AN12572

Ethernet Backplane Driver Support

Supports: LSDK 21.08

Rev. 4 — 26 November 2021

Application Note

1 Overview

This document describes how to enable backplane support for Layerscape and QorIQ devices with embedded support for this type of connection.

Ethernet operation over electrical backplanes, also referred to as “Backplane Ethernet,” combines the IEEE 802.3 Media Access Control (MAC) and MAC Control sublayers with a family of Physical Layers defined to support operation over a modular chassis backplane. Usually, there is no external PHY involved and the connection is made at the SoC’s PCS (Physical Coding Sublayer) level. Based on the link quality, a signal equalization is required. In cases where the link is realized based on passive direct attach cables, the link may need to be established with only the default (recommended) parameters for equalization. The standard states that a start-up algorithm should be in place in order to get the link up.

1.1 BaseKR support

Support for 10GBase-KR, 40GBase-KR, and partial support of 25GBase-KR

> **NOTE**
>
> Contact local NXP sales representative for more details on 25GBase-KR.

Contents

Layerscape and QorIQ devices comes with embedded support for backplane connections at different baud rates.

- 10G is present in custom boards with the following devices: T2080, LS1046A, LS1088A, LS2088A, and LX2160A
- 25G is present in custom boards with the following device: LX2160A
- 40G is present in custom boards with the following device: LX2160A

The enablement of backplane support is done in two parts. One refers to support from the device tree while the other is contained in the Linux kernel driver.

In the device tree, the following values are valid `backplane-mode`:

- `10gbase-kr`
- `25gbase-kr`
- `40gbase-kr4`

In the Linux kernel driver, the implementation is different depending on each of the above mentioned cases. However, the following changes are common for all:

- Advertise the link partner with the correct working mode.
- Put the lane in the correct BaseKR mode.
- Use the recommended (if it is the case) parameters for pre- and post-tap coefficients in the lane initialization phase. This affects the starting point of the algorithm.

- Optionally, update the constraint relation between tap coefficients if this is needed.

1.2 Physical layer signaling system

The backplane Ethernet extends the family of 10GBASE-R physical layer signaling system to include the BASE-KR. This specifies 10/25/40 Gb/s operation over two differential, controlled impedance pairs of traces (one pair for transmit and one pair for receive). This system employs the 10GBASE-R PCS, the serial PMA, and the BASE-KR PMD sublayers.

The BASE-KR PMD's control function implements the BASE-KR start-up protocol. This protocol facilitates timing recovery and equalization while also providing a mechanism through which the receiver can tune the transmit equalizer to optimize performance over the backplane interconnect. The BASE-KR PHY may optionally include Forward Error Correction (FEC).

Details about the aforementioned layers can be found in Clause 49, 51, and 74 of the IEEE Std 802.3.

1.3 Auto-negotiation

Auto-negotiation allows the devices at both ends of a link segment to advertise abilities, acknowledge receipt, and discover the common modes of operation that both devices share. It also rejects the use of operational modes not shared by both devices. Auto-negotiation does not test link segment characteristics.

1.4 Link training

Link training occurs after auto-negotiation has determined the link to be a Base-KR, but before auto-negotiation is done. It continuously exchanges messages (training frames) between the local and the remote device as part of the start-up phase. Link training also tunes the analog parameters of the remote and local SerDes transmitter to improve the link quality. Both LP (link partner/remote device) and LD (local device) perform link training in parallel. Link training stops when both sides decide that the link is passable. Then the link is considered up.

1.5 Backplane linux releases

Linux kernel with backplane support can be obtained from the following code aurora repository:

https://source.codeaurora.org/external/qoriq/qoriq-components/linux-extras/.

Different backplane releases are provided on top of LSDK base releases. Use the appropriate tag according to desired LSDK release, kernel version, and backplane release. The tags for backplane releases are created by using the following rules:

```
BACKPLANE - <LSDK_release> - <Kernel_version> - <Backplane_release>
```

2 Enable backplane support

2.1 Setup

Hardware setup

- Two custom boards (SoC from supported device list), with the XFI retimers bypassed
- Passive direct attach cable (l <= 1M)

Software setup

- Linux kernel with backplane support enabled
- Device tree for custom boards with backplane PHY devices

NOTE

The QDS custom boards are used for this implementation/demonstration. You may use any custom board that enables access to the Backplane Ethernet feature.

2.2 Enable backplane connection from MC

This step is required only for devices based on DPAA2 architecture.

Use `MAC_LINK_TYPE_BACKPLANE` for all ports that will be used for backplane connections. In order to do that in the MC data path configuration file, add an entry like below for all ports used:

```
board_info {
        ports {
                        …
            mac@1 {
                link_type = "MAC_LINK_TYPE_BACKPLANE";
            };
            …
            };
        };
    };
```

Deploy this configuration file on the target board as per Data Path Configuration chapter from DPAA2 User Manual.

NOTE

Omitting this step can lead to an unreliable backplane connection. Random link-down or link-up events can be experienced. This is due to a concurrent access to MDIO bus between MC core (MC firmware) and GPP core (Linux kernel).

2.3 Enable backplane support in Linux kernel

2.3.1 Enable backplane PHY driver

Enable backplane driver support in Linux kernel by using the following kconfig option:

```
Device Drivers -> Network device support -> Support for backplane on Freescale XFI interface
```

Symbol: `MDIO_FSL_BACKPLANE`

Additional `kconfig` options available for backplane driver:

- **Select default KR setup**

  This option selects the default KR setup by using: recommended TECR value or custom defined TECR value.

  — Recommended TECR value

    Use recommended TECR value hardcoded in driver as default KR setup.

  — Custom defined TECR value

    Use custom defined TECR value as default KR setup.

- **Select default AMP_RED**

  This option selects the default amplitude reduction (AMP_RED) behavior by using the recommended AMP_RED, where algorithm resets AMP_RED to zero OR use default AMP_RED according to TECR value.

  — Recommended AMP_RED

    KR algorithm resets AMP_RED to zero during training.

  - Custom defined AMP_RED

    Use default AMP_RED according to TECR value.

- **Enable backplane debugfs support**

  Enables advanced backplane debug through debugfs interface.

- **Enable advanced trace for debug monitoring**

  Enables advanced debug monitoring for backplane status using trace system. This option requires FTRACE enabled.

2.3.2 Add backplane PHY devices in device tree

2.3.2.1 SerDes device and internal MDIO buses

The SerDes device and all internal MDIO buses should be listed in the SoC's common device tree source file:

```
<linux_kernel_repo>/arch/arm64/boot/dts/freescale/fsl-<device>xa.dtsi
```

To see if the SerDes module is listed, examine a block like the one below:

```
serdes1: serdes@1ea0000  {
         compatible = "fsl,serdes-10g";
         reg = <0x0 0x1ea0000 0 0x00002000>;
         fsl,lane-reg = <0x9C0 0x980 0x940 0x900 0x8C0 0x880 0x840 0x800>;
         /* lanes H, G, F, E, D, C, B, A */
         little-endian;
};
```

If the SerDes module is listed, then it means that the serdes1 (label for SerDes node) is registered and can be used. The only client of this node is the kernel backplane PHY driver which uses the node's unit address as a base address. The base address is mapped in the SOC's memory space to further access specific MDIO registers used to control the backplane connection.

In the DTS, there must be a `serdes1` node like the one represented above. If the node is not present, then it must be added. The device base address is listed in SoC's CCSR memory map.

Currently the following types of SerDes modules are supported as available values for the property '`compatible`' depending on the SoC used:

- `fsl,serdes-10g`
- `fsl,serdes-28g`

Also, the correct endianness must be specified to allow access dependent on target endianness: little-endian or big-endian.

Next look after internal MDIO buses listed in the device tree. See the block below as an example:

```
pcs_mdio1:  mdio@0x8c07000 {
        compatibile = "fsl, fman-memac-mdio";
        reg = <0x0 0x8c07000 0x0 0x1000>;
        device_type = "mdio";
        little-endian;
};
```

The block above shows that `pcs_mdio1` is listed in the device tree. The unit address of this node (`0x8c7000`) is the WRIOP internal physical port 1 base address as it is mapped in the SoC memory space. The address `0x8c0000` is the WRIOP port block base address as it is listed in SoC reference manual. The address `0x7000` is the physical port offset in the WRIOP internal memory map. All `pcs_mdio` ports have an offset of `0x4000` between them, so the next port will be located at `0xb000` and so on. The attribute `fsl, fman-memac-mdio` means that the FSL MDIO driver will be used to access this MDIO bus. It is required to use a dedicated MDIO bus driver to access internal MDIO buses, because it uses proprietary MDIO control registers block and offset. See the DPAA2 User Manual for details about MDIO registers block.

The kernel MDIO driver used is:

```
<linux_kernel_repo>/drivers/net/Ethernet/freescale/xgmac_mdio.c
```

Internal MDIO buses should be listed for all PCS ports that support backplane KR connection, in the device tree. This is because for every port used, the management registers are accessed through the MDIO bus. See DPAA2 architecture for details on how internal MDIO registers block is mapped for every physical port and MDIO registers subchapter of SerDes chapter from SoC reference manual.

If no internal MDIO bus is listed, then add one internal MDIO bus for every PCS port target that will be used in a backplane connection.

2.3.2.2 Backplane PHY devices

PCS ports are specific to each board. Backplane PHY devices should be added in board-specific device trees:

```
<linux_kernel_repo>/arch/arm64/boot/dts/freescale/fsl-<device>-<qds, rdb>.dts.
```

A backplane PHY device is registered on an internal MDIO bus. The block below is an example:

```
pcs_mdio1: mdio@8c07000 {
        pcs1: ethernet-phy@0 {
                reg = <0>;
        };
};
&pcs1{
        compatible = "ethernet-phy-ieee802.3-c45";
        backplane-mode = "10gbase-kr";
        reg = <0x0>;
        fsl,lane-handle = <&serdes1>;
        fsl,lane-reg = <0x9C0 0x40>; /* lane H */
};
```

pcs1 is listed on the MDIO bus and should be discovered when this bus is probed. The kernel backplane PHY driver should also register a PHY driver using PHY hardware ID (read using MDIO bus).

The `backplane-mode` attribute informs the kernel backplane PHY driver about how to configure a specific SerDes lane. Currently, a SerDes lane can be configured as:

- 10gbase-kr
- 25gbase-kr
- 40gbase-kr4

The `fsl, lane-handle` attribute is used to identify which SerDes lane the PCS port belongs to. In this case “serdes1” is used.

The `fsl, lane-reg` attribute is used to identify the SerDes lane used to send and receive data. `0x9c0` is the lane H offset in the SerDes1 internal memory map. See each platform's SoC Reference Manual for details and to find the other lane’s offsets.

For LS2088A boards backplane PHY devices are added already for use with lane from H to E.

NOTE

For SerDes 1 lane are numbered in the reversed order compared to WRIO physical ports and MACs.

If the backplane PHY device is not registered on the internal MDIO buses for a specific board, then it can be added in the DTS.

2.3.2.3 Connect with Backplane PHY device handle

The kernel PHY driver is instantiated by the kernel MAC driver, there should be a specified connection between the MAC and a specific PHY in the device tree. In the following example, the backplane PHY from SerDes lane H is used:

```
&dpmac1{
        phy-handle = <&pcs1>;
        phy-connection-type = "10gbase-kr";
};
```

2.4 Enable backplane support in U-Boot

This step is only required for T2080 devices.

Specify all KR ports by using the property `fsl_10gkr_copper` in environment variable `hwconfig`. The values assigned to this property identify the port that is to be enabled in KR mode: `fm1_10g1`, `fm1_10g2`.

For example: `hwconfig=fsl_10gkr_copper:fm1_10g1,fm1_10g2`

2.5 SerDes setup

- Enable XFI protocol on SerDes lane by using correct RCW (Reset Configuration Word)
- Initialize the SerDes lane registers with recommended values for modes:
  - Ethernet 10GBASE-KR
  - Ethernet 25GBASE-KR
  - Ethernet 40GBASE-KR4

  SerDes lane registers can be initialized:
  - directly from initial RCW loaded

  or these registers can be updated later:
  - from U-Boot by using command: `mw` - memory write (fill)

    ```
    => mw.l <tecr0_address> <tecr0_hex_value>
    ```

    Example for LX2160 platform of how to set up initial KR parameters to official recommended values from U-Boot:
    - recommended initial KR parameters: `RATIO_PREQ = 0x2`, `RATIO_PST1Q = 0xd`, `ADPT_EQ = 0x20`
    - resulting TECR registers values: `TECR0=10828d00`, `TECR1=20000000`

    U-Boot commands needed for each specific interface with correct addresses for each lane required to set up TECR0/TECR1 registers:

    ```
    dpmac.2 - 40GBase-KR4 interface:
    mw 0x01ea0b30 10828d00
    mw 0x01ea0b34 20000000
    mw 0x01ea0a30 10828d00
    mw 0x01ea0a34 20000000
    mw 0x01ea0930 10828d00
    mw 0x01ea0934 20000000
    mw 0x01ea0830 10828d00
    mw 0x01ea0834 20000000

    dpmac.3 - 10GBase-KR interface:
    mw 0x01ea0f30 10828d00
    mw 0x01ea0f34 20000000
    ```

```
dpmac.4 - 10GBase-KR interface:
mw 0x01ea0e30 10828d00
mw 0x01ea0e34 20000000
```

NOTE

In case of 40G interfaces, MC resets at boot time the de-emphasis value to zero on all component lanes.

Therefore, if lane registers initialization from U-Boot is desired then U-Boot mw commands must be used only after MC startup in order to have effect and not being altered by MC.

Otherwise if U-Boot mw commands are used before MC startup, then pst1q parameter is not correctly set from U-Boot on 40G interfaces and will start with initial value as zero.

— from Linux by using command: `devmem`

```
# devmem <tecr0_address> <tecr0_hex_value>
```

NOTE

In case the kconfig option 'Custom defined TECR value' is used, then AMP_RED (amplitude reduction) must be set at recommended value for KR according to SerDes module RM: `0b000000`.

Otherwise in case of kconfig option 'Recommended TECR value' is used, this action is not required because AMP_RED is automatically set to zero by the backplane driver.

- Check the link capabilities with AN - software.
- Train the link - software.

2.6 Board configuration

Hardware adjustments

XFI retimers soldered on boards must be bypassed and PCS output signals should be routed directly to SFP+ cages pins. This operation requires physical rework to wire jumpers across the pin pads of each retimer device. This is a very important operation and should be carried out carefully.

Particular boards that support direct serdes-to-serdes connection (for example, LX2) don't need any hardware adjustments. Backplane support on these boards can be enabled by using a direct SerDes-to-SerDes connection which means each individual lane from one board is directly connected to the corresponding lane from the second board.

On SerDes1 module, XFI/Base-KR protocol will be activated on desired lanes.

Connection cables

Connect with passive direct attach cable.

Two custom boards will be connected back to back with a passive direct attach copper cable, with a maximum length of 1m (for example: SFP-H10GB-CU1M).

2.7 Interoperability

Interoperability with third party device was tested with Broadcom switch BCM956846KQ and two LS2088AQDS boards in 10GBase-KR setup using the latest training algorithm. Training was successfully performed and traffic without errors was sent on the KR channel through the switch.

2.8 Running link training

Link training is automatically performed during the auto-negotiation process.

- For DPAA1 devices auto-negotiation and link training will start once the interface is brought up.
- For DPAA2 devices auto-negotiation and link training will start at linux boot.

If the link training was successfully completed, a message similar with the following should be displayed in linux log for each KR interface trained:

```
[5.757611] fsl_backplane 8c0f000:00: dpaa2_mac dpmac.3: 10GBase-KR link trained, Tx equalization:
RATIO_PREQ = 0x0, RATIO_PST1Q = 0xd, ADPT_EQ = 0x20
```

3 Use cases

ping

In order to run a backplane *ping use case*, two boards must be connected back to back with a passive direct attach copper cable. Start MC with specified DPC file. Apply DPL using `fsl_mc apply dpl` command from U-boot and then boot Linux on both boards. After booting Linux, the interfaces must be configured properly for the two ports connected together using two IP addresses from the same IP class.

For example, use:

- On first board: `ifconfig ni0 1.1.1.1`
- On second board: `ifconfig ni0 1.1.1.2`

Once the interfaces are configured, traffic can be sent between the two the boards through the backplane link:

- On first board: `ping 1.1.1.2`
- On second board: `ping 1.1.1.1`

netperf

The backplane *netperf use case* is similar to the ping use case described above, and it is used for backplane performance benchmark. The board configuration must be done identically as described above. The difference is how traffic is sent between the two boards.

For example, using UDP streams:

- On first board: `netperf -H 1.1.1.2 -l 60 -t UDP_STREAM -N &`
- On second board: `netperf -H 1.1.1.1 -l 60 -t UDP_STREAM -N &`

4 BaseKR statistics

BaseKR algorithm statistics are available for the backplane driver by using ethtool PHY statistics counters. PHY statistics counters are displayed by using the following command:

```
ethtool --phy-statistics <intf>
```

Example: `ethtool --phy-statistics fm1-mac9`

This is an example of PHY statistics output. List of counter meanings is mentioned below:

Counters	Value	Description
LP detected	1	Link Partner detected
PCS Link up	1	Link state at the time of running ethtool command

Table continues on the next page...

Table continued from the previous page...

Counters	Value	Description
PCS Link lost detected count	2	Number of times the link was detected as lost
AN Link up	0	AN Link state at the time of running ethtool command
AN Link lost detected count	2	Number of times AN link was detected as lost
Autonegotiation complete	1	Autonegotiation was successfully completed
Autonegotiation restarted count	2	Number of times the Autonegotiation was restarted
PCS reporting high BER	0	PCS detected a high Bit Error Rate
BER counter	0	Bit Error Rate (BER) counter
Initial RATIO_PREQ	3	Initial ratio of full swing transition bit to pre-cursor used by the algorithm at startup
Initial RATIO_PST1Q	10	Initial ratio of full swing transition bit to post-cursor used by the algorithm at startup
Initial ADPT_EQ	41	Initial value of transmitter adjustment value used by the algorithm at startup
Current RATIO_PREQ	3	Current value of pre-cursor ratio at the time of running ethtool command
Current RATIO_PST1Q	10	Current value of post-cursor ratio at the time of running ethtool command
Current ADPT_EQ	41	Current value of transmitter adjustment value at the time of running ethtool command
Tuned RATIO_PREQ	3	Final value of pre-cursor ratio tuned by the training algorithm
Tuned RATIO_PST1Q	10	Final value of post-cursor ratio tuned by the training algorithm
Tuned ADPT_EQ	41	Final value of transmitter adjustment value tuned by the training algorithm
Initial TECR0	270741511	Initial value of TECR0 register used by the algorithm at startup
Tuned TECR0	270741511	Final value of TECR0 register tuned by the training algorithm
LT complete	1	Link training was successfully completed
LT duration	145	Total duration for all steps of Link training (in msec)
Link training steps	13	Total number of Link training steps
Link training restarted	39	Number of times the Link training was restarted
Link training fail count	26	Number of times the Link training failed
Link training timeout count	26	Number of times the Link training resulted in timeout

Table continues on the next page...

Table continued from the previous page...

Counters	Value	Description
Remote Tx tuning cycle count	0	Total number of Remote Tx tuning cycles for all training steps
Local Tx tuning cycle count	0	Total number of Local Tx tuning cycles for all training steps
Coefficient Updates to LP	0	Total number of Coefficient Updates requests sent to link partner
Coefficient Updates from LP	0	Total number of Coefficient Updates requests received from link partner
C(+1) increment count	2	Number of post-cursor increments
C(0) increment count	0	Number of main-cursor increments
C(-1) increment count	1	Number of pre-cursor increments
C(+1) decrement count	0	Number of post-cursor decrements
C(0) decrement count	0	Number of main-cursor decrements
C(-1) decrement count	0	Number of pre-cursor decrements
LD Preset count	1	Number of local device preset counts
LD Init count	1	Number of local device initialization counts
LD receiver ready	1	Local Device receiver is ready
LP receiver ready	1	Link Partner receiver is ready
Rx EQ Median Gaink2	15	Rx EQ Median Gaink2 value from all snapshots
PRBS sequence bit errors	0	PRBS Sequence bit errors counter

NOTE

On DPAA2 devices 'PHY statistics' must be collected on MAC interface by using the command: `ethtool -phy-statistics <macX>` and therefore linux kernel must be built with the following config options:

```
CONFIG_FSL_DPAA2_MAC=y
CONFIG_FSL_DPAA2_MAC_NETDEVS=y
```

5 BaseKR algorithm trace

BaseKR Algorithm Trace is based on Linux kernel `ftrace`. In order to use it, enable the following in Kernel:

- FTRACE
- Kernel Function Tracer

To facilitate early boot debugging, use the boot option, `trace_event=[event-list]` in `bootargs` environment variable.

The following trace events are available specifically for BaseKR Algorithm Trace:

- xgkr_debug_log - logs debug and trace information about KR algorithm
- xgkr_coe_update - logs information about KR coefficients update
- xgkr_coe_status - logs information about KR coefficients status

- xgkr_bin_snapshots - logs information about collected Bin snapshots: Bin1, Bin2, Bin3, Bin Offset, BinM1, and BinLong
- xgkr_gain_snapshots - logs information about collected Gain snapshots: GainK2, GainK3, and OSESTAT

Example:

```
setenv bootargs "console=ttyAMA0,115200 root=/dev/ram0 ramdisk_size=0x2000000
trace_event=xgkr_debug_log,xgkr_coe_update,xgkr_coe_status,xgkr_bin_snapshots,xgkr_gain_snapshots"
```

The traces are logged in the file: `/sys/kernel/debug/tracing/trace`.

After the training was performed, the collected trace log can be displayed by using the following command:

```
cat /sys/kernel/debug/tracing/trace
```

Below is represented a debug log for a specific interface after a successful training:

```
root@lx2160aqds:~# cat /sys/kernel/debug/tracing/trace | grep dpmac.3

swapper/0-1 [007] .... 5.363068: xgkr_debug_log: dpmac.3: fsl_backplane_resume:
kworker/7:0-54 [007] .... 5.368241: xgkr_debug_log: dpmac.3: fsl_backplane_config_aneg: Running
Training Algorithm v1.4.39
kworker/7:0-54 [007] .... 5.368242: xgkr_debug_log: dpmac.3: fsl_backplane_config_aneg: Bin Modules
order: BinLong before BinM1
kworker/7:0-54 [007] .... 5.368243: xgkr_debug_log: dpmac.3: fsl_backplane_config_aneg: Rx 4th Happy
condition on slide 4 is disabled
kworker/7:0-54 [007] .... 5.368243: xgkr_debug_log: dpmac.3: fsl_backplane_config_aneg: Rx Less Happy
condition is enabled
kworker/7:0-54 [007] .... 5.368244: xgkr_debug_log: dpmac.3: fsl_backplane_config_aneg: Rx Even Less
Happy condition is enabled
kworker/7:0-54 [007] .... 5.368245: xgkr_debug_log: dpmac.3: fsl_backplane_config_aneg: Rx Seemingly
Happy condition is enabled
kworker/7:0-54 [007] .... 5.368251: xgkr_debug_log: dpmac.3/ln0: fsl_backplane_config_aneg: initial
TECR0 = 0x10828d00, TECR1 = 0x20000000
kworker/7:0-54 [007] .... 5.368253: xgkr_debug_log: dpmac.3/ln0: fsl_backplane_config_aneg: starting
with: RATIO_PREQ = 0x2, RATIO_PST1Q = 0xd, ADPT_EQ = 0x20
kworker/7:0-54 [007] .... 5.368254: xgkr_debug_log: dpmac.3/ln0: init_xgkr: reset = true
kworker/7:1-1559 [007] .... 5.530684: xgkr_debug_log: dpmac.3/ln0: train_local_tx: Starting training
for Local Tx
kworker/7:1-1559 [007] .... 5.530707: xgkr_debug_log: dpmac.3/ln0: train_local_tx: Init Handshake:
first INIT received from LP
kworker/7:1-1559 [007] .... 5.530708: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ initialize:
kworker/7:1-1559 [007] .... 5.567954: xgkr_debug_log: dpmac.3/ln0: train_remote_tx: Starting training
for Remote Tx
kworker/7:1-1559 [007] .... 5.567979: xgkr_debug_log: dpmac.3/ln0: train_remote_tx: sending ld_update
= INIT
kworker/7:1-1559 [007] .... 5.568569: xgkr_debug_log: dpmac.3/ln0: train_remote_tx: continue sending
ld_update = INIT until LP responds to init: lp_status = 0x00000000
kworker/7:1-1559 [007] .... 5.608333: xgkr_debug_log: dpmac.3/ln0: train_remote_tx: Init Handshake:
LP responded to INIT after 40 ms and 64 requests / lp_status = 0x00000015
kworker/7:1-1559 [007] .... 5.620513: xgkr_debug_log: dpmac.3/ln0: train_local_tx: recv request:
0x00000001 / ld_status = 0x00000000
kworker/7:1-1559 [007] .... 5.620514: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ check_request:
recv request C(-1) INC
kworker/7:1-1559 [007] .... 5.620516: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ inc_dec: trying
to INC on C(-1) = 0x2 -> 0x1kworker/7:1-1559 [007] .... 5.620517: xgkr_debug_log: dpmac.3/ln0:
train_local_tx\ inc_dec: checking HW restrictions for: ratio_preq = 0x1, adpt_eq = 0x20, ratio_pst1q
= 0xd
kworker/7:1-1559 [007] .... 5.620519: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ inc_dec: HW
restrictions passed for: ratio_preq = 0x1, adpt_eq = 0x20, ratio_pst1q = 0xd
kworker/7:1-1559 [007] .... 5.620520: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ inc_dec: INC
```

```
performed, tuning tecr to update C(-1) = 0x1
kworker/7:1-1559 [007] .... 5.620523: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ update_ld_status:
C(-1) status = UPDATED / ld_status = 0x00000001
kworker/7:1-1559 [007] .... 5.620571: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_cdr_lock:
CDR_LOCK = 0: reset Rx lane and retry: 1
kworker/7:1-1559 [007] .... 5.659950: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_cdr_lock:
cdr_lock recovered: exit with CDR_LOCK = 1
kworker/7:1-1559 [007] .... 5.660682: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin1
is BIN_LATE
kworker/7:1-1559 [007] .... 5.660683: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin2
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.660684: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin3
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.660686: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: exit
with RX is Seemingly Happy, proceed to BinLong/BinM1
kworker/7:1-1559 [007] .... 5.660687: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ process_BinLong:
ld_update = 0x00000000
kworker/7:1-1559 [007] .... 5.662053: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin1
is BIN_LATE
kworker/7:1-1559 [007] .... 5.662054: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin2
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.662055: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin3
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.662056: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: exit
with RX is Seemingly Happy, proceed to BinLong/BinM1
kworker/7:1-1559 [007] .... 5.662058: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ process_BinLong:
ld_update = 0x00000000
kworker/7:1-1559 [007] .... 5.663438: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin1
is BIN_LATE
kworker/7:1-1559 [007] .... 5.663439: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin2
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.663440: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin3
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.663441: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: exit
with RX is Seemingly Happy, proceed to BinLong/BinM1
kworker/7:1-1559 [007] .... 5.663442: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ process_BinLong:
ld_update = 0x00000000
kworker/7:1-1559 [007] .... 5.664759: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin1
is BIN_LATE
kworker/7:1-1559 [007] .... 5.664760: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin2
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.664761: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin3
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.664762: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: exit
with RX is Seemingly Happy, proceed to BinLong/BinM1
kworker/7:1-1559 [007] .... 5.664763: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ process_BinM1:
ld_update = 0x00000001
kworker/7:1-1559 [007] .... 5.676635: xgkr_debug_log: dpmac.3/ln0: train_local_tx: recv request:
0x00000001 / ld_status = 0x00000000
kworker/7:1-1559 [007] .... 5.676637: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ check_request:
recv request C(-1) INC
kworker/7:1-1559 [007] .... 5.676638: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ inc_dec: trying to
INC on C(-1) = 0x1 -> 0x0
kworker/7:1-1559 [007] .... 5.676640: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ inc_dec: checking
HW restrictions for: ratio_preq = 0x0, adpt_eq = 0x20, ratio_pst1q = 0xd
kworker/7:1-1559 [007] .... 5.676641: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ inc_dec: HW
restrictions passed for: ratio_preq = 0x0, adpt_eq = 0x20, ratio_pst1q = 0xd
kworker/7:1-1559 [007] .... 5.676643: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ inc_dec: INC
performed, tuning tecr to update C(-1) = 0x0
kworker/7:1-1559 [007] .... 5.676646: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ update_ld_status:
```

```
C(-1) status = UPDATED / ld_status = 0x00000001
kworker/7:1-1559 [007] .... 5.677310: xgkr_debug_log: dpmac.3/ln0: train_local_tx: recv request:
0x00000001 / ld_status = 0x00000001
kworker/7:1-1559 [007] .... 5.689400: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin1
is BIN_LATE
kworker/7:1-1559 [007] .... 5.689401: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin2
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.689402: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin3
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.689403: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: exit
with RX is Seemingly Happy, proceed to BinLong/BinM1
kworker/7:1-1559 [007] .... 5.689404: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ process_BinM1:
ld_update = 0x00000001
kworker/7:1-1559 [007] .... 5.701392: xgkr_debug_log: dpmac.3/ln0: train_local_tx: recv request:
0x00000001 / ld_status = 0x00000000
kworker/7:1-1559 [007] .... 5.701394: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ check_request:
recv request C(-1) INC
kworker/7:1-1559 [007] .... 5.701395: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ inc_dec: INC
failed, COE_MAX limit reached on C(-1) = 0x0
kworker/7:1-1559 [007] .... 5.701397: xgkr_debug_log: dpmac.3/ln0: train_local_tx\ update_ld_status:
C(-1) status = MAX / ld_status = 0x00000003
kworker/7:1-1559 [007] .... 5.702780: xgkr_debug_log: dpmac.3/ln0: train_local_tx: recv request:
0x00000001 / ld_status = 0x00000003
kworker/7:1-1559 [007] .... 5.713346: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_cdr_lock:
CDR_LOCK = 0: reset Rx lane and retry: 1
kworker/7:1-1559 [007] .... 5.751962: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_cdr_lock:
cdr_lock recovered: exit with CDR_LOCK = 1
kworker/7:1-1559 [007] .... 5.752355: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin1
is BIN_LATE
kworker/7:1-1559 [007] .... 5.752356: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin2
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.752357: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin3
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.752358: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: exit
with RX is Seemingly Happy, proceed to BinLong/BinM1
kworker/7:1-1559 [007] .... 5.752360: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ process_BinM1:
ld_update = 0x00000001
kworker/7:1-1559 [007] .... 5.752950: xgkr_debug_log: dpmac.3/ln0: train_local_tx: Training complete
for Local Tx
kworker/7:1-1559 [007] .... 5.754270: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin1
is BIN_LATE
kworker/7:1-1559 [007] .... 5.754271: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin2
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.754272: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin3
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.754273: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: exit
with RX is Seemingly Happy, proceed to BinLong/BinM1
kworker/7:1-1559 [007] .... 5.754274: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ process_BinM1:
ld_update = 0x00000000
kworker/7:1-1559 [007] .... 5.755572: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin1
is BIN_LATE
kworker/7:1-1559 [007] .... 5.755573: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin2
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.755574: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin3
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.755575: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: exit
with RX is Seemingly Happy, proceed to BinLong/BinM1
kworker/7:1-1559 [007] .... 5.755577: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ process_BinM1:
ld_update = 0x00000000
kworker/7:1-1559 [007] .... 5.756880: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin1
```

```
is BIN_LATE
kworker/7:1-1559 [007] .... 5.756881: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin2
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.756882: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: Bin3
is BIN_TOGGLE
kworker/7:1-1559 [007] .... 5.756883: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ is_rx_happy: exit
with RX is Seemingly Happy, proceed to BinLong/BinM1
kworker/7:1-1559 [007] .... 5.756884: xgkr_debug_log: dpmac.3/ln0: train_remote_tx\ process_BinM1:
ld_update = 0x00000000
kworker/7:1-1559 [007] .... 5.757516: xgkr_debug_log: dpmac.3/ln0: train_remote_tx: Training complete
for Remote Tx
kworker/7:1-1559 [007] .... 5.757605: xgkr_debug_log: dpmac.3/ln0: xgkr_start_train_step: Lane 0
trained at TECR0 = 0x20808d00, TECR1 = 0x20000000
kworker/7:1-1559 [007] .... 5.757606: xgkr_debug_log: dpmac.3/ln0: xgkr_start_train_step: Lane 0 Tx
equalization: RATIO_PREQ = 0x0, RATIO_PST1Q = 0xd, ADPT_EQ = 0x20
kworker/7:1-1559 [007] .... 5.757608: xgkr_debug_log: dpmac.3/ln0: xgkr_start_train_step: Lane 0
training duration: 228 ms
kworker/7:1-1559 [007] .... 6.415943: xgkr_debug_log: dpmac.3: fsl_backplane_aneg_done:
```

If the link training was successfully completed, the following similar messages should be displayed in the debug log for each KR interface trained:

```
xgkr_debug_log: dpmac.3/ln0: train_local_tx: Training complete for Local Tx
xgkr_debug_log: dpmac.3/ln0: train_remote_tx: Training complete for Remote Tx
xgkr_debug_log: dpmac.3/ln0: xgkr_start_train_step: Lane 0 trained at TECR0 = 0x20808d00, TECR1
= 0x20000000
xgkr_debug_log: dpmac.3/ln0: xgkr_start_train_step: Lane 0 Tx equalization: RATIO_PREQ = 0x0,
RATIO_PST1Q = 0xd, ADPT_EQ = 0x20
xgkr_debug_log: dpmac.3/ln0: xgkr_start_train_step: Lane 0 training duration: 228 ms
```

6 Backplane debugfs

Backplane debugfs support is based on linux debugfs and provides simple access to backplane driver status variables and parameters for read and write purpose.

Backplane debugfs allows the possibility to monitor and control KR algorithm. In order to use backplane debugfs, enable the following kconfig in Kernel: `Enable backplane debugfs support`

Symbol: `FSL_BACKPLANE_DEBUGFS`

Backplane debugfs is available at the following path for each KR interface enabled:

```
/sys/kernel/debug/fsl_backplane/<interface>/
```

Each KR interface has the following debugfs options available:

- PHY debugfs options (available per phy):
  - PHY command options:

    file:

    - `cmd`: contains phy command options

    values:

    - `retrain`: force the training algorithm to restart

    Example:

    ```
    echo retrain > cmd
    ```

The following message will be displayed to confirm training force restart: `Forced restart KR training`

- Lane debugfs options (available per each individual lane):
  — Lane configuration options:

  file:

  ◦ `cfg`: contains lane configuration options

  values:

  ◦ `train_en`: enable training algorithm

  ◦ `train_dis`: disable training algorithm

  Example:

```
echo train_dis > cfg
```

  The following message will be displayed to confirm training algorithm was disabled: `Disabled training algorithm`

  — Display training parameters:

  file:

  ◦ `train_params`: contains current training parameters

  ◦ `tuned_params`: contains final tuned parameters

  Example:

```
cat train_params
```

  — Force equalization parameters:

  file:

  ◦ `set_preq`: set up the force value for `preq`. It requires set_apply command to apply the forced value.

  ◦ `set_pstq`: set up the force value for `pst1q`. It requires set_apply command to apply the forced value.

  ◦ `set_adpteq`: set up the force value for `adapt_eq`. It requires set_apply command to apply the forced value.

  ◦ `set_apply`: write `1` to apply all the forced values described above

  ◦ `set_ampred`: set up the force value for `amp_red`. This parameter is immediately applied with the forced value.

  Example:

```
echo 2 > set_preq
echo d > set_pstq
echo 1 > set_apply
```

  The following message will be displayed to confirm the forced setup: `Forced KR setup applied`

  Example:

```
echo 2 > set_ampred
```

  The following message will be displayed to confirm the forced amp_red: `Forced amp_red applied`

A Revision history

The table below summarizes the revisions to this document.

Table 1. Revision history

Revision	Date	Topic cross-reference	Change description
Rev 4	26 November 2021	Backplane PHY devices Connect with Backplane PHY device handle BaseKR support	Updated topics for LSDK 21.08
Rev 3	03 May 2021		Internal revision supporting LSDK 20.12
Rev 2	October 2019		Updated for LSDK 19.09
Rev 1	September 2019		Initial public release

Date of release: 26 November 2021

Document identifier: AN12572