ef_vi User Guide

SF-114063-CD , Issue 5

2017/05/25 15:51:40

Solarflare Communications Inc

ef_vi User Guide

Copyright © 2017 SOLARFLARE Communications, Inc. All rights reserved.

The software and hardware as applicable (the “Product”) described in this document, and this document, are protected by copyright laws, patents and other intellectual property laws and international treaties. The Product described in this document is provided pursuant to a license agreement, evaluation agreement and/or non-disclosure agreement. The Product may be used only in accordance with the terms of such agreement. The software as applicable may be copied only in accordance with the terms of such agreement.

Onload is licensed under the GNU General Public License (Version 2, June 1991). See the LICENSE file in the distribution for details. The Onload Extensions Stub Library is Copyright licensed under the BSD 2-Clause License.

Onload contains algorithms and uses hardware interface techniques which are subject to Solarflare Communications Inc patent applications. Parties interested in licensing Solarflare’s IP are encouraged to contact Solarflare’s Intellectual Property Licensing Group at:

Director of Intellectual Property Licensing
Intellectual Property Licensing Group
Solarflare Communications Inc,// 7505 Irvine Center Drive
Suite 100
Irvine, California 92618

You will not disclose to a third party the results of any performance tests carried out using Onload or EnterpriseOnload without the prior written consent of Solarflare.

The furnishing of this document to you does not give you any rights or licenses, express or implied, by estoppel or otherwise, with respect to any such Product, or any copyrights, patents or other intellectual property rights covering such Product, and this document does not contain or represent any commitment of any kind on the part of SOLARFLARE Communications, Inc. or its affiliates.

The only warranties granted by SOLARFLARE Communications, Inc. or its affiliates in connection with the Product described in this document are those expressly set forth in the license agreement, evaluation agreement and/or non-disclosure agreement pursuant to which the Product is provided. EXCEPT AS EXPRESSLY SET FORTH IN SUCH AGREEMENT, NEITHER SOLARFLARE COMMUNICATIONS, INC. NOR ITS AFFILIATES MAKE ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND (EXPRESS OR IMPLIED) REGARDING THE PRODUCT OR THIS DOCUMENTATION AND HEREBY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT, AND ANY WARRANTIES THAT MAY ARISE FROM COURSE OF DEALING, COURSE OF PERFORMANCE OR USAGE OF TRADE. Unless otherwise expressly set forth in such agreement, to the extent allowed by applicable law (a) in no event shall SOLARFLARE Communications, Inc. or its affiliates have any liability under any legal theory for any loss of revenues or profits, loss of use or data, or business interruptions, or for any indirect, special, incidental or consequential damages, even if advised of the possibility of such damages; and (b) the total liability of SOLARFLARE Communications, Inc. or its affiliates arising from or relating to such agreement or the use of this document shall not exceed the amount received by SOLARFLARE Communications, Inc. or its affiliates for that copy of the Product or this document which is the subject of such liability.

The Product is not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

SF-114063-CD

Last Revised: 52017

Issue 5

Contents

1 ef_vi **1**

1.1 Introduction 1

2 Overview **3**

2.1 Capabilities 3

2.2 Flexibility 3

2.3 Scalability 4

2.4 Use cases 4

2.4.1 Sockets acceleration 4

2.4.2 Packet capture 4

2.4.3 Packet replay 4

2.4.4 Application as an end-station 4

2.4.5 Software defined bridging, switching and routing 5

3 Concepts **7**

3.1 Virtual Interface 7

3.1.1 Virtual Interface Set. 8

3.1.2 Event queue 8

3.1.3 Transmit descriptor ring 8

3.1.4 Receive descriptor ring 8

3.2 Protection Domain 9

3.3 Memory Region 9

3.4 Packet Buffer 9

3.4.1 Jumbo Packets 10

3.4.2 Packet Buffer Descriptor 10

3.5 Programmed I/O 10

3.6 Filters 10

3.6.1 Multiple Filters 11

3.7 Virtual LANs 11

3.8 TX Alternatives 11

 Copyright © Solarflare Communications 2017

4 Example Applications 13
4.1 eflatency 13
4.1.1 Usage 14
4.2 efsend 14
4.3 efsink 14
4.3.1 Usage 15
4.4 efforward 15
4.5 efsend_timestamping 15
4.6 efsend_pio 15
4.7 efsink_packed 16
4.8 efforward_packed 16
4.9 efrss 16
4.10 efdelegated_client 16
4.10.1 Usage 16
4.11 efdelegated_server 17
4.11.1 Usage 17
4.12 Building the Example Applications 17

5 Using ef_vi 19
5.1 Components 19
5.2 Compiling and Linking 19
5.3 Setup 20
5.3.1 Using Virtual Interface Sets. 21
5.4 Creating packet buffers 21
5.5 Transmitting Packets 21
5.5.1 Transmitting Jumbo Frames 22
5.5.2 Programmed I/O 22
5.5.3 TX Alternatives 23
5.6 Handling Events 23
5.6.1 Blocking on a file descriptor 24

Copyright © Solarflare Communications 2017

5.7 Receiving packets 24
5.7.1 Finding the Packet Data 25
5.7.2 Receiving Jumbo Packets 25
5.8 Adding Filters 25
5.9 Filters available per Firmware Variant 27
5.10 Freeing Resources 29
5.11 Design Considerations 30
5.11.1 Interrupts 30
5.11.2 Thread Safety 30
5.11.3 Packet Buffer Addressing 30
5.11.4 Virtual machines 31
5.12 Known Limitations 31
5.12.1 Timestamping 31
5.12.2 Minimum Fill Level 31
5.13 Example 32

6 Worked Example **33**
6.1 Setup 33
6.2 Creating Packet buffers 34
6.3 Adding Filters 34
6.4 Receiving packets 35
6.5 Handling Events 35
6.6 Transmitting packets 36

7 Data Structure Index **37**
7.1 Data Structures 37

8 File Index **39**
8.1 File List 39

 Copyright © Solarflare Communications 2017

9 Data Structure Documentation **41**

9.1 ef_event Union Reference 41
9.1.1 Detailed Description 43
9.1.2 Field Documentation 43
9.1.2.1 generic 43
9.1.2.2 rx 43
9.1.2.3 rx_discard 43
9.1.2.4 rx_multi 43
9.1.2.5 rx_multi_discard 43
9.1.2.6 rx_no_desc_trunc 44
9.1.2.7 rx_packed_stream 44
9.1.2.8 sw 44
9.1.2.9 tx 44
9.1.2.10 tx_alt 44
9.1.2.11 tx_error 44
9.1.2.12 tx_timestamp 44
9.2 ef_eventq_state Struct Reference 45
9.2.1 Detailed Description 45
9.2.2 Field Documentation 45
9.2.2.1 evq_ptr 45
9.2.2.2 sync_flags 45
9.2.2.3 sync_timestamp_major 46
9.2.2.4 sync_timestamp_minimum 46
9.2.2.5 sync_timestamp_minor 46
9.2.2.6 sync_timestamp_synchronised 46
9.3 ef_filter_cookie Struct Reference 46
9.3.1 Detailed Description 47
9.3.2 Field Documentation 47
9.3.2.1 filter_id 47

Copyright © Solarflare Communications 2017

9.3.2.2 filter_type 47
9.4 ef_filter_spec Struct Reference 47
9.4.1 Detailed Description 48
9.4.2 Field Documentation 48
9.4.2.1 data 48
9.4.2.2 flags 48
9.4.2.3 type 48
9.5 ef_iovec Struct Reference 49
9.5.1 Detailed Description 49
9.5.2 Member Function Documentation 49
9.5.2.1 EF_VI_ALIGN() 49
9.5.3 Field Documentation 49
9.5.3.1 iov_len 49
9.6 ef_memreg Struct Reference 50
9.6.1 Detailed Description 50
9.6.2 Field Documentation 50
9.6.2.1 mr_dma_addrs 50
9.6.2.2 mr_dma_addrs_base 50
9.7 ef_packed_stream_packet Struct Reference 51
9.7.1 Detailed Description 51
9.7.2 Field Documentation 51
9.7.2.1 ps_cap_len 51
9.7.2.2 ps_flags 51
9.7.2.3 ps_next_offset 52
9.7.2.4 ps_orig_len 52
9.7.2.5 ps_pkt_start_offset 52
9.7.2.6 ps_ts_nsec 52
9.7.2.7 ps_ts_sec 52
9.8 ef_packed_stream_params Struct Reference 53

 Copyright © Solarflare Communications 2017

9.8.1 Detailed Description 53
9.8.2 Field Documentation 53
9.8.2.1 psp_buffer_align 53
9.8.2.2 psp_buffer_size 53
9.8.2.3 psp_max_usable_buffers 54
9.8.2.4 psp_start_offset 54
9.9 ef_pd Struct Reference 54
9.9.1 Detailed Description 54
9.9.2 Field Documentation 55
9.9.2.1 pd_cluster_dh 55
9.9.2.2 pd_cluster_name 55
9.9.2.3 pd_cluster_sock 55
9.9.2.4 pd_cluster_viset_index 55
9.9.2.5 pd_cluster_viset_resource_id 56
9.9.2.6 pd_flags 56
9.9.2.7 pd_intf_name 56
9.9.2.8 pd_resource_id 56
9.10 ef_pio Struct Reference 56
9.10.1 Detailed Description 57
9.10.2 Field Documentation 57
9.10.2.1 pio_buffer 57
9.10.2.2 pio_io 57
9.10.2.3 pio_len 57
9.10.2.4 pio_resource_id 58
9.11 ef_vi Struct Reference 58
9.11.1 Detailed Description 59
9.11.2 Field Documentation 59
9.11.2.1 ep_state 59
9.11.2.2 evq_base 59

 Copyright © Solarflare Communications 2017

9.11.2.3 evq_mask 60
9.11.2.4 inited 60
9.11.2.5 io 60
9.11.2.6 linked_pio 60
9.11.2.7 nic_type 60
9.11.2.8 ops 61
9.11.2.9 rx_buffer_len 61
9.11.2.10 rx_discard_mask 61
9.11.2.11 rx_prefix_len 61
9.11.2.12 rx_ts_correction 61
9.11.2.13 timer_quantum_ns 62
9.11.2.14 tx_alt_hw2id 62
9.11.2.15 tx_alt_id2hw 62
9.11.2.16 tx_alt_num 62
9.11.2.17 tx_push_thresh 62
9.11.2.18 tx_ts_correction_ns 63
9.11.2.19 vi_clustered 63
9.11.2.20 vi_flags 63
9.11.2.21 vi_i 63
9.11.2.22 vi_io_mmap_bytes 63
9.11.2.23 vi_io_mmap_ptr 64
9.11.2.24 vi_is_normal 64
9.11.2.25 vi_is_packed_stream 64
9.11.2.26 vi_mem_mmap_bytes 64
9.11.2.27 vi_mem_mmap_ptr 64
9.11.2.28 vi_out_flags 65
9.11.2.29 vi_ps_buf_size 65
9.11.2.30 vi_qs 65
9.11.2.31 vi_qs_n 65

Copyright © Solarflare Communications 2017

9.11.2.32 vi_resource_id 65
9.11.2.33 vi_rxq 66
9.11.2.34 vi_stats 66
9.11.2.35 vi_txq 66
9.12 ef_vi_layout_entry Struct Reference 66
9.12.1 Detailed Description 67
9.12.2 Field Documentation 67
9.12.2.1 evle_description 67
9.12.2.2 evle_offset 67
9.12.2.3 evle_type 67
9.13 ef_vi_nic_type Struct Reference 67
9.13.1 Detailed Description 68
9.13.2 Field Documentation 68
9.13.2.1 arch 68
9.13.2.2 revision 68
9.13.2.3 variant 68
9.14 ef_vi_rxq Struct Reference 69
9.14.1 Detailed Description 69
9.14.2 Field Documentation 69
9.14.2.1 descriptors 69
9.14.2.2 ids 69
9.14.2.3 mask 70
9.15 ef_vi_rxq_state Struct Reference 70
9.15.1 Detailed Description 70
9.15.2 Field Documentation 70
9.15.2.1 added 70
9.15.2.2 bytes_acc 71
9.15.2.3 in_jumbo 71
9.15.2.4 last_desc_i 71

Copyright © Solarflare Communications 2017

9.15.2.5 posted . . . 71
9.15.2.6 removed . . . 71
9.15.2.7 rx_ps_credit_avail . . . 72
9.16 ef_vi_set Struct Reference . . . 72
9.16.1 Detailed Description . . . 72
9.16.2 Field Documentation . . . 72
9.16.2.1 vis_pd . . . 72
9.16.2.2 vis_res_id . . . 73
9.17 ef_vi_state Struct Reference . . . 73
9.17.1 Detailed Description . . . 73
9.17.2 Field Documentation . . . 73
9.17.2.1 evq . . . 73
9.17.2.2 rxq . . . 74
9.17.2.3 txq . . . 74
9.18 ef_vi_stats Struct Reference . . . 74
9.18.1 Detailed Description . . . 74
9.18.2 Field Documentation . . . 74
9.18.2.1 evq_gap . . . 75
9.18.2.2 rx_ev_bad_desc_i . . . 75
9.18.2.3 rx_ev_bad_q_label . . . 75
9.18.2.4 rx_ev_lost . . . 75
9.19 ef_vi_stats_field_layout Struct Reference . . . 75
9.19.1 Detailed Description . . . 76
9.19.2 Field Documentation . . . 76
9.19.2.1 evsfl_name . . . 76
9.19.2.2 evsfl_offset . . . 76
9.19.2.3 evsfl_size . . . 76
9.20 ef_vi_stats_layout Struct Reference . . . 77
9.20.1 Detailed Description . . . 77

9.20.2 Field Documentation 77
9.20.2.1 evsl_data_size 77
9.20.2.2 evsl_fields 77
9.20.2.3 evsl_fields_num 78
9.21 ef_vi_transmit_alt_overhead Struct Reference 78
9.21.1 Detailed Description 78
9.21.2 Field Documentation 78
9.21.2.1 mask 78
9.21.2.2 post_round 79
9.21.2.3 pre_round 79
9.22 ef_vi_txq Struct Reference 79
9.22.1 Detailed Description 79
9.22.2 Field Documentation 79
9.22.2.1 descriptors 80
9.22.2.2 ids 80
9.22.2.3 mask 80
9.23 ef_vi_txq_state Struct Reference 80
9.23.1 Detailed Description 81
9.23.2 Field Documentation 81
9.23.2.1 added 81
9.23.2.2 previous 81
9.23.2.3 removed 81
9.23.2.4 ts_nsec 82
9.24 ef_vi::ops Struct Reference 82
9.24.1 Detailed Description 82
9.24.2 Field Documentation 83
9.24.2.1 eventq_poll 83
9.24.2.2 eventq_prime 83
9.24.2.3 eventq_timer_clear 83

Copyright © Solarflare Communications 2017

9.24.2.4 eventq_timer_prime . . . 83
9.24.2.5 eventq_timer_run . . . 84
9.24.2.6 eventq_timer_zero . . . 84
9.24.2.7 receive_init . . . 84
9.24.2.8 receive_push . . . 84
9.24.2.9 transmit . . . 84
9.24.2.10 transmit_alt_discard . . . 85
9.24.2.11 transmit_alt_go . . . 85
9.24.2.12 transmit_alt_select . . . 85
9.24.2.13 transmit_alt_select_default . . . 85
9.24.2.14 transmit_alt_stop . . . 85
9.24.2.15 transmit_copy_pio . . . 86
9.24.2.16 transmit_copy_pio_warm . . . 86
9.24.2.17 transmit_pio . . . 86
9.24.2.18 transmit_pio_warm . . . 86
9.24.2.19 transmit_push . . . 86
9.24.2.20 transmitv . . . 87
9.24.2.21 transmitv_init . . . 87

10 File Documentation **89**

10.1 000_main.dox File Reference . . . 89
10.1.1 Detailed Description . . . 89
10.2 010_overview.dox File Reference . . . 89
10.2.1 Detailed Description . . . 90
10.3 020_concepts.dox File Reference . . . 90
10.3.1 Detailed Description . . . 90
10.4 030_apps.dox File Reference . . . 90
10.4.1 Detailed Description . . . 91
10.5 040_using.dox File Reference . . . 91
10.5.1 Detailed Description . . . 91

 Copyright © Solarflare Communications 2017

10.6 050_examples.dox File Reference 91
10.6.1 Detailed Description 92
10.7 base.h File Reference 92
10.7.1 Detailed Description 93
10.7.2 Function Documentation 93
10.7.2.1 ef_driver_close() 93
10.7.2.2 ef_driver_open() 94
10.7.2.3 ef_eventq_wait() 94
10.8 capabilities.h File Reference 94
10.8.1 Detailed Description 95
10.8.2 Enumeration Type Documentation 96
10.8.2.1 ef_vi_capability 96
10.8.3 Function Documentation 97
10.8.3.1 ef_vi_capabilities_get() 97
10.8.3.2 ef_vi_capabilities_max() 98
10.8.3.3 ef_vi_capabilities_name() 98
10.9 ef_vi.h File Reference 98
10.9.1 Detailed Description 105
10.9.2 Macro Definition Documentation 105
10.9.2.1 EF_EVENT_RX_MULTI_CONT 105
10.9.2.2 EF_EVENT_RX_MULTI_SOP 105
10.9.2.3 EF_EVENT_RX_PS_NEXT_BUFFER 106
10.9.2.4 ef_eventq_poll 106
10.9.2.5 ef_eventq_prime 106
10.9.2.6 ef_vi_receive_init 107
10.9.2.7 ef_vi_receive_push 107
10.9.2.8 ef_vi_transmit 108
10.9.2.9 ef_vi_transmit_alt_discard 109
10.9.2.10 ef_vi_transmit_alt_go 109

 Copyright © Solarflare Communications 2017

10.9.2.11 ef_vi_transmit_alt_select 110
10.9.2.12 ef_vi_transmit_alt_select_normal 110
10.9.2.13 ef_vi_transmit_alt_stop 111
10.9.2.14 ef_vi_transmit_copy_pio 111
10.9.2.15 ef_vi_transmit_copy_pio_warm 112
10.9.2.16 ef_vi_transmit_pio 113
10.9.2.17 ef_vi_transmit_pio_warm 114
10.9.2.18 ef_vi_transmit_push 114
10.9.2.19 ef_vi_transmitv 115
10.9.2.20 ef_vi_transmitv_init 116
10.9.3 Typedef Documentation 117
10.9.3.1 ef_request_id 117
10.9.3.2 ef_vi 117
10.9.4 Enumeration Type Documentation 117
10.9.4.1 anonymous enum 117
10.9.4.2 anonymous enum 118
10.9.4.3 anonymous enum 118
10.9.4.4 ef_vi_arch 118
10.9.4.5 ef_vi_flags 119
10.9.4.6 ef_vi_layout_type 120
10.9.4.7 ef_vi_out_flags 120
10.9.4.8 ef_vi_rx_discard_err_flags 120
10.9.5 Function Documentation 121
10.9.5.1 ef_eventq_capacity() 121
10.9.5.2 ef_eventq_current() 121
10.9.5.3 ef_eventq_has_event() 122
10.9.5.4 ef_eventq_has_many_events() 122
10.9.5.5 ef_vi_driver_interface_str() 123
10.9.5.6 ef_vi_flags() 123

Copyright © Solarflare Communications 2017

10.9.5.7 ef_vi_instance() 124
10.9.5.8 ef_vi_receive_buffer_len() 124
10.9.5.9 ef_vi_receive_capacity() 124
10.9.5.10 ef_vi_receive_fill_level() 125
10.9.5.11 ef_vi_receive_get_bytes() 125
10.9.5.12 ef_vi_receive_get_timestamp() 126
10.9.5.13 ef_vi_receive_get_timestamp_with_sync_flags() 127
10.9.5.14 ef_vi_receive_post() 128
10.9.5.15 ef_vi_receive_prefix_len() 128
10.9.5.16 ef_vi_receive_query_layout() 129
10.9.5.17 ef_vi_receive_set_buffer_len() 129
10.9.5.18 ef_vi_receive_set_discards() 130
10.9.5.19 ef_vi_receive_space() 130
10.9.5.20 ef_vi_receive_unbundle() 131
10.9.5.21 ef_vi_resource_id() 131
10.9.5.22 ef_vi_set_tx_push_threshold() 132
10.9.5.23 ef_vi_transmit_alt_num_ids() 132
10.9.5.24 ef_vi_transmit_alt_query_overhead() 133
10.9.5.25 ef_vi_transmit_alt_usage() 133
10.9.5.26 ef_vi_transmit_capacity() 134
10.9.5.27 ef_vi_transmit_fill_level() 134
10.9.5.28 ef_vi_transmit_init() 134
10.9.5.29 ef_vi_transmit_init_undo() 135
10.9.5.30 ef_vi_transmit_space() 136
10.9.5.31 ef_vi_transmit_unbundle() 137
10.9.5.32 ef_vi_version_str() 137
10.10 memreg.h File Reference 138
10.10.1 Detailed Description 138
10.10.2 Function Documentation 139

 Copyright © Solarflare Communications 2017

10.10.2.1 ef_memreg_alloc() . . . 139
10.10.2.2 ef_memreg_dma_addr() . . . 140
10.10.2.3 ef_memreg_free() . . . 140
10.11 packedstream.h File Reference . . . 141
10.11.1 Detailed Description . . . 141
10.11.2 Macro Definition Documentation . . . 142
10.11.2.1 EF_VI_PS_FLAG_BAD_IP_CSUM . . . 142
10.11.3 Function Documentation . . . 142
10.11.3.1 ef_vi_packed_stream_get_params() . . . 142
10.11.3.2 ef_vi_packed_stream_unbundle() . . . 142
10.12 pd.h File Reference . . . 143
10.12.1 Detailed Description . . . 144
10.12.2 Enumeration Type Documentation . . . 144
10.12.2.1 ef_pd_flags . . . 144
10.12.3 Function Documentation . . . 145
10.12.3.1 ef_pd_alloc() . . . 145
10.12.3.2 ef_pd_alloc_by_name() . . . 146
10.12.3.3 ef_pd_alloc_with_vport() . . . 146
10.12.3.4 ef_pd_free() . . . 147
10.12.3.5 ef_pd_interface_name() . . . 147
10.13 pio.h File Reference . . . 148
10.13.1 Detailed Description . . . 148
10.13.2 Function Documentation . . . 149
10.13.2.1 ef_pio_free() . . . 149
10.13.2.2 ef_pio_link_vi() . . . 149
10.13.2.3 ef_pio_memcpy() . . . 150
10.13.2.4 ef_pio_unlink_vi() . . . 150
10.13.2.5 ef_vi_get_pio_size() . . . 151
10.14 timer.h File Reference . . . 151

 Copyright © Solarflare Communications 2017

10.14.1 Detailed Description . . . 152
10.14.2 Macro Definition Documentation . . . 152
10.14.2.1 ef_eventq_timer_clear . . . 152
10.14.2.2 ef_eventq_timer_prime . . . 153
10.14.2.3 ef_eventq_timer_run . . . 153
10.14.2.4 ef_eventq_timer_zero . . . 154
10.15vi.h File Reference . . . 155
10.15.1 Detailed Description . . . 157
10.15.2 Enumeration Type Documentation . . . 157
10.15.2.1 anonymous enum . . . 157
10.15.2.2 ef_filter_flags . . . 158
10.15.3 Function Documentation . . . 158
10.15.3.1 ef_eventq_put() . . . 158
10.15.3.2 ef_filter_spec_init() . . . 159
10.15.3.3 ef_filter_spec_set_block_kernel() . . . 159
10.15.3.4 ef_filter_spec_set_block_kernel_multicast() . . . 160
10.15.3.5 ef_filter_spec_set_block_kernel_unicast() . . . 160
10.15.3.6 ef_filter_spec_set_eth_local() . . . 161
10.15.3.7 ef_filter_spec_set_eth_type() . . . 161
10.15.3.8 ef_filter_spec_set_ip4_full() . . . 162
10.15.3.9 ef_filter_spec_set_ip4_local() . . . 163
10.15.3.10ef_filter_spec_set_ip6_full() . . . 163
10.15.3.11ef_filter_spec_set_ip6_local() . . . 164
10.15.3.12ef_filter_spec_set_ip_proto() . . . 165
10.15.3.13ef_filter_spec_set_multicast_all() . . . 166
10.15.3.14ef_filter_spec_set_multicast_mismatch() . . . 167
10.15.3.15ef_filter_spec_set_port_sniff() . . . 168
10.15.3.16ef_filter_spec_set_tx_port_sniff() . . . 168
10.15.3.17ef_filter_spec_set_unicast_all() . . . 169

 Copyright © Solarflare Communications 2017

10.15.3.18 ef_filter_spec_set_unicast_mismatch() 169
10.15.3.19 ef_filter_spec_set_vlan() 170
10.15.3.20 ef_vi_alloc_from_pd() 171
10.15.3.21 ef_vi_alloc_from_set() 172
10.15.3.22 ef_vi_filter_add() 173
10.15.3.23 ef_vi_filter_del() 174
10.15.3.24 ef_vi_flush() 174
10.15.3.25 ef_vi_free() 175
10.15.3.26 ef_vi_get_mac() 175
10.15.3.27 ef_vi_mtu() 176
10.15.3.28 ef_vi_pace() 176
10.15.3.29 ef_vi_prime() 177
10.15.3.30 ef_vi_set_alloc_from_pd() 177
10.15.3.31 ef_vi_set_filter_add() 178
10.15.3.32 ef_vi_set_filter_del() 178
10.15.3.33 ef_vi_set_free() 180
10.15.3.34 ef_vi_stats_query() 180
10.15.3.35 ef_vi_stats_query_layout() 181
10.15.3.36 ef_vi_transmit_alt_alloc() 181
10.15.3.37 ef_vi_transmit_alt_free() 182
10.15.3.38 ef_vi_transmit_alt_query_buffering() 182

Index **185**

Chapter 1

ef_vi

Solarflare's ef_vi API is a flexible interface for passing Ethernet frames between applications and the network. It is the internal API used by Onload for sending and receiving packets.

1.1 Introduction

ef_vi grants an application direct access to the Solarflare network adapter datapath to deliver lower latency and reduced per message processing overheads. It can be used directly by applications that want the very lowest latency send and receive API, and that do not require a POSIX socket interface.

The key features of ef_vi are:

- **User-space:** Ef_vi can be used by unprivileged user-space applications.
- **Kernel bypass**: Data path operations do not require system calls.
- **Low CPU overhead**: Data path operations consume very few CPU cycles.
- **Low latency**: Suitable for ultra-low latency applications.
- **High packet rates**: Supports millions of packets per second per core.
- **Zero-copy**: Particularly efficient for filtering and forwarding applications.
- **Flexibility**: Supports many use cases (see Use cases).
- **Redistributable**: ef_vi is free software distributed under a LGPL license.

Each ef_vi instance provides a Virtual Interface for an application to use:

- Each virtual interface provides a TX channel for passing packets to the adapter. Packets may be transmitted onto the wire, or looped-back in the adapter for delivery back into the host, or both.
- Each virtual interface also provides an RX channel for receiving packets from the adapter. These can be packets received from the wire, or looped-back from the TX path.

 Copyright © Solarflare Communications 2017

 Copyright © Solarflare Communications 2017

Chapter 2

Overview

This part of the documentation gives an overview of ef_vi and how it is often used.

2.1 Capabilities

Ef_vi is a low level OSI level 2 interface which sends and receives raw Ethernet frames. It is essentially a thin wrapper around the VNIC (virtual network interface controller) interface offered by the network adapters. It exposes, and can be used with, many of the advanced capabilities of Solarflare network adapters, including:

- Checksum offloads
- Hardware time stamping (RX and TX)
- Switching functions, including:
  - multicast replication
  - loopback
  - VLAN tag insert/strip
- Load spreading by hashing (also known as receive-side scaling) and flow steering
- Data path sniffing
- PIO mode for ultra-low latency
- Scatter gather transmit and receive
- PCI pass-through and SR-IOV for virtualised environments.

But because the ef_vi API operates at this low level, any application using it must implement the higher layer protocols itself, and also deal with any exceptions or other unusual conditions.

2.2 Flexibility

A key advantage of ef_vi when compared to other similar technologies is the flexibility it offers. Each ef_vi instance can be thought of as a virtual port on the network adapter's switch. Ef_vi can be used to handle all packets associated with a physical port, or just a subset. This means that ef_vi can be used in parallel with the standard Linux kernel network stack and other acceleration technologies such as Solarflare's OpenOnload sockets acceleration. For example, a trading application might use ef_vi to receive UDP market data, but use Onload sockets to make TCP trades.

 Copyright © Solarflare Communications 2017

2.3 Scalability

Ef_vi is also very scalable. Each physical network port on Solarflare's network adapters supports up to 1024 VNIC interfaces. There can be many independent channels between software and the adapter, which can be used to spread load over many cores, or to accelerate large numbers of virtual machines and applications simultaneously.

2.4 Use cases

This section gives some examples of how ef_vi can and is being used:

2.4.1 Sockets acceleration

Ef_vi can be used to replace the BSD sockets API, or another API, for sending and receiving streams of traffic. A common example is handling multicast UDP datagrams in electronic trading systems, where low latency is needed and message rates can be very high.

In this scenario the application establishes an ef_vi instance, and specifies which packets it would like to receive via this path. For example, an application can select UDP packets with a given destination IP address and port number. All other packets arriving at the network interface continue to be delivered to the regular driver in the kernel stack via a separate path, so only the packets that need to be accelerated are handled by ef_vi.

Applications can create multiple ef_vi instances if needed to handle different streams of packets, or to spread load over multiple threads. If transmitting threads each have their own ef_vi instance then they can transmit packets concurrently without interlocking and without sharing state. This improves efficiency considerably.

2.4.2 Packet capture

Solarflare's SolarCapture software is built on top of the ef_vi API. Like traditional capture APIs ef_vi can be used to capture all of the packets arriving at a network port, or a subset. Ef_vi can capture traffic from a 10 gigabit link at line rate at all packet sizes with a single core, and can provide a hardware timestamp for each captured packet.

In "sniffing" mode an ef_vi instance receives a copy of packets transmitted and/or received by other applications on the host.

2.4.3 Packet replay

Ef_vi can transmit packets from host memory to the network at very high rates, so can be used to construct high performance packet replay applications. (This is another feature of the SolarCapture software).

2.4.4 Application as an end-station

Ef_vi can select packets by destination MAC address, which allows an application to behave as if it were a separate end-station on the Ethernet network. This can be used to implement arbitrary protocols over Ethernet, or to develop applications that simulate behaviour of an end station for benchmarking or test purposes.

Other applications and virtual machines on the host can use the adapter at the same time, via kernel stack, via ef_vi or via OpenOnload. The application simulating an end-station can communicate with those applications via the adapter, as well as with other remote applications over the physical network.

 Copyright © Solarflare Communications 2017

2.4.5 Software defined bridging, switching and routing

Ef_vi is ideally suited to applications that forward packets between physical or virtual network ports. Zero-copy makes the forwarding path very efficient, and forwarding can be achieved with very low latency.

An ef_vi instance can be configured to receive "mismatching" packets. That is, all packets not wanted by other applications or virtual machines on the same host. This makes it possible to forward packets to another network segment without knowing the MAC addresses involved, and without cutting the host OS off from the network.

Applications include: High performance firewall applications, network monitoring, intrusion detection, and custom switching and routing. Packets can be forwarded between physical ports and/or between virtual ports. (Virtual ports are logical network ports associated with virtual machines using PCI pass-through).

Ef_vi is particularly well suited to Network Functions Virtualization (NFV).

 Copyright © Solarflare Communications 2017

Copyright © Solarflare Communications 2017

Chapter 3

Concepts

This part of the documentation describes the concepts involved in ef_vi.

3.1 Virtual Interface

Each ef_vi instance provides a *virtual interface* to the network adapter.

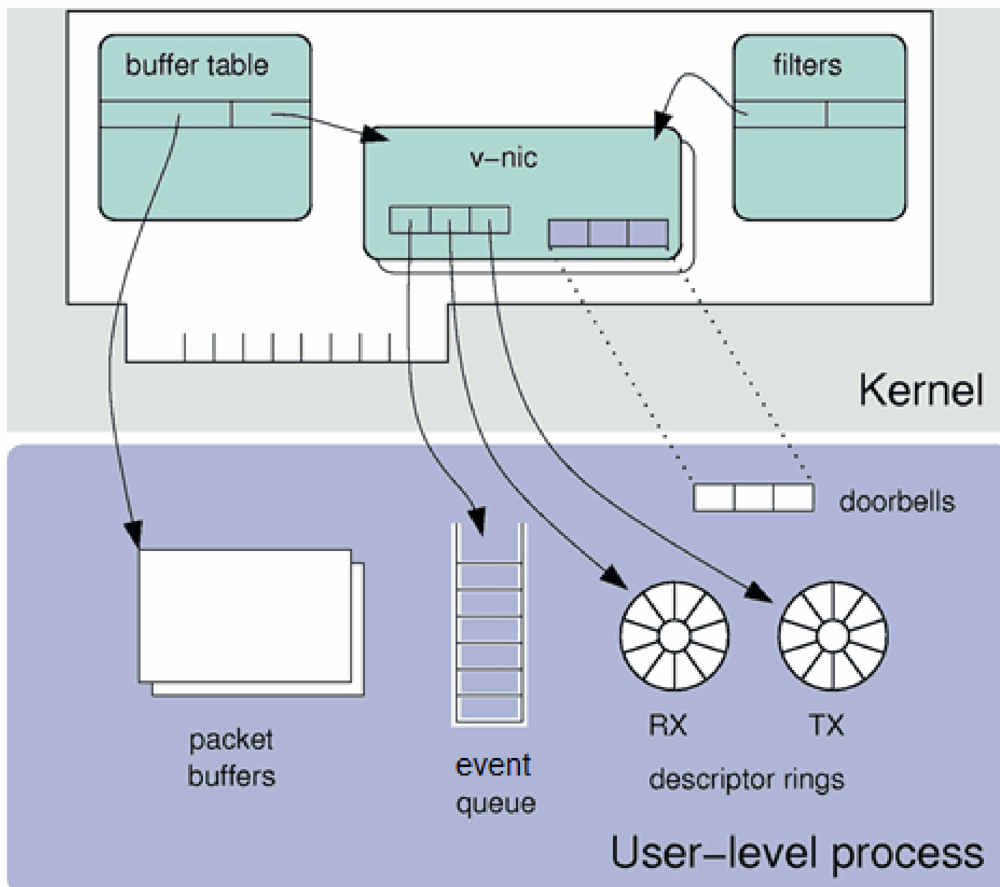

Figure 3.1 Virtual Interface Components

 Copyright © Solarflare Communications 2017

A virtual interface includes the following components:

- an Event queue
- a Transmit descriptor ring
- a Receive descriptor ring.

A virtual interface can be allocated one of each of these components, but if the event queue is omitted an alternative virtual interface that has an event queue must be specified.

A virtual interface also has a few hardware resources:

- a doorbell register to inform the card that new RX buffers are available for it to use
- a doorbell register to inform the card that new TX buffers are ready for it to send
- some timers
- a share of an interrupt.

3.1.1 Virtual Interface Set.

A set of virtual interfaces can be created, to distribute load on the matching filters automatically, via *receive side scaling* (RSS).

Note

For this to be of use, multiple filters or a wildcard filter are required. A single stream filter will have the same RSS hash for all packets. See Filters.

3.1.2 Event queue

An *event queue* is a channel for passing information from the network adapter to the application. It is used to notify the application of events such as the arrival of packets.

3.1.3 Transmit descriptor ring

The *transmit descriptor ring* is used to pass packets from the application to the adapter. Each entry in the ring is a descriptor which references a buffer containing packet data. Each packet is described by one or more descriptors.

The transmission of packets proceeds in the background, and the adapter notifies the application when they have finished via the event queue.

3.1.4 Receive descriptor ring

The *receive descriptor ring* is used to pass packets from the adapter to the application. The application must pre-allocate buffers and post them to the receive descriptor ring. Each entry in the ring is a descriptor which references a 'free' buffer that the adapter can place a packet into.

When the adapter delivers a packet to an ef_vi instance, it copies the packet data into the next available receive buffer and notifies the application via the event queue. Large packets can be scattered over multiple receive buffers.

Copyright © Solarflare Communications 2017

3.2 Protection Domain

A *protection domain* identifies a separate address space for the DMA addresses passed to the adapter. It is used to protect multiple ef_vi applications from one another, or to allow them to share resources:

- Each Virtual Interface is associated with one protection domain.
- Multiple VIs can be associated with the same protection domain.
- Each Memory Region is registered with one or more protection domains.
- A memory region can only be used by a virtual interface that is in the same protection domain.
- Memory regions that are registered with multiple protection domains can be used as a shared resource, for example for zero-copy forwarding. See also Packet Buffer Addressing.

Note

Traditionally device drivers pass the physical addresses of memory buffers to I/O devices. This is usually acceptable because device drivers run within the privileged kernel, and so are isolated from untrusted user-space applications.
Applications using ef_vi cannot in general use unprotected physical addresses, because by manipulating those addresses prior to passing them to the adapter it would be possible to access memory and devices not otherwise accessible by the application. Protection domains are used to solve this problem.

3.3 Memory Region

Any *memory region* used for transmit or receive buffers must be *registered* using the ef_memreg interface. This ensures the memory region meets the requirements of ef_vi:

- The memory is pinned, so that it can't be swapped out to disk.
- The memory is mapped for DMA, so that the network adapter can access it. The adapter translates the DMA addresses provided by the application to I/O addresses used on the PCIe bus.
- The memory region is page-aligned, for performance.
- The size of the memory region is a multiple of the packet buffer size, so no memory is wasted.

3.4 Packet Buffer

A *packet buffer* is a memory allocations on the host which the card will read from when sending packets, or write to when receiving packets. They are usually 2KB in size.

Packets buffers are mapped by the card in such a way that only virtual interfaces in the same protection domain can access them, unless physical addressing mode is explicitly requested. (This feature can only be granted to a group of users by the root user setting an option on the driver.)

 Copyright © Solarflare Communications 2017

3.4.1 Jumbo Packets

Typically, some portion of a packet buffer will be used for meta-data, leaving enough space for a standard sized packet. On receive, large packets will be spread out over multiple packet buffers.

When sending, multiple buffers may be used either to accommodate larger sends, or for convenience (for example: splitting off a standard header that is common to multiple packets).

3.4.2 Packet Buffer Descriptor

Each packet buffer is referred to by a descriptor, which contains:

- a pointer
- an offset
- a length.

It is those descriptors which are actually placed onto the receive and transmit descriptor rings.

3.5 Programmed I/O

The ef_pio interface exposes a region of memory on the network adapter that can be used for low-latency sends. When using this interface packet data is pushed to the adapter using CPU instructions, instead of being pulled by the adapter using DMA. This reduces latency because it avoids the latency associated with a DMA read.

Applications can get even better latency by writing packet data to the adapter in advance, before the latency critical path. On the critical path the packet data can optionally be updated before being transmitted. This improves latency because it reduces the amount of data that needs to be passed to the adapter on the critical path.

3.6 Filters

Filters select which packets are delivered to a virtual interface. Packets that are not selected are ignored and allowed to pass on to the kernel.

Each filter specifies the characteristics of packets for selection. These characteristics are typically packet header fields, including Ethernet MAC address, VLAN tags, IP addresses and port numbers.

A selected packet can be:

- Stolen: the packet is delivered to the virtual interface, but not to the kernel stack.
- Replicated: a copy is delivered to the virtual interface, and might also be delivered to other consumers. Used for multicast packets.
- Sniffed: the packet is delivered to the virtual interface, and to the kernel stack.

Note

The set of header fields and filter modes that are available vary between adapter model and firmware variant.

Copyright © Solarflare Communications 2017

3.6.1 Multiple Filters

An ef_vi application can set multiple types of filters on the same virtual interface. Setting an invalid filter or combination of filters causes an error.

3.7 Virtual LANs

Ef_vi only has limited support for *Virtual LANs* (VLANs). This is because ef_vi operates at the Ethernet frame level, whereas VLANs are usually handled at a higher level:

- Received packets can be filtered by VLAN, but this requires a recent adapter running full feature firmware. There are also limitations on what other filters can simultaneously be used. For more details, see ef_filter_spec_set_vlan().
- Transmitted packets can have their VLAN set by adding the desired VLAN tag to the extended header. Unlike checksums, ef_vi does not provide an offload for this.

3.8 TX Alternatives

TX alternatives is a feature available on Solarflare SFN8000 series adapters to provide multiple alternative queues for transmission, that can be used to minimize latency. Different possible responses can be pushed through the TX path on the NIC, and held in different queues ready to transmit. When it is decided which response to transmit, the appropriate alternative queue is selected, and the queued packets are sent. Because the packets are already prepared, and are held close to the wire, latency is greatly reduced.

 Copyright © Solarflare Communications 2017

Chapter 4

Example Applications

Solarflare ef_vi comes with a range of example applications - including source code and make files. This is a quick guide to using them, both for testing ef_vi's effectiveness in an environment, and as starting points for developing applications.

Not all of these applications are available in the current Onload release; but source code is available on request.

Most of these applications have additional options to test physical addressing mode, or hardware timestamping. Run with "--help" to check this.

Application	Description
eflatency	Measure latency by pinging a simple message between two interfaces.
efsend	Send UDP packets on a specified interface.
efsink	Receive streams of packets on a single interface.
efforward	Forward packets between two interfaces without modification.
efsend_timestamping	Send UDP packets on a specified interface, with TX timestamping.
efsend_pio	Send UDP packets on a specified interface using Programmed I/O.
efsink_packed	Receive streams of packets on a single interface using packed streams.
efforward_packed	Forward packets between two interfaces without modification using packed stream mode for receive.
efrss	Forward packets between two interfaces without modification, spreading the load over multiple virtual interfaces and threads.
efdelegated_client	Delegate TCP sends to the ef_vi layer-2 API in order to get lower latency.
efdelegated_server	Server application used together with efdelegated_client.

4.1 eflatency

The eflatency application echoes a single packet back and forth repeatedly, measuring the round-trip time.

This is the most useful example application for testing lowest possible latency. It is not a very good sample for building an application, because:

- it uses only one filter

 Copyright © Solarflare Communications 2017

- it operates knowing that there is only ever a single packet on the wire, and so:
  - does not need to refill the rings
  - does not handle multiple event types.

4.1.1 Usage

Server: `eflatency pong` *`interface`*

Client: `eflatency ping` *`interface`*

where:

- *interface* is the multicast interface on the server or client machine (e.g. `eth0`)

There are various additional options. See the help text for details.

4.2 efsend

The efsend application sends UDP packets on a specified interface.

The application sends a UDP packet, waits for transmission of the packet to finish and then sends the next.

The number of packets sent, the size of the packet, the amount of time to wait between sends can be controlled.

4.3 efsink

The efsink application is a demonstration of capturing packets, and the flexibility of filters.

It supports all filter types that ef_vi supports. By default it just reports the amount of data captured, but it also demonstrates simple actions upon the packet data, with the option to hexdump incoming packets.

It is a very useful jumping off point as it shows:

- creation of a virtual interface
- creation and installation of filters
- polling the event queue.

 Copyright © Solarflare Communications 2017

4.3.1 Usage

To receive a single multicast group:

efsink _local interface_ udp:<multicast-addr>:port

To receive multiple multicast groups:

efsink _interface_ udp:<multicast-addr>:port udp:<multicast-group>:port

To receive all multicast traffic:

efsink _interface_ multicast-all

The efsink application does not send packets.

4.4 efforward

The efforward application listens for traffic on one interface and echoes it out of a second; and vice versa. It demonstrates a very simple high-performance bridge.

Some route configuration on the clients might be necessary to get this working, but it is a very simple example, and is very easy to start adding packet re-writing rules etc.

Although this is a viable starting point for a bridging application, a better option might be the SolarCapture API, which includes a more complete pre-made bridging application.

4.5 efsend_timestamping

The efsend_timestamping application sends UDP packets on a specified interface.

The application sends a UDP packet, waits for transmission of the packet to finish and then sends the next.

This application requests tx timestamping, allowing it to report the time each packet was transmitted.

The number of packets sent, the size of the packet, the amount of time to wait between sends can be controlled.

4.6 efsend_pio

The efsend_pio application that sends UDP packets on a specified interface.

Packet data is copied to the NIC's PIO buffer before being sent, which typically results in lower latency sends compared to accessing packet data stored on the host via DMA, which is the method used by the efsend sample app.

The application sends a UDP packet, waits for transmission of the packet to finish and then sends the next.

The number of packets sent, the size of the packet, the amount of time to wait between sends can be controlled.

4.7 efsink_packed

The efsink_packed application is a variant of efsink that demonstrates usage of the packed-stream firmware.

4.8 efforward_packed

The efforward_packed application is a variant of efforward that demonstrates usage of the packed-stream firmware.

4.9 efrss

The efrss application is a variant of efforward. It demonstrates automatically spreading the load over multiple threads, using a vi_set and RSS.

4.10 efdelegated_client

The efdelegated_client application demonstrates usage of OpenOnload's "Delegated Sends" feature. This API allows you to do delegate TCP sends for a particular socket to some other mechanism. For example, this sample uses the ef_vi layer-2 API in order to get lower latency than is possible with a normal send() call.

The API essentially boils down to first retrieving the packet headers, adding your own payload to form a raw packet, sending the packet and finally telling Onload what it was you sent so it can update the internal TCP state of the socket.

This sample application allows you to compare the performance of normal sends using the kernel stack, using OpenOnload and using ef_vi with the delegated sends API. It establishes a TCP connection to the server process, which starts sending UDP multicast messages. The client receives these messages, and replies to a subset of them with a TCP message. The server measures the latency from the UDP send to the TCP receive.

4.10.1 Usage

For normal socket-based sends, run as follows:

Server: `onload -p latency ./efdelegated_server` *`mcast-intf`* **Client:** `onload -p latency ./efdelegated_client` *`mcast-intf server`*

For "delegated" sends, run as follows:

Server: `onload -p latency ./efdelegated_server` *`mcast-intf`* **Client:** `onload -p latency ./efdelegated_client -d` *`mcast-intf server`*

where:

- *mcast-intf* is the multicast interface on the server or client machine (e.g. `eth0`)
- *server* is the IP address of the server machine (e.g. `192.168.0.10`)

There are various additional options. See the help text for details.

4.11 efdelegated_server

The efdelegated_server application is used together with efdelegated_client to measure the performance of OpenOnload's "Delegated Sends" feature.

This application is a very basic simulation of an exchange: It accepts a TCP connection from a client. It sends UDP multicast messages, and expects that the client will send a message over the TCP connection in response to a subset of the UDP messages. It measures the time taken from sending a UDP message to receiving the corresponding reply.

When used with OpenOnload this application uses hardware timestamps taken on the adapter so that the latency measured is very accurate. (Note that for 7000-series adapters this requires a Performance Monitor license).

4.11.1 Usage

See Usage for efdelegated_client.

4.12 Building the Example Applications

The ef_vi example applications are built along with the Onload installation and will be present in the `/Onload-<version>/build/gnu_x86_64/tests/ef_vi` subdirectory. In the build directory there will be `gnu`, `gnu_x86_64`, `x86_64_linux-<kernel version>` directories:

- files under the `gnu` directory are 32-bit (if these are built)
- files under the `gnu_x86_64` directory are 64-bit.

Source code files for the example applications exist in the `/Onload-<version>/src/tests/ef_vi` subdirectory.

After running the onload_install command, example applications exist in the `/Onload-<version>/build/gnu_x86_64/tests/ef_vi` subdirectory.

To rebuild the example applications you must have the `Onload-<version>/scripts` subdirectory in your path and use the following procedure:

```
[root@server01 Onload-<version>]# cd scripts/
[root@server01 scripts]# export PATH="$PWD:$PATH"
[root@server01 scripts]# cd ../build/gnu_x86_64/tests/ef_vi/
[root@server01 ef_vi]# make clean
[root@serverr01 ef_vi]# make
```

 Copyright © Solarflare Communications 2017

 Copyright © Solarflare Communications 2017

Chapter 5

Using ef_vi

This part of the documentation gives information on using ef_vi to write and build applications.

5.1 Components

All components required to build and link a user application with the Solarflare ef_vi API are distributed with Onload. When Onload is installed all required directories/files are located under the Onload distribution directory:

5.2 Compiling and Linking

Applications or libraries using ef_vi will need to include the header files in src/include/etherfabric/

The application will need to be linked with libciul1.a or libciul.so, which can be found under the "build" directory after running scripts/onload_build or scripts/onload_install.

If compiling your application against one version of Onload, and running on a system with a different version of Onload, some care is required. Onload currently preserves compatibility and provides a stable API between the ef_vi user-space and the kernel drivers, so that applications compiled using an older ef_vi library will work when run with newer drivers. Compatibility in the other direction (newer ef_vi libraries running with older drivers) is not guaranteed. Finally, Onload does not currently maintain compatibility between compiling against one version of the ef_vi libraries, and then running against another.

The simplest approach is to link statically to libciul, as this ensures that the version of the library used will match the one you have compiled against. If linking dynamically, it is recommended that you keep libciul.so and the application binary together. onload_install does not install libciul.so into system directories to avoid the installed version being used in place of the version you compiled against.

For those wishing to use ef_vi in combination with Onload there should be no problem linking statically to libciul and dynamically to the other libraries to allow the Onload intercepts to take effect. The ef_delegated example application does exactly this.

 Copyright © Solarflare Communications 2017

5.3 Setup

Applications requiring specific features can check the versions of software:

- use ef_vi_version_str() to get the version of ef_vi
- use ef_vi_driver_interface_str() to get the char driver interface required by this build of ef_vi.

Applications can also check for specific capabilities:

- use ef_vi_capabilities_get() to get the value of a given capability
- use ef_vi_capabilities_max() to get the number of available capabilities
- use ef_vi_capabilities_name() to get a human-readable string describing a given capability.

Users of ef_vi must do the following to setup:

1. Obtain a driver handle by calling ef_driver_open().
2. Allocate a protection domain by calling one of the following:
   - ef_pd_alloc()
   - ef_pd_alloc_by_name()
   - ef_pd_alloc_with_vport().
3. Allocate a virtual interface (VI), encapsulated by the type ef_vi, by calling ef_vi_alloc_from_pd().

```
/* Allocate a protection domain */
int ef_pd_alloc(ef_pd *pd,
                ef_driver_handle pd_dh,
                int ifindex,
                enum ef_pd_flags flags);

int ef_pd_alloc_by_name(ef_pd *pd,
                        ef_driver_handle pd_dh,
                        const char* cluster_or_intf_name,
                        enum ef_pd_flags flags);

/* Get the interface for a protection domain. */
const char* ef_pd_interface_name(ef_pd *pd);
```

Figure: Create a Protection Domain

```
/* Allocate a virtual interface */
int ef_vi_alloc_from_pd(ef_vi *vi, ef_driver_handle vi_dh,
                        ef_pd *pd, ef_driver_handle pd_dh,
                        int eventq_cap, int rxq_cap, int txq_cap,
                        ef_vi *opt_evq, ef_driver_handle opt_evq_dh,
                        enum ef_vi_flags flags));
```

Figure: Allocate a Virtual Interface

Copyright © Solarflare Communications 2017

5.3.1 Using Virtual Interface Sets.

A virtual interface set can be used instead of a single virtual interface, to distribute load using RSS. Functionality is almost the same as working with a single virtual interface:

- To allocate the virtual interfaces:
  - use ef_vi_set_alloc_from_pd() to allocate the set
  - then use ef_vi_alloc_from_set() to allocate each virtual interface in the set
- To add a filter:
  - use ef_vi_set_filter_add() to add the filter onto the set, rather than adding it to each virtual interface individually (which would cause replication on a 7000-series NIC).

The efrss sample gives an example of usage.

5.4 Creating packet buffers

Memory used for packet buffers is allocated using standard functions such as posix_memalign(). A packet buffer should be at least as large as the value returned from ef_vi_receive_buffer_len().

The packet buffers must be pinned so that they cannot be paged, and they must be registered for DMA with the network adapter. These requirements are enforced by calling ef_memreg_alloc() to register the allocated memory for use with ef_vi.

The type `ef_iovec` encapsulates a set of buffers. The adapter uses a special address space to identify locations in these buffers, and such addresses are designated by the type `ef_addr`.

```
/* Allocate a memory region */
int ef_memreg_alloc(ef_memreg* mr, ef_driver_handle mr_dh,
                    ef_pd* pd, ef_driver_handle pd_dh,
                    void* p_mem, int len_bytes);
```

Figure: Allocate a Memory Region

To improve performance, registered memory regions for packet buffers should be aligned on (minimum) 4KB boundaries for regular pages or 2MB boundaries when using use huge pages.

5.5 Transmitting Packets

To transmit packets, the basic process is:

1. Write the packet contents (including all headers) into one or more packet buffers.

   The packet buffer memory must have been previously registered with the protection domain.

 Copyright © Solarflare Communications 2017

2. Post a descriptor for the filled packet buffer onto the TX descriptor ring, by calling ef_vi_transmit_init() and ef_vi_transmit_push(), or ef_vi_transmit().

   A doorbell is "rung" to inform the adapter that the transmit ring is non-empty. If the transmit descriptor ring is empty when the doorbell is rung, 'TX PUSH' is used. In 'TX_PUSH', the doorbell is rung and the address of the packet buffer is written in one shot improving latency. TX_PUSH can cause ef_vi to poll for events, to check if the transmit descriptor ring is empty, before sending which can lead to a latency versus throughput trade off in some scenarios.

3. Poll the event queue to find out when the transmission is complete.

   See Handling Events.

   When transmitting, polling the event queue is less critical; but does still need to be done. The events of interest are `EF_EVENT_TYPE_TX` or `EF_EVENT_TYPE_TX_WITH_TIMESTAMP` telling you that a transmit completed, and `EF_EVENT_TYPE_TX_ERROR` telling you that a transmit failed. EF_EVENT_TX_Q_ID() can be used to extract the id of the referenced packet, or you can just rely on the fact that ef_vi always transmits packets in the order they are submitted.

4. Handle the resulting event.

   Reclaim the packet buffer for re-use.

```
// construct packet with proper headers

// Post on the transmit ring
ef_vi_transmit_init(&vi, addr, len, id);
// Ring doorbell
ef_vi_transmit_push(&vi);
```

Figure: Transmit Packets

5.5.1 Transmitting Jumbo Frames

Packets of a size smaller than the interface MTU but larger than the packet buffer size must be sent from multiple buffers as jumbo packets. A single `EF_EVENT_TYPE_TX` (or `EF_EVENT_TYPE_TX_ERROR`) event is raised for the entire transmit:

- Use ef_vi_transmitv() to chain together multiple segments with a higher total length.
- MTU is not enforced. If the transmit is to remain within MTU, the application must check and enforce this.
- The segments must at least split along natural (4k packet) boundaries, but smaller segments can be used if desired..

5.5.2 Programmed I/O

Programmed IO is usable only on 7000-series cards, not on the older cards. It allows for faster transmit, especially of small packets, but the hardware resources available for it are limited.

For this reason, a PIO buffer must be explicitly allocated and associated with a virtual interface before use, by calling ef_pio_link_vi().

Data is copied into the PIO buffer with ef_pio_memcpy().

When the PIO buffer is no longer required it should be unlinked by calling ef_pio_unlink_vi(), and then freed by calling ef_pio_free().

Copyright © Solarflare Communications 2017

5.5.3 TX Alternatives

TX alternatives is a features available on Solarflare SFN8000 series adapters. To use TX alternatives for a given virtual interface, you must set the EF_VI_TX_ALT flag when you allocate the virtual interface. You must then allocate a set of TX alternatives for the virtual interface by calling ef_vi_transmit_alt_alloc().

```
int ef_vi_transmit_alt_alloc(struct ef_vi* vi,
      ef_driver_handle vi_dh,
                              int num_alts, int buf_space);
```

This creates a set of TX alternatives. The TX alternatives remain allocated until the virtual interface is freed.

The TX alternatives in the set are given sequential IDs, from 0 upwards. You can get the number of TX alternatives in a set by calling ef_vi_transmit_alt_num_ids().

```
unsigned ef_vi_transmit_alt_num_ids(ef_vi* vi);
```

You can then buffer responses on the different TX alternatives, choose which to transmit, and discard any that are no longer required:

- All TX alternatives are initially in the STOP state, and any packets sent to them are buffered.
- To send packets to a particular TX alternative, select the TX alternative by calling ef_vi_transmit_alt_select(), and then send the packets to the virtual interface using normal send calls such as ef_vi_transmit().
- To transmit the packets that are buffered on a TX alternative, call ef_vi_transmit_alt_go().

  This transitions the TX alternative to the GO state. While the TX alternative remains in this state, any further packets sent to it are transmitted immediately.
- To transition the TX alternative back to the STOP state, call ef_vi_transmit_alt_stop().
- To discard the packets buffered on a TX alternative, transition it to the DISCARD state by calling ef_vi_transmit_alt_discard().

```
int ef_vi_transmit_alt_select(struct ef_vi*, unsigned alt_id);

int ef_vi_transmit_alt_stop(struct ef_vi*, unsigned alt_id);

int ef_vi_transmit_alt_go(struct ef_vi*, unsigned alt_id);

int ef_vi_transmit_alt_discard(struct ef_vi*, unsigned alt_id);
```

As packets are transmitted or discarded, events of type EF_EVENT_TYPE_TX_ALT are returned to your application. Your application should normally wait until all packets have been processed before transitioning to a different state.

5.6 Handling Events

The event queue is a channel from the adapter to software which notifies software when packets arrive from the network, and when transmits complete (so that the buffers can be freed or reused). Application threads retrieve these events in one of the following ways:

- A thread can busy-wait for an event notification by calling ef_eventq_poll() repeatedly in a tight loop. This gives the lowest latency.
- A thread can block until event notifications arrive (or a timeout expires) by calling ef_eventq_wait(). This frees the CPU for other usage.

The batch size for polling must be at least EF_VI_EVENT_POLL_MIN_EVS. It should be greater than the batch size for refilling to detect when the receive descriptor ring is going empty.

When timestamping is enabled, a number of timestamps per second are added to the event queue, even when no packets are being received. It is important that the application regularly polls the VI event queue, to avoid an event queue overflow (EF_EVENT_TYPE_OFLOW) which can result in an undetermined state for the VI.

 Copyright © Solarflare Communications 2017

5.6.1 Blocking on a file descriptor

Ef_vi supports integration with other types of I/O via the select, poll and epoll interfaces. Each virtual interface is associated with a file descriptor. The ef_vi layer supports blocking on a file descriptor until it has events ready, when it becomes readable. This feature provides the functionality that is already provided by ef_eventq_wait() with the added benefit that as you are blocking on a file descriptor, you can block for events on a virtual interface along with other file descriptors at the same time.

The file descriptor to use for blocking is the driver handle that was used to allocate the virtual interface.

Before you can block on the file descriptor, you need to prime interrupts on the virtual interface. This is done by calling ef_vi_prime().

```
int ef_vi_prime(ef_vi* vi, ef_driver_handle dh,
                unsigned current_ptr);
```

When this function is called, you must tell it how many events you've read from the eventq which can be retrieved by using ef_eventq_current(). Then you can simply block on the file descriptor becoming readable by using select(), poll(), epoll(), etc. When the file descriptor is returned as readable, you can then get the associated events by polling the eventq in the normal way. Note that at this point, you must call ef_vi_prime() again (with the current value from ef_eventq_current()) before blocking on the file descriptor again.

The efpingpong example code has been updated to offer a simple example.

5.7 Receiving packets

To receive packets, the basic process is:

1. Post descriptors for empty packet buffers onto the RX descriptor ring, by calling ef_vi_receive_init() and ef↩_vi_receive_push(), or ef_vi_receive_post().

   Receive descriptors should be posted in multiples of 8. When an application pushes 10 descriptors, ef_vi will push 8 and ef_vi will ignore descriptor batch sizes <8. Users should beware that if the ring is empty and the application pushes <8 descriptors before blocking on the event queue, the application will remain blocked as there are no descriptors available to receive packets so nothing gets posted to the event queue.

   Posting descriptors is relatively slow. It should ideally be done in batches, by a thread that is not on the critical path. A small batch size means that the ring is kept more full, but a large batch size is more efficient. A size of 8, 16 or 32 is probably the best compromise.

2. Poll the event queue to see that they are now filled.

   See Handling Events.

   Packets are written into the buffers in FIFO order.

3. Handle the resulting event and the incoming packet.

   Since the adapter is cut-through, errors in receiving packets like multicast mismatch, CRC errors, etc. are delivered along with the packet. The software must detect these errors and recycle the associated packet buffers.

Copyright © Solarflare Communications 2017

5.7.1 Finding the Packet Data

- If you're using the ef_sfw wrapper, then RX_PKT_PTR() returns a pointer to the start of the payload of the packet. (Remember that this includes the headers, as ef_vi does not do any protocol handling.)
- Otherwise, you'll need to create similar functionality yourself. Use the queue ID to find the packet, and ef_vi_receive_prefix_len() to find the offset for the packet data.

5.7.2 Receiving Jumbo Packets

Packets of a size smaller than the interface MTU but larger than the packet buffer size are delivered in multiple buffers as jumbo packets. An event is raised for each packet buffer that is filled:

- The EF_EVENT_RX_CONT() macro can be used to check if this is not the last part of the packet, and that the next receive (on this RX descriptor ring) should also be examined as being part of this jumbo frame.
- The length given in each event is the total length of the packet so far (so the length in the last part of the frame is the total packet length.)
- If `EF_EVENT_RX_DISCARD_TRUNC` is set, this indicates that the packet buffer has been dropped, and so the jumbo packet has been truncated. But there might still be more parts of the jumbo packet that arrive after the drop. All packet buffers should be discarded until one is received with `EF_EVENT_RX_SOP` set, marking the start of a new packet.

5.8 Adding Filters

Filters are the means by which the adapter decides where to deliver packets it receives from the network. By default all packets are delivered to the kernel network stack. Filters are added by an ef_vi application to direct received packets to a given virtual interface.

- If a filter cannot be added, for example because an incompatible filter already exists, an error is returned.
- By default the 'all' filters are sending everything to the kernel. They are equivalent to setting promiscuous mode on the NIC, and super-user rights (specifically `CAP_NET_ADMIN`) are needed to use this filter.
- On SFN5000 and SFN6000 series NICs each filter can only exist for one virtual interface, and so each packet which arrives can be forwarded only to a single application (unless two applications share a stack). Only the first application to insert a specific filter will succeed; other applications will then get an error.

  Note that this includes filters inserted both by other ef_vi applications and by Onload - which typically uses only fully connected and listen filters.
- The SFN7000 and SFN8000 series NICs are not subject to this restriction. If two applications insert the same filter, a copy of the packets is delivered to each application and applications remain unaware of each other.
- IP filters do not match IP fragments, which are therefore received by the kernel stack. If this is an issue, layer 2 filters should be installed by the user.
- There are no ranges, or other local wildcard support. To filter on a range of values, one of the following is required:
  - insert multiple filters, one per value in the range (NICs support upwards of a thousand filters easily)
  - have the interesting traffic sent to a specific MAC address, and use a MAC address filter

 Copyright © Solarflare Communications 2017

- have the interesting traffic sent to a specific VLAN, and use a VLAN filter.

- Cookies are used to remove filters.
- De-allocating a virtual interface removes any filters set for the virtual interface.

Filters are checked in the following order, which is roughly most-specific first:

1. Fully connected TCP/UDP. (Specifies local and remote port and IP)
2. Listen socket. (Specifies local port and IP, but allows any remote IP/port)
3. Destination MAC address, and optionally VLAN. (Useful for multicast reception, though IP can be used instead if preferred. Also useful for custom protocols.)
4. Everything else. (All unicast, all multicast.)

```
void ef_filter_spec_init(ef_filter_spec* fs,
                         enum ef_filter_flags flags);

int ef_filter_spec_set_ip4_local(ef_filter_spec* fs,
                                 int protocol,
                                 unsigned host_be32, int port_be16);

int ef_filter_spec_set_ip4_full(ef_filter_spec* fs,
                                int protocol,
                                unsigned host_be32, int port_be16,
                                unsigned rhost_be32, int rport_be16);

int ef_filter_spec_set_vlan(ef_filter_spec* fs,
                            int vlan_id);

int ef_filter_spec_set_eth_local(ef_filter_spec* fs,
                                 int vlan_id,
                                 const void *mac);

int ef_filter_spec_set_unicast_all(ef_filter_spec* fs);

int ef_filter_spec_set_multicast_all(
      ef_filter_spec* fs);

int ef_filter_spec_set_unicast_mismatch(
      ef_filter_spec* fs);

int ef_filter_spec_set_multicast_mismatch(
      ef_filter_spec* fs);

int ef_filter_spec_set_port_sniff(ef_filter_spec* fs,
                                  int promiscuous);

int ef_filter_spec_set_tx_port_sniff(
      ef_filter_spec* fs);

int ef_filter_spec_set_block_kernel(
      ef_filter_spec* fs);

int ef_filter_spec_set_block_kernel_multicast(
      ef_filter_spec* fs);

int ef_filter_spec_set_block_kernel_unicast(
      ef_filter_spec* fs);

int ef_vi_filter_add(ef_vi* vi,ef_driver_handle dh,
                     const ef_filter_spec* fs,
                     ef_filter_cookie* filter_cookie_out);

int ef_vi_filter_del(ef_vi* vi, ef_driver_handle dh,
                     ef_filter_cookie* filter_cookie);

int ef_vi_set_filter_add(ef_vi_set*,
                         ef_driver_handle dh,
                         const ef_filter_spec* fs,
                         ef_filter_cookie* filter_cookie_out);

int ef_vi_set_filter_del(ef_vi_set*,
      ef_driver_handle dh,
                         ef_filter_cookie* filter_cookie);
```

Figure: Creating Filters

 Copyright © Solarflare Communications 2017

5.9 Filters available per Firmware Variant

The kernel will install a destination MAC address filter for each Solarflare interface. The kernel will install a broadcast MAC address filter for each Solarflare interface.

A broadcast MAC address is treated like any other MAC address. Multiple applications can insert the same filter and all will receive a copy of the received traffic.

As a rule, more-specific filters will capture traffic over less-specific filters.

When the Solarflare adapter is configured to use the full_featured firmware variant filters are applied in the following order:

- MATCH_ETHER_TYPE
- MATCH_SRC_IP
- MATCH_SRC_PORT
- MATCH_DST_IP
- MATCH_DST_PORT
- MATCH_IP_PROTO
- MATCH_OUTER_VLAN
- MATCH_INNER_VLAN
- MATCH_DST_MAC

- MATCH_OUTER_VLAN
- MATCH_INNER_VLAN
- MATCH_DST_MAC
- MATCH_OUTER_VLAN
- MATCH_DST_MAC

- MATCH_UNKNOWN_UCAST_DST
- MATCH_OUTER_VLAN
- MATCH_INNER_VLAN

- MATCH_UNKNOWN_MCAST
- MATCH_OUTER_VLAN
- MATCH_INNER_VLAN

- MATCH_UNKNOWN_UCAST_DST

 Copyright © Solarflare Communications 2017

- MATCH_OUTER_VLAN

- MATCH_UNKNOWN_MCAST_DST
- MATCH_OUTER_VLAN

- MATCH_UNKNOWN_UCAST_DST
- MATCH_UNKNOWN_MCAST_DST

When the Solarflare adapter is configured to use the low_latency firmware variant filters are applied in the following order:

- MATCH_ETHER_TYPE
- MATCH_SRC_IP
- MATCH_SRC_PORT
- MATCH_DST_IP
- MATCH_DST_PORT
- MATCH_IP_PROTO

- MATCH_ETHER_TYPE
- MATCH_DST_IP
- MATCH_DST_PORT
- MATCH_IP_PROTO

- MATCH_DST_MAC
- MATCH_OUTER_VLAN

- MATCH_DST_MAC

- MATCH_UNKNOWN_UCAST_DST
- MATCH_UNKNOWN_MCAST_DST

When the Solarflare adapter is configured to use the packed_stream firmware variant filters are applied in the following order:

- MATCH_ETHER_TYPE

 Copyright © Solarflare Communications 2017

- MATCH_DST_IP
- MATCH_DST_PORT
- MATCH_IP_PROTO

- MATCH_DST_MAC

- MATCH_UNKNOWN_UCAST_DST
- MATCH_OUTER_VLAN

- MATCH_UNKNOWN_MCAST_DST
- MATCH_OUTER_VLAN

- MATCH_UNKNOWN_UCAST_DST
- MATCH_UNKNOWN_MCAST_DST

5.10 Freeing Resources

Users of ef_vi must do the following to free resources:

1. Release and free memory regions by calling ef_memreg_free().
2. Release and free a virtual interface by calling ef_vi_free().
3. Release and free a protection domain by calling ef_pd_free().
4. Close a driver handle by calling ef_driver_close().

```
/* Release and free a memory region */
int ef_memreg_free(ef_memreg* mr,
                   ef_driver_handle mr_dh);
```

Figure: Release and Free a Memory Region

```
/* Release and free a virtual interface */
int ef_vi_free(ef_vi* vi,
               ef_driver_handle vi_dh);
```

Figure: Release and Free a Virtual Interface

```
/* Release and free a protection domain */
int ef_pd_free(ef_pd *pd,
               ef_driver_handle pd_dh);
```

Figure: Release and Free a Protection Domain

 Copyright © Solarflare Communications 2017

5.11 Design Considerations

This section outlines some considerations that are required when designing an ef_vi application.

5.11.1 Interrupts

Interrupts are not enabled by ef_vi by default.

Interrupts are enabled only if ef_eventq_wait() is called. If there are no events immediately ready, then this function will enable an interrupt, and sleep until that interrupt fires. Interrupts are then disabled again.

5.11.2 Thread Safety

There is no thread-safety on ef_vi functions. This is for speed. If thread-safety is required, it must be provided by the ef_vi application.

The usual use-case is to have multiple virtual interface structures for independent operation. There is then no need to lock.

5.11.3 Packet Buffer Addressing

Different configuration modes are available for addressing packet buffers.

- *Network Adapter Buffer Table Mode* uses a block of memory on the adapter to store a translation table mapping between buffer IDs and physical addresses:
  - When using a SFN5000 or SFN6000 series adapter there are 65536 entries in the buffer table. Each entry maps a 4KB page of memory that holds two 2KB packet buffers, and so a maxiumum of 131072 packet buffers are available. The kernel uses some of these, leaving about 120,000 packet buffers available for ef_vi.
  - The SFN7000 series adapters have *Large Buffer Table Support*. Each entry can map a larger region of memory, or a huge page, enabling them to support many more packet buffers without the need to use Scalable Packet Buffer Mode.

  This is the default mode.
- *Scalable Packet Buffer Mode* allocates packet buffers from the kernel IOMMU. It uses Single Root I/O Virtualization (SR-IOV) virtual functions (VF) to provide memory protection and translation. This removes the buffer limitation of the buffer table.
  - SR-IOV must be enabled
  - the kernel must support an IOMMU.

  An ef_vi application can enable this mode by setting the environment variable `EF_VI_PD_FLAGS=vf`.
- *Physical Addressing Mode* uses actual physical addresses to identify packet buffers. An ef_vi application can therefore direct the adapter to access memory that is not in the application address space. For example, this can be used for zero-copy from the kernel buffer cache. any piece of memory.

  Physical Addressing Mode allows stacks to use large amounts of packet buffer memory, avoiding address translation limitations on some adapters and without the need to configure and use SR-IOV virtual functions.

 Copyright © Solarflare Communications 2017

- No memory protection is provided. Physical addressing mode removes memory protection from the network adapter's access of packet buffers. Unprivileged user-level code is provided and directly handles the raw physical memory addresses of packet buffers. User-level code provides physical memory addresses directly to the adapter with the ability to read or write arbitrary memory locations.
- It is important to ensure that packet buffers are page aligned.

An ef_vi application can enable this mode by setting the environment variable `EF_VI_PD_FLAGS=phys`.

The sfc_char module option must also be enabled in a file in the /etc/modprobe.d directory where N is the integer group ID of the user running the ef_vi application. Set to -1 means ALL users are permitted access.:

```
options sfc_char phys_mode_gid=<N>
```

For more information about these configuration modes, see the chapter titled *Packet Buffers* in the *Onload User Guide* (SF-104474-CD).

5.11.4 Virtual machines

Ef_vi can be used in virtual machines provided PCI passthrough is used. With PCI passthrough a slice of the network adapter is mapped directly into the address space of the virtual machine so that device drivers in the VM OS can access the network adapter directly. To isolate VMs from one another and from the hypervisor, I/O addresses are also virtualised. I/O addresses generated by the network adapter are translated to physical memory addresses by the system IOMMU.

When ef_vi is used in virtual machines two levels of address translation are performed by default. Firstly a translation from DMA address to I/O address, performed by the adapter to isolate the application from other applications and the kernel. Then a translation from I/O address to physical address by the system IOMMU, isolating the virtual machines. Physical address mode can also be used in virtual machines, in which case the adapter translation is omitted.

5.12 Known Limitations

5.12.1 Timestamping

When timestamping is enabled, the VI must be polled regularly (even when no packets are available). Timestamps consume four event queue slots per second and failing to poll the VI can result in event queue overflow. When there are no packets to receive, stale timestamps are not returned to the application, but polling ensures that they are cleared from the event queue.

5.12.2 Minimum Fill Level

You must poll for at least EF_VI_EVENT_POLL_MIN_EVS at a time. You will also need to initially fill the RX ring with at least 16 packet buffers to ensure that the card begins acquiring packets. (It's OK to underrun once the application has started; although of course doing so risks drops.)

It's (very slightly) more efficient to refill the ring in batch sizes of 8/16/32 or 64 anyway.

Copyright © Solarflare Communications 2017

5.13 Example

Below is a simple example showing a starting framework for an ef_vi application. This is not a complete program. There is no initialization, and much of the other required code is only indicated by comments.

```
static void handle_poll(ef_vi *vi)
{
  ef_event events[POLL_BATCH_SIZE];
  int n_ev = ef_eventq_poll(&vi, events, POLL_BATCH_SIZE);
  for( i = 0; i < n_ev; ++i ) {
    switch( EF_EVENT_TYPE(events[i]) ) {
    case EF_EVENT_TYPE_RX:
      // Accumulate used buffer
      break;
    case EF_EVENT_TYPE_TX:
      /* Each EF_EVENT_TYPE_TX can signal multiple completed sends */
      int num_completed = ef_vi_transmit_unbundle(vi, events[i], &dma_id);
      break;
    case EF_EVENT_TYPE_RX_DISCARD:
    case EF_EVENT_TYPE_RX_NO_DESC_TRUNC:
      /* Discard events also use up buffers */
      // Accumulate buffer in user space
      break;
    default:
      /* Other error types */
    }
  }
}

static void refill_rx_ring(ef_vi *vi)
{
  if( ef_vi_receive_space(&vi) < REFILL_BATCH_SIZE )
    return;
  int refill_count = REFILL_BATCH_SIZE;
  /* Falling too low? */
  if( ef_vi_receive_space(&vi) > ef_vi_receive_capacity(&vi) / 2 )
    refill_count = ef_vi_receive_space(&vi);
  /* Enough free buffers? */
  if( refill_count > free_bufs_in_sw )
    refill_count = free_bufs_in_sw;
  /* Round down to batch size */
  refill_count &= ~(REFILL_BATCH_SIZE - 1);
  if( refill_count ) {
    while( refill_count ) {
      ef_vi_receive_init(...);
      --refill_count;
    }
    ef_vi_receive_push(&vi);
  }
}

int main(int argc, char argv[]) (
  while( 1 ) {
    poll_events(&vi);
    refill_rx_ring(&vi);
  }
}
```


Copyright © Solarflare Communications 2017

Chapter 6

Worked Example

This part of the documentation examines a simplified version of eflatency. This is a small application which listens for packets and replies, with as low latency as possible.

This documentation discusses the tradeoffs that have been chosen, some performance issues to avoid, and some possible additions that have been omitted for clarity.

See also the supplied code for eflatency , which includes many of these improvements.

6.1 Setup

The first step is to set up ef_vi:

- #include the various headers we need (etherfabric/pd.h, vi.h, memreg.h)
- open the driver
- allocate a protection domain
- allocate a virtual interface from the protection domain.

```
ef_driver_handle  driver_handle;
ef_vi             vi;
ef_pd             pd;
static void do_init(int ifindex){
  ef_driver_open(&driver_handle);
  ef_pd_alloc(&pd, driver_handle, ifindex, EF_PD_DEFAULT );
  ef_vi_alloc_from_pd(&vi, driver_handle, &pd, driver_handle,
                      -1, -1, -1, NULL, -1, 0);
```

The following improvements could be made:

- check the return values from these functions, in case the card has run out of resources and is unable to allocate more virtual interfaces
- offer physical buffer mode here.

 Copyright © Solarflare Communications 2017

6.2 Creating Packet buffers

The next step is to allocate a memory region and register it for packet buffers.

```
const int BUF_SIZE = 2048;  /* Hardware always wants 2k buffers */
int bytes = N_BUFS * BUF_SIZE;
void* p;
posix_memalign(&p, 4096, bytes)  /* allocate aligned memory */
ef_memreg_alloc(&memreg, driver_handle, &pd, driver_handle,
                p, bytes); /* Make it available to ef_vi */
```

This is all that is strictly necessary to set up the packet buffers.

However, the packet buffer is 2048 bytes long, whereas the normal MTU size for a transmitted packet is only 1500 bytes. There is some spare memory in each packet buffer. Performance can be improved by using this space to cache some of the packet meta-data, so that it does not have to be recalculated:

- The DMA address of the packet is cached. It is determined by getting the DMA address of the base of the memory chunk, and then incrementing in 2KB chunks.
- The packet ID is also cached.

A structure is used to store the cached meta-data and a few pointers in the buffer. An array is used to track all the buffers:

```
#define MEMBER_OFFSET(c_type, mbr_name) \
                     ((uint32_t) (uintptr_t)(&((c_type*)0)->mbr_name))
#define CACHE_ALIGN  __attribute__((aligned(EF_VI_DMA_ALIGN)))
struct pkt_buf {
  struct pkt_buf* next;
  /* We're not actually going to use this;
     but chaining multiple buffers together is a common and useful trick. */
  ef_addr          dma_buf_addr;
  int              id;
  uint8_t          dma_buf[1] CACHE_ALIGN;
  /* Not strictly required, but cache aligning the payload is a speed
     boost, so do it. */
};
/* We're also going to want to keep track of all our buffers, so have an
   array of them.  Not strictly needed, but convenient.  */
    struct pkt_buf* pkt_bufs [N_BUFS];
    for( i = 0; i < N_BUFS; ++i ) {
      struct pkt_buf* pb = (struct pkt_buf*) ((char*) p + i * 2048);
      pb->id = i;
      pb->dma_buf_addr = ef_memreg_dma_addr(&memreg, i * 2048);
      pb->dma_buf_addr += MEMBER_OFFSET(struct pkt_buf, dma_buf);
      pkt_bufs[i] = pb;
    }
```

Note

When receiving, the hardware will only fill up to 1824 bytes per buffer. Larger jumbo frames are split across multiple buffers.

6.3 Adding Filters

Next, a filter is specified and added, so that the virtual interface receives traffic. Assuming there is a sockaddr to work from:

```
struct sockaddr_in sa_local;  /* TODO: Fill this out somehow */
ef_filter_spec_init(&filter_spec, EF_FILTER_FLAG_NONE);
TRY(ef_filter_spec_set_ip4_local(&filter_spec, IPPROTO_UDP,
                                  sa_local.sin_addr.s_addr,
                                  sa_local.sin_port));
TRY(ef_vi_filter_add(&vi, driver_handle, &filter_spec, NULL));
```


Copyright © Solarflare Communications 2017

6.4 Receiving packets

At this point, packets will start arriving at the interface, be diverted to the application, and immediately be dropped.

So the next step is to push some packet buffers to the RX descriptor ring, to receive the incoming packets.

For efficiency, the code pushes packet buffers eight at a time.

```
unsigned rx_posted = 0;  /* We need to keep track of which buffers are
                            already on the ring */
void rx_post( int n ) {
  for( int i = 0; i < n; ++i ) {
    struct pkt_buf* pb = pkt_bufs[rx_posted % N_RX_BUFS];
    ef_vi_receive_init(&vi, pb->dma_buf_addr, pb->id);
    ++rx_posted;
  }
  ef_vi_receive_push(&vi);

}
```

So now, there are packet buffers on the descriptor ring. But once they are filled, the application will start dropping again.

6.5 Handling Events

The next step is to handle these incoming packets, by polling the event queue.

```
void rx_wait(void) {
  /* Again, for efficiency, poll multiple events at once.  */
  ef_event       evs[NUM_POLL_EVENTS];
  int            n_ev, i;

  while( 1 ) {
    n_ev = ef_eventq_poll(&vi, evs, NUM_POLL_EVENTS);
    if( n_ev > 0 )
      for( i = 0; i < n_ev; ++i )
        switch( EF_EVENT_TYPE(evs[i]) ) {
        case EF_EVENT_TYPE_RX:
          handle_rx_packet(EF_EVENT_RX_RQ_ID(evs[i]),
                           EF_EVENT_RX_BYTES(evs[i]) );
          break;
        case EF_EVENT_TYPE_RX_DISCARD:
          /* Interesting to print out the cause of the discard */
          fprintf(stderr, "ERROR: RX_DISCARD type=%d",
                  EF_EVENT_RX_DISCARD_TYPE(evs[i]));
          /* but let's handle it like a normal packet anyway */
          handle_rx_packet(EF_EVENT_RX_RQ_ID(evs[i]),
                           EF_EVENT_RX_BYTES(evs[i]) );
          break;
} } }
```

This code is calling a `handle_rx_packet()` function, passing it the packet id (which corresponds directly to the `pkt_bufs` array - see Creating Packet buffers) and the length of data. The body of this function is not shown, but it should do the following:

- note that this packet buffer has been consumed, and so is ready to be re-posted:
  - the `rx_post()` function(see Receiving packets) must also be updated to use this information, so a buffer is not re-posted until it is marked as consumed
- ensure that the received packet is processed according to the purpose of the application:
  - if the application can always process incoming packets as fast as they are received, then it can do its work inline, and immediately repost the buffer on the ring
  - otherwise, the application should probably post an item on a work queue for another thread to act upon, and arrange for the refill to come from a larger pool of buffers
- optionally, handle discards in some different way (perhaps not raising the work event).

 Copyright © Solarflare Communications 2017

6.6 Transmitting packets

The next step is to implement the transmit side. The hard part is filling out the payload, and getting all the fields of IP and UDP correct. (ef_vi is usually used to transmit UDP, as it's a much simpler protocol to implement than TCP.)

There's some sample code to fill out the headers in the functions `ci_*_hdr_init()`, which can be found in `src/lib/citools/ippacket.c`.

After that, to transmit one of the `pb` structures (see Creating Packet buffers), a single function call is needed:

```
ef_vi_transmit(&vi, pb->dma_buf_addr, frame_length, 0);
```

But the application must also keep track of when that buffer is used, and when it is free. This means adding some complexity to the poll loop (see Handling Events). The absolute minimum is:

```
case EF_EVENT_TYPE_TX:
  ef_vi_transmit_unbundle(&vi, &evs[i], ids);
  break;
```

This is only sufficient if the TX buffers and the RX buffers are from different pools.

Note

> In ping pong there is only ever one outstanding send. The application does not transmit another packet until the remote side has processed the current one, and so the application does not even need to keep track of its state.

One option would be to free up sent buffers, to a pool ready to be filled with data. Other applications may instead fill a few packet buffers with data, and then transmit them as and when, only keeping track to make sure that the TX ring never overfills.

Copyright © Solarflare Communications 2017

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

ef_event
A token that identifies something that has happened 41
ef_eventq_state
State of event queue 45
ef_filter_cookie
Cookie identifying a filter 46
ef_filter_spec
Specification of a filter 47
ef_iovec
Ef_iovec is similar to the standard struct iovec. An array of these is used to designate a scatter/gather list of I/O buffers 49
ef_memreg
Memory that has been registered for use with ef_vi 50
ef_packed_stream_packet
Per-packet meta-data 51
ef_packed_stream_params
Packed-stream mode parameters 53
ef_pd
A protection domain 54
ef_pio
A Programmed I/O region 56
ef_vi
A virtual interface 58
ef_vi_layout_entry
Layout of the data that is delivered into receive buffers 66
ef_vi_nic_type
The type of NIC in use 67
ef_vi_rxq
RX descriptor ring 69
ef_vi_rxq_state
State of RX descriptor ring 70
ef_vi_set
A virtual interface set within a protection domain 72

Copyright © Solarflare Communications 2017

ef_vi_state
State of a virtual interface 73
ef_vi_stats
Statistics for a virtual interface 74
ef_vi_stats_field_layout
Layout for a field of statistics 75
ef_vi_stats_layout
Layout for statistics 77
ef_vi_transmit_alt_overhead
Per-packet overhead information 78
ef_vi_txq
TX descriptor ring 79
ef_vi_txq_state
State of TX descriptor ring 80
ef_vi::ops
Driver-dependent operations 82

 Copyright © Solarflare Communications 2017

Chapter 8

File Index

8.1 File List

Here is a list of all documented files with brief descriptions:

base.h
Base definitions for EtherFabric Virtual Interface HAL 92
capabilities.h
Capabilities API for EtherFabric Virtual Interface HAL 94
cluster_protocol.h . . . . **??**
ef_vi.h
Virtual Interface definitions for EtherFabric Virtual Interface HAL 98
internal.h . . . . **??**
memreg.h
Registering memory for EtherFabric Virtual Interface HAL 138
packedstream.h
Packed streams for EtherFabric Virtual Interface HAL 141
pd.h
Protection Domains for EtherFabric Virtual Interface HAL 143
pio.h
Programmed Input/Output for EtherFabric Virtual Interface HAL 148
timer.h
Timers for EtherFabric Virtual Interface HAL 151
vi.h
Virtual packet / DMA interface for EtherFabric Virtual Interface HAL 155

 Copyright © Solarflare Communications 2017

Chapter 9

Data Structure Documentation

9.1 ef_event Union Reference

A token that identifies something that has happened.

```
#include <ef_vi.h>
```

Data Fields

- struct {
  unsigned **type**:16
  } generic
- struct {
  unsigned **type**:16
  unsigned **q_id**:16
  unsigned **rq_id**:32
  unsigned **len**:16
  unsigned **flags**:16
  } rx
- struct {
  unsigned **type**:16
  unsigned **q_id**:16
  unsigned **rq_id**:32
  unsigned **len**:16
  unsigned **flags**:16
  unsigned **subtype**:16
  } rx_discard
- struct {
  unsigned **type**:16
  unsigned **q_id**:16
  unsigned **desc_id**:16
  } tx

- struct {
  unsigned **type**:16
  unsigned **q_id**:16
  unsigned **desc_id**:16
  unsigned **subtype**:16
  } tx_error

- struct {
  unsigned **type**:16
  unsigned **q_id**:16
  unsigned **rq_id**:32
  unsigned **ts_sec**:32
  unsigned **ts_nsec**:32
  } tx_timestamp

- struct {
  unsigned **type**:16
  unsigned **q_id**:16
  unsigned **alt_id**:16
  } tx_alt

- struct {
  unsigned **type**:16
  unsigned **q_id**:16
  } rx_no_desc_trunc

- struct {
  unsigned **type**:16
  unsigned **q_id**:16
  unsigned **flags**:16
  unsigned **n_pkts**:16
  unsigned **ps_flags**:8
  } rx_packed_stream

- struct {
  unsigned **type**:16
  unsigned **data**
  } sw

- struct {
  unsigned **type**:16
  unsigned **q_id**:16
  unsigned **n_descs**:16
  unsigned **flags**:16
  } rx_multi

- struct {
  unsigned **type**:16
  unsigned **q_id**:16
  unsigned **n_descs**:16
  unsigned **flags**:16
  unsigned **subtype**:16
  } rx_multi_discard

 Copyright © Solarflare Communications 2017

9.1.1 Detailed Description

A token that identifies something that has happened.

Examples include packets received, packets transmitted, and errors.

Users should not access this structure, but should instead use the macros provided.

Definition at line 155 of file ef_vi.h.

9.1.2 Field Documentation

9.1.2.1 generic

```
struct { ... } generic
```

A generic event, to query the type when it is unknown

9.1.2.2 rx

```
struct { ... } rx
```

An event of type EF_EVENT_TYPE_RX

9.1.2.3 rx_discard

```
struct { ... } rx_discard
```

An event of type EF_EVENT_TYPE_RX_DISCARD

9.1.2.4 rx_multi

```
struct { ... } rx_multi
```

An event of type EF_EVENT_TYPE_RX_MULTI

9.1.2.5 rx_multi_discard

```
struct { ... } rx_multi_discard
```

An event of type EF_EVENT_TYPE_RX_MULTI_DISCARD

 Copyright © Solarflare Communications 2017

9.1.2.6 rx_no_desc_trunc

```
struct { ... } rx_no_desc_trunc
```

An event of type EF_EVENT_TYPE_RX_NO_DESC_TRUNC

9.1.2.7 rx_packed_stream

```
struct { ... } rx_packed_stream
```

An event of type EF_EVENT_TYPE_RX_PACKED_STREAM

9.1.2.8 sw

```
struct { ... } sw
```

An event of type EF_EVENT_TYPE_SW

9.1.2.9 tx

```
struct { ... } tx
```

An event of type EF_EVENT_TYPE_TX

9.1.2.10 tx_alt

```
struct { ... } tx_alt
```

An event of type EF_EVENT_TYPE_TX_ALT

9.1.2.11 tx_error

```
struct { ... } tx_error
```

An event of type EF_EVENT_TYPE_TX_ERROR

9.1.2.12 tx_timestamp

```
struct { ... } tx_timestamp
```

An event of type EF_EVENT_TYPE_TX_WITH_TIMESTAMP

The documentation for this union was generated from the following file:

- ef_vi.h

Copyright © Solarflare Communications 2017

9.2 ef_eventq_state Struct Reference

State of event queue.

```
#include <ef_vi.h>
```

Data Fields

- ef_eventq_ptr evq_ptr
- int32_t **evq_clear_stride**
- uint32_t sync_timestamp_major
- uint32_t sync_timestamp_minor
- uint32_t sync_timestamp_minimum
- uint32_t sync_timestamp_synchronised
- uint32_t sync_flags

9.2.1 Detailed Description

State of event queue.

Users should not access this structure.

Definition at line 570 of file ef_vi.h.

9.2.2 Field Documentation

9.2.2.1 evq_ptr

```
ef_eventq_ptr evq_ptr
```

Event queue pointer

Definition at line 572 of file ef_vi.h.

9.2.2.2 sync_flags

```
uint32_t sync_flags
```

Time synchronisation flags

Definition at line 584 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.2.2.3 sync_timestamp_major

```
uint32_t sync_timestamp_major
```

Timestamp (major part)

Definition at line 576 of file ef_vi.h.

9.2.2.4 sync_timestamp_minimum

```
uint32_t sync_timestamp_minimum
```

Smallest possible seconds value for given sync_timestamp_major

Definition at line 580 of file ef_vi.h.

9.2.2.5 sync_timestamp_minor

```
uint32_t sync_timestamp_minor
```

Timestamp (minor part)

Definition at line 578 of file ef_vi.h.

9.2.2.6 sync_timestamp_synchronised

```
uint32_t sync_timestamp_synchronised
```

Timestamp synchronised with adapter

Definition at line 582 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

9.3 ef_filter_cookie Struct Reference

Cookie identifying a filter.

```
#include <vi.h>
```


Copyright © Solarflare Communications 2017

Data Fields

- int filter_id
- int filter_type

9.3.1 Detailed Description

Cookie identifying a filter.

Definition at line 485 of file vi.h.

9.3.2 Field Documentation

9.3.2.1 filter_id

```
int filter_id
```

ID of the filter

Definition at line 487 of file vi.h.

9.3.2.2 filter_type

```
int filter_type
```

Type of the filter

Definition at line 489 of file vi.h.

The documentation for this struct was generated from the following file:

- vi.h

9.4 ef_filter_spec Struct Reference

Specification of a filter.

```
#include <vi.h>
```

 Copyright © Solarflare Communications 2017

Data Fields

- unsigned type
- unsigned flags
- unsigned data [12]

9.4.1 Detailed Description

Specification of a filter.

Definition at line 469 of file vi.h.

9.4.2 Field Documentation

9.4.2.1 data

```
unsigned data[12]
```

Data for filter

Definition at line 475 of file vi.h.

9.4.2.2 flags

```
unsigned flags
```

Flags for filter

Definition at line 473 of file vi.h.

9.4.2.3 type

```
unsigned type
```

Type of filter

Definition at line 471 of file vi.h.

The documentation for this struct was generated from the following file:

- vi.h

Copyright © Solarflare Communications 2017

9.5 ef_iovec Struct Reference

ef_iovec is similar to the standard struct iovec. An array of these is used to designate a scatter/gather list of I/O buffers.

```
#include <ef_vi.h>
```

Public Member Functions

- ef_addr iov_base EF_VI_ALIGN (8)

Data Fields

- unsigned iov_len

9.5.1 Detailed Description

ef_iovec is similar to the standard struct iovec. An array of these is used to designate a scatter/gather list of I/O buffers.

Definition at line 413 of file ef_vi.h.

9.5.2 Member Function Documentation

9.5.2.1 EF_VI_ALIGN()

```
ef_addr iov_base EF_VI_ALIGN (
        8 )
```

base address of the buffer

9.5.3 Field Documentation

9.5.3.1 iov_len

```
unsigned iov_len
```

length of the buffer

Definition at line 417 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

 Copyright © Solarflare Communications 2017

9.6 ef_memreg Struct Reference

Memory that has been registered for use with ef_vi.

```
#include &lt;memreg.h&gt;
```

Data Fields

- ef_addr * mr_dma_addrs
- ef_addr * mr_dma_addrs_base

9.6.1 Detailed Description

Memory that has been registered for use with ef_vi.

Definition at line 46 of file memreg.h.

9.6.2 Field Documentation

9.6.2.1 mr_dma_addrs

```
ef_addr* mr_dma_addrs
```

Addresses of DMA buffers within the reserved system memory

Definition at line 48 of file memreg.h.

9.6.2.2 mr_dma_addrs_base

```
ef_addr* mr_dma_addrs_base
```

Base addresses of reserved system memory

Definition at line 50 of file memreg.h.

The documentation for this struct was generated from the following file:

- memreg.h

9.7 ef_packed_stream_packet Struct Reference

Per-packet meta-data.

```
#include <packedstream.h>
```

Data Fields

- uint16_t ps_next_offset
- uint8_t ps_pkt_start_offset
- uint8_t ps_flags
- uint16_t ps_cap_len
- uint16_t ps_orig_len
- uint32_t ps_ts_sec
- uint32_t ps_ts_nsec

9.7.1 Detailed Description

Per-packet meta-data.

Definition at line 47 of file packedstream.h.

9.7.2 Field Documentation

9.7.2.1 ps_cap_len

```
uint16_t ps_cap_len
```

Number of bytes of packet payload stored.

Definition at line 55 of file packedstream.h.

9.7.2.2 ps_flags

```
uint8_t ps_flags
```

EF_VI_PS_FLAG_* flags.

Definition at line 53 of file packedstream.h.

 Copyright © Solarflare Communications 2017

9.7.2.3 ps_next_offset

```
uint16_t ps_next_offset
```

Offset of next packet from start of this struct.

Definition at line 49 of file packedstream.h.

9.7.2.4 ps_orig_len

```
uint16_t ps_orig_len
```

Length of the frame on the wire.

Definition at line 57 of file packedstream.h.

9.7.2.5 ps_pkt_start_offset

```
uint8_t ps_pkt_start_offset
```

Offset of packet payload from start of this struct.

Definition at line 51 of file packedstream.h.

9.7.2.6 ps_ts_nsec

```
uint32_t ps_ts_nsec
```

Hardware timestamp (nanoseconds).

Definition at line 61 of file packedstream.h.

9.7.2.7 ps_ts_sec

```
uint32_t ps_ts_sec
```

Hardware timestamp (seconds).

Definition at line 59 of file packedstream.h.

The documentation for this struct was generated from the following file:

- packedstream.h

9.8 ef_packed_stream_params Struct Reference

Packed-stream mode parameters.

```
#include <packedstream.h>
```

Data Fields

- int psp_buffer_size
- int psp_buffer_align
- int psp_start_offset
- int psp_max_usable_buffers

9.8.1 Detailed Description

Packed-stream mode parameters.

The application should query these parameters using ef_vi_packed_stream_get_params() to determine buffer size etc.

Definition at line 90 of file packedstream.h.

9.8.2 Field Documentation

9.8.2.1 psp_buffer_align

```
int psp_buffer_align
```

Alignment requirement for start of packed-stream buffers.

Definition at line 95 of file packedstream.h.

9.8.2.2 psp_buffer_size

```
int psp_buffer_size
```

Size of each packed-stream buffer.

Definition at line 92 of file packedstream.h.

 Copyright © Solarflare Communications 2017

9.8.2.3 psp_max_usable_buffers

```
int psp_max_usable_buffers
```

The maximum number of packed-stream buffers that the adapter can deliver packets into without software consuming any completion events. The application can post more buffers, but they will only be used as events are consumed.

Definition at line 107 of file packedstream.h.

9.8.2.4 psp_start_offset

```
int psp_start_offset
```

Offset from the start of a packed-stream buffer to where the first packet will be delivered.

Definition at line 100 of file packedstream.h.

The documentation for this struct was generated from the following file:

- packedstream.h

9.9 ef_pd Struct Reference

A protection domain.

```
#include <pd.h>
```

Data Fields

- enum ef_pd_flags pd_flags
- unsigned pd_resource_id
- char * pd_intf_name
- char * pd_cluster_name
- int pd_cluster_sock
- ef_driver_handle pd_cluster_dh
- unsigned pd_cluster_viset_resource_id
- int pd_cluster_viset_index

9.9.1 Detailed Description

A protection domain.

Definition at line 75 of file pd.h.

9.9.2 Field Documentation

9.9.2.1 pd_cluster_dh

```
ef_driver_handle pd_cluster_dh
```

Driver handle for the application cluster associated with the protection domain

Definition at line 91 of file pd.h.

9.9.2.2 pd_cluster_name

```
char* pd_cluster_name
```

Name of the application cluster associated with the protection domain

Definition at line 85 of file pd.h.

9.9.2.3 pd_cluster_sock

```
int pd_cluster_sock
```

Socket for the application cluster associated with the protection domain

Definition at line 88 of file pd.h.

9.9.2.4 pd_cluster_viset_index

```
int pd_cluster_viset_index
```

Index of VI wanted within a cluster.

Definition at line 96 of file pd.h.

 Copyright © Solarflare Communications 2017

9.9.2.5 pd_cluster_viset_resource_id

```
unsigned pd_cluster_viset_resource_id
```

Resource ID of the virtual interface set for the application cluster associated with the protection domain

Definition at line 94 of file pd.h.

9.9.2.6 pd_flags

```
enum ef_pd_flags pd_flags
```

Flags for the protection domain

Definition at line 77 of file pd.h.

9.9.2.7 pd_intf_name

```
char* pd_intf_name
```

Name of the interface associated with the protection domain

Definition at line 81 of file pd.h.

9.9.2.8 pd_resource_id

```
unsigned pd_resource_id
```

Resource ID of the protection domain

Definition at line 79 of file pd.h.

The documentation for this struct was generated from the following file:

- pd.h

9.10 ef_pio Struct Reference

A Programmed I/O region.

```
#include <pio.h>
```


Copyright © Solarflare Communications 2017

Data Fields

- uint8_t * pio_buffer
- uint8_t * pio_io
- unsigned pio_resource_id
- unsigned pio_len

9.10.1 Detailed Description

A Programmed I/O region.

Definition at line 47 of file pio.h.

9.10.2 Field Documentation

9.10.2.1 pio_buffer

```
uint8_t* pio_buffer
```

The buffer for the Programmed I/O region

Definition at line 49 of file pio.h.

9.10.2.2 pio_io

```
uint8_t* pio_io
```

The I/O region of the virtual interface that is linked with the Programmed I/O region

Definition at line 52 of file pio.h.

9.10.2.3 pio_len

```
unsigned pio_len
```

The length of the Programmed I/O region

Definition at line 56 of file pio.h.

 Copyright © Solarflare Communications 2017

9.10.2.4 pio_resource_id

```
unsigned pio_resource_id
```

The resource ID for the Programmed I/O region

Definition at line 54 of file pio.h.

The documentation for this struct was generated from the following file:

- pio.h

9.11 ef_vi Struct Reference

A virtual interface.

```
#include <ef_vi.h>
```

Data Structures

- struct ops

  Driver-dependent operations.

Data Fields

- unsigned inited
- unsigned vi_resource_id
- unsigned vi_i
- unsigned rx_buffer_len
- unsigned rx_prefix_len
- uint64_t rx_discard_mask
- int rx_ts_correction
- int tx_ts_correction_ns
- char * vi_mem_mmap_ptr
- int vi_mem_mmap_bytes
- char * vi_io_mmap_ptr
- int vi_io_mmap_bytes
- int vi_clustered
- int vi_is_packed_stream
- int vi_is_normal
- unsigned vi_ps_buf_size
- ef_vi_ioaddr_t io
- struct ef_pio * linked_pio
- char * evq_base
- unsigned evq_mask
- unsigned timer_quantum_ns
- unsigned tx_push_thresh
- ef_vi_txq vi_txq

Copyright © Solarflare Communications 2017

- ef_vi_rxq vi_rxq
- ef_vi_state * ep_state
- enum ef_vi_flags vi_flags
- enum ef_vi_out_flags vi_out_flags
- ef_vi_stats * vi_stats
- struct ef_vi * vi_qs [EF_VI_MAX_QS]
- int vi_qs_n
- unsigned tx_alt_num
- unsigned * tx_alt_id2hw
- unsigned * tx_alt_hw2id
- struct ef_vi_nic_type nic_type
- struct ef_vi::ops ops

9.11.1 Detailed Description

A virtual interface.

An ef_vi represents a virtual interface on a specific NIC. A virtual interface is a collection of an event queue and two DMA queues used to pass Ethernet frames between the transport implementation and the network.

Users should not access this structure.

Definition at line 685 of file ef_vi.h.

9.11.2 Field Documentation

9.11.2.1 ep_state

```
ef_vi_state* ep_state
```

The state of the virtual interface

Definition at line 742 of file ef_vi.h.

9.11.2.2 evq_base

```
char* evq_base
```

Base of the event queue for the virtual interface

Definition at line 727 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.11.2.3 evq_mask

```
unsigned evq_mask
```

Mask for offsets within the event queue for the virtual interface

Definition at line 729 of file ef_vi.h.

9.11.2.4 inited

```
unsigned inited
```

True if the virtual interface has been initialized

Definition at line 687 of file ef_vi.h.

9.11.2.5 io

```
ef_vi_ioaddr_t io
```

I/O address for the virtual interface

Definition at line 721 of file ef_vi.h.

9.11.2.6 linked_pio

```
struct ef_pio* linked_pio
```

Programmed I/O region linked to the virtual interface

Definition at line 724 of file ef_vi.h.

9.11.2.7 nic_type

```
struct ef_vi_nic_type nic_type
```

The type of NIC hosting the virtual interface

Definition at line 763 of file ef_vi.h.

Copyright © Solarflare Communications 2017

9.11.2.8 ops

```
struct ef_vi::ops ops
```

Driver-dependent operations.

9.11.2.9 rx_buffer_len

```
unsigned rx_buffer_len
```

The length of a receive buffer

Definition at line 694 of file ef_vi.h.

9.11.2.10 rx_discard_mask

```
uint64_t rx_discard_mask
```

The mask to select which errors cause a discard event

Definition at line 698 of file ef_vi.h.

9.11.2.11 rx_prefix_len

```
unsigned rx_prefix_len
```

The length of the prefix at the start of a received packet

Definition at line 696 of file ef_vi.h.

9.11.2.12 rx_ts_correction

```
int rx_ts_correction
```

The timestamp correction (ticks) for received packets

Definition at line 700 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.11.2.13 timer_quantum_ns

```
unsigned timer_quantum_ns
```

The timer quantum for the virtual interface, in nanoseconds

Definition at line 731 of file ef_vi.h.

9.11.2.14 tx_alt_hw2id

```
unsigned* tx_alt_hw2id
```

Mapping from hardware TX alternative IDs to end-user IDs

Definition at line 760 of file ef_vi.h.

9.11.2.15 tx_alt_id2hw

```
unsigned* tx_alt_id2hw
```

Mapping from end-user TX alternative IDs to hardware IDs

Definition at line 758 of file ef_vi.h.

9.11.2.16 tx_alt_num

```
unsigned tx_alt_num
```

Number of TX alternatives for the virtual interface

Definition at line 756 of file ef_vi.h.

9.11.2.17 tx_push_thresh

```
unsigned tx_push_thresh
```

The threshold at which to switch from using TX descriptor push to using a doorbell

Definition at line 735 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.11.2.18 tx_ts_correction_ns

```
int tx_ts_correction_ns
```

The timestamp correction (ns) for transmitted packets

Definition at line 702 of file ef_vi.h.

9.11.2.19 vi_clustered

```
int vi_clustered
```

True if the virtual interface is in a cluster

Definition at line 712 of file ef_vi.h.

9.11.2.20 vi_flags

```
enum ef_vi_flags vi_flags
```

The flags for the virtual interface

Definition at line 744 of file ef_vi.h.

9.11.2.21 vi_i

```
unsigned vi_i
```

The instance ID of the virtual interface

Definition at line 691 of file ef_vi.h.

9.11.2.22 vi_io_mmap_bytes

```
int vi_io_mmap_bytes
```

Length of virtual interface I/O region

Definition at line 710 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.11.2.23 vi_io_mmap_ptr

```
char* vi_io_mmap_ptr
```

Pointer to virtual interface I/O region

Definition at line 708 of file ef_vi.h.

9.11.2.24 vi_is_normal

```
int vi_is_normal
```

True if no special mode is enabled for the virtual interface

Definition at line 716 of file ef_vi.h.

9.11.2.25 vi_is_packed_stream

```
int vi_is_packed_stream
```

True if packed stream mode is enabled for the virtual interface

Definition at line 714 of file ef_vi.h.

9.11.2.26 vi_mem_mmap_bytes

```
int vi_mem_mmap_bytes
```

Length of virtual interface memory

Definition at line 706 of file ef_vi.h.

9.11.2.27 vi_mem_mmap_ptr

```
char* vi_mem_mmap_ptr
```

Pointer to virtual interface memory

Definition at line 704 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.11.2.28 vi_out_flags

```
enum ef_vi_out_flags vi_out_flags
```

Flags returned when the virtual interface is allocated

Definition at line 746 of file ef_vi.h.

9.11.2.29 vi_ps_buf_size

```
unsigned vi_ps_buf_size
```

The packed stream buffer size for the virtual interface

Definition at line 718 of file ef_vi.h.

9.11.2.30 vi_qs

```
struct ef_vi* vi_qs[EF_VI_MAX_QS]
```

Virtual queues for the virtual interface

Definition at line 751 of file ef_vi.h.

9.11.2.31 vi_qs_n

```
int vi_qs_n
```

Number of virtual queues for the virtual interface

Definition at line 753 of file ef_vi.h.

9.11.2.32 vi_resource_id

```
unsigned vi_resource_id
```

The resource ID of the virtual interface

Definition at line 689 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.11.2.33 vi_rxq

```
ef_vi_rxq vi_rxq
```

The RX descriptor ring for the virtual interface

Definition at line 740 of file ef_vi.h.

9.11.2.34 vi_stats

```
ef_vi_stats* vi_stats
```

Statistics for the virtual interface

Definition at line 748 of file ef_vi.h.

9.11.2.35 vi_txq

```
ef_vi_txq vi_txq
```

The TX descriptor ring for the virtual interface

Definition at line 738 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

9.12 ef_vi_layout_entry Struct Reference

Layout of the data that is delivered into receive buffers.

```
#include <ef_vi.h>
```

Data Fields

- enum ef_vi_layout_type evle_type
- int evle_offset
- const char * evle_description

 Copyright © Solarflare Communications 2017

9.12.1 Detailed Description

Layout of the data that is delivered into receive buffers.

Definition at line 1913 of file ef_vi.h.

9.12.2 Field Documentation

9.12.2.1 evle_description

```
const char* evle_description
```

Description of the layout

Definition at line 1919 of file ef_vi.h.

9.12.2.2 evle_offset

```
int evle_offset
```

Offset to the data

Definition at line 1917 of file ef_vi.h.

9.12.2.3 evle_type

```
enum ef_vi_layout_type evle_type
```

The type of layout

Definition at line 1915 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

9.13 ef_vi_nic_type Struct Reference

The type of NIC in use.

```
#include <ef_vi.h>
```

 Copyright © Solarflare Communications 2017

Data Fields

- unsigned char arch
- char variant
- unsigned char revision

9.13.1 Detailed Description

The type of NIC in use.

Users should not access this structure.

Definition at line 646 of file ef_vi.h.

9.13.2 Field Documentation

9.13.2.1 arch

```
unsigned char arch
```

Architecture of the NIC

Definition at line 648 of file ef_vi.h.

9.13.2.2 revision

```
unsigned char revision
```

Revision of the NIC

Definition at line 652 of file ef_vi.h.

9.13.2.3 variant

```
char variant
```

Variant of the NIC

Definition at line 650 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

Copyright © Solarflare Communications 2017

9.14 ef_vi_rxq Struct Reference

RX descriptor ring.

```
#include <ef_vi.h>
```

Data Fields

- uint32_t mask
- void * descriptors
- uint32_t * ids

9.14.1 Detailed Description

RX descriptor ring.

Users should not access this structure.

Definition at line 604 of file ef_vi.h.

9.14.2 Field Documentation

9.14.2.1 descriptors

```
void* descriptors
```

Pointer to descriptors

Definition at line 608 of file ef_vi.h.

9.14.2.2 ids

```
uint32_t* ids
```

Pointer to IDs

Definition at line 610 of file ef_vi.h.

Copyright © Solarflare Communications 2017

9.14.2.3 mask

```
uint32_t mask
```

Mask for indexes within ring, to wrap around

Definition at line 606 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

9.15 ef_vi_rxq_state Struct Reference

State of RX descriptor ring.

```
#include <ef_vi.h>
```

Data Fields

- uint32_t posted
- uint32_t added
- uint32_t removed
- uint32_t in_jumbo
- uint32_t bytes_acc
- uint16_t last_desc_i
- uint16_t rx_ps_credit_avail

9.15.1 Detailed Description

State of RX descriptor ring.

Users should not access this structure.

Definition at line 549 of file ef_vi.h.

9.15.2 Field Documentation

9.15.2.1 added

```
uint32_t added
```

Descriptors added to the ring

Definition at line 553 of file ef_vi.h.

9.15.2.2 bytes_acc

`uint32_t bytes_acc`

Bytes received as part of a jumbo (7000-series only)

Definition at line 559 of file ef_vi.h.

9.15.2.3 in_jumbo

`uint32_t in_jumbo`

Packets received as part of a jumbo (7000-series only)

Definition at line 557 of file ef_vi.h.

9.15.2.4 last_desc_i

`uint16_t last_desc_i`

Last descriptor index completed (7000-series only)

Definition at line 561 of file ef_vi.h.

9.15.2.5 posted

`uint32_t posted`

Descriptors posted to the nic

Definition at line 551 of file ef_vi.h.

9.15.2.6 removed

`uint32_t removed`

Descriptors removed from the ring

Definition at line 555 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.15.2.7 rx_ps_credit_avail

```
uint16_t rx_ps_credit_avail
```

Credit for packed stream handling (7000-series only)

Definition at line 563 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

9.16 ef_vi_set Struct Reference

A virtual interface set within a protection domain.

```
#include <vi.h>
```

Data Fields

- unsigned vis_res_id
- struct ef_pd * vis_pd

9.16.1 Detailed Description

A virtual interface set within a protection domain.

Definition at line 328 of file vi.h.

9.16.2 Field Documentation

9.16.2.1 vis_pd

```
struct ef_pd* vis_pd
```

Protection domain from which the virtual interface set is allocated

Definition at line 332 of file vi.h.

Copyright © Solarflare Communications 2017

9.16.2.2 vis_res_id

```
unsigned vis_res_id
```

Resource ID for the virtual interface set

Definition at line 330 of file vi.h.

The documentation for this struct was generated from the following file:

- vi.h

9.17 ef_vi_state Struct Reference

State of a virtual interface.

```
#include <ef_vi.h>
```

Data Fields

- ef_eventq_state evq
- ef_vi_txq_state txq
- ef_vi_rxq_state rxq

9.17.1 Detailed Description

State of a virtual interface.

Users should not access this structure.

Definition at line 617 of file ef_vi.h.

9.17.2 Field Documentation

9.17.2.1 evq

```
ef_eventq_state evq
```

Event queue state

Definition at line 619 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.17.2.2 rxq

`ef_vi_rxq_state rxq`

RX descriptor ring state

Definition at line 623 of file ef_vi.h.

9.17.2.3 txq

`ef_vi_txq_state txq`

TX descriptor ring state

Definition at line 621 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

9.18 ef_vi_stats Struct Reference

Statistics for a virtual interface.

```
#include <ef_vi.h>
```

Data Fields

- uint32_t rx_ev_lost
- uint32_t rx_ev_bad_desc_i
- uint32_t rx_ev_bad_q_label
- uint32_t evq_gap

9.18.1 Detailed Description

Statistics for a virtual interface.

Users should not access this structure.

Definition at line 631 of file ef_vi.h.

9.18.2 Field Documentation

Copyright © Solarflare Communications 2017

9.18.2.1 evq_gap

```
uint32_t evq_gap
```

Gaps in the event queue (empty slot followed by event)

Definition at line 639 of file ef_vi.h.

9.18.2.2 rx_ev_bad_desc_i

```
uint32_t rx_ev_bad_desc_i
```

RX events with a bad descriptor

Definition at line 635 of file ef_vi.h.

9.18.2.3 rx_ev_bad_q_label

```
uint32_t rx_ev_bad_q_label
```

RX events with a bad queue label

Definition at line 637 of file ef_vi.h.

9.18.2.4 rx_ev_lost

```
uint32_t rx_ev_lost
```

RX events lost

Definition at line 633 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

9.19 ef_vi_stats_field_layout Struct Reference

Layout for a field of statistics.

```
#include <vi.h>
```

 Copyright © Solarflare Communications 2017

Data Fields

- char * evsfl_name
- int evsfl_offset
- int evsfl_size

9.19.1 Detailed Description

Layout for a field of statistics.

Definition at line 994 of file vi.h.

9.19.2 Field Documentation

9.19.2.1 evsfl_name

```
char* evsfl_name
```

Name of statistics field

Definition at line 996 of file vi.h.

9.19.2.2 evsfl_offset

```
int evsfl_offset
```

Offset of statistics fiel, in bytesd

Definition at line 998 of file vi.h.

9.19.2.3 evsfl_size

```
int evsfl_size
```

Size of statistics field, in bytes

Definition at line 1000 of file vi.h.

The documentation for this struct was generated from the following file:

- vi.h

Copyright © Solarflare Communications 2017

9.20 ef_vi_stats_layout Struct Reference

Layout for statistics.

```
#include <vi.h>
```

Data Fields

- int evsl_data_size
- int evsl_fields_num
- ef_vi_stats_field_layout evsl_fields [ ]

9.20.1 Detailed Description

Layout for statistics.

Definition at line 1004 of file vi.h.

9.20.2 Field Documentation

9.20.2.1 evsl_data_size

```
int evsl_data_size
```

Size of data for statistics

Definition at line 1006 of file vi.h.

9.20.2.2 evsl_fields

```
ef_vi_stats_field_layout evsl_fields[ ]
```

Array of fields of statistics

Definition at line 1010 of file vi.h.

 Copyright © Solarflare Communications 2017

9.20.2.3 evsl_fields_num

```
int evsl_fields_num
```

Number of fields of statistics

Definition at line 1008 of file vi.h.

The documentation for this struct was generated from the following file:

- vi.h

9.21 ef_vi_transmit_alt_overhead Struct Reference

Per-packet overhead information.

```
#include <ef_vi.h>
```

Data Fields

- uint32_t pre_round
- uint32_t mask
- uint32_t post_round

9.21.1 Detailed Description

Per-packet overhead information.

This structure is used by ef_vi_transmit_alt_usage() to calculate the amount of buffering needed to store a packet. It should be filled in by ef_vi_transmit_alt_query_overhead() or similar. Its members are not intended to be meaningful to the application and should not be obtained or interpreted in any other way.

Definition at line 663 of file ef_vi.h.

9.21.2 Field Documentation

9.21.2.1 mask

```
uint32_t mask
```

Rounding mask

Definition at line 667 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.21.2.2 post_round

```
uint32_t post_round
```

Bytes to add after rounding

Definition at line 669 of file ef_vi.h.

9.21.2.3 pre_round

```
uint32_t pre_round
```

Bytes to add before rounding

Definition at line 665 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

9.22 ef_vi_txq Struct Reference

TX descriptor ring.

```
#include <ef_vi.h>
```

Data Fields

- uint32_t mask
- void * descriptors
- uint32_t * ids

9.22.1 Detailed Description

TX descriptor ring.

Users should not access this structure.

Definition at line 591 of file ef_vi.h.

9.22.2 Field Documentation

 Copyright © Solarflare Communications 2017

9.22.2.1 descriptors

```
void* descriptors
```

Pointer to descriptors

Definition at line 595 of file ef_vi.h.

9.22.2.2 ids

```
uint32_t* ids
```

Pointer to IDs

Definition at line 597 of file ef_vi.h.

9.22.2.3 mask

```
uint32_t mask
```

Mask for indexes within ring, to wrap around

Definition at line 593 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

9.23 ef_vi_txq_state Struct Reference

State of TX descriptor ring.

```
#include <ef_vi.h>
```

Data Fields

- uint32_t previous
- uint32_t added
- uint32_t removed
- uint32_t ts_nsec

 Copyright © Solarflare Communications 2017

9.23.1 Detailed Description

State of TX descriptor ring.

Users should not access this structure.

Definition at line 534 of file ef_vi.h.

9.23.2 Field Documentation

9.23.2.1 added

```
uint32_t added
```

Descriptors added to the ring

Definition at line 538 of file ef_vi.h.

9.23.2.2 previous

```
uint32_t previous
```

Previous slot that has been handled

Definition at line 536 of file ef_vi.h.

9.23.2.3 removed

```
uint32_t removed
```

Descriptors removed from the ring

Definition at line 540 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.23.2.4 ts_nsec

```
uint32_t ts_nsec
```

Timestamp in nanoseconds

Definition at line 542 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

9.24 ef_vi::ops Struct Reference

Driver-dependent operations.

```
#include <ef_vi.h>
```

Data Fields

- int(* transmit)(struct ef_vi *, ef_addr base, int len, ef_request_id)
- int(* transmitv)(struct ef_vi *, const ef_iovec *, int iov_len, ef_request_id)
- int(* transmitv_init)(struct ef_vi *, const ef_iovec *, int iov_len, ef_request_id)
- void(* transmit_push)(struct ef_vi *)
- int(* transmit_pio)(struct ef_vi *, int offset, int len, ef_request_id dma_id)
- int(* transmit_copy_pio)(struct ef_vi *, int pio_offset, const void *src_buf, int len, ef_request_id dma_id)
- void(* transmit_pio_warm)(struct ef_vi *)
- void(* transmit_copy_pio_warm)(struct ef_vi *, int pio_offset, const void *src_buf, int len)
- int(* transmit_alt_select)(struct ef_vi *, unsigned alt_id)
- int(* transmit_alt_select_default)(struct ef_vi *)
- int(* transmit_alt_stop)(struct ef_vi *, unsigned alt_id)
- int(* transmit_alt_go)(struct ef_vi *, unsigned alt_id)
- int(* transmit_alt_discard)(struct ef_vi *, unsigned alt_id)
- int(* receive_init)(struct ef_vi *, ef_addr, ef_request_id)
- void(* receive_push)(struct ef_vi *)
- int(* eventq_poll)(struct ef_vi *, ef_event *, int evs_len)
- void(* eventq_prime)(struct ef_vi *)
- void(* eventq_timer_prime)(struct ef_vi *, unsigned v)
- void(* eventq_timer_run)(struct ef_vi *, unsigned v)
- void(* eventq_timer_clear)(struct ef_vi *)
- void(* eventq_timer_zero)(struct ef_vi *)

9.24.1 Detailed Description

Driver-dependent operations.

Definition at line 767 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.24.2 Field Documentation

9.24.2.1 eventq_poll

```
int(* eventq_poll) (struct ef_vi *, ef_event *, int evs_len)
```

Poll an event queue

Definition at line 807 of file ef_vi.h.

9.24.2.2 eventq_prime

```
void(* eventq_prime) (struct ef_vi *)
```

Prime a virtual interface allowing you to go to sleep blocking on it

Definition at line 809 of file ef_vi.h.

9.24.2.3 eventq_timer_clear

```
void(* eventq_timer_clear) (struct ef_vi *)
```

Stop an event-queue timer

Definition at line 815 of file ef_vi.h.

9.24.2.4 eventq_timer_prime

```
void(* eventq_timer_prime) (struct ef_vi *, unsigned v)
```

Prime an event queue timer with a new timeout

Definition at line 811 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.24.2.5 eventq_timer_run

```
void(* eventq_timer_run) (struct ef_vi *, unsigned v)
```

Start an event queue timer running

Definition at line 813 of file ef_vi.h.

9.24.2.6 eventq_timer_zero

```
void(* eventq_timer_zero) (struct ef_vi *)
```

Prime an event queue timer to expire immediately

Definition at line 817 of file ef_vi.h.

9.24.2.7 receive_init

```
int(* receive_init) (struct ef_vi *, ef_addr, ef_request_id)
```

Initialize an RX descriptor on the RX descriptor ring

Definition at line 803 of file ef_vi.h.

9.24.2.8 receive_push

```
void(* receive_push) (struct ef_vi *)
```

Submit newly initialized RX descriptors to the NIC

Definition at line 805 of file ef_vi.h.

9.24.2.9 transmit

```
int(* transmit) (struct ef_vi *, ef_addr base, int len, ef_request_id)
```

Transmit a packet from a single packet buffer

Definition at line 769 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.24.2.10 transmit_alt_discard

```
int(* transmit_alt_discard) (struct ef_vi *, unsigned alt_id)
```

Transition a TX alternative to the DISCARD state

Definition at line 801 of file ef_vi.h.

9.24.2.11 transmit_alt_go

```
int(* transmit_alt_go) (struct ef_vi *, unsigned alt_id)
```

Transition a TX alternative to the GO state

Definition at line 799 of file ef_vi.h.

9.24.2.12 transmit_alt_select

```
int(* transmit_alt_select) (struct ef_vi *, unsigned alt_id)
```

Select a TX alternative as the destination for future sends

Definition at line 793 of file ef_vi.h.

9.24.2.13 transmit_alt_select_default

```
int(* transmit_alt_select_default) (struct ef_vi *)
```

Select the "normal" data path as the destination for future sends

Definition at line 795 of file ef_vi.h.

9.24.2.14 transmit_alt_stop

```
int(* transmit_alt_stop) (struct ef_vi *, unsigned alt_id)
```

Transition a TX alternative to the STOP state

Definition at line 797 of file ef_vi.h.

9.24.2.15 transmit_copy_pio

```
int(* transmit_copy_pio) (struct ef_vi *, int pio_offset, const void *src_buf, int len, ef_request_id dma_id)
```

Copy a packet to Programmed I/O region and transmit it

Definition at line 784 of file ef_vi.h.

9.24.2.16 transmit_copy_pio_warm

```
void(* transmit_copy_pio_warm) (struct ef_vi *, int pio_offset, const void *src_buf, int len)
```

Copy a packet to Programmed I/O region and warm transmit path

Definition at line 790 of file ef_vi.h.

9.24.2.17 transmit_pio

```
int(* transmit_pio) (struct ef_vi *, int offset, int len, ef_request_id dma_id)
```

Transmit a packet already resident in Programmed I/O

Definition at line 781 of file ef_vi.h.

9.24.2.18 transmit_pio_warm

```
void(* transmit_pio_warm) (struct ef_vi *)
```

Warm Programmed I/O transmit path for subsequent transmit

Definition at line 788 of file ef_vi.h.

9.24.2.19 transmit_push

```
void(* transmit_push) (struct ef_vi *)
```

Submit newly initialized TX descriptors to the NIC

Definition at line 779 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

9.24.2.20 transmitv

```
int(* transmitv) (struct ef_vi *, const ef_iovec *, int iov_len, ef_request_id)
```

Transmit a packet from a vector of packet buffers

Definition at line 772 of file ef_vi.h.

9.24.2.21 transmitv_init

```
int(* transmitv_init) (struct ef_vi *, const ef_iovec *, int iov_len, ef_request_id)
```

Initialize TX descriptors on the TX descriptor ring, for a vector of packet buffers

Definition at line 776 of file ef_vi.h.

The documentation for this struct was generated from the following file:

- ef_vi.h

 Copyright © Solarflare Communications 2017

Chapter 10

File Documentation

10.1 000_main.dox File Reference

Additional Doxygen-format documentation for ef_vi.

10.1.1 Detailed Description

Additional Doxygen-format documentation for ef_vi.

Author

Solarflare Communications, Inc.

Date

2016/06/13

Copyright

Copyright © 2016 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.2 010_overview.dox File Reference

Additional Doxygen-format documentation for ef_vi.

10.2.1 Detailed Description

Additional Doxygen-format documentation for ef_vi.

Author

Solarflare Communications, Inc.

Date

2016/06/13

Copyright

Copyright © 2016 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.3 020_concepts.dox File Reference

Additional Doxygen-format documentation for ef_vi.

10.3.1 Detailed Description

Additional Doxygen-format documentation for ef_vi.

Author

Solarflare Communications, Inc.

Date

2016/06/13

Copyright

Copyright © 2016 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.4 030_apps.dox File Reference

Additional Doxygen-format documentation for ef_vi.

10.4.1 Detailed Description

Additional Doxygen-format documentation for ef_vi.

Author

Solarflare Communications, Inc.

Date

2016/06/13

Copyright

Copyright © 2016 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.5 040_using.dox File Reference

Additional Doxygen-format documentation for ef_vi.

10.5.1 Detailed Description

Additional Doxygen-format documentation for ef_vi.

Author

Solarflare Communications, Inc.

Date

2017/02/21

Copyright

Copyright © 2017 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.6 050_examples.dox File Reference

Additional Doxygen-format documentation for ef_vi.

10.6.1 Detailed Description

Additional Doxygen-format documentation for ef_vi.

Author

Solarflare Communications, Inc.

Date

2016/06/13

Copyright

Copyright © 2016 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.7 base.h File Reference

Base definitions for EtherFabric Virtual Interface HAL.

```
#include <etherfabric/ef_vi.h>
```

Macros

- #define EF_VI_NIC_PAGE_SHIFT 12

  How much to shift an address to get the page number.
- #define EF_VI_NIC_PAGE_SIZE (1<<EF_VI_NIC_PAGE_SHIFT)

  The size of a page of memory on the NIC, in bytes.
- #define EF_ADDR_FMT "%" CI_PRIx64

  Format for outputting an ef_addr.
- #define EF_INVALID_ADDR ((ef_addr) -1)

  An address that is always invalid.

Typedefs

- typedef int ef_driver_handle

  An ef_driver_handle is needed to allocate resources.

Functions

- int ef_eventq_wait (ef_vi *vi, ef_driver_handle vi_dh, unsigned current_ptr, const struct timeval *timeout)

  Block, waiting until the event queue is non-empty.
- int ef_driver_open (ef_driver_handle *dh_out)

  Obtain a driver handle.
- int ef_driver_close (ef_driver_handle dh)

  Close a driver handle.

10.7.1 Detailed Description

Base definitions for EtherFabric Virtual Interface HAL.

Author

Solarflare Communications, Inc.

Date

2017/02/21

Copyright

Copyright © 2017 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.7.2 Function Documentation

10.7.2.1 ef_driver_close()

```
int ef_driver_close (
            ef_driver_handle dh )
```

Close a driver handle.

Parameters

dh	The handle to the driver to close.

Returns

0 on success, or a negative error code.

Close a driver handle.

This should be called to free up resources when the driver handle is no longer needed, but the application is to contimue running.

Any associated virtual interface, protection domain, or driver structures must not be used after this call has been made.

Note

Resources are also freed when the application exits, and so this function does not need to be called on exit.

10.7.2.2 ef_driver_open()

```
int ef_driver_open (
            ef_driver_handle * dh_out )
```

Obtain a driver handle.

Parameters

dh_out	Pointer to an ef_driver_handle, that is updated on return with the new driver handle.

Returns

0 on success, or a negative error code.

Obtain a driver handle.

10.7.2.3 ef_eventq_wait()

```
int ef_eventq_wait (
            ef_vi * vi,
            ef_driver_handle vi_dh,
            unsigned current_ptr,
            const struct timeval * timeout )
```

Block, waiting until the event queue is non-empty.

Parameters

vi	The virtual interface on which to wait.
vi_dh	Driver handle associated with the virtual interface.
current_ptr	Must come from ef_eventq_current().
timeout	Maximum time to wait for, or 0 to wait forever.

Returns

0 on success, or a negative error code:
-ETIMEDOUT on time-out).

Block, waiting until the event queue is non-empty. This enables interrupts.

Note that when this function returns it is not guaranteed that an event will be present in the event queue, but in most cases there will be.

10.8 capabilities.h File Reference

Capabilities API for EtherFabric Virtual Interface HAL.

```
#include <etherfabric/base.h>
```

 Copyright © Solarflare Communications 2017

Enumerations

- enum ef_vi_capability {
EF_VI_CAP_PIO = 0, EF_VI_CAP_PIO_BUFFER_SIZE, EF_VI_CAP_PIO_BUFFER_COUNT, EF_VI_C↩AP_HW_MULTICAST_LOOPBACK,
EF_VI_CAP_HW_MULTICAST_REPLICATION, EF_VI_CAP_HW_RX_TIMESTAMPING, EF_VI_CAP_H↩W_TX_TIMESTAMPING, EF_VI_CAP_PACKED_STREAM,
EF_VI_CAP_PACKED_STREAM_BUFFER_SIZES, EF_VI_CAP_VPORTS, EF_VI_CAP_PHYS_MODE, EF_VI_CAP_BUFFER_MODE,
EF_VI_CAP_MULTICAST_FILTER_CHAINING, EF_VI_CAP_MAC_SPOOFING, EF_VI_CAP_RX_FILTE↩R_TYPE_UDP_LOCAL, EF_VI_CAP_RX_FILTER_TYPE_TCP_LOCAL,
EF_VI_CAP_RX_FILTER_TYPE_UDP_FULL, EF_VI_CAP_RX_FILTER_TYPE_TCP_FULL, EF_VI_CAP↩_RX_FILTER_TYPE_IP_VLAN, EF_VI_CAP_RX_FILTER_TYPE_UDP6_LOCAL,
EF_VI_CAP_RX_FILTER_TYPE_TCP6_LOCAL, EF_VI_CAP_RX_FILTER_TYPE_UDP6_FULL, EF_VI_↩CAP_RX_FILTER_TYPE_TCP6_FULL, EF_VI_CAP_RX_FILTER_TYPE_IP6_VLAN,
EF_VI_CAP_RX_FILTER_TYPE_ETH_LOCAL, EF_VI_CAP_RX_FILTER_TYPE_ETH_LOCAL_VLAN, E↩F_VI_CAP_RX_FILTER_TYPE_UCAST_ALL, EF_VI_CAP_RX_FILTER_TYPE_MCAST_ALL,
EF_VI_CAP_RX_FILTER_TYPE_UCAST_MISMATCH, EF_VI_CAP_RX_FILTER_TYPE_MCAST_MISM↩ATCH, EF_VI_CAP_RX_FILTER_TYPE_SNIFF, EF_VI_CAP_TX_FILTER_TYPE_SNIFF,
EF_VI_CAP_RX_FILTER_IP4_PROTO, EF_VI_CAP_RX_FILTER_ETHERTYPE, EF_VI_CAP_RXQ_SIZ↩ES, EF_VI_CAP_TXQ_SIZES,
EF_VI_CAP_EVQ_SIZES, EF_VI_CAP_ZERO_RX_PREFIX, EF_VI_CAP_TX_PUSH_ALWAYS, EF_VI_↩CAP_NIC_PACE,
EF_VI_CAP_RX_MERGE, EF_VI_CAP_TX_ALTERNATIVES, EF_VI_CAP_TX_ALTERNATIVES_VFIFOS, EF_VI_CAP_TX_ALTERNATIVES_CP_BUFFERS,
EF_VI_CAP_RX_FW_VARIANT, EF_VI_CAP_TX_FW_VARIANT, EF_VI_CAP_MAX }

  Possible capabilities.

Functions

- int ef_vi_capabilities_get (ef_driver_handle handle, int ifindex, enum ef_vi_capability cap, unsigned long ∗value)

  Get the value of the given capability.
- int ef_vi_capabilities_max (void)

  Gets the maximum supported value of ef_vi_capability.
- const char ∗ ef_vi_capabilities_name (enum ef_vi_capability cap)

  Gets a human-readable string describing the given capability.

10.8.1 Detailed Description

Capabilities API for EtherFabric Virtual Interface HAL.

Author

Solarflare Communications, Inc.

Date

2015/06/08

Copyright

Copyright © 2015 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.8.2 Enumeration Type Documentation

10.8.2.1 ef_vi_capability

```
enum ef_vi_capability
```

Possible capabilities.

Enumerator

EF_VI_CAP_PIO	Hardware capable of PIO
EF_VI_CAP_PIO_BUFFER_SIZE	PIO buffer size supplied to each VI
EF_VI_CAP_PIO_BUFFER_COUNT	Total number of PIO buffers
EF_VI_CAP_HW_MULTICAST_LOOPBACK	Can packets be looped back by hardware
EF_VI_CAP_HW_MULTICAST_REPLICATION	Can mcast be delivered to many VIs
EF_VI_CAP_HW_RX_TIMESTAMPING	Hardware timestamping of received packets
EF_VI_CAP_HW_TX_TIMESTAMPING	Hardware timestamping of transmitted packets
EF_VI_CAP_PACKED_STREAM	Is firmware capable of packed stream mode
EF_VI_CAP_PACKED_STREAM_BUFFER_SIZES	Packed stream buffer sizes supported in kB, bitmask
EF_VI_CAP_VPORTS	NIC switching, ef_pd_alloc_with_vport
EF_VI_CAP_PHYS_MODE	Is physical addressing mode supported?
EF_VI_CAP_BUFFER_MODE	Is buffer addressing mode (NIC IOMMU) supported
EF_VI_CAP_MULTICAST_FILTER_CHAINING	Chaining of multicast filters
EF_VI_CAP_MAC_SPOOFING	Can functions create filters for 'wrong' MAC addr
EF_VI_CAP_RX_FILTER_TYPE_UDP_LOCAL	Filter on local IP + UDP port
EF_VI_CAP_RX_FILTER_TYPE_TCP_LOCAL	Filter on local IP + TCP port
EF_VI_CAP_RX_FILTER_TYPE_UDP_FULL	Filter on local and remote IP + UDP port
EF_VI_CAP_RX_FILTER_TYPE_TCP_FULL	Filter on local and remote IP + TCP port
EF_VI_CAP_RX_FILTER_TYPE_IP_VLAN	Filter on any of above four types with addition of VLAN
EF_VI_CAP_RX_FILTER_TYPE_UDP6_LOCAL	Filter on local IP + UDP port
EF_VI_CAP_RX_FILTER_TYPE_TCP6_LOCAL	Filter on local IP + TCP port
EF_VI_CAP_RX_FILTER_TYPE_UDP6_FULL	Filter on local and remote IP + UDP port
EF_VI_CAP_RX_FILTER_TYPE_TCP6_FULL	Filter on local and remote IP + TCP port
EF_VI_CAP_RX_FILTER_TYPE_IP6_VLAN	Filter on any of above four types with addition of VLAN
EF_VI_CAP_RX_FILTER_TYPE_ETH_LOCAL	Filter on local MAC address
EF_VI_CAP_RX_FILTER_TYPE_ETH_LOCAL_VLAN	Filter on local MAC+VLAN
EF_VI_CAP_RX_FILTER_TYPE_UCAST_ALL	Filter on "all unicast"
EF_VI_CAP_RX_FILTER_TYPE_MCAST_ALL	Filter on "all multicast"
EF_VI_CAP_RX_FILTER_TYPE_UCAST_MISMATCH	Filter on "unicast mismatch"
EF_VI_CAP_RX_FILTER_TYPE_MCAST_MISMATCH	Filter on "multicast mismatch"
EF_VI_CAP_RX_FILTER_TYPE_SNIFF	Availability of RX sniff filters
EF_VI_CAP_TX_FILTER_TYPE_SNIFF	Availability of TX sniff filters
EF_VI_CAP_RX_FILTER_IP4_PROTO	Filter on IPv4 protocol
EF_VI_CAP_RX_FILTER_ETHERTYPE	Filter on ethertype
EF_VI_CAP_RXQ_SIZES	Available RX queue sizes, bitmask

Copyright © Solarflare Communications 2017

Enumerator

EF_VI_CAP_TXQ_SIZES	Available TX queue sizes, bitmask
EF_VI_CAP_EVQ_SIZES	Available event queue sizes, bitmask
EF_VI_CAP_ZERO_RX_PREFIX	Availability of zero length RX packet prefix
EF_VI_CAP_TX_PUSH_ALWAYS	Is always enabling TX push supported?
EF_VI_CAP_NIC_PACE	Availability of NIC pace feature
EF_VI_CAP_RX_MERGE	Availability of RX event merging mode
EF_VI_CAP_TX_ALTERNATIVES	Availability of TX alternatives
EF_VI_CAP_TX_ALTERNATIVES_VFIFOS	Number of TX alternatives vFIFOs
EF_VI_CAP_TX_ALTERNATIVES_CP_BUFFERS	Number of TX alternatives common pool buffers
EF_VI_CAP_RX_FW_VARIANT	RX firmware variant
EF_VI_CAP_TX_FW_VARIANT	TX firmware variant
EF_VI_CAP_MAX	Maximum value of capabilities enumeration

Definition at line 62 of file capabilities.h.

10.8.3 Function Documentation

10.8.3.1 ef_vi_capabilities_get()

```
int ef_vi_capabilities_get (
            ef_driver_handle handle,
            int ifindex,
            enum ef_vi_capability cap,
            unsigned long * value )
```

Get the value of the given capability.

Parameters

handle	The ef_driver_handle associated with the interface that you wish to query.
ifindex	The index of the interface that you wish to query. You can use if_nametoindex() to obtain this. This should be the underlying physical interface, rather than a bond, VLAN, or similar.
cap	The capability to get.
value	Pointer to location at which to store the value.

Returns

0 on success (capability is supported and value field is updated), or a negative error code:
-EOPNOTSUPP if the capability is not supported
-ENOSYS if the API does not know how to retreive support for the supplied capability
other negative error if support could not be retrieved

 Copyright © Solarflare Communications 2017

10.8.3.2 ef_vi_capabilities_max()

```
int ef_vi_capabilities_max (
            void  )
```

Gets the maximum supported value of ef_vi_capability.

Returns

The maximum capability value, or a negative error code.

This function returns the maximum supported value of ef_vi_capability, so that all capabilities can be iterated. For example:

```
max = ef_vi_capabilities_max();
for( cap = 0; cap <= max; ++cap ) {
  rc = ef_vi_capabilities_get(driver_handle, ifindex, cap, &val);
  if( rc == 0 ) printf("%s %d\n", ef_vi_capabilities_name(cap), val);
}
```

10.8.3.3 ef_vi_capabilities_name()

```
const char* ef_vi_capabilities_name (
            enum ef_vi_capability cap )
```

Gets a human-readable string describing the given capability.

Parameters

cap	The capability for which to get the description.

Returns

A string describing the capability, or a negative error code.

10.9 ef_vi.h File Reference

Virtual Interface definitions for EtherFabric Virtual Interface HAL.

Data Structures

- union ef_event

    A token that identifies something that has happened.
- struct ef_iovec

Copyright © Solarflare Communications 2017

ef_iovec is similar to the standard struct iovec. An array of these is used to designate a scatter/gather list of I/O buffers.

- struct ef_vi_txq_state

  State of TX descriptor ring.
- struct ef_vi_rxq_state

  State of RX descriptor ring.
- struct ef_eventq_state

  State of event queue.
- struct ef_vi_txq

  TX descriptor ring.
- struct ef_vi_rxq

  RX descriptor ring.
- struct ef_vi_state

  State of a virtual interface.
- struct ef_vi_stats

  Statistics for a virtual interface.
- struct ef_vi_nic_type

  The type of NIC in use.
- struct ef_vi_transmit_alt_overhead

  Per-packet overhead information.
- struct ef_vi

  A virtual interface.
- struct ef_vi::ops

  Driver-dependent operations.
- struct ef_vi_layout_entry

  Layout of the data that is delivered into receive buffers.

Macros

- #define EF_VI_DMA_ALIGN 64

  Cache line sizes for alignment purposes.
- #define EF_VI_MAX_QS 32

  The maximum number of queues per virtual interface.
- #define EF_VI_EVENT_POLL_MIN_EVS 2

  The minimum size of array to pass when polling the event queue.
- #define EF_REQUEST_ID_MASK 0xffffffff

  Mask to use with an ef_request_id.
- #define EF_EVENT_TYPE(e) ((e).generic.type)

  Type of event in an ef_event e.
- #define EF_EVENT_RX_BYTES(e) ((e).rx.len)

  Get the number of bytes received.
- #define EF_EVENT_RX_Q_ID(e) ((e).rx.q_id)

  Get the RX descriptor ring ID used for a received packet.
- #define EF_EVENT_RX_RQ_ID(e) ((e).rx.rq_id)

  Get the dma_id used for a received packet.
- #define EF_EVENT_RX_CONT(e) ((e).rx.flags & EF_EVENT_FLAG_CONT)

  True if the CONTinuation Of Packet flag is set for an RX event.
- #define EF_EVENT_RX_SOP(e) ((e).rx.flags & EF_EVENT_FLAG_SOP)

Copyright © Solarflare Communications 2017

True if the Start Of Packet flag is set for an RX event.
- #define EF_EVENT_RX_PS_NEXT_BUFFER(e)

  True if the next buffer flag is set for a packed stream event.
- #define EF_EVENT_RX_ISCSI_OKAY(e) ((e).rx.flags & EF_EVENT_FLAG_ISCSI_OK)

  True if the iSCSIOK flag is set for an RX event.
- #define EF_EVENT_FLAG_SOP 0x1

  Start Of Packet flag.
- #define EF_EVENT_FLAG_CONT 0x2

  CONTinuation Of Packet flag.
- #define EF_EVENT_FLAG_ISCSI_OK 0x4

  iSCSI CRC validated OK flag.
- #define EF_EVENT_FLAG_MULTICAST 0x8

  Multicast flag.
- #define EF_EVENT_FLAG_PS_NEXT_BUFFER 0x10

  Packed Stream Next Buffer flag.
- #define EF_EVENT_TX_Q_ID(e) ((e).tx.q_id)

  Get the TX descriptor ring ID used for a transmitted packet.
- #define EF_EVENT_RX_DISCARD_Q_ID(e) ((e).rx_discard.q_id)

  Get the RX descriptor ring ID used for a discarded packet.
- #define EF_EVENT_RX_DISCARD_RQ_ID(e) ((e).rx_discard.rq_id)

  Get the dma_id used for a discarded packet.
- #define EF_EVENT_RX_DISCARD_CONT(e) ((e).rx_discard.flags&EF_EVENT_FLAG_CONT)

  True if the CONTinuation Of Packet flag is set for an RX_DISCARD event.
- #define EF_EVENT_RX_DISCARD_SOP(e) ((e).rx_discard.flags&EF_EVENT_FLAG_SOP)

  True if the Start Of Packet flag is set for an RX_DISCARD event.
- #define EF_EVENT_RX_DISCARD_TYPE(e) ((e).rx_discard.subtype)

  Get the reason for an EF_EVENT_TYPE_RX_DISCARD event.
- #define EF_EVENT_RX_DISCARD_BYTES(e) ((e).rx_discard.len)

  Get the length of a discarded packet.
- #define EF_EVENT_RX_MULTI_Q_ID(e) ((e).rx_multi.q_id)

  Get the RX descriptor ring ID used for a received packet.
- #define EF_EVENT_RX_MULTI_CONT(e)

  True if the CONTinuation Of Packet flag is set for an RX HT event.
- #define EF_EVENT_RX_MULTI_SOP(e)

  True if the Start Of Packet flag is set for an RX HT event.
- #define EF_EVENT_TX_ERROR_Q_ID(e) ((e).tx_error.q_id)

  Get the TX descriptor ring ID used for a transmit error.
- #define EF_EVENT_TX_ERROR_TYPE(e) ((e).tx_error.subtype)

  Get the reason for a TX_ERROR event.
- #define EF_VI_SYNC_FLAG_CLOCK_SET 1

  The adapter clock has previously been set in sync with the system.
- #define EF_VI_SYNC_FLAG_CLOCK_IN_SYNC 2

  The adapter clock is in sync with the external clock (PTP)
- #define EF_EVENT_TX_WITH_TIMESTAMP_Q_ID(e) ((e).tx_timestamp.q_id)

  Get the TX descriptor ring ID used for a timestamped packet.
- #define EF_EVENT_TX_WITH_TIMESTAMP_RQ_ID(e) ((e).tx_timestamp.rq_id)

  Get the dma_id used for a timestamped packet.
- #define EF_EVENT_TX_WITH_TIMESTAMP_SEC(e) ((e).tx_timestamp.ts_sec)

  Get the number of seconds from the timestamp of a transmitted packet.

Copyright © Solarflare Communications 2017

- #define EF_EVENT_TX_WITH_TIMESTAMP_NSEC(e) ((e).tx_timestamp.ts_nsec)

  Get the number of nanoseconds from the timestamp of a transmitted packet.
- #define EF_EVENT_TX_WITH_TIMESTAMP_SYNC_MASK (EF_VI_SYNC_FLAG_CLOCK_SET | EF_V↩ I_SYNC_FLAG_CLOCK_IN_SYNC)

  Mask for the sync flags in the timestamp of a transmitted packet.
- #define EF_EVENT_TX_WITH_TIMESTAMP_SYNC_FLAGS(e) ((e).tx_timestamp.ts_nsec & EF_EVENT↩ _TX_WITH_TIMESTAMP_SYNC_MASK)

  Get the sync flags from the timestamp of a transmitted packet.
- #define EF_EVENT_TX_ALT_Q_ID(e) ((e).tx_alt.q_id)

  Get the TX descriptor ring ID used for a TX alternative packet.
- #define EF_EVENT_TX_ALT_ALT_ID(e) ((e).tx_alt.alt_id)

  Get the TX alternative ID used for a TX alternative packet.
- #define EF_EVENT_RX_NO_DESC_TRUNC_Q_ID(e) ((e).rx_no_desc_trunc.q_id)

  Get the RX descriptor ring ID used for a received packet that was truncated due to a lack of descriptors.
- #define EF_EVENT_SW_DATA_MASK 0xffff

  Mask for the data in a software generated event.
- #define EF_EVENT_SW_DATA(e) ((e).sw.data)

  Get the data for an EF_EVENT_TYPE_SW event.
- #define EF_EVENT_FMT "[ev:%x]"

  Output format for an ef_event.
- #define EF_EVENT_PRI_ARG(e) (unsigned) (e).generic.type

  Get the type of an event.
- #define ef_vi_receive_init(vi, addr, dma_id) (vi)->ops.receive_init((vi), (addr), (dma_id))

  Initialize an RX descriptor on the RX descriptor ring.
- #define ef_vi_receive_push(vi) (vi)->ops.receive_push((vi))

  Submit newly initialized RX descriptors to the NIC.
- #define EF_VI_RECEIVE_BATCH 15

  Maximum number of receive completions per receive event.
- #define ef_vi_transmitv_init(vi, iov, iov_len, dma_id) (vi)->ops.transmitv_init((vi), (iov), (iov_len), (dma_id))

  Initialize TX descriptors on the TX descriptor ring, for a vector of packet buffers.
- #define ef_vi_transmit_push(vi) (vi)->ops.transmit_push((vi))

  Submit newly initialized TX descriptors to the NIC.
- #define ef_vi_transmit(vi, base, len, dma_id) (vi)->ops.transmit((vi), (base), (len), (dma_id))

  Transmit a packet from a single packet buffer.
- #define ef_vi_transmitv(vi, iov, iov_len, dma_id) (vi)->ops.transmitv((vi), (iov), (iov_len), (dma_id))

  Transmit a packet from a vector of packet buffers.
- #define ef_vi_transmit_pio(vi, offset, len, dma_id) (vi)->ops.transmit_pio((vi), (offset), (len), (dma_id))

  Transmit a packet already resident in Programmed I/O.
- #define ef_vi_transmit_copy_pio(vi, pio_offset, src_buf, len, dma_id)

  Transmit a packet by copying it into the Programmed I/O region.
- #define ef_vi_transmit_pio_warm(vi) (vi)->ops.transmit_pio_warm((vi))

  Warm Programmed I/O transmit path for subsequent transmit.
- #define ef_vi_transmit_copy_pio_warm(vi, pio_offset, src_buf, len) (vi)->ops.transmit_copy_pio_warm((vi), (pio_offset), (src_buf), (len))

  Copy a packet to Programmed I/O region and warm transmit path.
- #define EF_VI_TRANSMIT_BATCH 64

  Maximum number of transmit completions per transmit event.
- #define ef_vi_transmit_alt_select(vi, alt_id) (vi)->ops.transmit_alt_select((vi), (alt_id))

  Select a TX alternative as the destination for future sends.

 Copyright © Solarflare Communications 2017

- #define ef_vi_transmit_alt_select_normal(vi) (vi)->ops.transmit_alt_select_default((vi))
  Select the "normal" data path as the destination for future sends.
- #define ef_vi_transmit_alt_stop(vi, alt_id) (vi)->ops.transmit_alt_stop((vi), (alt_id))
  Transition a TX alternative to the STOP state.
- #define ef_vi_transmit_alt_go(vi, alt_id) (vi)->ops.transmit_alt_go((vi), (alt_id))
  Transition a TX alternative to the GO state.
- #define ef_vi_transmit_alt_discard(vi, alt_id) (vi)->ops.transmit_alt_discard((vi), (alt_id))
  Transition a TX alternative to the DISCARD state.
- #define ef_eventq_prime(vi) (vi)->ops.eventq_prime((vi))
  Prime a virtual interface allowing you to go to sleep blocking on it.
- #define ef_eventq_poll(evq, evs, evs_len) (evq)->ops.eventq_poll((evq), (evs), (evs_len))
  Poll an event queue.

Typedefs

- typedef uint32_t ef_eventq_ptr
  A pointer to an event queue.
- typedef uint64_t ef_addr
  An address.
- typedef char * ef_vi_ioaddr_t
  An address of an I/O area for a virtual interface.
- typedef int ef_request_id
  A DMA request identifier.
- typedef struct ef_vi ef_vi
  A virtual interface.

Enumerations

- enum {
  EF_EVENT_TYPE_RX, EF_EVENT_TYPE_TX, EF_EVENT_TYPE_RX_DISCARD, EF_EVENT_TYPE_TX_ERROR,
  EF_EVENT_TYPE_RX_NO_DESC_TRUNC, EF_EVENT_TYPE_SW, EF_EVENT_TYPE_OFLOW, EF_EVENT_TYPE_TX_WITH_TIMESTAMP,
  EF_EVENT_TYPE_RX_PACKED_STREAM, EF_EVENT_TYPE_RX_MULTI, EF_EVENT_TYPE_TX_ALT,
  EF_EVENT_TYPE_RX_MULTI_DISCARD }
  Possible types of events.
- enum {
  EF_EVENT_RX_DISCARD_CSUM_BAD, EF_EVENT_RX_DISCARD_MCAST_MISMATCH, EF_EVENT_RX_DISCARD_CRC_BAD, EF_EVENT_RX_DISCARD_TRUNC,
  EF_EVENT_RX_DISCARD_RIGHTS, EF_EVENT_RX_DISCARD_EV_ERROR, EF_EVENT_RX_DISCARD_OTHER }
  The reason for an EF_EVENT_TYPE_RX_DISCARD event.
- enum { EF_EVENT_TX_ERROR_RIGHTS, EF_EVENT_TX_ERROR_OFLOW, EF_EVENT_TX_ERROR_2BIG, EF_EVENT_TX_ERROR_BUS }
  The reason for an EF_EVENT_TYPE_TX_ERROR event.

Copyright © Solarflare Communications 2017

- enum ef_vi_flags {
  EF_VI_FLAGS_DEFAULT = 0x0, EF_VI_ISCSI_RX_HDIG = 0x2, EF_VI_ISCSI_TX_HDIG = 0x4, EF_VI_ISCSI_RX_DDIG = 0x8,
  EF_VI_ISCSI_TX_DDIG = 0x10, EF_VI_TX_PHYS_ADDR = 0x20, EF_VI_RX_PHYS_ADDR = 0x40, EF_VI_TX_IP_CSUM_DIS = 0x80,
  EF_VI_TX_TCPUDP_CSUM_DIS = 0x100, EF_VI_TX_TCPUDP_ONLY = 0x200, EF_VI_TX_FILTER_IP = 0x400, EF_VI_TX_FILTER_MAC = 0x800,
  EF_VI_TX_FILTER_MASK_1 = 0x1000, EF_VI_TX_FILTER_MASK_2 = 0x2000, EF_VI_TX_FILTER_MASK_3 = (0x1000 | 0x2000), EF_VI_TX_PUSH_DISABLE = 0x4000,
  EF_VI_TX_PUSH_ALWAYS = 0x8000, EF_VI_RX_TIMESTAMPS = 0x10000, EF_VI_TX_TIMESTAMPS = 0x20000, EF_VI_RX_PACKED_STREAM = 0x80000,
  EF_VI_RX_PS_BUF_SIZE_64K = 0x100000, EF_VI_RX_EVENT_MERGE = 0x200000, EF_VI_TX_ALT = 0x400000, EF_VI_ENABLE_EV_TIMER = 0x800000 }

  Flags that can be requested when allocating an ef_vi.
- enum ef_vi_out_flags { EF_VI_OUT_CLOCK_SYNC_STATUS = 0x1 }

  Flags that can be returned when an ef_vi has been allocated.
- enum ef_vi_rx_discard_err_flags {
  EF_VI_DISCARD_RX_L4_CSUM_ERR = 0x1, EF_VI_DISCARD_RX_L3_CSUM_ERR = 0x2, EF_VI_DISCARD_RX_ETH_FCS_ERR = 0x4, EF_VI_DISCARD_RX_ETH_LEN_ERR = 0x8,
  EF_VI_DISCARD_RX_TOBE_DISC = 0x10 }

  Flags that define which errors will cause RX_DISCARD events.
- enum ef_vi_arch { EF_VI_ARCH_FALCON, EF_VI_ARCH_EF10 }

  NIC architectures that are supported.
- enum ef_vi_layout_type { EF_VI_LAYOUT_FRAME, EF_VI_LAYOUT_MINOR_TICKS, EF_VI_LAYOUT_PACKET_LENGTH }

  Types of layout that are used for receive buffers.

Functions

- ef_vi_inline unsigned ef_vi_resource_id (ef_vi *vi)

  Return the resource ID of the virtual interface.
- ef_vi_inline enum ef_vi_flags ef_vi_flags (ef_vi *vi)

  Return the flags of the virtual interface.
- ef_vi_inline unsigned ef_vi_instance (ef_vi *vi)

  Return the instance ID of the virtual interface.
- const char * ef_vi_version_str (void)

  Return a string that identifies the version of ef_vi.
- const char * ef_vi_driver_interface_str (void)

  Returns a string that identifies the char driver interface required.
- ef_vi_inline int ef_vi_receive_prefix_len (ef_vi *vi)

  Returns the length of the prefix at the start of a received packet.
- ef_vi_inline int ef_vi_receive_buffer_len (ef_vi *vi)

  Returns the length of a receive buffer.
- ef_vi_inline void ef_vi_receive_set_buffer_len (ef_vi *vi, unsigned buf_len)

  Sets the length of receive buffers.
- ef_vi_inline int ef_vi_receive_space (ef_vi *vi)

  Returns the amount of free space in the RX descriptor ring.
- ef_vi_inline int ef_vi_receive_fill_level (ef_vi *vi)

  Returns the fill level of the RX descriptor ring.
- ef_vi_inline int ef_vi_receive_capacity (ef_vi *vi)

 Copyright © Solarflare Communications 2017

Returns the total capacity of the RX descriptor ring.

- int ef_vi_receive_post (ef_vi ∗vi, ef_addr addr, ef_request_id dma_id)

  Initialize an RX descriptor on the RX descriptor ring, and submit it to the NIC.
- int ef_vi_receive_get_timestamp (ef_vi ∗vi, const void ∗pkt, struct timespec ∗ts_out)

  Deprecated: use ef_vi_receive_get_timestamp_with_sync_flags() instead.
- int ef_vi_receive_get_timestamp_with_sync_flags (ef_vi ∗vi, const void ∗pkt, struct timespec ∗ts_out, unsigned ∗flags_out)

  Retrieve the UTC timestamp associated with a received packet, and the clock sync status flags.
- int ef_vi_receive_get_bytes (ef_vi ∗vi, const void ∗pkt, uint16_t ∗bytes_out)

  Retrieve the number of bytes in a received packet in RX event merge mode.
- int ef_vi_receive_unbundle (ef_vi ∗ep, const ef_event ∗event, ef_request_id ∗ids)

  Unbundle an event of type EF_EVENT_TYPE_RX_MULTI.
- int ef_vi_receive_set_discards (ef_vi ∗vi, unsigned discard_err_flags)

  Set which errors cause an EF_EVENT_TYPE_RX_DISCARD event.
- ef_vi_inline int ef_vi_transmit_space (ef_vi ∗vi)

  Returns the amount of free space in the TX descriptor ring.
- ef_vi_inline int ef_vi_transmit_fill_level (ef_vi ∗vi)

  Returns the fill level of the TX descriptor ring.
- ef_vi_inline int ef_vi_transmit_capacity (ef_vi ∗vi)

  Returns the total capacity of the TX descriptor ring.
- int ef_vi_transmit_init (ef_vi ∗vi, ef_addr addr, int bytes, ef_request_id dma_id)

  Initialize a TX descriptor on the TX descriptor ring, for a single packet buffer.
- void ef_vi_transmit_init_undo (ef_vi ∗vi)

  Remove all TX descriptors from the TX descriptor ring that have been initialized since last transmit.
- int ef_vi_transmit_unbundle (ef_vi ∗ep, const ef_event ∗event, ef_request_id ∗ids)

  Unbundle an event of type of type EF_EVENT_TYPE_TX or EF_EVENT_TYPE_TX_ERROR.
- unsigned ef_vi_transmit_alt_num_ids (ef_vi ∗vi)

  Return the number of TX alternatives allocated for a virtual interface.
- int ef_vi_transmit_alt_query_overhead (ef_vi ∗vi, struct ef_vi_transmit_alt_overhead ∗params)

  Query per-packet overhead parameters.
- ef_vi_inline ef_vi_pure uint32_t ef_vi_transmit_alt_usage (const struct ef_vi_transmit_alt_overhead ∗params, uint32_t pkt_len)

  Calculate a packet's buffer usage.
- void ef_vi_set_tx_push_threshold (ef_vi ∗vi, unsigned threshold)

  Set the threshold at which to switch from using TX descriptor push to using a doorbell.
- int ef_eventq_has_event (const ef_vi ∗vi)

  Returns true if ef_eventq_poll() will return event(s)
- int ef_eventq_has_many_events (const ef_vi ∗evq, int n_events)

  Returns true if there are a given number of events in the event queue.
- int ef_eventq_capacity (ef_vi ∗vi)

  Returns the capacity of an event queue.
- ef_vi_inline unsigned ef_eventq_current (ef_vi ∗evq)

  Get the current offset into the event queue.
- int ef_vi_receive_query_layout (ef_vi ∗vi, const ef_vi_layout_entry ∗∗const layout_out, int ∗layout_len_out)

  Gets the layout of the data that the adapter delivers into receive buffers.

 Copyright © Solarflare Communications 2017

10.9.1 Detailed Description

Virtual Interface definitions for EtherFabric Virtual Interface HAL.

Author

Solarflare Communications, Inc.

Date

2017/02/21

Copyright

Copyright © 2017 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.9.2 Macro Definition Documentation

10.9.2.1 EF_EVENT_RX_MULTI_CONT

```
#define EF_EVENT_RX_MULTI_CONT(
              e )
```

Value:

```
((e).rx_multi.flags & \
                                          EF_EVENT_FLAG_CONT)
```

True if the CONTinuation Of Packet flag is set for an RX HT event.

Definition at line 322 of file ef_vi.h.

10.9.2.2 EF_EVENT_RX_MULTI_SOP

```
#define EF_EVENT_RX_MULTI_SOP(
              e )
```

Value:

```
((e).rx_multi.flags & \
                                          EF_EVENT_FLAG_SOP)
```

True if the Start Of Packet flag is set for an RX HT event.

Definition at line 325 of file ef_vi.h.

10.9.2.3 EF_EVENT_RX_PS_NEXT_BUFFER

```
#define EF_EVENT_RX_PS_NEXT_BUFFER(
              e )
```

Value:

```
((e).rx_packed_stream.flags &    \
                                    EF_EVENT_FLAG_PS_NEXT_BUFFER)
```

True if the next buffer flag is set for a packed stream event.

Definition at line 286 of file ef_vi.h.

10.9.2.4 ef_eventq_poll

```
#define ef_eventq_poll(
              evq,
              evs,
              evs_len ) (evq)->ops.eventq_poll((evq), (evs), (evs_len))
```

Poll an event queue.

Parameters

evq	The event queue to poll.
evs	Array in which to return polled events.
evs_len	Length of the evs array, must be >= EF_VI_EVENT_POLL_MIN_EVS.

Returns

The number of events retrieved.

Poll an event queue. Any events that have been raised are added to the given array. Most events correspond to packets arriving, or packet transmission completing. This function is critical to latency, and must be called as often as possible.

This function returns immediately, even if there are no outstanding events. The array might not be full on return.

Definition at line 1857 of file ef_vi.h.

10.9.2.5 ef_eventq_prime

```
#define ef_eventq_prime(
              vi ) (vi)->ops.eventq_prime((vi))
```

Prime a virtual interface allowing you to go to sleep blocking on it.

Parameters

vi	The virtual interface to prime.

Returns

None.

Prime a virtual interface allowing you to go to sleep blocking on it.

Definition at line 1837 of file ef_vi.h.

10.9.2.6 ef_vi_receive_init

```
#define ef_vi_receive_init(
              vi,
              addr,
              dma_id ) (vi)->ops.receive_init((vi), (addr), (dma_id))
```

Initialize an RX descriptor on the RX descriptor ring.

Parameters

vi	The virtual interface for which to initialize an RX descriptor.
addr	DMA address of the packet buffer to associate with the descriptor, as obtained from ef_memreg_dma_addr().
dma_id	DMA id to associate with the descriptor. This is completely arbitrary, and can be used for subsequent tracking of buffers.

Returns

0 on success, or a negative error code.

Initialize an RX descriptor on the RX descriptor ring, and prepare the associated packet buffer (identified by its DMA address) to receive packets. This function only writes a few bytes into host memory, and is very fast.

Definition at line 1030 of file ef_vi.h.

10.9.2.7 ef_vi_receive_push

```
#define ef_vi_receive_push(
              vi ) (vi)->ops.receive_push((vi))
```

Submit newly initialized RX descriptors to the NIC.

 Copyright © Solarflare Communications 2017

Parameters

vi	The virtual interface for which to push descriptors.

Returns

None.

Submit newly initialized RX descriptors to the NIC. The NIC can then receive packets into the associated packet buffers.

For Solarflare 7000-series NICs, this function submits RX descriptors only in multiples of 8. This is to conform with hardware requirements. If the number of newly initialized RX descriptors is not exactly divisible by 8, this function does not submit any remaining descriptors (up to 7 of them).

Definition at line 1049 of file ef_vi.h.

10.9.2.8 ef_vi_transmit

```
#define ef_vi_transmit(
              vi,
              base,
              len,
              dma_id ) (vi)->ops.transmit((vi), (base), (len), (dma_id))
```

Transmit a packet from a single packet buffer.

Parameters

vi	The virtual interface for which to initialize and push a TX descriptor.
base	DMA address of the packet buffer to associate with the descriptor, as obtained from ef_memreg_dma_addr().
len	The size of the packet to transmit.
dma_id	DMA id to associate with the descriptor. This is completely arbitrary, and can be used for subsequent tracking of buffers.

Returns

0 on success, or a negative error code:
-EAGAIN if the descriptor ring is full.

Transmit a packet from a single packet buffer. This Initializes a TX descriptor on the TX descriptor ring, and submits it to the NIC. The NIC can then transmit a packet from the associated packet buffer.

This function simply wraps ef_vi_transmit_init() and ef_vi_transmit_push(). It is provided as a convenience. It is less efficient than submitting the descriptors in batches by calling the functions separately, but unless there is a batch of packets to transmit, calling this function is often the right thing to do.

Definition at line 1388 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

10.9.2.9 ef_vi_transmit_alt_discard

```
#define ef_vi_transmit_alt_discard(
              vi,
              alt_id ) (vi)->ops.transmit_alt_discard((vi), (alt_id))
```

Transition a TX alternative to the DISCARD state.

Parameters

vi	The virtual interface associated with the TX alternative.
alt_id	The TX alternative to transition to the DISCARD state.

Returns

0 on success, or a negative error code.

Transitions a TX alternative to the DISCARD state. Packets buffered in the alternative are discarded.

As packets are discarded, events of type EF_EVENT_TYPE_TX_ALT are returned to the application. The application should normally wait until all packets have been discarded before transitioning to a different state.

Memory for the TX alternative remains allocated, and is not freed until the virtual interface is freed.

Definition at line 1712 of file ef_vi.h.

10.9.2.10 ef_vi_transmit_alt_go

```
#define ef_vi_transmit_alt_go(
              vi,
              alt_id ) (vi)->ops.transmit_alt_go((vi), (alt_id))
```

Transition a TX alternative to the GO state.

Parameters

vi	The virtual interface associated with the TX alternative.
alt_id	The TX alternative to transition to the GO state.

Returns

0 on success, or a negative error code.

Transitions a TX alternative to the GO state. Packets buffered in the alternative are transmitted to the network.

 Copyright © Solarflare Communications 2017

As packets are transmitted events of type EF_EVENT_TYPE_TX_ALT are returned to the application. The application should normally wait until all packets have been sent before transitioning to a different state.

Definition at line 1691 of file ef_vi.h.

10.9.2.11 ef_vi_transmit_alt_select

```
#define ef_vi_transmit_alt_select(
              vi,
              alt_id ) (vi)->ops.transmit_alt_select((vi), (alt_id))
```

Select a TX alternative as the destination for future sends.

Parameters

vi	The virtual interface associated with the TX alternative.
alt_id	The TX alternative to select.

Returns

0 on success, or a negative error code.

Selects a TX alternative as the destination for future sends. Packets can be sent to it using normal send calls such as ef_vi_transmit(). The action then taken depends on the state of the TX alternative:

- if the TX alternative is in the STOP state, the packet is buffered for possible future transmission
- if the TX alternative is in the GO state, the packet is immediately transmitted.

Definition at line 1647 of file ef_vi.h.

10.9.2.12 ef_vi_transmit_alt_select_normal

```
#define ef_vi_transmit_alt_select_normal(
              vi ) (vi)->ops.transmit_alt_select_default((vi))
```

Select the "normal" data path as the destination for future sends.

Parameters

vi	A virtual interface associated with a TX alternative.

Selects the "normal" data path as the destination for future sends. The virtual interface then transmits packets to

the network immediately, in the normal way. This call undoes the effect of ef_vi_transmit_alt_select().

Definition at line 1661 of file ef_vi.h.

10.9.2.13 ef_vi_transmit_alt_stop

```
#define ef_vi_transmit_alt_stop(
              vi,
              alt_id ) (vi)->ops.transmit_alt_stop((vi), (alt_id))
```

Transition a TX alternative to the STOP state.

Parameters

vi	The virtual interface associated with the TX alternative.
alt_id	The TX alternative to transition to the STOP state.

Transitions a TX alternative to the STOP state. Packets that are sent to a TX alternative in the STOP state are buffered on the adapter.

Definition at line 1673 of file ef_vi.h.

10.9.2.14 ef_vi_transmit_copy_pio

```
#define ef_vi_transmit_copy_pio(
              vi,
              pio_offset,
              src_buf,
              len,
              dma_id )
```

Value:

```
(vi)->ops.transmit_copy_pio((vi), (pio_offset), (src_buf),            \
                            (len), (dma_id))
```

Transmit a packet by copying it into the Programmed I/O region.

Parameters

vi	The virtual interface from which to transmit.
pio_offset	The offset within its Programmed I/O region to the start of the packet. This must be aligned to at least a 64-byte boundary.
src_buf	The source buffer from which to read the packet.
len	Length of the packet to transmit. This must be at least 16 bytes.
dma_id	DMA id to associate with the descriptor. This is completely arbitrary, and can be used for subsequent tracking of buffers.

Copyright © Solarflare Communications 2017

Returns

0 on success, or a negative error code:
-EAGAIN if the descriptor ring is full.

Transmit a packet by copying it into the Programmed I/O region.

The src_buf parameter must point at a complete packet that is copied to the adapter and transmitted. The source buffer need not be registered, and is available for re-use immediately after this call returns.

This call does not copy the packet data into the local copy of the adapter's Programmed I/O buffer. As a result it is slightly faster than calling ef_pio_memcpy() followed by ef_vi_transmit_pio().

The Programmed I/O region used by this call must not be reused until an event indicating TX completion is handled (see Transmitting Packets), thus completing the transmit operation for the packet. Failure to do so might corrupt an ongoing transmit.

The Programmed I/O region can hold multiple smaller packets, referenced by different offset parameters. All other constraints must still be observed, including:

- alignment
- minimum size
- maximum size
- avoiding reuse until transmission is complete.

Definition at line 1503 of file ef_vi.h.

10.9.2.15 ef_vi_transmit_copy_pio_warm

```
#define ef_vi_transmit_copy_pio_warm(
              vi,
              pio_offset,
              src_buf,
              len ) (vi)->ops.transmit_copy_pio_warm((vi), (pio_offset), (src_buf), (len))
```

Copy a packet to Programmed I/O region and warm transmit path.

Parameters

vi	The virtual interface from which transmit is planned.
pio_offset	The offset within its Programmed I/O region to the start of the packet. This must be aligned to at least a 64-byte boundary.
src_buf	The source buffer from which to read the packet.
len	Length of the packet to transmit. This must be at least 16 bytes.

 Copyright © Solarflare Communications 2017

Returns

None

Copy a packet to Programmed I/O region and warm transmit path

The application can call this function in advance of calls to ef_vi_transmit_copy_pio() to reduce latency jitter caused by code and state being evicted from cache during delays between transmits. This is also effective before the first transmit using Programmed I/O.

No data is sent but this function will copy data to the Programmed I/O region. Therefore all constraints regarding copying to the Programmed I/O region must be met. This includes not reusing a region previously transmitted from until the corresponding TX completion has been handled. See ef_vi_transmit_copy_pio() for full details of the constraints.

While this may also benefit a subsequent call to ef_vi_transmit_pio(), it follows a different code path. See ef_vi_↵
transmit_pio_warm() for a warming function designed to warm for ef_vi_transmit_pio().

Definition at line 1559 of file ef_vi.h.

10.9.2.16 ef_vi_transmit_pio

```
#define ef_vi_transmit_pio(
              vi,
              offset,
              len,
              dma_id ) (vi)->ops.transmit_pio((vi), (offset), (len), (dma_id))
```

Transmit a packet already resident in Programmed I/O.

Parameters

vi	The virtual interface from which to transmit.
offset	The offset within its Programmed I/O region to the start of the packet. This must be aligned to at least a 64-byte boundary.
len	Length of the packet to transmit. This must be at least 16 bytes.
dma↵_id	DMA id to associate with the descriptor. This is completely arbitrary, and can be used for subsequent tracking of buffers.

Returns

0 on success, or a negative error code:
-EAGAIN if the descriptor ring is full.

Transmit a packet already resident in Programmed I/O.

The Programmed I/O region used by this call must not be reused until an event indicating TX completion is handled (see Transmitting Packets), thus completing the transmit operation for the packet. Failure to do so might corrupt an ongoing transmit.

The Programmed I/O region can hold multiple packets, referenced by different offset parameters. All other constraints must still be observed, including:

- alignment
- minimum size
- maximum size
- avoiding reuse until transmission is complete.

Definition at line 1460 of file ef_vi.h.

10.9.2.17 ef_vi_transmit_pio_warm

```
#define ef_vi_transmit_pio_warm(
              vi ) (vi)->ops.transmit_pio_warm((vi))
```

Warm Programmed I/O transmit path for subsequent transmit.

Parameters

vi	The virtual interface from which transmit is planned.

Returns

None

Warm Programmed I/O transmit path for a subsequent transmit.

The application can call this function in advance of calls to ef_vi_transmit_pio() to reduce latency jitter caused by code and state being evicted from cache during delays between transmits. This is also effective before the first transmit using Programmed I/O.

While this may also benefit a subsequent call to ef_vi_transmit_copy_pio(), it follows a different code path. See ef_vi_transmit_copy_pio_warm() for a warming function designed to warm for ef_vi_transmit_copy_pio().

Definition at line 1525 of file ef_vi.h.

10.9.2.18 ef_vi_transmit_push

```
#define ef_vi_transmit_push(
              vi ) (vi)->ops.transmit_push((vi))
```

Submit newly initialized TX descriptors to the NIC.

Parameters

vi	The virtual interface for which to push descriptors.

Copyright © Solarflare Communications 2017

Returns

None.

Submit newly initialized TX descriptors to the NIC. The NIC can then transmit packets from the associated packet buffers.

New TX descriptors must have been initialized using ef_vi_transmit_init() or ef_vi_transmitv_init() before calling this function, and so in particular it is not legal to call this function more than once without initializing new descriptors in between those calls.

Definition at line 1361 of file ef_vi.h.

10.9.2.19 ef_vi_transmitv

```
#define ef_vi_transmitv(
              vi,
              iov,
              iov_len,
              dma_id ) (vi)->ops.transmitv((vi), (iov), (iov_len), (dma_id))
```

Transmit a packet from a vector of packet buffers.

Parameters

vi	The virtual interface for which to initialize a TX descriptor.
iov	Start of the iovec array describing the packet buffers.
iov_len	Length of the iovec array.
dma_id	DMA id to associate with the descriptor. This is completely arbitrary, and can be used for subsequent tracking of buffers.

Returns

0 on success, or a negative error code:
-EAGAIN if the descriptor ring is full.

Transmit a packet from a vector of packet buffers. This initializes a TX descriptor on the TX descriptor ring, and submits it to the NIC. The NIC can then transmit a packet from the associated packet buffers.

This function simply wraps ef_vi_transmitv_init() and ef_vi_transmit_push(). It is provided as a convenience. It is less efficient than submitting the descriptors in batches by calling the functions separately, but unless there is a batch of packets to transmit, calling this function is often the right thing to do.

Building a packet by concatenating a vector of buffers allows:

- sending a packet that is larger than a packet buffer
  - the packet is split across multiple buffers in a vector

- optimizing sending packets with only small differences:
  - the packet is split into those parts that are constant, and those that vary between transmits
  - each part is written into its own buffer
  - after each transmit, the buffers containing varying data must be updated, but the buffers containing constant data are re-used
  - this minimizes the amount of data written between transmits.

Definition at line 1426 of file ef_vi.h.

10.9.2.20 ef_vi_transmitv_init

```
#define ef_vi_transmitv_init(
              vi,
              iov,
              iov_len,
              dma_id ) (vi)->ops.transmitv_init((vi), (iov), (iov_len), (dma_id))
```

Initialize TX descriptors on the TX descriptor ring, for a vector of packet buffers.

Parameters

vi	The virtual interface for which to initialize a TX descriptor.
iov	Start of the iovec array describing the packet buffers.
iov_len	Length of the iovec array.
dma_id	DMA id to associate with the descriptor. This is completely arbitrary, and can be used for subsequent tracking of buffers.

Returns

0 on success, or a negative error code:
-EAGAIN if the descriptor ring is full.

Initialize TX descriptors on the TX descriptor ring, for a vector of packet buffers. The associated packet buffers (identified in the iov vector) must contain the packet to transmit. This function only writes a few bytes into host memory, and is very fast.

Building a packet by concatenating a vector of buffers allows:

- sending a packet that is larger than a packet buffer
  - the packet is split across multiple buffers in a vector
- optimizing sending packets with only small differences:
  - the packet is split into those parts that are constant, and those that vary between transmits
  - each part is written into its own buffer
  - after each transmit, the buffers containing varying data must be updated, but the buffers containing constant data are re-used
  - this minimizes the amount of data written between transmits.

Definition at line 1343 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

10.9.3 Typedef Documentation

10.9.3.1 ef_request_id

```
typedef int ef_request_id
```

A DMA request identifier.

This is an integer token specified by the transport and associated with a DMA request. It is returned to the VI user with DMA completion events. It is typically used to identify the buffer associated with the transfer.

Definition at line 141 of file ef_vi.h.

10.9.3.2 ef_vi

```
typedef struct ef_vi ef_vi
```

A virtual interface.

An ef_vi represents a virtual interface on a specific NIC. A virtual interface is a collection of an event queue and two DMA queues used to pass Ethernet frames between the transport implementation and the network.

Users should not access this structure.

10.9.4 Enumeration Type Documentation

10.9.4.1 anonymous enum

```
anonymous enum
```

Possible types of events.

Enumerator

EF_EVENT_TYPE_RX	Good data was received.
EF_EVENT_TYPE_TX	Packets have been sent.
EF_EVENT_TYPE_RX_DISCARD	Data received and buffer consumed, but something is wrong.
EF_EVENT_TYPE_TX_ERROR	Transmit of packet failed.
EF_EVENT_TYPE_RX_NO_DESC_TRUNC	Received packet was truncated due to a lack of descriptors.
EF_EVENT_TYPE_SW	Software generated event.
EF_EVENT_TYPE_OFLOW	Event queue overflow.
EF_EVENT_TYPE_TX_WITH_TIMESTAMP	TX timestamp event.
EF_EVENT_TYPE_RX_PACKED_STREAM	A batch of packets was received in a packed stream.
EF_EVENT_TYPE_RX_MULTI	A batch of packets was received on a RX event merge vi.
EF_EVENT_TYPE_TX_ALT	Packet has been transmitted via a "TX alternative".

Definition at line 245 of file ef_vi.h.

10.9.4.2 anonymous enum

```
anonymous enum
```

The reason for an EF_EVENT_TYPE_RX_DISCARD event.

Enumerator

EF_EVENT_RX_DISCARD_CSUM_BAD	IP header or TCP/UDP checksum error
EF_EVENT_RX_DISCARD_MCAST_MISMATCH	Hash mismatch in a multicast packet
EF_EVENT_RX_DISCARD_CRC_BAD	Ethernet CRC error
EF_EVENT_RX_DISCARD_TRUNC	Frame was truncated
EF_EVENT_RX_DISCARD_RIGHTS	No ownership rights for the packet
EF_EVENT_RX_DISCARD_EV_ERROR	Event queue error, previous RX event has been lost
EF_EVENT_RX_DISCARD_OTHER	Other unspecified reason

Definition at line 329 of file ef_vi.h.

10.9.4.3 anonymous enum

```
anonymous enum
```

The reason for an EF_EVENT_TYPE_TX_ERROR event.

Enumerator

EF_EVENT_TX_ERROR_RIGHTS	No ownership rights for the packet
EF_EVENT_TX_ERROR_OFLOW	TX pacing engine work queue was full
EF_EVENT_TX_ERROR_2BIG	Oversized transfer has been indicated by the descriptor
EF_EVENT_TX_ERROR_BUS	Bus or descriptor protocol error occurred when attempting to read the memory referenced by the descriptor

Definition at line 380 of file ef_vi.h.

10.9.4.4 ef_vi_arch

```
enum ef_vi_arch
```

NIC architectures that are supported.

Copyright © Solarflare Communications 2017

Enumerator

EF_VI_ARCH_FALCON	5000 and 6000-series NICs
EF_VI_ARCH_EF10	7000 and 8000-series NICs

Definition at line 523 of file ef_vi.h.

10.9.4.5 ef_vi_flags

```
enum ef_vi_flags
```

Flags that can be requested when allocating an ef_vi.

Enumerator

EF_VI_FLAGS_DEFAULT	Default setting
EF_VI_ISCSI_RX_HDIG	Receive iSCSI header digest enable: hardware verifies header digest (CRC) when packet is iSCSI.
EF_VI_ISCSI_TX_HDIG	Transmit iSCSI header digest enable: hardware calculates and inserts header digest (CRC) when packet is iSCSI.
EF_VI_ISCSI_RX_DDIG	Receive iSCSI data digest enable: hardware verifies data digest (CRC) when packet is iSCSI.
EF_VI_ISCSI_TX_DDIG	Transmit iSCSI data digest enable: hardware calculates and inserts data digest (CRC) when packet is iSCSI.
EF_VI_TX_PHYS_ADDR	Use physically addressed TX descriptor ring
EF_VI_RX_PHYS_ADDR	Use physically addressed RX descriptor ring
EF_VI_TX_IP_CSUM_DIS	IP checksum calculation and replacement is disabled
EF_VI_TX_TCPUDP_CSUM_DIS	TCP/UDP checksum calculation and replacement is disabled
EF_VI_TX_TCPUDP_ONLY	Drop transmit packets that are not TCP or UDP
EF_VI_TX_FILTER_IP	Drop packets with a mismatched IP source address (5000 and 6000 series only)
EF_VI_TX_FILTER_MAC	Drop packets with a mismatched MAC source address (5000 and 6000 series only)
EF_VI_TX_FILTER_MASK_1	Set lowest bit of queue ID to 0 when matching within filter block (5000 and 6000 series only)
EF_VI_TX_FILTER_MASK_2	Set lowest 2 bits of queue ID to 0 when matching within filter block (5000 and 6000 series only)
EF_VI_TX_FILTER_MASK_3	Set lowest 3 bits of queue ID to 0 when matching within filter block (5000 and 6000 series only)
EF_VI_TX_PUSH_DISABLE	Disable using TX descriptor push, so always use doorbell for transmit
EF_VI_TX_PUSH_ALWAYS	Always use TX descriptor push, so never use doorbell for transmit (7000 series and newer)
EF_VI_RX_TIMESTAMPS	Add timestamp to received packets (7000 series and newer)
EF_VI_TX_TIMESTAMPS	Add timestamp to transmitted packets (7000 series and newer)
EF_VI_RX_PACKED_STREAM	Enable packed stream mode for received packets (7000 series or newer)
EF_VI_RX_PS_BUF_SIZE_64K	Use 64KiB packed stream buffers, instead of the 1024KiB default (7000 series and newer).

Copyright © Solarflare Communications 2017

Enumerator

EF_VI_RX_EVENT_MERGE	Enable RX event merging mode for received packets See ef_vi_receive_unbundle() and ef_vi_receive_get_bytes() for more details on using RX event merging mode.
EF_VI_TX_ALT	Enable the "TX alternatives" feature (8000 series and newer).
EF_VI_ENABLE_EV_TIMER	Controls, on 8000 series and newer, whether the hardware event timer is enabled

Definition at line 426 of file ef_vi.h.

10.9.4.6 ef_vi_layout_type

```
enum ef_vi_layout_type
```

Types of layout that are used for receive buffers.

Enumerator

EF_VI_LAYOUT_FRAME	An Ethernet frameo
EF_VI_LAYOUT_MINOR_TICKS	Hardware timestamp (minor ticks) - 32 bits
EF_VI_LAYOUT_PACKET_LENGTH	Packet length - 16 bits

Definition at line 1902 of file ef_vi.h.

10.9.4.7 ef_vi_out_flags

```
enum ef_vi_out_flags
```

Flags that can be returned when an ef_vi has been allocated.

Enumerator

EF_VI_OUT_CLOCK_SYNC_STATUS	Clock sync status

Definition at line 498 of file ef_vi.h.

10.9.4.8 ef_vi_rx_discard_err_flags

```
enum ef_vi_rx_discard_err_flags
```

Flags that define which errors will cause RX_DISCARD events.

 Copyright © Solarflare Communications 2017

Enumerator

EF_VI_DISCARD_RX_L4_CSUM_ERR	TCP or UDP checksum error
EF_VI_DISCARD_RX_L3_CSUM_ERR	IP checksum error
EF_VI_DISCARD_RX_ETH_FCS_ERR	Ethernet FCS error
EF_VI_DISCARD_RX_ETH_LEN_ERR	Ethernet frame length error
EF_VI_DISCARD_RX_TOBE_DISC	To be discard in software (includes frame length error)

Definition at line 505 of file ef_vi.h.

10.9.5 Function Documentation

10.9.5.1 ef_eventq_capacity()

```
int ef_eventq_capacity (
            ef_vi * vi )
```

Returns the capacity of an event queue.

Parameters

vi	The event queue to query.

Returns

The capacity of an event queue.

Returns the capacity of an event queue. This is the maximum number of events that can be stored into the event queue before overflow.

It is up to the application to avoid event queue overflow by ensuring that the maximum number of events that can be delivered into an event queue is limited to its capacity. In general each RX descriptor and TX descriptor posted can cause an event to be generated.

In addition, when time-stamping is enabled time-sync events are generated at a rate of 4 per second. When TX timestamps are enabled you may get up to one event for each descriptor plus two further events per packet.

10.9.5.2 ef_eventq_current()

```
ef_vi_inline unsigned ef_eventq_current (
            ef_vi * evq )
```

Get the current offset into the event queue.

 Copyright © Solarflare Communications 2017

Parameters

evq	The event queue to query.

Returns

The current offset into the eventq.

Get the current offset into the event queue.

Definition at line 1891 of file ef_vi.h.

10.9.5.3 ef_eventq_has_event()

```
int ef_eventq_has_event (
            const ef_vi * vi )
```

Returns true if ef_eventq_poll() will return event(s)

Parameters

vi	The virtual interface to query.

Returns

True if ef_eventq_poll() will return event(s).

Returns true if ef_eventq_poll() will return event(s).

10.9.5.4 ef_eventq_has_many_events()

```
int ef_eventq_has_many_events (
            const ef_vi * evq,
            int n_events )
```

Returns true if there are a given number of events in the event queue.

Parameters

evq	The event queue to query.
n_events	Number of events to check.

 Copyright © Solarflare Communications 2017

Returns

True if the event queue contains at least `n_events` events.

Returns true if there are a given number of events in the event queue.

This looks ahead in the event queue, so has the property that it will not ping-pong a cache-line when it is called concurrently with events being delivered.

This function returns quickly. It is useful for an application to determine whether it is falling behind in its event processing.

10.9.5.5 ef_vi_driver_interface_str()

```
const char* ef_vi_driver_interface_str (
            void  )
```

Returns a string that identifies the char driver interface required.

Returns

A string that identifies the char driver interface required by this build of ef_vi.

Returns a string that identifies the char driver interface required by this build of ef_vi.

Returns the current version of the drivers that are running - useful to check that it is new enough.

10.9.5.6 ef_vi_flags()

```
ef_vi_inline enum ef_vi_flags ef_vi_flags (
            ef_vi * vi )
```

Return the flags of the virtual interface.

Parameters

vi	The virtual interface to query.

Returns

The flags of the virtual interface.

Return the flags of the virtual interface.

Definition at line 845 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

10.9.5.7 ef_vi_instance()

```
ef_vi_inline unsigned ef_vi_instance (
            ef_vi * vi )
```

Return the instance ID of the virtual interface.

Parameters

vi	The virtual interface to query.

Returns

The instance ID of the virtual interface.

Return the instance ID of the virtual interface.

Definition at line 860 of file ef_vi.h.

10.9.5.8 ef_vi_receive_buffer_len()

```
ef_vi_inline int ef_vi_receive_buffer_len (
            ef_vi * vi )
```

Returns the length of a receive buffer.

Parameters

vi	The virtual interface to query.

Returns

The length of a receive buffer.

Returns the length of a receive buffer.

When a packet arrives that does not fit within a single receive buffer, it is spread over multiple buffers.

The application must ensure that receive buffers are at least as large as the value returned by this function, else there is a risk that a DMA may overrun the buffer.

Definition at line 938 of file ef_vi.h.

10.9.5.9 ef_vi_receive_capacity()

```
ef_vi_inline int ef_vi_receive_capacity (
            ef_vi * vi )
```

Returns the total capacity of the RX descriptor ring.

 Copyright © Solarflare Communications 2017

Parameters

vi	The virtual interface to query.

Returns

The total capacity of the RX descriptor ring, as slots for descriptor entries.

Returns the total capacity of the RX descriptor ring.

Definition at line 1007 of file ef_vi.h.

10.9.5.10 ef_vi_receive_fill_level()

```
ef_vi_inline int ef_vi_receive_fill_level (
            ef_vi * vi )
```

Returns the fill level of the RX descriptor ring.

Parameters

vi	The virtual interface to query.

Returns

The fill level of the RX descriptor ring, as slots for descriptor entries.

Returns the fill level of the RX descriptor ring. This is the number of slots that hold a descriptor (and an associated unfilled packet buffer). The fill level should be kept as high as possible, so there are enough slots available to handle a burst of incoming packets.

Definition at line 991 of file ef_vi.h.

10.9.5.11 ef_vi_receive_get_bytes()

```
int ef_vi_receive_get_bytes (
            ef_vi * vi,
            const void * pkt,
            uint16_t * bytes_out )
```

Retrieve the number of bytes in a received packet in RX event merge mode.

 Copyright © Solarflare Communications 2017

Parameters

vi	The virtual interface that received the packet.
pkt	The first packet buffer for the received packet.
bytes_out	Pointer to a uint16_t, that is updated on return with the number of bytes in the packet.

Note that this function returns the number of bytes in a received packet, not received into a single buffer, ie it must only be called for the first buffer in a packet. For jumbos it will return the full length of the jumbo. Buffers prior to the last buffer in the packet will be filled completely.

The length does not include the length of the packet prefix.

Returns

0 on success, or a negative error code

10.9.5.12 ef_vi_receive_get_timestamp()

```
int ef_vi_receive_get_timestamp (
            ef_vi * vi,
            const void * pkt,
            struct timespec * ts_out )
```

Deprecated: use ef_vi_receive_get_timestamp_with_sync_flags() instead.

Parameters

vi	The virtual interface that received the packet.
pkt	The received packet.
ts_out	Pointer to a timespec, that is updated on return with the UTC timestamp for the packet.

Returns

0 on success, or a negative error code.

This function is now deprecated. Use ef_vi_receive_get_timestamp_with_sync_flags() instead.

Retrieve the UTC timestamp associated with a received packet.

This function must be called after retrieving the associated RX event via ef_eventq_poll(), and before calling ef_eventq_poll() again.

If the virtual interface does not have RX timestamps enabled, the behavior of this function is undefined.

Copyright © Solarflare Communications 2017

10.9.5.13 ef_vi_receive_get_timestamp_with_sync_flags()

```
int ef_vi_receive_get_timestamp_with_sync_flags (
            ef_vi * vi,
            const void * pkt,
            struct timespec * ts_out,
            unsigned * flags_out )
```

Retrieve the UTC timestamp associated with a received packet, and the clock sync status flags.

Parameters

vi	The virtual interface that received the packet.
pkt	The first packet buffer for the received packet.
ts_out	Pointer to a timepsec, that is updated on return with the UTC timestamp for the packet.
flags_out	Pointer to an unsigned, that is updated on return with the sync flags for the packet.

Returns

0 on success, or a negative error code:

- ENOMSG - Synchronisation with adapter has not yet been achieved.
  This only happens with old firmware.
- ENODATA - Packet does not have a timestamp.
  On current Solarflare adapters, packets that are switched from TX to RX do not get timestamped.
- EL2NSYNC - Synchronisation with adapter has been lost.
  This should never happen!

Retrieve the UTC timestamp associated with a received packet, and the clock sync status flags.

This function:

- must be called after retrieving the associated RX event via ef_eventq_poll(), and before calling ef_eventq_poll() again
- must only be called for the first segment of a jumbo packet
- must not be called for any events other than RX.

If the virtual interface does not have RX timestamps enabled, the behavior of this function is undefined.

This function will also fail if the virtual interface has not yet synchronized with the adapter clock. This can take from a few hundred milliseconds up to several seconds from when the virtual interface is allocated.

On success the ts_out and flags_out fields are updated, and a value of zero is returned. The flags_out field contains the following flags:

- EF_VI_SYNC_FLAG_CLOCK_SET is set if the adapter clock has ever been set (in sync with system)
- EF_VI_SYNC_FLAG_CLOCK_IN_SYNC is set if the adapter clock is in sync with the external clock (PTP).

In case of error the timestamp result (∗ts_out) is set to zero, and a non-zero error code is returned (see Return value above).

 Copyright © Solarflare Communications 2017

10.9.5.14 ef_vi_receive_post()

```
int ef_vi_receive_post (
            ef_vi * vi,
            ef_addr addr,
            ef_request_id dma_id )
```

Initialize an RX descriptor on the RX descriptor ring, and submit it to the NIC.

Parameters

vi	The virtual interface for which to initialize and push an RX descriptor.
addr	DMA address of the packet buffer to associate with the descriptor, as obtained from ef_memreg_dma_addr().
dma_id	DMA id to associate with the descriptor. This is completely arbitrary, and can be used for subsequent tracking of buffers.

Returns

0 on success, or a negative error code.

Initialize an RX descriptor on the RX descriptor ring, and submit it to the NIC. The NIC can then receive a packet into the associated packet buffer.

This function simply wraps ef_vi_receive_init() and ef_vi_receive_push(). It is provided as a convenience, but is less efficient than submitting the descriptors in batches by calling the functions separately.

Note that for Solarflare 7000-series NICs, this function submits RX descriptors only in multiples of 8. This is to conform with hardware requirements. If the number of newly initialized RX descriptors is not exactly divisible by 8, this function does not submit any remaining descriptors (including, potentially, the RX descriptor initialized in this call).

10.9.5.15 ef_vi_receive_prefix_len()

```
ef_vi_inline int ef_vi_receive_prefix_len (
            ef_vi * vi )
```

Returns the length of the prefix at the start of a received packet.

Parameters

vi	The virtual interface to query.

Returns

The length of the prefix at the start of a received packet.

Returns the length of the prefix at the start of a received packet.

 Copyright © Solarflare Communications 2017

The NIC may be configured to deliver meta-data in a prefix before the packet payload data. This call returns the size of the prefix.

When a large packet is received that is scattered over multiple packet buffers, the prefix is only present in the first buffer.

Definition at line 917 of file ef_vi.h.

10.9.5.16 ef_vi_receive_query_layout()

```
int ef_vi_receive_query_layout (
            ef_vi * vi,
            const ef_vi_layout_entry **const layout_out,
            int * layout_len_out )
```

Gets the layout of the data that the adapter delivers into receive buffers.

Parameters

vi	The virtual interface to query.
layout_out	Pointer to an ef_vi_layout_entry*, that is updated on return with a reference to the layout table.
layout_len_out	Pointer to an int, that is updated on return with the length of the layout table.

Returns

0 on success, or a negative error code.

Gets the layout of the data that the adapter delivers into receive buffers. Depending on the adapter type and options selected, there can be a meta-data prefix in front of each packet delivered into memory. Note that this prefix is per-packet, not per buffer, ie for jumbos the prefix will only be present in the first buffer of the packet.

The first entry is always of type EF_VI_LAYOUT_FRAME, and the offset is the same as the value returned by ef_vi_receive_prefix_len().

10.9.5.17 ef_vi_receive_set_buffer_len()

```
ef_vi_inline void ef_vi_receive_set_buffer_len (
            ef_vi * vi,
            unsigned buf_len )
```

Sets the length of receive buffers.

Parameters

vi	The virtual interface for which to set the length of receive buffers.
buf_len	The length of receive buffers.

 Copyright © Solarflare Communications 2017

Sets the length of receive buffers for this VI. The new length is used for subsequent calls to ef_vi_receive_init() and ef_vi_receive_post().

This call has no effect for 5000 and 6000-series (Falcon) adapters.

Definition at line 955 of file ef_vi.h.

10.9.5.18 ef_vi_receive_set_discards()

```
int ef_vi_receive_set_discards (
            ef_vi * vi,
            unsigned discard_err_flags )
```

Set which errors cause an EF_EVENT_TYPE_RX_DISCARD event.

Parameters

vi	The virtual interface to configure.
discard_err_flags	Flags which indicate which errors will cause discard events

Returns

0 on success, or a negative error code.

Set which errors cause an EF_EVENT_TYPE_RX_DISCARD event

10.9.5.19 ef_vi_receive_space()

```
ef_vi_inline int ef_vi_receive_space (
            ef_vi * vi )
```

Returns the amount of free space in the RX descriptor ring.

Parameters

vi	The virtual interface to query.

Returns

The amount of free space in the RX descriptor ring, as slots for descriptor entries.

Returns the amount of free space in the RX descriptor ring. This is the number of slots that are available for pushing a new descriptor (and an associated unfilled packet buffer).

Definition at line 972 of file ef_vi.h.

 Copyright © Solarflare Communications 2017

10.9.5.20 ef_vi_receive_unbundle()

```
int ef_vi_receive_unbundle (
            ef_vi * ep,
            const ef_event * event,
            ef_request_id * ids )
```

Unbundle an event of type EF_EVENT_TYPE_RX_MULTI.

Parameters

ep	The virtual interface that has raised the event.
event	The event, of type EF_EVENT_TYPE_RX_MULTI.
ids	Array of size EF_VI_RECEIVE_BATCH, that is updated on return with the DMA ids that were used in the original ef_vi_receive_init() call.

Returns

The number of valid ef_request_ids (can be zero).

Unbundle an event of type EF_EVENT_TYPE_RX_MULTI.

In RX event merge mode the NIC will coalesce multiple packet receptions into a single RX event. This reduces PCIe load, enabling higher potential throughput at the cost of latency.

This function returns the number of descriptors whose reception has completed, and updates the ids array with the ef_request_ids for each completed DMA request.

After calling this function, the RX descriptors for the completed RX event are ready to be re-used.

In order to determine the length of each packet ef_vi_receive_get_bytes() must be called, or the length examined in the packet prefix (see ef_vi_receive_query_layout()).

10.9.5.21 ef_vi_resource_id()

```
ef_vi_inline unsigned ef_vi_resource_id (
            ef_vi * vi )
```

Return the resource ID of the virtual interface.

Parameters

vi	The virtual interface to query.

Returns

The resource ID of the virtual interface.

Return the resource ID of the virtual interface.

Definition at line 831 of file ef_vi.h.

10.9.5.22 ef_vi_set_tx_push_threshold()

```
void ef_vi_set_tx_push_threshold (
            ef_vi * vi,
            unsigned threshold )
```

Set the threshold at which to switch from using TX descriptor push to using a doorbell.

Parameters

vi	The virtual interface for which to set the threshold.
threshold	The threshold to set, as the number of outstanding transmits at which to switch.

Returns

0 on success, or a negative error code.

Set the threshold at which to switch from using TX descriptor push to using a doorbell. TX descriptor push has better latency, but a doorbell is more efficient.

The default value for this is controlled using the EF_VI_TX_PUSH_DISABLE and EF_VI_TX_PUSH_ALWAYS flags to ef_vi_init().

This is not supported by all Solarflare NICs. At the time of writing, 7000-series NICs support this, but it is ignored by earlier NICs.

10.9.5.23 ef_vi_transmit_alt_num_ids()

```
unsigned ef_vi_transmit_alt_num_ids (
            ef_vi * vi )
```

Return the number of TX alternatives allocated for a virtual interface.

Parameters

vi	The virtual interface to query.

Returns

The number of TX alternatives, or a negative error code.

Gets the number of TX alternatives for the given virtual interface.

Copyright © Solarflare Communications 2017

10.9.5.24 ef_vi_transmit_alt_query_overhead()

```
int ef_vi_transmit_alt_query_overhead (
            ef_vi * vi,
            struct ef_vi_transmit_alt_overhead * params )
```

Query per-packet overhead parameters.

Parameters

vi	Interface to be queried
params	Returned overhead parameters

Returns

0 on success or -EINVAL if this VI doesn't support alternatives.

This function returns parameters which are needed by the ef_vi_transmit_alt_usage() function below.

10.9.5.25 ef_vi_transmit_alt_usage()

```
ef_vi_inline ef_vi_pure uint32_t ef_vi_transmit_alt_usage (
            const struct ef_vi_transmit_alt_overhead * params,
            uint32_t pkt_len )
```

Calculate a packet's buffer usage.

Parameters

params	Parameters returned by ef_vi_transmit_alt_query_overhead
pkt_len	Packet length in bytes

Returns

Packet buffer usage in bytes including per-packet overhead

This function calculates the number of bytes of buffering which will be used by the NIC to store a packet with the given length.

The returned value includes per-packet overheads, but does not include any other overhead which may be incurred by the hardware. Note that if the application has successfully requested N bytes of buffering using ef_vi_transmit_alt_alloc() then it is guaranteed to be able to store at least N bytes of packet data + per-packet overhead as calculated by this function.

It is possible that the application may be able to use more space in some situations if the non-per-packet overheads are low enough.

It is important that callers do not use ef_vi_capabilities_get() to query the available buffering. That function does not take into account non-per-packet overheads and so is likely to return more space than can actually be used by the application. This function is provided instead to allow applications to calculate their buffer usage accurately.

Definition at line 1761 of file ef_vi.h.

Copyright © Solarflare Communications 2017

10.9.5.26 ef_vi_transmit_capacity()

```
ef_vi_inline int ef_vi_transmit_capacity (
            ef_vi * vi )
```

Returns the total capacity of the TX descriptor ring.

Parameters

vi	The virtual interface to query.

Returns

The total capacity of the TX descriptor ring, as slots for descriptor entries.

Returns the total capacity of the TX descriptor ring.

Definition at line 1283 of file ef_vi.h.

10.9.5.27 ef_vi_transmit_fill_level()

```
ef_vi_inline int ef_vi_transmit_fill_level (
            ef_vi * vi )
```

Returns the fill level of the TX descriptor ring.

Parameters

vi	The virtual interface to query.

Returns

The fill level of the TX descriptor ring, as slots for descriptor entries.

Returns the fill level of the TX descriptor ring. This is the number of slots that hold a descriptor (and an associated filled packet buffer). The fill level should be low or 0, unless a large number of packets have recently been posted for transmission. A consistently high fill level should be investigated.

Definition at line 1267 of file ef_vi.h.

10.9.5.28 ef_vi_transmit_init()

```
int ef_vi_transmit_init (
            ef_vi * vi,
```

 Copyright © Solarflare Communications 2017

```
                ef_addr addr,
                int bytes,
                ef_request_id dma_id )
```

Initialize a TX descriptor on the TX descriptor ring, for a single packet buffer.

Parameters

vi	The virtual interface for which to initialize a TX descriptor.
addr	DMA address of the packet buffer to associate with the descriptor, as obtained from ef_memreg_dma_addr().
bytes	The size of the packet to transmit.
dma←_id	DMA id to associate with the descriptor. This is completely arbitrary, and can be used for subsequent tracking of buffers.

Returns

0 on success, or a negative error code:
-EAGAIN if the descriptor ring is full.

Initialize a TX descriptor on the TX descriptor ring, for a single packet buffer. The associated packet buffer (identified by its DMA address) must contain the packet to transmit. This function only writes a few bytes into host memory, and is very fast.

10.9.5.29 ef_vi_transmit_init_undo()

```
void ef_vi_transmit_init_undo (
                ef_vi * vi )
```

Remove all TX descriptors from the TX descriptor ring that have been initialized since last transmit.

Parameters

vi	The virtual interface for which to remove initialized TX descriptors from the TX descriptor ring.

Returns

None

Remove all TX descriptors from the TX descriptor ring that have been initialized since last transmit. This will undo the effects of calls made to ef_vi_transmit_init or ef_vi_transmitv_init since the last "push".

Initializing and then removing descriptors can have a warming effect on the transmit code path for subsequent transmits. This can reduce latency jitter caused by code and state being evicted from cache during delays between transmitting packets. This technique is also effective before the first transmit.

 Copyright © Solarflare Communications 2017

10.9.5.30 ef_vi_transmit_space()

```
ef_vi_inline int ef_vi_transmit_space (
            ef_vi * vi )
```

Returns the amount of free space in the TX descriptor ring.

 Copyright © Solarflare Communications 2017

Parameters

vi	The virtual interface to query.

Returns

The amount of free space in the TX descriptor ring, as slots for descriptor entries.

Returns the amount of free space in the TX descriptor ring. This is the number of slots that are available for pushing a new descriptor (and an associated filled packet buffer).

Definition at line 1247 of file ef_vi.h.

10.9.5.31 ef_vi_transmit_unbundle()

```
int ef_vi_transmit_unbundle (
            ef_vi * ep,
            const ef_event * event,
            ef_request_id * ids )
```

Unbundle an event of type of type EF_EVENT_TYPE_TX or EF_EVENT_TYPE_TX_ERROR.

Parameters

ep	The virtual interface that has raised the event.
event	The event, of type EF_EVENT_TYPE_TX or EF_EVENT_TYPE_TX_ERROR
ids	Array of size EF_VI_TRANSMIT_BATCH, that is updated on return with the DMA ids that were used in the originating ef_vi_transmit_*() calls.

Returns

The number of valid ef_request_ids (can be zero).

Unbundle an event of type of type EF_EVENT_TYPE_TX or EF_EVENT_TYPE_TX_ERROR.

The NIC might coalesce multiple packet transmissions into a single TX event in the event queue. This function returns the number of descriptors whose transmission has completed, and updates the ids array with the ef_request_ids for each completed DMA request.

After calling this function, the TX descriptors for the completed TX event are ready to be re-initialized. The associated packet buffers are no longer in use by ef_vi. Each buffer can then be freed, or can be re-used (for example as a packet buffer for a descriptor on the TX ring, or on the RX ring).

10.9.5.32 ef_vi_version_str()

```
const char* ef_vi_version_str (
            void )
```

Return a string that identifies the version of ef_vi.

 Copyright © Solarflare Communications 2017

Returns

A string that identifies the version of ef_vi.

Return a string that identifies the version of ef_vi. This should be treated as an unstructured string. At time of writing it is the version of OpenOnload or EnterpriseOnload in which ef_vi is distributed.

Note that Onload will check this is a version that it recognizes. It recognizes the version strings generated by itself, and those generated by older official releases of Onload (when the API hasn't changed), but not those generated by older patched releases of Onload. Consequently, ef_vi applications built against patched versions of Onload will not be supported by future versions of Onload.

10.10 memreg.h File Reference

Registering memory for EtherFabric Virtual Interface HAL.

```
#include <etherfabric/base.h>
```

Data Structures

- struct ef_memreg

  Memory that has been registered for use with ef_vi.

Typedefs

- typedef struct ef_memreg ef_memreg

  Memory that has been registered for use with ef_vi.

Functions

- int ef_memreg_alloc (ef_memreg ∗mr, ef_driver_handle mr_dh, struct ef_pd ∗pd, ef_driver_handle pd_dh, void ∗p_mem, size_t len_bytes)

  Register a memory region for use with ef_vi.
- int ef_memreg_free (ef_memreg ∗mr, ef_driver_handle mr_dh)

  Unregister a memory region.
- ef_vi_inline ef_addr ef_memreg_dma_addr (ef_memreg ∗mr, size_t offset)

  Return the DMA address for the given offset within a registered memory region.

10.10.1 Detailed Description

Registering memory for EtherFabric Virtual Interface HAL.

Author

Solarflare Communications, Inc.

Date

2015/02/16

Copyright

Copyright © 2015 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.10.2 Function Documentation

10.10.2.1 ef_memreg_alloc()

```
int ef_memreg_alloc (
            ef_memreg * mr,
            ef_driver_handle mr_dh,
            struct ef_pd * pd,
            ef_driver_handle pd_dh,
            void * p_mem,
            size_t len_bytes )
```

Register a memory region for use with ef_vi.

Parameters

mr	The ef_memreg object to initialize.
mr_dh	Driver handle for the ef_memreg.
pd	Protection domain in which to register memory.
pd_dh	Driver handle for the protection domain.
p_mem	Start of memory region to be registered. This must be page-aligned, and so be on a 4K boundary.
len_bytes	Length of memory region to be registered. This must be a multiple of the packet buffer size (currently 2048 bytes).

Returns

0 on success, or a negative error code.

Register memory for use with ef_vi.

Before calling this function, the memory must be allocated. using malloc or similar.

After calling this function, the memory is registered, and can be used for DMA buffers. ef_memreg_dma_addr() can then be used to obtain DMA addresses for buffers within the registered area.

Registered memory is associated with a particular protection domain, and the DMA addresses can be used only with virtual interfaces that are associated with the same protection domain. Memory can be registered with multiple protection domains so that a single pool of buffers can be used with multiple virtual interfaces.

Memory that is registered is pinned, and therefore it cannot be swapped out to disk.

Note

If an application that has registered memory forks, then copy-on-write semantics can cause new pages to be allocated which are not registered. This problem can be solved either by ensuring that the registered memory regions are shared by parent and child (e.g. by using MAP_SHARED), or by using madvise(MADV_DONT↩ FORK) to prevent the registered memory from being accessible in the child.

 Copyright © Solarflare Communications 2017

10.10.2.2 ef_memreg_dma_addr()

```
ef_vi_inline ef_addr ef_memreg_dma_addr (
            ef_memreg * mr,
            size_t offset )
```

Return the DMA address for the given offset within a registered memory region.

Parameters

mr	The ef_memreg object to query.
offset	The offset within the ef_memreg object.

Returns

The DMA address for the given offset within a registered memory region.

Return the DMA address for the given offset within a registered memory region.

Note that DMA addresses are only contiguous within each 4K block of a memory region.

Definition at line 130 of file memreg.h.

10.10.2.3 ef_memreg_free()

```
int ef_memreg_free (
            ef_memreg * mr,
            ef_driver_handle mr_dh )
```

Unregister a memory region.

Parameters

mr	The ef_memreg object to unregister.
mr_dh	Driver handle for the ef_memreg.

Returns

0 on success, or a negative error code.

Unregister a memory region.

Note

To free all the resources, the driver handle associated with the memory must also be closed by calling ef_driver_close().

10.11 packedstream.h File Reference

Packed streams for EtherFabric Virtual Interface HAL.

```
#include <etherfabric/base.h>
```

Data Structures

- struct ef_packed_stream_packet
  Per-packet meta-data.
- struct ef_packed_stream_params
  Packed-stream mode parameters.

Macros

- #define EF_VI_PS_FLAG_CLOCK_SET 0x1
  Set if the adapter clock has ever been set (in sync with system).
- #define EF_VI_PS_FLAG_CLOCK_IN_SYNC 0x2
  Set if the adapter clock is in sync with the external clock (PTP).
- #define EF_VI_PS_FLAG_BAD_FCS 0x4
  Set if a bad Frame Check Sequence has occurred.
- #define EF_VI_PS_FLAG_BAD_L4_CSUM 0x8
  Set if a layer-4 (TCP/UDP) checksum error is detected, or if a good layer-4 checksum is not detected (depending on adapter).
- #define EF_VI_PS_FLAG_BAD_L3_CSUM 0x10
  Set if a layer-3 (IPv4) checksum error is detected, or if a good layer-3 checksum is not detected (depending on adapter).
- #define EF_VI_PS_FLAG_BAD_IP_CSUM
  Retained for backwards compatibility. Do not use.

Functions

- int ef_vi_packed_stream_get_params (ef_vi *vi, ef_packed_stream_params *psp_out)
  Get the parameters for packed-stream mode.
- int ef_vi_packed_stream_unbundle (ef_vi *vi, const ef_event *ev, ef_packed_stream_packet **pkt_iter, int *n_pkts_out, int *n_bytes_out)
  Unbundle an event of type EF_EVENT_TYPE_RX_PACKED_STREAM.

10.11.1 Detailed Description

Packed streams for EtherFabric Virtual Interface HAL.

Author

Solarflare Communications, Inc.

Date

2015/02/16

Copyright

Copyright © 2015 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.11.2 Macro Definition Documentation

10.11.2.1 EF_VI_PS_FLAG_BAD_IP_CSUM

```
#define EF_VI_PS_FLAG_BAD_IP_CSUM
```

Value:

```
(EF_VI_PS_FLAG_BAD_L4_CSUM |        \
                                  EF_VI_PS_FLAG_BAD_L3_CSUM)
```

Retained for backwards compatibility. Do not use.

Definition at line 81 of file packedstream.h.

10.11.3 Function Documentation

10.11.3.1 ef_vi_packed_stream_get_params()

```
int ef_vi_packed_stream_get_params (
            ef_vi * vi,
            ef_packed_stream_params * psp_out )
```

Get the parameters for packed-stream mode.

Parameters

vi	The virtual interface to query.
psp_out	Pointer to an ef_packed_stream_params, that is updated on return with the parameters for packed-stream mode.

Returns

0 on success, or a negative error code:
-EINVAL if the virtual interface is not in packed-stream mode.

Get the parameters for packed-stream mode.

10.11.3.2 ef_vi_packed_stream_unbundle()

```
int ef_vi_packed_stream_unbundle (
            ef_vi * vi,
```


Copyright © Solarflare Communications 2017

```
const ef_event * ev,
ef_packed_stream_packet ** pkt_iter,
int * n_pkts_out,
int * n_bytes_out )
```

Unbundle an event of type EF_EVENT_TYPE_RX_PACKED_STREAM.

Parameters

vi	The virtual interface that has raised the event.
ev	The event, of type EF_EVENT_TYPE_RX_PACKED_STREAM.
pkt_iter	Pointer to an ef_packed_stream_packet*, that is updated on return with the value for the next call to this function. See below for more details.
n_pkts_out	Pointer to an int, that is updated on return with the number of packets unpacked.
n_bytes_out	Pointer to an int, that is updated on return with the number of bytes unpacked.

Returns

0 on success, or a negative error code.

Unbundle an event of type EF_EVENT_TYPE_RX_PACKED_STREAM.

This function should be called once for each EF_EVENT_TYPE_RX_PACKED_STREAM event received.

If EF_EVENT_RX_PS_NEXT_BUFFER(*ev) is true, *pkt_iter should be initialized to the value returned by ef_packed_stream_packet_first().

When EF_EVENT_RX_PS_NEXT_BUFFER(*ev) is not true, *pkt_iter should contain the value left by the previous call. After each call *pkt_iter points at the location where the next packet will be delivered.

The return value is 0, or a negative error code. If the error code is -ENOMSG, -ENODATA or -EL2NSYNC then there was a problem with the hardware timestamp: see ef_vi_receive_get_timestamp_with_sync_flags() for details.

10.12 pd.h File Reference

Protection Domains for EtherFabric Virtual Interface HAL.

```
#include <etherfabric/base.h>
```

Data Structures

- struct ef_pd

  A protection domain.

Macros

- #define EF_PD_VLAN_NONE -1

  May be passed to ef_pd_alloc_with_vport() to indicate that the PD is not associated with a particular VLAN.

 Copyright © Solarflare Communications 2017

Typedefs

- typedef struct ef_pd ef_pd
  A protection domain.

Enumerations

- enum ef_pd_flags {
  EF_PD_DEFAULT = 0x0, EF_PD_VF = 0x1, EF_PD_PHYS_MODE = 0x2, EF_PD_RX_PACKED_STREAM = 0x4,
  EF_PD_VPORT = 0x8, EF_PD_MCAST_LOOP = 0x10, EF_PD_MEMREG_64KiB = 0x20 }
  Flags for a protection domain.

Functions

- int ef_pd_alloc (ef_pd *pd, ef_driver_handle pd_dh, int ifindex, enum ef_pd_flags flags)
  Allocate a protection domain.
- int ef_pd_alloc_by_name (ef_pd *pd, ef_driver_handle pd_dh, const char *cluster_or_intf_name, enum ef_pd_flags flags)
  Allocate a protection domain for a named interface or cluster.
- int ef_pd_alloc_with_vport (ef_pd *pd, ef_driver_handle pd_dh, const char *intf_name, enum ef_pd_flags flags, int vlan_id)
  Allocate a protection domain with vport support.
- const char * ef_pd_interface_name (ef_pd *pd)
  Look up the interface being used by the protection domain.
- int ef_pd_free (ef_pd *pd, ef_driver_handle pd_dh)
  Free a protection domain.

10.12.1 Detailed Description

Protection Domains for EtherFabric Virtual Interface HAL.

Author

Solarflare Communications, Inc.

Date

2015/02/16

Copyright

Copyright © 2015 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.12.2 Enumeration Type Documentation

10.12.2.1 ef_pd_flags

```
enum ef_pd_flags
```

Flags for a protection domain.

Enumerator

EF_PD_DEFAULT	Default flags
EF_PD_VF	Protection domain uses a virtual function and the system IOMMU instead of NIC buffer table.
EF_PD_PHYS_MODE	Protection domain uses physical addressing mode
EF_PD_RX_PACKED_STREAM	Protection domain supports packed stream mode
EF_PD_VPORT	Protection domain supports virtual ports
EF_PD_MCAST_LOOP	Protection domain supports HW multicast loopback
EF_PD_MEMREG_64KiB	Protection domain uses >= 64KB registered memory mappings

Definition at line 48 of file pd.h.

10.12.3 Function Documentation

10.12.3.1 ef_pd_alloc()

```
int ef_pd_alloc (
        ef_pd * pd,
        ef_driver_handle pd_dh,
        int ifindex,
        enum ef_pd_flags flags )
```

Allocate a protection domain.

Parameters

pd	Memory to use for the allocated protection domain.
pd_dh	The ef_driver_handle to associate with the protection domain.
ifindex	Index of the interface to use for the protection domain.
flags	Flags to specify protection domain properties.

Returns

0 on success, or a negative error code.

Allocate a protection domain.

Allocates a 'protection domain' which specifies how memory should be protected for your VIs. For supported modes - see Packet Buffer Addressing

Note

If you are using a 'hardened' kernel (e.g. Gentoo-hardened) then this is the first call which will probably fail. Currently, the only workaround to this is to run as root.

Use "if_nametoindex" to find the index of an interface, which needs to be the physical interface (i.e. eth2, not eth2.6 or bond0 or similar.)

10.12.3.2 ef_pd_alloc_by_name()

```
int ef_pd_alloc_by_name (
            ef_pd * pd,
            ef_driver_handle pd_dh,
            const char * cluster_or_intf_name,
            enum ef_pd_flags flags )
```

Allocate a protection domain for a named interface or cluster.

Parameters

pd	Memory to use for the allocated protection domain.
pd_dh	An ef_driver_handle.
cluster_or_intf_name	Name of cluster, or name of interface.
flags	Flags to specify protection domain properties.

Returns

0 on success, or a negative error code.

Allocate a protection domain, trying first from a cluster of the given name, or if no cluster of that name exists assume that `cluster_or_intf_name` is the name of an interface.

When `cluster_or_intf_name` gives the name of a cluster it may optionally be prefixed with a channel number. For example: "0@cluster". In this case the specified channel instance within the cluster is allocated.

10.12.3.3 ef_pd_alloc_with_vport()

```
int ef_pd_alloc_with_vport (
            ef_pd * pd,
            ef_driver_handle pd_dh,
            const char * intf_name,
            enum ef_pd_flags flags,
            int vlan_id )
```

Allocate a protection domain with vport support.

Parameters

pd	Memory to use for the allocated protection domain.
pd_dh	The ef_driver_handle to associate with the protection domain.
intf_name	Name of interface to use for the protection domain.
flags	Flags to specify protection domain properties.
vlan_id	The vlan id to associate with the protection domain.

Copyright © Solarflare Communications 2017

Returns

0 on success, or a negative error code.

Allocate a protection domain with vport support.

10.12.3.4 ef_pd_free()

```
int ef_pd_free (
            ef_pd * pd,
            ef_driver_handle pd_dh )
```

Free a protection domain.

Parameters

pd	Memory used by the protection domain.
pd_dh	The ef_driver_handle associated with the protection domain.

Returns

0 on success, or a negative error code.

Free a protection domain.

To free up all resources, you must also close the associated driver handle.

You should call this when you're finished; although they will be cleaned up when the application exits, if you don't.

Be very sure that you don't try and re-use the vi/pd/driver structure after it has been freed.

10.12.3.5 ef_pd_interface_name()

```
const char* ef_pd_interface_name (
            ef_pd * pd )
```

Look up the interface being used by the protection domain.

Parameters

pd	Memory used by the protection domain.

Returns

The interface being used by the protection domain.

Look up the interface being used by the protection domain.

 Copyright © Solarflare Communications 2017

10.13 pio.h File Reference

Programmed Input/Output for EtherFabric Virtual Interface HAL.

```
#include <etherfabric/base.h>
```

Data Structures

- struct ef_pio

  A Programmed I/O region.

Typedefs

- typedef struct ef_pio ef_pio

  A Programmed I/O region.

Functions

- int ef_vi_get_pio_size (ef_vi *vi)

  Get the size of the Programmed I/O region.
- int ef_pio_free (ef_pio *pio, ef_driver_handle pio_dh)

  Free a Programmed I/O region.
- int ef_pio_link_vi (ef_pio *pio, ef_driver_handle pio_dh, struct ef_vi *vi, ef_driver_handle vi_dh)

  Link a Programmed I/O region with a virtual interface.
- int ef_pio_unlink_vi (ef_pio *pio, ef_driver_handle pio_dh, struct ef_vi *vi, ef_driver_handle vi_dh)

  Unlink a Programmed I/O region from a virtual interface.
- int ef_pio_memcpy (ef_vi *vi, const void *base, int offset, int len)

  Copy data from memory into a Programmed I/O region.

10.13.1 Detailed Description

Programmed Input/Output for EtherFabric Virtual Interface HAL.

Author

Solarflare Communications, Inc.

Date

2015/02/16

Copyright

Copyright © 2015 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.13.2 Function Documentation

10.13.2.1 ef_pio_free()

```
int ef_pio_free (
            ef_pio * pio,
            ef_driver_handle pio_dh )
```

Free a Programmed I/O region.

Parameters

pio	The Programmed I/O region.
pio_dh	The ef_driver_handle for the Programmed I/O region.

Returns

0 on success, or a negative error code.

Free a Programmed I/O region.

The Programmed I/O region must not be linked when this function is called. See ef_pio_unlink_vi().

To free up all resources, the associated driver handle must then be closed by calling ef_driver_close()).

10.13.2.2 ef_pio_link_vi()

```
int ef_pio_link_vi (
            ef_pio * pio,
            ef_driver_handle pio_dh,
            struct ef_vi * vi,
            ef_driver_handle vi_dh )
```

Link a Programmed I/O region with a virtual interface.

Parameters

pio	The Programmed I/O region.
pio_dh	The ef_driver_handle for the Programmed I/O region.
vi	The virtual interface to link with the Programmed I/O region.
vi_dh	The ef_driver_handle for the virtual interface.

Returns

0 on success, or a negative error code.

 Copyright © Solarflare Communications 2017

Link a Programmed I/O region with a virtual interface.

10.13.2.3 ef_pio_memcpy()

```
int ef_pio_memcpy (
            ef_vi * vi,
            const void * base,
            int offset,
            int len )
```

Copy data from memory into a Programmed I/O region.

Parameters

vi	The virtual interface for the Programmed I/O region.
base	The base address of the memory to copy.
offset	The offset into the Programmed I/O region at which to copy the data to. You shouldn't try to copy memory to part of the PIO region that is already in use for an ongoing send as this may result in corruption.
len	The number of bytes to copy.

Returns

0 on success, or a negative error code.

This function copies data from the user buffer to the adapter's PIO buffer. It goes via an intermediate buffer to meet any alignment requirements that the adapter may have.

Please refer to the PIO transmit functions (e.g.: ef_vi_transmit_pio() and ef_vi_transmit_copy_pio()) for alignment requirements of packets.

The Programmed I/O region can hold multiple smaller packets, referenced by different offset parameters. All other constraints must still be observed, including:

- alignment
- minimum size
- maximum size
- avoiding reuse until transmission is complete.

10.13.2.4 ef_pio_unlink_vi()

```
int ef_pio_unlink_vi (
            ef_pio * pio,
            ef_driver_handle pio_dh,
            struct ef_vi * vi,
            ef_driver_handle vi_dh )
```

Unlink a Programmed I/O region from a virtual interface.

 Copyright © Solarflare Communications 2017

Parameters

pio	The Programmed I/O region.
pio_dh	The ef_driver_handle for the Programmed I/O region.
vi	The virtual interface to unlink from the Programmed I/O region.
vi_dh	The ef_driver_handle for the virtual interface.

Returns

0 on success, or a negative error code.

Unlink a Programmed I/O region from a virtual interface.

10.13.2.5 ef_vi_get_pio_size()

```
int ef_vi_get_pio_size (
            ef_vi * vi )
```

Get the size of the Programmed I/O region.

Parameters

vi	The virtual interface to query.

Returns

The size of the Programmed I/O region.

Get the size of the Programmed I/O region.

10.14 timer.h File Reference

Timers for EtherFabric Virtual Interface HAL.

Macros

- #define ef_eventq_timer_prime(q, v) (q)->ops.eventq_timer_prime(q, v)

  Prime an event queue timer with a new timeout.
- #define ef_eventq_timer_run(q, v) (q)->ops.eventq_timer_run(q, v)

  Start an event queue timer running.
- #define ef_eventq_timer_clear(q) (q)->ops.eventq_timer_clear(q)

  Stop an event queue timer.
- #define ef_eventq_timer_zero(q) (q)->ops.eventq_timer_zero(q)

  Prime an event queue timer to expire immediately.

 Copyright © Solarflare Communications 2017

10.14.1 Detailed Description

Timers for EtherFabric Virtual Interface HAL.

Author

Solarflare Communications, Inc.

Date

2015/02/16

Copyright

Copyright © 2015 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.14.2 Macro Definition Documentation

10.14.2.1 ef_eventq_timer_clear

```
#define ef_eventq_timer_clear(
              q ) (q)->ops.eventq_timer_clear(q)
```

Stop an event queue timer.

Parameters

q	Pointer to ef_vi structure for the event queue.

Returns

None.

Stop an event queue timer.

The timer is stopped if it is already running, and no timeout-event is delivered.

The timer will not run when the next event arrives on the event queue.

Note

This is implemented as a macro, that calls the relevant function from the ef_vi::ops structure.

Definition at line 111 of file timer.h.

10.14.2.2 ef_eventq_timer_prime

```
#define ef_eventq_timer_prime(
            q,
            v ) (q)->ops.eventq_timer_prime(q, v)
```

Prime an event queue timer with a new timeout.

Parameters

q	Pointer to ef_vi structure for the event queue.
v	Initial value for timer (specified in μs).

Returns

None.

Prime an event queue timer with a new timeout.

The timer is stopped if it is already running, and no timeout-event is delivered.

The specified timeout is altered slightly, to avoid lots of timers going off in the same tick (bug1317). The timer is then primed with this new timeout.

The timer is then ready to run when the next event arrives on the event queue. When the timer-value reaches zero, a timeout-event will be delivered.

Note

This is implemented as a macro, that calls the relevant function from the ef_vi::ops structure.

Definition at line 70 of file timer.h.

10.14.2.3 ef_eventq_timer_run

```
#define ef_eventq_timer_run(
            q,
            v ) (q)->ops.eventq_timer_run(q, v)
```

Start an event queue timer running.

Parameters

q	Pointer to ef_vi structure for the event queue.
v	Initial value for timer (specified in μs).

Copyright © Solarflare Communications 2017

Returns

None.

Start an event queue timer running.

The timer is stopped if it is already running, and no timeout-event is delivered.

The specified timeout is altered slightly, to avoid lots of timers going off in the same tick (bug1317). The timer is then primed with this new timeout., and starts running immediately.

When the timer-value reaches zero, a timeout-event will be delivered.

Note

This is implemented as a macro, that calls the relevant function from the ef_vi::ops structure.

Definition at line 93 of file timer.h.

10.14.2.4 ef_eventq_timer_zero

```
#define ef_eventq_timer_zero(
              q ) (q)->ops.eventq_timer_zero(q)
```

Prime an event queue timer to expire immediately.

Parameters

q	Pointer to ef_vi structure for the event queue.

Returns

None.

Prime an event queue timer to expire immediately.

The timer is stopped if it is already running, and no timeout-event is delivered.

The timer is then primed with a new timeout of 0.

When the next event arrives on the event queue, a timeout-event will be delivered.

Note

This is implemented as a macro, that calls the relevant function from the ef_vi::ops structure.

Definition at line 132 of file timer.h.

 Copyright © Solarflare Communications 2017

10.15 vi.h File Reference

Virtual packet / DMA interface for EtherFabric Virtual Interface HAL.

```
#include <etherfabric/ef_vi.h>
#include <etherfabric/base.h>
```

Data Structures

- struct ef_vi_set

  A virtual interface set within a protection domain.
- struct ef_filter_spec

  Specification of a filter.
- struct ef_filter_cookie

  Cookie identifying a filter.
- struct ef_vi_stats_field_layout

  Layout for a field of statistics.
- struct ef_vi_stats_layout

  Layout for statistics.

Enumerations

- enum ef_filter_flags { EF_FILTER_FLAG_NONE = 0x0, EF_FILTER_FLAG_MCAST_LOOP_RECEIVE = 0x2 }

  Flags for a filter.
- enum { EF_FILTER_VLAN_ID_ANY = -1 }

  Virtual LANs for a filter.

Functions

- int ef_vi_alloc_from_pd (ef_vi ∗vi, ef_driver_handle vi_dh, struct ef_pd ∗pd, ef_driver_handle pd_dh, int evq←_capacity, int rxq_capacity, int txq_capacity, ef_vi ∗evq_opt, ef_driver_handle evq_dh, enum ef_vi_flags flags)

  Allocate a virtual interface from a protection domain.
- int ef_vi_free (ef_vi ∗vi, ef_driver_handle nic)

  Free a virtual interface.
- int ef_vi_transmit_alt_alloc (struct ef_vi ∗vi, ef_driver_handle vi_dh, int num_alts, size_t buf_space)

  Allocate a set of TX alternatives.
- int ef_vi_transmit_alt_free (struct ef_vi ∗vi, ef_driver_handle vi_dh)

  Free a set of TX alternatives.
- int ef_vi_transmit_alt_query_buffering (struct ef_vi ∗vi, int ifindex, ef_driver_handle vi_dh, int n_alts)

  Query available buffering.
- int ef_vi_flush (ef_vi ∗vi, ef_driver_handle nic)

  Flush the virtual interface.
- int ef_vi_pace (ef_vi ∗vi, ef_driver_handle nic, int val)

  Pace the virtual interface.
- unsigned ef_vi_mtu (ef_vi ∗vi, ef_driver_handle vi_dh)

Return the virtual interface MTU.

- int ef_vi_get_mac (ef_vi ∗vi, ef_driver_handle vi_dh, void ∗mac_out)

    Get the Ethernet MAC address for the virtual interface.
- int ef_eventq_put (unsigned resource_id, ef_driver_handle evq_dh, unsigned ev_bits)

    Send a software-generated event to an event queue.
- int ef_vi_set_alloc_from_pd (ef_vi_set ∗vi_set, ef_driver_handle vi_set_dh, struct ef_pd ∗pd, ef_driver_↩ handle pd_dh, int n_vis)

    Allocate a virtual interface set within a protection domain.
- int ef_vi_set_free (ef_vi_set ∗vi_set, ef_driver_handle vi_set_dh)

    Free a virtual interface set.
- int ef_vi_alloc_from_set (ef_vi ∗vi, ef_driver_handle vi_dh, ef_vi_set ∗vi_set, ef_driver_handle vi_set_dh, int index_in_vi_set, int evq_capacity, int rxq_capacity, int txq_capacity, ef_vi ∗evq_opt, ef_driver_handle evq_dh, enum ef_vi_flags flags)

    Allocate a virtual interface from a virtual interface set.
- int ef_vi_prime (ef_vi ∗vi, ef_driver_handle dh, unsigned current_ptr)

    Prime a virtual interface.
- void ef_filter_spec_init (ef_filter_spec ∗filter_spec, enum ef_filter_flags flags)

    Initialize an ef_filter_spec.
- int ef_filter_spec_set_ip4_local (ef_filter_spec ∗filter_spec, int protocol, unsigned host_be32, int port_be16)

    Set an IP4 Local filter on the filter specification.
- int ef_filter_spec_set_ip4_full (ef_filter_spec ∗filter_spec, int protocol, unsigned host_be32, int port_be16, unsigned rhost_be32, int rport_be16)

    Set an IP4 Full filter on the filter specification.
- int ef_filter_spec_set_ip6_local (ef_filter_spec ∗filter_spec, int protocol, const struct in6_addr ∗host, int port↩ _be16)

    Set an IP6 Local filter on the filter specification.
- int ef_filter_spec_set_ip6_full (ef_filter_spec ∗filter_spec, int protocol, const struct in6_addr ∗host, int port↩ _be16, const struct in6_addr ∗rhost, int rport_be16)

    Set an IP6 Full filter on the filter specification.
- int ef_filter_spec_set_vlan (ef_filter_spec ∗filter_spec, int vlan_id)

    Add a Virtual LAN filter on the filter specification.
- int ef_filter_spec_set_eth_local (ef_filter_spec ∗filter_spec, int vlan_id, const void ∗mac)

    Set an Ethernet MAC Address filter on the filter specification.
- int ef_filter_spec_set_unicast_all (ef_filter_spec ∗filter_spec)

    Set a Unicast All filter on the filter specification.
- int ef_filter_spec_set_multicast_all (ef_filter_spec ∗filter_spec)

    Set a Multicast All filter on the filter specification.
- int ef_filter_spec_set_unicast_mismatch (ef_filter_spec ∗filter_spec)

    Set a Unicast Mismatch filter on the filter specification.
- int ef_filter_spec_set_multicast_mismatch (ef_filter_spec ∗filter_spec)

    Set a Multicast Mismatch filter on the filter specification.
- int ef_filter_spec_set_port_sniff (ef_filter_spec ∗filter_spec, int promiscuous)

    Set a Port Sniff filter on the filter specification.
- int ef_filter_spec_set_tx_port_sniff (ef_filter_spec ∗filter_spec)

    Set a TX Port Sniff filter on the filter specification.
- int ef_filter_spec_set_block_kernel (ef_filter_spec ∗filter_spec)

    Set a Block Kernel filter on the filter specification.
- int ef_filter_spec_set_block_kernel_multicast (ef_filter_spec ∗filter_spec)

    Set a Block Kernel Multicast filter on the filter specification.
- int ef_filter_spec_set_eth_type (ef_filter_spec ∗filter_spec, uint16_t ether_type_be16)

Copyright © Solarflare Communications 2017

Add an EtherType filter on the filter specification.
- int ef_filter_spec_set_ip_proto (ef_filter_spec *filter_spec, uint8_t ip_proto)

  Add an IP protocol filter on the filter specification.
- int ef_filter_spec_set_block_kernel_unicast (ef_filter_spec *filter_spec)

  Set a Block Kernel Unicast filter on the filter specification.
- int ef_vi_filter_add (ef_vi *vi, ef_driver_handle vi_dh, const ef_filter_spec *filter_spec, ef_filter_cookie *filter_cookie_out)

  Add a filter to a virtual interface.
- int ef_vi_filter_del (ef_vi *vi, ef_driver_handle vi_dh, ef_filter_cookie *filter_cookie)

  Delete a filter from a virtual interface.
- int ef_vi_set_filter_add (ef_vi_set *vi_set, ef_driver_handle vi_set_dh, const ef_filter_spec *filter_spec, ef_filter_cookie *filter_cookie_out)

  Add a filter to a virtual interface set.
- int ef_vi_set_filter_del (ef_vi_set *vi_set, ef_driver_handle vi_set_dh, ef_filter_cookie *filter_cookie)

  Delete a filter from a virtual interface set.
- int ef_vi_stats_query_layout (ef_vi *vi, const ef_vi_stats_layout **const layout_out)

  Retrieve layout for available statistics.
- int ef_vi_stats_query (ef_vi *vi, ef_driver_handle vi_dh, void *data, int do_reset)

  Retrieve a set of statistic values.

10.15.1 Detailed Description

Virtual packet / DMA interface for EtherFabric Virtual Interface HAL.

Author

Solarflare Communications, Inc.

Date

2015/02/16

Copyright

Copyright © 2015 Solarflare Communications, Inc. All rights reserved. Solarflare, OpenOnload and EnterpriseOnload are trademarks of Solarflare Communications, Inc.

10.15.2 Enumeration Type Documentation

10.15.2.1 anonymous enum

```
anonymous enum
```

Virtual LANs for a filter.

Enumerator

EF_FILTER_VLAN_ID_ANY	Any Virtual LAN

Definition at line 479 of file vi.h.

10.15.2.2 ef_filter_flags

```
enum ef_filter_flags
```

Flags for a filter.

Enumerator

EF_FILTER_FLAG_NONE	No flags
EF_FILTER_FLAG_MCAST_LOOP_RECEIVE	If set, the filter will receive looped back packets for matching (see ef_filter_spec_set_tx_port_sniff())

Definition at line 460 of file vi.h.

10.15.3 Function Documentation

10.15.3.1 ef_eventq_put()

```
int ef_eventq_put (
            unsigned resource_id,
            ef_driver_handle evq_dh,
            unsigned ev_bits )
```

Send a software-generated event to an event queue.

Parameters

resource_id	The ID of the event queue.
evq_dh	The ef_driver_handle for the event queue.
ev_bits	Data for the event. The lowest 16 bits only are used, and all other bits must be clear.

Returns

0 on success, or a negative error code.

Send a software-generated event to an event queue.

An application can use this feature to put its own signals onto the event queue. For example, a thread might block waiting for events. An application could use a software-generated event to wake up the thread, so the thread could then process some non-ef_vi resources.

10.15.3.2 ef_filter_spec_init()

```
void ef_filter_spec_init (
            ef_filter_spec * filter_spec,
            enum ef_filter_flags flags )
```

Initialize an ef_filter_spec.

Parameters

filter_spec	The ef_filter_spec to initialize.
flags	The flags to set in the ef_filter_spec.

Returns

None.

Initialize an ef_filter_spec.

This function must be called to initialize a filter before calling the other filter functions.

The EF_FILTER_FLAG_MCAST_LOOP_RECEIVE flag does the following:

- if set, the filter will receive looped back packets for matching (see ef_filter_spec_set_tx_port_sniff())
- otherwise, the filter will not receive looped back packets.

10.15.3.3 ef_filter_spec_set_block_kernel()

```
int ef_filter_spec_set_block_kernel (
            ef_filter_spec * filter_spec )
```

Set a Block Kernel filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.

 Copyright © Solarflare Communications 2017

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set a Block Kernel filter on the filter specification.

This filter blocks all packets from reaching the kernel.

This filter is not supported by 5000-series and 6000-series adapters.

10.15.3.4 ef_filter_spec_set_block_kernel_multicast()

```
int ef_filter_spec_set_block_kernel_multicast (
            ef_filter_spec * filter_spec )
```

Set a Block Kernel Multicast filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set a Block Kernel Multicast filter on the filter specification.

This filter blocks all multicast packets from reaching the kernel.

This filter is not supported by 5000-series and 6000-series adapters.

10.15.3.5 ef_filter_spec_set_block_kernel_unicast()

```
int ef_filter_spec_set_block_kernel_unicast (
            ef_filter_spec * filter_spec )
```

Set a Block Kernel Unicast filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

 Copyright © Solarflare Communications 2017

Set a Block Kernel Unicast filter on the filter specification.

This filter blocks all unicast packets from reaching the kernel.

This filter is not supported by 5000-series and 6000-series adapters.

10.15.3.6 ef_filter_spec_set_eth_local()

```
int ef_filter_spec_set_eth_local (
            ef_filter_spec * filter_spec,
            int vlan_id,
            const void * mac )
```

Set an Ethernet MAC Address filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.
vlan_id	The ID of the virtual LAN on which to filter, or EF_FILTER_VLAN_ID_ANY to match all VLANs.
mac	The MAC address on which to filter, as a six-byte array.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set an Ethernet MAC Address filter on the filter specification.

This filter intercepts all packets that match the given MAC address and VLAN.

10.15.3.7 ef_filter_spec_set_eth_type()

```
int ef_filter_spec_set_eth_type (
            ef_filter_spec * filter_spec,
            uint16_t ether_type_be16 )
```

Add an EtherType filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.
ether_type_be16	The EtherType on which to filter, in network order.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

 Copyright © Solarflare Communications 2017

Add an EtherType filter on the filter specification

The EtherType filter can be combined with other filters as follows:

- Ethernet MAC Address filters: supported. See ef_filter_spec_set_eth_local().
- Other filters: not supported, -EPROTONOSUPPORT is returned.

This filter is not supported by 5000-series and 6000-series adapters. 7000-series adapters require a firmware version of at least v4.6 for full support for these filters. v4.5 firmware supports such filters only when not combined with a MAC address. Insertion of such filters on firmware versions that do not support them will fail.

Due to a current firmware limitation, this method does not support ether_type IP or IPv6 and will return no error if these values are specified.

10.15.3.8 ef_filter_spec_set_ip4_full()

```
int ef_filter_spec_set_ip4_full (
            ef_filter_spec * filter_spec,
            int protocol,
            unsigned host_be32,
            int port_be16,
            unsigned rhost_be32,
            int rport_be16 )
```

Set an IP4 Full filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.
protocol	The protocol on which to filter (IPPROTO_UDP or IPPROTO_TCP).
host_be32	The local host address on which to filter, as a 32-bit big-endian value (e.g. the output of htonl()).
port_be16	The local port on which to filter, as a 16-bit big-endian value (e.g. the output of htons()).
rhost_be32	The remote host address on which to filter, as a 32-bit big-endian value (e.g. the output of htonl()).
rport_be16	The remote port on which to filter, as a 16-bit big-endian value (e.g. the output of htons()).

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set an IP4 Full filter on the filter specification.

This filter intercepts all packets that match the given protocol and host/port combinations.

Note

You cannot specify a range, or a wildcard, for any parameter.

 Copyright © Solarflare Communications 2017

10.15.3.9 ef_filter_spec_set_ip4_local()

```
int ef_filter_spec_set_ip4_local (
            ef_filter_spec * filter_spec,
            int protocol,
            unsigned host_be32,
            int port_be16 )
```

Set an IP4 Local filter on the filter specification.

Set various types of filters on the filter spec

Parameters

filter_spec	The ef_filter_spec on which to set the filter.
protocol	The protocol on which to filter (IPPROTO_UDP or IPPROTO_TCP).
host_be32	The local host address on which to filter, as a 32-bit big-endian value (e.g. the output of htonl()).
port_be16	The local port on which to filter, as a 16-bit big-endian value (e.g. the output of htons()).

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set an IP4 Local filter on the filter specification.

This filter intercepts all packets that match the given protocol and host/port combination.

Note

You cannot specify a range, or a wildcard, for any parameter.

10.15.3.10 ef_filter_spec_set_ip6_full()

```
int ef_filter_spec_set_ip6_full (
            ef_filter_spec * filter_spec,
            int protocol,
            const struct in6_addr * host,
            int port_be16,
            const struct in6_addr * rhost,
            int rport_be16 )
```

Set an IP6 Full filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.
protocol	The protocol on which to filter (IPPROTO_UDP or IPPROTO_TCP).
host	The local host address on which to filter, as a pointer to a struct in6_addr.
port_be16	The local port on which to filter, as a 16-bit big-endian value (e.g. the output of htons()).
rhost	The remote host address on which to filter, as a pointer to a struct in6_addr.
rport_be16	The remote port on which to filter, as a 16-bit big-endian value (e.g. the output of htons()).

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set an IP6 Full filter on the filter specification.

This filter intercepts all packets that match the given protocol and host/port combinations.

Note

You cannot specify a range, or a wildcard, for any parameter.

10.15.3.11 ef_filter_spec_set_ip6_local()

```
int ef_filter_spec_set_ip6_local (
            ef_filter_spec * filter_spec,
            int protocol,
            const struct in6_addr * host,
            int port_be16 )
```

Set an IP6 Local filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.
protocol	The protocol on which to filter (IPPROTO_UDP or IPPROTO_TCP).
host	The local host address on which to filter, as a pointer to a struct in6_addr.
port_be16	The local port on which to filter, as a 16-bit big-endian value (e.g. the output of htons()).

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set an IP6 Local filter on the filter specification.

This filter intercepts all packets that match the given protocol and host/port combination.

Note

You cannot specify a range, or a wildcard, for any parameter.

Copyright © Solarflare Communications 2017

10.15.3.12 ef_filter_spec_set_ip_proto()

```
int ef_filter_spec_set_ip_proto (
            ef_filter_spec * filter_spec,
            uint8_t ip_proto )
```

Add an IP protocol filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.
ip_proto	The IP protocol on which to filter.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Add an IP protocol filter on the filter specification

The IP protocol filter can be combined with other filters as follows:

- Ethernet MAC Address filters: supported. See ef_filter_spec_set_eth_local().
- Other filters: not supported, -EPROTONOSUPPORT is returned.

This filter is not supported by 5000-series and 6000-series adapters. Other adapters require a firmware version of at least v4.5. Insertion of such filters on firmware versions that do not support them will fail.

Due to a current firmware limitation, this method does not support ip_proto=6 (TCP) or ip_proto=17 (UDP) and will return no error if these values are used.

10.15.3.13 ef_filter_spec_set_multicast_all()

```
int ef_filter_spec_set_multicast_all (
            ef_filter_spec * filter_spec )
```

Set a Multicast All filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set a Multicast All filter on the filter specification.

This filter must be used with caution. It intercepts all multicast packets that arrive, including IGMP group membership queries, which must normally be handled by the kernel to avoid any membership lapses.

 Copyright © Solarflare Communications 2017

10.15.3.14 ef_filter_spec_set_multicast_mismatch()

```
int ef_filter_spec_set_multicast_mismatch (
            ef_filter_spec * filter_spec )
```

Set a Multicast Mismatch filter on the filter specification.

 Copyright © Solarflare Communications 2017

Parameters

filter_spec	The ef_filter_spec on which to set the filter.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set a Multicast Mismatch filter on the filter specification.

This filter intercepts all multicast traffic that would otherwise be discarded; that is, all traffic that does not match either an existing multicast filter or a kernel subscription.

This filter is not supported by 5000-series and 6000-series adapters.

10.15.3.15 ef_filter_spec_set_port_sniff()

```
int ef_filter_spec_set_port_sniff (
            ef_filter_spec * filter_spec,
            int promiscuous )
```

Set a Port Sniff filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.
promiscuous	True to enable promiscuous mode on any virtual interface using this filter.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set a Port Sniff filter on the filter specification.

This filter enables sniff mode for the virtual interface. All filtering on that interface then copies packets instead of intercepting them. Consequently, the kernel receives the filtered packets; otherwise it would not.

If promiscuous mode is enabled, this filter copies all packets, instead of only those matched by other filters.

This filter is not supported by 5000-series and 6000-series adapters.

10.15.3.16 ef_filter_spec_set_tx_port_sniff()

```
int ef_filter_spec_set_tx_port_sniff (
            ef_filter_spec * filter_spec )
```

Set a TX Port Sniff filter on the filter specification.

 Copyright © Solarflare Communications 2017

Parameters

filter_spec	The ef_filter_spec on which to set the filter.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set a TX Port Sniff filter on the filter specification.

This filter loops back a copy of all outgoing packets, so that your application can process them.

This filter is not supported by 5000-series and 6000-series adapters.

10.15.3.17 ef_filter_spec_set_unicast_all()

```
int ef_filter_spec_set_unicast_all (
            ef_filter_spec * filter_spec )
```

Set a Unicast All filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set a Unicast All filter on the filter specification.

This filter must be used with caution. It intercepts all unicast packets that arrive, including ARP resolutions, which must normally be handled by the kernel for routing to work.

10.15.3.18 ef_filter_spec_set_unicast_mismatch()

```
int ef_filter_spec_set_unicast_mismatch (
            ef_filter_spec * filter_spec )
```

Set a Unicast Mismatch filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.

 Copyright © Solarflare Communications 2017

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Set a Unicast Mismatch filter on the filter specification.

This filter intercepts all unicast traffic that would otherwise be discarded; that is, all traffic that does not match either an existing unicast filter or a kernel subscription.

This filter is not supported by 5000-series and 6000-series adapters.

10.15.3.19 ef_filter_spec_set_vlan()

```
int ef_filter_spec_set_vlan (
            ef_filter_spec * filter_spec,
            int vlan_id )
```

Add a Virtual LAN filter on the filter specification.

Parameters

filter_spec	The ef_filter_spec on which to set the filter.
vlan_id	The ID of the virtual LAN on which to filter.

Returns

0 on success, or a negative error code:
-EPROTONOSUPPORT indicates that a filter is already set that is incompatible with the new filter.

Add a Virtual LAN filter on the filter specification.

The Virtual LAN filter can be combined with other filters as follows:

- Ethernet MAC Address filters: supported. See ef_filter_spec_set_eth_local().
- EtherType filters: supported.
- IP protocol filters: supported.
- IP4 filters:
  - 7000-series adapter with full feature firmware: supported. Packets that match the IP4 filter will be received only if they also match the VLAN.
  - Otherwise: not supported. Packets that match the IP4 filter will always be received, whatever the VLAN.
- Other filters: not supported, -EPROTONOSUPPORT is returned.

Copyright © Solarflare Communications 2017

10.15.3.20 ef_vi_alloc_from_pd()

```
int ef_vi_alloc_from_pd (
            ef_vi * vi,
            ef_driver_handle vi_dh,
            struct ef_pd * pd,
            ef_driver_handle pd_dh,
            int evq_capacity,
            int rxq_capacity,
            int txq_capacity,
            ef_vi * evq_opt,
            ef_driver_handle evq_dh,
            enum ef_vi_flags flags )
```

Allocate a virtual interface from a protection domain.

Parameters

vi	Memory for the allocated virtual interface.
vi_dh	The ef_driver_handle to associate with the virtual interface.
pd	The protection domain from which to allocate the virtual interface.
pd_dh	The ef_driver_handle to associate with the protection domain.
evq_capacity	The number of events in the event queue (maximum 4096), or: • 0 for no event queue • -1 for the default size (EF_VI_EVQ_SIZE if set, otherwise it is rxq_capacity + txq_capacity + extra bytes if timestamps are enabled).
rxq_capacity	The number of slots in the RX descriptor ring, or: • 0 for no event queue • -1 for the default size (EF_VI_RXQ_SIZE if set, otherwise 512).
txq_capacity	The number of slots in the TX descriptor ring, or: • 0 for no TX descriptor ring • -1 for the default size (EF_VI_TXQ_SIZE if set, otherwise 512).
evq_opt	event queue to use if evq_capacity=0.
evq_dh	The ef_driver_handle of the evq_opt event queue.
flags	Flags to select hardware attributes of the virtual interface.

Returns

>= 0 on success (value is Q_ID), or a negative error code.

Copyright © Solarflare Communications 2017

Allocate a virtual interface from a protection domain.

This allocates an RX and TX descriptor ring, an event queue, timers and interrupt etc. on the card. It also initializes the (opaque) structures needed to access them in software.

An existing virtual interface can be specified, to resize its descriptor rings and event queue.

When setting the sizes of the descriptor rings and event queue for a new or existing virtual interface:

- the event queue should be left at its default size unless extra rings are added
- if extra descriptor rings are added, the event queue should also be made correspondingly larger
- the maximum size of the event queue effectively limits how many descriptor ring slots can be supported without risking the event queue overflowing.

10.15.3.21 ef_vi_alloc_from_set()

```
int ef_vi_alloc_from_set (
            ef_vi * vi,
            ef_driver_handle vi_dh,
            ef_vi_set * vi_set,
            ef_driver_handle vi_set_dh,
            int index_in_vi_set,
            int evq_capacity,
            int rxq_capacity,
            int txq_capacity,
            ef_vi * evq_opt,
            ef_driver_handle evq_dh,
            enum ef_vi_flags flags )
```

Allocate a virtual interface from a virtual interface set.

Parameters

vi	Memory for the allocated virtual interface.
vi_dh	The ef_driver_handle to associate with the virtual interface.
vi_set	The virtual interface set from which to allocate the virtual interface.
vi_set_dh	The ef_driver_handle to associate with the virtual interface set.
index_in_vi_set	Index of the virtual interface within the set to allocate, or -1 for any.
evq_capacity	The number of events in the event queue (maximum 4096), or: • 0 for no event queue • -1 for the default size.

Parameters

rxq_capacity	The number of slots in the RX descriptor ring, or: • 0 for no RX queue • -1 for the default size (EF_VI_RXQ_SIZE if set, otherwise 512).
txq_capacity	The number of slots in the TX descriptor ring, or: • 0 for no TX queue • -1 for the default size (EF_VI_TXQ_SIZE if set, otherwise 512).
evq_opt	event queue to use if evq_capacity=0.
evq_dh	The ef_driver_handle of the evq_opt event queue.
flags	Flags to select hardware attributes of the virtual interface.

Returns

>= 0 on success (value is Q_ID), or a negative error code.

Allocate a virtual interface from a virtual interface set.

This allocates an RX and TX descriptor ring, an event queue, timers and interrupt etc. on the card. It also initializes the (opaque) structures needed to access them in software.

An existing virtual interface can be specified, to resize its descriptor rings and event queue.

When setting the sizes of the descriptor rings and event queue for a new or existing virtual interface:

- the event queue should be left at its default size unless extra rings are added
- if extra descriptor rings are added, the event queue should also be made correspondingly larger
- the maximum size of the event queue effectively limits how many descriptor ring slots can be supported without risking the event queue overflowing.

10.15.3.22 ef_vi_filter_add()

```
int ef_vi_filter_add (
            ef_vi * vi,
            ef_driver_handle vi_dh,
            const ef_filter_spec * filter_spec,
            ef_filter_cookie * filter_cookie_out )
```

Add a filter to a virtual interface.

 Copyright © Solarflare Communications 2017

Parameters

vi	The virtual interface on which to add the filter.
vi_dh	The ef_driver_handle for the virtual interface.
filter_spec	The filter to add.
filter_cookie_out	Optional pointer to an ef_filter_cookie, that is updated on return with a cookie for the filter.

Returns

0 on success, or a negative error code.

Add a filter to a virtual interface.

filter_cookie_out can be NULL. If not null, then the returned value can be used in ef_vi_filter_del() to remove this filter.

After calling this function, any local copy of the filter can be deleted.

10.15.3.23 ef_vi_filter_del()

```
int ef_vi_filter_del (
            ef_vi * vi,
            ef_driver_handle vi_dh,
            ef_filter_cookie * filter_cookie )
```

Delete a filter from a virtual interface.

Parameters

vi	The virtual interface from which to delete the filter.
vi_dh	The ef_driver_handle for the virtual interface.
filter_cookie	The filter cookie for the filter to delete, as set on return from ef_vi_filter_add().

Returns

0 on success, or a negative error code.

Delete a filter from a virtual interface.

10.15.3.24 ef_vi_flush()

```
int ef_vi_flush (
            ef_vi * vi,
            ef_driver_handle nic )
```

Flush the virtual interface.

Copyright © Solarflare Communications 2017

Parameters

vi	The virtual interface to flush.
nic	The ef_driver_handle for the NIC hosting the interface.

Returns

0 on success, or a negative error code.

Flush the virtual interface.

After this function returns, it is safe to reuse all buffers which have been pushed onto the NIC.

10.15.3.25 ef_vi_free()

```
int ef_vi_free (
            ef_vi * vi,
            ef_driver_handle nic )
```

Free a virtual interface.

Parameters

vi	The virtual interface to free.
nic	The ef_driver_handle for the NIC hosting the interface.

Returns

0 on success, or a negative error code.

Free a virtual interface.

This should be called when a virtual interface is no longer needed.

To free up all resources, you must also close the associated driver handle using ef_driver_close and free up memory from the protection domain ef_pd_free. See Freeing Resources

If successful:

- the memory for state provided for this virtual interface is no longer required
- no further events from this virtual interface will be delivered to its event queue.

10.15.3.26 ef_vi_get_mac()

```
int ef_vi_get_mac (
            ef_vi * vi,
            ef_driver_handle vi_dh,
            void * mac_out )
```

Get the Ethernet MAC address for the virtual interface.

Parameters

vi	The virtual interface to query.
vi_dh	The ef_driver_handle for the NIC hosting the interface.
mac_out	Pointer to a six-byte buffer, that is updated on return with the Ethernet MAC address.

Returns

0 on success, or a negative error code.

Get the Ethernet MAC address for the virtual interface.

This is not a cheap call, so cache the result if you care about performance.

10.15.3.27 ef_vi_mtu()

```
unsigned ef_vi_mtu (
             ef_vi * vi,
             ef_driver_handle vi_dh )
```

Return the virtual interface MTU.

Parameters

vi	The virtual interface to query.
vi_dh	The ef_driver_handle for the NIC hosting the interface.

Returns

The virtual interface Maximum Transmission Unit.

Return the virtual interface MTU. (This is the maximum size of Ethernet frames that can be transmitted through, and received by the interface).

The returned value is the total frame size, including all headers, but not including the Ethernet frame check.

10.15.3.28 ef_vi_pace()

```
int ef_vi_pace (
             ef_vi * vi,
             ef_driver_handle nic,
             int val )
```

Pace the virtual interface.

Parameters

vi	The virtual interface to pace.
nic	The ef_driver_handle for the NIC hosting the interface.
val	The minimum inter-packet gap for the TXQ.

Copyright © Solarflare Communications 2017

Returns

0 on success, or a negative error code.

Pace the virtual interface.

This sets a minimum inter-packet gap for the TXQ:

- if val is -1 then the TXQ is put into the "pacing" bin, but no gap is enforced
- otherwise, the gap is $(2^{val})*100$ns.

This can be used to give priority to latency sensitive traffic over bulk traffic.

10.15.3.29 ef_vi_prime()

```
int ef_vi_prime (
            ef_vi * vi,
            ef_driver_handle dh,
            unsigned current_ptr )
```

Prime a virtual interface.

Parameters

vi	The virtual interface to prime.
dh	The ef_driver_handle to associate with the virtual interface.
current_ptr	Value returned from ef_eventq_current().

Returns

0 on success, or a negative error code.

Prime a virtual interface. This enables interrupts so you can block on the file descriptor associated with the ef_driver_handle using select/poll/epoll, etc.

Passing the current event queue pointer ensures correct handling of any events that occur between this prime and the epoll_wait call.

10.15.3.30 ef_vi_set_alloc_from_pd()

```
int ef_vi_set_alloc_from_pd (
            ef_vi_set * vi_set,
            ef_driver_handle vi_set_dh,
            struct ef_pd * pd,
            ef_driver_handle pd_dh,
            int n_vis )
```

Allocate a virtual interface set within a protection domain.

 Copyright © Solarflare Communications 2017

Parameters

vi_set	Memory for the allocated virtual interface set.
vi_set_dh	The ef_driver_handle to associate with the virtual interface set.
pd	The protection domain from which to allocate the virtual interface set.
pd_dh	The ef_driver_handle of the associated protection domain.
n_vis	The number of virtual interfaces in the virtual interface set.

Returns

0 on success, or a negative error code.

Allocate a virtual interface set within a protection domain.

A virtual interface set is usually used to spread the load of handling received packets. This is sometimes called receive-side scaling, or RSS.

10.15.3.31 ef_vi_set_filter_add()

```
int ef_vi_set_filter_add (
            ef_vi_set * vi_set,
            ef_driver_handle vi_set_dh,
            const ef_filter_spec * filter_spec,
            ef_filter_cookie * filter_cookie_out )
```

Add a filter to a virtual interface set.

Parameters

vi_set	The virtual interface set on which to add the filter.
vi_set_dh	The ef_driver_handle for the virtual interface set.
filter_spec	The filter to add.
filter_cookie_out	Optional pointer to an ef_filter_cookie, that is updated on return with a cookie for the filter.

Returns

0 on success, or a negative error code:
Add a filter to a virtual interface set.

filter_cookie_out can be NULL. If not null, then the returned value can be used in ef_vi_filter_del() to delete this filter.

After calling this function, any local copy of the filter can be deleted.

10.15.3.32 ef_vi_set_filter_del()

```
int ef_vi_set_filter_del (
            ef_vi_set * vi_set,
```

 Copyright © Solarflare Communications 2017

```
ef_driver_handle vi_set_dh,
ef_filter_cookie * filter_cookie )
```

Delete a filter from a virtual interface set.

 Copyright © Solarflare Communications 2017

Parameters

vi_set	The virtual interface set from which to delete the filter.
vi_set_dh	The ef_driver_handle for the virtual interface set.
filter_cookie	The filter cookie for the filter to delete.

Returns

0 on success, or a negative error code.

Delete a filter from a virtual interface set.

10.15.3.33 ef_vi_set_free()

```
int ef_vi_set_free (
            ef_vi_set * vi_set,
            ef_driver_handle vi_set_dh )
```

Free a virtual interface set.

Parameters

vi_set	Memory for the allocated virtual interface set.
vi_set_dh	The ef_driver_handle to associate with the virtual interface set.

Returns

0 on success, or a negative error code.

Free a virtual interface set.

To free up all resources, you must also close the associated driver handle.

10.15.3.34 ef_vi_stats_query()

```
int ef_vi_stats_query (
            ef_vi * vi,
            ef_driver_handle vi_dh,
            void * data,
            int do_reset )
```

Retrieve a set of statistic values.

Parameters

vi	The virtual interface to query.
vi_dh	The ef_driver_handle for the virtual interface.
data	Pointer to a buffer, into which the statistics are retrieved. The size of this buffer should be equal to the evsl_data_bytes field of the layout description, that can be fetched using ef_vi_stats_query_layout().
do_reset	True to reset the statistics after retrieving them.

Returns

0 on success, or a negative error code.

Retrieve a set of statistic values.

If do_reset is true, the statistics are reset after reading.

10.15.3.35 ef_vi_stats_query_layout()

```
int ef_vi_stats_query_layout (
            ef_vi * vi,
            const ef_vi_stats_layout **const layout_out )
```

Retrieve layout for available statistics.

Parameters

vi	The virtual interface to query.
layout_out	Pointer to an ef_vi_stats_layout*, that is updated on return with the layout for available statistics.

Returns

0 on success, or a negative error code.

Retrieve layout for available statistics.

10.15.3.36 ef_vi_transmit_alt_alloc()

```
int ef_vi_transmit_alt_alloc (
            struct ef_vi * vi,
            ef_driver_handle vi_dh,
            int num_alts,
            size_t buf_space )
```

Allocate a set of TX alternatives.

Parameters

vi	The virtual interface that is to use the TX alternatives.
vi_dh	The ef_driver_handle for the NIC hosting the interface.
num_alts	The number of TX alternatives for which to allocate space.
buf_space	The buffer space required for the set of TX alternatives, in bytes.

Returns

0 Success.

 Copyright © Solarflare Communications 2017

-EINVAL The num_alts or buf_space parameters are invalid, or the VI was allocated without EF_VI_TX_ALT set.
-EALREADY A set of TX alternatives has already been allocated for use with this VI.
-ENOMEM Insufficient memory was available (either host memory or packet buffers on the adapter).
-EBUSY Too many alternatives requested, or alternatives requested on too many distinct VIs.

Allocate a set of TX alternatives for use with a virtual interface. The virtual interface must have been allocated with the EF_VI_TX_ALT flag.

The space remains allocated until ef_vi_transmit_alt_free() is called or the virtual interface is freed.

TX alternatives provide a mechanism to send with very low latency. They work by pre-loading packets into the adapter in advance, and then calling ef_vi_transmit_alt_go() to transmit the packets.

Packets are pre-loaded into the adapter using normal send calls such as ef_vi_transmit(). Use ef_vi_transmit_alt_select() to select which "alternative" to load the packet into.

Each alternative has three states: STOP, GO and DISCARD. The ef_vi_transmit_alt_stop(), ef_vi_transmit_alt_go() and ef_vi_transmit_alt_discard() calls transition between the states. Typically an alternative is placed in the STOP state, selected, pre-loaded with one or more packets, and then later on the critical path placed in the GO state.

When packets are transmitted via TX alternatives, events of type EF_EVENT_TYPE_TX_ALT are returned to the application. The application is responsible for ensuring that all of the packets in an alternative have been sent before transitioning from GO or DISCARD to the STOP state.

The `buf_space` parameter gives the amount of buffering to allocate for this set of TX alternatives, in bytes. Note that if this buffering is exceeded then packets sent to TX alternatives may be truncated or dropped, and no error is reported in this case.

10.15.3.37 ef_vi_transmit_alt_free()

```
int ef_vi_transmit_alt_free (
            struct ef_vi * vi,
            ef_driver_handle vi_dh )
```

Free a set of TX alternatives.

Parameters

vi	The virtual interface whose alternatives are to be freed.
vi_dh	The ef_driver_handle for the NIC hosting the interface.

Returns

0 on success, or a negative error code.

Release the set of TX alternatives allocated by ef_vi_transmit_alt_alloc().

10.15.3.38 ef_vi_transmit_alt_query_buffering()

```
int ef_vi_transmit_alt_query_buffering (
            struct ef_vi * vi,
```

 Copyright © Solarflare Communications 2017

```
int ifindex,
ef_driver_handle vi_dh,
int n_alts )
```

Query available buffering.

Parameters

vi	Interface to be queried
ifindex	The index of the interface that you wish to query. You can use if_nametoindex() to obtain this. This should be the underlying physical interface, rather than a bond, VLAN, or similar.
vi_dh	The ef_driver_handle for the NIC hosting the interface.
n_alts	Intended number of alternatives

Returns

-EINVAL if this VI doesn't support alternatives, else the number of bytes available

Owing to per-packet and other overheads, the amount of data which can be stored in TX alternatives is generally slightly less than the amount of memory available on the hardware.

This function allows the caller to find out how much user-visible buffering will be available if the given number of alternatives are allocated on the given VI.

 Copyright © Solarflare Communications 2017

Index

000_main.dox, 89
010_overview.dox, 89
020_concepts.dox, 90
030_apps.dox, 90
040_using.dox, 91
050_examples.dox, 91

added
    ef_vi_rxq_state, 70
    ef_vi_txq_state, 81
arch
    ef_vi_nic_type, 68

base.h, 92
    ef_driver_close, 93
    ef_driver_open, 93
    ef_eventq_wait, 94
bytes_acc
    ef_vi_rxq_state, 70

capabilities.h, 94
    ef_vi_capabilities_get, 97
    ef_vi_capabilities_max, 97
    ef_vi_capabilities_name, 98
    ef_vi_capability, 96

data
    ef_filter_spec, 48
descriptors
    ef_vi_rxq, 69
    ef_vi_txq, 79

EF_EVENT_RX_MULTI_CONT
    ef_vi.h, 105
EF_EVENT_RX_MULTI_SOP
    ef_vi.h, 105
EF_EVENT_RX_PS_NEXT_BUFFER
    ef_vi.h, 105
EF_VI_ALIGN
    ef_iovec, 49
EF_VI_PS_FLAG_BAD_IP_CSUM
    packedstream.h, 142
ef_driver_close
    base.h, 93
ef_driver_open
    base.h, 93
ef_event, 41
    generic, 43
    rx, 43
    rx_discard, 43
    rx_multi, 43
    rx_multi_discard, 43
    rx_no_desc_trunc, 43
    rx_packed_stream, 44
    sw, 44
    tx, 44
    tx_alt, 44
    tx_error, 44
    tx_timestamp, 44
ef_eventq_capacity
    ef_vi.h, 121
ef_eventq_current
    ef_vi.h, 121
ef_eventq_has_event
    ef_vi.h, 122
ef_eventq_has_many_events
    ef_vi.h, 122
ef_eventq_poll
    ef_vi.h, 106
ef_eventq_prime
    ef_vi.h, 106
ef_eventq_put
    vi.h, 158
ef_eventq_state, 45
    evq_ptr, 45
    sync_flags, 45
    sync_timestamp_major, 45
    sync_timestamp_minimum, 46
    sync_timestamp_minor, 46
    sync_timestamp_synchronised, 46
ef_eventq_timer_clear
    timer.h, 152
ef_eventq_timer_prime
    timer.h, 152
ef_eventq_timer_run
    timer.h, 153
ef_eventq_timer_zero
    timer.h, 154
ef_eventq_wait
    base.h, 94
ef_filter_cookie, 46
    filter_id, 47
    filter_type, 47
ef_filter_flags
    vi.h, 158

 Copyright © Solarflare Communications 2017

ef_filter_spec, 47
    data, 48
    flags, 48
    type, 48
ef_filter_spec_init
    vi.h, 159
ef_filter_spec_set_block_kernel
    vi.h, 159
ef_filter_spec_set_block_kernel_multicast
    vi.h, 160
ef_filter_spec_set_block_kernel_unicast
    vi.h, 160
ef_filter_spec_set_eth_local
    vi.h, 161
ef_filter_spec_set_eth_type
    vi.h, 161
ef_filter_spec_set_ip4_full
    vi.h, 162
ef_filter_spec_set_ip4_local
    vi.h, 162
ef_filter_spec_set_ip6_full
    vi.h, 163
ef_filter_spec_set_ip6_local
    vi.h, 164
ef_filter_spec_set_ip_proto
    vi.h, 164
ef_filter_spec_set_multicast_all
    vi.h, 166
ef_filter_spec_set_multicast_mismatch
    vi.h, 166
ef_filter_spec_set_port_sniff
    vi.h, 168
ef_filter_spec_set_tx_port_sniff
    vi.h, 168
ef_filter_spec_set_unicast_all
    vi.h, 169
ef_filter_spec_set_unicast_mismatch
    vi.h, 169
ef_filter_spec_set_vlan
    vi.h, 170
ef_iovec, 49
    EF_VI_ALIGN, 49
    iov_len, 49
ef_memreg, 50
    mr_dma_addrs, 50
    mr_dma_addrs_base, 50
ef_memreg_alloc
    memreg.h, 139
ef_memreg_dma_addr
    memreg.h, 139
ef_memreg_free
    memreg.h, 140
ef_packed_stream_packet, 51
    ps_cap_len, 51
    ps_flags, 51
    ps_next_offset, 51
    ps_orig_len, 52
    ps_pkt_start_offset, 52
    ps_ts_nsec, 52
    ps_ts_sec, 52
ef_packed_stream_params, 53
    psp_buffer_align, 53
    psp_buffer_size, 53
    psp_max_usable_buffers, 53
    psp_start_offset, 54
ef_pd, 54
    pd_cluster_dh, 55
    pd_cluster_name, 55
    pd_cluster_sock, 55
    pd_cluster_viset_index, 55
    pd_cluster_viset_resource_id, 55
    pd_flags, 56
    pd_intf_name, 56
    pd_resource_id, 56
ef_pd_alloc
    pd.h, 145
ef_pd_alloc_by_name
    pd.h, 145
ef_pd_alloc_with_vport
    pd.h, 146
ef_pd_flags
    pd.h, 144
ef_pd_free
    pd.h, 147
ef_pd_interface_name
    pd.h, 147
ef_pio, 56
    pio_buffer, 57
    pio_io, 57
    pio_len, 57
    pio_resource_id, 57
ef_pio_free
    pio.h, 149
ef_pio_link_vi
    pio.h, 149
ef_pio_memcpy
    pio.h, 150
ef_pio_unlink_vi
    pio.h, 150
ef_request_id
    ef_vi.h, 117
ef_vi, 58
    ef_vi.h, 117
    ep_state, 59
    evq_base, 59
    evq_mask, 59
    inited, 60
    io, 60
    linked_pio, 60
    nic_type, 60
    ops, 60
    rx_buffer_len, 61

 Copyright © Solarflare Communications 2017

rx_discard_mask, 61
rx_prefix_len, 61
rx_ts_correction, 61
timer_quantum_ns, 61
tx_alt_hw2id, 62
tx_alt_id2hw, 62
tx_alt_num, 62
tx_push_thresh, 62
tx_ts_correction_ns, 62
vi_clustered, 63
vi_flags, 63
vi_i, 63
vi_io_mmap_bytes, 63
vi_io_mmap_ptr, 63
vi_is_normal, 64
vi_is_packed_stream, 64
vi_mem_mmap_bytes, 64
vi_mem_mmap_ptr, 64
vi_out_flags, 64
vi_ps_buf_size, 65
vi_qs, 65
vi_qs_n, 65
vi_resource_id, 65
vi_rxq, 65
vi_stats, 66
vi_txq, 66
ef_vi.h, 98
EF_EVENT_RX_MULTI_CONT, 105
EF_EVENT_RX_MULTI_SOP, 105
EF_EVENT_RX_PS_NEXT_BUFFER, 105
ef_eventq_capacity, 121
ef_eventq_current, 121
ef_eventq_has_event, 122
ef_eventq_has_many_events, 122
ef_eventq_poll, 106
ef_eventq_prime, 106
ef_request_id, 117
ef_vi, 117
ef_vi_arch, 118
ef_vi_driver_interface_str, 123
ef_vi_flags, 119, 123
ef_vi_instance, 123
ef_vi_layout_type, 120
ef_vi_out_flags, 120
ef_vi_receive_buffer_len, 124
ef_vi_receive_capacity, 124
ef_vi_receive_fill_level, 125
ef_vi_receive_get_bytes, 125
ef_vi_receive_get_timestamp, 126
ef_vi_receive_get_timestamp_with_sync_flags, 126
ef_vi_receive_init, 107
ef_vi_receive_post, 127
ef_vi_receive_prefix_len, 128
ef_vi_receive_push, 107
ef_vi_receive_query_layout, 129
ef_vi_receive_set_buffer_len, 129
ef_vi_receive_set_discards, 130
ef_vi_receive_space, 130
ef_vi_receive_unbundle, 130
ef_vi_resource_id, 131
ef_vi_rx_discard_err_flags, 120
ef_vi_set_tx_push_threshold, 132
ef_vi_transmit, 108
ef_vi_transmit_alt_discard, 108
ef_vi_transmit_alt_go, 109
ef_vi_transmit_alt_num_ids, 132
ef_vi_transmit_alt_query_overhead, 132
ef_vi_transmit_alt_select, 110
ef_vi_transmit_alt_select_normal, 110
ef_vi_transmit_alt_stop, 111
ef_vi_transmit_alt_usage, 133
ef_vi_transmit_capacity, 133
ef_vi_transmit_copy_pio, 111
ef_vi_transmit_copy_pio_warm, 112
ef_vi_transmit_fill_level, 134
ef_vi_transmit_init, 134
ef_vi_transmit_init_undo, 135
ef_vi_transmit_pio, 113
ef_vi_transmit_pio_warm, 114
ef_vi_transmit_push, 114
ef_vi_transmit_space, 135
ef_vi_transmit_unbundle, 137
ef_vi_transmitv, 115
ef_vi_transmitv_init, 116
ef_vi_version_str, 137
ef_vi::ops, 82
eventq_poll, 83
eventq_prime, 83
eventq_timer_clear, 83
eventq_timer_prime, 83
eventq_timer_run, 83
eventq_timer_zero, 84
receive_init, 84
receive_push, 84
transmit, 84
transmit_alt_discard, 84
transmit_alt_go, 85
transmit_alt_select, 85
transmit_alt_select_default, 85
transmit_alt_stop, 85
transmit_copy_pio, 85
transmit_copy_pio_warm, 86
transmit_pio, 86
transmit_pio_warm, 86
transmit_push, 86
transmitv, 86
transmitv_init, 87
ef_vi_alloc_from_pd
vi.h, 170
ef_vi_alloc_from_set
vi.h, 172

 Copyright © Solarflare Communications 2017

ef_vi_arch
ef_vi.h, 118
ef_vi_capabilities_get
capabilities.h, 97
ef_vi_capabilities_max
capabilities.h, 97
ef_vi_capabilities_name
capabilities.h, 98
ef_vi_capability
capabilities.h, 96
ef_vi_driver_interface_str
ef_vi.h, 123
ef_vi_filter_add
vi.h, 173
ef_vi_filter_del
vi.h, 174
ef_vi_flags
ef_vi.h, 119, 123
ef_vi_flush
vi.h, 174
ef_vi_free
vi.h, 175
ef_vi_get_mac
vi.h, 175
ef_vi_get_pio_size
pio.h, 151
ef_vi_instance
ef_vi.h, 123
ef_vi_layout_entry, 66
evle_description, 67
evle_offset, 67
evle_type, 67
ef_vi_layout_type
ef_vi.h, 120
ef_vi_mtu
vi.h, 176
ef_vi_nic_type, 67
arch, 68
revision, 68
variant, 68
ef_vi_out_flags
ef_vi.h, 120
ef_vi_pace
vi.h, 176
ef_vi_packed_stream_get_params
packedstream.h, 142
ef_vi_packed_stream_unbundle
packedstream.h, 142
ef_vi_prime
vi.h, 177
ef_vi_receive_buffer_len
ef_vi.h, 124
ef_vi_receive_capacity
ef_vi.h, 124
ef_vi_receive_fill_level
ef_vi.h, 125
ef_vi_receive_get_bytes
ef_vi.h, 125
ef_vi_receive_get_timestamp
ef_vi.h, 126
ef_vi_receive_get_timestamp_with_sync_flags
ef_vi.h, 126
ef_vi_receive_init
ef_vi.h, 107
ef_vi_receive_post
ef_vi.h, 127
ef_vi_receive_prefix_len
ef_vi.h, 128
ef_vi_receive_push
ef_vi.h, 107
ef_vi_receive_query_layout
ef_vi.h, 129
ef_vi_receive_set_buffer_len
ef_vi.h, 129
ef_vi_receive_set_discards
ef_vi.h, 130
ef_vi_receive_space
ef_vi.h, 130
ef_vi_receive_unbundle
ef_vi.h, 130
ef_vi_resource_id
ef_vi.h, 131
ef_vi_rx_discard_err_flags
ef_vi.h, 120
ef_vi_rxq, 69
descriptors, 69
ids, 69
mask, 69
ef_vi_rxq_state, 70
added, 70
bytes_acc, 70
in_jumbo, 71
last_desc_i, 71
posted, 71
removed, 71
rx_ps_credit_avail, 71
ef_vi_set, 72
vis_pd, 72
vis_res_id, 72
ef_vi_set_alloc_from_pd
vi.h, 177
ef_vi_set_filter_add
vi.h, 178
ef_vi_set_filter_del
vi.h, 178
ef_vi_set_free
vi.h, 180
ef_vi_set_tx_push_threshold
ef_vi.h, 132
ef_vi_state, 73
evq, 73
rxq, 73

 Copyright © Solarflare Communications 2017

txq, 74
ef_vi_stats, 74
evq_gap, 74
rx_ev_bad_desc_i, 75
rx_ev_bad_q_label, 75
rx_ev_lost, 75
ef_vi_stats_field_layout, 75
evsfl_name, 76
evsfl_offset, 76
evsfl_size, 76
ef_vi_stats_layout, 77
evsl_data_size, 77
evsl_fields, 77
evsl_fields_num, 77
ef_vi_stats_query
vi.h, 180
ef_vi_stats_query_layout
vi.h, 181
ef_vi_transmit
ef_vi.h, 108
ef_vi_transmit_alt_alloc
vi.h, 181
ef_vi_transmit_alt_discard
ef_vi.h, 108
ef_vi_transmit_alt_free
vi.h, 182
ef_vi_transmit_alt_go
ef_vi.h, 109
ef_vi_transmit_alt_num_ids
ef_vi.h, 132
ef_vi_transmit_alt_overhead, 78
mask, 78
post_round, 78
pre_round, 79
ef_vi_transmit_alt_query_buffering
vi.h, 182
ef_vi_transmit_alt_query_overhead
ef_vi.h, 132
ef_vi_transmit_alt_select
ef_vi.h, 110
ef_vi_transmit_alt_select_normal
ef_vi.h, 110
ef_vi_transmit_alt_stop
ef_vi.h, 111
ef_vi_transmit_alt_usage
ef_vi.h, 133
ef_vi_transmit_capacity
ef_vi.h, 133
ef_vi_transmit_copy_pio
ef_vi.h, 111
ef_vi_transmit_copy_pio_warm
ef_vi.h, 112
ef_vi_transmit_fill_level
ef_vi.h, 134
ef_vi_transmit_init
ef_vi.h, 134
ef_vi_transmit_init_undo
ef_vi.h, 135
ef_vi_transmit_pio
ef_vi.h, 113
ef_vi_transmit_pio_warm
ef_vi.h, 114
ef_vi_transmit_push
ef_vi.h, 114
ef_vi_transmit_space
ef_vi.h, 135
ef_vi_transmit_unbundle
ef_vi.h, 137
ef_vi_transmitv
ef_vi.h, 115
ef_vi_transmitv_init
ef_vi.h, 116
ef_vi_txq, 79
descriptors, 79
ids, 80
mask, 80
ef_vi_txq_state, 80
added, 81
previous, 81
removed, 81
ts_nsec, 81
ef_vi_version_str
ef_vi.h, 137
ep_state
ef_vi, 59
eventq_poll
ef_vi::ops, 83
eventq_prime
ef_vi::ops, 83
eventq_timer_clear
ef_vi::ops, 83
eventq_timer_prime
ef_vi::ops, 83
eventq_timer_run
ef_vi::ops, 83
eventq_timer_zero
ef_vi::ops, 84
evle_description
ef_vi_layout_entry, 67
evle_offset
ef_vi_layout_entry, 67
evle_type
ef_vi_layout_entry, 67
evq
ef_vi_state, 73
evq_base
ef_vi, 59
evq_gap
ef_vi_stats, 74
evq_mask
ef_vi, 59
evq_ptr

 Copyright © Solarflare Communications 2017

ef_eventq_state, 45
evsfl_name
ef_vi_stats_field_layout, 76
evsfl_offset
ef_vi_stats_field_layout, 76
evsfl_size
ef_vi_stats_field_layout, 76
evsl_data_size
ef_vi_stats_layout, 77
evsl_fields
ef_vi_stats_layout, 77
evsl_fields_num
ef_vi_stats_layout, 77

filter_id
ef_filter_cookie, 47
filter_type
ef_filter_cookie, 47
flags
ef_filter_spec, 48

generic
ef_event, 43

ids
ef_vi_rxq, 69
ef_vi_txq, 80
in_jumbo
ef_vi_rxq_state, 71
inited
ef_vi, 60
io
ef_vi, 60
iov_len
ef_iovec, 49

last_desc_i
ef_vi_rxq_state, 71
linked_pio
ef_vi, 60

mask
ef_vi_rxq, 69
ef_vi_transmit_alt_overhead, 78
ef_vi_txq, 80
memreg.h, 138
ef_memreg_alloc, 139
ef_memreg_dma_addr, 139
ef_memreg_free, 140
mr_dma_addrs
ef_memreg, 50
mr_dma_addrs_base
ef_memreg, 50

nic_type
ef_vi, 60

ops
ef_vi, 60

packedstream.h, 141
EF_VI_PS_FLAG_BAD_IP_CSUM, 142
ef_vi_packed_stream_get_params, 142
ef_vi_packed_stream_unbundle, 142
pd.h, 143
ef_pd_alloc, 145
ef_pd_alloc_by_name, 145
ef_pd_alloc_with_vport, 146
ef_pd_flags, 144
ef_pd_free, 147
ef_pd_interface_name, 147
pd_cluster_dh
ef_pd, 55
pd_cluster_name
ef_pd, 55
pd_cluster_sock
ef_pd, 55
pd_cluster_viset_index
ef_pd, 55
pd_cluster_viset_resource_id
ef_pd, 55
pd_flags
ef_pd, 56
pd_intf_name
ef_pd, 56
pd_resource_id
ef_pd, 56
pio.h, 148
ef_pio_free, 149
ef_pio_link_vi, 149
ef_pio_memcpy, 150
ef_pio_unlink_vi, 150
ef_vi_get_pio_size, 151
pio_buffer
ef_pio, 57
pio_io
ef_pio, 57
pio_len
ef_pio, 57
pio_resource_id
ef_pio, 57
post_round
ef_vi_transmit_alt_overhead, 78
posted
ef_vi_rxq_state, 71
pre_round
ef_vi_transmit_alt_overhead, 79
previous
ef_vi_txq_state, 81
ps_cap_len
ef_packed_stream_packet, 51
ps_flags
ef_packed_stream_packet, 51
ps_next_offset

 Copyright © Solarflare Communications 2017

ef_packed_stream_packet, 51
ps_orig_len
ef_packed_stream_packet, 52
ps_pkt_start_offset
ef_packed_stream_packet, 52
ps_ts_nsec
ef_packed_stream_packet, 52
ps_ts_sec
ef_packed_stream_packet, 52
psp_buffer_align
ef_packed_stream_params, 53
psp_buffer_size
ef_packed_stream_params, 53
psp_max_usable_buffers
ef_packed_stream_params, 53
psp_start_offset
ef_packed_stream_params, 54

receive_init
ef_vi::ops, 84
receive_push
ef_vi::ops, 84
removed
ef_vi_rxq_state, 71
ef_vi_txq_state, 81
revision
ef_vi_nic_type, 68
rx
ef_event, 43
rx_buffer_len
ef_vi, 61
rx_discard
ef_event, 43
rx_discard_mask
ef_vi, 61
rx_ev_bad_desc_i
ef_vi_stats, 75
rx_ev_bad_q_label
ef_vi_stats, 75
rx_ev_lost
ef_vi_stats, 75
rx_multi
ef_event, 43
rx_multi_discard
ef_event, 43
rx_no_desc_trunc
ef_event, 43
rx_packed_stream
ef_event, 44
rx_prefix_len
ef_vi, 61
rx_ps_credit_avail
ef_vi_rxq_state, 71
rx_ts_correction
ef_vi, 61
rxq
ef_vi_state, 73

sw
ef_event, 44
sync_flags
ef_eventq_state, 45
sync_timestamp_major
ef_eventq_state, 45
sync_timestamp_minimum
ef_eventq_state, 46
sync_timestamp_minor
ef_eventq_state, 46
sync_timestamp_synchronised
ef_eventq_state, 46

timer.h, 151
ef_eventq_timer_clear, 152
ef_eventq_timer_prime, 152
ef_eventq_timer_run, 153
ef_eventq_timer_zero, 154
timer_quantum_ns
ef_vi, 61
transmit
ef_vi::ops, 84
transmit_alt_discard
ef_vi::ops, 84
transmit_alt_go
ef_vi::ops, 85
transmit_alt_select
ef_vi::ops, 85
transmit_alt_select_default
ef_vi::ops, 85
transmit_alt_stop
ef_vi::ops, 85
transmit_copy_pio
ef_vi::ops, 85
transmit_copy_pio_warm
ef_vi::ops, 86
transmit_pio
ef_vi::ops, 86
transmit_pio_warm
ef_vi::ops, 86
transmit_push
ef_vi::ops, 86
transmitv
ef_vi::ops, 86
transmitv_init
ef_vi::ops, 87
ts_nsec
ef_vi_txq_state, 81
tx
ef_event, 44
tx_alt
ef_event, 44
tx_alt_hw2id
ef_vi, 62
tx_alt_id2hw

Copyright © Solarflare Communications 2017

ef_vi, 62
tx_alt_num
ef_vi, 62
tx_error
ef_event, 44
tx_push_thresh
ef_vi, 62
tx_timestamp
ef_event, 44
tx_ts_correction_ns
ef_vi, 62
txq
ef_vi_state, 74
type
ef_filter_spec, 48

variant
ef_vi_nic_type, 68
vi.h, 155
ef_eventq_put, 158
ef_filter_flags, 158
ef_filter_spec_init, 159
ef_filter_spec_set_block_kernel, 159
ef_filter_spec_set_block_kernel_multicast, 160
ef_filter_spec_set_block_kernel_unicast, 160
ef_filter_spec_set_eth_local, 161
ef_filter_spec_set_eth_type, 161
ef_filter_spec_set_ip4_full, 162
ef_filter_spec_set_ip4_local, 162
ef_filter_spec_set_ip6_full, 163
ef_filter_spec_set_ip6_local, 164
ef_filter_spec_set_ip_proto, 164
ef_filter_spec_set_multicast_all, 166
ef_filter_spec_set_multicast_mismatch, 166
ef_filter_spec_set_port_sniff, 168
ef_filter_spec_set_tx_port_sniff, 168
ef_filter_spec_set_unicast_all, 169
ef_filter_spec_set_unicast_mismatch, 169
ef_filter_spec_set_vlan, 170
ef_vi_alloc_from_pd, 170
ef_vi_alloc_from_set, 172
ef_vi_filter_add, 173
ef_vi_filter_del, 174
ef_vi_flush, 174
ef_vi_free, 175
ef_vi_get_mac, 175
ef_vi_mtu, 176
ef_vi_pace, 176
ef_vi_prime, 177
ef_vi_set_alloc_from_pd, 177
ef_vi_set_filter_add, 178
ef_vi_set_filter_del, 178
ef_vi_set_free, 180
ef_vi_stats_query, 180
ef_vi_stats_query_layout, 181
ef_vi_transmit_alt_alloc, 181
ef_vi_transmit_alt_free, 182
ef_vi_transmit_alt_query_buffering, 182
vi_clustered
ef_vi, 63
vi_flags
ef_vi, 63
vi_i
ef_vi, 63
vi_io_mmap_bytes
ef_vi, 63
vi_io_mmap_ptr
ef_vi, 63
vi_is_normal
ef_vi, 64
vi_is_packed_stream
ef_vi, 64
vi_mem_mmap_bytes
ef_vi, 64
vi_mem_mmap_ptr
ef_vi, 64
vi_out_flags
ef_vi, 64
vi_ps_buf_size
ef_vi, 65
vi_qs
ef_vi, 65
vi_qs_n
ef_vi, 65
vi_resource_id
ef_vi, 65
vi_rxq
ef_vi, 65
vi_stats
ef_vi, 66
vi_txq
ef_vi, 66
vis_pd
ef_vi_set, 72
vis_res_id
ef_vi_set, 72

 Copyright © Solarflare Communications 2017