NXP Semiconductors

Layerscape Software Development Kit User Guide Supports: LSDK 19.09-update-311219

This PDF was created from content on https://nxp.com. For the most current NXP product documentation, go to https://nxp.com.

PDF excerpt generated on September 11, 2022

ENETC Linux ethernet driver

Kernel Configuration Options

Driver modules

The following table describes the Ethernet driver modules.

Driver	Description
"fsl-enetc"	ENETC Physical Function (PF) ethernet driver
"fsl-enetc-vf"	ENETC Virtual Function (VF) ethernet driver

Config tree view

To enable the ENETC PF and VF driver modules using `make menuconfig`:

```
Device Drivers  --->
        [*]Network device support  --->
                [*]Ethernet driver support  --->
                        [*]Freescale devices
                                <*>ENETC PF driver
                                <*>ENETC VF driver
```

Config option identifiers

Option	Values	Description
CONFIG_FSL_ENETC	y/m/n	ENETC Physical Function (PF) Ethernet driver
CONFIG_FSL_ENETC_VF	y/m/n	ENETC Virtual Function (VF) Ethernet driver

Device tree node

The ENETC drivers are **PCI device drivers** and the ENETC PCI Root Complex Integrated Endpoint (RCIE) is described through the following PCIe device tree node:

```
pcie@1f0000000 {/* Integrated Endpoint Root Complex */
        compatible = "pci-host-ecam-generic";
        reg = <0x01 0xf0000000 0x0 0x100000>;
        #address-cells = <3>;
        #size-cells = <2>;
        #interrupt-cells = <1>;
        msi-parent = <&its>;
        device_type= "pci";
        bus-range= <0x0 0x0>;
        dma-coherent;
        msi-map = <0 &its ...>;
        iommu-map = <0 &smmu ...>;

        ranges = <…>
                        /* PF0-6, BAR0 - non-prefetchable memory */
                        /* PF0-6, BAR2 - prefetchable memory */
                        /* PF0, VF-BAR0 - non-prefetchable memory */
                        /* PF0, VF-BAR2 - prefetchable memory */
                        /* PF1, VF-BAR0 - non-prefetchable memory*/
                        /* PF1, VF-BAR1 - prefetchable memory */
                        […]

        enetc_port0: ethernet@0,0 {
                compatible = “fsl,enetc”;
                reg = <0x000000 0 0 0 0>;
        };
        enetc_port1: ethernet@0,1 {
                compatible = “fsl,enetc”;
                reg = <0x000100 0 0 0 0>;
```

```
        };
        ethernet@0,2 {
                compatible = “fsl,enetc”;
                reg = <0x000200 0 0 0 0>;
                fixed-link {[…]};
        };
        […]
        ethernet@0,6 {
                compatible = “fsl,enetc”;
                reg = <0x000600 0 0 0 0>;
                fixed-link {[…]};
        };
};
```

For device tree binding definitions of the ENETC nodes refer to kernel document: `Documentation/devicetree/bindings/net/fsl-enetc.txt`

Source files

Source file	Description
`enetc_pf.c, enetc_pf.h`	ENETC PF driver, ENETC PSI and Port specific code
`enetc_vf.c`	ENETC VF driver, ENETC VSI specific code
`enetc.c, enetc.h`	Packet processing and other PF and VF common logic
`enetc_hw.h`	ENETC h/w specific defines (reg offsets, BDR structs etc.)
`enetc_ethtool.c`	ethtool support
`enetc_cbdr.c`	ENETC Control Buffer Descriptor Ring support
`enetc_msg.c`	ENETC VF-PF Messaging support

enetc_mdio.h, enetc_mdio.c, enetc_pci_mdio.c	MDIO driver for ENETC

Linux runtime verification

ENETC PF driver probing

```
# modprobe fsl-enetc
fsl_enetc 0000:00:00.0: enabling device (0400 -> 0402)
libphy: Freescale ENETC MDIO Bus: probed
fsl_enetc 0000:00:00.0 eth1: ENETC PF driver v1.0
fsl_enetc 0000:00:00.0 eth115: renamed from eth1
udevd[1667]: renamed network interface eth1 to eth115
fsl_enetc 0000:00:00.0 eno0: renamed from eth115
udevd[1667]: renamed network interface eth115 to eno0

fsl_enetc 0000:00:00.1: device is disabled, skipping

fsl_enetc 0000:00:00.2: enabling device (0400 -> 0402)
fsl_enetc 0000:00:00.2 eth1: ENETC PF driver v1.0
fsl_enetc 0000:00:00.2 eth114: renamed from eth1
udevd[1667]: renamed network interface eth1 to eth114
fsl_enetc 0000:00:00.2 eno2: renamed from eth114
udevd[1667]: renamed network interface eth114 to eno2

fsl_enetc 0000:00:00.6: enabling device (0400 -> 0402)
fsl_enetc 0000:00:00.6 eth1: ENETC PF driver v1.0
fsl_enetc 0000:00:00.6 eth113: renamed from eth1
udevd[1673]: renamed network interface eth1 to eth113
fsl_enetc 0000:00:00.6 eno3: renamed from eth113
udevd[1673]: renamed network interface eth113 to eno3
```

ENETC VF driver probing

For example: probing ENETC **VF0** of ENETC **PF0** (Port0)

```
# echo 1 > /sys/bus/pci/devices/0000\:00\:00.0/sriov_numvfs
fsl_enetc_vf 0000:00:01.0: enabling device (0000 -> 0002)
fsl_enetc_vf 0000:00:01.0 eth1: ENETC VF driver v1.0
fsl_enetc_vf 0000:00:01.0 eth112: renamed from eth1
udevd[1678]: renamed network interface eth1 to eth112
fsl_enetc_vf 0000:00:01.0 eno0vf0: renamed from eth112
udevd[1678]: renamed network interface eth112 to eno0vf0
```

Supported features

The following table provides the list of supported ENETC interfaces.

Supported ENETC interfaces

Ethernet Ports	PCIe PF id	Interface support	Supported VFs
Port 0	0	External SGMII/USXGMII (10/100/1G/2.5Gbps)	2
Port 1	1	External RGMII (10/100/1Gbps)	2
Port 2	2	Internal - Ethernet Switch @ 2.5Gbps	n/a
Port 3	6	Internal - Ethernet Switch @ 1Gbps	n/a

Supported Linux Ethernet driver features

Overview of the major ENETC driver features:

- PF and VF PCI Endpoint drivers

- Basic net-device features
- Multi-queue support – 1 Rx queue per CPU, 8 Tx queues
- Per Rx/Tx queue group MSI-X support (interrupt vector)
- SMMU support
- Rx H/W checksum offload for L3 INET_CSUM (CHECKSUM_COMPLETE)
- Unicast and multicast MAC filtering H/W offload
- VLAN filtering H/W offload
- VLAN insertion/ extraction H/W offload
- Scatter-gather (S/G) on Rx and Tx
- Jumbo frames of up to 9.6 KB
- Rx flow hashing (RSS)
- Rx flow steering (RFS)
- QoS – TC offloading with H/W MQPRIO
- Statistics & debug H/W counters (ethtool -S) and register dump (ethtool -d)
- VF primary MAC address config and MAC anti spoofing

Known limitations

List of major known limitations for the current driver release.

- **External MDIO read issue**: External MDIO reads 0 every now and then, this is a known hardware issue (see H/W errata doc).
  The current software workaround is to use a global lock across all ENETC register accesses. While the workaround solves the hardware issue, it introduces some limitations on the software side. One is performance impact due to locking on the fast path. Another issue is that modularity and virtualization of the ENETC VF driver is limited. Since the VF driver needs to share a global lock with the PF driver now, the VF driver can no longer be run independently of the PF driver.

    - **VF module build issue on Kernel 4.14**: This issue is related to the MDIO read issue above.The global ENETC registers lock (‘enetc_gregs’), required by the MDIO read hardware issue workaround, limits the modularity of the VF driver as the lock needs to be shared between the PF and VF ENETC drivers. Currently on Linux kernel v4.14 the ENETC VF driver fails to build as an external kernel module

because of this shared lock.**Workaround**: Both ENETC PF and VF drivers should be built as kernel built-in modules, instead of external kernel modules.

- **VF module probing denied due to duplicate symbol 'enetc_gregs'**: This issue is also related to the MDIO read issue above.The current MDIO read issue workaround if breaking modularity of the VF driver by exporting the common ‘enetc_gregs’ symbol from both PF and VF. If the VF drivers is built as and external module, modprobe will issue the following error:

```
# modprobe fsl-enetc-vf
fsl_enetc_vf: exports duplicate symbol enetc_gregs (owned by fsl_enetc)
```

Workaround: Both ENETC PF and VF drivers should be built as kernel built-in modules, instead of external kernel modules.

- **Occasional unresponsive link on ENEC Port0**: On a few LS1028A RDB boards Linux may fail to detect external link auto-negotiation completion for the ENETC Port0 PHY device - AR8033 (SGMII) - at the first attempt to bring the ethernet interface up.

  Workaround: Bringing down and up again the ENETC Port0 ethernet interface may solve the issue. If ifconfig down/up fails to bring up the link, then likely the PHY interrupts didn't trigger. Another workaround in this case would be to switch to polling mode link status verification. Polling mode for the ENETC Port0 PHY can be achieved either by disabling the CONFIG_AT803X_PHY kernel config (the Generic PHY driver will be used) or by removing the 2 interrupt property lines from the ENETC Port 0 device tree node, from the *fsl-ls1028a-rdb.dts* file.

- **‘nomsi’ boot option gives call trace on LS1028ARDB**: ENETC does not support the 'nomsi' option - MSIX allocation must not fail to successfully probe ENETC. However, on LS1028ARDB, MSI allocation failure produces call trace in the ENETC driver.

  Workaround: Remove ENETC driver from the kernel if 'nomsi' boot option is used, since ENETC is not functional without MSI support anyway.