Silicom FM10420 RDIF Programmer Guide

-

Linux OS

REVISION HISTORY

Date	Change description
29-Jun-16	1.0 Initial document
29-Mar-17	1.2 Update rdif-6.0.10.7.5
01-May-17	1.3 Update get state registers
27-Sep-17	1.4 Update installation procedure
02-Nov-17	1.5 Fix mir command example
05-Nov-17	1.6 Update Température registres table Section 1.5.3
08-Nov-17	1.7 Working with VF ports
14-Nov-17	1.8 Update to support rdif-6.0.10.7.12
10-Jan-18	1.9 Update to support rdif-6.0.10.7.14
18-Feb-18	2.0 Add Vlan translation example, add PE3100G2DQiRM-Qx4
14-Mar-18	2.1 Fix example 3.4.4 3.4.2
28-Mar-18	2.2 Update to support rdif-6.0.10.7.17.1

1 INTRODUCTION

1.1 General

The subject of this programmer guide is the Silicom NIC with Switch capabilities Adapters based on Intel FM10420 chip in a standard PCIe form factor adapters with Redirect capabilities and w/o physical bypass capablitis .

Silicom's Bypass-Series adapters are designed with Bypass circuitry aimed at maximizing the network's up-time.

1.2 Purpose

This document describes the **Redirect functionality** of the Silicom redirect Intel-based product line and provides a functional description of the API available for use to control the **Redirect interfaces** via user-level application.

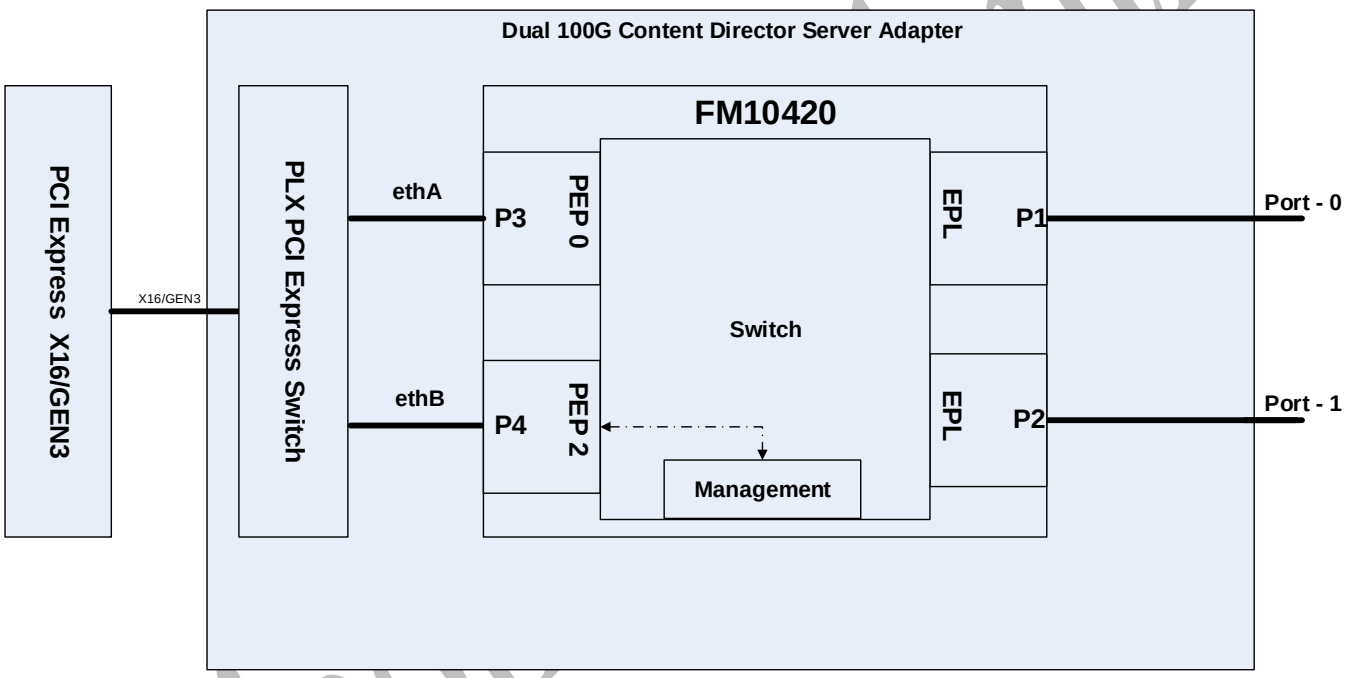
1.3 Scope

This document includes a description of the **Silicom Redirect function** and its operation, and a description of the **APIs supported by the product**, including the method of configuring the redirector function, accessing the redirector function, and responses returned from it. The document's appendix provides a list of capabilities and supported features for each product, sample commands and tools for operating the different controlling methods.

1.4 Product block-diagram and port connections

In this document, we provide instructions for managing the ports of the Intel FM10420 chip. To facilitate the configuration of the product, the diagram below describes the products ports connections.

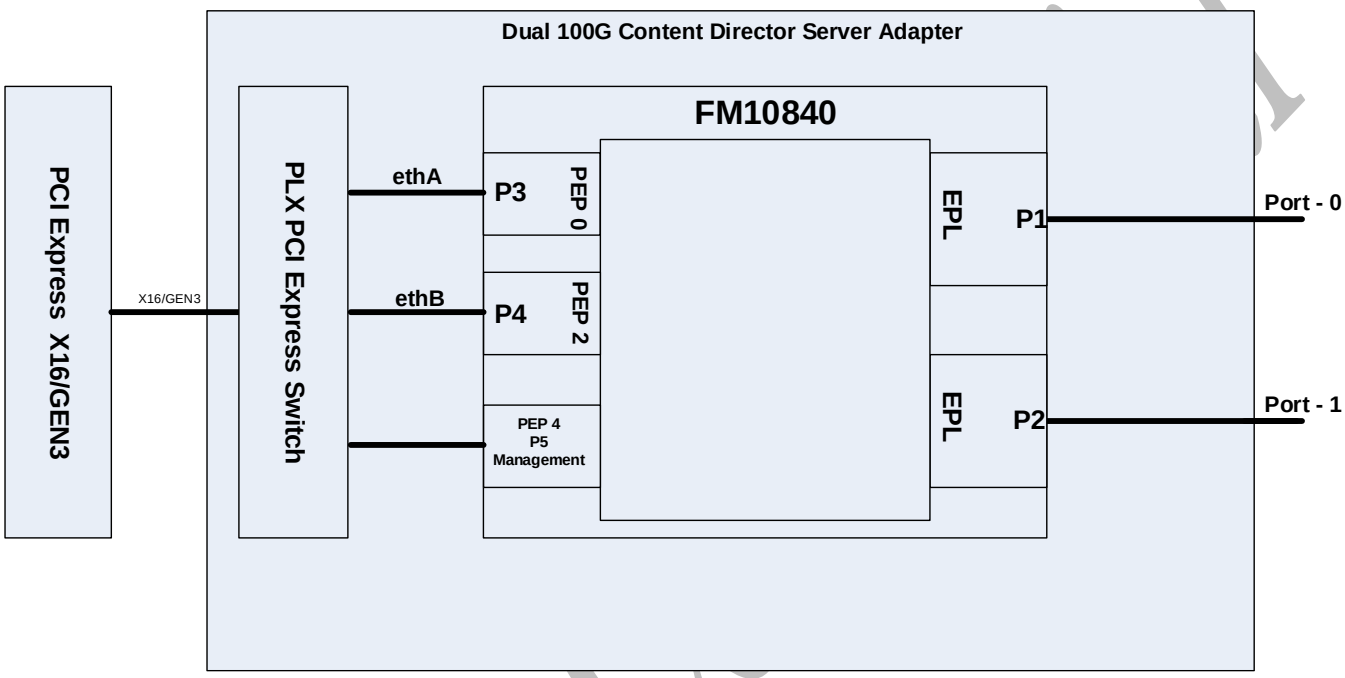
1.4.1 PE3100G2DQiRL / PE3100G2DQiR-QX4 - Dual port Fiber 100 Gigabit Ethernet PCI Express Content Director Server Adapter



Ports P1 and P2 refers to EPL ports of the Intel FM10420 (SFP connections). Ports P3 and P4 refers to PEP ports of Intel FM10420 PCI-E. Port P4 is also used for manage the FM10420 switch capabilities.

The traffic from each of the switch's ports to any other port (P1-P4) can be managed and configured independently. This means that it is possible to make a configuration such that traffic from any port can be routed to any other port of the Intel switch.

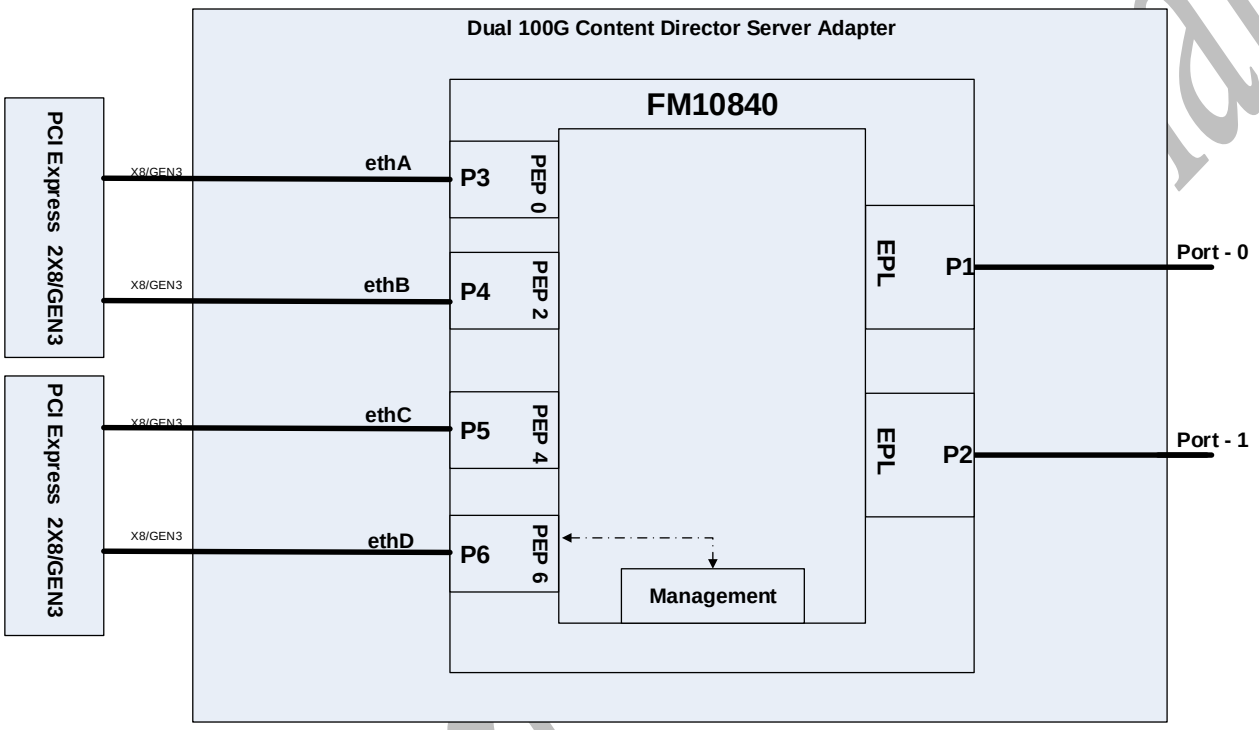
1.4.2 PE3100G2DQIR8-QX4: Dual port Fiber 100 Gigabit Ethernet PCI Express Content Director Server Adapter



Ports P1 and P2 refers to EPL ports of the Intel FM10840 (external ports). Ports P3 , Port 4 and P5 refers to PEP ports of Intel FM10820 PCI-E. Port P5 is used for manage the FM10840 switch capabilities.

The traffic from each of the switch's ports to any other port (P1-P5) can be managed and configured independently. This means that it is possible to make a configuration such that traffic from any port can be routed to any other port of the Intel switch.

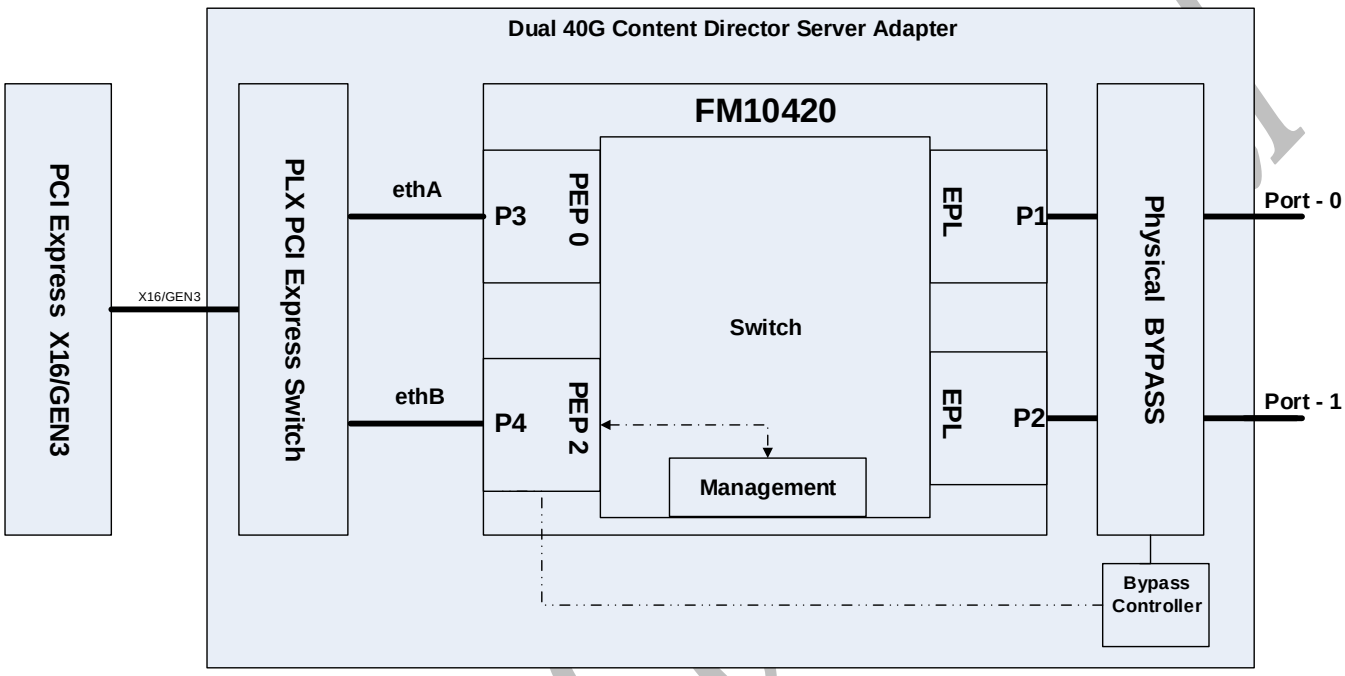
1.4.3 PE3100G2DQiRM-Qx4: Dual port Fiber 100 Gigabit Ethernet PCI Express Content Director Server Adapter



Ports P1 and P2 refers to EPL ports of the Intel FM10840 (external ports). Ports P3, P4, P5 and P6 refers to PEP ports of Intel FM10820 PCI-E. Port P6 is also used for manage the FM10840 switch capabilities.

The traffic from each of the switch's ports to any other port (P1-P6) can be managed and configured independently. This means that it is possible to make a configuration such that traffic from any port can be routed to any other port of the Intel switch.

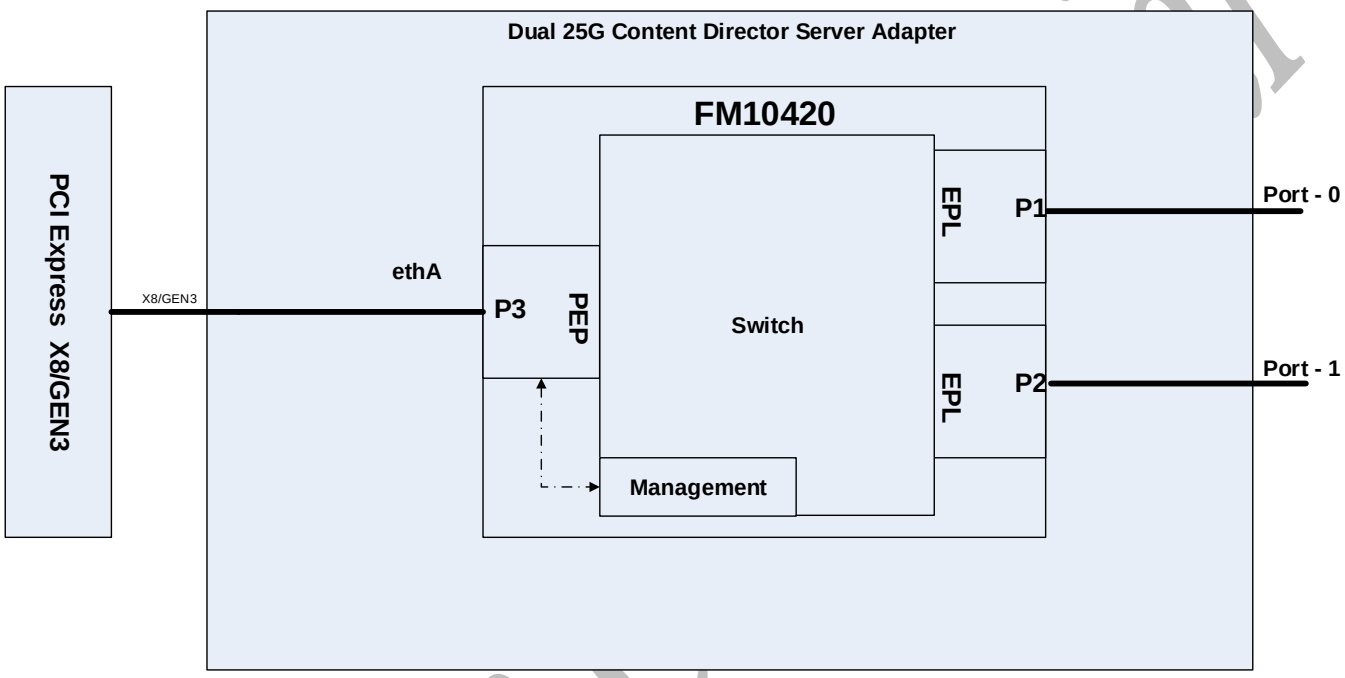
1.4.4 PE340G2DBiR: Dual port Fiber 40 Gigabit Ethernet PCI Express Content Director Bypass Server Adapter



Ports P1 and P2 refers to EPL ports of the Intel FM10420 (external ports). Ports P3 and P4 refers to PEP ports of Intel FM10420 PCI-E. Port P4 is also used for manage the FM10420 switch capabilities and management of the physical bypass capabilities.

The traffic from each of the switch's ports to any other port (P1-P4) can be managed and configured independently. This means that it is possible to make a configuration such that traffic from any port can be routed to any other port of the Intel switch.

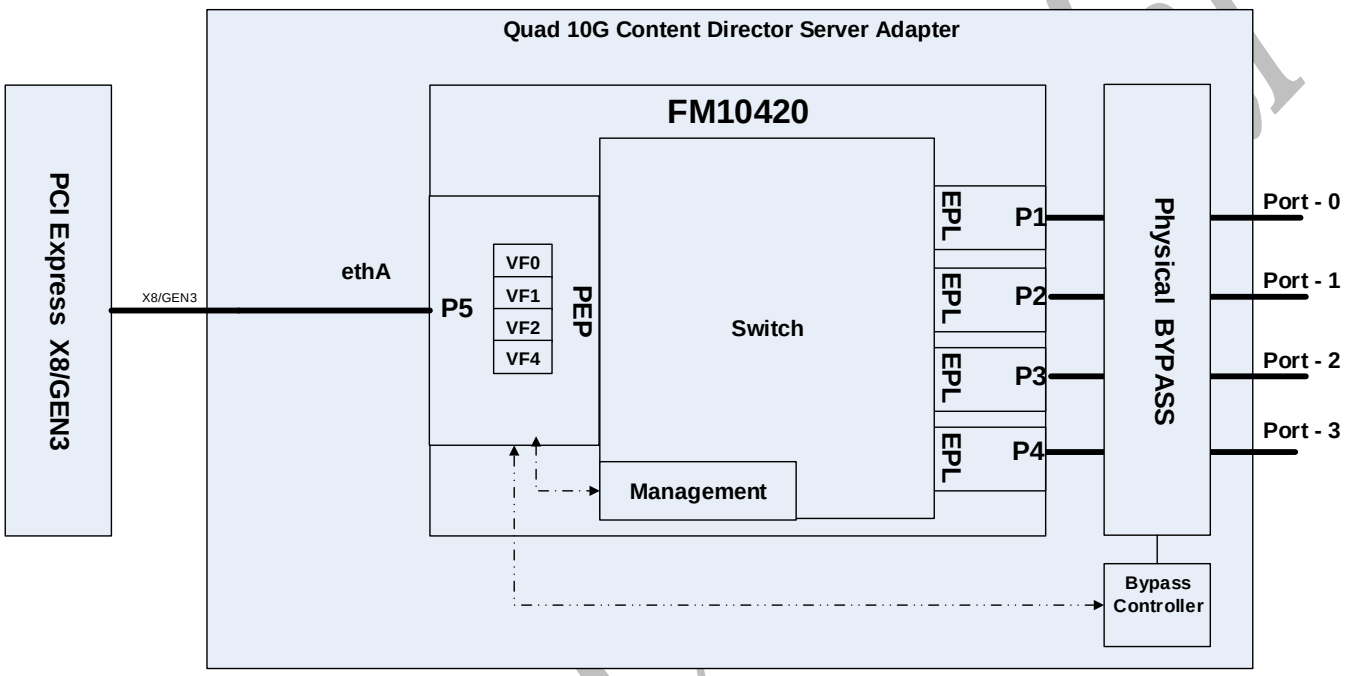
1.4.5 PE325G2DSiR-XR: Dual Port Fiber 25 Gigabit Ethernet PCI Express Content Director Server Adapter



Ports P1 and P2 refers to EPL ports of the Intel FM10420 (SFP connections). Ports P3 refers to PEP ports of Intel FM10420 PCI-E. Port P4 is also used for manage the FM10420 switch capabilities.

The traffic from each of the switch's ports to any other port (P1-P3) can be managed and configured independently. This means that it is possible to make a configuration such that traffic from any port can be routed to any other port of the Intel switch.

1.4.6 PE310G4DBiR: Quad port Fiber 10 Gigabit Ethernet PCI Express Content Director Bypass Server Adapter



Ports P1-P4 refers to EPL ports of the Intel FM10420 (external ports). Ports P5 refers to PEP ports of Intel FM10420 PCI-E. Port P5 is also used for manage the FM10420 switch capabilities and management of the physical bypass capabilities.

VF0-VF4 – are virtual ports, which can be created by the FM10K driver (it is possible to create up to 64 virtual ports).
The traffic from each switch's ports/virtual port to any other port/virtual ports (P1-P5 /V0-V4) can be managed and configured independently. This means that it is possible to make a configuration such that traffic can be routed to any other port/virtual port of the Intel FM10420 switch.

1.5 General configuration description

The configuration of the FM10420 switch includes 3 steps:

1. Definition of the allowed traffic flow with set_port_mask, defining to which egress port traffic from each ingress port can go.
2. Defining the main traffic path, within the boundaries defined in step 1(note that the rule ID of these rules needs to be higher than the rules on section 3)
3. Defining the rules to divert specific traffic flow from its main path.

1.5.1 Allowed traffic flow:

This section describes how to configure the traffic to be routed from ingress port to the egress ports (**default state - no traffic is allowed between the ports**).

The command used for this function is:
rdifctl set_port_mask Z A, B, C, D…
Where Z is the ingress port being configured, A, B, C, D defines the egress port that traffic is allowed to be routed to.

Sample:
rdifctl set_port_mask 1 4, 5, 6 will enable ingress traffic from port-1 to go to ports 4, 5, 6.

1.5.2 Defining the main traffic path:

In this section, we define the main traffic path within the boundary of the allowed egress ports.

Defining the main traffic path is accomplished via the establishment of a generic rule, followed by additional rules that can be located at the end of the rule list, after all the other specific rules, per section 1.5.2.

For example, if we want the main traffic coming into P2 to go to P0, then we issue the command:

rdifctl dir port 2 redir_port 0 rule_id <last>

***note that the rule ID of these rules needs to be higher than the rules on section 3**

1.5.3 Defining rules:

After defining the main traffic path, you need to define rules that change the path (defined in section 2).

Each rule has its own Rule ID. By default, the first rule named Rule ID 1, after which the numbering will continue forward, depending on the desired rule priority.
The first rule g the highest priority, and the priority declines as the rule ID number increases.

Rules can include Drop, Redir and Permit.
Classification can be define per Src/Dest-IP, Src/Dest-Port, Vlan, protocol type and others.
The list of available classifications described in section 2
Sample:
rdifctl dir port 2 redir_port 3 src_IP 168.16.1.0 rule_id 1 (packets with src IP 168.16.1.0 will be bypassed from P2 to P3).

1.6 Generic Configuration modes

Samples of different configuration modes provided in Appendix C.

Generally, a user can define his own specific configuration in any way desired by defining the port masking and per-rule traffic.

The initial port setting is that the no traffic is enabled between the ports. From that starting point, the user can mask and limit traffic with masking and rules as needed.

2 User mode Software API

This section defines the different calls for interfacing with the Redirector functionality.

2.1 Functions interface.

This section defines the user-level functions interface that can be used to interface the Redirector functions.
Together with the drivers provided on the product CD, we have included library files. These library modules are located on the product CD under Lib folder.

```
enum rdi_conf {
    RDI_INIT = 1,
    RDI_CLEAR,
    RDI_CLEAR_GROUP,
    RDI_SET_CFG,
    RDI_SET_DROP,
    RDI_SET_DIR,
    RDI_SET_MIR,
    RDI_GET_CFG,
    RDI_INSTALL,
    RDI_INSTALL_GROUP,
    RDI_GET_CNT,
    RDI_ENTRY_REMOVE,
    RDI_ENTRY_QUERY,
    RDI_ENTRY_QUERY_LIST,
    RDI_GET_DEV_NUM,
    RDI_SET_PORT_MASK,
    RDI_GET_PORT_MASK,
    RDI_SET_PERMIT,
    RDI_SET_MOD0 = 49,
    RDI_SET_MOD1 = 50,
    RDI_SET_MOD2,
    RDI_SET_L2_HASH,
    RDI_SET_L3_HASH,
    RDI_ADD_RULE,

    RDI_LBG_QUERY_LIST,
    RDI_LBG_PORT_QUERY_LIST,
    RDI_LBG_REMOVE,
    RDI_LBG_ADD,
```

```
    RDI_LBG_PORT_REMOVE,
    RDI_LBG_PORT_ADD,

    RDI_GET_L2_HASH,
    RDI_GET_L3_HASH,

    RDI_READ_PHY = 100,
    RDI_WRITE_PHY,
        RDI_CPLD_READ,
    RDI_CPLD_WRITE,

    RDI_GET_VLAN_STAT = 200,
    RDI_GET_STAT,
    RDI_GET_POWER,
} ;

enum rdi_action {
    RDI_ACT_PERMIT = 0,
    RDI_ACT_DROP = 1,
    RDI_ACT_TRAP,
    RDI_ACI_MIRROR,
    RDI_ACT_LOG,
    RDI_ACT_COUNT,
    RDI_ACT_NOTIFY,
    RDI_ACT_POLICE,
    RDI_ACT_SET_VLAN,
    RDI_ACT_SET_VLAN_PRI,
    RDI_ACT_SET_SWITCH_PRI,
    RDI_ACT_SET_DSCP,
    RDI_ACT_SET_USER,
    RDI_ACT_LOAD_BALANCE,
    RDI_ACT_TRAP_ALWAYS,
    RDI_ACT_REDIRECT,
    RDI_ACT_NOROUTE,
    RDI_ACT_ROUTE,
} ;

typedef struct rdi_id_list{
    unsigned int rule_num;
    short id_list[2048];
} rdi_id_list_t;
```

rdi_id_list_t:rule_num – total rules number.
rdi_id_list_t:id_list – rules ID array.

```
typedef struct rdi_query_list {
   rdi_id_list_t  rdi_id_list;
   int ret;
}rdi_query_list_t;

typedef struct rdi_mac_s {
   short flag;
   unsigned char mac[6];
} rdi_mac_t;
```

rdi_mac_t:flag – field enable flag; when it set, mac field value is used in rule qualifier list.
rdi_mac_t:mac – MAC value.

```
typedef struct rdi_ip6_s {
   short flag;
   unsigned char ip[16];
} rdi_ip6_t;



typedef struct rdi_mem {
   int group;
   int rule_id;
   int rule_act;
   int port;
   int redir_port;
   int src_port;
   int dst_port;
   unsigned int src_ip;
   unsigned int dst_ip;
   unsigned int src_ip_mask;
   unsigned int dst_ip_mask;
   int src_port_mask;
   int dst_port_mask;
   int src_port_max;
```

```
    int dst_port_max;
    int ip_protocol;
    int vlan;
    int vlan_mask;
    int vlan_max;
    int mirror_port;
    int mpls_type;
    int mpls_label;
    short mpls_exp_bits;
    short mpls_s_bit;
    int mpls_label_mask;
    short mpls_exp_bits_mask;
    short mpls_s_bit_mask;
    int ether_type;
    rdi_udf_t rdi_udf;
    rdi_mac_t src_mac;
    rdi_mac_t dst_mac;
    rdi_ip6_t src_ip6;
    rdi_ip6_t dst_ip6;
    rdi_ip6_t src_ip6_mask;
    rdi_ip6_t dst_ip6_mask;
    int vlan_act;
    int vlan_pri_act;
    int vlan_tag;
    int usr_act;
} rdi_mem_t;


typedef struct rdi_query_rule {
    rdi_mem_t rdi_mem;
    int ret;
} rdi_query_rule_t;

typedef struct rdi_id_list {
    unsigned int rule_num;
    short id_list[2048];
} rdi_id_list_t;


typedef struct rdi_query_list {
    rdi_id_list_t rdi_id_list;
```

```
    int ret;
}rdi_query_list_t;

typedef struct rdi_vlan_stat_cnt {
    unsigned long long vland;
    unsigned long long tvlan;
    unsigned long long tvland;
}rdi_vlan_stat_cnt_t;

typedef struct rdib_stat_cnt {
    unsigned long long total;
    unsigned long long txnoerror;
    unsigned long long rxnoerror;
    unsigned long long rxdrop;
    unsigned long long txdrop;
}rdib_stat_cnt_t;


typedef struct rdif_stat_cnt {
    unsigned long long  cntRxUcstPkts;
    unsigned long long  cntRxUcstPktsNonIP;
    unsigned long long  cntRxUcstPktsIPv4;
    unsigned long long  cntRxUcstPktsIPv6;
    unsigned long long  cntRxBcstPkts;
    unsigned long long  cntRxBcstPktsNonIP;
    unsigned long long  cntRxBcstPktsIPv4;
    unsigned long long  cntRxBcstPktsIPv6;
    unsigned long long  cntRxMcstPkts;
    unsigned long long  cntRxMcstPktsNonIP;
    unsigned long long  cntRxMcstPktsIPv4;
    unsigned long long  cntRxMcstPktsIPv6;
    unsigned long long  cntRxPausePkts;
    unsigned long long  cntRxCBPausePkts;
    unsigned long long  cntRxFCSErrors;
    unsigned long long  cntRxSymbolErrors;
    unsigned long long  cntRxFrameSizeErrors;
    unsigned long long  cntRxMinTo63Pkts;
    unsigned long long  cntRx64Pkts;
    unsigned long long  cntRx65to127Pkts;
    unsigned long long  cntRx128to255Pkts;
    unsigned long long  cntRx256to511Pkts;
```

```
unsigned long long  cntRx512to1023Pkts;
unsigned long long  cntRx1024to1522Pkts;
unsigned long long  cntRx1523to2047Pkts;
unsigned long long  cntRx2048to4095Pkts;
unsigned long long  cntRx4096to8191Pkts;
unsigned long long  cntRx8192to10239Pkts;
unsigned long long  cntRx10240toMaxPkts;
unsigned long long  cntRxFragmentPkts;
unsigned long long  cntRxUndersizedPkts;
unsigned long long  cntRxJabberPkts;
unsigned long long  cntRxOversizedPkts;
unsigned long long  cntRxGoodOctets;
unsigned long long  cntRxOctetsNonIp;
unsigned long long  cntRxOctetsIPv4;
unsigned long long  cntRxOctetsIPv6;
unsigned long long  cntRxBadOctets;
unsigned long long  cntRxPriorityPkts;
unsigned long long  cntRxPriorityOctets;
unsigned long long  cntTxUcstPkts;
unsigned long long  cntTxBcstPkts;
unsigned long long  cntTxMcstPkts;
unsigned long long  cntTxPausePkts;
unsigned long long  cntTxFCSErroredPkts;
unsigned long long  cntTxErrorDropPkts;
unsigned long long  cntTxTimeOutPkts;
unsigned long long  cntTxLoopbackPkts;
unsigned long long  cntTxMinTo63Pkts;
unsigned long long  cntTx64Pkts;
unsigned long long  cntTx65to127Pkts;
unsigned long long  cntTx128to255Pkts;
unsigned long long  cntTx256to511Pkts;
unsigned long long  cntTx512to1023Pkts;
unsigned long long  cntTx1024to1522Pkts;
unsigned long long  cntTx1523to2047Pkts;
unsigned long long  cntTx2048to4095Pkts;
unsigned long long  cntTx4096to8191Pkts;
unsigned long long  cntTx8192to10239Pkts;
unsigned long long  cntTx10240toMaxPkts;
unsigned long long  cntTxOctets;
unsigned long long  cntTxErrorOctets;
unsigned long long  cntTxCMDropPkts;
```

```
    unsigned long long  cntFIDForwardedPkts;
    unsigned long long  cntFloodForwardedPkts;
    unsigned long long  cntSpeciallyHandledPkts;
    unsigned long long  cntParseErrDropPkts;
    unsigned long long  cntParityErrorPkts;
    unsigned long long  cntTrappedPkts;
    unsigned long long  cntPauseDropPkts;
    unsigned long long  cntSTPDropPkts;
    unsigned long long  cntReservedTrapPkts;
    unsigned long long  cntSecurityViolationPkts;
    unsigned long long  cntVLANTagDropPkts;
    unsigned long long  cntVLANIngressBVPkts;
    unsigned long long  cntVLANEgressBVPkts;
    unsigned long long  cntGlortMissDropPkts;
    unsigned long long  cntFFUDropPkts;
    unsigned long long  cntPolicerDropPkts;
    unsigned long long  cntTTLDropPkts;
    unsigned long long  cntCmPrivDropPkts;
    unsigned long long  cntSmp0DropPkts;
    unsigned long long  cntSmp1DropPkts;
    unsigned long long  cntRxHog0DropPkts;
    unsigned long long  cntRxHog1DropPkts;
    unsigned long long  cntTxHog0DropPkts;
    unsigned long long  cntTxHog1DropPkts;
    unsigned long long  cntRateLimit0DropPkts;
    unsigned long long  cntRateLimit1DropPkts;
    unsigned long long  cntBadSmpDropPkts;
    unsigned long long  cntTriggerDropRedirPkts;
    unsigned long long  cntTriggerDropPkts;
    unsigned long long  cntTriggerRedirPkts;
    unsigned long long  cntTriggerMirroredPkts;
    unsigned long long  cntBroadcastDropPkts;
    unsigned long long  cntDLFDropPkts;
    unsigned long long  cntRxCMDropPkts;
    unsigned long long  cntUnderrunPkts;
    unsigned long long  cntOverrunPkts;
    unsigned long long  cntCorruptedPkts;
    unsigned long long  cntStatsDropCountTx;
    unsigned long long  cntStatsDropCountRx;
} rdif_stat_cnt_t;
```

```
typedef struct _rdi_hashRotationValue {
   /** The shift amount in the operation is one plus this value. */
   unsigned char   exponent;

   /** The amount the input hash value is multiplied by before shifting. */
   unsigned short mantissa;

} rdi_hashRotationValue;

#define RDI_L2_HASH_SMAC       1<<0
#define RDI_L2_HASH_DMAC       1<<1
#define RDI_L2_HASH_ETHER_TYPE  1<<2
#define RDI_L2_HASH_VLAN_ID     1<<3
#define RDI_L2_HASH_VLAN_PRI    1<<4
#define RDI_L2_HASH_SYM_MAC     1<<5
#define RDI_L2_HASH_USE_L3_HASH  1<<6
#define RDI_L2_HASH_L2_IF_IP     1<<7

typedef struct rdi_l2_hash {

   unsigned int l2_hash_set;
   /** Indicates the inclusion bit mask for the SMAC field.
    *  The valid range is 0 (disable) to 0xffffffffffff (all bits
    *  included).  The default is 0xffffffffffff.
    *
    *  Acts as a boolean
    *  and any value different then 0 will enable this specific key.
    *   */
   unsigned char src_mac_mask;
   /** Indicates the inclusion bit mask for the DMAC field.
    *  The valid range is 0 (disable) to 0xffffffffffff (all bits
    *  included).  The default is 0xffffffffffff.
    *
    *  Acts as a boolean
    *  and any value different then 0 will enable this specific key.
    */
   unsigned char dst_mac_mask;

   /** Indicates the inclusion bit mask for the EtherType field.
    *  The valid range is 0 (disable) to 0xffff (all bits
    *  included).  The default is 0xffff.
```

```
     *
     *  Acts as a boolean
     *  and any value different then 0 will enable this specific key.
     *  */
    unsigned char ether_type_mask;

    /** Indicates the inclusion bit mask for the VLAN ID 1 field.
     *  The valid range is 0 (disable) to 0xfff (all bits
     *  included).  The default is 0xfff.
     *
     *  Acts as a boolean
     *  and any value different then 0 will enable this specific key.
     */
    unsigned char vlan_id_mask;

    /** Indicates the inclusion bit mask for the VLAN Priority 1 field.
     *  The valid range is 0 (disable) to 0xf. The default is 0xf (all
     *  bits included).
     *
     *  Acts as a boolean
     *  and any value different then 0 will enable this specific key.
     */
    unsigned char  vlan_pri;


    /** Enable symmetrizing of the source and destination MAC fields.
     *  The default is FALSE.
     *
     *   */
    unsigned char  sym_mac;

    /** Include the L34 hash value in the L2 hash. The default is TRUE.
     */
    unsigned char use_l3_hash;

    /** Include the Layer 2 header in the L2 hash in case of an IPv4/IPv6
     *  packet. The default is TRUE.
     */
    unsigned char  use_l2_if_ip;

} rdi_l2_hash_t;
```

```
#define RDI_L3_HASH_SIP       1<<0
#define RDI_L3_HASH_DIP       1<<1
#define RDI_L3_HASH_SPORT  1<<2
#define RDI_L3_HASH_DPORT     1<<3
#define RDI_L3_HASH_DSCP      1<<4
#define RDI_L3_HASH_ISL_USR  1<<5
#define RDI_L3_HASH_PROTO     1<<6
#define RDI_L3_HASH_FLOW     1<<7
#define RDI_L3_HASH_SYM_L3_FIELDS     1<<8
#define RDI_L3_HASH_SYM_L4_FIELDS     1<<9
#define RDI_L3_HASH_ECMPROT     1<<10
#define RDI_L3_HASH_PROTOCOL1     1<<11
#define RDI_L3_HASH_PROTOCOL2     1<<12

#define RDI_L3_HASH_USE_PROTOCOL1    1<<13
#define RDI_L3_HASH_USE_PROTOCOL2     1<<14


typedef struct rdi_l3_hash {

   unsigned int l3_hash_set;

   /** Indicates the inclusion byte mask for the SIP field. Each bit of this
    *  mask indicates a full byte of the SIP with bit 0 corresponding to
    *  byte 0, bit 1 to byte 1, etc. The valid range is 0 (disable) to
    *  0xffff (all bytes included). The default is 0xffff.
    *                                                  \lb\lb
    *  Acts as a boolean
    *  and any value different then 0 will enable this specific key.
    *
    *   */
   unsigned char  src_ip_mask;

   /** Indicates the inclusion byte mask for the DIP field. Each bit of this
    *  mask indicates a full byte of the SIP with bit 0 corresponding to
    *  byte 0, bit 1 to byte 1, etc. The valid range is 0 (disable) to
    *  0xffff (all bytes included). The default is 0xffff.
    *                                                  \lb\lb
    *  Acts as a boolean
```

```
 *  and any value different then 0 will enable this specific key.
 *
 *   */
unsigned char  dst_ip_mask;

/** Indicates the inclusion bit mask for the layer 4 source port.
 *  The valid range is 0 (disable) to 0xffff (all bits included).
 *  The default is 0xffff.
 *                                                  \lb\lb
 *  Acts as a boolean
 *  and any value different then 0 will enable this specific key.
 *
 *   */
unsigned char  src_port_mask;

/** Indicates the inclusion bit mask for the layer 4 destination port.
 *  The valid range is 0 (disable) to 0xffff (all bits included).
 *  The default is 0xffff.
 *                                                  \lb\lb
 *  Acts as a boolean
 *  and any value different then 0 will enable this specific key.
 *
 *   */
unsigned char  dst_port_mask;

/** Indicates the inclusion bit mask for the DSCP field value.
 *  The valid range is 0 (disable) to 0xff (all bits included).
 *  The default is 0xff.
 *
 *   */
unsigned int dscp_mask;

/** Indicates the inclusion bit mask for the ISL_USER field value.
 *  The valid range is 0 (disable) to 0xff (all bits included).
 *  The default is 0.
 *
 *  */
unsigned int isl_usr_mask;

/** Indicates the inclusion bit mask for the layer 3 protocol field
 *  value. The valid range is 0 (disable) to 0xff (all bits included).
```

```
 *  The default is 0xff.
 *                                                  \lb\lb
 *  Acts as a boolean
 *  and any value different then 0 will enable this specific key.
 *
 *   */
unsigned char  proto_mask;


/** Indicates the inclusion bit mask for the IPv6 flow field value.
 *  The valid range is 0 (disabled) to 0xfffff (all bits included).
 *  The default is 0xfffff.
 *
 *   */
unsigned int  flow_mask;

/** Enable symmetrizing of the SIP & DIP fields. This ensures that frames
 *  with opposite SIP & DIP fields will hash the same with respect to
 *  those fields. The default is FALSE.
 *
 *   */
unsigned char  sym_l3_fields;

/** Enable symmetrizing of the layer 4 source and destination port fields.
 *  This ensures that frames with opposite source & destination port fields
 *  will hash the same with respect to those fields. The default is FALSE.
 *
 *   */
unsigned char    sym_l4_fields;

/** Specifies one of three 12-bit hash rotations for use in ECMP binning.
 *  Default is 0.
 *
 *  */
unsigned int  ECMPRotation;

/** When the useProtocol1 flag in this structure is set, the layer 4
 *  source and destination  will contribute to the hash only if the
 *  frame's protocol field matches the value of this attribute. The
 *  attribute value ranges from 0 to 255 with a default value of 1.
 *
```

```
     *   */
    unsigned int  protocol1;

    /** When the useProtocol2 flag in this structure is set, the layer 4
     *  source and destination  will contribute to the hash only if the
     *  frame's protocol field matches the value of this attribute. The
     *  attribute value ranges from 0 to 255 with a default value of 1.
     *
     *   */
    unsigned int  protocol2;

    /** Enable use of the configured protocol1 member of this structure into
     *  the L3 Hash calculation. Default is FALSE.
     *
     *  */
    unsigned char    useProtocol1;

    /** Enable use of the configured protocol2 member of this structure into
     *  the L3 Hash calculation. Default is FALSE.
     *
     *  */
    unsigned char    useProtocol2;

} rdi_l3_hash_t;



typedef union rdi_stat_cnt {
    rdib_stat_cnt_t rdib;
    rdif_stat_cnt_t rdif;
}rdi_stat_cnt_t;




typedef struct rdi_rule_stat_cnt {
    unsigned long long counter;
}rdi_rule_stat_cnt_t;
```

```
typedef struct rdi_rule_stat {
    rdi_rule_stat_cnt_t rdi_rule_stat_cnt;
    int ret_val;
}rdi_rule_stat_t;

typedef struct rdi_stat {
    rdi_stat_cnt_t rdi_stat_cnt;
    int ret_val;
}rdi_stat_t;

typedef struct rdi_vlan_stat {
    rdi_vlan_stat_cnt_t rdi_vlan_stat_cnt;
    int ret_val;
}rdi_vlan_stat_t;



typedef struct rdi_sfi_diag {
    unsigned short tx_power;
    unsigned short rx_power;
    unsigned int rsv[3];
} rdi_sfi_diag_t;

typedef enum rdi_type_s {
    RDI_BCM_DEV=1,
    RDI_FLCM_DEV
}rdi_type_t;



typedef struct if_rdi {
    int rdi_cmd;
    int unit;
    int cfg;
    rdi_mem_t rdi_mem;

    int if_index; /* network device index of management interface */

    int rule_id;
    int group;
```

```
    int port;
    unsigned int mask;
    rdi_query_list_t rdi_query_list;
    rdi_rule_stat_t rdi_rule_stat;
    rdi_vlan_stat_t rdi_vlan_stat;
    rdi_stat_t rdi_stat;
    rdi_l2_hash_t l2_hash;
    rdi_l3_hash_t l3_hash;
    int action;

    /* diagnostic fieds */
    int phy_addr;
    int addr;
    int dev;
    int val;

} if_rdi_t;



typedef struct rdi_lbg_list {
    int num;
    int list[16];
} rdi_lbg_list_t;


typedef struct rdi_lbg_query_list {
    rdi_lbg_list_t rdi_lbg_list;
    int ret;
}rdi_lbg_query_list_t;


typedef struct if_rdi_lbg {
    int rdi_cmd;
    int unit;
    rdi_lbg_query_list_t rdi_query_list;
    int lbg;
    int port;

} if_rdi_lbg_t;
```

```
typedef struct rdi_mask {
        unsigned char ingress[16];
        unsigned char egress[16];
        int ret_val;
} rdi_mask_t;

typedef struct if_rdi_mask {
   unsigned int rdi_cmd;
   unsigned int unit;
   rdi_mask_t mask;
} if_rdi_mask_t;
```

Redirector configuration parameters are passed in the rdi_mem structure. The fields of the rdi_mem structure are as follows:

Field	Description	Value/Note
rdi_mem:group	Group ID	
rdi_mem:rule_id	Rule ID	
rdi_mem:rule_act	Rule Action	RDI_SET_DROP, RDI_SET_DIR, RDI_SET_MIR from enum rdi_conf
rdi_mem:port	port to which the rule will be added	1…68, mandatory for rdi_add_rule_dir() and rdi_add_rule_drop() commands, default is 0
rdi_mem:mirror_port	port the matched packet mirrored to	1…68, mandatory for rdi_add_rule_mir() and command, default is 0
rdi_mem:redir_port	port the matched packet forwarded to	1…68, mandatory for rdi_add_rule_dir() and command, default is 0
rdi_mem:src_port	source L4 port	
rdi_mem:dst_port	destination L4 port	
rdi_mem:src_ip6	source IPv6 address	
rdi_mem:src_ip	source IPv4 address	
rdi_mem:dst_ip6	destination IPv6 address	
rdi_mem:dst_ip	destination IPv4 address	
rdi_mem:src_ip6_mask	source IPv6 address mask	

rdi_mem:src_ip_mask	source IPv4 address mask	
rdi_mem:dst_ip6_mask	destination IPv6 address mask	
rdi_mem:dst_ip_mask	destination IPv4 address mask	
rdi_mem:src_port_mask	Source L4 port mask	
rdi_mem:dst_port_mask	Destination L4 IP port mask	
rdi_mem:ip_proto	IP protocol number	
rdi_mem:vlan	VLAN tag	2-byte vlan tag includes vid, priority, cfi
rdi_mem:vlan_mask	VLAN tag mask	
rdi_mem:mirror_port	Mirror port	
rdi_mem:vlan_act	VLAN tag	VLAN tag for SET_VLAN command
rdi_mem:usr_act	VLAN tag	USR field in ISL tag for SET_USER command
rdi_mem:ether_type	EtherType field	
rdi_mem:src_mac	Source MAC	
rdi_mem:dst_mac	Destination MAC	

2.2 Loading the function library module

To compile and install this library module follow the steps below:

1. un-pack the **rdif_x.x.x** archive file
2. change into **rdif_x.x.x**. folder
3. run: **./install**

This will install the library module **librdif.so** in /usr/local/lib, daemon **rdifd** in /bin sample utility **rdifctl** and **rdif** script.
rdif script starts **rdifd** daemon.

To use the software and perform configuration commands:

1. Copy relevant CFG file (fm_platform_attributes.cfg), from the driver package under the RRC_BPD_xxx\Linux\Redirect\RD_RRC_Control\CFG_files\<product PN> directory to the \etc\rdi\ directory
2. **For Bypass adapters: start BP driver (bprdctl_start) before loading the rdif**
3. Load / unload the redirector driver and start / stop **rdifd** daemon: **rdif start / rdif stop.**
4. Configure redirector device: **rdifctl** - see Appendix C
5. Start / stop **rdifd** daemon: **rdifd / rdifd stop.**
6. Run rdifd daemon in verbose mode: **rdifd –v.**

2.3 Basic Commands

2.3.1 rdi_set_port_mask()

Description - Set egress port mask.

Syntax:

int rdi_set_port_mask (int unit , rdi_mask_t *mask);

Input: unit - number of the rdi device,
mask – egress & ingress ports mask

Output: **-1** on failure,
0 on success

2.3.2 rdi_get_port_mask()

Description - Get egress port mask.

Syntax:

int rdi_get_port_mask (int unit , int port, rdi_mask_t *mask);

Input: unit - number of the rdi device,
mask – egress & ingress ports mask

Output: **-1** on failure,
0 on success
mask

2.3.3 rdi_set_cfg()

Description - Set Redirector device to a predefined configuration (Definitions).

Syntax:

int rdi_set_cfg (int unit , int cfg_mode);

Input: unit - number of the rdi device,
enum rdd_cfg value

Output: **-1** on failure,
0 on success

2.3.4 rdi_get_cfg()

Description – Get current configuration mode (Definitions)

Syntax:

int rdi_get_cfg (int unit);

Input: unit - number of the rdi device,

Output rdi_conf value on success (according to enum rdd_cfg),
-1 on failure

2.3.5 rdi_add_rule_drop()

Description – Drop matching incoming packets for specific rdi_mem:port.

Syntax:

int rdi_add_rule_drop(int unit, rdi_mem_t * p_rdi);
Input: **unit** - number of the rdi device,
pointer to struct rdi_mem,
Output: **-1** on failure,
Rule ID on success

2.3.6 rdi_add_rule_permit()

Description – Permit matching incoming packets for specific rdi_mem:port.

Syntax:

int rdi_add_rule_drop(int unit, rdi_mem_t * p_rdi);
Input: **unit** - number of the rdi device,
pointer to struct rdi_mem,
Output: **-1** on failure,
Rule ID on success

2.3.7 rdi_add_rule_dir()

Description: Redirect matching packets from rdi_mem:port to rdi_mem:redir_port.

Syntax:

int rdi_add_rule_dir(int unit, rdi_mem_t * p_rdi);

Input: **unit** - number of the rdi device,
pointer to struct rdi_mem,

Output: **-1** on failure
Rule ID on success.

2.3.8 rdi_add_rule_mir()

Description: Copying matching packets from rdi_mem:port to rdi_mem:mirror_port.

Syntax:

int rdi_add_rule_mir(int unit, rdi_mem_t * p_rdi);

Input: **unit** - number of the rdi device,
pointer to struct rdi_mem,

Output: **-1** on failure
Rule ID on success.

2.3.9 rdi_add_rule()

Description: Adding rule from rdi_action list.

Syntax:

int rdi_add_rule(int unit, rdi_mem_t * p_rdi, int action);

Input: **unit** - number of the rdi device,
pointer to struct rdi_mem,
int action – rdi_action enum

Output: **-1** on failure
Rule ID on success.

2.3.10 rdi_clear_rules()

Description – Clear all rule stack and return to defined configuration mode.

Syntax:

int rdi_clear_rules(int unit);

Input: **unit** - number of the rdi device,

Output: ***-1*** on failure
0 on success.

2.3.11 rdi_clear_rules_group()

Description – clear all stacks for a specific group.

Syntax:

int rdi_clear_rules_group(int unit);

Input: **unit** - number of the rdi device,
group – group number,

Output: ***-1*** on failure
0 on success.

2.3.12 rdi_entry_remove()

Description – Remove specific rule from hardware tables.

int rdi_entry_remove (int unit, int rule_id);

Input: **unit** - number of the rdi device,
rule_id value,

Output: **-1** on failure,
0 on success

2.3.13 rdi_entry_query()

Description – Query information about specific rule.

int rdi_entry_query (int unit, struct rdi_mem *rdi_mem);

Input: **unit** - number of the rdi device,
struct rdi_mem *rdi_mem, rule_id is mandatory,

Output:
struct rdi_mem *rdi_mem
-1 on failure,
0 on success

2.3.14 rdi_entry_query_list()

Description – Query rule_id list.

int rdi_entry_query_list (int unit, struct rdi_query_list *rdi_query_list);

Input: **unit** - number of the rdi device,

Output:
struct rdi_query_list *rdi_query_list
-1 on failure,
0 on success

2.3.15 rdi_get_rule_counters()

Description – Query packets counter for specific rule.

int rdi_get_rule_counters(int unit, int rule_id, void *val);

Input: **unit** - number of the rdi device,
rule_id value,

Output:
val – 64-bit counter value
-1 on failure,
0 on success

2.3.16 rdi_get_stat ()

Description – Query per port statistics.

int rdi_get_stat(int unit, int port, rdif_stat_cnt_t *rdi_stat);

Input: **unit** - number of the rdi device,
port value,

Output:
rdi_stat,
-1 on failure,
0 on success

2.3.17 rdi_get_rule_stat ()

Description – Query per port statistics.

int rdi_get_stat(int unit, int group, rdi_rule_stat_cnt_t *rdi_stat);

Input: **unit** - number of the rdi device,
group - group number,

Output:
rdi_stat,
-1 on failure,
0 on success

2.3.18 rdi_set_port_mask()

Description - Set egress port mask.

Syntax:

int rdi_set_port_mask (int unit , int port, unsigned int mask);

Input: unit - number of the rdi device,
port – ingress port
mask – egress port mask

Output: **-1** on failure,
0 on success

2.3.19 rdi_get_port_mask()

Description - Get egress port mask.

Syntax:

int rdi_set_port_mask (int unit , int port, unsigned int *mask);

Input: unit - number of the rdi device,
port – ingress port

Output: **-1** on failure,
0 on success

2.3.20 rdi_set_l2_hash()

Description - Set L2 hash.

Syntax:

int rdi_set_port_mask (int unit , rdi_l2_hash_t *l2_hash);

Input: unit - number of the rdi device,
rdi_l2_hash_t *l2_hash

Output: **-1** on failure,
0 on success

2.3.21 rdi_set_l3_hash()

Description - Set L3 hash.

Syntax:

int rdi_set_port_mask (int unit , rdi_l3_hash_t *l3_hash);

Input: unit - number of the rdi device,
rdi_l3_hash_t *l3_hash

Output: **-1** on failure,
0 on success

2.3.22 rdi_lbg_query_entry_list ()

Description – Query list all existing load balancing groups.

int rdi_lbg_query_entry_list (int unit, struct rdi_lbg_query_list *rdi_query_list);

Input: **unit** - number of the rdi device,

Output:
struct rdi_query_list *rdi_query_list
-1 on failure,
0 on success

2.3.23 rdi_lbg_port_query_entry_list ()

Description – Query list all member ports in a load balancing group.

int rdi_lbg_query_entry_list (int unit, int lbg, struct rdi_lbg_query_list *rdi_query_list);

Input: **unit** - number of the rdi device,
lbg – number of the LBG

Output:
struct rdi_query_list *rdi_query_list
-1 on failure,
0 on success

2.3.24 rdi_lbg_add ()

Description – Create a load balancing group.

int rdi_lbg_add (int unit, int *lbg);

Input: **unit** - number of the rdi device,

Output:
lbg – number of the LBG
-1 on failure,
0 on success

2.3.25 rdi_lbg_remove ()

Description – Delete a load balancing group.

int rdi_lbg_add (int unit, int *lbg);

Input: **unit** - number of the rdi device,
lbg – number of the LBG

Output:
-1 on failure,
0 on success

2.3.26 rdi_lbg_port_add ()

Description – Add port to the LBG.

int rdi_lbg_port_add (int unit, int lbg, int port);

Input: **unit** - number of the rdi device,
lbg – number of the LBG
port – port number

Output:
-1 on failure,
0 on success

2.3.27 rdi_lbg_port_remove ()

Description – Delete a LBG member port.

int rdi_lbg_port_remove (int unit, int lbg, int port);

Input: **unit** - number of the rdi device,
lbg – number of the LBG
port – port number

Output:
-1 on failure,
0 on success

2.3.28 rdi_set_mod1()

Description - Set Redirector to mod1: ISL tagging for all ports and LBG on internal (host) ports.

Syntax:

int rdi_set_mod1 (int unit , int cfg_mode);

Input: unit - number of the rdi device.

Output: **-1** on failure,
0 on success

2.3.29 rdi_set_mod2 ()

Description - Set Redirector to mod2: ISL tagging for all ports

Syntax:

int rdi_set_mod2 (int unit , int cfg_mode);

Input: unit - number of the rdi device.

Output: **-1** on failure,

0 on success

2.3.30 rdi_set_mod0 ()

Description - Set Redirector to mod0 (initial): clear ISL tagging

Syntax:

int rdi_set_mod0 (int unit , int cfg_mode);

Input: unit - number of the rdi device.

Output: **-1** on failure,
0 on success

3 APPENDIX

3.1 Appendix A – User level functions- description and examples

The user-level function interface is available with the use of the librdi library module provided with the product software CD. Description of its functionality and its installation is described in Function interface section 2.

We have also provided a sample application – rdifctl - with the product software CD under /rdi_ctl/util folder.

Before installing the application package, install the fm10k driver

Install and load attached fm10k network driver:

1. tar xvzf fm10k-xxx.tar.gz
2. cd fm10k-xxx
3. make install
4. modprobe fm10k (to load with virtual port : modprobe fm10k max_vfs=x where x is the number of the VF ports)
5. bring up network interfaces: ifconfig <interface name> up
6. set ip address

Installing the sample application package – rdifctl:

To install the rdifctl, do the following:

1. tar xvzf rdif-xxx.tar.gz
2. cd rdif-xxx
3. Copy the relevant CFG file (fm_platform_attributes.cfg), from the driver package under the RRC_BPD_xxx\Linux\Redirect\RD_RRC_Control\CFG_files\<product PN> directory to the \etc\rdi\ directory
4. ./clean
5. ./install

Load and configure the rdif

1. For BP adapters: start BP driver (bprdctl_start) before loading the rdif.
2. ./rdif start
3. Open another terminal window
4. Use the rdifctl to define the switch rules (see Appendix B for RDIFCTL Sample Configuration Modes)

To use it, please type:
/bin/rdifctl help

Below are samples of how to access this lib with the user-level application.

```
/*
 * User level function using sample console application
*/
int main(int argc, char **argp, char **envp){

struct rdi_mem rdi_mem;
                .
/*
* Command line parsing
*/
memset(rdi_mem, 0, sizeof(struct rdi_mem) );


/* Configuration command:    */


/* Drop matching packets command – drop all packets coming on port0 with 192.168.0.1
source IP address,  Rule Id is 200*/

rdi_mem.port=0;
rdi_mem.src_port=bswap_32(inet_addr("192.168.0.1"));
rdi_mem.rule_id=200;
rdi_add_rule_drop(0, &rdi_mem);

/* Redirect matching packets command – redirect all packets coming on port0 with
192.168.0.1 source IP address to port1,  Rule Id is 300*/

rdi_mem.port=0;
rdi_mem.redir_port=1;
```

rdi_mem.rule_id=300;
rdi_mem.src_port=bswap_32(inet_addr("192.168.0.1"));
rdi_add_rule_dir(0, &rdi_mem);

/* **Mirror matching packets command** – copy all packets coming on port0 with 192.168.0.1 source IP address to port1*/

rdi_mem.port=0;
rdi_mem.mirror_port=1;
rdi_mem.src_port=bswap_32(inet_addr("192.168.0.1"));
rdi_add_rule_mir(0, &rdi_mem);

3.2 Appendix B – RDIFCTL Sample Program Description

RDIFCTL program is a simple Redirector Control utility.

This application is included with the product software CD under the Util folder. This sample program can be used to help write a customer-specific application program that will fit their own needs and control the Redirector functionality.

Follow Appendix A for installing the FM10K and the application package.

Usage: rdifctl <dev_num> <command> [parameters]
Commands List:

- **set_cfg - set the device to predefined configuration**
  Example:
  rdifctl set_cfg 5
  <5> for MON2 (default mode) - egress disabled
- **get_cfg - get current configuration mode**
- **get_dev_num - get total number of rdi devices**
- **get_port_link <port> - get link status**
- **temp_write <addr> < 1 > <reg>**
- **temp_read <addr> < 1 >**
  See Appendix D – Température registres table for register and temp sensor info
  Example:
  In order to read the temperature, form the temp sensor, need first to issue temp_write command to the temp sensor register and then temp_read command. The output is in Celsius degrees
  rdifctl temp_write 02e 1 0x25
  rdifctl temp_read 02e 1
- **temp1_write <addr> <length (1)> <reg> - use only for 0x4c address**

- **dir - add the rule of a port with direction matching packets to another port**
  Example:
  rdifctl dir port 2 redir_port 1
  all ingress traffic to port 2 will be redirected to port 1
- **drop - drop matching packets**
  Example:
  rdifctl drop port 1 src_ip 196.0.0.126
  All port 1 ingress packet with source ip 196.0.0.126 will be dropped
- **permit - permit matching packets**
- **mir - copy matching frame to mirror_port (mirror must be created previously, see mir_create)**
- **set_prio - set switch priority for the packet**
  Example:
  rdifctl set_prio prio 15 port 1 src_ip 10.10.10.10
  set priority 15 for ingress packets on port 1 with src_ip 10.10.10.10
  Priority level 0-15. Priority 15 is the highest level.
- **set_vlan vlan_act <vlan_act> - set vlan**
  Example:
  rdifctl set_vlan vlan_act 101 port 3 src_ip 10.10.10.10
  Set Vlan 101 to ingress packets on port 1 with src_ip 10.10.10.10
- **add_vlan_promisc <port> - add the port to all 2...4095 VLANs**
- **rem_vlan_promisc <port> - remove the port from all 2...4095 VLANs**
- **stat port <port> - get statistic for specific port (port is mandatory)**
- **prio_stat port <port> - get priority statistic for specific port (port is mandatory)**
- **reset_stat port <port> - reset statistic for specific port (port is mandatory)**
- **rule_stat <rule_id> <group>- get statistic (pkts counter) for specific rule (rule_id is mandatory)**
- **query_list <group> - query rule_id list**
- **clear - clear rule stack**
- **clear_group <group> - clear rule stack for specific group**
- **set_port_mask <ingress_port> <egress_port_list example: 1,5,7> Example:**
  rdifctl set_port_mask 5 1,2,3,4
  Enable traffic flow from ingress port 5 to egress ports 1,2,3,4
- **get_port_mask <ingress_port> - get egress port list**
- **set_reg <addr> <val> - write to RRC register**
- **get_reg <addr> - read from RRC register**
- **sfp_write <port num> <offset> <page> <len> <data1,data2,data3...>**
- **sfp_read <port num> <offset> <page> <len>**
- **set_gpio_dir <gpio> <dir> <value> - set GPIO direction & value**

- **gpio: gpio num; dir: 0 - input; 1 - output; 2 - open drain**
- **prbs <prbs> <dir> <port> - prbs test prbs supp. 7,15,23,31,11,9**
- **get_port_state <port>**
  Example:
  rdifctl get_port_state 1
  mode 0, state 0, info0 7f info1 5f info2 5f info3 5f
  See for Appendix E – port_state registres info

- **loopback <txrx/rxtx/off (1/2/0)> <port>**
  Example:
  rdifctl dev 0 loopback 2 1
  Perform loopback between rx and tx of port 1 on card dev_0

- **remove <rule_id> <group> remove rule**
- **query <rule_id> <group> query rule**
- **lbg - add the rule of a port with send matching packets to load balance group (LBG)**
  Example:
  rdifctl lb port 1 lbg_num 0 src_ip 196.0.0.126 rule_id 100
  All ingress packet from port 1 with source ip 196.0.0.126 will be forwarded to load balance group 0
- **lbg_query_list - query LBG list**
- **lbg_create <port list, example: 1,2> - create LBG**
  Example:
  rdifctl dev 1 lb_creat 52002,52003
  LBG group 0 will be created between virtual ports 52002 and 52003
  The lbg group number is added sequentially
- **lbg_del <lbg> - delete LBG**
- **mir_query_list - query mirror list**
- **mir_query_port_list <mirror_port> - query mirror ports list**
- **mir_create <mirror_port example: 1> <mirror_ports_list example: 2,3> - create mirror**
  Example:
  rdifctl mir_create 1 2,3
  rdifctl mir mir_port 1 dst_ip 10.10.10.184
  All traffic ingress traffic to port 2, 3 with dst_ip 10.10.10.184 will be mirrored to port 1

- **mir_add_port <mirror_port> <port> - add port to mirror**
- **mir_del_port <mirror_port> <port> - delete port from mirror**
- **mir_add_vlan <mirror_port> <vlan_id> - specifies the mirrored frame encapsulation vlan id. Valid range 1...4095; 0 for no vlan encapsulation**
  Example:
  rdifctl mir_create 1 2,3
  rdifctl mir_add_vlan 1 11
  All mirrored traffic egressed from port 1 will be encapsulate with vlan id 11
- **mir_del <mirror_port> - delete mirror (mir rule should be removed before deleting the mirrir port)**
- **l3_hash <hash params> - set l3 hash**
- **l2_hash <hash params> - set l2 hash**
- **get_l3_hash - get l3 hash**
- **get_l2_hash - get l2 hash**
- **info - print Program Information.**
- **help - print this message.**

[parameters] :

for 'permit', 'dir', 'set_prio' and 'drop' commands:
rule_id <rule_id>
src_ip <src_ip>
dst_ip <dst_ip>
dst_port <dst_port>
src_port <src_port>
src_ip_mask <src_ip_mask>
dst_ip_mask <dst_ip_mask>
src_ip6 <src_ip6>
dst_ip6 <dst_ip6>
src_ip6_mask <src_ip6_mask>
dst_ip6_mask <dst_ip6_mask>
group number <group>
ip_proto <ip_proto>
src_port_mask <src_port_mask>
dst_port_mask <dst_port_mask>
vlan <vlan>
vlan_tag <vlan_tag> 1, 2, 3 ,4 for none, standard, user A, user B
vlan_mask <vlan_mask>
prio <0...15>
mpls_type <multi | uni; mandatory for MPLS>
mpls_header <MPLS headers, one or two headers (up to 8 byte)>
mpls_header_mask <MPLS headers mask>

ether_type <ether_type>
src_mac <Source MAC address>

dst_mac <Destination MAC address>
port <1...5>
group - ACL number <0...15>
redir_port <1...5> (mandatory for dir command)
mir_port (mandatory for mir command)
lbg_num Load Balance Group (LBG) number (for lb command)

for l3_hash:

src_ip_hash, mask of src ip
dst_ip_hash, mask of dst ip
src_port_hash, mask of src port
dst_port_hash, mask of dst port
dscp_hash, 0x0-0xff
isl_usr_hash, 0x0-0xff
proto_hash, protocol mask

for l2_hash:

profile_idx, 0...16 default 0
src_mac_hash, in MAC format
dst_mac_hash, in MAC format
ether_type_hash, 0x0-0xfff
vlan_id_hash, 0x0-0xfff
vlan_pri_hash, 0x0-0xf
vlan2_id_hash, 0x0-0xfff
vlan2_pri_hash, 0x0-0xf
sym_mac_hash, on|off

Entire numerical parameters are in decimal format (123) or hex format (0xabc),

MAC is in aa:bb:cc:dd:ee:ff format.
Example:

rdifctl drop port 1 src_ip 196.0.0.126
rdifctl mir mir_port 5 dst_ip 10.10.10.184
rdifctl mir mir_port 5 dst_ip 10.10.10.184
rdifctl set_port_mask 5 1,2,3,4
rdifctl temp_write 0x4c 1 1
rdifctl temp_read 0x4c 1
rdifctl set_vlan vlan_act 9 port 1

rdifctl mir_add_vlan 1 11

Two MPLS example:

rdifctl drop port 1 mpls_type uni mpls_header 0x0000104000001140 mpls_header_mask 0xffffffffffffffff

Single MPLS lable example:

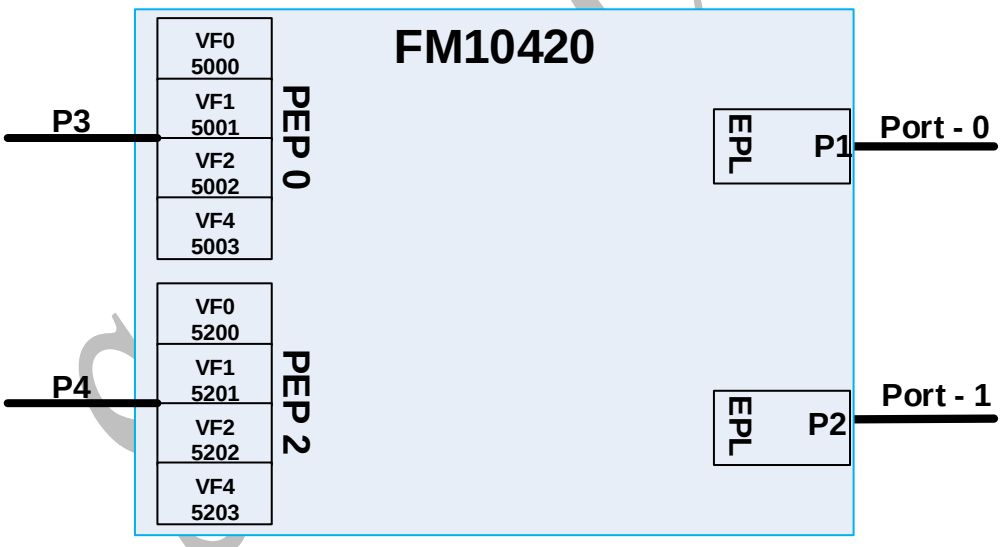
rdifctl drop port 1 mpls_type uni mpls_header 0x1140 mpls_header_mask 0xffffffff

3.3 Appendix E – Working with VF ports

Virtual ports can be created by the FM10K driver on the PEP ports (it is possible to create up to 64 virtual ports).
Example:
modprobe fm10k max_vfs=a,b,c,d where a , b, c, d are the number of the VF ports which will be created on the existing PEP ports.

Example 1:
In case that there is one card installed with 2 PEP (PEP 0 and PEP 2) ports and we would like to define 2 VF ports per PEP:
modprobe fm10k max_vfs=2,2

The first PEP (PEP 0) port will have VF 50000, 50001 and the second PEP (PEP 2) port will have VF 52000, 52001.

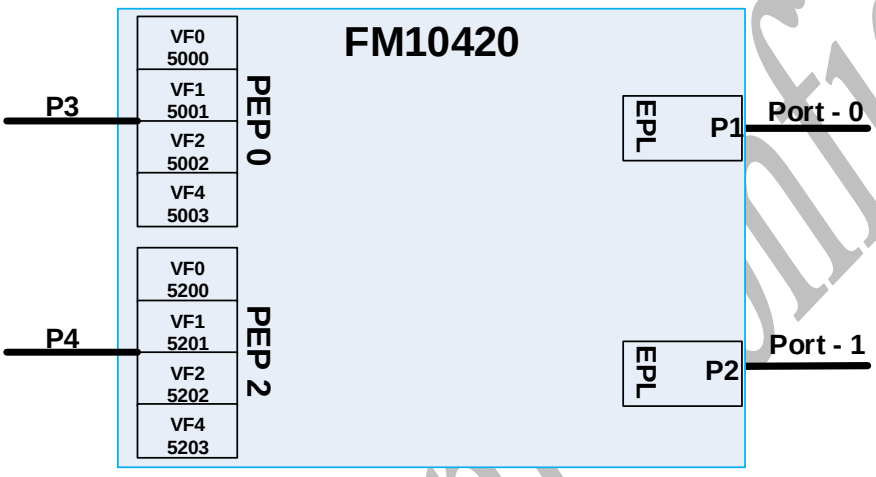


Example 2:

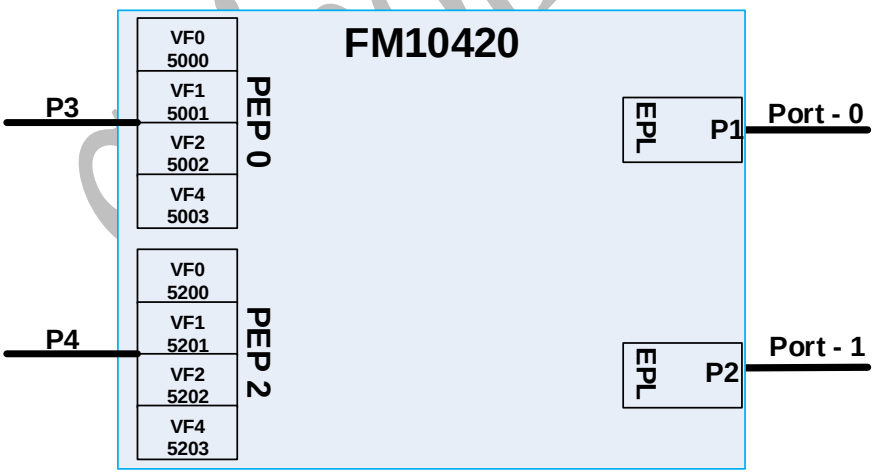
In case that there are 2 cards installed with 2 PEP ports per card and we would like to define 2 VF ports per each PEP:
modprobe fm10k max_vfs=2,2,2,2

On the first card:
The first PEP (PEP 0) port will have VF 50000, 50001 and the second PEP (PEP 2) port will have VF 52000, 52001 port number and VF.
On the second card:
The first PEP (PEP 0) port will have VF 50000, 50001 and the second PEP (PEP 2) port will have VF 52000, 52001 port number and VF.

Card 1:



Card 2:



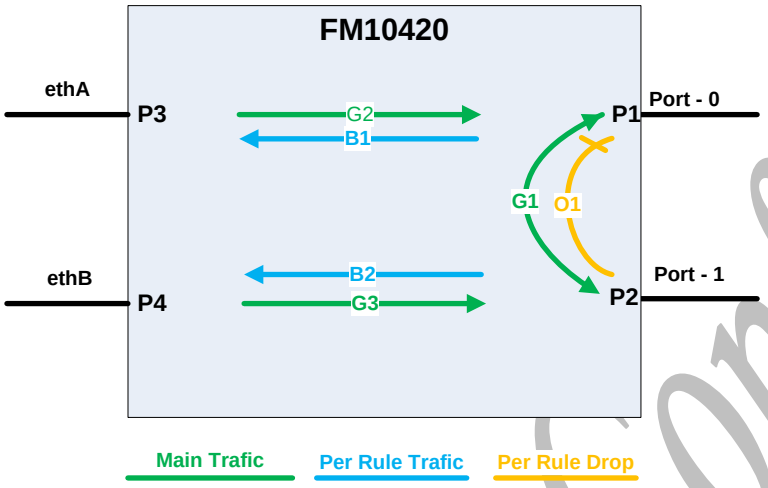
3.4 Appendix D – RDIFCTL Sample Configuration Modes

In this section, we provide examples of several common operation modes that may give users a better understanding of how to work and configure the redirector switch. Samples are provided for the 3 configuration steps (as per section 1.5) that are used to achieve each mode implemented.

Most of these examples will be for an Inline system configuration in which one side is the WAN interface, the other side is the LAN interface, and traffic between them must go to the appliance for monitoring and inspection.

3.4.1 Example 1 – Applicable for Dual 100G and Dual 40G cards.

In this configuration mode, the traffic will start with Bypass between the two external ports (P1< -- >P2). Then, per defined rules, specific traffic will go into the interfaces (P1->P3, P2->P4), and each interface will have the option to return the traffic back to the external ports (P3->P1, P4->P2). Rules for dropping packets between the bypass ports (P1< -- >P2) can also be defined.



Steps in configuration:

1. Allowed traffic flow:
   rdifctl set_port_mask 3 1 (P3->P1)
   rdifctl set_port_mask 4 2 (P4->P2)
   rdifctl set_port_mask 1 3,2 (P1->P3,2)
   rdifctl set_port_mask 2 4,1 (P2->P4,1)

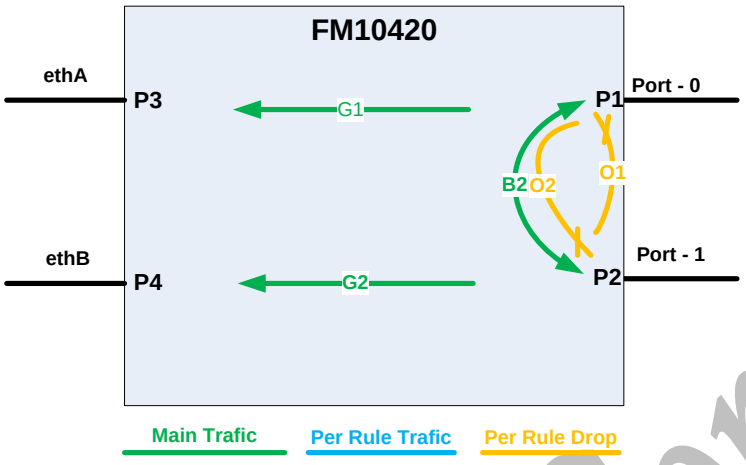
2. Defining main traffic path (green lines):
   rdifctl dir port 1 redir_port 2 rule_id 1000 (G1 in the BD)
   rdifctl dir port 2 redir_port 1 rule_id 1001 (G1 in the BD)
   rdifctl dir port 3 redir_port 1 rule_id 1002 (G2 in the BD)
   rdifctl dir port 4 redir_port 2 rule_id 1003 (G3 in the BD)

   *note that the rule ID of these rules needs to be higher than the rules on section 3

3. Defining rules for redirect and drop
   rdifctl dir port 1 redir_port 3 src_IP 168.16.1.0 rule_id 1 (packets with src IP 168.16.1.0 will be forward from P1 to P3) (B1 in the BD)
   rdifctl dir port 2 redir_port 4 dst_IP 168.16.1.0 rule_id 2 (packets with dst IP 168.16.1.0 will be forward from P2 to P4) (B2 in the BD)
   rdifctl drop port 1 ip_proto 17 rule_id 3 (packets with UDP protocol coming into port P1 will be dropped (O1 in the BD)
   More rules can be added here per the needs.

3.4.2 Example 2 (TAP based mode)

In this configuration mode, the traffic will start with Bypass between the two external ports (P1< -- >P2) and traffic going into the interfaces (P1->P3, P2->P4). Rules to drop packets between the bypass ports (P1< -- >P2) .



Steps in configuration:

1. Allowed traffic flow:
   rdifctl set_port_mask 3 0 (P3-> no one)
   rdifctl set_port_mask 4 0 (P4-> no one)
   rdifctl set_port_mask 1 3,2 (P1->P3,2)
   rdifctl set_port_mask 2 4,1 (P2->P4,1)
   rdifctl dir port 1 redir_port 2 rule_id 100 (G1 in the BD)
   rdifctl dir port 2 redir_port 1 rule_id 101 (G1 in the BD)

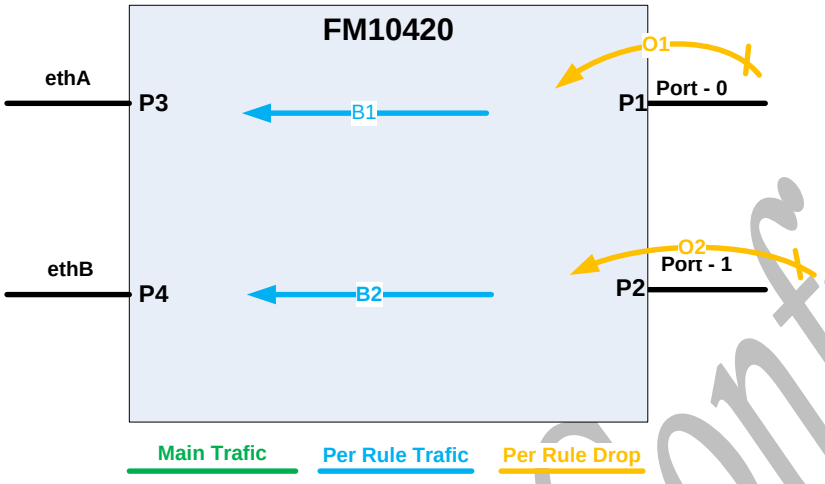
2. Defining main traffic path (green lines):
   rdifctl mir_create 3 1(G1 in the BD)
   rdifctl mir_create 2 4 (G2 in the BD)
   rdifctl dir port 1 redir_port 2 rule_id 102 (B2 in the BD)
   rdifctl dir port 2 redir_port 1 rule_id 102 (B2 in the BD)

   *note that the rule ID of these rules needs to be higher than the rules on section 3

3. Defining rules for redirect and drop
   rdifctl drop port 1 ip_proto 17 rule_id 1 (packets with UDP protocol coming into port P2 will be dropped (O1 in the BD)
   rdifctl drop port 2 ip_proto 17 rule_id 2 (packets with UDP protocol coming into port P3 will be dropped (O2 in the BD)
   More rules can be added here per the needs.

3.4.3 Example 3 (Monitor based mode)

In this configuration mode, no traffic is going into the interfaces (P49->P1, P50->P2). Per defined rules, specific traffic will be added to pass into the interfaces (P1->P49, P2->P50).



Steps in configuration:

1. Allowed traffic flow:
   rdifctl set_port_mask 3 0 (P3-> no one)
   rdifctl set_port_mask 4 0 (P4-> no one)
   rdifctl set_port_mask 1 3 (P1->P3)
   rdifctl set_port_mask 2 4 (P2->P4)

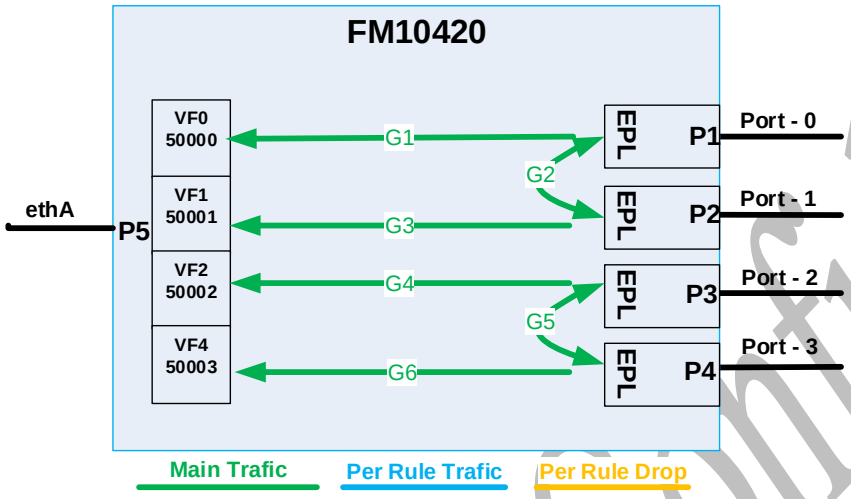
2. Defining main traffic path (green lines):
   N/A

3. Defining rules for redirect and drop
   rdifctl drop port 1 ip proto 17 rule_id 1 (packets with UDP protocol coming into P1 will be dropped) (O1 in the BD)
   rdifctl drop port 2 ip proto 17 rule_id 2 (packets with UDP protocol coming into P2 will be dropped) (O2 in the BD)
   rdifctl dir port 1 redir_port 3 src_IP 168.16.1.0 rule_id 3 (packets with src IP 168.16.1.0 will be forward from P1 to P3) (B1 in the BD)
   rdifctl dir port 2 redir_port 4 dst_IP 168.16.1.0 rule_id 4 (packets with dst IP 168.16.1.0 will be forward from P2 to P4) (B2 in the BD)

   More rules can be added here per the needs.

3.4.4 Example 4 - VF enabled configuration - TAP

In this configuration mode, the traffic will start with TAP mode, traffic from P1 will go to both P2 and P50000 (P1-->P2&50000 & P2-->P1&P50001, P3-->P4&50002 & P4-->P3&P50003). Rules for dropping packets between the bypass ports (P1<-->P2, P3<-->P4) can also be defined and for dropping packets going to the VFs.



Steps in configuration:

1. Allowed traffic flow:
   rdifctl set_port_mask 1 2,5 (P1->P2,P5(P5xxx) Enable G2 + G1 to all P5xxx)
   rdifctl set_port_mask 2 1,5 (P2->P1,P5(P5xxx) Enable G2 + G3 to all P5xxx)
   rdifctl set_port_mask 3 4,5 (P3->P4,P5(P5xxx) Enable G4 + G5 to all P5xxx))
   rdifctl set_port_mask 4 3,5 (P4->P1,P5(P5xxx) Enable G6 + G5 to all P5xxx))

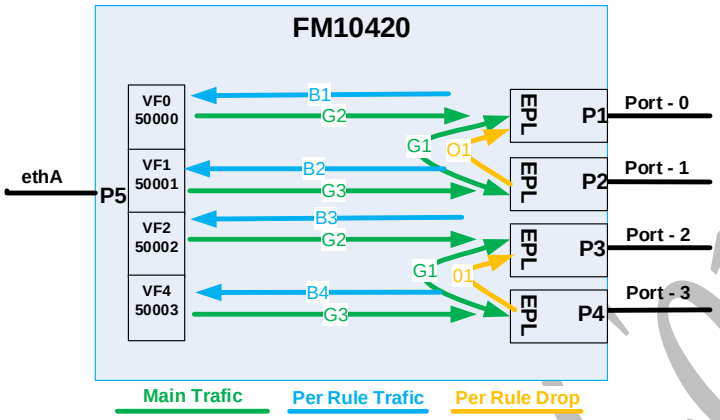
2. Defining main traffic path (green lines):
   rdifctl mir_create 2 1 (copy P1 traffic to P2 – G2 on the BD)
   rdifctl mir_create 1 2 (copy P2 traffic to P1 – G2 on the BD)
   rdifctl dir port 1 redir_port 50000 rule_id 10000 (FW P1 traffic to VF0 – G1 on the BD)
   rdifctl dir port 2 redir_port 50001 rule_id 10001 (FW P2 traffic to VF1- G3 on the BD

   rdifctl mir_create 4 3 (copy P3 traffic to P4 - G5 on the BD)
   rdifctl mir_create 3 4 (copy P4 traffic to P3- G5 on the BD)
   rdifctl dir port 3 redir_port 50002 rule_id 10004 (FW P3 traffic to VF2 - G4 on the BD)
   rdifctl dir port 4 redir_port 50003 rule_id 10005 (FW P4 traffic to VF3 – G6 on the BD)

3. Defining rules for redirect and drop
   More rules can be added here per the needs in the ID range 7-10000

3.4.5 Example 5 - VF enabled configuration 2

In this configuration mode, the traffic will start with Bypass between the two external ports (P1<-->P2 & P3<-->P4). Then, per defined rules, specific traffic will go into the interfaces (P1->P50000, P2->P50001, P3->P50002, P4->P50003), and each interface will have the option to return the traffic back to the external ports (P50000->P1, P50001->P2, P50002->P3, P50003->P4). Rules for dropping packets between the bypass ports (P1<-->P2, P3<-->P4) can also be defined.



Steps in configuration:

1. Allowed traffic flow:
   rdifctl set_port_mask 1 2,5 (P1->P2,P5(P5xxx) Enable G1 + B1 to all P5xxx)
   rdifctl set_port_mask 2 1,5 (P2->P1,P5(P5xxx) Enable G1 + B2 to all P5xxx)
   rdifctl set_port_mask 3 4,5 (P3->P4,P5(P5xxx) Enable G1 + B1 to all P5xxx))
   rdifctl set_port_mask 4 3,5 (P4->P1,P5(P5xxx) Enable G1 + B2 to all P5xxx))
   rdifctl set_port_mask 5 1,2,3,4 (P5->P1,2,3,4 Enable G2&3 from all P5xxx)

2. Defining main traffic path (green lines):
   rdifctl dir port 1 redir_port 2 rule_id 10000 (bypass-0 G1 in the BD)
   rdifctl dir port 2 redir_port 1 rule_id 10001 (bypass-0 G1 in the BD)
   rdifctl dir port 3 redir_port 4 rule_id 10002 (bypass-1 G1 in the BD)
   rdifctl dir port 4 redir_port 3 rule_id 10003 (bypass-1 G1 in the BD)
   rdifctl dir port 50000 redir_port 1 rule_id 10004 (bypass-0 G2 in the BD)
   rdifctl dir port 50001 redir_port 2 rule_id 10005 (bypass-0 G3 in the BD)
   rdifctl dir port 50002 redir_port 3 rule_id 10006 (bypass-1 G2 in the BD)
   rdifctl dir port 50003 redir_port 4 rule_id 10007 (bypass-1 G3 in the BD)

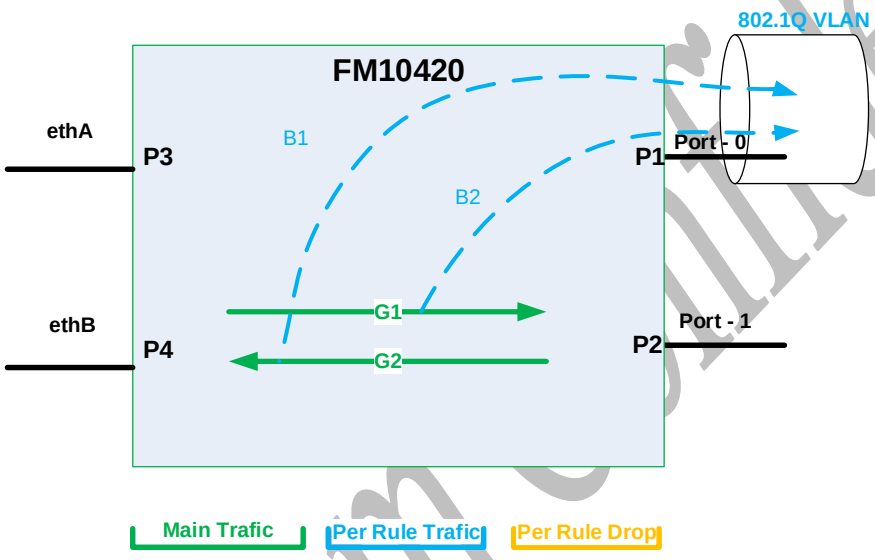
3. Defining rules for redirect and drop
   rdifctl dir port 1 redir_port 50000 src_IP 168.16.1.1 rule_id 1 (packets with src IP 168.16.1.1 will be forward from P2 to P0) (bypass-0 B1 in the BD)
   rdifctl dir port 2 redir_port 50001 dst_IP 168.16.1.1 rule_id 2 (packets with dst IP 168.16.1.1 will be forward from P3 to P1) (bypass-0 B2 in the BD)
   rdifctl drop port 2 ip_proto 17 rule_id 3 (packets with UDP protocol coming into port P2 will be dropped (bypass-0 O1 in the BD)

   rdifctl dir port 3 redir_port 50002 src_ip 168.16.1.1 rule_id 4 (packets with src IP 168.16.1.1 will be forward from P2 to P0) (bypass-1 B1 in the BD)
   rdifctl dir port 4 redir_port 50003 dst_ip 168.16.1.1 rule_id 4 (packets with dst IP 168.16.1.1 will be forward from P3 to P1) (bypass-1 B2 in the BD)
   rdifctl drop port 4 ip_proto 17 rule_id 6 (packets with UDP protocol coming into port P2 will be dropped (bypass-0 O1 in the BD)

   More rules can be added here per the needs in the ID range 7-10000

3.4.6 Example 6 - Mirror traffic and add Vlan -

In this configuration mode, all traffic from Port 2 will be redirected to port 4 and traffic from port 4 will be redirected to Port 2.
All traffic from Port 2 and Port 4 will be encapsulated with Vlan ID 11 and mirrored to Port 1.

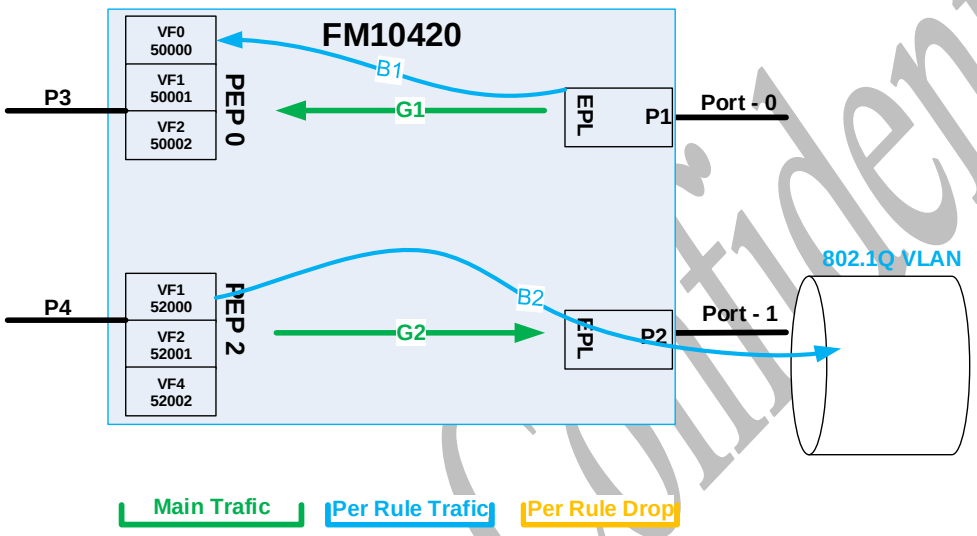


1. Allowed traffic flow:
   rdifctl set_port_mask 4 2 (P4->P2 Enable G1)
   rdifctl set_port_mask 2 4 (P2->P4 Enable G2)

2. Defining main traffic path (green lines):
   rdifctl dir port 4 redir_port 2 rule_id 10000 (G1 in the BD)
   rdifctl dir port 2 redir_port 4 rule_id 10001 (G2 in the BD)
   rdifctl mir_create 1 2,4 (create mirror group port 2,4 are mirrored to port 1)
   rdifctl mir_add_vlan 1 11 (vlan 11 added to all traffic which is mirror to port 1)

3.4.7 Example 7 - Add Vlan with PF/VF ports

In this configuration mode, traffic from Port 1 will be redirected to VF port 50000 and traffic from VF port 52000 will be encapsulated with Vlan 300 and will be redirected to Port 2.



1. Allowed traffic flow:
   rdifctl set_port_mask 1 3 (P1->P3 Enable G1)
   rdifctl set_port_mask 4 2 (P4->P2 Enable G2)

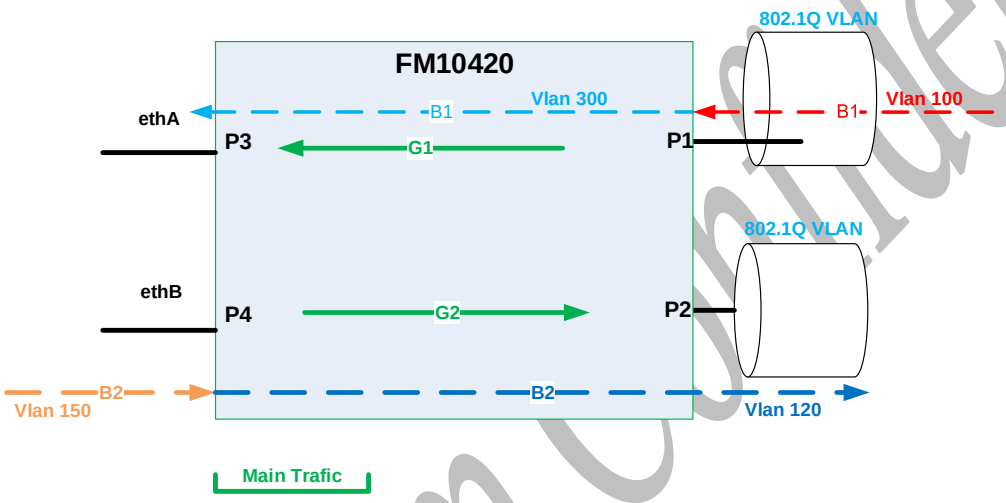
2. Defining main traffic path (green lines):
   rdifctl set_vlan vlan_act 300 port 52000 rule_id 1000

   (set_vlan vlan_act" rule for specific logical port MUST be before "dir/redir_port" rule. Otherwise, dir/redir will override VLAN settings.

   rdifctl dir port 1 redir_port 50000 rule_id 10001 (B1 in the BD)
   rdifctl dir port 52000 redir_port 2 rule_id 10002 (B2 in the BD)

3.4.8 Example 8 – Vlan translation

In this configuration mode, traffic from Port 1 with Vlan 100 will be translated to Vlan 300 and will be redirected to port 3.
Traffic from port 4 with Vlan 150 will be translated to Vlan 120 and will be redirected to Port 2.



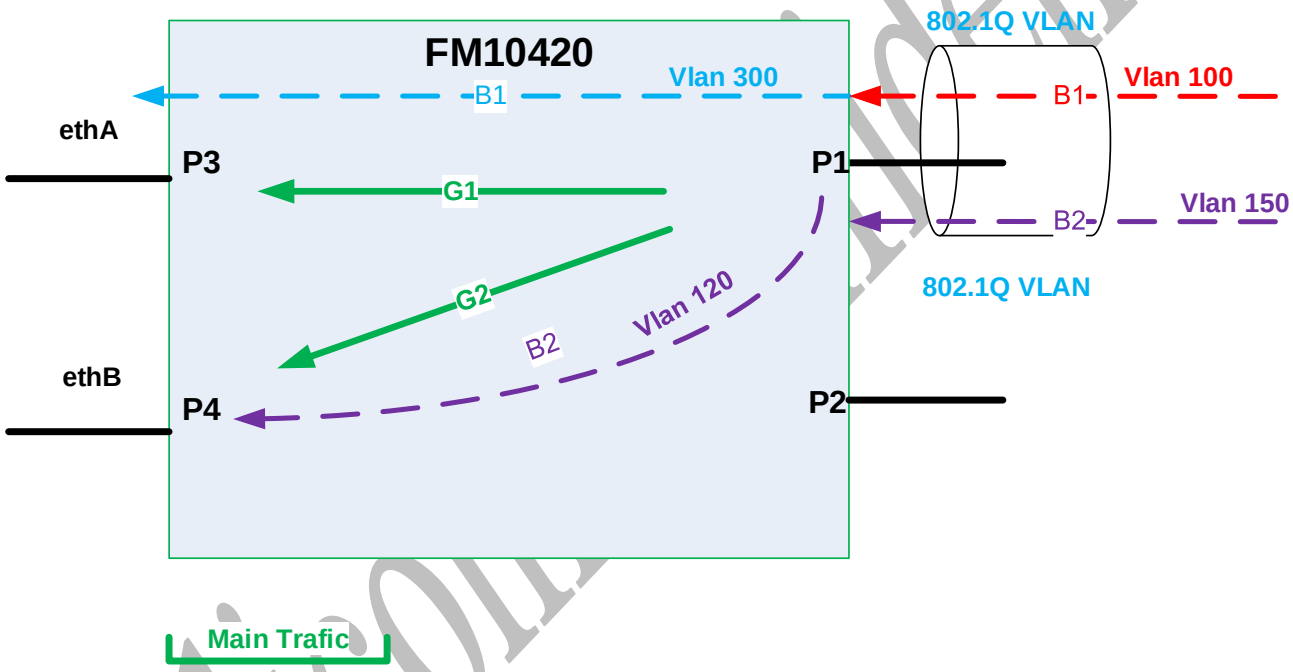
1. Allowed traffic flow:
   rdifctl set_port_mask 1 3 (P1->P3 Enable G1)
   rdifctl set_port_mask 4 2 (P4->P2 Enable G2)

2. Defining main traffic path:

   rdifctl set_vlan vlan_act 300 port 1 vlan 100 rule_id 1000 (Vlan translation B1 in the BD)
   rdifctl set_vlan vlan_act 120 port 4 vlan 150 rule_id 1001 (Vlan translation B2 in the BD)
   rdifctl dir port 1 redir_port 3 rule_id 10002 (G1 in the BD)
   rdifctl dir port 4 redir_port 2 rule_id 10003 (G1 in the BD)

3.4.9 Example 9 –redirect traffic to 2 different ports (using Group)

In this configuration mode, traffic from Port 1 with Vlan 100 will be translated to Vlan 300 and will be redirected to port 3.
Traffic from port 1 with Vlan 150 will be translated to Vlan 120 and will be redirected to Port 4.
"Group" parameter should be used when performing 2 different actions with same packet, different "group" number for each type of action.



1. Allowed traffic flow:
   rdifctl set_port_mask 1 3,4
2. Defining main traffic path:

   rdifctl set_vlan vlan_act 300 port 1 vlan 100 group 1 rule_id 1000 (Vlan translation B1 in the BD)
   rdifctl set_vlan vlan_act 120 port 1 vlan 150 group 1 rule_id 1001 (Vlan translation B2 in the BD)
   rdifctl dir port 1 redir_port 3 vlan 100 group 2 rule_id 1002 (packets with vlan 300 will be forward from P1 to P3) (B1 in the BD)
   rdifctl dir port 1 redir_port 4 vlan 150 group 2 rule_id 1003 (packets with vlan 120 will be forward from P1 to P4) (B2 in the BD)

3.5 Appendix D – Température registres table

Product	Temp Sensor	Address	Thermal Diode and placement
PE310G4DBIR-SR / LR	MAX6639	0x2c	Two diodes: DX1 (Reg x00h): Internal Diode on RRC DX2 (Reg x001): external transistor on card edge
PE310G4DBIR-ER	MAX6639	0x2c	Two diodes: DX1 (Reg x00h): Internal Diode on RRC DX2 (Reg x001): external transistor on adapter center
PE310G4DBIR-T **	ADT7461ARMZ	0x4c	One diode: (Reg 0x01,0x10) Internal Diode on RRC
PE340G2DBIR*	ADT7475	0x2e	Three diodes: D1 (Reg 0x25): Internal Diode on RRC D2 (Reg 0x27): Internal Diode on PLX D3 (Reg 0x26): Internal sensor on ADT7475
PE3100G2DQIR*	ADT7475	0x2e	Three diodes: D1 (Reg 0x25): Internal Diode on RRC D2 (Reg 0x27): Internal Diode on PLX D3 (Reg 0x26): Internal sensor on ADT7475
PE3100G2DQIRL**	ADT7461	0x4c	One diode: (Reg 0x01,0x10) : Internal Diode on RRC
PE325G2DSIR	ADT7461	0x4c	One diode: (Reg 0x01,0x10) Internal Diode on RRC
PE3100G2DQIRM-QX4	LM96163	0x4c	Thermal Internal Diode in RRC device: rdifctl temp1_write **0x4c** 1 0 rdifctl temp_read **0x4c** 1 Thermal External on Card near RRC:

			rdifctl temp1_write **0x4c** 1 1 rdifctl temp_read **0x4c** 1

* Need to reduce 64 from the output in order to get the temp in Celsius
** use command **temp1_write** and **temp1_read**

3.6 Appendix E – port_state registres info

Mode:

0 - Port Mode Up

1 - Port Admin Down - the port is administratively set down with the SERDES transmitting an idle pattern but the receiver disabled.
2 - Port Admin Power Down - the port is administratively set down with the SERDES shut down and no signal transmitted.
3 - Port Mode a Built In Self Test
4 - Port Remote Fault
5 - Port Local Fault

State:

0 - Port State UP (Link UP)
4 - Port State Remote Fault
5 - Port State Down (Link Down)
6 - Partially Up (Some lanes have either signal or partial synchronization)
7 - Local Fault
8 - DFE tuning in progress

Info0-3 refers to the 4 lanes.
RxRdy (bit 0):
1 indicates that the Receiver PLL has locked.
TxRdy (bit 1):
1 indicates that the Transmitter has calibrated and is ready to send data.
RxSigStrengthEn (bit 2):
1 indicates that signal strength will be measured.
RxSigStrength (bit 3-4):
This field indicates the relative health of the received serial signal (0 = no signal, 3 = good signal). The RxSigStrengthEn signal must be asserted to validate the data on these bits. This field should be ignored when loopback is enabled.
Align Status (bit 5):
1 indicates that all lanes are aligned. This bit will be used only on multi-lanes port configuration. This means for 10G port using 4 lanes (XAUI, CX4 and KX4), 40G port using 4 lanes (XLAUI, SR4, CR4 and KR4). This bit will only be set for lane 0.