Silicom

Redirector Programmer Guide

-

Linux OS

REVISION HISTORY

Date	Change description
29-Jul-08	Initial document
22-Dec-08	2.0 – updated with new commands – del, enquire, list,
08-Mar-09	2.1 – updated with multi device support
29-Jun-09	4.0 – updated with statistics support

1 INTRODUCTION

1.1 General

The Silicom Redirect 10 Gigabit adapters are PCI Express network interface cards with Bypass and Redirect capabilities.

Silicom's Bypass-Series adapters are designed with a Bypass circuitry in order to provide maximum up time for the network.

1.2 Purpose

This document describes the Redirect functionality of Silicom bypass/redirect product line and provide functional description of the API available for use to control the Redirect interfaces via user-level application.

.

1.3 Scope

This document includes description of the Silicom Redirect function and its operation and describes list of APIs supported by the bypass products and the method of accessing it and responses returned from it. This document appendix provides list of capabilities and supported features for each product, sample commands, tools to operate the different controlling methods.

.

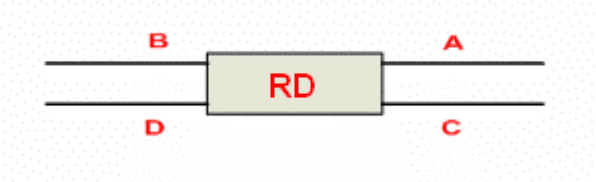
1.4 Target Release

PE10G2DBI-SR-SD

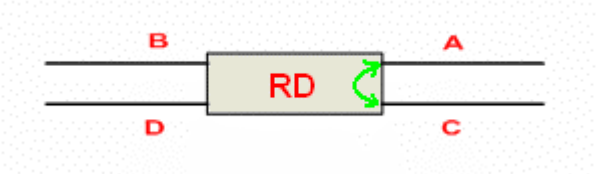
Further new products may also compliant to these APIs as well.

1.5 Definitions

The diagram below is a simple view of the Redirector box, which includes a Broadcom chip with 4 10G ports (A, B, C, D).
Ports A & C are the external ports and ports B & D are the ports connected to the two ports NIC on this product.

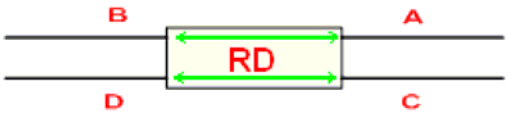


INLINE1 mode:



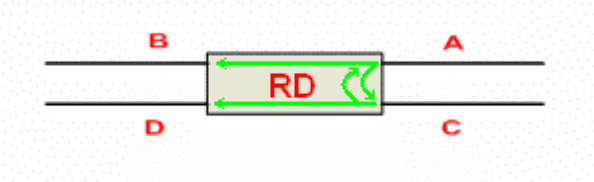
Forward all packets incoming on port A to port C. Forward all packets incoming on port C to port A. Any packet incoming to port B and D will be dropped. No packet incoming on port A and C will go out on port B or D.

INLINE2 mode:



When setting up this configuration state the RD will forward all packets from port A to port B and vice versa and from port C to port D and vise versa. No packet will be sent in any other direction (from A to C or A to D or C to A or C to B).

TAP mode:



Forward all packets from port A to port C and also to port B (in parallel to both). Forward all packets from port C to port A and also to port D (in parallel to both). Any packet incoming on port B and D will be dropped. No packet will be sent from port A to port D. No packet will be sent from port C to port B.

MON1 mode:

Forward all packets from port A to port B. Forward all packets from port C to port D. Any packet incoming from any other direction will be dropped.

MON2 mode:

No packet will be passed from any port to any port.

2 User mode Software API

This section defines the different calls for interface 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 located on the product CD under Lib folder.

```
enum rdd_cfg
{INLINE1, INLINE2, TAP, MON1, MON2};

enum rdi_conf {
   RDI_INIT=1,
   RDI_CLEAR,
   RDI_SET_CFG,
   RDI_SET_DROP,
   RDI_SET_DIR,
   RDI_SET_MIR,
   RDI_GET_CFG,
   RDI_INSTALL,
   RDI_GET_CNT,
   RDI_ENTRY_REMOVE,
   RDI_ENTRY_QUERY,
   RDI_ENTRY_QUERY_LIST,
} ;

typedef struct rdi_id_list{
   unsigned int rule_num;
   int id_list[120];
} rdi_id_list_t;


typedef struct rdi_query_list {
   rdi_id_list_t rdi_id_list[4];
   int ret;
}rdi_query_list_t;
```

Redirector configuration parameters are passed in the rdi_mem structure. The fields of the rdi_mem structure are shown below:

Field	Description	Value/Note
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	0…3, 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	0…3, mandatory for rdi_add_rule_mir() and command, default is 0
rdi_mem:redir_port	port the matched packet forwarded to	0…3, 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_ip	source IP address	
rdi_mem:dst_ip	destination IP address	
rdi_mem:src_ip_mask	source IP address mask	
rdi_mem:dst_ip_mask	destination IP 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 ID	
rdi_mem:vlan_mask	VLAN ID mask	

2.2 PROC interface

PROC interface (under /proc/net/rdi) is used for getting additional info about network interfaces of specific redirector device. Master interface is on the first place, slave – on the second (see Bypass Programmer Guide for additional info).

Example:

ls /proc/net/rdi/

dev0 dev1

cat /proc/net/rdi/dev0

eth7

eth8

2.3 Loading the function library module

In order to compile and install this library module follow the steps below:

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

This will install the library module **librdi.so** in /usr/local/lib, daemon **rdid** in /bin sample utility **rdictl** and **rdi** script.
rdi script performs redirector driver **rdi_mod** loading and starts **rdid** daemon.

In order to use the software and perform configuration commands:

1. Load / unload the redirector driver and start / stop **rdid** daemon: **rdi start / rdi stop.**
2. Configure redirector device: **rdictl** - see Appendix C
3. Start / stop **rdid** daemon: **rdid / rdid stop.**
4. Run rdid daemon in verbose mode: **rdid –v.**

2.4 Basic Commands

2.4.1 rdi_get_dev_num()

Description – Get total number of rdi devices.

Syntax:

int get_dev_num();

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

2.4.2 rdi_init()

Description – Software initialization

Syntax:

int rdi_init(int unit);

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

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

2.4.3 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

2.4.4 rdi_get_cfg()

Description – Get current configuration mode (Definitions)

Syntax:

int rdi_get_cfg (int unit);

Input: unit - number of the rdi device,

Output enum rdi_conf value on success,
-1 on failure

2.4.5 rdi_add_rule_drop()

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

Syntax:

int rdi_add_rule_drop(int unit, struct if_rdi * p_rdi);

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

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

2.4.6 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, struct if_rdi * p_rdi);

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

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

2.4.7 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, struct if_rdi * p_rdi);

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

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

2.4.8 rdi_install_rule()

Description -Install rule stack to the hardware.

Syntax:

int rdi_install_rule(int unit);

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

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

2.4.9 rdi_clear_rule()

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

Syntax:

int rdi_clear_rule(int unit);

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

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

2.4.10 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

2.4.11 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

2.4.12 rdi_entry_query_list()

Description – Query rule_id list.

int rdi_entry_query (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

2.4.13 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

2.4.14 rdi_get_stat ()

Description – Query per port statistics.

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

```
typedef struct rdi_stat {
   unsigned long long total;
   unsigned long long txnoerror;
   unsigned long long rxnoerror;
   unsigned long long rxdrop;
   unsigned long long txdrop;
}rdi_stat_t;
```

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

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

3 APPENDIX

3.1 Appendix A – Operating as standard NIC

Use **INLINE2** configuration for set the product to work as a standard NIC.

3.2 Appendix B – 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 provided also sample application – rdictl with the product software CD under /rdi_ctl/util folder.

Installing the sample application package – rdictl:

To install the rdictl, do the following:

1. un-pack the rdi_ctl-xxx archive
2. change into rdi-ctl-x.x.x folder
3. run: ./install
4. run: rdi start

To use it please type:
/bin/rdictl help

Below are samples of how to access this lib with 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) );

/*
* Software initialization command */
```

```
*/
if((rdi_init())<0) {
        printf("Initialization failed\n")
        return -1;
}
```

/* Configuration command: */

```
rdi_set_cfg(1, INLINE_1);
```

/* **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(1, &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(1, &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(1, &rdi_mem);
```

/* **Install rule stack** */

```
rdi_install_rule(1);
…….
}
```

3.3 Appendix C – RDICTL sample program description

RDICTL program is a simple Redirector Control utility.
This application is included with the product software CD under Util folder.
This sample program can be used to help writing customer specific application program that will fit to its own need and will control the Redirector functionality.
Below is description of how to use and operate this sample program.

1 Compiling and using the application

In order to compile and run this sample program follow the steps below:

1. un-pack the rdi_ctl-xxx archive file
2. change into rdi_ctl-xxx folder
3. run> ./install
4. run> rdi start

This will compile, place the program in /bin folder so it can be used from any folder and load Redirector's drivers.

2 Usage

rdictl <command> [parameters]
Commands List:
set_cfg - set the device to predefined configuration
dev - device number
get_dev_num - get total number of rdi devices.
get_cfg - get current device configuration.
dir - add the rule of a port with direction matching packets to another port
drop - drop matching packets
stat - get statistic for specific port (port is mandatory)
rule_stat <rule_id> - get statistic (pkts counter) for specific rule (rule_id is mandatory)
vlan_stat - get VLAN statistic for specific port (port is mandatory)
query_list - query rule_id list
remove <rule_id> - remove rule
query <rule_id> - query rule
mir - mirror matching packets
-f - load configuration from /etc/rdi/rdi.conf file
-fp - load configuration from specific file location. For example:
rdictl -fp /usr/temp/curr.cfg
info - print Program Information.
help - print this message.
[parameters] :
for 'dir' and 'drop' commands:

src_ip <src_ip>
dst_ip <dst_ip>
src_port <src_port>
dst_port <dst_port>
src_ip_mask <src_ip_mask>
dst_ip_mask <dst_ip_mask>
ip_proto <ip_proto>
src_port_mask <src_port_mask>
dst_port_mask <dst_port_mask>
vlan <vlan>
vlan_mask <vlan_mask>
port <0...3>(mandatory for dir, drop, mir commands)
redir_port <0...3> (mandatory for dir command)
mir_port <0...3> (mandatory for mir command)
for 'set_cfg' commands:
<1...5> for INLINE1, INLINE2, TAP, MON1, MON2

Configuration file format:

Configuration file has the same as rdictl command line format. For example:

mode
set_cfg dev 0 1
rules
dir dev 1 port 0 redir_port 1
drop dev 0 port 1 src_ip 192.168.0.1

Command line examples:

Set the product to INLINE2 state and drop all TCP packets with 192.168.0.1:
 # rdictl dev 1 set_cfg 2 drop ip_proto 6 src_ip 192.168.0.1

Set the product to TAP mode:
 # rdictl dev 1 set_cfg 3