The **"go 0x80300000 read 0x8c0b000 3 1" -** command launches the MDIO binary that is transferred at address 0x80300000. Its input arguments are:

- Operation type: read/write, read in the above case
- MAC base address, **0x8c0b000** in the current case
- MMD device address, PCS in this case - offset **3**
- Register address in the MMD space, in this example - Status register - address **1**

NOTE

Run the above command multiple times to avoid latched data.

3.2.1.2 SerDes sanity check result interpretation

The **output data** row from the read operation shows that the **Receive Link Status (bit 2 in Status Register)** is set (value of 0x00000004 - see the previous snippet). It means that there is a valid link in the PCS in the digital loopback mode, without running the MC firmware. This excludes a configuration issue on the SerDes.

If the **Receive Link Status bit** is not set, this points either to a configuration issue (RCW for example) or a problem in the design (decoupling caps, AVDD out of spec, and so on).

The next phases involve sending traffic from the LX2 to the peer from the right side, while different loopback modes are set (Figure 3 Loopback modes for **40G MAC**). The idea is to incrementally verify that the traffic can be sent and received by the DPAA2 hardware without a problem.

3.2.2 Digital loopback

This case may seem redundant, because digital loopback was described in the previous section. The approach described here uses packets to verify both transmit and receive paths. Packets are gated in the serializer on the Rx side and looped from Tx to Rx.

The purpose of this test is to validate that the traffic (for example, ARP requests) can be successfully sent and received by the DPNI interface.

The simplified view of the traffic flow is shown in Figure 5.

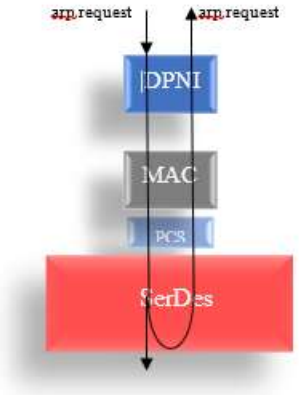

Figure 5. Digital loopback

The previous section shows that there is always a valid link present in the digital loopback.

The goal in this situation is to validate that the transmitted packets are looped and received back.

To eliminate any possible configuration problems related to link propagation between DPMAC and DPNI, some changes are applied in the U-Boot code and in the DPC file.

NOTE

The modifications from the U-Boot are valid also for the PSC loopback (see PCS loopback).

3.2.2.1 U-Boot changes

The objective of these changes is to report that the link is always up when a PHY is configured in the dpaa2 Ethernet driver probing sequence. For the PHY-less setup, these updates have no effect.

```
diff --git a/drivers/net/ldpaa_eth/ldpaa_eth.c b/drivers/net/ldpaa_eth/ldpaa_eth.c
index a3b9c152b2..1612827db2 100644
--- a/drivers/net/ldpaa_eth/ldpaa_eth.c
+++ b/drivers/net/ldpaa_eth/ldpaa_eth.c
@@ -17,7 +17,7 @@
 #include <fsl-mc/ldpaa_wriop.h>
 #include "ldpaa_eth.h"
-
+#define DEBUG
 #ifdef CONFIG_PHYLIB
 static int init_phy(struct eth_device *dev)
 {
diff --git a/drivers/net/phy/aquantia.c b/drivers/net/phy/aquantia.c
index 8ece926dd3..a149501119 100644
--- a/drivers/net/phy/aquantia.c
+++ b/drivers/net/phy/aquantia.c
@@ -554,6 +554,10 @@ int aquantia_startup(struct phy_device *phydev)
        int i = 0;
        phydev->duplex = DUPLEX_FULL;
+
+       phydev->link = 1;
+       return 0;
+
        /* if the AN is still in progress, wait till timeout. */
        if (!aquantia_link_is_up(phydev)) {
diff --git a/drivers/net/phy/phy.c b/drivers/net/phy/phy.c
index 33bce26e7f..3b32d7f234 100644
--- a/drivers/net/phy/phy.c
+++ b/drivers/net/phy/phy.c
@@ -222,6 +222,8 @@ int genphy_update_link(struct phy_device *phydev)
 {
        unsigned int mii_reg;
+       phydev->link = 1;
+       return 0;
        /*
         * Wait if the link is up, and autonegotiation is in progress
         * (ie - we're capable and it's not done)
```

The first update enables debug prints in the DPAA2 driver and the other two change the generic PHY driver and the AQR driver (AQR PHY is present on the LX2 RDB) behavior such that a linkup is always reported. If the link down event is reported by those functions, then the U-Boot init sequence for the DPAA2 Ethernet interfaces always fails when a ping command is launched.

NOTE

The U-Boot version used in the current document is here.

3.2.2.2 DPC changes

```
[…]
board_info {
                ports {
```

```
                        mac@2 {
                               link_type = "MAC_LINK_TYPE_FIXED";
                               debug_link_check="off";
                        };
[…]
```

The debug_link_check is always on, if it is not specified in the DPC. It has no effect if it is used on a MAC other than TYPE_FIXED. This option must be used in a debug scenario where you want to put the MAC or PCS into the loopback for testing purposes. If there is a problem at the peer or SERDES level, a "link down" event propagates to the DPNI. In this situation, the Rx/Tx queues are disabled by the software layer (Linux DPAA2 driver or U-Boot) and no traffic can be sent/received. When this variable is set to "off", the link is always reported as up, even if it is not. Traffic can be sent/received by the WRIOP ports, making the PCS/MAC loopback a valid setup. This way you can validate that packets can be received/sent at the PCS/MAC level.

3.2.2.3 Compiling U-Boot and DPC

After the U-Boot and DPC changes are applied, generate the firmware image and the DPC binary blob and write them to the target board. This document does not detail any build steps. They can be taken from here or from the *Layerscape Software Development Kit User Guide*.

There is an alternative to modify the DPC without deploying it into the flash on the running target.

This operation must be executed before starting the MC firmware. It saves the effort for compiling the DPC, deploying it, and flashing it to the board. This method can be used for any possible change that may be applied on the DPC.

```
#Do not start the MC firmware.(remove "fsl_mc start mc 0x20a00000  0x20e00000" from env and reboot
the target, if the MC was already started)
#Assuming that the DPC is at address 0x20e00000 in flash (mapped to DDR), move the DPC to #a new DDR
address (to be able to dynamically change it)
fdt addr 0x20e00000
fdt move 0x20e00000 0x85000000 0x5000
#Add the debug_link_check option
fdt addr 0x85000000
fdt rm /board_info/ports/mac@2
fdt mknode /board_info/ports mac@2
fdt set    /board_info/ports/mac@2 link_type  "MAC_LINK_TYPE_FIXED";
fdt set    /board_info/ports/mac@2 debug_link_check  "off";
#start the MC using the DPC from the new DDR address
fsl_mc start mc 0x20a00000 0x85000000;
```

3.2.2.4 Digital loopback verification steps

The stages of running traffic in the digital loopback mode are as follows:

```
#after the firmware and DPC have been deployed on the target, reboot. For dynamically changing the
DPC without rebooting and flashing it, see previous section.
#Note MC firmware must be started in order to be able to use the interfaces
#A log as below should be seen:

fsl-mc: Booting Management Complex ... SUCCESS
fsl-mc: Management Complex booted (version: 10.29.0, boot status: 0x1)
Hit any key to stop autoboot:  0

#set lanes A-D in digital loopback
mw 0x1EA08A0 0x10000000
mw 0x1EA09A0 0x10000000
mw 0x1EA0AA0 0x10000000
mw 0x1EA0BA0 0x10000000

#set ethact to be MAC2
```

```
setenv ethact DPMAC2@xlaui4

#run the ping command
=> ping 1.1.1.2

ldpaa_dpmac_bind, DPMAC Type= dpmac
ldpaa_dpmac_bind, DPMAC ID= 2
ldpaa_dpmac_bind, DPMAC State= 0
ldpaa_dpmac_bind, DPNI Type= dpni
ldpaa_dpmac_bind, DPNI ID= 0
ldpaa_dpmac_bind, DPNI State= 0
DPMAC link status: 1 - up
DPNI link status: 1 - up
Using DPMAC2@xlaui4 device
```

When the ping command is launched, do not expect to see an echo response. The purpose of this test is to receive the packets back, because the SerDes is in a digital loopback. The command is let to run for a couple of seconds. Then it is interrupted by pressing CTRL + C. The following output or a similar one (DPNI counters can differ but must not be 0) should be displayed in the console:

```
DPNI counters ..
DPNI_CNT_ING_ALL_FRAMES= 3
DPNI_CNT_ING_ALL_BYTES= 180
DPNI_CNT_ING_MCAST_FRAMES= 0
DPNI_CNT_ING_MCAST_BYTES= 0
DPNI_CNT_ING_BCAST_FRAMES= 3
DPNI_CNT_ING_BCAST_BYTES= 180
DPNI_CNT_EGR_ALL_FRAMES= 3
DPNI_CNT_EGR_ALL_BYTES= 126
DPNI_CNT_EGR_MCAST_FRAMES= 0
DPNI_CNT_EGR_MCAST_BYTES= 0
DPNI_CNT_EGR_BCAST_FRAMES= 3
DPNI_CNT_EGR_BCAST_BYTES= 126
DPNI_CNT_ING_FILTERED_FRAMES= 0
DPNI_CNT_ING_DISCARDED_FRAMES= 0
DPNI_CNT_ING_NOBUFFER_DISCARDS= 0
DPNI_CNT_EGR_DISCARDED_FRAMES= 0
DPNI_CNT_EGR_CNF_FRAMES= 0

DPMAC counters ..
DPMAC_CNT_ING_BYTE=15
DPMAC_CNT_ING_FRAME_DISCARD=11
DPMAC_CNT_ING_ALIGN_ERR =11
DPMAC_CNT_ING_BYTE=15
DPMAC_CNT_ING_ERR_FRAME=23
DPMAC_CNT_EGR_BYTE =23
DPMAC_CNT_EGR_ERR_FRAME =27
Abort
ping failed; host 1.1.1.2 is not alive
```

3.2.2.5 Digital loopback result interpretation

The **DPNI_CNT_ING_ALL_FRAMES** and **DPNI_CNT_EGR_ALL_FRAMES** are equal to 3.

It means that 3 ARP requests were sent and received by the WRIOP port (DPNI).

The sent packets are arp and not echo requests, because before sending the echo request, the destination must be identified.

In parallel, it can be noticed that the PCS has a link due to the digital loopback configuration:

```
# Load the MDIO binary

go 0x80300000 read 0x8c0b000 3 1

## Starting application at 0x80300000 ...
Example expects ABI version 9
Actual U-Boot ABI version 9
Internal MDIO read / write
argc = 5
operation = "read"
mac offset = "0x8c0b000"
dev addr = "3"
reg addr = "1"
mdio_cfg bsy set
mdio_data bsy set

output data: 0x00000004

## Application terminated, rc = 0x0
```

NOTE

Run the above command multiple times to avoid latched data.

This result excludes any issues in both DPMAC and DPNI.

In case of a negative result (if no packets were received at the DPNI level) ~~and if no packets were received on the DPNI,~~ a possible issue is at the PCS (though very unlikely). The next stage checks this block by putting it into loopback.

3.2.3 PCS loopback

In a PCS loopback, data is transmitted normally. On the Rx side, it is discarded at the PMA level (from a SerDes perspective). The ARP requests sent from the U-Boot are looped inside a PCS and received in the WRIOP port (DPNI).

NOTE

It is not mandatory to have a link in the PCS when testing this configuration. If there is a link problem for whatever reason (at the peer side for example) or packets cannot be observed going in or out from LX2, a PCS loopback is a way to ensure that packets can be received without any issues at the DPNI/DPMAC level. From a clock perspective, there is no problem to use the PCS loop, because even if the peer is in a wrong state (it does not have a valid clock), the clock at the PCS (when in loopback) is recovered from the internal PLL.

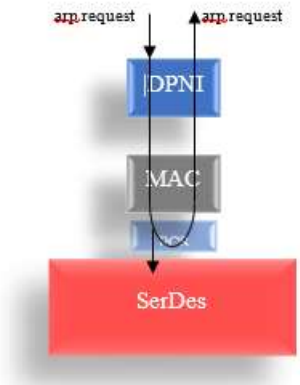

Figure 6. PCS loopback