S32G PFE Master/Slave Simple Demo Building

by John Li (nxa08200)

This article explains the process of building a Demo in a simple way to realize M-core PFE Master and A-core Slave on the S32G3 RDB3 board.

The software version is RTD4.0.0+PFE MCAL1.0.0 (built-in PFE FW 1.4.0)+(PFE FW 1.4.0 optional)+BSP34(PFE Linux Driver 1.1.0).

Please note that this article is a training and auxiliary document. This article is not a substitute for the official document. Please refer to the official document.

History	Comments	Author
V1	● Create the doc	John.Li
V2	● Translate to Eng.	John.Li

Contents

1 Required SW and Tools

SW/Tool	Name	Comments
PFE MCAL SW	PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2.zip	Include FW 1.4.0 in \example_application\pfe_firmware\s32g_pfe_class.c/h
PFE FW	PFE-FW_S32G_1.4.0.zip	optinal: 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	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(The default included PFE version is 1.1.0)	

NOTE: Versions match:

- \ 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 explantion

2.1 RDB3 PFE Network Topology

The serdes description in the network topology S32G3 chip reference manual:

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.

Then in the design on RDB3, only consider the case of EMAC0/1 from Serdes_1, as explained in the Linux BSP manual:

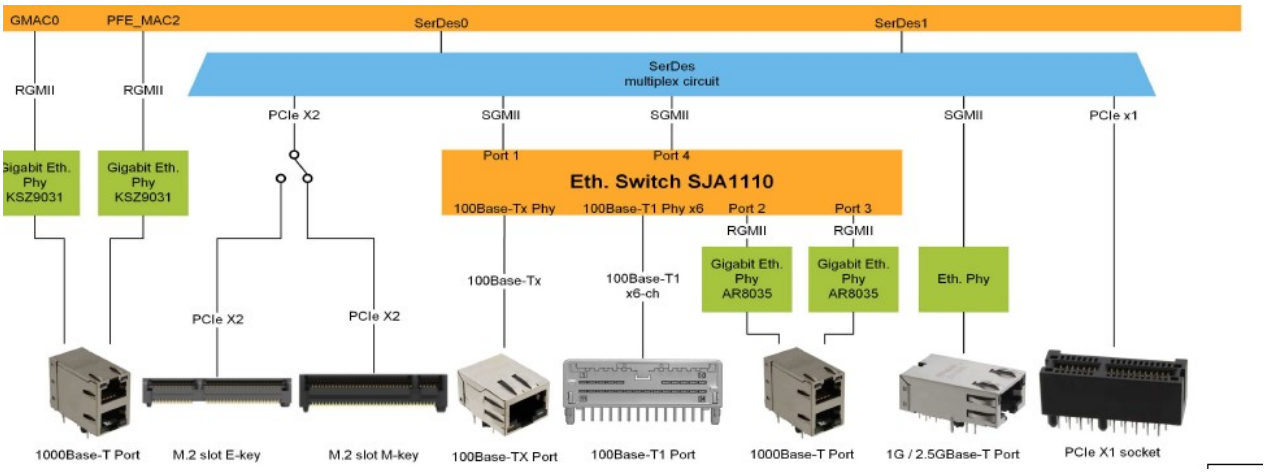
S32G399A RDB3 default port mapping in U-Boot:

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:

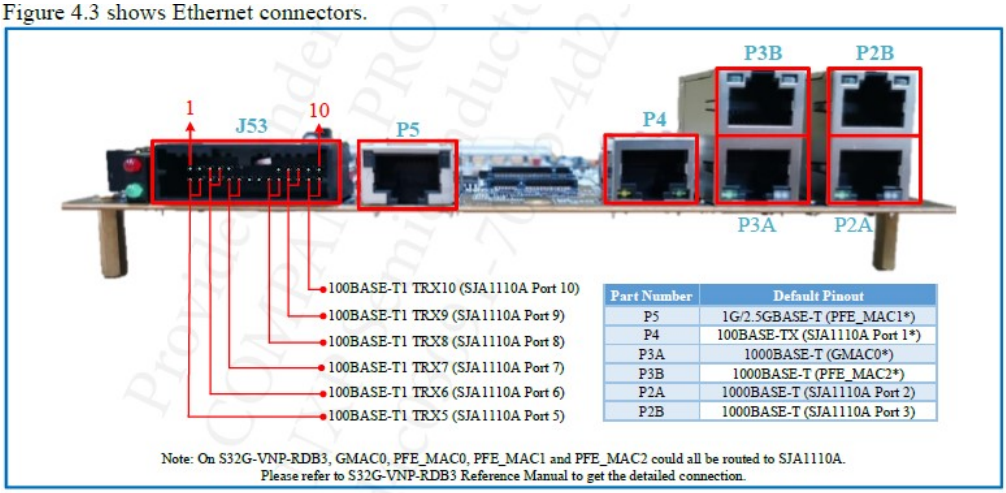
SoC port	interface	board port	Linux name	note
PFE_EMAC_0	SGMII(2.5G) to switch SJA1110A	P2, P4	`pfe0`	–
PFE_EMAC_1	SGMII(2.5G)	P5	`pfe1`	–
PFE_EMAC_2	RGMII	P3 top	`pfe2`	–
GMAC	RGMII	P3 bottom	`eth0`	–

Corresponding RDB3 hardware design:



And RDB3 user guide description:

Figure 4.3 shows Ethernet connectors.



Part Number	Default Pinout
P5	1G/2.5GBASE-T (PFE_MAC1*)
P4	100BASE-TX (SJA1110A Port 1*)
P3A	1000BASE-T (GMAC0*)
P3B	1000BASE-T (PFE_MAC2*)
P2A	1000BASE-T (SJA1110A Port 2)
P2B	1000BASE-T (SJA1110A Port 3)

Note: On S32G-VNP-RDB3, GMAC0, PFE_MAC0, PFE_MAC1 and PFE_MAC2 could all be routed to SJA1110A.
Please refer to S32G-VNP-RDB3 Reference Manual to get the detailed connection.

Then correspond to the description of MCAL:

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.

So note that the above description is not for RDB3. In fact, EMAC0 uses SGMII interface to connect to SJA1110 on RDB3.

EMAC1 is connected to RGMII port on processor board.

EMAC1 on RDB3 can be configured as SGMII, which comes out directly from P5, or as RGMII, which comes out from the P3A port used by GMAC0.

Note that the Master project of MCAL uses this design by default, and the three ethctrl devices are: EMAC0-SGMII, EMAC1-RGMII, AUX. interface.

Also note that when ATF is initialized, the RGMII interface of IOMUX of EMAC0/1/2 has been pre-configured.

Therefore, there is a certain conflict between the Linux software settings and the MCAL PFE master example project, which needs to be modified. If the influence of SJA1110 is not considered, if EMAC1 is configured as SGMII interface (Linux default), it is connected from P5 port; if it is configured as RGMII interface (MCAL default, Linux needs to be modified), it is connected from P3A.

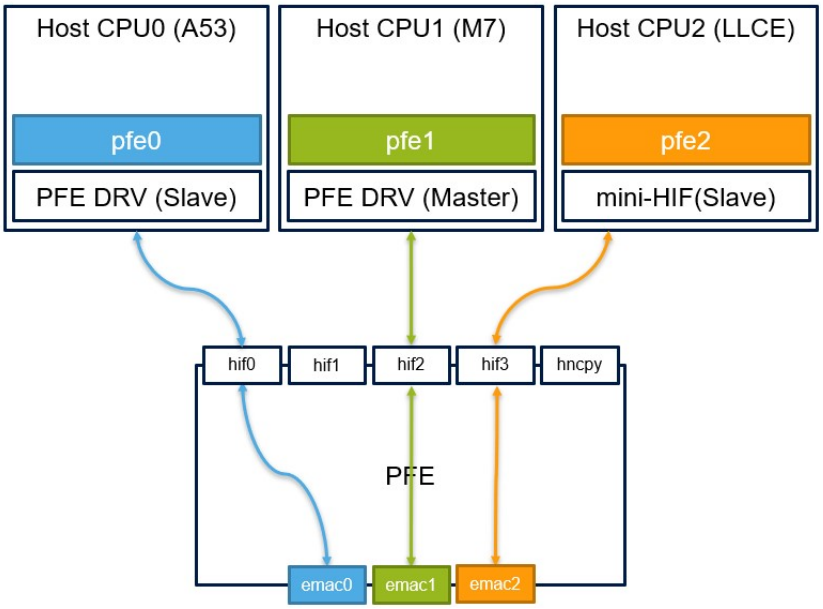
2.2 Master/Slave Demoe explanation

The PFE driver can run in the multi-instance mode in the same system, and multiple instances can be connected to the PFE through the HIF interface. The intended use case is to run a master instance within a host, and one or more slave instances from other hosts that have access to PFE, even from operating systems (OS) other than the host OS running the master instance.

The "master" instance implements the full PFE platform driver and performs general PFE hardware initialization and startup. The host driver has access to the full PFE register space.

A "slave" instance implements only part of the platform driver functionality.

Slave instances should not start initializing before the master instance is functioning properly.



2.3 This simple Demo explanation

In real products, a bootloader based on the M7_0 core is generally used to start the M and A cores. This bootloader is responsible for the initialization of all M core and A core resources, resolves resource conflicts between the M core and A core, and starts the M and A cores. So in theory, running the M PFE Master Mcal driver plus the A PFE Slave Linux driver also requires a bootloader. The reference document <<S32G_Bootloader_V*>>, (Johnli), can be searched on the public community.

This article discusses a simple method, which is:

- The S32G3 RDB3 board is configured to boot from the SD card, insert the SD card, and put the Linux image driven by the PFE SLAVE in it.
- After power-on, run the lauterbach debugging script of the PFE Master project: run_main_G3_REV1_1.cmm, this script will restart the entire S32G3.
- Then use the wait 10S operation in the script. At this time, Linux has started, and use the Uboot code to call ATF to complete the PFE-related pre-init, partition reset, and clock and pin initialization (as analyzed above, RGMII IOMUX of EMAC0~2 Already configured), then the Slave driver will wait for a while, wait for the MCAL Master driver to load, and continue to run the PFE Master MCAL code, and the Linux-side Slave driver will also be loaded correctly. Then you can test the whole M Master/A Slave Demo.

Summary: The above method is actually to complete the PFE-related hardware initialization work that the bootloader should do by Linux, so as to quickly build a demo, so that customers can use it as a standard reference for NXP release when doing real product development

3 M Core Master Driver Compilation Instructions

3.1 Install RTD_MCAL Driver

Double-click SW32_RTD_4.4_4.0.0_D2210.exe to install. The installation wizard will ask for the path of EB Tresos Studio (C:\EB\tresos\).

After the RTD is installed correctly, you will see the source code and related documents under the installation path (default: C:\NXP\SW32_RTD_4.4_4.0.0). And you will see a .link file in the links folder under the corresponding EB Tresos Studio installation path, which indicates that EB Tresos Studio has been associated with RTD. After opening EB Tresos Studio to create or import an RTD configuration project, it will be associated with the RTD path. If the EB path is not provided during installation, you can also manually create a file with the suffix .link in the corresponding path of EB, and write it into the RTD installation path just now.

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

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

Note that other .link files should be changed to backup names, for example, SW32G_RTD_4.4_3.0.2.link.bak, to prevent conflicts. For details of MCAL, please refer to the document <<S32G_RTD_MCAL_V*.pdf>>(John.li).

3.2 Install PFE_MCAL Driver

Copy \pfe\Driver PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2 \eclipse\plugins\ Eth_43_PFE_TS_T40D11M10I0R0 to C:\SW32_RTD_4.4_4.0.0 \eclipse\plugins.

3.3 Compile PFE master Project

The document PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2 \example_application\readme.txt explains the relevant situation of the PFE project, it is recommended to read.

Open 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 The project MasterProject_G3_REV1_1 (Check Copy projects into workspace).

Right-click on the project MasterProject_G3, select Properties->Resource->Location: C:\EB\tresos\workspace\MasterProject_G3_REV1_1. This is the project directory Configuration Port->Code Generation: Default Generation Path= ..\generate_G3_REV1_1. This is where the code is generated, so:

Right-click the project MasterProject_G3_REV1_1, select generate project, use EB to generate code, copy C:\EB\tresos\workspace\generate_G3_REV1_1* to PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2 \example_application

Modify compilation related files: 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 # compiler install folder
TRESOS_DIR ?= C:/EB/tresos  # EB install folder
PLUGINS_DIR ?= C:/NXP/SW32_RTD_4.4_4.0.0/eclipse/plugins  #RTD plugins install folder
```

Then modify the Makefile:

```
HW ?= S32G3 # default is G2, modify to G3
DEBUG ?= FALSE # The default is not modified, you can use the compilation parameters
COMPILER ?= GCC # The default is not modified, you can use the compilation parameters
```

In Cygwin, the command to compile MasterProject_G3:

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

Output image as follows:

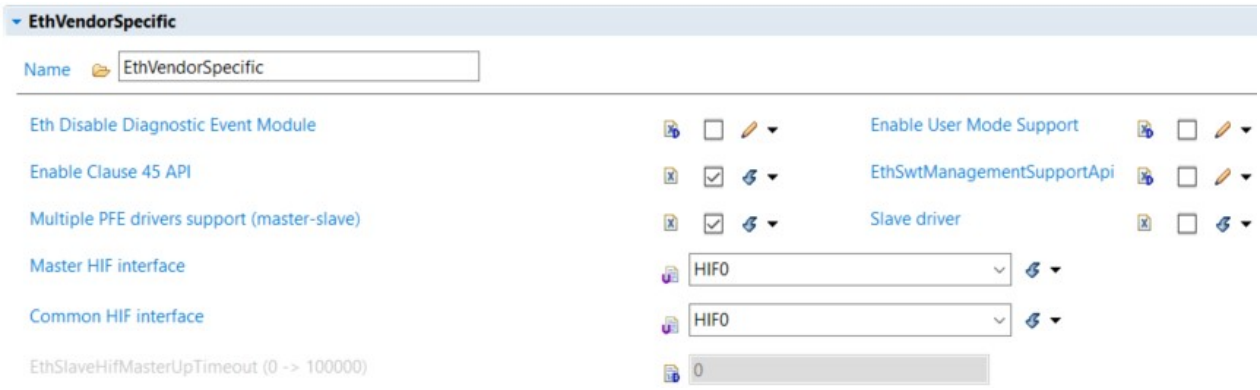
```
-o output_G3_REV1_1/main_G3_REV1_1.elf
```

3.4 PFE Master Project Explanation

1. EB Configuration:

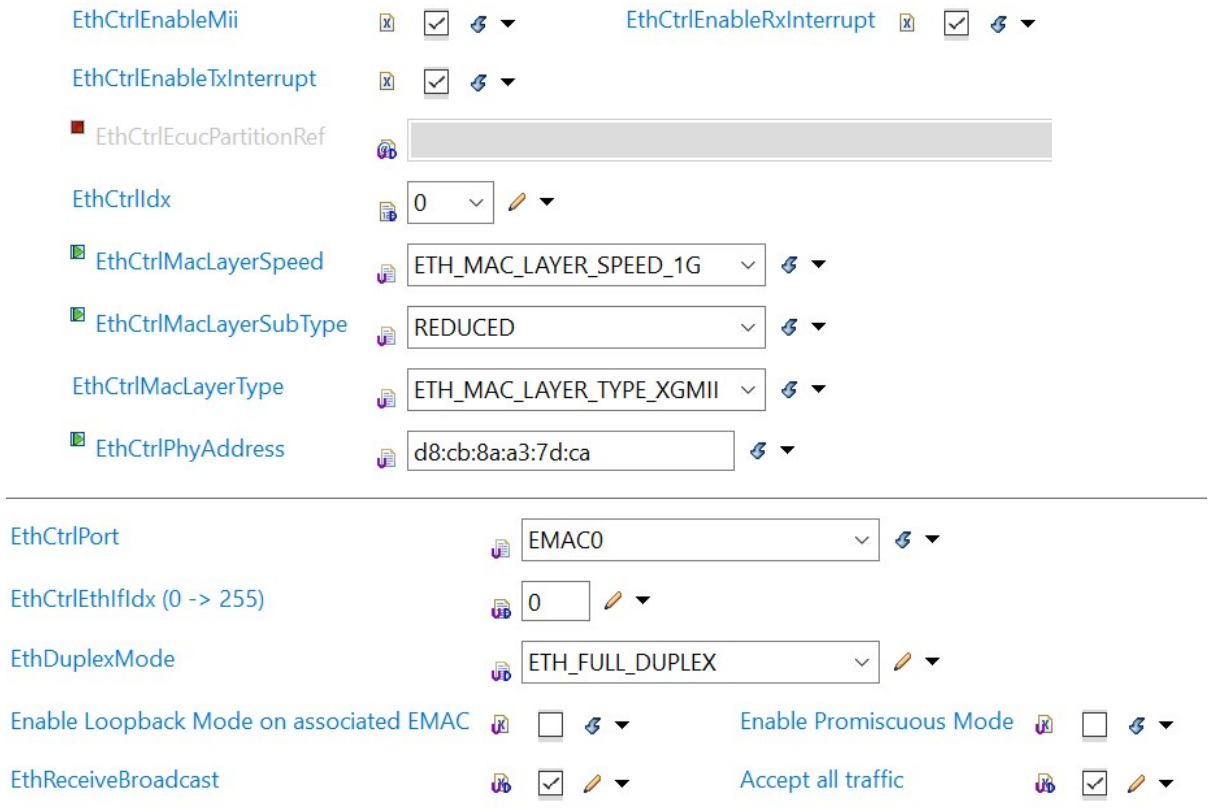
   MasterProject_G3_REV1_1->someId(…)->Eth_43_PFE->Eth_1(…)->General:

- 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: (Because eMAC0 needs external SJA1110 test, this demo does not use this port test, theoretically it can be deleted).

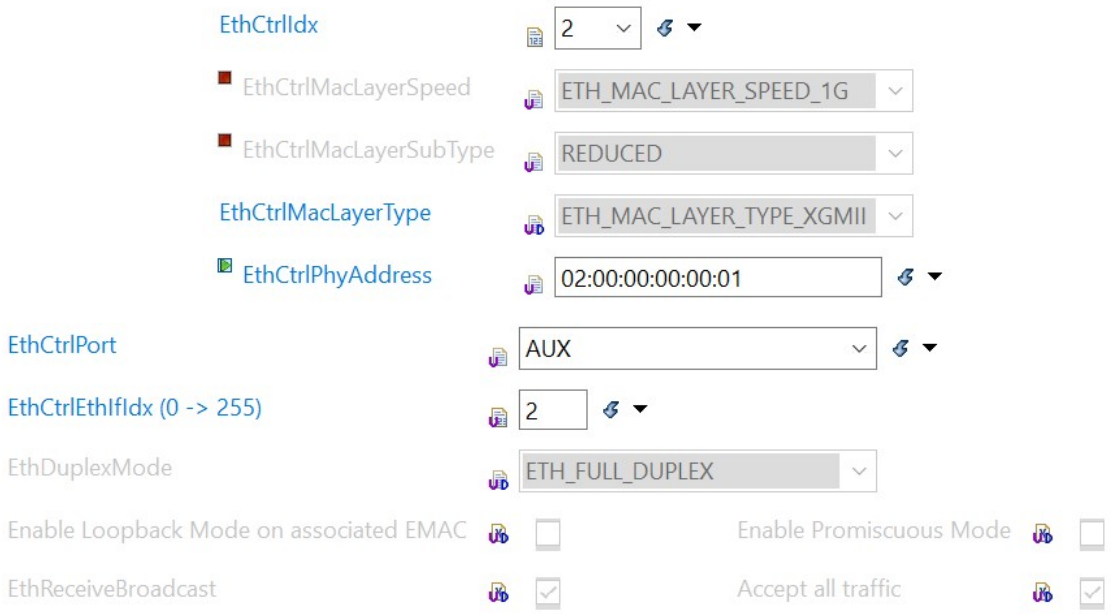
- Accept all traffic=checked: This parameter is for the case where the HIF associated to this instance and the EMAC corresponding to this controller are part of a bridge domain. If enabled, all ingress traffic received on the corresponding EMAC will be received by this controller. If disabled, only management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



The associated EthConfigEgressFiFo/IngressFiFo/ConfigSchedule needs to be configured. It is recommended to use different names for the Egress/Ingress configuration of different EthCtrlConfig.

EthCtrlConfig_1 is associated with EMAC1, which is similar to the above, and uses the RGMII interface. This Demo uses this interface.

EthCtrlConfig_2 is associated with the AUX interface, as follows: (Note, if the M/A cores are required to connect to the network, you need to configure the AUX interface on the M core, and then add this AUX, the AUX on the Linux side, and the outgoing emac to a network bridge).



2. Pacakage sends codes explanation:

   Master Demo codes provides an ARP broadcast pacakage:

```
/* 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

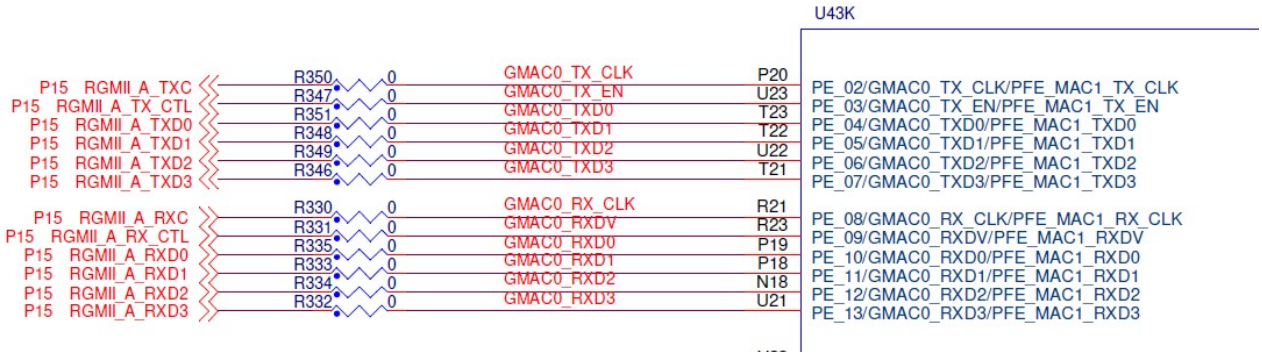
```
|-> SampleAppTask2
|   |-> So the following for loop sends broadcast packets on EMAC0/1/AUX in turn
/* Send a test frame from all the PFE controllers */
   for (CtrlIndex = 0U; CtrlIndex < ETH_43_PFE_NUM_CONTROLLER_CFG; CtrlIndex++)
   {
       tempRet |= Tst_Pfe_SendDummyFrame(CtrlIndex, broadcastMac);
|   |   |->
/* 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 Support RGMII Inteface explanation

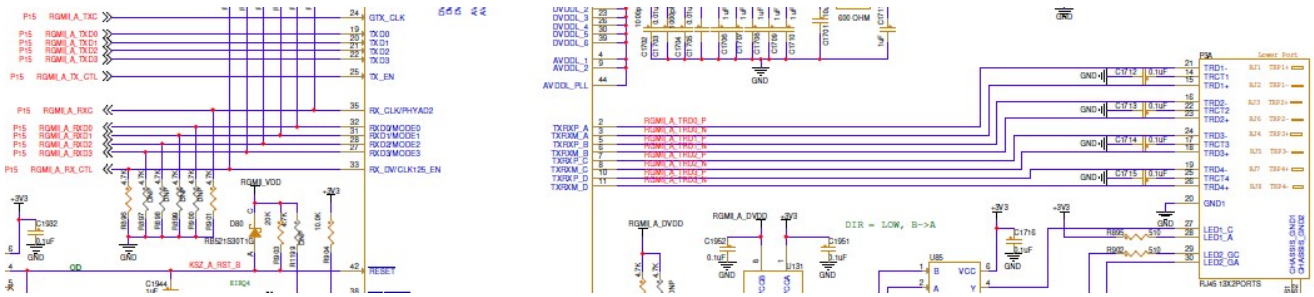
EMAC0 connected to EthCtrl0 of the Master project itself is an SGMII interface, and EMAC1 connected to EthCtrl1 is an RGMII interface.

1. Hardware link:

   On the RDB3 board, on PFE_EMAC1, the design is as follows:



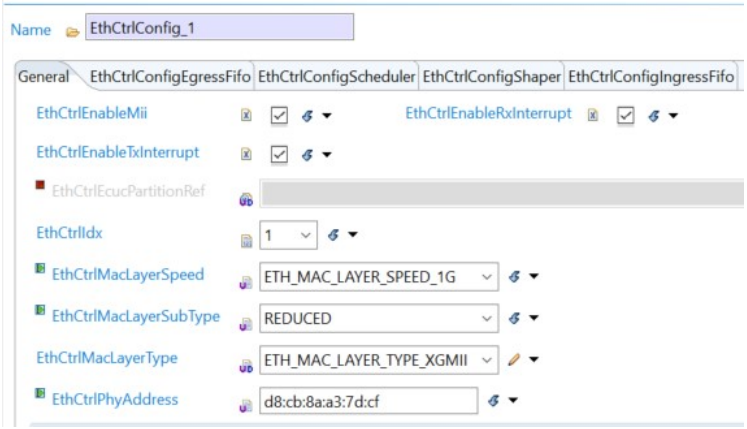
   The external is connected to the P3A RJ45 interface through the KSZ9031 PHY:



PFE_MAC1 multiplexes the IO of GMAC0.

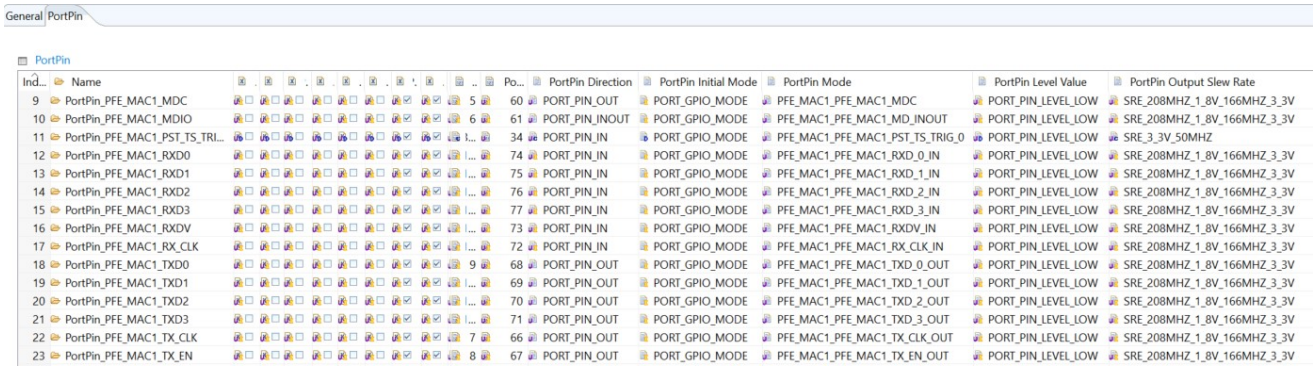
2. SW explanation

So the default PFE_EMAC1 is set to RGMII interface in the PFE Master project, as follows: MasterProject_G3->SomeId->Eth_43_PFE->Eth_1->EthCtrlConfig->EthCtrlConfig_1



The PORT module has been pre-configured. Set the IOMUX of GMAC0 to PFE_EMAC1 as follows (you can ignore it here, because this Demo uses Linux ATF to initialize IOMUX):

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



Note that the sending function is sent on the three EthCtrl interfaces in a loop, so if the previous eMAC initialization fails, it will affect the normal eMAC interface later, so if it is the customer's own board, pay attention to whether EMAC0 has a clock and whether it can be initialized correctly .

In addition, in the MCU clock setting, the TX/RX source of PFE_EMAC1 should be confirmed, it cannot be the serdas source, as follows:

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

->McuCgm2ClockMux2:

- CGM2 Clock Mux2 Source = PERIPH_PLL_PHI5_CLK //is not SERDES_1_XPCS_1_TX
- Clock Mux2 Divider0 Frequency (PFE_MAC_1_TX_CLK, GMAC_1_TX_CLK, REC_CLK) (dynamic range)= 1.25E8 //clock is the same

->McuCgm2ClockMux5:

- CGM2 Clock Mux5 Source = PFE_MAC_1_EXT_RX_CLK// is not SERDES_1_XPCS_1_CDR
- Clock Mux5 Frequency (PFE_MAC_1_RX_CLK, SEQ_CLK) (dynamic range) = 1.25E8 // clock is the same

In the ATF code on the Linux side, the IOMUX of PFE_MAC1 and GMAC0 is configured. When Uboot is configured to initialize the network port to EMAC1, it will be initialized to the IOMUX of PFE_MAC1. For other modifications, refer to the following modification instructions on the Linux side.

3.6 In order to Cooperation with Linux to Do PFE Initialization, Remove MCU/PORT Initialization

Modify \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 Even if it is a Master project, the initialization is removed.
endif
```

The code description is as follows:

```
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 In order to Cooperation with Linux to Do PFE Initialization, Remove Eth_43_PFE_PreInit(NULL_PTR);

Eth_43_PFE_PreInit requires configuration before PFE CLOCK initialization, and this Demo uses Linux uboot to configure, so it is recommended to remove it on the MCAL side: (Actually, because Uboot is configured as emac0/1/2=sgmii, sgmii, rgmii, so if then Mcal configuration does not conflict with Uboot, there is no problem if the code execution is not removed):

…\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 Script Modification

This Demo needs to wait for Linux to run to initialize the partition reset and Clock of PFE, so the MCAL sample needs to be executed later, and the modification is as follows:

…\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
```

Note that after the script is executed, lauterbach will stop at the entrance of the main function, and it needs to be executed as soon as possible. Otherwise, the slave driver on the Linux side will report a loading failure after waiting for a certain period of time for the M master driver to fail to load.

You can re-insmod it:

```
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 Core Slave Driver Compilation

For related background knowledge such as Linux compilation, please refer to the BSP document <<S32G3_LinuxBSP_34.0_User_Manual.pdf>>, or <<S32G_Kernel_BSP32_V*.pdf>>, <<S32G_Uboot_BSP*.pdf>>, <<S32G_ATF_BSP32_V*.pdf>> (Johnli, Chinese version, in public Search on the community, the version is relatively old).

4.1 Modify the DTS as Aux Interface

\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 Modify the ATF default used PIT

PFE's MCAL codes:

```
/*****************************************************************************/
/* 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);
}
```

The default is to use PIT0, (please refer to the GPT configuration of EB), and the ATF code of BSP34 also uses PIT0, which will cause conflicts, so it needs to be modified to PIT1, the patch is as follows:

```
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 Compilation Method

When compiling the Yocto project, after executing:

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

in conf/local.conf file add:

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

Then recompile the Yocto project. After compiling, the pfe slave driver module generated by default is in:

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

Notice:

In the process of writing this article, NXP is migrating the git server from codeaurora to github, so the Yocto compilation of BSP34 will fail if follow the elder doc, and it is necessary to check with NXP support window how to correct the git server address in the manifest.

4.4 Standlone Compilation Method

The way to compile the Linux Slave driver separately is as follows:

```
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

Refer the doc: pfeng/doc/PFE_S32G_A53_LNX_UserManual.pdf

3.1.2 Variant 2: Standalone build

For how to compiling PFE driver for standalone.

How to compile the slave driver:

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

Then, put the compiled output pfe-slave.ko to

/lib/modules/5.10.120-rt70+g0b76731696c1/kernel/drivers/net/ethernet/nxp/pfe/, removed the former pfe.ko. and then boot the board, enter the folder:

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

Execuate the command:

depmod -a

And then check:

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
```

Notice:

In the process of writing this article, NXP lost the tag of BLN_PFE-DRV_S32G_A53_LNX_RTM_1.1.0 when migrating the git server from codeaurora to github. This problem has been reported.

4.5 Uboot Boot Configuration

The default uboot configuration are:

```
=>  pri
ethact=eth_pfeng  // GMAC Uboot default use PFE EMAC, not the GMAC
hwconfig=pcie0:mode=rc,clock=ext;pcie1:mode=sgmii,clock=ext,fmhz=125,xpcs_mode=2G5 //xpcs_mode=both will lead theis demo fail, need check in the future
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 //default emac1configure to sgmii
pfengemac=0 //default pfe init just configure emac0, but the default emac is sgmii
```

Uboot code explanation:

```
\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;
}
```

So Uboot will initialize emac0/emac1/emac2 to sgmii/sgmii/rgmii by default according to the pfengemac variable: equivalent to Eth_43_PFE_PreInit(NULL_PTR) of Mcal.

- If the external switcher is not considered, the external emac configuration is emac1-sgmii-p5 by default. Notice.

Eth_43_PFE_PreInit is configured as an SGMII interface by default, so even if pfengemac=0 is not configured correctly, it will not affect it, and it can be changed to pfengemac=1.

- If modify the Uboot parameters to :

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

The external emac configuration is emac2-rgmii-p3b. But you need to add emac2-related EthCtrl in MCAL, and configure serdes0 in uboot, which is not considered in this Demo.

- If modify the Uboot parameters to :

```
pfeng_mode=enable,none,rgmii,none       //do not close emac0/2 will lead this demo fail, need check in the future
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 */
```

The external emac configuration is emac1-rgmii-p3a. The pfe status checked in uboot is:

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

4.6 Prepare the Linux Image

In this article, use the method of downloading the default Linux Demo image and replacing the DTB/PFE driver/Uboot configuration is as follows:

1. Burn the Demo image

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

2. Standalone compiled the PFE slave module, put into the default folder and remove the master driver module:

```
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 compiled the DTB and replace the older one:

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

4. Modify the Uboot parameters as before description.

5 M Master/A Slave Demo Test

Take the VLAN L2 bridge mode as an example: consider two emac tests here, which are:

- emac1-sgmii-p5(Default)。
- emac1-rgmii-p3a。

5.1 PFE_EMAC1(SGMII) Test Steps

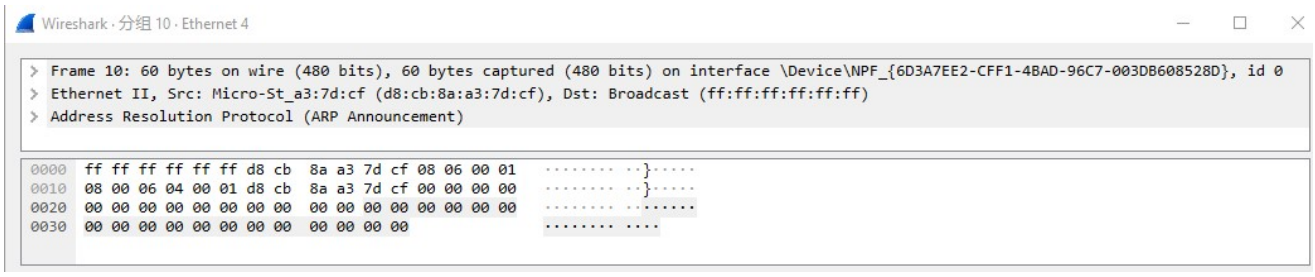
PFE Master package sending test as follows:

1. Set the RDB3 board to the Sdcard boot mode, and the Linux debugging serial port will print out the Linux startup message after power-on.
2. Run lauterbach Trace32, Open script …\
   PFE-DRV_S32G_M7_MCAL_1.0.0_QLP2\example_application\
   run_slave_app_G3_REV1_1.cmm, Click “Do” to run. At this time, S32G3 will restart, and then Linux will run again. After 10 seconds, the MCAL program will stop at the main function entry. Click “Go” to run the program immediately, otherwise the Slave driver loading on the Linux side will time out and fail. Under normal circumstances, the Linux side prints as:

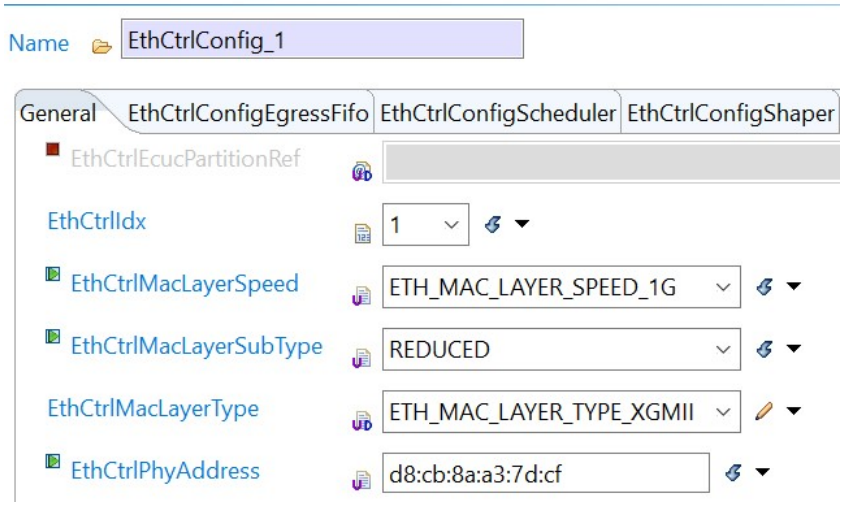
```
root@s32g399ardb3:~# dmesg |grep pfe
```

```
[    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. Connect the P5 interface of S32G3 RDB3 to the PC, open wireshark, and select the network port we need check.
4. Click “Go” to run the program on trace32, and then you can see the packets captured by wireshark as follows:



It can be seen that the source MAC address is configured for EMAC1:

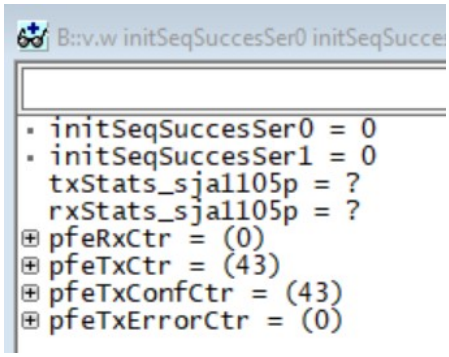


5. Lauterbach script run_main_G3.cmm opened the follow watch variants:

   v.w initSeqSuccesSer0 initSeqSuccesSer1 txStats_sja1105p rxStats_sja1105p pfeRxCtr pfeTxCtr pfeTxConfCtr pfeTxErrorCtr

   checked as follows:

   pfeTxCtrt and pfeTxConfCtr counting continuously:



6. On Linux, use ipconfig up to up AUX Interface:

```
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. Run the fci command to configure the bridge, and add HIF0 of MCAL, HIF3 and EMAC1 of Linux to a bridge.

```
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. You can use TCPdump to view the received broadcast packets sent from MCAL

```
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 ping S32G Linux or vice versa

ifconfig aux0sl 192.168.0.2

Then configure the PC as the same network, and then you can ping each other from the PC or S32G3 Linux side.

5.2 PFE_EMAC1(RGMII) Test Method

PFE_EMAC1-RGMII-P3A test method is similar with beyond, but need attention:

- The uboot code on the Linux side needs to be modified.

As explained before, modify Uboot source code and Uboot configuration parameters.

- RJ45 port link to P3A

6 M Master/A Slave Demo FCI Configure Explanation

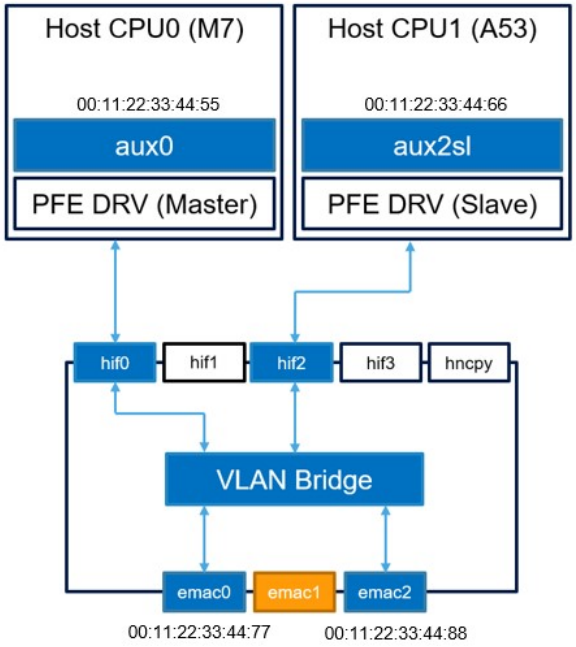
The following uses M Master/A Slave as an example to illustrate the FCI configuration, considering:

6.1 VLAN L2 Bridge

- Ethernet switch.
- Interfaces can be assigned to bridge domains using VLANs
- Use MAC addresses to route traffic
- L2 Bridge with self-learning or statically assigned MAC address
- Multiple BD types can be identified:

1. Default BD: The factory default VLAN ID for this bridge domain is 1. This field is used to handle ingress frames, which are either Ethernet frames with a VLAN ID equal to the VLAN ID of the default BD, or without a VLAN ID.
2. Fallback BD: This domain is used to handle ingress frames with unknown VLAN tags. Unknown VLAN tag indicates that the VLAN tag does not match any existing standard or default BD.
3. Standard BD: Standard user-defined bridge domains. Used by VLAN-aware bridges. These BDs process ingress frames that have a VLAN tag that matches the BD's VLAN ID.

Example: 1 bridge domain, VLAN_ID=10, physical interfaces in VLAN 10: hif0, hif2, emac0, emac2.



FCI Command as follows:

Method 1:

Create a new VLAN bridge domain with VLAN ID 10.

```
libfci_cli bd-add –vlan 10
```

Configuring Unicast/Multicast Hit/Miss Actions for Bridge Domains.

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

Add EMAC and HIF as members of VLAN bridging domain with VLAN tagging disabled.

```
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
```

Configure physical interface according to "VLAN_BRIDGE" mode, enable traffic promiscuous, block state mode "NORMAL" (enable MAC address learning)

```
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
```

Method 2:

Create a new VLAN bridge domain with VLAN ID 10.

```
libfci_cli bd-add –vlan 10
```

Configuring Unicast/Multicast Hit/Miss Actions for Bridge Domains.

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

Add EMAC and HIF as members of VLAN bridging domain with VLAN tagging disabled.

```
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
```

Create a new static MAC table entry and set the egress port

```
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
```

Configure physical interface according to "VLAN_BRIDGE" mode, enable traffic promiscuous, block state mode "FW_ONLY" (disable MAC address learning)

```
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
```

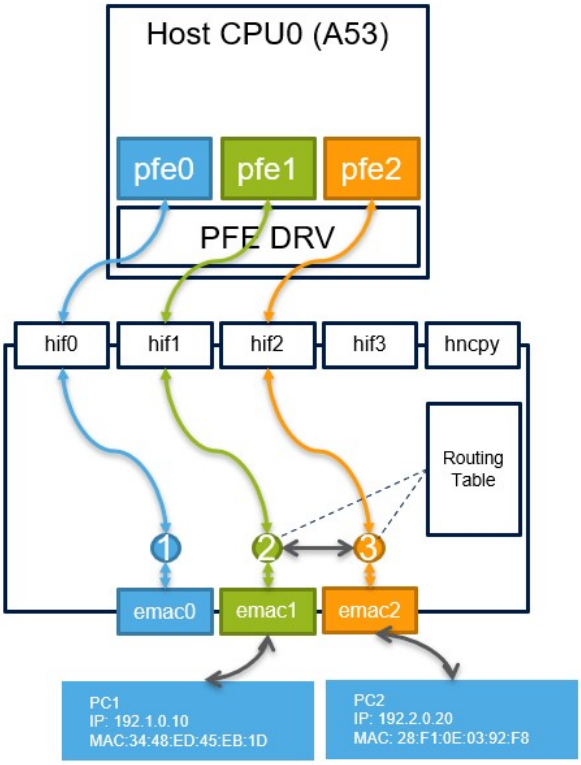
6.2 IPV4/V6 Router

A host application can change the operating mode of a physical interface to IP router and assign routing rules to the routing table using the FCI API.

Example:

- Fast-path routing between EMAC1 and EMAC2 and slow-path routing within the host CPU.

- Once configured, route points 2 and 3 forward traffic received through EMAC1 or EMAC2 that match the routing table entry to the desired destination interface (EMAC2 or EMAC1), while sending the rest of the traffic to the default interface (CPU0), where the host software Custom slow path routing can be performed.



Create a new route for the physical interface:

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

Create a new simple contract:

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

Print the routing and tracking:

libfci_cli route-print

libfci_cli cntk-print

Configure the physical interface to: "ROUTER" mode, disable traffic promiscuous:

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

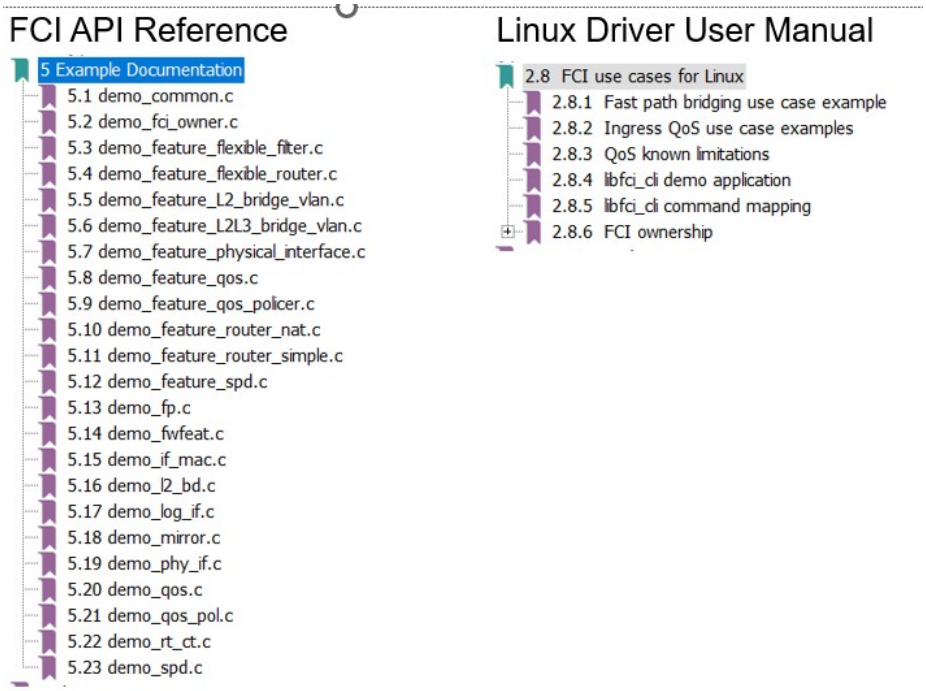
```
root@s32g399ardb3:~# libfci_cli route-print
DISCLAIMER: This is a DEMO application. It is not part of the production code deliverables.
| route      | IP   | src-mac           | dst-mac           | egress interface |
|============|======|===================|===================|==================|
|          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 Others

As follows：

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

Please refer:



7 How to Use M core to Configure FCI

7.1 M core FCI Message Proxy Explanation

As mentioned above, the Master/Slave driver is responsible for configuring PFE through the FCI interface by the Master, so if the FCI command is run on the Slave Linux side, it actually passes the configuration command to the M Master side through the RPC call, and then It is driven by M Master to call the FCI interface and configure it into PFE. The following code:

The interrupt Callback is registered as follows:

```
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

Interrupt Callback function pfe_platform_idex_rpc_cbk as follows:

```
#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
…
```

So you can add a breakpoint at the entry of the callback function to debug the FCI Message passed from A Slave Linux.

7.2 The method of actively Configuring FCI by the M core.

Based on the consideration of fast bootup and functional safety, some customers may consider configuring PFE with a functionally safe and fast-starting M core, so it is hoped to use the M Master driver to configure FCI. Our Master project also provides related samples and interfaces. as follows:

Refer the doc: 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
```

Therefore, add the corresponding compilation macro when making, and you can open the FCI configuration sample. The code call is as follows:

```
Main
|->
#if (defined(PFE_CFG_FCI_ENABLE) && (defined(FCI_L2BR_TEST) || defined(FCI_RTABLE_TEST)))
  /* FCI test */
  SampleAppFciTestTask();
#endif
|   |-> The following is the L2 bridge mode, which adds HIF0/1, EMAC0/1 to a bridge:
```

For the corresponding Linux-side FCI commands, please refer to Section 7.1 VLAN L2 Bridge.

```
#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
|   |   |-> Created a router between emac0/1, Linux command reference section 7.2, IPV4 router.
/*   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);
```