promisc ON --mode VLAN_BRIDGE --bs FW_ONLY
```

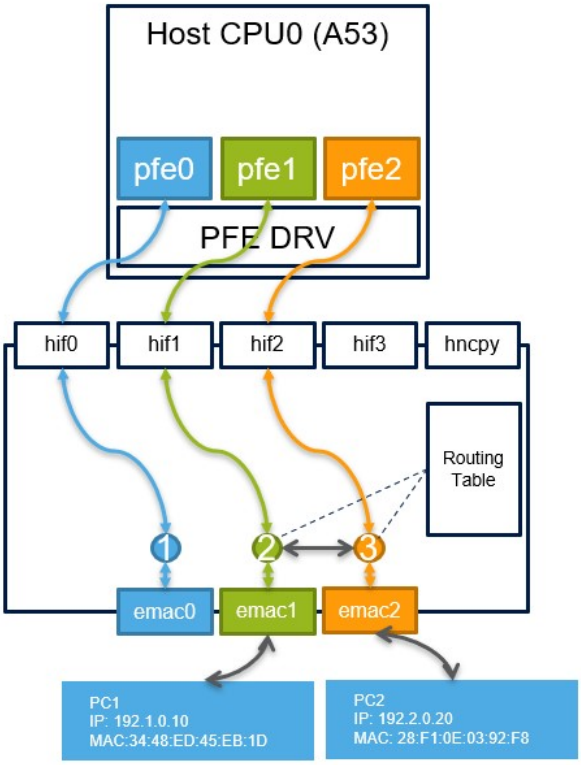
6.2 IPV4/V6 路由器

主机应用程序可以将物理接口的操作模式更改为 IP 路由器，并使用 FCI API 将路由规则指定到路由表中。

例子：

- EMAC1 和 EMAC2 之间的快速路径路由和主机 CPU 内的慢速路径路由。

- 配置后，路由点 2 和 3 将通过与路由表条目匹配的 EMAC1 或 EMAC2 接收的流量转发到所需的目标接口（EMAC2 或 EMAC1），同时将其余流量发送到默认接口 (CPU0)，其中主机软件可以执行自定义的慢速路径路由。



为物理接口创建新路由

libfci_cli route-add --route 0 --ip4 --dst-mac 34-48-ED-45-EB-1D --i emac1

libfci_cli route-add --route 1 --ip4 --dst-mac 28-F1-0E-03-92-F8 --i emac2

创建新的简单 contrack

libfci_cli cntk-add --proto ICMP --sip 192.1.0.10 --dip 192.2.0.20 --sport 0 --dport 0 --route 1 --no-reply

libfci_cli cntk-add --proto ICMP --sip 192.2.0.20 --dip 192.1.0.10 --sport 0 --dport 0 --route 0 --no-reply

打印路线和跟踪

libfci_cli route-print

libfci_cli cntk-print

将物理接口配置为：“ROUTER”模式，禁用流量混杂

libfci_cli phyif-update -i emac1 --mode=ROUTER -E --promisc OFF libfci_cli phyif-update -i emac2 --mode=ROUTER -E --promisc OFF

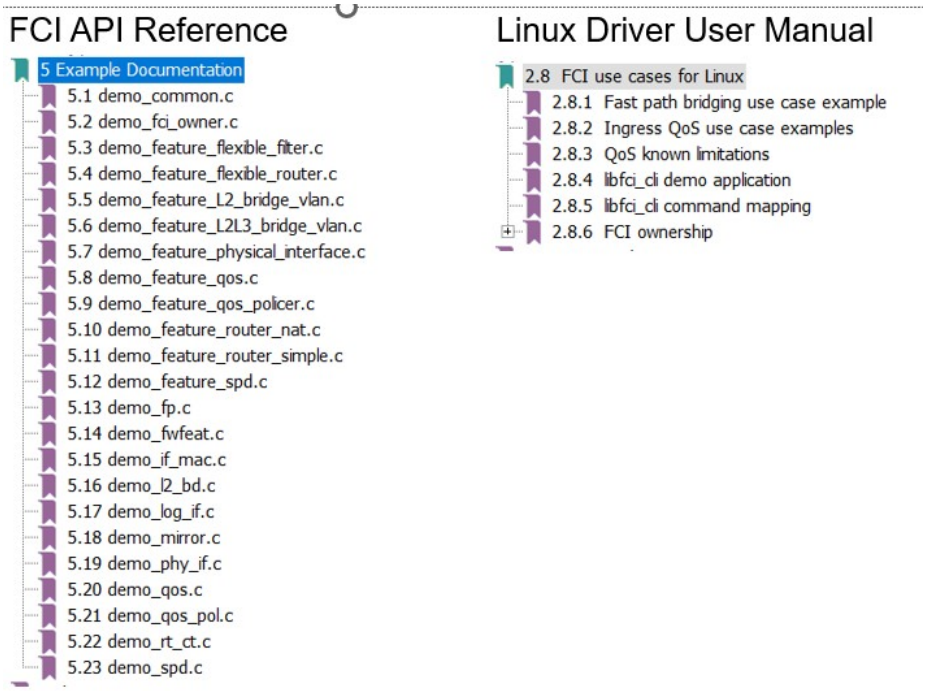
```
root@s32g399ardb3:~# libfci_cli route-print
DISCLAIMER: This is a DEMO application. It is not part of the production code deliverables.
| route      | IP   | src-mac           | dst-mac           | egress interface |
|============|======|===================|===================|==================|
|          0 | IPv4 | 00:01:be:be:ef:22 | 34:48:ed:45:eb:1d | emac1            |
|          1 | IPv4 | 00:01:be:be:ef:33 | 28:f1:0e:03:92:f8 | emac2            |
Command successfully executed.
root@s32g399ardb3:~# libfci_cli cntk-print
DISCLAIMER: This is a DEMO application. It is not part of the production code deliverables.
conntrack:
    proto:   1 (ICMP)
    flags:   [ TTL_DECR NO_REPLY ] ; [ --- ]
    orig:    src=192.1.0.10         dst=192.2.0.20         sport=0        dport=0        vlan=0        route=1
    reply: r-src=192.2.0.20       r-dst=192.1.0.10       r-sport=0      r-dport=0      r-vlan=0      r-route=0
    stats:   orig_hit: 0 orig_hit_bytes: 0   reply_hit: 0 reply_hit_bytes: 0
conntrack:
    proto:   1 (ICMP)
    flags:   [ TTL_DECR NO_REPLY ] ; [ --- ]
    orig:    src=192.2.0.20         dst=192.1.0.10         sport=0        dport=0        vlan=0        route=0
    reply: r-src=192.1.0.10       r-dst=192.2.0.20       r-sport=0      r-dport=0      r-vlan=0      r-route=0
    stats:   orig_hit: 0 orig_hit_bytes: 0   reply_hit: 0 reply_hit_bytes: 0
Command successfully executed.
```

6.3 其它

其它如：

- flexible router
- flexible Parser
- L2L3 VLAN Bridge
- NAT
- Mirror
- Ingress/Egress QoS
- …

请参考：



7 使用 M 核来配置 FCI 的方法

7.1 M 核 FCI Message Proxy 说明

如上所说，Master/Slave 驱动是由 Master 负责来通过 FCI 接口配置 PFE 的，所以如果是在 Slave Linux 这边运行 FCI 命令，它实际是通过 RPC 调用把配置命令传递到 M Master 这边，然后再由 M Master 驱动来调用 FCI 接口配置进 PFE 的。如下代码：

中断 Callback 注册如下：

```
Main
|-> SampleAppInitTask
|   |-> SampleAppEthInit
|   |   |-> Eth_43_PFE_Init
|   |   |   |-> Eth_PFE_LLD_PlatformDrvPrepare
|   |   |   |   |->CreateHifDrv
|   |   |   |   |   |->pfe_idex_init(prHifDrv, rPlatformCfg.master_if, ptrPlatform->hif, &pfe_platform_idex_rpc_cbk, (void *)ptrPlatform, NULL_PTR))
```

```
@details     The callback will be called at any time when RPC request
*            will be received.
@param[in]   cbk Callback to be called
```

中断 Callback 函数 pfe_platform_idex_rpc_cbk 如下：

```
#if defined(PFE_CFG_FCI_ENABLE)
        case (uint32_t)PFE_PLATFORM_RPC_PFE_FCI_PROXY:
        {
  #endif /* PFE_CFG_FCI_ENABLE */
|-> fci_process_ipc_message // Process the FCI message
…
```

所以可以在此 callback 函数入口增加断点，来调试从 A Slave Linux 传过来的 FCI Message。

7.2 M 核主动配置 FCI 的方法

基于快速启动和功能安全的考虑，可能一些客户会考虑用功能安全和可以快速启动的 M 核来配置 PFE，所以希望是使用 M Master 驱动来配置 FCI，我们的 Master 工程也提供相关 sample 和接口，如下：

参考文档：PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2\example_application\readme.txt

```
3.2 Options for the make command
…
FCI_L2BR_TEST=TRUE
    This option can be used to enable FCI L2 bridge demo, see section 1.4.1
  FCI_RTABLE_TEST=TRUE
    This option can be used to enable FCI routing table demo, see section 1.4.1
```

所以在 make 时加上相应编译宏，可以打开 FCI 配置 sample，代码调用如下：

```
Main
|->
#if (defined(PFE_CFG_FCI_ENABLE) && (defined(FCI_L2BR_TEST) || defined(FCI_RTABLE_TEST)))
  /* FCI test */
  SampleAppFciTestTask();
#endif
```

| |->以下为 L2 网桥模式，他是把 HIF0/1, EMAC0/1 加入到一个网桥中：

对应的 Linux 端 FCI 命令请参考 7.1 节 VLAN L2 网桥。

```
#if defined(FCI_L2BR_TEST)
  /* Create bridge domain */
  if (VLAN_USED > 1U) /* For VLAN 0 and 1 it exists already */
  {
```

```
    if (!sample_fci_bd_create(cl, VLAN_USED))
    {
        NXP_LOG_ERROR("sample_fci_bd_create failed\n");
        return E_NOT_OK;
    }
  }

  /* Add interfaces to a bridge domain */
  if (!sample_fci_bd_add_if(cl, VLAN_USED, "emac0", FALSE))
  {
    NXP_LOG_ERROR("sample_fci_bd_add_if failed\n");
    return E_NOT_OK;
  }
  if (!sample_fci_bd_add_if(cl, VLAN_USED, "emac1", FALSE))
  {
    NXP_LOG_ERROR("sample_fci_bd_add_if failed\n");
    return E_NOT_OK;
  }
  if (!sample_fci_bd_add_if(cl, VLAN_USED, "hif0", FALSE))
  {
    NXP_LOG_ERROR("sample_fci_bd_add_if failed\n");
    return E_NOT_OK;
  }
  if (!sample_fci_bd_add_if(cl, VLAN_USED, "hif1", FALSE))
  {
    NXP_LOG_ERROR("sample_fci_bd_add_if failed\n");
    return E_NOT_OK;
  }

  /* Set bridge domain policy */
  if (!sample_fci_bd_set_policy(cl, VLAN_USED, 0, 1, 0, 1))
  {
    NXP_LOG_ERROR("sample_fci_bd_set_policy failed\n");
    return E_NOT_OK;
  }
```

```
    /* Set interfaces mode to bridge */
    sample_fci_phy_if_set_mode(cl, "emac0", FPP_IF_OP_VLAN_BRIDGE);
    sample_fci_phy_if_set_mode(cl, "emac1", FPP_IF_OP_VLAN_BRIDGE);
    sample_fci_phy_if_set_mode(cl, "hif0", FPP_IF_OP_VLAN_BRIDGE);
    sample_fci_phy_if_set_mode(cl, "hif1", FPP_IF_OP_VLAN_BRIDGE);

    /*    Enable interfaces */
    sample_fci_phy_if_enable(cl, "emac0");
    sample_fci_phy_if_enable(cl, "emac1");
    sample_fci_phy_if_enable(cl, "hif0");
    sample_fci_phy_if_enable(cl, "hif1");

    /* Turn on promisc mode on interfaces */
    sample_fci_phy_if_promisc_on(cl, "emac0");
    sample_fci_phy_if_promisc_on(cl, "emac1");
    sample_fci_phy_if_promisc_on(cl, "hif0");
    sample_fci_phy_if_promisc_on(cl, "hif1");
#endif
```

| |->

```
#if defined(FCI_RTABLE_TEST)
    /* Configure IPv4 router using bi-directional conntrack */
    sample_fci_setup_ipv4_router_bd(cl);
#endif
```

| | |->他是在 emac0/1 间创建了一个 router，Linux 命令参考 7.2 节，IPV4 路由器。

```
/*    Reset the router */
    sample_fci_router_reset(cl);

    /*    Create routes */
    sample_fci_router_register_ipv4_routes(cl);

    /*    Create bi-directional conntrack */
    sample_fci_router_register_bd_ipv4_conntrack(cl);
```

```
  /*    Set interface mode */
  sample_fci_phy_if_set_mode(cl, "emac0", FPP_IF_OP_ROUTER);
  sample_fci_phy_if_set_mode(cl, "emac1", FPP_IF_OP_ROUTER);

  /*    Enable interfaces */
  sample_fci_phy_if_enable(cl, "emac0");
  sample_fci_phy_if_enable(cl, "emac1");

  /* Catch messages from the FCI endpoint */
  fci_catch(cl);
```