# BCM88790

## Device Driver

## Programming Guide

# Table of Contents

# Chapter 1: Device Overview

The BCM88790 is a self-routing switching device that integrates 192 full-duplex serial links. Each link has a configurable rate of up to 53.125 Gb/s. Each link has a built-in encoder/decoder, with configurable forward BCM88790 correction (FEC) functionality.

The BCM88790 is used to build a single-stage or a three-stage switching fabric by interconnecting a number of fabric element (FE) devices.

The BCM88790 routes data cells as well as control cells. It contains on-chip buffers for data and control cells. The BCM88790 embodies four independent 192 × 192 switches; one switch for control cells and three switches for data cells.

The BCM88790 switches destination-routed data and control cells according to the cell's specified destination (FAP-ID) and the content of its routing tables. The routing tables contain dynamic fabric topology information collected via reachability messages.

For more details on BCM88790 features and specifications, refer to the *BCM88790 Self-Routing Switching Element with 50 Gb/s SerDes* data sheet (88790-DS1xx).

## 1.1 Three Data Switches

The FE contains three independent data switches (fabric pipes). The assignment of data cells to a pipe is based on the cell cast (unicast/multicast) and the cell priority. Each switch has its own link memories; it makes its own routing/load-balancing decisions, and maintains its flow control states. For each SerDes link, there is a priority/weight arbitration that determines from which switch to pick the next cell to transmit over the link.

## 1.2 Device Connectivity

### 1.2.1 Configuration and Management

A local processor can be used for device configuration and management, connecting through a PCIe Gen3 interface or a BSC interface.

### 1.2.2 Topology and Routing (Reachability Messages)

The BCM88790 receives and transmits reachability messages. The BCM88790 uses these reachability messages to build its routing tables.

## 1.3 Data and Control

The BCM88790 processes data and control cells arriving independently at the serial links and checks their integrity. If a faulty cell is detected, a counter is advanced and the cell is dropped; otherwise, the cell is forwarded into the switch.

Packet data cells are created as a result of packet segmentation (performed by the Fabric Access Processor [FAP]). The packet data cells are switched according to their final output destination, while load-balancing paths to the destination. The path to the destination is determined using the routing tables. The routing tables are automatically learned and dynamically updated according to reachability messages received from other devices.

The BCM88790 supports VSC256 cell format. VSC256 uses 16-byte control cells and variable-size data cells which include 64 and up to 256 bytes payload.

The BCM88790 does not support the VSC128 cell format and cannot be integrated into a system with a device that supports the VSC128 cell format.

Control cells are fixed-size 16 bytes, consisting of payload information and 1-byte CRC. There are three types of control cells.

- Flow-status cells
- Credit cells
- Reachability cells

## 1.4 Multicast Data

The BCM88790 also supports fabric multicast data cells. Fabric multicast is replicated in a single-stage FE2 mode, in a three-stage FE2 mode, and in a three-stage FE3 mode (for three-stage modes, see Chapter 3, Device Modes of Operation). In FE1 mode, the multicast cells are not replicated, but are load balanced between the available links into the second-level FE2 devices. For replication in FE2 and FE3 modes, the BCM88790 multicast replication table are used. The tables are maintained by the CPU. The multicast table supports up to 512K multicast groups and is accessed using an MC-ID. For each MC-ID, the multicast table holds a list of all the destination devices to which the cell must be replicated.

In the three-stage FE2 mode, each entry in the multicast table leads to a list of FE3 devices. In a single-stage FE2 and three-stage FE3 modes, each entry leads to a list of FAP destinations. For each destination indicated in the multicast table, the destination is further searched in the routing tables (such as unicast data cells) to find the links through which the destination of this copy is reachable. The copy is then switched to one of these links while load-balancing between them. Unlike the unicast tables, the multicast table is set and maintained by the CPU. The CPU updates the table when a new multicast group is created, or when a multicast group is changed.

# Chapter 2: Initialization Template

## 2.1 Overview

The system initialization involves the following steps:

- Identifying devices in the system.
- Providing SW and HW abstraction layer callbacks.
- Performing static device initialization, based on user-defined Configuration Properties.
- Attaching an identifier (unit) to the device. The identifier can then be used to modify the device configuration. dynamically, using BCM Application Programming Interface (API) calls.

The first three steps are carried out by the reference application software (bcm-diag-shell) on top of the BCM API layer. See references to the Broadcom® Device Enumerator (BDE) in the Broadcom *BCM5XX Network Switching Software Platform Guide Software Development Kit* (56XX-PG8xx-R) for detailed information. The last step is handled by dedicated BCM API functions, used to attach the device to the software, to initialize the software resources necessary to control the device, and to initialize the device itself.

The BCM API is independent of the OS and BSP. This is achieved by using SW and HW abstraction layers, described in details in the Broadcom *BCM5XX Network Switching Software Platform Guide Software Development Kit* (56XX-PG8xx-R).

Every BCM API call passes through a dispatch layer that allows controlling local and remote devices, as well as devices of different architectures.

## 2.2 Configuration Properties

Parts of the device configuration are static, set during device initialization. These parts, as well as the dynamic attributes required for the device initialization, are provided as configuration properties, also referred to as System-on-a-Chip (SoC) properties. These properties can be provided to the driver in several ways, for example, in a form of a configuration file. Refer to the Broadcom *BCM5XX Network Switching Software Platform Guide Software Development Kit* (56XX-PG8xx-R) for details.

## 2.3 BCM API

The BCM API is the software library used for configuring the BCM88790. The list of BCM APIs offers functional view of the device architecture, including a use-model and allocation services. Each BCM API is used for updating configurations that can be controlled at run time.

The BCM API is partitioned into several application modules; each applies to a specific subject. For the Traffic Manager (TM) application, the relevant modules are port, cosq, multicast, stk, linkscan, and fabric.

Refer to the *BCM5XX Network Switching Software Platform Guide Software Development Kit* (56XX-PG8xx-R) for a detailed description of BCM APIs. The set of APIs supported by a BCM88790 device is a subset of the BCM API library. This subset is available at the API coverage matrix which is a part of the SW SDK release documentation.

## 2.4 Initialization Properties

The following SoC properties are required for proper device initialization:

- The BCM88790 device core clock frequency (MHz):
  - core_clock_speed = 1000

**NOTE:** The BCM88790 core clock configuration is restricted to 1000 MHz.

A configuration with kHz accuracy is possible using a khz suffix:

  - core_clock_speed_khz = 1000000
- System reference core clock:
  - Recommended value is 1200 MHz.
  - For a system containing a legacy device, the system reference clock needs to be set according to the legacy device.
  - Configuration with kHz accuracy is possible using a khz suffix:
    - system_ref_core_clock = 1200
    - system_ref_core_clock_khz = 1200000
    - Default: 1200 MHz

## 2.5 Generic SoC Properties

The following general SoC properties are applicable to the BCM88790:

- DMA operation:
  - table_dma_enable: Enable/disable table DMA operations.
  - tdma_timeout_usec: Table DMA operation timeout.
  - tdma_intr_enable: Table DMA done is interrupt driven or by polling.
  - tslam_dma_enable: Enable/disable tslam DMA operations.
  - tslam_timeout_usec: Tslam DMA operation timeout.
  - tslam_int_enable: Tslam DMA done is interrupt driven or by polling.
- Interrupts:
  - schan_timeout_usec: Schan operation done is interrupt driven or by polling.
  - schan_intr_enable: Schan operation done is interrupt driven or by polling.
- SerDes:
  - miim_timeout_usec: MIIM operation timeout.

## 2.6 Driver Reference

### 2.6.1 Configuration Flow

Use the following sequence to initialize the BCM88790 device:

1. Initialize SW Abstraction Layer (SAL).
   – Call `sal_core_init`() to initialize the core SAL.
   – Optionally, call `sal_appl_init` () to initialize the application SAL.
2. Set up the configuration manager (CM). The CM manager is used for the hardware abstraction layer (HAL) initialization. Provide callback functions to display debug information, using `soc_cm_init(cm_structure).`
3. Detect BCM88790 devices and assign with device and revision identifier (devID, revID).
   – Done by probing the PCIe bus using BDE, using `soc_cm_device_supported`(devID, revID).
   – Assigned by user application.
4. Create driver instance and get the device handle (unit-id). This handle identifies the configured device and is an argument of all BCM API calls.
   `unit = soc_cm_device_create(devID, revID, (void *)my_data)`
5. Initialize the hardware abstraction layer (HAL). BCM API is BSP-independent. All BSP-dependent operations and attributes, such as endianess and device base-address. Use an abstraction layer maintained by the configuration manager (CM). The customer application must implement the interface of this abstraction layer. The customer must provide a set of callback functions *device vectors* (`soc_cm_device_vectors_t dev_vectors`), and the call `soc_cm_device_init` (unit, and `dev_vectors`).
   a. Retrieve initialization configuration (SoC properties).
      `dev_vectors.config_var_get`
   b. Attach/detach interrupt handler to/from IRQ vector.
      - `dev_vectors.interrupt_connect`
      - `dev_vectors.interrupt_disconnect`
   c. Read/write access.
      - `dev_vectors.read`
      - `dev_vectors.write`
   d. Shared memory allocation.
      - `dev_vectors.salloc`
      - `dev_vectors.sfree`
   e. Flush/Invalidate shared memory.
      - `dev_vectors.sflush`
      - `dev_vectors.sinval`
   f. Address translation (logical to physical and vice versa).
      - `dev_vectors.l2p`
      - `dev_vectors.p2l`
   g. PCI-specific endianess.
      - `dev_vectors.big_endian_pio`
      - `dev_vectors.big_endian_packet`
      - `dev_vectors.big_endian`

h. Device base address.

```
dev_vectors.base_address
```

At this point, the device is initialized and can be controlled by the BCM API layer.

- Create a BCM unit instance and initialize BCM modules by calling the following API.
  `bcm_init()`

**NOTE:** Internally, this API also calls `bcm_attach()`.

- Set the device Module Id. The Module Id is unique on a system level. The module Id value may affect the behavior of some APIs, and therefore must occur before the rest of application-specific configuration.
  `bcm_stk_modid_set()`
- The rest of the configuration is an application-specific, and covers various aspects of the device configuration, such as defining the fabric multicast configuration, flow control, etc. This configuration can be set to an initial state, and be modified dynamically if needed. Various configuration aspects are covered in the following sections of this document.
- As the initial configuration is set, the device can be enabled for sending and receiving traffic using the API
  `bcm_stk_module_enable()`

An existing driver instance can be *destroyed*, which results in deallocating all allocated driver resources for the unit.

- Detach the unit from BCM using bcm_detach(). After that, BCM API calls cannot be performed on the device.
- *Destroy* the driver instance using `soc_cm_device_destroy()`.

## 2.6.2 Application Reference

A reference example to basic applications can be found in the SDK code at the following path:

```
SDK_ROOT/src/appl/reference/*
```

These applications run on Broadcom reference SVK platform.

The main application-initialization sequence steps are:

1. Initializing the SDK.
2. A set of sequential steps, each step covering a distinct set of features. The term *application* hereafter applies to any one of such steps. The list of applications can be found in the following file.
   `SDK_ROOT/src/appl/reference/dnxf/appl_ref_init_deinit.c`

Use INIT_DNX shell command to trigger the main application-initialization sequence.

### 2.6.2.1 Enable/Disable Applications

Each reference application can be enabled or disabled to run upon initialization using the following SoC property:
`appl_enable_<appl_suffix>=0/1.`

To enable/disable all SDK reference applications (excluding SDK initialization step, which is always enabled) use the following SoC property.
`appl_enable=0/1`

Typical TM reference applications include the following:

- Set module ID: Each FE device must have a unique module ID. A module ID assignment application is enabled using the following SoC property.

  ```
  appl_enable_stk_init=0/1
  ```

- Init interrupts: Interrupts init application is enabled using the following SoC property.

  ```
  appl_enable_intr_init=0/1
  ```

- Isolate links: Before enabling retimer on a pair of links, the links must be isolated. The links isolation application is enabled by the following SoC property.

  ```
  appl_enable_links_isolate=0/1
  ```

For further information regarding retimer links, refer to Section 5.8.5, Retimer Links.

#### 2.6.2.2 Change Application Attributes

To provide additional information to the application layer, the following SoC property is used with a set of suffixes:

```
appl_param…<param_suffix>
```

- Set module ID: Set modid for a certain unit.

  ```
  appl_param_module_id=<modid>
  ```

- Isolate a link.

  ```
  appl_param_link_isolate_<link>=<0/1>
  ```

# Chapter 3: Device Modes of Operation

## 3.1 Overview

The BCM88790 has several device-level operational modes. These modes relate to the device position in the Folded CLOS network:

- FE13 mode
- FE2 mode

## 3.2 FE13 Mode

FE13 mode is used in three-stage fabrics for the first and third stages. The BCM88790 links are partitioned into two groups. One group is assigned to logical FE1 and the other to logical FE3. In a three-stage fabric, the FE1 makes up the first stage, while FE3s makes up the third (last) stage.

FE1 receives data and control cells from Fabric Access Processor (FAP) devices and load-balances them among its entire output links to the second stage (FE2), while considering the routing tables.

FE3 receives data and control cells from the previous stage (FE2 devices) and routes them to the destination FAP device according to the routing table. If a destination FAP device is reachable through multiple output links, the BCM88790 load-balances the cells to the FAP among these links. The FE3 replicates fabric multicast cells toward the FAP destinations, as specified in the multicast table.

Local switching in FE13 – The BCM88790 supports local switching in an FE13 device. Cells received from the FAP can be switched back to the FAP side without going through the FE2. Local switching is optional and can be enabled for unicast cells or/and multicast cells.

FE1 and FE3 have 96 input links and 96 output links each, having altogether 192 input links and 192 output links. Each logical device operates independently of the other logical device, where only some of the settings are supported. The FE1 uses links 0 to 95 as input links, and links 96 to 191 as output links. The FE3 uses links 96 to 191 as input links, and links 0 to 95 as output links.

## 3.3 FE2 Mode

The FE2 mode is used for a single-stage and three-stage fabric topologies. In a single-stage configuration, all BCM88790 devices are configured to the FE2 mode. In a three-stage configuration, all middle-stage BCM88790 devices are configured to the FE2 mode.

The FE2 logical-device receives data and control cells from the previous stage (FE1 for a three-stage fabric, or from the FAP in a single-stage fabric) and routes them to the next stage (FE3 or FAP), while adhering to the routing table. If a destination FAP device is reachable through multiple output links, the BCM88790 load-balances the cells among these links. In an FE2 mode, the BCM88790 replicates fabric multicast cells as listed in the multicast table.

## 3.4 Driver Reference

### 3.4.1 Device Mode

The following SoC variables are supported:

- FE mode:
  - Single stage
  - Multistage FE2
  - Multistage FE13

```
fabric_device_mode = < SINGLE_STAGE_FE2 | MULTI_STAGE_FE2 | MULTI_STAGE_FE13 >
```

Default: `SINGLE_STAGE_FE2`

# Chapter 4: PHY

## 4.1 Overview

The Blackhawk SerDes (PHY) serial interface IP is used with the BCM88790. The SerDes is used in Fabric physical interfaces. The SerDes block integrates 192 SerDes links. Each SerDes is a fully integrated serialization/deserialization module. Each SerDes rate is configurable and is backward-compatible with the BCM88950 and BCM88670.

Every group of eight SerDes links (each Blackhawk core) has two dedicated PLLs. Each PLL receives an external reference clock and can generate a configurable PLL output clock, also called VCO rate (PLL output clock = reference-clock x N).

Each SerDes link in the core can work with the out clock rate of either PLL0 or PLL1. Each SerDes link assigned to one of the PLLs can work with the following:

- PAM4 mode: PLL output clock doubled
- NRZ mode: PLL output clock
- NRZ mode: Half of PLL output clock

See Table 8 for the valid configurations per PLL output clock.

**NOTE:** If all link rates in the core are derived from the same VCO rate, all links will be assigned to one PLL, while the second will be inactive.

## 4.2 SoC Variables

- `load_firmware`

  **Description**

  Firmware load configuration. Represented by the following bitmap:

  – **Byte 0**: Firmware load method.

    **Values**

    - 0: Do not load firmware
    - 1: Normal load
    - 2: Fast

  – **Byte 1**: bit 0: Firmware CRC calculation enable.

  – **Byte 1**: bit 4: Firmware verification (byte-by-byte) enable.

  **Examples**

  – Setting 0x102 performs fast firmware load and CRC calculation.

  – Setting 0x1002 performs fast firmware load and firmware verification.

- `port_init_speed`

  **Description**

  Default speed by which the port is initialized.

  **Values**

  For supported SerDes rates, see Table 8.

  **Notes**

  The SDK does not provide a default speed. For an activated port, a valid speed must be provided via the SoC property.

  If the speed of the port is set to –1 via the SoC property, the port is initialized without a speed.

To finish the port initialization, call the following APIs:

– `bcm_port_resource_set` – For more information, see Section 4.3.4, SerDes Configuration.
– `bcm_port_enable_set`

- `phy_tx_polarity_flip, phy_rx_polarity_flip`

  **Description**

  Set TX/RX polarity.

  **Values**

  – 0: No polarity flip
  – 1: Polarity flip

  **Note**

  This SoC property is set after the lane swap. Configure this SoC property per logical lane and not per SerDes.

- `port_init_cl72`

  **Description**

  Initialize port with link training enabled.

  **Values**

  – 0: Link training disabled
  – 1: Link training enabled

  **Notes**

  – Internally, the chip selects the proper link training protocol: cl72, cl93, or cl136.
  – Link training is not functional when an internal loopback is set.

- `serdes_lane_config`

  **Description**

  Set SerDes lane configurations.

  For suffixes and values, see the following table.

**Table 1: serdes_lane_config Suffixes**

| Suffix | SoC Property Values | Notes |
|---|---|---|
| `dfe` | `on`: DFE is on<br>`off`: Both DFE and LP DFE are off<br>`lp`: Both DFE and LP DFE are on | If link training is enabled, DFE must be set. |
| `media_type` | `backplane`<br>`copper`<br>`optics` | — |
| `unreliable_los` | `0`<br>`1` | — |
| `cl72_auto_polarity_en` | `0`<br>`1` | — |
| `cl72_restart_timeout_en` | `0`<br>`1` | — |
| `channel_mode` | `force_nr`<br>`force_er` | This property is relevant only for PAM4 mode. Do not use it for NRZ. See Table 6 for recommended configurations. |

**Example**

When `serdes_lane_config_dfe_sfi0=lp` is configured, both DFE and LP DFE are on for logical port 0.

- `port_tx_pam4_precoder`

  **Description**

  Enable/disable precoding on the Blackhawk TX.

  **Values**

  – enable
  – disable

  **Example**

  To enable precoding on logical port 0, set `port_tx_pam4_precoder_sfi0=enable`.

  **Notes**

  – Before enabling the Blackhawk TX precoding, verify that the link-partner RX supports decoding of precoding and has enabled it.
  – This feature is required on ES channels (IL > ~22 dB). Setting the mode to ES without enabling the TX precoding results in an ES that is not fully operational.
  – Enabling the precoder/decoding can be done only if link training is disabled.

- `port_lp_tx_precoder`

  **Description**

  Enable/disable decoding of precoding on the Blackhawk RX, by stating that link partner has enabled/disabled precoding on its TX.

  **Values**

  – enable
  – disable

  **Example**

  To enable decoding on logical port 0 RX: port_lp_tx_precoder_sfi0=enable

  **Notes**

  – Before enabling the Blackhawk RX decoding, verify that the link-partner TX supports precoding and has enabled it.
  – This feature is required on ES channels (IL > ~22 dB). Setting the mode to ES without enabling the TX precoding results in an ES that is not fully operational.
  – Enabling the precoder/decoding can be done only if link training is disabled.

- `serdes_fabric_clk_freq_in`

  **Description**

  Each group of 48 links has a LCPLL that controls the ref clock.

  This property configures the LCPLL input reference clock (external to the BCM88790).

  To configure LCPLL ref clocks per 48-link group, add the group index to the property.

  **Values**

  – 1: 156.25 MHz
  – 3: 312.5 MHz (default value)

  **Example**

  – `serdes_fabric_clk_freq_in1=1`: Configures 156.25 MHz input ref clock for links 48 to 95
  – `serdes_fabric_clk_freq_in=3`: Configures 312.5 MHz input ref clock for all links (0 to 191)

- `serdes_fabric_clk_freq_out`

  **Description**

  Each group of 48 links has a LCPLL that controls the ref clock.

  This property configures the LCPLL output ref clock, internally derived from the input clock.

  To configure LCPLL ref clock per 48-link group, add the group index to the property.

  LCPLL can be bypassed, i.e., to pass the input reference clock without manipulation.

  **Values**

  – 3: 312.5 MHz
  – bypass (default value)

  **Example**

  – `serdes_fabric_clk_freq_out2=3`: Configures 312.5 MHz output clock for links 96 to 143
  – `serdes_fabric_clk_freq_out=bypass`: Configures bypass for all links (0 to 191)

  **Note**

  To bypass LCPLL set output reference clock value to `bypass`. If LCPLL is bypassed, input LCPLL must be set to 312.5 MHz

- `serdes_tx_taps`

  **Description**

  Setting TX FIR parameters (taps).

  **Values**

  Each tap is represented by a decimal number and negative values are supported where applicable.

  The following TX FIR modes are supported:

  – 3-tap mode: Encoded as follows: <signaling mode>:<pre>:<main>:<post>
  – 6-tap mode: Encoded as follows: <signaling mode>:<pre>:<main>:<post>:<pre2>:<post2>:<post3>

  Signaling mode is either pam4 or nrz.

  **Example**

  `serdes_tx_taps=nrz:-6:69:12` configures pre (-6), main (69), post (12) in NRZ signaling mode.

  **Notes**

  – PAM4 and NRZ signaling modes have different TX FIR tap ranges and restrictions.
  – For further information on the TX FIR configuration, refer to the sections “TX FIR PAM4 Mode of Operation” and “TX FIR NRZ Mode of Operation” in the *SerDes Configuration and Debugging Guide for StrataDNX™ 16-nm and 7-nm Devices* document (DBG16S-AN1xx).
  – The order of setting the TX FIR parameters and the speed is not important, but they need to be synchronized for an operational port. For example, if the port is set to NRZ mode, it is possible to set TX taps for PAM4 signaling mode, as long as the speed will be changed to PAM4 as well.
  – TX FIR parameters apply only if link training is disabled.

## 4.3 Driver Reference

### 4.3.1 Indexing

For the BCM APIs controlling SerDes, the `port_id` index indicates the Fabric Link-ID, where applicable.

### 4.3.2 General

The properties listed in the following table can be set or overridden using the `bcm_port_phy_control_set` API. To retrieve the current configuration, use the `bcm_port_phy_control_get` API.

**Table 2: bcm_port_phy_control_set General Controls**

| API Controls | Description | Notes |
|---|---|---|
| `BCM_PORT_PHY_CONTROL_RX_POLARITY` | Get the lane polarity for RX | Lane polarity cannot be changed dynamically for DNXF devices, so `bcm_port_phy_control_set` cannot be used with this control. Use the `phy_rx_polarity_flip_phy` SOC property instead. The control can still be used with `bcm_port_phy_control_get` to get the lane RX polarity. |
| `BCM_PORT_PHY_CONTROL_TX_POLARITY` | Get the lane polarity for TX | Lane polarity cannot be changed dynamically for DNXF devices, so `bcm_port_phy_control_set` cannot be used with this control. Use the `phy_tx_polarity_flip_phy` SOC property instead. The control can still be used with `bcm_port_phy_control_get` to get the lane TX polarity. |
| `BCM_PORT_PHY_CONTROL_PHASE_INTERP` | Force sync TX of the interpolator with any RX interpolator | Set only |
| `BCM_PORT_PHY_CONTROL_TX_PPM_ADJUST` | Adjust the TX phase shift to have better EMI | — |
| `BCM_PORT_PHY_CONTROL_RX_LANE_SQUELCH` | Reset the RX path | enable: Reset RX path<br>disable: Release RX reset |
| `BCM_PORT_PHY_CONTROL_TX_PAM4_PRECODER _ENABLE` | Use to enable/disable precoding on the Blackhawk TX | Before enabling the Blackhawk TX precoding, verify that link-partner RX supports decoding of precoding and has enabled it.<br>This feature is required on ES channels (IL > ~22 dB). Setting the mode to ES without enabling the TX precoding results in an ES that is not fully operational.<br>Enabling the precoder/decoding can be done only if link training is disabled.<br>**NOTE:** Disable the precoder when switching from PAM4 to NRZ. |
| `BCM_PORT_PHY_CONTROL_LP_TX_PRECODER _ENABLE` | Use to enable/disable decoding of precoding on the Blackhawk RX | Before enabling the Blackhawk RX decoding, verify that the link-partner TX supports precoding and has enabled it.<br>This feature is required on ES channels (IL > ~22 dB).<br>Setting the mode to ES without enabling the TX precoding results in an ES that is not fully operational.<br>Enabling the precoder/decoding can be done only if link training is disabled. |

## 4.3.3 Overriding Equalization Parameters

### 4.3.3.1 Receiver (RX)

RX equalization parameters can be overridden using the `bcm_port_phy_control_set` API. To retrieve the current configuration, use the `bcm_port_phy_control_get` API. The parameters listed in the following table can be overridden.

**Table 3: `bcm_port_phy_control_set` Receiver Equalization Controls**

| SOC Name | Description | Notes |
|---|---|---|
| `BCM_PORT_PHY_CONTROL_RX_PEAK_FILTER` | Use for Peaking Filter | — |
| `BCM_PORT_PHY_CONTROL_RX_LOW_FREQ_PEAK_FILTER` | Use for Low Frequency Peaking Filter | — |
| `BCM_PORT_PHY_CONTROL_RX_HIGH_FREQ_PEAK_FILTER` | Use for High Frequency Peaking Filter | — |
| `BCM_PORT_PHY_CONTROL_RX_VGA` | Use for VGA | — |
| `BCM_PORT_PHY_CONTROL_RX_TAP1` | Use to set DFE-Tap 1 | Irrelevant in PAM4 mode |
| `BCM_PORT_PHY_CONTROL_RX_TAP2` | Use to set DFE-Tap 2 | — |
| `BCM_PORT_PHY_CONTROL_RX_TAP3` | Use to set DFE-Tap 3 | — |
| `BCM_PORT_PHY_CONTROL_RX_TAP4` | Use to set DFE-Tap 4 | — |
| `BCM_PORT_PHY_CONTROL_RX_TAP5` | Use to set DFE-Tap 5 | — |
| `BCM_PORT_PHY_CONTROL_RX_TAP6` | Use to set DFE-Tap 6 | — |
| `BCM_PORT_PHY_CONTROL_RX_ADAPTATION_RESUME` | Use to release VGA and DFE-Taps | Set only |

### 4.3.3.2 Transmitter (TX)

TX equalization parameters can be overridden using the `bcm_port_phy_tx_set` API. To retrieve the current configuration, use the `bcm_port_phy_tx_get` API.

The API receives a struct of the `type bcm_port_phy_tx_t`, containing:

- All the taps:
  - `pre`
  - `main`
  - `post`
  - `pre2`
  - `post2`
  - `post3`
- `tx_tap_mode`. Options:
  - 3 taps: Configures only pre, main and post.
  - 6 taps: Configures all taps pre, main, post, pre2, post2 and post3.
- `signalling_mode`: Configures provided taps for PAM4 signaling mode or NRZ signaling mode.

**NOTE:**

- PAM4 and NRZ signaling modes have different TX FIR tap ranges and restrictions.
- For further information on the TX FIR configuration, refer to the sections “TX FIR PAM4 Mode of Operation” and “TX FIR NRZ Mode of Operation” in the *SerDes Configuration and Debugging Guide for StrataDNX™ 16-nm and 7-nm Devices* application note (DBG16S-AN1xx).
- The order of setting the TX FIR parameters and the speed is not important, but they need to be synchronized for an operational port.
  For example, if the port is set to NRZ mode, it is possible to set TX taps for PAM4 signaling mode, as long as the port speed is changed to PAM4 as well.
- TX FIR parameters can be set only if link training is disabled.

### 4.3.4 SerDes Configuration

This section describes the BCM88790 SerDes configuration.

The following SerDes properties are relevant to the APIs described in this section:

- Rate of the SerDes link. See supported values in Table 8.
- Physical coding sublayer. See Section 5.2, Physical Coding Sublayer.
- PHY lane configuration.
- Link training. BCM88790 supports three different protocols of link training: clause 72, clause 93, and clause 136. The BCM88790 internally selects the proper protocol.

#### 4.3.4.1 Setting SerDes Configuration

The `bcm_port_resource_set` API is used to configure SerDes.

The `bcm_port_resource_t` struct is one of the `bcm_port_resource_set` API arguments.

The following table contains a list of all the struct members relevant for the API. Struct members that are not listed in the table must be set to 0. To initialize the struct members, use `bcm_port_resource_t_init`.

**Table 4: `bcm_port_resource_t` Struct Members for SerDes Configuration**

| Struct Member | Additional Information |
|---|---|
| `port` | Set to fabric Link-ID or 0. |
| `speed` | Rate of the SerDes link |
| `fec_type` | Physical coding sublayer type. See Section 5.2, Physical Coding Sublayer. |
| `link_training` | — |
| `phy_lane_config` | Bitmap containing the following SerDes parameters:<br>■ DFE<br>■ LP DFE<br>■ BR DFE<br>■ medium<br>■ unreliable los<br>■ scrambling disable<br>■ cl72 polarity auto enable<br>■ cl72 restart timeout enable<br>■ force es<br>■ force ns |

**NOTE:** When calling the `bcm_port_resource_set` API and changing the FEC or the speed, the following properties are set to default, according to the FEC and speed:

- `bcmPortControlLowLatencyLLFCEnable`
- `BcmPortControlFecErrorDetectEnable`
- `bcmPortControlLlfcCellsCongestionIndEnable`
- `bcmPortControlLLFCAfterFecEnable`
- `bcmPortControlControlCellsFecBypassEnable`

For more details about the properties above, see Section 5.2, Physical Coding Sublayer.

### 4.3.4.2 Setting phy_lane_config

To set/clear/get each `phy_lane_config` parameter, use the dedicated macros listed in the following table. The format of the macros is in the following form:

`BCM_PORT_RESOURCE_PHY_LANE_CONFIG_<parameter>_SET/GET/CLEAR`.

**Table 5: `phy_lane_config` Parameter Macros**

| Macro Parameter | Additional Information |
|---|---|
| `DFE_SET/GET/CLEAR` | Set if `resource.link_training` is enabled.<br>Set if `BCM_PORT_RESOURCE_PHY_LANE_CONFIG_LP_DFE` is `SET`. |
| `LP_DFE_SET/GET/CLEAR` | — |
| `BR_DFE_SET/GET/CLEAR` | — |
| `MEDIUM_SET/GET/CLEAR` | Set a new medium type.<br>Supported medium types:<br>■ `BCM_PORT_RESOURCE_PHY_LANE_CONFIG_MEDIUM_BACKPLANE`<br>■ `BCM_PORT_RESOURCE_PHY_LANE_CONFIG_MEDIUM_COPPER_CABLE`<br>■ `BCM_PORT_RESOURCE_PHY_LANE_CONFIG_MEDIUM_OPTICS`<br>To set a new medium type:<br>1. Apply `BCM_PORT_RESOURCE_PHY_LANE_CONFIG_MEDIUM_CLEAR` to clear the existing medium type from `phy_lane_config`.<br>2. Set the new medium type using `BCM_PORT_RESOURCE_PHY_LANE_CONFIG_MEDIUM_SET`. |
| `UNRELIABLE_LOS_SET/GET/CLEAR` | — |
| `CL72_POLARITY_AUTO_EN_SET/GET/CLEAR` | — |
| `CL72_RESTART_TIMEOUT_EN_SET/GET/CLEAR` | — |
| `FORCE_ES_SET/GET/CLEAR` | This bit is relevant only for PAM4 mode. For NRZ, it must be 0.<br>In PAM4 mode, set together the macros `FORCE_ES_SET` and `FORCE_NS_SET`, see Table 6, ES and NS Configuration. |
| `FORCE_NS_SET/GET/CLEAR` | This bit is relevant only for PAM4 mode. For NRZ, it must be 0.<br>For PAM4 mode, set together the macros `FORCE_ES_SET` and `FORCE_NS_SET`, see Table 6, ES and NS Configuration. |

**Table 6: ES and NS Configuration**

| FORCE_ES | FORCE_NS | Description |
|---|---|---|
| 0 | 0 | **NRZ mode**<br>This is the only valid option. Since this configuration is not relevant for NRZ, the bits should not be set.<br>**PAM4 mode**<br>This option means that the Blackhawk RX will auto detect the appropriate mode.<br>■ If link training is enabled, ES and NS bits must be 0.<br>■ If link training is disabled, this option is not valid. |
| 1 | 0 | **Force ES**<br>Valid only for PAM4 mode. If auto selection is not used, use this setup for channels with loss > 22 dB. |
| 0 | 1 | **Force NS**<br>Valid only for PAM4 mode. If auto selection is not used, use this setup for channels with loss between 0 dB and ~22 dB. |
| 1 | 1 | This option is not valid. |

### 4.3.4.3 Getting the Default SerDes Configuration

To set a SerDes configuration, the combination of all SerDes parameters (speed, fec_type, phy_lane_config, and link_training) must be valid.

The following sequence might fail:

1. Calling `bcm_port_resource_get`.
2. Changing a parameter in the struct, for example `speed`.
3. Calling `bcm_port_resource_set`

This happens because the parameters that suited the old speed do not necessarily suit the new speed.

To set a new SerDes configuration, either call the `bcm_port_resource_set` API with a legal combination, or use the `bcm_port_resource_default_get` API.

The `bcm_port_resource_default_get` API receives the configurations given in `bcm_port_resource_t` as input, and uses them to calculate the default values of the struct members not provided by the user. In other words, the default values returned from the API are dependent on the input parameters.

**Table 7: bcm_port_resource_default_get Arguments**

| Argument | Additional Information |
|---|---|
| `port` | Fabric link-ID. |
| `flags` | Set to 0. |
| `resource` | For the `fec_type`, `phy_lane_config` and `link_training`, there are two options: Set a valid value, or for an automatic calculation by the API, set the struct member to `BCM_PORT_RESOURCE_DEFAULT_REQUEST`.<br>`speed` is a mandatory struct member and must be set.<br>`phy_lane_config` is referred to as one value. To set only some parameters use the following sequence:<br>■ Set `phy_lane_config` to `BCM_PORT_RESOURCE_DEFAULT_REQUEST`.<br>■ Call `bcm_port_resource_default_get`.<br>■ Override the defaults of the parameters to be changed, using the macros described in Table 5, `phy_lane_config` Parameter Macros. |

#### 4.3.4.4 Changing PLL Out Clock

A SerDes link rate update requires PLL out clock (VCO rate) change, when the following conditions are met:

- Both PLLs of the Blackhawk core are utilized, meaning that the speeds of the SerDes links are taken from two rows of Table 4, `bcm_port_resource_t` Struct Members for SerDes Configuration.
- The required speed cannot be derived from the current PLL out clocks, that is the speed is taken from a third row in Table1.

Changing the PLL out clock requires changing the speed of all the SerDes links associated with the PLL. It can be performed by the following methods:

- `bcm_port_resource_multi_set` API.
- Dynamic power on/off and `bcm_port_resource_set` API sequence.

The `bcm_port_resource_multi_set` API can be used to change the speed of a group of SerDes links at a single step. The API receives the following arguments:

- `nport` – Number of fabric links.
- `resource` – An array of `bcm_port_resource_t` structs, each one represents one fabric link.
- `Resource[i].port` – Must be Fabric Link-ID and cannot be 0.

**NOTE:** `bcm_port_resource_multi_set` is not limited to changing only the speed, but can be used in general to change the SerDes configuration of a group of SerDes links at a single step, in the same manner `bcm_port_resource_set` is used for a single link.

Dynamic power on/off and `bcm_port_resource_set` API sequence.

Use the following sequence:

1. Power off all links in the Blackhawk core assigned to the PLL of the SerDes link for which a speed change is required.
2. Power on all those links.
3. Call `bcm_port_resource_set` API for each one of the links.

### 4.3.5 SerDes Rates

The following table provides information about the supported SerDes rates.

**Table 8: SerDes Rates**

| LCPLLOUT | | VCO Rate | PAM4 Rate | NRZ Rate | NRZ Rate |
|---|---|---|---|---|---|
| (MHz) | N | (GHz) | (Gb/s) | (Gb/s) | (Gb/s) |
| 312.5 | 66 | 20.625 | — | 20.625 | 10.3125 |
| | 67 | 20.9375 | — | — | — |
| | 72 | 22.5 | — | — | — |
| | 73.6 | 23 | — | 23 | 11.5 |
| | 80 | 25 | 50 | 25 | — |
| | 82.5 | 25.78125 | — | 25.78125 | — |
| | 85 | 26.5625 | 53.125 | — | — |

**NOTE:** When changing the SerDes link speed, if link training is off, the TX FIR parameters of the port need to be reconfigured.

## 4.4 Diagnostics

All topics discussed in this chapter can be further explored by referring to the *SerDes Configuration and Debugging Guide for StrataDNX™ 16-nm and 7-nm Devices* document (DBG16S-AN1xx).

### 4.4.1 Transmitting and Receiving PRBS Patterns

Defining the PRBS polynomial, triggering PRBS transmission and triggering PRBS reception are controlled by the BCM API `bcm_port_control_set` with the appropriate API controls.

Reading PRBS counters is controlled by the `bcm_port_control_get` API.

**Table 9: bcm_port_phy_control_set/bcm_port_phy_control_get PRBS Controls**

| Configuration | API Control | Value Options |
|---|---|---|
| Select between PHY (SerDes) PRBS and MAC PRBS | `bcmPortControlPrbsMode` | 0 - PHY PRBS<br>1 - MAC PRBS |
| Set Polynomial | `bcmPortControlPrbsPolynomial` | ■ BCM_PORT_PRBS_POLYNOMIAL_X7_X6_1 = x^7 + x^6 + 1<br>■ BCM_PORT_PRBS_POLYNOMIAL_X9_X5_1 = x^9 + x^5 + 1<br>■ BCM_PORT_PRBS_POLYNOMIAL_X10_X7_1 = x^10 + x^7 + 1<br>■ BCM_PORT_PRBS_POLYNOMIAL_X11_X9_1 = x^11 + x^9 + 1<br>■ BCM_PORT_PRBS_POLYNOMIAL_X13_X12_X2_1 = x^13 + x^12 + x^2 + 1<br>■ BCM_PORT_PRBS_POLYNOMIAL_PAM4_13Q<br>■ BCM_PORT_PRBS_POLYNOMIAL_X15_X14_1 = x^15 + x^14 + 1<br>■ BCM_PORT_PRBS_POLYNOMIAL_X20_X3_1 = x^20 + x^3 + 1<br>■ BCM_PORT_PRBS_POLYNOMIAL_X23_X18_1 = x^23 + x^18 + 1<br>■ BCM_PORT_PRBS_POLYNOMIAL_X31_X28_1 = x^31 + x^28 + 1<br>■ BCM_PORT_PRBS_POLYNOMIAL_X49_X40_1 = x^49 + x^40 + 1<br>■ BCM_PORT_PRBS_POLYNOMIAL_X58_X31_1 = x^58 + x^39 + 1 |
| Invert RX SerDes lane polarity when triggering PRBS pattern transmission | `bcmPortControlPrbsRxInvertData` | 1 - Invert<br>0 - Do not invert |
| Invert TX SerDes lane polarity when triggering PRBS pattern transmission | `bcmPortControlPrbsTxInvertData` | 1 - Invert<br>0 - Do not invert |
| Enable PRBS transmission | `bcmPortControlPrbsTxEnable` | 1 - Enable<br>0 - Disable |
| Enable PRBS receive | `bcmPortControlPrbsRxEnable` | 1 - Enable<br>0 - Disable |
| Get PRBS status | `bcmPortControlPrbsRxStatus` | The RX status returns the number of errors occurred from the last call to this API. The RX status also returns the following indications:<br>■ –1: PRBS is not locked.<br>■ –2: (in PHY PRBS mode only) PRBS was not locked since the last call to this API (but is currently locked). |

**NOTE:**

- MAC PRBS: Only `BCM_PORT_PRBS_POLYNOMIAL_X31_X28_1` is supported.
- PHY PRBS: For Blackhawk RX proper operation, use `BCM_PORT_PRBS_POLYNOMIAL_X31_X28_1` and `BCM_PORT_PRBS_POLYNOMIAL_X58_X31_1` only.
  `BCM_PORT_PRBS_POLYNOMIAL_PAM4_13Q` is still supported for the startup pattern during link-training and other tests as required by the different standards
- Inverting RX/TX SerDes lane polarity is supported only for PHY PRBS.
- Other than `bcmPortControlPrbsRxStatus`, which is get-only, all the PRBS controls described in Table 9 can be used both to set configuration (`bcm_port_control_set` API) and to retrieve the configuration (`bcm_port_control_get` API).

#### 4.4.1.1 Transmitting and Receiving PRBS Sequence

Use the following sequence to transmit and receive PRBS.

1. `bcm_port_control_set(bcmPortControlPrbsMode)`
2. `bcm_port_control_set(bcmPortControlPrbsPolynomial)`
3. `bcm_port_control_set(bcmPortControlPrbsRxEnable)`
4. `sleep 1ms`
5. `bcm_port_control_set(bcmPortControlPrbsTxEnable)`
6. `bcm_port_control_get(bcmPortControlPrbsRxStatus)`

**NOTE:** This sequence is mandatory for MAC PRBS. For PHY PRBS, the order of `bcmPortControlPrbsRxEnable` and `bcmPortControlPrbsTxEnable` is not important, and the sleep part is not required.

#### 4.4.1.2 Disabling PRBS Sequence

Use the following sequence to disable PRBS.

1. `bcm_port_control_set(bcmPortControlPrbsTxEnable)`
2. `bcm_port_control_set(bcmPortControlPrbsRxEnable)`

**NOTE:** This sequence is mandatory for MAC PRBS. For PHY PRBS the order of `bcmPortControlPrbsRxEnable` and `bcmPortControlPrbsTxEnable` is not important.

#### 4.4.1.3 TX Pattern

In addition to sending a PRBS pattern, it is possible to send predefined patterns using the `bcm_port_phy_control_set` API with the following controls:

- `BCM_PORT_PHY_CONTROL_TX_PATTERN_LENGTH`.
- `BCM_PORT_PHY_CONTROL_TX_PATTERN_DATA0 (LSB) - BCM_PORT_PHY_CONTROL_TX_PATTERN_DATA7 (MSB)` – Defines the pattern data.
- `BCM_PORT_PHY_CONTROL_TX_PATTERN_GEN_ENABLE`.

**IMPORTANT:** After warm boot, the PRBS configuration and TX pattern configuration must be reapplied.

### 4.4.2 Putting a SerDes in Loopback

The loopback modes are described in the Section 5.9.4, Link Loopbacks in Fabric Links and in Loopback Modes in the *SerDes Configuration and Debugging Guide for StrataDNX™ 16-nm and 7-nm Devices* document (DBG16S-AN1xx).

### 4.4.3 SerDes Status

The following controls can be used to retrieve SerDes status using the `bcm_port_phy_control_get` API.

- To read RX signal energy indication, use the control:
  `BCM_PORT_PHY_CONTROL_RX_SIGNAL_DETECT`
- To validate link training success (if clause 72, clause 93, or clause 136 are enabled), use:
  `BCM_PORT_PHY_CONTROL_CL72_STATUS`.
- Calling `bcm_port_phy_control_set(BCM_PORT_PHY_CONTROL_DUMP)` dumps the status of the port. PPM offset can be read from the status dump.

#### 4.4.3.1 Eye-Scan, BER and BER-Projection

Refer to the Eye-scan, BER and BER-Projection chapter in the *SerDes Configuration and Debugging Guide for StrataDNX™ 16-nm and 7-nm Devices* document (DBG16S-AN1xx).

#### 4.4.3.2 SerDes Diagnostics Shell

The Shell commands tree can be browsed from within the Shell.

- To get to the next level of a command, type the command and press enter. For example:
  - Typing `phy` shows the options listed in the following table.
  - Typing `phy prbs` provides the `phy prbs` command options set, get, and clear.
- To get the command information (synopsis, description, arguments, and examples), enter the full command followed by the keyword `usage`. For example: `phy prbs set usage`.

The following table describes the supported shell commands.

**Table 10: Diagnostics Shell Commands for SerDes**

| Diagnostics | Description |
|---|---|
| phy info | Dump PHY info |
| phy prbs set/get/clear | Enable or disable PRBs and get PRBs results |
| phy measure | Measure the SerDes rate |
| phy cl72 | Enable or Disable link training |
| phy raw | Read or write PHY registers directly |
| phy dsc | Dump PHY DSC info |
| phy list | List register info according to given pattern |
| phy set | Configure PHY registers according to register name or address |
| phy get | Get values for given registers or pattern |
| phy eyescan | Execute eyescan for given ports |
| phy berproj | Calculate post FEC BER projection |
| phy fecstat | Dumps FEC statistic counters for enabled ports |
| phy prbsstat | Creates a thread to collect PRBS statistics for selected ports |

# Chapter 5: Fabric Links

## 5.1 Fabric Cell Formats

The BCM88790 supports only VSC256.

## 5.2 Physical Coding Sublayer

The BCM88790 can support different Physical Coding Sublayer (PCS) per link. The PCS determines the line encoding and its error correction capabilities.

Each fabric link supports one of the following PCS options:

- 64/66
- 64/66 KR-FEC (based on 10GBASE-KR)
- Reed-Solomon FEC
- Low Latency Reed-Solomon FEC
- 15T Reed-Solomon FEC
- 15T Low Latency Reed-Solomon FEC

**NOTE:** KR-FEC is supported with link rates up to 12.5 Gb/s.

For further information on the Physical Coding Sublayer concept, refer to the BCM88790 data sheet (88790-DS1xx).

## 5.3 KR-FEC Low Latency Link Level Flow Control

There is an option to add 32 bits overhead to each KR-FEC frame. The overhead is used to carry LLFC information. This low latency mode improves the LLFC reaction time.

## 5.4 RS-FEC and KR-FEC Error Indication

This feature is an FEC control that can be enabled or disabled per link. This feature allows the RX-MAC to obtain an error indication from the FEC layer that tells the MAC if the data has uncorrectable errors and whether the cells in this frame should be discarded. Enabling the error indication improves the mean time to false packet acceptance and increases the FEC frame time delay.

## 5.5 Power Efficiency

The BCM88790 has 192 links. Each group of four SerDes links, which are part of the same MAC, can be powered off to optimize power consumption.

When dynamically powering up a quad, all the port parameters must be reconfigured.

## 5.6 Lane Swap

Lanes are mapped to both RX and TX, or not mapped at all. A lane cannot be mapped to only RX or TX.

If a lane is mapped, both the lane and the RX SerDes must have identical indexes (no RX swaps allowed). Mapping is restricted to lanes inside the same quad.

A lane can be dynamically repaired to a different TX SerDes, only if all the ports in the octet (Blackhawk SerDes core) are deactivated. Unmapped lanes cannot be assigned to a port.

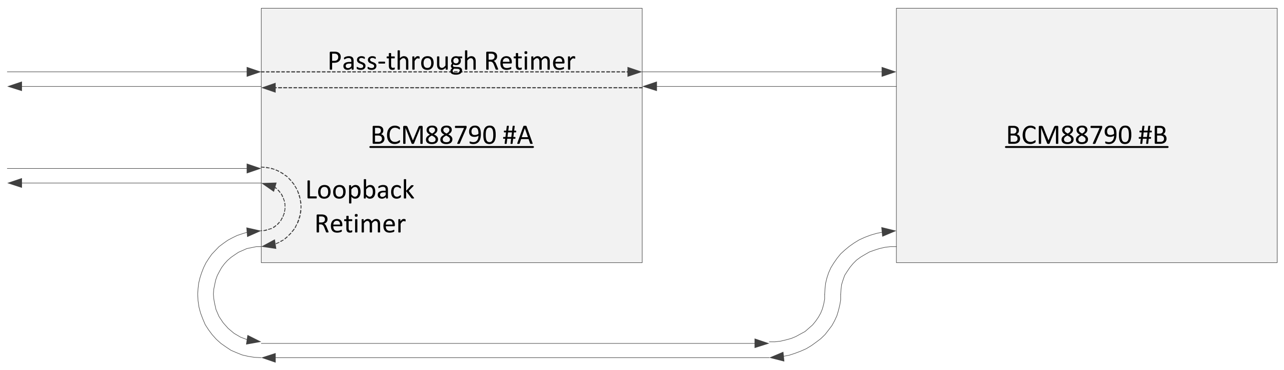
## 5.7 Retimer Links

The retimer functionality enables links to be used as retimers. This feature can be used at the operational modes FE2 and FE13.

Two retimer modes are available:

- Loopback retimer
- Pass-through retimer

**Figure 1: Retimer Link Modes**



### 5.7.1 Loopback Retimer

This function is bidirectional, connecting two lanes within the same SerDes quad. The loopback retimer can be used to connect a lane to itself, connecting its RX to TX.

## 5.7.2 Pass-through Retimer

This function is bidirectional, connecting two specific predefined lanes on both sides of the device. The predefined link pairs are listed in the following table.

**Table 11: Predefined Link Pairs for Pass-through Retimer**

| Quad Pair | Lane Pair | Quad Pair | Lane Pair |
|---|---|---|---|
| 0<>24 | 0<>96 | 12<>36 | 48<>144 |
| | 1<>97 | | 49<>145 |
| | 2<>98 | | 50<>146 |
| | 3<>99 | | 51<>147 |
| 1<>25 | 4<>100 | 13<>37 | 52<>148 |
| | 5<>101 | | 53<>149 |
| | 6<>102 | | 54<>150 |
| | 7<>103 | | 55<>151 |
| 2<>38 | 8<>152 | 14<>26 | 56<>104 |
| | 9<>153 | | 57<>105 |
| | 10<>154 | | 58<>106 |
| | 11<>155 | | 59<>107 |
| 3<>39 | 12<>156 | 15<>27 | 60<>108 |
| | 13<>157 | | 61<>109 |
| | 14<>158 | | 62<>110 |
| | 15<>159 | | 63<>111 |
| 4<>40 | 16<>160 | 16<>28 | 64<>112 |
| | 17<>161 | | 65<>113 |
| | 18<>162 | | 66<>114 |
| | 19<>163 | | 67<>115 |
| 5<>41 | 20<>164 | 17<>29 | 68<>116 |
| | 21<>165 | | 69<>117 |
| | 22<>166 | | 70<>118 |
| | 23<>167 | | 71<>119 |
| 6<>42 | 24<>168 | 18<>30 | 72<>120 |
| | 25<>169 | | 73<>121 |
| | 26<>170 | | 74<>122 |
| | 27<>171 | | 75<>123 |
| 7<>43 | 28<>172 | 19<>31 | 76<>124 |
| | 29<>173 | | 77<>125 |
| | 30<>174 | | 78<>126 |
| | 31<>175 | | 79<>127 |
| 8<>44 | 32<>176 | 20<>32 | 80<>128 |
| | 33<>177 | | 81<>129 |
| | 34<>178 | | 82<>130 |
| | 35<>179 | | 83<>131 |

**Table 11: Predefined Link Pairs for Pass-through Retimer (Continued)**

| Quad Pair | Lane Pair | Quad Pair | Lane Pair |
|---|---|---|---|
| 9<>45 | 36<>180 | 21<>33 | 84<>132 |
| | 37<>181 | | 85<>133 |
| | 38<>182 | | 86<>134 |
| | 39<>183 | | 87<>135 |
| 10<>46 | 40<>184 | 22<>34 | 88<>136 |
| | 41<>185 | | 89<>137 |
| | 42<>186 | | 90<>138 |
| | 43<>187 | | 91<>139 |
| 11<>47 | 44<>188 | 23<>35 | 92<>140 |
| | 45<>189 | | 93<>141 |
| | 46<>190 | | 94<>142 |
| | 47<>191 | | 95<>143 |

## 5.8 Driver Reference

### 5.8.1 Physical Coding Sublayer

The following is the physical coding sublayer SoC property:

```
port_fec_<logical port>=<FEC value>
```

Supported values:

- 0: No FEC
- 1: 64/66 KR-FEC
- 5: Reed-Solomon FEC
- 6: Low Latency Reed-Solomon FEC
- 7: 15T Reed-Solomon FEC
- 8: Low Latency 15T Reed-Solomon FEC

To configure the physical coding sublayer after the init sequence, use the `bcm_port_resource_set` API (see Section 4.3.4, SerDes Configuration).

`resource.fec_type` represents the physical coding sublayer.

Supported values:

- `bcmPortPhyFecNone`: No FEC
- `bcmPortPhyFecBaseR`: 64/66 KR-FEC
- `bcmPortPhyFecRs206`: Reed-Solomon FEC
- `bcmPortPhyFecRs108`: Low Latency Reed-Solomon FEC
- `bcmPortPhyFecRs545`: 15T Reed-Solomon FEC
- `bcmPortPhyFecRs304`: Low Latency 15T Reed-Solomon FEC

**NOTE:** 64/66 KR-FEC is supported with link rates up to 12.5 Gb/s.

Additional port properties are configured by calling `bcm_port_control_set` with various `bcm_port_control_t`.

- `bcmPortControlLowLatencyLLFCEnable`: Adds/removes congestion indications (RCI/GCI/LLFC) from the frame (applicable over PCS KR-FEC only).
- `bcmPortControlFecErrorDetectEnable`: Adds/removes error indications from the frame (applicable over both PCS KR-FEC and PCS RS-FEC).
- `bcmPortControlLlfcCellsCongestionIndEnable`: Extracts CIG indications from LLFC cells (supported for KR-FEC and should be disabled if the remote link is BCM88750/BCM88650/BCM88660).
- `bcmPortControlLLFCAfterFecEnable`: When set, congestion indications (RCI/GCI/LLFC) are extracted from the frame after error correction at the expense of latency (applicable over PCS RS-FEC only).
- `bcmPortControlControlCellsFecBypassEnable`: Determines if FEC is bypassed by credit and flow status control cells (applicable over PCS RS-FEC only).

### 5.8.2 Power Efficiency

#### 5.8.2.1 Power Off/On SoC Property

If `serdes_qrtt_active` is True, this SerDes quad is activated.

If the quad is deactivated at init but expected to be used in the future, it must be dynamically powered on and reconfigured.

Deactivating a quad reduces the device power consumption.

The Suffix _N denotes quad N:

```
serdes_qrtt_active<_N> = <0|1>
```

Default: 1

#### 5.8.2.2 Dynamic Power On/Off

##### 5.8.2.2.1 Power Off Link

Prior to power off, the link must be isolated. Isolate the link using `bcm_fabric_link_control_set` with control type `bcmFabricLinkIsolate`.

The actual power off is done in a quad resolution. The quad power off sequence is applied only if all the fabric links in a specific quad are detached. Use the following API to power off the link:

```
int bcm_port_detach(int unit, bcm_pbmp_t pbmp, bcm_pbmp_t *detached)
```

##### 5.8.2.2.2 Power Up Link

Power on is performed in a quad resolution. The quad power on sequence is applied only if at least one link in a specific quad is attached. Use the following API to power off the link:

```
int bcm_port_probe(int unit, bcm_pbmp_t pbmp, bcm_pbmp_t *okay_pbmp)
```

After power up, fabric links are reconfigured using the following sequence:

1. Isolate the link using `bcm_fabric_link_control_set` with control type `bcmFabricLinkIsolate`.
2. Configure the link physical coding sublayer, speed, link training and PHY lane configuration using the `bcm_port_resource_set` API.
3. Enable/disable the link packet cell packing using the link control type `bcmFabricLinkPcpEnable`.
4. If connected to a legacy device, use `bcm_fabric_link_remote_pipe_mapping_set` to configure remote pipe mapping.
5. If additional SerDes properties are required to be different from the default, configure the SerDes property using `bcm_port_phy_control_set`.

   For example, configure TX FIR using `bcm_port_phy_tx_set`, see Chapter 4, PHY.
6. Use `bcm_port_enable_set` to enable the link.
7. Unisolate the link using `bcm_fabric_link_control_set` with control type `bcmFabricLinkIsolate`.

### 5.8.3 Link Activation

- The `bcm_port_enable_set` enables/disables a link on both the MAC and SerDes level.
- The port control `bcmPortControlLinkDownPowerOn` enables/disables ports on the MAC level.
- 0: The Link is disabled on the MAC level; the SerDes power-state is not changed (SerDes in TX reset state).
- 1: Activates the link on the MAC level; the SerDes power-state is not changed.

### 5.8.4 TX Lane Swap

This section covers the following topics:

- Configuring TX lane swapping
- Setting TX lanes as not mapped

Both of these actions can be performed using either an SOC property or an API.

#### 5.8.4.1 Configuring TX Lane Swap Using an SOC Property

Use the following SOC property:

```
lane_to_serdes_map_fabric_lane<lane_id>=rx<srd_rx_id>:tx<srd_tx_id>
```

For example, to map fabric link TX 0 to SerDes TX 1, use:

```
lane_to_serdes_map_fabric_lane0=rx0:tx1
```

#### 5.8.4.2 Configuring TX Lane Swap Using an API

Use the following API:

```
int bcm_port_lane_to_serdes_map_set/get(int unit, int flags, int map_size,
bcm_port_lane_to_serdes_map_t *serdes_map);

typedef struct bcm_port_lane_to_serdes_map_s {
    int serdes_rx_id;
    int serdes_tx_id;
} bcm_port_lane_to_serdes_map_t;
```

##### 5.8.4.2.1 API Parameters

The parameters for this API are:

- `map_size` – The array size must be 192.
- `serdes_map` – An array representing lane mapping for all lanes. Each lane has both RX and TX mapping. The array index represents a lane number.
- `flags` – For BCM88790, use the `BCM_PORT_LANE_TO_SERDES_FABRIC_SIDE` flag.

#### 5.8.4.3 Setting a Lane as Not Mapped Using an SOC Property

Use the following SOC property:

```
lane_to_serdes_map_fabric_lane<lane_id>=NOT_MAPPED
```

**NOTE:** By default, a lane that does not appear in the SOC properties is unmapped.

### 5.8.4.4 Setting a Lane as Not Mapped Using an API

Use the following API:

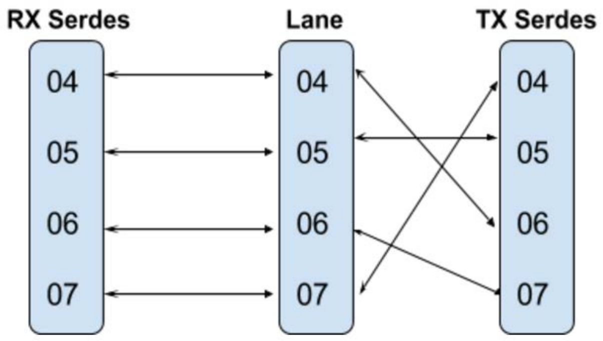
```
serdes_map[lane].serdes_rx_id=BCM_PORT_LANE_TO_SERDES_NOT_MAPPED;
serdes_map[lane].serdes_tx_id=BCM_PORT_LANE_TO_SERDES_NOT_MAPPED;
```

**NOTE:** A fabric port can be deactivated using the `bcm_port_detach` API.

### 5.8.4.5 Lane Mapping Example

The following figure shows an example lane mapping configuration.

**Figure 2: Lane Mapping Example**



To apply the mapping shown in Figure 2 using SOC properties, do the following:

```
lane_to_serdes_map_fabric_lane4=rx4:tx6
lane_to_serdes_map_fabric_lane5=rx5:tx5
lane_to_serdes_map_fabric_lane6=rx6:tx7
lane_to_serdes_map_fabric_lane7=rx7:tx4
```

To apply the mapping shown in Figure 2 using an API, do the following:

```
int flags = BCM_PORT_LANE_TO_SERDES_FABRIC_SIDE;
bcm_port_lane_to_serdes_map_get(unit, flags, 192, serdes_map);

serdes_map[4].serdes_rx_id=4; serdes_map[4].serdes_tx_id=6;
serdes_map[5].serdes_rx_id=5; serdes_map[5].serdes_tx_id=5;
serdes_map[6].serdes_rx_id=6; serdes_map[6].serdes_tx_id=7;
serdes_map[7].serdes_rx_id=7; serdes_map[7].serdes_tx_id=4;

bcm_port_lane_to_serdes_map_set(unit, flags, MAX_NUM_FABRIC_LANES, serdes_map);
```

**NOTE:** Keep the mapping for active lanes identical. Remapping of active ports is not allowed. Before you set the new mapping, detach (deactivate) the ports by using the `bcm_port_detach` API with all the ports on the same PM. For the correct sequence of link activation/deactivation, see Section 5.5, Power Efficiency.

### 5.8.5 Retimer Links

#### 5.8.5.1 Enabling the Retimer

To enable the retimer, use the BCM API `bcm_fabric_link_control_set` with control type `bcmFabricLinkRetimerConnect`:

```
   bcm_fabric_link_control_set(unit, source_link, bcmFabricLinkRetimerConnect, dest_link)
```

The API enables retimer bidirectionally. Configuration in both directions is not required.

The retimer cannot be enabled on lanes that are already configured as retimer.

**NOTE:**

- Before enabling the retimer on a pair of links make sure that both SerDes are locked against the link-partners.
- If error code BCM_E_INTERNAL is returned by the API when attempting to configure the loopback retimer, the user should detach and re-probe both relevant links.

#### 5.8.5.2 Disabling the Retimer

To disable the retimer, use the BCM API `bcm_fabric_link_control_set` with control type `bcmFabricLinkRetimerConnect`, setting `dest_link` to (-1).

The API disables retimer bidirectionally. Configuration in both directions is not required.

**NOTE:**

- During retimer mode, retimer links must be isolated. Thus:
  - Before enabling retimer on a link or a pair of links, the links must be isolated. Links can be isolated via link isolation SoC property or link isolation API.
  - After disabling retimer on a pair of links, the links must be de-isolated.
- If the retimer link is defined as FE2, according to `fabric_device_mode` SoC property, and its link partner is also defined as FE2, according to the same SoC property, an error in initialization occurs. To avoid this scenario, isolate the retimer links using the link isolation SoC property.

For more information about link isolation SoC property, refer to Section 2.6.2, Application Reference.

For more information about link isolation API, refer to Chapter 12, Graceful Shutdown.

### 5.8.5.3 Retimer Link Configuration Examples

The following table provides retimer link configuration examples.

**Table 12: Retimer Configuration Examples**

| Enable/ Disable | Required Configuration | Example |
|---|---|---|
| Enable | Enabling pass-through retimer between lanes 0 and 96 | `bcm_fabric_link_control_set(unit, 0, bcmFabricLinkRetimerConnect, 96)` |
| | Enabling loopback retimer between lanes 88 and 90 | `bcm_fabric_link_control_set(unit, 88, bcmFabricLinkRetimerConnect, 90)` |
| | Enabling loopback retimer on lane 100 (connect RX to TX) | `bcm_fabric_link_control_set(unit, 100, bcmFabricLinkRetimerConnect, 100)` |
| Disable | Disabling pass-through retimer, after connecting lanes 0 and 96 | `bcm_fabric_link_control_set(unit, 0, bcmFabricLinkRetimerConnect, -1)` |
| | Disabling loopback retimer, after connecting lanes 88 and 90 | `bcm_fabric_link_control_set(unit, 88, bcmFabricLinkRetimerConnect, -1)` |
| | Disabling loopback retimer on lane 100 (disconnect RX from TX) | `bcm_fabric_link_control_set(unit, 100, bcmFabricLinkRetimerConnect, -1)` |

### 5.8.5.4 Retimer FEC

To enable FEC on a retimer link, use the BCM API `bcm_fabric_link_control_set` with `bcmFabricLinkRetimerFecEnable` control.

- If FEC is enabled: FEC type is defined according to Section 5.2, Physical Coding Sublayer.
- If FEC is disabled: Any encoding, including 64/66, is removed from the link.

**NOTE:** This configuration can be set only prior to setting the link as retimer.

## 5.9 Diagnostics

### 5.9.1 Link Statistics

The `bcm_stat_get` and `bcm_stat_multi_get` functions collect link statistics per statistics category *bcm_stat_val_t* and link.

- snmpBcmTxControlCells – TX Control cell counter
- snmpBcmTxDataCells – TX Data cell counter
- snmpBcmTxDataBytes – TX Data byte counter
- snmpBcmRxCrcErrors – RX CRC error counter
- snmpBcmRxFecCorrectable – RX KR-FEC\RS-FEC correctable error (available for FEC and legacy FEC ports only.
- snmpBcmRxControlCells – RX Control cell counter
- snmpBcmRxDataCells – RX Data cell counter
- snmpBcmRxDataBytes – RX Data byte counter
- snmpBcmRxDroppedRetransmittedControl – RX dropped retransmitted control
- snmpBcmTxAsynFifoRate – TX Asyn FIFO rate at units of 80 bits
- snmpBcmRxAsynFifoRate – RX Asyn FIFO rate at units of 80 bits
- snmpBcmRxFecUncorrectable – RX RS-FEC/KR-FEC uncorrectable errors (available for FEC ports only)
- snmpBcmRxRsFecBitError – RX RS-FEC bit error counter.
- snmpBcmRxRsFecSymbolError – RX RS-FEC symbol error counter
- snmpBcmSignalQualityLastError- Get the last calculated bit error rate (BER)

  The snmpBcmSignalQualityLastError value is encoded as a mantissa and exponent, where mantissa is represented by 16 MSB unsigned short, and exponent is represented by 16 LSB signed short.

The function `bcm_stat_clear(unit, port)` clears port statistics.

The function `bcm_stat_clear_single(unit, port, type)` clears port statistics per statistics category.

**NOTE:** Using `bcm_stat_clear_single` to clear a statistics category on a port may impact the other statistics categories on the same port.

Because the FEC counters might continue counting when fabric ports are out of sync, a workaround exists to register a callback to clean up the problematic counters on a link-up event. For information, refer to `src/appl/reference/dnx/appl_ref_linkscan_init.c`. The following four statistics counters are relevant in this workaround:

- snmpBcmRxFecCorrectable
- snmpBcmRxFecUncorrectable
- snmpBcmRxRsFecBitError
- snmpBcmRxRsFecSymbolError

Statistics thread SoC properties:

- `bcm_stat_interval` – Statistics thread interval (microseconds).
- `counter_thread_pri` – Counter thread priority (0 is highest and 255 is lowest).
- `bcm_stat_sync_timeout` – Statistics thread sync operation timeout.

**Table 13: Diagnostics Shell Commands for Link Statistics**

| Command | Description |
|---|---|
| show counters | Displays main counters per each link. |
| show counters full | Displays all counters per each link. |
| clear counters | Clears statistics. |
| counter […] | Type `counter ?` for more information. |

## 5.9.2 Collectible Statistics

### 5.9.2.1 API

Counters that are not controlled by the counter thread can be obtained using the `bcm_fabric_stat_get` API, which receives the following arguments:

- Flags (currently there are no flags supported)
- Index

  The index argument is of type `bcm_fabric_stat_index_t` and encodes the statistic dimensions.

  Statistical data can be per:

  – Device

  – Link

  – Link and Pipe

**Table 14: Supported Index Types**

| Index Type | Marco |
|---|---|
| Device | `BCM_FABRIC_STAT_INDEX_DEVICE_SET()` |
| Link | `BCM_FABRIC_STAT_INDEX_LINK_SET(link)` |
| Link and Pipe | `BCM_FABRIC_STAT_INDEX_LINK_PIPE_SET(link, pipe)` |

- Statistic type

  All statistic type names comply with the following pattern: `bcmFabricStat[Index Type][Stat]`

  – `Index Type` – Indicates what macro should be used to create the index parameter (see the preceding table).

  – `Stat` – Name of the statistic.

  Statistic argument is of type `bcm_fabric_stat_t` and supports the following values:

  – `bcmFabricStatDeviceReachDrop` – Device Reachability drop counter

  – `bcmFabricStatDeviceGlobalDrop` – Device Global drop counter

  – `bcmFabricStatQueueLinkPipeCurrOccupancyBytes` – Current occupancy, per link and pipe in bytes

  – `bcmFabricStatQueueLinkMaxWmkLevel` – RCI watermark level per link
- Value

  The returned counter value is of type uint64.

### 5.9.2.2 Example Usage

```
uint64 value;
uint32 unit = 0;
uint32 flags = 0;
uint32 link = 100;
uint32 pipe = 0;
bcm_fabric_stat_index_t index;

/* Get counter value with index type - Device*/
index = BCM_FABRIC_STAT_INDEX_DEVICE_SET;
bcm_fabric_stat_get(unit, flags, index, bcmFabricStatDeviceReachDrop, &value);
bcm_fabric_stat_get(unit, flags, index, bcmFabricStatDeviceGlobalDrop, &value);

/* Get counter value with index type – Link and Pipe */
index = BCM_FABRIC_STAT_INDEX_LINK_PIPE_SET(link, pipe);
bcm_fabric_stat_get(unit, flags, index, bcmFabricStatQueueLinkPipeCurrOccupancyBytes,
&value);

/* Get counter value with index type – Link */
index = BCM_FABRIC_STAT_INDEX_LINK_SET(link);
bcm_fabric_stat_get(unit, flags, index, bcmFabricStatQueueLinkMaxWmkLevel, &value);
```

## 5.9.3 Link Status

The link status and the following fabric statistics can be gathered by using the `bcm_fabric_link_status_get` API:

- `BCM_FABRIC_LINK_STATUS_CRC_ERROR`: A cell with a CRC error was received on this link.
- `BCM_FABRIC_LINK_STATUS_SIZE_ERROR`: A data cell with the wrong size was received.
- `BCM_FABRIC_LINK_STATUS_MISALIGN`: Loss of FEC sync is detected on this link.
- `BCM_FABRIC_LINK_STATUS_CODE_GROUP_ERROR`: FEC uncorrectable error.
- `BCM_FABRIC_LINK_STATUS_NO_SIG_LOCK`: SerDes signal lock error.
- `BCM_FABRIC_LINK_STATUS_NO_SIG_ACCEPT`: No PMD lock in the SerDes.
- `BCM_FABRIC_LINK_STATUS_ERRORED_TOKENS`: The tokens are increased when cells with no CRC errors and no decoding errors arrive. The tokens are decreased when a faulty cell is detected. A value less than 63 indicates that an unacceptable amount of faulty cells were received on the link.
- `errored_token_count`: Error in the token count.

**NOTE:** A high rate of CRC, size or code group errors, or misalign signals indicates a bad link that should be shut down or requires SerDes tuning. An occurrence of these signals or errors may result in a one-time drop of one or more packets.

**NOTE:** If a *No Signal Lock* or *No Signal Accept* error is received, the first response should be to look at the DSC dump.

**Table 15: Diagnostics Shell Commands for Link Status**

| Command | Description |
|---|---|
| fabric link status | Displays link `status-stick status-cleared` on read. |

## 5.9.4 Link Loopbacks

The BCM88790 supports five types of loopbacks.

1. PHY xloop (external loop): By external card (does not require driver configuration).
2. PHY gloop (global loop): This loop is set on SerDes input. It is a deep loopback that includes all FIFOs and PCS layers.

   **NOTE**: PHY gloop loopback is applicable only if the TX lane swap is aligned.

   The `$SDK/src/examples/dnxc/cint_phy_loopback_on_lane_swapped_port.c` cint provides an example of the following sequence:

   a. Align the TX lane swap.
   b. Set the PHY gloop loopback.
   c. Restore the TX lane swap.
   d. Remove the PHY gloop loopback.
3. PHY rloop (remote loop): Loops the SerDes RX to TX, causing all traffic to be stopped on the SerDes receiving boundary and returned to the transmitting link partner.

   **NOTE**: PHY rloop loopback is applicable only if lanes are not swapped externally to the device.
4. MAC Async FIFO: Loop TX to RX before async FIFO.
5. MAC rloop (remote loop): MAC remote loop is implemented using retimer functionality.

Each lane can be configured to, at most, one loopback at a time.

### 5.9.4.1 Driver Reference

Loopbacks can be configured from BCM APIs or SoC APIs.

At the BCM level, use `bcm_port_loopback_set` to set a loopback.

This API receives one of the following:

- `BCM_PORT_LOOPBACK_MAC` (mapped to MAC async)
- `BCM_PORT_LOOPBACK_MAC_REMOTE` (mapped to MAC rloop)
- `BCM_PORT_LOOPBACK_PHY` (mapped to PHY gloop)
- `BCM_PORT_LOOPBACK_PHY_REMOTE` (mapped to PHY rloop)
- `BCM_PORT_LOOPBACK_NONE` (disables the loop)

Enabling a loopback type for a link disables all other loopback types previously configured for that link.

# Chapter 6: Fabric Traffic Prioritization

## 6.1 Fabric Pipes

The BCM88790 has data path pipes for data cells that can be operated in three modes:

- Single switch mode: Incoming data cells flow through the same pipe (same as the FE600). Each link has a dedicated input link FIFO shared with all data cells received from the link, and a dedicated output link FIFO shared for all data cells transmitted to the link.
- Dual switch mode: The data cell switch is replicated into two separate data path pipes (same as the FE1600), primary and secondary.
- Triple switch mode: The data cell switch is replicated into three separate data path pipes (same as the FE3200).

The multiple switch mode is designed to enhance support for unified fabric sharing TDM/OTN cross-connect traffic, multicast traffic, and unicast traffic on the device and on the same fabric links. Each pipe can be assigned to a specific cast (unicast/multicast) and priority.

Each pipe has dedicated resources (dedicated input FIFO per link per pipe, dedicated data cell switch per pipe, and dedicated output FIFO per link per pipe). Data cells received from each link may be switched into one of the pipes and are stored in dedicated input link FIFOs. Each pipe switches cells from its input link FIFOs into the destination output links and stores the cells in dedicated output link FIFOs. Each output link schedules (strict priority or WFQ) between the cells coming from the multiple pipes, and merges them into the output link.

Each pipe is configured independently (such as RCI and GCI flow control generation and thresholds, LLFC thresholds, drop thresholds, and so on).

The BCM88690/BCM88670/BCM88770 devices support a triple-pipe fabric interface, and can generate cells to all pipes as well as receive cells from all pipes.

BCM88650/BCM88750 devices support a dual-pipe fabric interface. Therefore, each BCM88650 can be mapped into two of the fabric pipes.

If a data switch is not used, its memory is used to extend the memory of used data switches, increasing overall performance.

## 6.2 Fabric Pipes Arbitration

Along the data paths, for every shared resource there is an arbiter that is scheduled between the three data paths. The scheduling is weight based with byte granularity.

The following Weighted Fair Queuing schedulers (WFQs) can be configured:

- Fabric ingress WFQ
- Fabric middle stage WFQ
- Fabric egress WFQ

The weight is in the range of 1–127 for each switch (for example, 1:127:4, 1:16:2, 2:3:1); the smaller the weight, the higher the bandwidth. Weight 0 stands for strict priority.

## 6.3 Cell Priority

Incoming cells are assigned with internal priority to associate them to the respective priority discard levels.

- TDM and Unicast cell priority is extracted from the cell header.
- Multicast cell priority can be defined using the cell multicast priority indication (supported by VSC256_V2), or by using a range of MC IDs.

  When Multicast cells are mapped according to Multicast-ID ranges, two ranges of multicast ID are defined: low-priority range and middle-priority range.

  – If only the low priority range is defined, a multicast cell that does not match the range is assigned priority 1.
  – If middle-priority range is defined, a multicast cell that does not match the ranges is assigned priority 2.
  – If no range is defined, a multicast cell that does not match the ranges is assigned the default non-TDM priority.

**NOTE:** The mode that maps MC-ID to MC cell priority is not supported when working in three fabric pipe mode where the pipes are assigned to UC, low priority MC, and high priority MC.

## 6.4 TDM/OTN Support

The BCM88790 supports 65 to 250 bytes VSC256 TDM/OTN cells. TDM/OTN cells are marked as TDM in the cell header using a predefined priority. Generally, priority 3 is marked as TDM priority, but it is possible to mark none or more than one priority as TDM.

One can assign a unique pipe for TDM/OTN traffic.

TDM/OTN cells are considered in the fabric as non-discardable and require low latency. Typically, they are routed to the unique data switch where they have their own resources, with adequate priority to the MAC-TX access.

If a unique pipe is not assigned to TDM/OTN cells, TDM/OTN cells are mapped into the data-shared pipe and get priority over other data cells. The BCM88790 has a configurable threshold per priority on an output link buffer, and as the threshold is crossed, non-TDM/OTN cells are dropped to ensure delivery of TDM/OTN cells.

## 6.5 Pipe Mapping

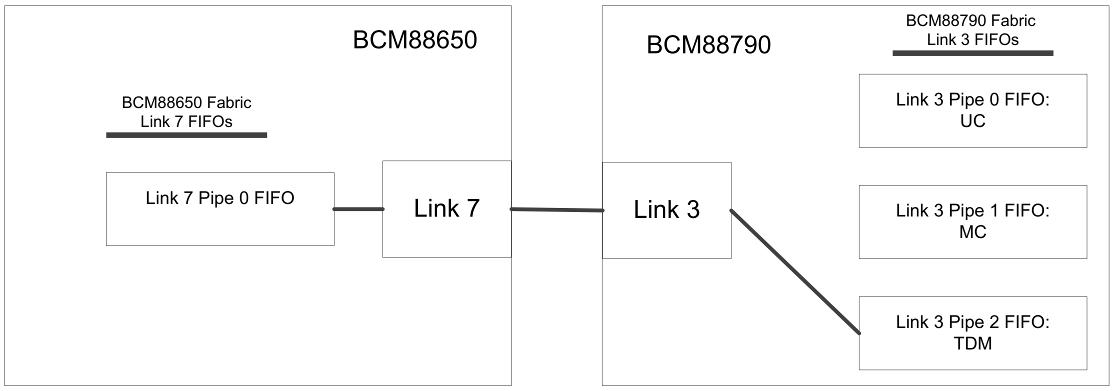
Generally, a packet that was assigned to one of the fabric pipes must stay in the same pipe through the entire fabric, source FAP, and the destination FAP. This means that the same mapping from cell cast and cell priority to fabric pipe should be configured in all devices in the system.

However, if a BCM88790 is connected to a legacy device (BCM88650 or BCM88750) supporting just a subset of the fabric pipes, a manual assignment must be set per the BCM88790 link. In such a case, the BCM88790 converts the cell format from the remote device format to the required format and vice versa. For example:

- A BCM88650 is configured to a single pipe mode and all traffic is TDM traffic: Pipe 0–TDM.
- A BCM88790 is configured to 3 pipes mode: Pipe 0–UC, pipe 1–MC pipe 2–TDM.
- BCM88650 link 7 is connected to BCM88790 link 3.

  In such a case, a manual assignment should be configured over BCM88790 link 3 from remote pipe 0 to local pipe 3 (see Figure 3).

**Figure 3: Manual Assignment**



## 6.6 Device Reference

### 6.6.1 Fabric Pipes

- Configure number of fabric pipes.
  - `fabric_num_pipes=<1|2|3>`
  - Default: 1
- Assign traffic cast (unicast/multicast) and priority (0/1/2/3) to a pipe.
  - `fabric_pipe_map=<0|1|2>`
  - Assigning a specific traffic cast is done by a suffix *uc* or *mc* and a number that represents the cell priority.

  **Example:**
  - `fabric_pipe_map_uc3=2`: Assigns UC priority 3 to pipe 2.
  - `fabric_pipe_map_mc=1`: Assigns MC (all priorities) to pipe 1.
  - `fabric_pipe_map=0`: Assigns all traffic types to pipe 0.

**NOTE:** The following notes apply:

- By default, UC\MC priority 3 represents UC\MC TDM. However, it is possible to configure which priorities are considered as TDM.
- The configuration is according to the longest prefix SoC property.
- Assigning UC priority 2 to pipe 3 and UC priority 0,1, and 3 to pipe 1 can be done by the following SoC properties:

```
fabric_pipe_map_uc2=3
fabric_pipe_map_uc=1
```

**NOTE:** Whenever more than one pipe is used, a pipe configuration must be explicitly set.

The following pipe mapping examples are supported:

- UC, MC, TDM (3-pipe system):
  - `fabric_pipe_map_uc=0`
  - `fabric_pipe_map_mc=1`
  - `fabric_pipe_map_uc3=2`
  - `fabric_pipe_map_mc3=2`
- UC, high priority MC (priority 3), low priority MC (priorities 0, 1, 2) (3-pipe system):
  - `fabric_pipe_map_uc=0`
  - `fabric_pipe_map_mc=2`
  - `fabric_pipe_map_mc3=1`
- UC, high priority MC (priorities 2, 3), low priority MC (priorities 0, 1) (3-pipe system):
  - `fabric_pipe_map_uc=0`
  - `fabric_pipe_map_mc=2`
  - `fabric_pipe_map_mc2=1`
  - `fabric_pipe_map_mc3=1`
- UC, high priority MC (priorities 1, 2, 3), low priority MC (priority 0) (3-pipe system):
  - `fabric_pipe_map_uc=0`
  - `fabric_pipe_map_mc=1`
  - `fabric_pipe_map_mc0=2`
- Non-TDM, TDM (2-pipe system):
  - `fabric_pipe_map=0`
  - `fabric_pipe_map_uc3=1`
  - `fabric_pipe_map_mc3=1`
- UC, MC (2-pipe system):
  - `fabric_pipe_map_uc=0`
  - `fabric_pipe_map_mc=1`

## 6.6.2 System Information

To determine if there is a multiple pipe device in the system:

```
system_contains_multiple_pipe_device = < 1 | 0 >
```

### 6.6.3 Fabric Pipes Arbitration

In all the following configurations, the variable gport_info.in_gport = port will specify for which of the eight DCS (Data Cell Switch) the configuration is done. This is done by internally mapping between port and DCS. To configure all ports/DCS use gport_info.in_gport = –1.

#### 6.6.3.1 WFQ Configuration

1. Retrieve the relevant handle:

   The following handles may be used:

   – Global fabric pipe handle: Configures all pipe WFQs in the device.
   ```
   gport_info.in_gport= port /*-1 for all ports*/;
   gport_info.cosq= pipe_index;
   bcm_cosq_gport_handle_get(unit, bcmCosqGportTypeFabricPipe, &gport_info);
   gport = gport_info. out_gport;
   ```
   – Ingress fabric pipe handle: Configures ingress fabric pipe shaper.
   ```
   gport_info.in_gport= port /*-1 for all ports*/;
   gport_info.cosq= pipe_index;
   bcm_cosq_gport_handle_get(unit, bcmCosqGportTypeFabricPipeIngress, &gport_info);
   gport = gport_info. out_gport;
   ```
   – Middle stage fabric pipe handle: Configures the middle stage fabric pipe shaper.
   ```
   gport_info.in_gport= port /*-1 for all ports*/;
   gport_info.cosq= pipe_index;
   bcm_cosq_gport_handle_get(unit, bcmCosqGportTypeFabricPipeMiddle, &gport_info);
   gport = gport_info. out_gport;
   ```
   – Egress fabric pipe handle – Configures egress fabric pipe shaper.
   ```
   gport_info.in_gport= port /*-1 for all ports*/;
   gport_info.cosq= pipe_index;
   bcm_cosq_gport_handle_get(unit, bcmCosqGportTypeFabricPipeEgress, &gport_info);
   gport = gport_info. out_gport;
   ```
2. Configure the required weight (1 – 0x7f):
   ```
   bcm_cosq_gport_sched_set(unit, gport, -1, 0, weight);
   ```

### 6.6.4 Multicast Cell Priority

Enable mapping internal multicast priority according to MC-ID ranges.

- `fe_mc_priority_map_enable = < 1 | 0 >`
  – 1: Map MC-ID to fabric priority
  – 0: extract MC cell priority from cell header.
- Default: 0

To configure multicast priority mapping according to MC-ID ranges:

Call `bcm_fabric_control_set` with various `bcm_fabric_control_t`.

- `bcmFabricMcLowPrioMin` and `bcmFabricMcLowPrioMax` set a range of multicast IDs as a low priority. The low-priority range is between `LowPrioMin` and `LowPrioMax`.
- `bcmFabricMcMidPrioMin` and `bcmFabricMcMidPrioMax` set a range of multicast IDs as a middle priority. The middle-priority range is between `MidPrioMin` and `MidPrioMax`.

## 6.6.5 TDM/OTN Support

Mark fabric priorities as TDM (should be globally configured over the system).

```
fabric_tdm_priority_min=<3 | NONE>
```

- NONE represents a system that does not support TDM and all the priorities can be used as regular priorities.
  Default: 3

## 6.6.6 Pipe Mapping

The manual assignment of fabric remote pipes to fabric local pipes is required if legacy devices support only a subset of the fabric pipes.

To configure a manual assignment, use `bcm_fabric_link_remote_pipe_mapping_set(int unit, bcm_port_t port, bcm_fabric_link_remote_pipe_mapping_t *mapping_config)`.

For example:

- BCM88650 is configured to a single pipe mode, and all traffic is TDM traffic: Pipe 0–TDM.
- BCM88790 is configured to 3-pipe mode: Pipe 0–UC, pipe 1–MC pipe 2–TDM.
- BCM88650 link 7 is connected to BCM88790 link 3.

In a case like this, configure a manual assignment over BCM88790 link 3 from remote pipe 0 to local pipe 3.

```
bcm_fabric_pipe_t  pipes[1];
bcm_fabric_link_remote_pipe_mapping_t config;
bcm_fabric_link_remote_pipe_mapping_t_init(&config);
/*Configure number if pipes supported by the remote device*/
config.num_of_remote_pipes = 1;
config.remote_pipe_mapping = pipes;

/*Map remote pipe 0 to local pipe 2*/
config.remote_pipe_mapping[0] = 2;

config.remote_pipe_mapping_max_size = 1;
bcm_fabric_link_remote_pipe_mapping_set(bcm88790_unit, 3, &config)
```

**NOTE:** The API `bcm_fabric_link_remote_pipe_mapping_set` affects traffic, as the port is disabled during API execution.

# Chapter 7: Link FIFOs and Flow Control

## 7.1 Overview

The BCM88790 has a Congestion Manager (CGM) that is responsible for:

- Managing the FE internal memory resources.
- Managing the cell flows over the FE.

The CGM uses a set of configurable static and profile-based thresholds. Two profiles are used to enable different enforcements between the links (for example, for different link speeds). In an FE13, two profiles are available for FE1 links and two for FE3 links.

For more details, refer to the CGM section in the BCM88790 data sheet (88790-DS1xx).

## 7.2 Driver Reference

### 7.2.1 Flow Control and Drop Thresholds

This section describes how to set and get the threshold configuration.

#### 7.2.1.1 Setting Threshold Configuration

Flow control and drop threshold values are configured using the `bcm_fabric_profile_threshold_set` API, which receives the following arguments:

- Unit
- Profile ID
- Threshold ID
- Flags
- Threshold Type
- Value

##### 7.2.1.1.1 Profile ID

Flow control thresholds are lumped into two profiles (0 and 1).

Profile ID determines which of the two profiles (0 or 1) is configured.

**NOTE:** Not all flow control thresholds support profiles (see Table 18 through Table 20 for support). Thresholds that do not support profiles must be configured with a Profile ID of zero value.

#### 7.2.1.1.2 Threshold ID

Each flow control threshold type has only one of the following identification types:

- Pipe
- Priority
- Pipe-Level
- Cast-Priority

Identification types are used to create a threshold ID. Threshold IDs identify a threshold in the device that can be set/get.

Simple threshold IDs are created by integer values, as the following table shows.

**Table 16: Integer Values for Simple Threshold IDs**

| Identification Type | Integer Values |
|---|---|
| Pipe | <0 to 2>, Set (–1) for all pipes |
| Priority | <0 to 3>, Set (–1) for all priorities |

The complex threshold IDs are created using the macros listed in Table 17.

**Table 17: Macros for Complex Threshold IDs**

| Identification Type | Macro |
|---|---|
| Pipe-Level | `BCM_FABRIC_TH_INDEX_PIPE_LEVEL_SET(pipe,level)` |
| Cast-Priority | `BCM_FABRIC_TH_INDEX_CAST_PRIO_SET(cast,priority)` |

Macro arguments:

- `pipe` – Specifies the pipe that the threshold is applied: <0 to 2>
  Set (–1) for all
- `level` – Specifies the level that the threshold is applied: <0 to 2>
  Set (–1) for all
- `priority` – Specifies the priority that the threshold is applied: <0 to 3>
  Set (–1) for all
- cast – Determines the type of cast that the threshold is applied:
  - `bcmCastUC` (unicast)
  - `bcmCastMC` (multicast)
  - `bcmCastAll` (unicast + multicast)

**NOTE:** Each threshold type has only one identification type. The threshold ID must be created using the correct macro or integer.

#### 7.2.1.1.3 Flags

The following flags are supported by the API:

- Local-route/non-local route FIFOs:
  - `BCM_FABRIC_LINK_TH_LR_ONLY` – Configure local route FIFOs only.
  - `BCM_FABRIC_LINK_TH_NLR_ONLY` – Configure non-local route FIFOs only.
  - If none of the flags are passed, a non-local route is configured.

**NOTE:** Not all flow-control thresholds support local-route FIFO configuration (see Table 18 through Table 20 for support).

- FE1/FE3 links:
  - `BCM_FABRIC_LINK_TH_FE1_LINKS_ONLY` – Configure FE1 links only.
  - `BCM_FABRIC_LINK_TH_FE3_LINKS_ONLY` – Configure FE3 links only.
  - If none of the flags are passed, both FE1 and FE3 links are configured.

#### 7.2.1.1.4 Threshold Types

All threshold type names comply with the following pattern: `bcmFabric[Stage][Name][Index]th`

- Stage – RX for DCH, Mid for DTM, TX for DTL, Shared for DFL
- Name – Threshold name
- Index – Pipe/prio/class (cast-priority)

Threshold types are described in the following table.

**Table 18: Drop/Admission Thresholds**

| Threshold Type | Identification Type | Local-Route FIFO Configuration | Profile |
|---|---|---|---|
| `bcmFabricMidTagDropClassTh` | cast-priority | — | Supported |
| `bcmFabricTxLinkLoadDropPipeTh` | pipe | — | Supported |
| `bcmFabricTxFragGuaranteedPipeTh` | pipe | — | Supported |
| `bcmFabricSharedDropClassTh` | cast-priority | — | — |
| `bcmFabricSharedMcCopiesDropPrioTh` | priority | — | Supported |
| `bcmFabricMidFragDropClassTh` | cast-priority | — | Supported |
| `bcmFabricSharedBankMcDropPrioTh` | priority | — | — |
| `bcmFabricTxTagDropClassTh` | cast-priority | Supported | Supported |
| `bcmFabricTxFragDropClassTh` | cast-priority | Supported | Supported |
| `bcmFabricRxFragDropPipeTh` | pipe | — | — |
| `bcmFabricMidMcCopiesFcPrioTh` | priority | — | — |
| `bcmFabricRxMcLowPrioDropPipeTh` | pipe | — | — |

**Table 19: Flow Control Generation**

| Threshold Type | Identification Type | Local-Route FIFO Configuration | Profiles |
|---|---|---|---|
| `bcmFabricRxLLFCFcPipeTh` | pipe | — | Supported |
| `bcmFabricTxFE1BypassLLFCFcPipeTh` | pipe | — | Supported |

**Table 20: FIFO Sizes**

| Threshold Type | Identification Type | Local-route FIFO Configuration | Profiles |
|---|---|---|---|
| `bcmFabricTxFifoSizePipeTh`[a] | pipe | Supported | Supported |
| `bcmFabricMidFifoSizePipeTh`[a,b] | pipe | — | Supported |
| `bcmFabricRxFifoSizePipeTh`[a] | pipe | — | Supported |

a. Threshold types `bcmFabricTxFifoSizePipeTh`, `bcmFabricMidFifoSizePipeTh`, and `bcmFabricRxFifoSizePipeTh` values must be divisible by 4.

b. Threshold type `bcmFabricMidFifoSizePipeTh` cannot modify threshold values for pipe 2. This threshold value is calculated automatically.

#### 7.2.1.1.5 Value

The threshold value to set. When set to (–1), the threshold functionally is disabled.

An example of threshold configuration is in the following location:

```
$SDK/src/bcm/dnxf/tune/cgm_tune.c
```

### 7.2.1.2 Getting Threshold Configuration

To get the flow control and drop threshold values for a specific profile, threshold ID, and threshold type, use the `bcm_fabric_profile_threshold_get` API, which receives the following arguments:

- Unit
- Profile ID
- Threshold ID
- Threshold Type
- Flags

The `bcm_fabric_profile_threshold_get` arguments are identical to the `bcm_fabric_profile_threshold_set` arguments.

The API returns the following:

- Value – The returned threshold value.

## 7.2.2 Attaching a Threshold Profile to a Link

This section describes setting links to a given threshold profile ID and getting the links attached to a given threshold profile ID.

### 7.2.2.1 Setting Links to a Given Threshold Profile ID

Associating links to a FIFO profile is done using the `bcm_fabric_link_profile_set` API, which receives the following arguments:

- Unit
- Profile ID
- Flags
- Links count
- Links

#### 7.2.2.1.1 Profile ID

Profile ID specifies to which of the two profile IDs (0 or 1) the links are attached.

#### 7.2.2.1.2 Links Count

This argument sets the number of links in the link array input argument.

#### 7.2.2.1.3 Links

This is an array of links that the given FIFO profile is attached to. If the Links count parameter is equal to the size of the links array, the profile is attached to all passed links.

An example of attaching a threshold profile to a link is in the following location:

`$SDK/src/bcm/dnxf/tune/cgm_tune.c`.

### 7.2.2.2 Getting the Links Attached to a Given Threshold Profile ID

To get all the links that are attached to a FIFO profile, use the `bcm_fabric_link_profile_get` API, which receives the following arguments:

- Unit
- Profile ID
- Flags
- Links Count Max

#### 7.2.2.2.1 Profile ID

This argument is the profile ID for which the information is retrieved.

#### 7.2.2.2.2 Links Count Max

This argument specifies the maximum number of links to be returned. If the retrieved number of links is bigger than this parameter, an error message is printed.

#### 7.2.2.2.3 Returns

The API returns the following information:

- Link Count – Returns the number of links associated with the given FIFO profile ID.
- Links – Returns an array of all FE links associated with the given FIFO profile ID.

# Chapter 8: Fabric Routing

## 8.1 Overview

The BCM88790 automatically maintains routing database. This database stores the output links through which each destination FAP-ID is reachable. The routing database is maintained according to the reachability control cells received from each of the input links. These reachability cells are sourced from the downstream devices.

## 8.2 Maximal FAP-ID

The BCM88790 supports up to 2K FAPs. Each FAP is represented by a FAP-ID from 0 to 2k-1. Generally, the BCM88790 receives reachability information in order to maintain its routing tables and publish a view of its routing tables. A definition of the actual maximum FAP-ID allows optimizing this mechanism and allows a faster reaction to changes in the routing table.

## 8.3 FE13 Local Routing

The BCM88790 FE13 supports local switching for both unicast and multicast traffic. A cell that arrives from the FAP can be switched back to the FAP side (if the destination FAP is connected directly to this FE13) without going through the FE2.

This feature enables the bandwidth to be reduced towards the FE2 and avoids oversubscribing the available fabric bandwidth towards the FE2.

## 8.4 Driver Reference

### 8.4.1 Maximal FAP-ID

The following API configures the maximal FAP-ID:

```
bcm_stk_module_max_set(unit, 0, max_module)
```

Default: 2047

### 8.4.2 FE13 Local Routing

The following SoC property enables/disables both unicast local routing and multicast local routing from FE1 to FE3.

```
fabric_local_routing_enable=<0 | 1>
```

#### 8.4.2.1 Unicast Local Routing

The following SoC property enables/disables unicast local routing from FE1 to FE3.

```
fabric_local_routing_enable_uc=<0 | 1>
```

#### 8.4.2.2 Multicast Local Routing

For multicast local routing explanation see Chapter 10, Multicast.

# Chapter 9: Topology

## 9.1 Diagnostics

### 9.1.1 Online Connectivity

This section describes the driver functions that retrieve the Online-Status from the device. Those functions return not only the information configured by the user, but also the real-time information from the device.

- `bcm_fabric_reachability_status_get`
  - Returns to the user the current reachability table.
  - The reachability table holds for each destination FAP and fabric link, whether the destination FAP is reachable through the link.
  - Links may not be reachable due to:
    - Physical error
    - Load-balancing consideration due to physical error in a far link
    - Missing line card or fabric card
- `bcm_fabric_link_connectivity_status_get` and `bcm_fabric_link_connectivity_status_single_get`
  - Logical presentation of the neighbor links
  - High-level status indications of the links
  - Driver enables displaying of the connectivity map that presents the immediate neighbors of the device. The connectivity map hold per each link:
    - Connected device ID
    - Type of device (FAP, FE1, FE2, FE3)
    - Link number at connected device (or BCM_FABRIC_LINK_NO_CONNECTIVITY if the connection is invalid)
  - There is also an indication of whether the connectivity map has changed since the last report and whether a new error has been detected since the last report.

**NOTE:** Valid only after init is done.

**Table 21: Diagnostics Shell Commands for Online Connectivity**

| Command | Description |
|---|---|
| fabric connectivity | Displays the connectivity map. |
| fabric reachability <module_id> | Displays the links which the relevant module is reachable from. |

# Chapter 10: Multicast

## 10.1 Overview

Multicasting is the ability to send a packet to a set of FAP devices. A multicast group is a group of destinations that require a copy of the incoming packet. It may represent a VLAN, IP-multicast group, or a subset of FAPs/PPs used for in-band switch broadcasting.

In the fabric-multicast scheme, the packets' cells are replicated at the FE2 and FE3 (three-stage fabric) stages. The cells are replicated up to the FAP level, where a single copy of the packet is reassembled. If a multicast group includes several ports that connect to the same FAP, then the FAP (or the packet processing device) replicates the packet to these ports.

The fabric supports up to 512K multicast IDs. Fabric multicast traffic is identified by the FAP according to its packet header. The packets are fragmented into cells, whose headers include the multicast-group ID and an indication that the cell is a fabric multicast cell.

## 10.2 Multicast Table Mode

The BCM88790 maintains a multicast table, which is set by the CPU. The multicast table supports up to 512K multicast groups; each group is identified by a unique MC-ID. The multicast table has an entry per MC-ID, each entry holding up to 192-bit vector. Each bit represents next stage device.

The multicast table supports one of the following six configuration options:

- 8 × 192 × 64K:
  - Support eight queries each clock.
  - Support up to 64K multicast IDs.
  - Support up to 192 next stage devices.
- 4 × 192 × 128K:
  - Support four queries each clock.
  - Support up to 128K multicast IDs.
  - Support up to 192 next stage devices.
- 8 × 96 × 128K:
  - Support eight queries each clock.
  - Support up to 128K multicast IDs.
  - Support up to 96 next stage devices.
- 2 × 192 × 256K:
  - Support two queries each clock.
  - Support up to 256K multicast IDs.
  - Support up to 192 next stage devices.
- 4 × 96 × 256K:
  - Support four queries each clock.
  - Support up to 256K multicast IDs.
  - Support up to 96 next stage devices.
- 2 × 96 × 512K:
  - Support two queries each clock.
  - Support up to 512K multicast IDs.
  - Support up to 96 next stage devices.

## 10.3 Fabric Multicast

In single-stage FE2 mode, the system supports a maximum of 192 FAP destinations, and each bit in the vector corresponds to a FAP-ID destination. When the BCM88790 receives a multicast cell, it accesses the multicast table using the MC-ID, and receives a vector indicating the FAP destinations that have to receive a copy of the cell. The device selects the destination FAP devices one by one. For each destination FAP, it accesses the unicast routing tables (as done for unicast cells) to determine the output links through which the destination FAP is reachable.

The BCM88790 device load-balances the multicast cells directed to the FAP, between all the reachable links. Per each copy, the device chooses one of the valid links connected to the FAP according to the load-balancing function result, and load-balancing information.

## 10.4 Multicast Modes

The multicast-routing table has two modes of operation:

- Direct-List-of-FAPs mode
- Indirect-List-of-FAPs mode

### 10.4.1 Direct-List-of-FAPs Mode

This mode is used in single-stage systems where there are, at most, 192/96 FAP devices (according to the MC table mode). The FAP-IDs of FAP devices in this mode are between 0 and 191/95.

This mode can also be used in three-stage systems (in FE3) where the FAP-IDs of the FAP destinations are within a range of 192 or 96 consecutive numbers (according to the MC table mode).

Each entry has a bitmap of FAP destinations, indicating all FAP destinations that are members of the MC-ID, That is, all FAP destinations that require a copy of a cell.

### 10.4.2 Indirect-List-of-FAPs

This mode should be used in three-stage systems (in FE2 and FE3), or in systems where the FAP-IDs are not within a range of 192 or 96 consecutive numbers (according to the MC table mode). This mode can also be used in a single-stage system.

Each entry in FE2 devices has a bitmap of LRCs that reach members in the MC-ID. Therefore, an LRC requires only a single copy of the MC cell. (the LRC-ID is assigned in the 0–191/95 range.)

Each entry in FE3 devices has a bitmap of LRC-IDs that connect to members of the MC-ID, that is, all FAP destinations requiring a copy. In FE3, the LRC is merely a one-to-one mapping from the FAP-ID to an Internal-FAP-ID in the 0–191/95 range.

To add a multicast group, configure all FE2 and FE3 devices in the system. FE2 devices in the system are configured with the same LRC bitmap. FE3 devices that connect to the same FAPs are set with the same bitmap of FAP internal IDs.

The instructions above are also valid when setting the multicast group by sending in-band configuration cells.

The LRC-ID and FAP internal IDs are determined by the user in the topology set configuration. For further information regarding the LRC and FAP internal ID, see Chapter 9, Topology.

## 10.5 Driver Reference

### 10.5.1 Multicast Table Mode

- `fe_mc_id_range=<64K | 128K_HALF | 128K | 256K_HALF | 256K | 512K_HALF>`
  - 64K – Supports eight queries at each clock, maximum of 64K multicast IDs, and up to 192 next-stage devices.
  - 128K_HALF – Supports eight queries at each clock, maximum of 128K multicast IDs, and up to 96 next-stage devices.
  - 128K – Supports four queries at each clock, maximum of 128K multicast IDs, and up to 192 next-stage devices.
  - 256K_HALF – Supports four queries at each clock, maximum of 256K multicast IDs, and up to 96 next-stage devices.
  - 256K – Supports two queries at each clock, maximum of 256K multicast IDs, and up to 192 next-stage devices.
  - 512K_HALF – Supports two queries at each clock, maximum of 512K multicast IDs, and up to 96 next-stage devices.

  Default: 128K_HALF

### 10.5.2 Multicast Mode

- `fabric_multicast_mode = <DIRECT/INDIRECT>`
  - Direct multicast is valid in systems where the FAP-IDs of the FAP destinations are within a range of 192 or 96 consecutive numbers (according to the MC table mode).
- Use `bcm_fabric_control_set` API with the control `bcmFabricControlDirectMcModidOffset` to configure the offset from which the FAP-IDs of the destination FAPs start. This means that the FAP-IDs are between offset and [offset+191/95] (according to the MC table mode).

**NOTE:**

- The `bcmFabricControlDirectMcModidOffset` feature is supported only in *direct* multicast mode
- Valid offset values are between 0 and 1856
- The offset value can be set only before the first multicast group is created

### 10.5.3 Multicast Group

1. Use `bcm_multicast_create` to create an empty multicast group.
2. Use `bcm_fabric_multicast_set` to set the member destinations of the multicast group.
   - When working in DIRECT mode, the destinations are the FAP-IDs that belong to the multicast group. In case modid offset is configured, subtract the offset from the FAP-ID prior to calling the API.

     Otherwise, the destinations are the LRCs that can reach FAP members of the multicast groups. A copy is replicated for each LRC and is load balanced across its links. The LRCs are as described in Chapter 9, Topology.
3. Set requirements:
   - After init, all multicast groups are set to not replicate cells.
   - To open or update multicast group, call the set function with the new/updated FAPs/IDs list.
   - To close the multicast group, call the `bcm_multicast_destroy` function.
   - In FE devices that do not take place in the cell's replication of a multicast group, the multicast group is expected to be empty.
   - When working in a direct mode, the `bcm_multicast_egress_set` function can be used. When using this function, the multicast can be configured without preconfiguration. In case modid offset is configured, subtract the offset from the FAP-ID prior to calling the API.

4. When setting several MC entries concurrently using the `bcm_fabric_multicast_multi_set` API, the time of nonsync between the FEs is increased.

   The `bcm_fabric_multicast_multi_set` allows setting MC groups in three stages to reduce the nonsync time to a minimum:

   a. Prepare the device for setting the tables (`flags==BCM_FABRIC_MULTICAST_SET_ONLY`).
   b. Commit operation – Actual write of MC table (`flags==BCM_FABRIC_MULTICAST_COMMIT_ONLY`).
   c. Status operation – Returns the status of the MC table. Waiting for the DMA-DONE ACK (`flags==BCM_FABRIC_MULTICAST_STATUS_ONLY`).

   When the DMA is available, the Set prepares the memory (allocation and writing) for the DMA operation. The Commit writes the enable bit, and the Status waits for the done indication.

   If the DMA is not available, the solution is not applicable. The *Set* and *Status* then do nothing. The *Commit* performs the sequential write. When calling the API with flags==0, it performs all steps.

   **CAUTION!** All three steps must be called, otherwise, the DMA operations stays in a nonstable state. After Warm Boot, `bcm_fabric_multicast_multi_set` must be called from the first stage.

   **NOTE:** When working in direct mode, the destinations in `bcm_fabric_module_vector_t` are the FAP-IDs that belong to the multicast group. If modid offset is configured, subtract the offset from the FAP-ID prior to setting it in `bcm_fabric_module_vector_t`.

# Chapter 11: Linkscan

## 11.1 Overview

The BCM88790 has the capability to scan the ports, monitoring for linkup and linkdown events. This is accomplished by a hardware linkscan (HW mode) or by a thread that constantly monitors the links status (SW mode). This feature allows calling user-supplied callbacks upon a link change event.

## 11.2 Driver Reference

### SoC Variables

- linkscan_thread_pri: Linkscan thread priority.
- bcm_linkscan_maxerr: The number of port update errors causing the bcm_linkscan module to suspend a port from update processing for the period of time set in `bcm_linkscan_errdelay`.
- bcm_linkscan_errdelay: The time, in microseconds, for which the `bcm_linkscan` module suspends a port from further updating processing after `bcm_linkscan_maxerr` errors are detected. After this delay, the error state for the port is cleared and normal linkscan processing resumes on the port.
- bcm_linkscan_pbmp: Specifies the ports at which `bcm_init` runs linkscan.
  Default: all
- bcm_linkscan_interval: The linkscan interval, in microseconds.
  Default: 250,000
- custom_feature_linkscan_remove_port_info: When a link status change is noticed, the registered handler is called. If this SoC property is enabled, no extra port info and only link status is included on the callback function. Using it can greatly improve the linkscan response time.
  Default: 0

### Dynamic Configuration

- Enable/disable links scanning by the API: `bcm_linkscan_enable_set`.

  Calling for `bcm_linkscan_enable_set` with nonzero scanning interval enables linkscan. This interval is used just in the SW mode linkscan.
- Setting a linkscan mode per port by the API `bcm_linkscan_mode_set` or per port bitmap by the API `bcm_linkscan_mode_set_pbm`

  The supported modes are:

  ```
  BCM_PORT_LINKSCAN_HW
  BCM_PORT_LINKSCAN_SW
  BCM_PORT_LINKSCAN_NONE
  ```
- Registering a callback is done by the API: `bcm_linkscan_register`.

**NOTE:** The callback function is performed in the context of the linkscan thread. It must not suspend indefinitely or invoke any API that may result in a deadlock.

## 11.3 Diagnostics

**Table 22: Diagnostics**

| Command | Description |
|---|---|
| linkscan [...] | Type `linkscan ?` for more information |

# Chapter 12: Graceful Shutdown

## 12.1 Overview

Graceful shutdown is the capability to update or reset an FE device in an operating system with minimal effect to traffic. The most common use of it is in removing fabric cards from the chassis for maintenance. It may also be used for more complex procedures, such as expanding a single-stage system to a multistage system.

## 12.2 Driver Reference

### 12.2.1 Device and Link Isolation

A link is isolated by calling `bcm_fabric_link_control_set` with the `bcmFabricLinkIsolate` control.

- `bcmFabricLinkIsolate = 1`: The link may be enabled in the SerDes level and in the MAC level. However, its link partner sees the link as disabled and does not use it for traffic distribution.
- `bcmFabricLinkIsolate = 0`: Undo the isolation operation.

The fabric is isolated by calling `bcm_fabric_control_set` with `bcmFabricIsolate` control.

- `bcmFabricIsolate = 1`: Isolation of the fabric element.
- `bcmFabricIsolate = 0`: Disable isolation of the fabric element.

This isolation can also be performed by toggling a dedicated pin on the board. If the device is isolated using the dedicated pin, some cases require users to perform the isolate and unisolate actions using the `bcm_fabric_control_set` API when reconnecting it back to the system. This procedure is not necessary when a cold boot or initialization/deinitialization sequence is performed.

To enable the pin functionality, use the following SoC property:

```
custom_feature_enable_gpd_with_pin = < 1 | 0 >
```

Default:

```
custom_feature_enable_gpd_with_pin = 0
```

# Chapter 13: Inband Clock Distribution

## 13.1 Overview

Similar to Synchronous Ethernet functionality, the BCM88790 can recover a clock from one of its SerDes and drive the clock to an external output clock interface. An external PLL can use the output clock to generate the reference clock for the SerDes transmitter. This functionally allows in-band distribution of a clock in the system from a central location through the FE devices toward all FAP devices in the system, which is useful in multichassis systems that require a common clock.

The BCM88790 is able to recover two clocks from any of the fabric SerDes, which can be used to support primary and secondary clock sources.

The following table lists the clock recovery which is dependent on the SerDes rate.

**Table 23: Recovered Clock Dependency per SerDes Rate**

| SerDes Rate | Recovered Clock |
|---|---|
| NRZ | SerDes rate divided by 20 |
| PAM4 | SerDes rate divided by 40 |

An additional logic-based clock divider can further divide the clock (range 2 to 16).

## 13.2 Driver Reference

### 13.2.1 API

To configure the primary clock, secondary clock and divider use:

```
bcm_switch_control_set(unit, inband_clock_distribution_control, arg).
bcm_switch_control_get(unit, inband_clock_distribution_control, arg).
```

The following `inband_clock_distribution_control` types are supported:

`bcmSwitchSynchronousPortClockSource`: Set/get port from which to recover the primary clock.

`bcmSwitchSynchronousPortClockSourceBkup`: Set/get port from which to recover the secondary clock.

`bcmSwitchSynchronousPortClockSourceDivCtrl`: Set/get divider for primary clock (Range 2 to 16).

`bcmSwitchSynchronousPortClockSourceBkupDivCtrl`: Set/get divider for secondary clock (Range 2 to 16).

### 13.2.2 SOC Properties

To configure the clocks:

```
sync_eth_clk_to_port_id_clk_<clk_id>=<port_id>
```

clk_id selects the clock: 0 = primary, 1= secondary.

To configure divider:

```
sync_eth_clk_divider_<clk_id>=<2-16>
```

clk_id selects the clock: 0 = primary, 1= secondary.

**NOTE:** If primary and secondary clocks are from the same fabric SerDes block, configuring different dividers is forbidden.

# Chapter 14: Inband Packet Path to CPU

## 14.1 Overview

The BCM88790 is capable to route packets to a specific CPU FIFO based on the destination ID that is stamped on the cell header. The BCM88790 has eight CPU FIFOs. A special FAP-ID is assigned to each FIFO. These FAP-IDs are published to the system by the reachability cells. As a result, all the relevant cells go through the same FE device (that owns the CPU FIFO).

These eight CPU FIFOs can be read by the CPU using built-in DMA engine. Each FIFO is connected to a specific FIFO-read DMA channel at the CMICX. This connection allows data to be extracted from the FIFO and quickly exported to the CPU. CMICX issues a POP command as soon as valid entries are available.

Inband packet path to CPU is applicable just for unicast cells smaller than 256 bytes (multicast cells to CPU are not supported). In addition, in order to not to affect the fabric multicast cells, FE CPU FIFOs should be picked in such a way that it does not affect the all-reachable vectors across the system. The all-reachable vector includes all the output links that can reach all active destination FAP devices (all-reachable). The all-reachable vector guarantees that all destinations of the multicast cell are reachable through the selected output link. Therefore, all-reachable ignored module ID range should be defined across all the system and special FAP-ID that is assigned to the FIFO CPU must be included in this range.

## 14.2 Device Reference

### 14.2.1 FIFO Read DMA

Use/do not use FIFO DMA:

```
fabric_cell_fifo_dma_enable=<1 | 0>
```

Default: 0

Size of the host memory stored allocated by the CPU:

```
fabric_cell_fifo_dma_buffer_size=<bytes size>
```

Default: 20480

The number of writes by the DMA until a threshold based interrupt is triggered.

```
fabric_cell_fifo_dma_threshold=<threshold_value>
```

The value 0 disables the number of write-based interrupts.

Default: 2

The amount of time in microseconds that passes from the first write by the DMA until a timeout based interrupt is triggered. Value 0 disables timeout based interrupts.

```
fabric_cell_fifo_dma_timeout=<timeout value>
```

Default: 1000

## 14.2.2 Multicast Fabric Routing Bitmap

Define the All-Reachable module ID ignored range.

All the modules from the max all_reachable module up to the max module define the all-reachable ignored range.

Max all-reachable module:

```
bcm_stk_module_max_set(unit, BCM_STK_MODULE_MAX_ALL_REACHABLE,
max_all_reachable_module)
```

Max module:

```
bcm_stk_module_max_set(unit, 0, max_module)
```

## 14.2.3 Receive CPU Packets

1. Register a callback function to be called when a packet received:
   ```
   bcm_rx_register(unit, name, -1, callback, priority, NULL, 0).
   ```
   The packet information will be included in the packet structure. The following fields are relevant: `dma_channel` (if `fifodma` is enabled), `src_mod`, `pkt_data`, `cos` (represents the pipe index) and `unit`.
2. Start packet reception for a given CPU FIFOs:
   ```
   bcm_rx_cfg_t_init(&cfg). /*Init struct*/
   cfg-> num_of_cpu_addresses = num_of_cpu_fifos /*up to 8 CPU FIFOs*/
   cfg-> cpu_address = cpu_fap_ids_array; /*CPU FIFO FAP-IDs - must be included in all-
   reachable ignored range*/
   bcm_rx_start(unit, &cfg).
   ```
3. Stop packet reception:
   ```
   bcm_rx_stop(unit, &cfg).
   ```

A cint example that explains how to create a path for the data CPU (the required configuration over both FE and FAP device) is in the following location:

```
$SDK/src/examples/sand/cint_fe_cpu_packets.c
```

**NOTE:**

- The device supports a simple packet parser to calculate the system headers and to provide a way to extract only the payload in the attached callback. The parser is automatically applied for received CPU packets. In this case, the driver updates pkt → pkt_len to hold the size of the payload only, where pkt → tot_len still holds the total size, including system headers.
- The device does not support packet reassembly. If a large packet is sent and requires several cells to be transmitted, the callback is called one time for each cell. Starting with SDK release 6.5.29, the application can use the `_idx` parameter of the `bcm_pkt_t` structure to reassemble the cells, if needed.

# Chapter 15: Warm Boot

## 15.1 Terminology

The Warm Boot is a process of restarting software on a live system, with minimal disruption to the data plane. The Warm Boot can be used to accomplish the following functions:

- In-Service Software Reload (ISSR): reloading the driver to the existing driver version.
- In-Service Software Upgrade (ISSU): reloading the driver to a newer driver version.

In both cases, the driver state is restored, so that applications can continue to read/update hardware using BCM API calls. In ISSR, the data plane is not affected, resulting in hitless switching. ISSU may involve changing the HW configuration to allow bug-fixes (for example, to implement modifications). Traffic-affecting configuration is controlled by the application, resulting in either no loss or well defined loss.

The driver state is maintained in a Software Database (SW DB). Restoring the driver state is achieved by the following methods:

- Internal recovery: Reading the current HW state from the device.
- Persistent memory: The application program saves and then restores the SW DB reflecting the driver state.
- Replay: Physical access to the device is disabled. BCM APIs are executed with parameters used for initial configuration. The device access is then reenabled. This process enables setting the driver state without affecting the device configuration.

If the driver state can be fully restored using the internal recovery method, the driver is recognized as supporting Level-1 warm boot. If the persistent memory method is also required, the driver is recognized as supporting Level-2 warm boot. The BCM88790 device supports Level-2 warm boot, and it does not support Level-1 warm boot.

**NOTE:** To use warm boot on a device, it should not be reset. This includes avoiding resetting the device from the board (using SYS_RST_N) between the warm boot sync and the actual warm boot.

## 15.2 Persistent Database Format

The database on the persistent storage is maintained in a file format. The user must create the storage file and register read and write callbacks for reading and writing to it. The user can register read and write callbacks that do operations other than writing the persistent storage file. This allows using custom storage formats maintained by the application. In addition, it allows maintaining multiple copies of the Software Database for redundancy.

## 15.3 Update Modes

The BCM88790 SDK updates the persistent storage according to the following:

Asynchronous (manual sync): During the API execution flow, in order to sync the persistent storage with current state of the SW DB, the application calls the dedicated API (see Section 15.4, Driver Reference).

## 15.4 Driver Reference

### 15.4.1 Compilation Flags

If a warm boot support is not required, the warm boot related source code can be excluded from the build.

**NOTE:** The SW DB is always allocated since it is required for normal driver operation.

To enable warm boot support, the following flag must be defined for SDK compilation:

- `BCM_WARM_BOOT_SUPPORT`

### 15.4.2 SoC Properties

The following SoC properties must be defined in order to support Warmboot for BCM88790:

- `warmboot_support =<on/off>` - disable/enable support per unit.
- `sw_state_max_size =<number>` - the amount of memory in bytes that is preallocated for the warmbootable SW databases of the SDK.

### 15.4.3 Pre-Init Configurations

Before applying the init sequence, the application must create the storage file and supply read/write callbacks using `soc_switch_stable_register().`

### 15.4.4 Driver API

Use the following sequence to perform the warm boot:

1. If working in an asynchronous mode, call `bcm_switch_control_set(bcmSwitchControlSync)` to copy the SW DB to the persistent storage.
2. Call `bcm_detach` to detach the current driver instance from the application.
3. Set the driver in a warm boot state by calling the macro `SOC_WARM_BOOT_START` (unit).
4. Call `bcm_attach` to reattach the driver to the application.
5. Unset warm boot state by calling the macro `SOC_WARM_BOOT_DONE`(unit).
   - `bcmSwitchStableSelect`: Returns the persistent storage location (that is, the stable_location SoC property value) (get only)
   - `bcmSwitchStableSize`: Returns the persistent storage size (that is, the stable_size SoC property value) (get only)
   - `bcmSwitchStableUsed`: Returns the actual usage of the persistent storage (get only)
   - `bcmSwitchWarmBoot`: Returns whether the system is in a warm boot bring-up state (get only)

To register user-defined read/write callbacks for custom persistent DB update, use the following API:

```
bcm_switch_stable_register(unit, read-callback, write-callback)
```

The callback prototype is as follows:

```
(*bcm_switch_read_func_t)(int unit, uint32 *buf, int offset, int nbytes)
```
Read

```
int (*bcm_switch_write_func_t)(int unit, uint32 *buf, int offset, int nbytes)
```
Write

## 15.5 BCM Diagnostics Shell

To initialize the BCM diagnostics shell application in the warm boot mode, set the boot flag:

```
BOOT_F_WARM_BOOT
```

It instructs the application to use `reload.soc` instead of `rc.soc`, and to perform warm boot initialization.

**Table 24: BCM Diagnostics Shell**

| Command | Description |
|---|---|
| warmboot […] | Enter `warmboot ?` for more information |

# Chapter 16: Easy Reload

## 16.1 Overview

Easy reload is a high-availability solution implemented in the BCM88790. Easy reloading performs the original bring-up sequence of the driver without writing to the hardware.

Easy reload does not maintain driver state. It is not guaranteed, therefore, driver state is fully restored by the end of the process. Please consult with Broadcom support if you consider using easy reload.

Easy reload, unlike Warm Boot, does not require non-volatile memory available.

To enable easy reload support, the following flag must be defined for the SDK compilation:

- `DBCM_EASY_RELOAD_WB_COMPAT_SUPPORT`

To load the device in an easy reload mode:

1. Change the boot flags (SOC_BOOT_FLAGS) to easy reload (BOOT_F_RELOAD).
2. Run BCM application.
3. Enable easy reload mode: xxreload on.
4. Replay: execute BCM APIs with parameters used for initial configuration.
5. Disable easy reload mode: xxreload off.

**NOTE:** xxreload is the BCM shell command. If a user does not use the BCM shell, the following macros can be used:

- `SOC_IS_RELOADING (unit)` – Get mode
- `SOC_RELOAD_MODE_SET (unit, reload_mode)` – Set mode

## 16.2 Diagnostics

**Table 25: Diagnostics**

| Command | Description |
|---|---|
| xxreload […] | Enter `xxreload ?` for more information |

# Chapter 17: Diagnostics Features

## 17.1 Diagnostics BCM Shell commands

The BCM shell is an interactive shell that allows users to access memory and registers on the switch devices and to call BCM API functions. The BCM shell is a part of the BCM SDK and can be used in customer systems. It is also used in BCM reference design systems.

The BCM shell makes it possible to execute:

- BCM shell commands
- Diagnostic APIs using the Cint Command Line Interpreter
- Diagnostics tests

## 17.2 Diagnostics Tests

The BCM88790 supports executing diagnostic tests (also referred to as TR tests) from the BCM shell.

**NOTE:** TR tests might put the device in reset during the test. When running TR tests, all threads that might access the device should be paused.

**NOTE:** Device configuration may be changed during TR tests, and reinitialization is required.

**Table 26: Diagnostic Test Command**

| Command | Description |
|---|---|
| TestRun <test-number> […] | Run specific `tr` test. Some tests require memory parameter. |

The following table shows a list of the tests you can run from the Broadcom SDK.

**Table 27: Broadcom SDK Test List**

| Test Number | Test Name | Description |
|---|---|---|
| 1 | Register reset defaults | Verify all register defaults are aligned between software and hardware. |
| 2 | PCI Compliance | — |
| 3 | Register read/write | Test register read/write. |
| 5 | BIST | Perform internal memory BIST. |
| 6 | Memories Test | — |
| 7 | Memories WR First Last Test | — |
| 8 | Memories Flip/Flop Test | — |
| 9 | Broadcast Registers Test | Test broadcast register write |
| 50 | Memory Fill/Verify | Memory=<memory name>: Specifies the memory name that will be tested.<br>Relevant Memories:<br>■ TBD |
| 51 | Memory Random Addr/Data | Memory=<memory name>: Specifies the memory name that will be tested.<br>Relevant Memories:<br>■ TBD |

**Table 27: Broadcom SDK Test List (Continued)**

| Test Number | Test Name | Description |
|---|---|---|
| 52 | Rand Mem Addr, write all | Memory=<memory name>: Specifies the memory name that will be tested.<br>Relevant Memories:<br>■ TBD |
| 60 | Linkscan MDIO | Check parallel MDIO access and linkscan, |
| 71 | Table DMA | Memory=<memory name>: Specifies the memory name that will be tested.<br>Relevant Memories:<br>■ TBD |
| 131 | Fabric snake test | Perform a stress test on the BCM88790 links. The test validates the link performance under a burst of cells.<br>Loopback=<loopback_name>: Valid <loopback_name> options are: PHY (default), MAC, and EXTERNAL<br>Action=<action_name>: Valid <action_name> options are: PREPARE_ONLY, RUN_ONLY, and ALL (default)<br>RunMode=<run_mode_name>: Valid <run_mode_name> options are: INFINITE, STOP, and NORMAL (default)<br>If a failure occurs, the test prints the relevant stage failure and/or the relevant interrupts.<br>**NOTE:**<br>■ The fabric snake test is supported when part of the links are in a power-off state, under the condition that powering-off is done in groups of eight links (all the ports of one Blackhawk core can either be on or off).<br>See Section 5.8.2.2, Dynamic Power On/Off for information regarding powering-off links.<br>■ All the powered-on links must be enabled. When using an external loopback, the links must also be up.<br>■ It is recommended to run the snake test after driver init. That is, do not run the test several times consecutively.<br>■ For the device to regain full functionality, it must be rebooted after an action is taken (PREPARE_ONLY, RUN_ONLY, or ALL).<br>■ To avoid seeing misleading counter statistics when executing a long hardware snake test, use the following sequence:<br>1. Boot the device<br>2. Wait 1 to 2 seconds<br>3. Clear the counters<br>4. Run tr 131 Loopback=<LB_TYPE> runmode=infinite<br>5. Wait as long as required.<br>6. Run tr 131 Loopback=<LB_TYPE> runmode=stop<br>■ When testing in systems with mixed loopbacks (for example, PHY and EXT), use the following sequence:<br>1. port loopback <pbmp> mode=<MAC\|PHY><br>2. tr 131 action=PREPARE_ONLY loopback=EXTERNAL<br>3. tr 131 action=RUN_ONLY |

**Table 27: Broadcom SDK Test List (Continued)**

| Test Number | Test Name | Description |
|---|---|---|
| 141 | DEINT INIT (SoC BCM) | Test soc/bcm/appl init deinit sequence. Deinit init Test Usage:<br>Options:<br>■ Repeat=<value>: The test will loop as many times as stated (default=1).<br>■ NoInit=<value>: 1: The test will not init the chip.<br>■ NoDeinit=<value>: 1: The test will not deinit the chip.<br>■ NoBcm=<value>: 1: The test will not run on BCM.<br>■ NoSoc=<value>: 1: The test will not run on SoC.<br>■ NoInt=<value>: 1: The test will not run on interrupts.<br>■ NoAppl=<value>: 1: The test will not run on application.<br>■ WarmBoot=<value>: 1: The test will perform warm reboot (default 0) |

## 17.3 Device Access

BCM Diag Shell commands are capable of accessing registers and memories (also referred as tables) and also the fields within them.

Register commands are Listreg, Getreg, Setreg, and Modreg. Memory commands are LISTmem, Dump, Write, and MODify.

For additional details on device access Diag Shell, refer to the “Examining Device Internals” section in the *BCM Diagnostic Shell* application note (StrataXGS-AN3xx).

## 17.4 Memory Built-In Self-Test

The device has a Memory Built-In Self-Test (MBIST) mechanism for internal memories. MBIST is invoked by software and performs a hardware test of the memories in the device. This test changes the contents of the memories and does not restore their original content.

MBIST is for testing memories when a device malfunction is suspected. Running MBIST is not mandatory. As described later in this section, with some effort it, is possible to run MBIST before each SDK run, but this is not the intention of the test.

After MBIST runs on a device, the device must be powered off or a system reset must occur (a hardware reset by deasserting the signal SYS_RST_N from the board). Without this action, the device may not work properly, and therefore it is enforced by software.

Powering off the device and performing a system reset of the device both involve a PCIe reset of the device, which brings down the PCIe link. Because of this, the customer BDE must support this operation if it occurs while the BDE and SDK are running.

A callback function skeleton, `dnxc_perform_system_reset()`, is available in `src/appl/reference/dnxc/appl_ref_board.c`. and can perform these operations automatically on the customer board. An example of how this function is registered in the sample application can be seen in the `dnxc_perform_system_reset()` in `src/appl/reference/dnxc/appl_ref_init_deinit.c`.

For the callback function to be used automatically after running MBIST and to allow the SDK to continue running later, use the SOC property `perform_system_reset_when_needed=1`.

If a device power off or system reset is not actually performed, the SDK will recognize this, fail, stop running, and not run when started again.

MBIST can be run by using an SOC property or by executing a command from the diagnostic shell:

- Use the `bist_enable` SOC property.

  When MBIST is enabled using this SOC property, it runs at the beginning of the device startup (in `init soc`). If no error is found, the device continues to be initialized and works properly after MBIST completes. If an error is found by MBIST, the device initialization will not continue. The `bist_enable` options are as follows:
  - `bist_enable=0` – Do not run MBIST. This is the default, and useful for attempting to work with faulty memory.
  - `bist_enable=1` – Run MBIST.
  - `bist_enable=2` – Run MBIST with increased verbosity. This option includes the name and run time for sub tests and shows all errors (and not only the first error). Displaying this information is useful on failed devices and helps Broadcom analyze the failure.
- Use the `tr 5` command to run MBIST for internal memories from the diagnostic shell. This command restarts the driver and runs MBIST during the driver start.

The MBIST error and informational messages are printed based on the logging configuration. Logging must be configured prior to running MBIST (before `init soc` if MBIST is invoked using the SOC property). Up to SDK release 6.3.1, configuring logging to print MBIST information is performed using the following command:

```
debug mbist verbose
```

If the MBIST code finds an error, the driver reports information useful for Broadcom to analyze the failure cause.

## 17.5 Temperature Monitor

There are four monitors embedded in the BCM88790. Each one monitors the temperature in a specific device quarter.

The following API retrieves the temperature monitor's current value and peak (maximum temperature from last read) value for each embedded monitor. The value unit is 0.1°C.

```
int
bcm_switch_temperature_monitor_get(
    int unit,
    int temperature_max,
    bcm_switch_temperature_monitor_t *temperature_array,
    int *temperature_count)
```

Parameters are as follows:

- `temperature_max` – Number of entries in temperature_array (should be 4 or bigger)
- `temperature_array` – Holds the retrieved current and peak temperature for each monitor.
- `temperature_count` – Number of BCM88790 temperature monitors.

**Table 28: BCM Diagnostics Shell**

| Command | Description |
|---|---|
| show pvt | Display temperature monitor |

## 17.6 Additional Diagnostics Shell Commands

**Table 29: Additional Diagnostics Shell Commands**

| Command | Description |
|---|---|
| `diag counters` | Optional command arguments:<br>`nz` – Do not print zero counters.<br>`packet_flow` – packet flow counter only.<br>`graphical` – graphical presentation. |
| `fabric link config` | Display per link configuration. |
| `fabric properties` | Display static SoC properties (properties that cannot be overridden by APIs). |
| `fabric traffic_profile` | Display the traffic profile (traffic cast and priority) that currently switched by BCM88790.<br>Note that using this diagnostic will affect the cell filter. |

# Appendix A: Debug Printing and Error Messaging

The BCM88790 driver supports logging according to the log verbosity level. The level is defined in the BCM and/or SoC level per module.

The following verbosity levels are supported:

- Error Trace: Enables printing error messages. When error occurs, the appropriate error message is displayed. Also, the calling stack to the function in which the error occurred is printed.
- Code Trace: Enables printing full trace of API calls. For each API in the calling sequence, “enter” and “exit” messages are printed.

  **Example:** BCM shell

To set error tracing in the fabric module, call the following sequence:

```
dbm bcm -all
dbm socdnx -all
dbm bcm +error
dbm bcm +fabric
dbm socdnx +error
dbm socdnx +fabric
```

# Related Documents

The references in this section may be used in conjunction with this document.

**NOTE:** Broadcom provides customer access to technical documentation and software through its Engineering Support Portal (ESP) and Downloads site.

For Broadcom documents, replace the “xx” in the document number with the largest number available in the repository to ensure that you have the most current version of the document.

| Document (or Item) Name | Number | Source |
|---|---|---|
| BCM8877X Data Sheet | 88770-DS1xx | ESP |
| *Network Switching Software Development Kit* | 56XX-PGxxx-R | ESP |
| *BCM Diagnostic Shell* | StrataXGS-AN3xx-R | ESP |
| *SerDes Configuration and Debugging Guide for StrataDNX™ 16-nm and 7-nm Devices* | DBG16S-AN1xx | ESP |

# Glossary

| Term | Description |
|---|---|
| BEC | Backward Error Correction |
| CRC | Cyclic Redundancy Check |
| FAP | Fabric Access Processor |
| FAP-ID | Unique system-level number identifying the FAP |
| FE | Fabric Element |
| FEC | Forward Error Correction |
| FIFO | First In First Out |
| GCI | Global Congestion Indication |
| LLFC | Link-Level Flow Control |
| LRC | Link Reachability Class |
| MAC | Media Access Controller |
| MC-ID | Multicast ID |
| PCP | Packet Cell Packing |
| RCI | Route-Congestion-Indication |
| SAND | Scalable Architecture for Networking Devices |
| SerDes | Serializer and deserializer |
| VSC | Variable-Sized Cells |

BROADCOM®