Application Note

CPE

VLAN Control Utility (vlanctl)

VLAN-AN100-R

5300 California Avenue • Irvine, CA 92617 • Phone: 949-926-5000 • Fax: 949-926-5203

June 30, 2015

Revision History

Revision	*Date*	*Change Description*
VLAN-AN100-R	06/30/15	Initial release Update from version released in February 04, 2011

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

Table of Contents

List of Tables

About This Document

Purpose and Audience

This document describes the architecture and commands for the Linux VLAN interface, the Broadcom VLAN Control Utility (vlanctl). The document is meant for software engineers using the 4.16L.XX CPE software.

Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use.

For a comprehensive list of acronyms and other terms used in Broadcom documents, go to: http://www.broadcom.com/press/glossary.php.

Document Conventions

The following conventions may be used in this document:

Convention	***Description***
Bold	User input and actions: for example, type **exit**, click **OK**, press **Alt+C**
`Monospace`	Code: `#include <iostream>` HTML: `<td rowspan = 3>` Command line commands and parameters: `wl [-l] <command>`
< >	Placeholders for *required* elements: enter your <username> or `wl <command>`
[ ]	Indicates *optional* command-line parameters: `wl [-l]` Indicates bit and byte ranges (inclusive): [0:3] or [7:0]

Technical Support

Broadcom provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates through its customer support portal (https://support.broadcom.com). For a CSP account, contact your Sales or Engineering support representative.

In addition, Broadcom provides other product support through its Downloads and Support site (http://www.broadcom.com/support/).

Overview

The ITU-T G.984.4 Standard defines the VLAN tagging operations required for the deployment of the GPON Traffic Models. The standard Linux kernel supports a limited set of VLAN tagging operations via the *vconfig* program, however, due to the complexity of the VLAN operations defined in the GPON Traffic Models, major changes would have to be made to *vconfig*. Due to the need for kernel upgrades and compatibility with existing modules, a new Linux VLAN Interface architecture was devised. This document describes the architecture of the new Linux VLAN Interface, herein called the Broadcom VLAN Control Interface, or simply vlanctl interface.

A VLAN Control Interface is implemented in Linux as a Virtual Device and is always attached to a Real Device. It may be used for routing or bridging Ethernet frames, much in the same way as a VLAN Interface generated by *vconfig*. Multiple VLAN Control Interfaces may be created for a given Real Device Interface on the Receive and Transmit directions.

It is important to note that a VLAN Control Interface does not need to have VLAN rules associated with it. The Interface can be used for traffic manipulations or routing on packets that do not have a VLAN tag.

Management of the VLAN Control Interfaces is handled by the *vlanctl* program.

Requirements

The following is a list of requirements for VLAN tagging operations gathered from the ITU-T G.984.4 standard, the DSL Forum WT-156 document, and from GPON customer requirements.

- The following frame types must be supported:
  - Untagged frames
  - Priority-tagged frames
  - Single-tagged frames
  - Double-tagged frames (Q-in-Q)
- Any combination of the following values can be used to identify the VLAN operation to be performed:
  - Outer VLAN Tag Protocol Identifier (TPID)
  - Outer VLAN Priority Bits (Pbits)
  - Outer VLAN Identifier (VID)
  - Inner VLAN Tag Protocol Identifier (TPID)
  - Inner VLAN Priority Bits (Pbits)
  - Inner VLAN Identifier (VID)
  - Ether Type
  - DSCP
  - IP Protocol
- The following values/tables must be supported for each VLAN interface:
  - A default TPID value.
  - One 64-entry DSCP-to-Pbits translation table.

- The following VLAN tagging operations must be supported for **Untagged** frames.
  - Insert a programmable VLAN tag.
  - Insert a programmable inner VLAN tag, and a programmable outer VLAN tag.
  - Insert a programmable inner VID, and a programmable outer VID, and obtain Pbits via the DSCP-to-Pbits translation table.
  - Insert a programmable VID and map the IPv4 DSCP value to a VLAN Priority (Pbits) using the port's DSCP-to-Pbits translation table.
- The following VLAN tagging operations must be supported for **Priority-tagged** frames:
  - Remove tag.
  - Replace VID with a programmable inner VID, and preserve the original Pbits.
  - Add a programmable outer VLAN tag.
  - Add a programmable outer VLAN tag, and replace the VID of the priority tag with a programmable value.
- The following VLAN tagging operations must be supported for **Single-tagged** frames:
  - Remove tag.
  - Replace tag with a programmable inner tag.
  - Replace VID with a programmable inner VID and preserve Pbits of original tag.
  - Replace tag with a programmable inner VLAN tag and add a programmable outer VLAN tag.
  - Add a programmable outer VLAN tag.
  - Add a VLAN tag by using a programmable outer VID and copying the Pbits from the original tag.
  - Add a programmable outer VLAN tag, and replace the VID of the original tag with a programmable VID value.
  - TPID and DEI handling on Add and Replace treatments:
    - Set TPID to 0x8100.
    - Use TPID and DEI from original tag.
    - Use the default TPID and copy DEI from original tag.
    - Use the default TPID and set DEI to 0.
    - Use the default TPID and set DEI to 1.
- The following VLAN tagging operations must be supported for **Double-tagged** frames:
  - Remove outer tag.
  - Remove both tags.
  - Remove outer tag and replace VID of inner tag with a programmable inner VID.
  - Replace outer VLAN tag with a programmable VLAN tag.
  - Replace VID of outer tag with a programmable outer VID.
  - Replace VID of outer tag with a programmable outer VID, and replace VID of inner tag with a programmable inner VID
  - TPID and DEI handling on replace treatments:
    - Set TPID to 0x8100
    - Use TPID and DEI from inner tag
    - Use the default TPID and copy DEI from inner tag

  - Use the default TPID and set DEI to 0
  - Use the default TPID and set DEI to 1
- The VLAN interface must be attachable to any LAN or WAN interface
- In S-VLAN filtering Traffic Scenarios (ITU-T G.984.4):
  – VLAN interfaces are attached to LAN Interfaces.
  – In the upstream direction, VLAN tagging operations must be performed before the frame is bridged to a WAN interface.
  – In the downstream direction, VLAN tagging operations must be performed after the frame is bridged to a LAN interface.
- In C-VLAN filtering Traffic Scenarios (ITU-T G.984.4):
  – VLAN interfaces are attached to WAN Interfaces.
  – In the upstream direction, VLAN tagging operations must be performed after the frame is bridged to a WAN interface.
  – In the downstream direction, VLAN tagging operations must be performed before the frame is bridged to a LAN interface.

High Level Architecture

A vlanctl interface has a symmetrical design. The same VLAN operations that can be applied to Receive frames can also be applied to Transmit frames. The Receive Direction refers to frames received from a real device interface, while the Transmit Direction corresponds to frames transmitted to a real device interface. The operations on both directions are fully configurable by the user via the vlanctl program. For simplicity, the architecture of the vlanctl interface is described in this document by abstracting from direction, as it applies equally to both received and transmitted frames.

Tagging operations are stored in Tagging Rule Tables, used to filter and tag frames. A given Tagging Rule Table applies to frames with a certain number of VLAN tags, and each entry in the table represents a Tagging Rule. A Tagging Rule comprises a filtering part and a treatment part. Incoming frames are parsed and matched against the filtering part of each tagging rule of the Tagging Rule Table that corresponds to the number of VLAN tags encountered in the frame. The filter matching process occurs in list order, i.e., the first rule that matches the packet in a given Tagging Rule Table is selected as the active rule, and the packet is then treated according to that rule.

Incoming frames may not match the programmed Tagging Rules of a given Tagging Rule Table. In this scenario, the default behavior of the Tagging Rule Table is applied. The default behavior of all tables are initially set to ACCEPT, but may be changed via a *vlanctl* command to DROP. It is also possible to define a default Tagging Rule on a given table by creating a Tagging Rule without specifying filters. If such a rule is created, all frames will match it so it important to place it at the end of the table. Treatments may also be programmed in such rule, and will be executed normally.

Tagging Rule Tables are always associated to a Real Device on both the Receive and Transmit directions. They are created and initialized when the first VOPI associated to the Real Device is created. Up to four VLAN Tags are supported by default, so 5 VLAN Tagging Tables (untagged, 1 tag, 2 tags, 3 tags and 4 tags) are created for each direction, for a total of 10 tables per Real Device. It is possible to support more than four VLAN tags at compile time.

Each Tagging Rule Table applies to its respective incoming frame type, i.e., a tagging rule in a table that corresponds to double-tagged frames will only apply to double-tagged frames received from the attached Real Interface, even though a rule in the single-tagged table might match the outer tag of the double-tagged frame.

A Tagging Rule Table is identified by a Real Device, a Direction, and the Number of Tags that it represents. A Tagging Rule is identified by an identification number that is assigned when the rule is created. Upon creation, a Tagging Rule may be appended to the end of a Tagging Rule Table, before an existing Tagging Rule, or after an existing Tagging Rule. Once created, a Tagging Rule may not be modified. If an existing Tagging Rule has to be modified, a new Tagging Rule containing the new filters and treatments may be inserted in its position, and then the original rule should be deleted.

The vlanctl Utility

The vlanctl utility is an application used to set-up and manage a vlanctl interface via a set of command line arguments. The vlanctl commands result in calls to the IOCTL API of the Broadcom VLAN kernel module. The definition of this API is outside the scope of this document.

The commands and arguments supported by vlanctl are described in the next sections of this document.

Notes:

- In some arguments, a tag number is required. Zero is used to indicate the outermost tag and subsequent numbers for the inner tags, if any. For instance, the outer tag of a double-tagged frame would be considered as tag number 0, and the inner tag would be considered as tag number 1. It is important to notice that as tags are added or removed the resulting outer tag is still considered tag 0. For instance, when a single-tagged frame is received the unique tag is considered tag number 0. If an outer tag is added the new tag becomes tag number 0, and the original tag is now considered tag number 1. Supporting tag numbering in the arguments allows vlanctl to operate in frames with any arbitrary number of tags.
- When defining filters and treatments, the user must think in terms of VLAN Headers, not VLAN Tags. A VLAN Header is composed by a TCI field (PBITS, CFI, and VID), followed by a Tag Ethertype which defines the type of the next header. For instance, a double-tagged frame would have the following headers: Ethernet Header (DA, SA, and Ethertype), the outer VLAN Header (TCI + Tag Ethertype), the inner VLAN Header (TCI + Tag Ethertype), and the L3 Header.
- When a Tagging rules is created, all respective filters and treatments MUST be specified before the command that adds the rule to the Tagging Rule Table is issued (e.g.: --rule-append, --rule-insert-before, or –rule-insert-after). Any filters or treatments specified after the rule insertion command will be ignored.
- Only one Tagging Rule may be specified to vlanctl at a time.

VLANCTL Commands

This section gives a summary of all the commands supported by vlanctl. Only one command can be issued at a time.

Command Qualifiers

These qualifiers are needed for many of the commands issued. The qualifiers specify the interface, traffic direction and level of VLAN tagging for the traffic before any changes are made by the vlanctl rules.

Table 1: Command Qualifiers

Command	***Description***
--if <if_name>	Sets the target Interface of a composite vlanctl command to <if_name>.
--rx	Sets the direction of a composite vlanctl command to RECEIVE. Exclusive with --tx.
--tx	Sets the direction of a composite vlanctl command to TRANSMIT. Exclusive with --rx
--tags <nbr_of_tags>	Sets the number of tags of a composite vlanctl command to <nbr_of_tags>.

Interface Commands

Table 2: Interface Commands

Command	Description
--if-create <real_if_name><if_index>	Creates a new vlanctl interface named “<real_if_name>.v<if_index>” and attaches it to the real device <real_if_name>. For example, if this command was executed for the eth0 real interface and the vlanctl interface index was set to 3, the resulting interface would be named “eth0.v3”. ***Example:*** `# vlanctl --if-create eth0 3 --if eth0`
--if-create-name <real_if_name> <VLAN_if_name>	Creates a new vlanctl named “VLAN_if_name” and attaches it to the real device “real_if_name”. ***Example:*** `# vlanctl --if-create-name eth2 voice`
--if-delete <VLAN_if_name>	Destroy the vlanctl interface named <VLAN_if_name>. ***Example:*** `# vlanctl --if-delete voice`
--if-suffix <separator>	Sets the separator character when creating a VLAN Control Interface on a real interface. <separator> is defaulted to ".v". This may need to be changed to match when VLAN interfaces are created using other methods that default to ".". This option must precede the --if-create command. ***Example:*** `# vlanctl --if-suffix . --if-create eth2 4 --if eth2`
--mcast	This will set IFF_MULTICAST privilege flags attribute to the created virtual interface. This affects the behavior for ONT or RG modes. ***Example:*** `# vlanctl --if-create eth2 4 --if eth2 --mcast`

Table 2: Interface Commands (Cont.)

Command	Description
--routed	Use to configure the interface as a router interface with an assigned MAC address. This option is not very useful for ONTs. Can be used in conjunction with untagged packets and the --filter-VLAN-dev-mac-addr filter to associate packets where the MAC address matches the interface MAC. Requires CONFIG_BCM_VLAN_ROUTED_WAN_USES_ROOT_DEV_MAC to be unset in the build. (Which is "Assign Same MAC address to Routed WAN Interface as root" in make menuconfig under the "Ethernet, Switch and VLAN Selection".) ***Example:*** `# vlanctl --if-create eth2 4 --if eth2 --routed`

--set-if-mode-ont	Set real device mode to ONT. This is the default mode.
	See --set-if-mode-rg for the alternative mode.
	• Multicast packets will be forwarded to all vlanctl interfaces that have the --mcast qualifier upon VLAN rule matching. Multicast packets will be matched against only the first VLAN rule. If there are no rule matches, the packet is handled by the default behavior of the corresponding VLAN rule table. This allows the user to explicitly control which VLAN interfaces will get the multicast packets.
	• Unicast packets will match only on the first VLAN rule.
	  – If a rule is hit, the packet is assigned to the specified receive interface of the rule.
	  – If all rules miss, then the packet will be assigned to the default vlanctl interface associated with the table (if the rule table is configured to ACCEPT packets on a miss.) Otherwise the packet will be dropped.
	Example: `# vlanctl --if-suffix . --routed --if-create eth2 100 --if eth4 --set-if-mode-ont`

Table 2: Interface Commands (Cont.)

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td>--set-if-mode-rg</td><td>Set real device mode to RG. This has the following implications on the receive direction (on the transmit direction the packets are always forwarded to the real interface):

Multicast packets will only be forwarded to VLAN interfaces upon VLAN rule matching. Multicast packets will be matched against ALL VLAN rules, i.e., the rule matching will NOT stop at the first matching rule. If there are no rule matches, the packet will be handled by the default behavior of the corresponding VLAN rule table. This allows the user to explicitly control which VLAN interfaces will get the multicast packets.
Unicast packets will match only on the first VLAN. The Unicast forwarding behavior is the following:

VLAN rule misses: Each VLAN rule table can be individually configured to DROP or ACCEPT packets on misses. If ACCEPT is configured, the packet will be assigned to the default vlanctl interface associated with the table.
VLAN rule hits: Received interfaces are always specified on a matching rule.
Untagged packets: In order to support "routed" (--routed option) VLAN interfaces, the user can specify a filter that matches the destination MAC address of the packet against the MAC address of the VLAN interface specified in the VLAN rule (--filter-VLAN-dev-mac-addr). This filter can be further defined to match the MAC addresses of all packet types, or unicast packets only.
Tagged packets: There are two cases:
a) If the specified VLAN interface is "routed" (--routed option) assign packet to the specified VLAN interface if the destination MAC address of the packet matches the MAC address of the VLAN interface. If the MAC addresses don't match, drop the packet.
b) If the specified VLAN interface is "bridged" (--routed option not used) assign packet to the specified VLAN interface.

Example: <code># vlanctl --if-suffix . --if-create eth2 100 --if eth4 --set-if-mode-rg</code></td></tr>
</table>

Rule Creation Commands

Table 3: Rule Creation Commands

Command	***Description***
--rule-append	Inserts a new Tagging Rule as the last rule of the specified Tagging Rule Table. Dependencies: --if, --rx or —-tx, and --tags. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --filter-dscp 55 --set-rxif eth2.4 --rule-append`
--rule-insert-before <rule-id>	Inserts a new Tagging Rule before the Tagging Rule whose identifier matches <rule-id> in the specified Tagging Rule Table. Dependencies: --if, --rx or —-tx, and --tags. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --filter-dscp 55 --set-rxif eth2.4 --rule-insert-before 1`
--rule-insert-after <rule-id>	Inserts a new Tagging Rule after the Tagging Rule whose identifier matches <rule-id> in the specified Tagging Rule Table. Dependencies: --if, --rx or —-tx, and --tags. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --filter-dscp 55 --set-rxif eth2.4 --rule-insert-before 1`
--rule-remove <rule-id>	Removes the Tagging Rule that matches <rule-id> from the specified Tagging Rule Table. Dependencies: --if, --rx or —-tx, and --tags. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --rule-remove 1`
--rule-remove-by-filter	Removes the Tagging Rule that matches the specified rule provided in the command. Dependencies: --if, --rx or --tx, and --tags. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --filter-dscp --rule-remove-by-filter`
--rule-remove-all <VLAN_if_name>	Removes all the Tagging Rules that matches the provided VLAN device. ***Example:*** `# vlanctl --rule-remove-all eth2.100`
--show-table	Lists all Tagging Rules stored in the specified Tagging Rule Table. Dependencies: --if, --rx or —-tx, and --tags. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --show-table`
--default-tpid <tpid>	Sets the default TPID value of a tagging rule table to <tpid>. When a table is created, its default TPID value is set to 0x8100. Dependencies: --if, --rx or —-tx, and --tags. Example: ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --default-tpid 0x9100`
--default-pbits <pbits>	Sets the default PBITS value of a tagging rule table to <pbits>. When a table is created, its default PBITS value is set to 0. Range is 0 to 7. Dependencies: --if, --rx or —-tx, and --tags. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --default-pbits 5`
--default-cfi <cfi>	Sets the default CFI value of a tagging rule table to <cfi>. When a table is created, its default CFI value is set to 0. Range is 0 or 1. Dependencies: --if, --rx or —-tx, and --tags. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --default-cfi 1`
--default-vid <vid>	Sets the default VID value of a tagging rule table to <vid>. When a table is created, its default VID value is set to 1 (as per IEEE 802.1Q). Dependencies: --if, --rx or —-tx, and --tags. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --default-vid 100`

Table 3: Rule Creation Commands (Cont.)

Command	***Description***
--default-miss-accept <default_rx_VLAN_interface>	In the RX direction, set the default action to accept packets when rules are missed. By default, packets received that miss all rules are dropped. When set to accept, these packets will be passed to the supplied <default_rx_VLAN_interface>. In the TX direction, set the default action to accept (transmit) packets that miss the rules, which is the default. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --default-miss-accept eth2.4`
--default-miss-drop	Sets the default action for this interface to drop (or not transmit) packets that miss the rules. Drop is the default action for the RX direction. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --default-miss-drop`
--cfg-dscp2pbits <dscp> <pbits>	Programs the entry number <dscp> of the DSCP-TO-PBITS translation table of a Real Device to the value specified by <pbits>. When a tagging rule table is created, the default values of the DSCP-TO-PBITS table are set by copying the lowest 3 bits of each DSCP value as the PBITS value, for each entry in the table. For instance, the following entries are programmed by default: DSCP=5:PBITS=5, DSCP=15:PBITS=7, etc. The DSCP-TO-PBITS translation table has 64 entries. Dependencies: --if. ***Example:*** `# vlanctl --if eth2 --cfg-dscp2pbits 55 3`
--show-dscp2pbits	Lists the values programmed in the DSCP-TO-PBITS table of the specified Real Device. Dependencies: --if. ***Example:*** `# vlanctl --if eth2 --show-dscp2pbits`
--cfg-tpid <tpid0> <tpid1> <tpid2> <tpid3>	Configures the TPID Table entries of a given Real Interface. The configured TPID values are used to identify VLAN Headers of packets received from and transmitted to the vlanctl interfaces created for a given Real Interface. Four values must always be specified. The default TPID values are 0x8100, 0x8100, 0x8100, and 0x8100. Dependencies: --if. ***Example:*** `# vlanctl --if eth2 --cfg-tpid 0x9100 0x8100 0x9100 0x8100`
--show-tpid	Lists the values programmed in the TPID Table of the specified Real Device. Dependencies: --if. ***Example:*** `# vlanctl --if eth2 --show-tpid`
--local-stats <VLAN_if_name>	Shows the statistics counters maintained for the vlanctl interface named <VLAN_if_name>. These counters are complimentary to the standard counters maintained for the device, which can be read via the Linux *ifconfig* command. ***Example:*** `# vlanctl --local-stats eth2`

Tagging Rule Filters

Table 4: Tagging Rule Filters

Command	***Description***
--filter-ethertype <ethertype>	Match the Ethertype field in the Ethernet Header of incoming frames against <ethertype>. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-ethertype 0x8864 --rule-append`
--filter-pbits <pbits> <tag_nbr>	Match the PBITS value of VLAN Header number <tag_nbr> of incoming frames against <pbits>. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-pbits 3 1 --rule-append`
--filter-cfi <cfi> <tag_nbr>	Match the CFI bit of VLAN Header number <tag_nbr> of incoming frames against <cfi>. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-cfi 1 1 --rule-append`
--filter-vid <vid> <tag_nbr>	Match the VID value of VLAN Header number <tag_nbr> of incoming frames against <vid>. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-vid 100 --rule-append`
--filter-tag-ethertype <ethertype> <tag_nbr>	Match the Ethertype field of the VLAN Header number <tag_nbr> of incoming frames against <ethertype>. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-tag-ethertype 0x9100 1 --rule-append`
--filter-ipproto <ipproto>	Match the IP Protocol value of incoming IP Header against <ipproto> value. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-ipproto 2 --rule-append`
--filter-dscp <dscp>	Match the DSCP value in the IPv4 header of incoming frames against <dscp>. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-dscp 23 --rule-append`
--filter-dscp2pbits <dscp2pbits>	Match the translated dscp2pbit value after the translation has occurred. Example: ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.1 --filter-dscp2pbits 4 --rule-append`
--filter-rxif <VLAN-if-name>	Match the received vlanctl interface against <VLAN-if-name>. This filter creates a TRANSMIT rule that will match only packets received from a specific vlanctl interface. ***Example:*** `# vlanctl --if eth2 --tx --tags 1 --set-rxif eth2.100 --filter-rxif eth1 --rule-append`
--filter-txif <VLAN-if-name>	Match the transmitting vlanctl interface against <VLAN-if-name>. This filter can be used to bind a Tagging Rule to a specific vlanctl interface on the TRANSMIT direction. This filter is not applicable for rules in the RECEIVE direction. TRANSMIT rules without this filter will apply to all frames transmitted from all vlanctl interface attached to the Real Device. ***Example:*** `# vlanctl --if eth2 --tx --tags 1 --filter-txif eth2.4 --rule-append`

Table 4: Tagging Rule Filters (Cont.)

Command	Description
--filter-skb-prio <priority>	Match the SKB priority of incoming frames against <priority>. **Example:** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-skb-prio 1 --rule-append`
--filter-skb-mark-flowid <flowid>	Match the Flow ID subfield of the SKB Mark field against <flowid>. The SKB Mark Flow ID subfield can be used as a way to correlate packet classification made by other Linux modules (such as ebtables and iptables) with Tagging Rules. A possible usage for this filter would be to direct packets generated by an application to a specific port of a real interface (such as a GPON port) based on layer 3 filters. In this example a socket would be created on a vlanctl interface, IP Tables rules would be created to identify flows and set Flow IDs, and Tagging rules would be created to match on such Flow IDs and apply treatments, such as setting the destination GEM Port and Queue. **Example:** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-skb-mark-flowid 12 --rule-append`
--filter-skb-mark-port <port>	Match the Port subfield of the SKB Mark field against <port>. This filter can be used to bind certain Tagging Rules with a specific Real Interface port (for instance a GPON Port). **Example:** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-skb-mark-port 5 --rule-append`
--filter-VLAN-dev-mac-addr <ignore_if_multicast>	Match the received frame destination MAC address against the received virtual interface. Set <ignore_if_multicast> to 0 to apply filter on all received frames. Set <ignore_if_multicast> to 1 to apply filter on unicast frames only. This filter is not applicable for rules in the TRANSMIT direction. **Example:** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-VLAN-dev-mac-addr 0 --rule-append`
--filter-unicast	Match all packets that are unicast. This can be combined with the multicast or broadcast flags. When these filter flags are set, traffic that does not match the flags will not match the rule. Without any flags set (or all flags set) all traffic will be accepted. **Example:** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-vid 100 1 --filter-unicast --rule-append`
--filter-multicast	Match all packets that are multicast. This can be combined with the unicast or broadcast flags. When these filter flags are set, traffic that does not match the flags will not match the rule. Without any flags set (or all flags set) all traffic will be accepted. **Example:** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-vid 100 1 --filter-multicast --rule-append`
--filter-broadcast	Match all packets that are broadcast. This can be combined with the unicast or multicast flags. When these filter flags are set, traffic that does not match the flags will not match the rule. Without any flags set (or all flags set) all traffic will be accepted. **Example:** `# vlanctl --if eth2 --rx --tags 1 --set-rxif eth2.100 --filter-vid 100 1 --filter-broadcast --rule-append`

Tagging Rule Treatments

Table 5: Tagging Rule Treatments

Command	Description
--pop-tag	Remove the outermost VLAN tag. If multiple tags are to be removed, this treatment should be specified for each VLAN tag to be removed. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --filter-vid 100 1 --pop-tag --rule-append`
--push-tag	Add the default VLAN tag of the corresponding Tagging Rule Table as the new outer tag. The default TPID value will be used as the new Ethertype value in the Ethernet header, the existing Ethertype of the Ethernet Header will be used as the Tag Ethertype field of the new tag, and the default PBITS, CFI, and VID will be used as the TCI of the new tag. If multiple tags are to be inserted, this treatment must be specified for each VLAN tag to be inserted. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --filter-ethertype 0x8864 --push-tag --set-pbits 3 0 --set-cfi 1 0 --set-vid 100 0 --rule-append`
--deaggr-tag	VLAN aggregation is a feature of EPON SFU, it can be enabled by using menuconfig to set "Ethernet and VLAN Selection/Support VLAN Aggregation" (BRCM_VLAN_AGGREGATION in the build profile). When the feature is enabled, the Linux bridge forwarding tables learn the VID and MAC address of the packets. When the VLAN action is --deaggr-tag, the VLAN module looks for a bridge forwarding database (FDB) by DA, and replaces the VID according to the VID of the forwarding database (FDB). • Upstream is VLAN aggregation—from LAN side, VID "aaa" and "bbb" will be modified as "xxx" and sent to WAN side. • Downstream is VALN de-aggregation—from WAN side, VID is xxx, look for (FDB) to get correct VID, then replace the VID as "aaa" or "bbb". ***Example:*** `# vlanctl --if eth1 --tx --tags 1 --filter-txif eth1.0 -deaggr-tag --rule-append.`
--set-ethertype <ethertype>	Set the Ethertype value of the Ethernet Header to <ethertype>. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --filter-ethertype 0x8864 --push-tag --set-ethertype 0x9200 --set-tag-ethertype 0x9100 0 --set-pbits 3 0 --set-cfi 1 0 --set-vid 100 0 --set-rxif eth2.3  --rule-append`
--set-pbits <pbits> <tag_nbr>	Set the PBITS value of the VLAN Header number <tag_nbr> to <pbits>. See `--set-ethertype` for command example.
--set-cfi <cfi> <tag_nbr>	Set the CFI bit of the VLAN Header number <tag_nbr> to <cfi>. See `--set-ethertype` for command example.
--set-vid <vid> <tag_nbr>	Set the VID of the VLAN Header number <tag_nbr> to <vid>. See `--set-ethertype` for command example.
--set-tag-ethertype <ethertype>	Set the Ethertype field of the VLAN Header number <tag_nbr> to <ethertype>. See `--set-ethertype` for command example.
--set-dscp <dscp>	Set the IPv4 DSCP value of the matching frame to <dscp>. ***Example:*** `# vlanctl --if eth2 --rx --tags 0 --filter-ethertype 0x8864 --set-dscp 21 --set-rxif eth2.3 --rule-append`

Table 5: Tagging Rule Treatments (Cont.)

Command	***Description***
--copy-pbits <from_tag_nbr> <to_tag_nbr>	Copy the PBITS value from VLAN Header number <from_tag_nbr> to VLAN Header number <to_tag_nbr>. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --push-tag --copy-pbits 1 0 --copy-cfi 1 0 --copy-vid 1 0 --copy-tag-ethertype 1 0 --set-rxif eth2.v100 --rule-append`
--copy-cfi <from_tag_nbr> <to_tag_nbr>	Copy the CFI value from VLAN Header number <from_tag_nbr> to VLAN Header number <to_tag_nbr>. See `--copy-pbits` for a command example.
--copy-vid <from_tag_nbr> <to_tag_nbr>	Copy the VID value from VLAN Header number <from_tag_nbr> to VLAN Header number <to_tag_nbr>. See `--copy-pbits` for a command example.
--copy-tag-ethertype <from_tag_nbr> <to_tag_nbr>	Copy the Ethertype value from VLAN Header number <from_tag_nbr> to VLAN Header number <to_tag_nbr>. See `--copy-pbits` for a command example.
--dscp2pbits <tag_nbr>	Translate the IPv4 DSCP into a PIBTS value, and write the translated PBITS value in the VLAN Header number <tag_nbr>. The DSCP-To-PBITS table of the respective Real Device is used for translation. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --dscp2pbits 1 --set-rxif eth2.100 --rule-append`
--set-rxif <VLAN_if_name>	Forward frames in the RECEIVE direction that match this rule to the vlanctl interface specified in <VLAN_if_name>. If not specified, the frame will be forwarded to a randomly chosen vlanctl interface. In the TRANSMIT direction this has no effect. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --dscp2pbits 1 --set-rxif eth2.100 --rule-append`
--drop-frame	Drop the matching frame. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --filter-vid 200 --set-rxif eth2.100 --drop-frame --rule-append`
--set-skb-prio <priority>	Set the SKB priority to <priority>. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --set-skb-prio 3 --set-rxif eth2.100 --rule-append`
--set-skb-mark-port <port>	Set the Port subfield of the SKB Mark field to <port>. The SKB Mark Port subfield is used by the Broadcom device drivers to send a frame to a specific port within a Real Interface. For instance, a GPON Real Interface may have been configured with multiple GEM Ports. When a packet is sent to that interface, the driver uses the SKB Mark Port subfield as the GEM Port to which the packets will be transmitted. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --filter-vid 200 0 --set-skb-mark-port 2 --set-rxif eth2.100 --rule-append`
--set-skb-mark-queue <queue>	Set the Queue subfield of the SKB Mark field to <queue>. The SKB Mark Queue subfield is used by the Broadcom device drivers to determine the queue to which transmit a frame. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --filter-vid 200 0 --set-skb-mark-queue 5 --set-rxif eth2.100 --rule-append`

Table 5: Tagging Rule Treatments (Cont.)

Command	Description
--set-skb-mark-flowid <flowid>	Set the Flow ID subfield of the SKB Mark field to <flowid>. The SKB Mark Flow ID subfield can be used as a way to correlate packet classification made by Tagging Rules with other Linux modules (such as ebtables and iptables). ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --filter-vid 200 0 --set-skb-mark-flowid 3 --set-rxif eth2.100 --rule-append`
--ovrd-learn-vid <vid>	Causes the accelerator learning bridge to use the specified <vid> in the command when learning the MAC source address. ***Example:*** `# vlanctl --if eth2 --rx --tags 1 --filter-vid 200 0 --ovrd-learn-vid 100 --set-rxif eth2.100 --rule-append`
--rule-type <type>	Set the type of rule. Set <type> to 0 for flow or 1 for QoS (Quality of Service). By default all rules are flow rules. • QoS rules may modify a packet but do not specify the routing or bridging of a packet. • QoS rules can modify packets but do not affect if the packet is dropped or accepted. • QoS rules that match a packet are applied after the flow rule actions are taken. Then the packet is handled according to the outcome of the flow rules, matching or not matching. ***Example:*** `# vlanctl --if eth2 --rx --tags 2 --rule-type 1 --filter-vid 200 1 --set-pbits 7 1 --set-rxif eth2.1 --rule-append`

Extra Commands

Table 6: Extra Commands

Command	Description
--create-flows <rx_VLAN_ifname> <tx_VLAN_ifname>	Setup and activate rules for a Layer 2 flow path going from the rx_VLAN_ifname to the tx_VLAN_ifname. Merges the receive and transmit rules into complete rules for the end-to-end flow-based data path. ***Example:*** `# vlanctl --create-flows gpondef.100 sid0.100`
--delete-flows <rx_VLAN_ifname> <tx_VLAN_ifname>	Deactivate and remove the flows for the path rx_VLAN_ifname to the tx_VLAN_ifname. ***Example:*** `# vlanctl --delete-flows gpondef.100 sid0.100`

Usage Examples

This section gives several usage examples.

Scenario 1: Creating an Eth4.2 Interface

Create an eth4.2 interface and have it receive packets that come in on eth4 with no tags.

```
vlanctl --if-suffix . --routed --if-create eth4 2 --if eth4 --set-if-mode-rg --mcast
vlanctl --filter-txif eth4.2 --if eth4 --tx --tags 0 --rule-append
vlanctl --set-rxif eth4.2 --if eth4 --tags 0 --rule-append
```

Scenario 2: Creating a SID

Create a SID and then map the G.int SID to a Gem port.

```
ifconfig sid0 up
vlanctl --if-create sid0 99
vlanctl --rx --tags 1 --if sid0 --set-rxif sid0.99 --rule-append
vlanctl --tx --tags 1 --if sid0 --rule-append
ifconfig sid0.99 up

gponif -c gpondef
gponif -a gpondef -g 0
ifconfig gpondef up
vlanctl --if-create gpondef 99
vlanctl --rx --tags 1 --if gpondef --filter-skb-mark-port 1 --filter-vid 99 0 --set-rxif
gpondef.99 --rule-append
vlanctl --tx --tags 1 --if gpondef --filter-txif gpondef.99 --set-skb-mark-queue 0 --set-skb-
mark-port 1 --rule-append
ifconfig gpondef.99 up

vlanctl --create-flows gpondef.99 sid0.99
vlanctl --create-flows sid0.99 gpondef.99
```

Scenario 3: Creating a VLAN 100 Interface

Create an interface that handles VLAN 100.

```
brctl delif br0 eth1
vlanctl --if-create-name eth1 eth1.1
// Receive frames with vid 100 and remove the tag
vlanctl --if eth1 --rx --tags 1 --filter-vid 100 0 --pop-tag --set-rxif eth1.1 --rule-append
// Drop frames with any other VID - make sure this rule is added after previous rule
vlanctl --if eth1 --rx --tags 1 --drop-frame --set-rxif eth1.1 --rule-append
// Add tag with vid 100 to the outgoing frames passing through eth1.1
vlanctl --if eth1 --tx --tags 0 --filter-txif eth1.1 --push-tag --set-vid 100 0 --set-pbits 0 0
--rule-append
ifconfig eth1.1 up
brctl addif br0 eth1.1
```

Scenario 4: Interface Manipulation

Create four vlanctl interfaces and add them to a Linux bridge.

```
brctl delif br0 eth0
brctl delif br0 eth1
vlanctl --if-suffix . --if-create eth0 0
vlanctl --if-suffix . --if-create eth0 1
vlanctl --if-suffix . --if-create eth1 0
vlanctl --if-suffix . --if-create eth1 1

ifconfig eth0.0 up
ifconfig eth0.1 up
ifconfig eth1.0 up
ifconfig eth1.1 up

brctl addbr br_VLAN
brctl addif br_VLAN eth0.0
brctl addif br_VLAN eth0.1
brctl addif br_VLAN eth1.0
brctl addif br_VLAN eth1.1
ifconfig br_VLAN up
```

Scenario 5: Building a QinQ (Double Tag) VLAN

This example shows how to build a QinQ (double tag) VLAN between a WAN and LAN interface.

In this example the WAN is eth4 and the LAN is eth0. The example can be extended to include other LAN interfaces in the bridge or to create multiple bridge interfaces between the LAN to WAN side.

Note that this example does not take into account any internal switch on the System-on-a-chip,

which may need to be configured using PVLANs (port based VLANs) to prevent the switch from routing traffic from port-to-port automatically. Such settings would be device-specific and are beyond the scope of this document.

```
// Remove the two interfaces from the standard bridge:
# brctl delif br0 eth0
# brctl delif br0 eth4

// Create 3 VLAN interfaces: one for WAN, one for tagged LAN and one for untagged LAN
# vlanctl --if-create-name eth4 wan_vlan --if eth4 --set-if-mode-rg
# vlanctl --if-create-name eth0 lan_vlan --if eth0 --set-if-mode-rg
# vlanctl --if-create-name eth0 lan_novlan --if eth0 --set-if-mode-rg

// Configure our default tpid values
# vlanctl --if eth4 --cfg-tpid 0x9100 0x9200 0x8100 0x8100
# vlanctl --if eth0 --cfg-tpid 0x9100 0x9200 0x8100 0x8100

// Filter untagged traffic in both directions.
// Also add the untagged LAN interface again to the normal bridge interface.
# vlanctl --if eth0 --rx --tags 0 --set-rxif lan_novlan --rule-append
# vlanctl --if eth0 --tx --tags 0 --filter-txif lan_novlan --rule-append
# ifconfig lan_novlan up
# brctl addif br0 lan_novlan
```

```
//Filter all tagged traffic and send it to the new VLAN bridge interface
# vlanctl --if eth0 --tx --tags 1 --filter-txif lan_vlan --rule-append
# vlanctl --if eth0 --rx --tags 1 --set-rxif lan_vlan --rule-append
# ifconfig lan_vlan up
```

Other Commands

Set the default TPID, VID and PBITS values, and modify a few entries of the DSCP to PBITS table.

```
vlanctl –-if eth0 –-rx –tags 0 –-default-tpid 0x9100 –-default-vid 1000 –-default-pbits 5 –-cfg-
dscp2pbits 9 0 –- cfg-dscp2pbits 10 1 –- cfg-dscp2pbits 62 7
```

Append a new rule for untagged frames on the receive direction that matches on a DSCP value and forwards the frame to a specific vlanctl interface.

```
vlanctl --if eth1 --rx --tags 0 --filter-dscp 63 --set-rxif eth1.v1 --rule-append
```

Append a new rule for untagged frames on the receive direction that match on Ethertype 0x8864, pushes two tags, and sets several tag fields.

```
vlanctl --if eth0 --rx --tags 0 --filter-ethertype 0x8864 --push-tag --push-tag --set-pbits 3 0
--set-cfi 1 0 --set-vid 30 0 --set-pbits 4 1 --set-cfi 1 1 --set-vid 40 1 --rule-append
```

Append a new rule for single-tagged frames on the receive direction that match on DSCP = 21 and IP-Protocol = 2 (IGMP), pushes one tag, sets the Ethertype, the outer tag Ethertype, and the IPv4 DSCP.

```
vlanctl --if eth0 --rx --tags 1 --filter-dscp 21 –-filter-ipproto 2   --push-tag --set-ethertype
0x9200 --set-tag-ethertype 0x9100 0 --set-dscp 43 --rule-append
```

Append a new rule for single-tagged frames on the transmit direction that matches on VID and DSCP, does a DSCP to PBITS translation and sets the VID.

```
vlanctl --if eth1 --tx --tags 1 --filter-vid 100 0 --filter-dscp 35 --dscp2pbits 0 --set-vid 200
0 --rule-append
```

Append a new rule for double-tagged frames on the receive direction that matches on several fields of both VLAN Headers, pops both tags and sets the DSCP.

```
vlanctl --if eth0 --rx --tags 2 --filter-pbits 3 0 --filter-vid 100 0 --filter-pbits 7 1 --filter-
vid 200 1 --pop-tag --pop-tag --set-dscp 50 --rule-append
```

Removing a Rule

This example shows how to remove a rule using the filter match.

```
# vlanctl --if-suffix . --if-create eth2 4 --if eth2
netdev path : eth2.4 -> eth2
# vlanctl --if eth2 --rx --tags 1 --filter-dscp 63 --set-rxif eth2.4 --rule-append
Created new Tag Rule: dev=eth2, dir=0, tags=1, id=0
# vlanctl --if eth2 --rx --tags 1 --filter-dscp 55 --set-rxif eth2.4 --rule-append
Created new Tag Rule: dev=eth2, dir=0, tags=1, id=1
# vlanctl --if eth2 --rx --tags 1 --show-table
(Shows two entries - one for DSCP value of 63 and one for 55)
# vlanctl --if eth2 --rx --tags 1 --filter-dscp 55 --rule-remove-by-filter
# vlanctl --if eth2 --rx --tags 1 --show-table
(Shows one entry - the one for DSCP value 63)
// Push a new tag on WAN traffic and send it to the WAN VLAN interface.
# vlanctl --if eth4 --tx --tags 1 --filter-txif wan_vlan --push-tag --set-ethertype 0x8100 --set-
vid 100 0 --rule-append
# vlanctl --if eth4 --rx --tags 2 --set-rxif wan_vlan --pop-tag --rule-append
# ifconfig wan_vlan up

// Create the WAN to LAN bridge and bring up interfaces
# brctl addbr br_vlan
# brctl addif br_vlan lan_vlan
# brctl addif br_vlan wan_vlan
# ifconfig br_vlan up
# ifconfig wan_vlan up
# ifconfig lan_vlan up
```


Integration with the Linux Networking Stack

The Broadcom VLAN Control Interface is a kernel module. When loaded, it registers its receive handler in netif_receive_skb() by setting the bcm_vlan_handle_frame_hook function pointer. This hook is placed right before the Linux handle_bridge hook, which is used by the standard Linux bridge code to process frames. All received frames are processed by the vlanctl interface receive function, which basically tries to match the receive device against all Real Devices that have attached vlanctl interfaces. If a match is found, the frame is parsed and will be matched against the Tagging Rules stored in the Tagging Rule Table corresponding to the number of existing tags in the frame, the attached Real Device, and the transmit direction. If a match is found, all treatments specified in the matching rule will be applied. Otherwise, the default behavior is applied (accept or discard). If the frame is accepted, the receive handler will set the net device pointer of the frame's SKB structure to point to the net device structure of the target vlanctl interface. If the vlanctl interface has been mapped to a bridge port, the frame is processed by the Linux Bridge code immediately after the vlanctl interface receive handler returns. Otherwise, the frame is passed to other protocol handlers for further processing.

The Linux vconfig module registers itself as a protocol handler for Ethertype 0x8100, and can be used simultaneously with the VLAN Control Interface. However, in order to increase performance, the developer should consider not loading vconfig when the vlanctl interface module is loaded, since vconfig will consume CPU cycles for each received tagged frame even if no vconfig VLAN interfaces are created. However, nothing prevents vconfig and vlanctl interfaces from co-existing, as long as they operate on different Real Devices. However, this should not be needed, as the vlanctl interface can be configured to behave exactly like vconfig.

When a vlanctl interface is created, a net_device struct is allocated via the alloc_netdev() API. Once the net device structure is initialized, it is registered into the kernel via the register_netdev() API.

On the transmit direction, the vlanctl interface transmit handler is registered in the hard_start_xmit function pointer of the device structure of all vlanctl interfaces. When a frame is transmitted to a vlanctl interface, the frame is parsed and is matched against all Tagging Rules stored in the Tagging Rule Table corresponding to the number of existing tags in the frame, the Real Device, and the transmit direction. If a match is found, all treatments specified in the matching rule will be applied. Otherwise, the default behavior is applied (accept or discard). If the frame is accepted, the transmit handler will set the net device pointer of the frame's SKB structure to point to the net device structure of the corresponding Real Device. The SKB is then queued into the transmit queue of the Real Device Interface via a call to dev_queue_xmit().

The only changes in the Linux kernel required to support vlanctl interfaces are the addition of the bcm_vlan_handle_frame_hook function pointer in netif_receive_skb(), and the definition of the IFF_BCM_VLAN private flag in if.h.

Debugging Tip

A good way to review the rules on an interface is to dump all the rules that are known on the interface to see if the order and precedence make sense and all the filters and actions desired are in place.

Here is an code example snippet to dump all the rules on an interface:

```
vlanctl --if eth4 --rx --tags 0 --show-table
vlanctl --if eth4 --rx --tags 1 --show-table
vlanctl --if eth4 --rx --tags 2 --show-table
vlanctl --if eth4 --tx --tags 0 --show-table
vlanctl --if eth4 --tx --tags 1 --show-table
vlanctl --if eth4 --tx --tags 2 --show-table
```

For instance, to debug the rule created after an OMCI configuration, use the following:

```
vlanctl --if gpon0 --tx --tags 0 --show-table
vlanctl --if gpon0 --rx --tags 0 --show-table
vlanctl --if gpon0 --tx --tags 1 --show-table
vlanctl --if gpon0 --rx --tags 1 --show-table
vlanctl --if gpondef --tx --tags 0 --show-table
vlanctl --if gpondef --rx --tags 0 --show-table
vlanctl --if gpondef --tx --tags 1 --show-table
vlanctl --if gpondef --rx --tags 1 --show-table
vlanctl --if gpon0.0 --rx --tags 0 --show-table
vlanctl --if gpon0.0 --tx --tags 0 --show-table
vlanctl --if gpon0.0 --rx --tags 1 --show-table
vlanctl --if gpon0.0 --tx --tags 1 --show-table
vlanctl --if veip0 --rx --tags 0 --show-table
vlanctl --if veip0 --tx --tags 0 --show-table
vlanctl --if veip0 --rx --tags 1 --show-table
vlanctl --if veip0 --tx --tags 1 --show-table
```

Connecting everything®

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

Phone: 949-926-5000
Fax: 949-926-5203
E-mail: info@broadcom.com
Web: www.broadcom.com

VLAN-AN100-R June 30, 2015