UDI NIC Test Specification

by

Gian-Carlo Bava (SCO)
Barry Feild (SCO)

Change History

Revision	Date	Description
0.6	11/05/99	DRAFT version for review

Approvers:

TBD

1 Introduction

This document describes the UDI Network Interface Controller (NIC) driver tests. UDI NIC driver tests verify UDI network driver channel operations, parameters, and sequences. Both send and receive are tested, as well as multiple NICs and stress.

This document defines test purposes for the UDI NIC driver test suite. Test purposes specify an operation for the driver to perform and an expected result for the operation. It also defines test cases which are a series of functional tests comprised of one or more test purposes.

1.1 References

This document references and depends upon the following documents

UDI NIC Driver Specification, Revision 1.0

2 Network Driver Test Purpose Definitions

Test purposes are the smallest addressable component used by a test case. Each test purpose specifies an operation for the driver to perform and an expected result for the operation. The test purpose generates a PASS or FAIL result based on the comparison of the actual result of the operation with the expected result. The test specification defines the operations for the driver to perform and the expected results of those operations. It does not describe how to implement the test operations.

Network drivers must implement all of the channel operations defined in the UDI NIC Driver Specification. The table below defines the NIC driver test purposes.

Most values in the Input Parameters/*Notes* column define parameters passed to the test purpose. These modify the test purpose behavior. Notes are in Italics and values that are variable are enclosed in <>.

Values in the Expected Results column are parameters passed by the operation, or members in a control block passed by the operation. Refer to the UDI NIC Driver Specification to differentiate parameters from structure members.

Purpose	Chan	Orig	Operation	Input Parameters/*Notes*	Expected Results
bind	ctrl	NSR	udi_nd_bind_req		
	ctrl	ND	udi_nsr_bind_ack		status=ok/invalid_state
unbind	ctrl	NSR	udi_nd_unbind_req		
	ctrl	ND	udi_nsr_unbind_ack		status=ok/invalid_state
enable	ctrl	NSR	udi_nd_enable_req		
	ctrl	ND	udi_nd_enable_ack		status=ok/hw_problem
disable	ctrl	NSR	udi_nd_disable_req		
addmulti	ctrl	NSR	udi_nd_ctrl_req, command = 1	indicator=<add_num > data=<mac_addresses>	
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood
delmulti	ctrl	NSR	udi_nd_ctrl_req, command = 2	indicator=<del_num> data=<mac_addresses>	
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood
allmultion	ctrl	NSR	udi_nd_ctrl_req, command = 3		
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood

allmultioff	ctrl	NSR	udi_nd_ctrl_req, command = 4	indicator=<add_num> data=<mac_addresses>	
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood
getcurrmac	ctrl	NSR	udi_nd_ctrl_req, command = 5		
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood
setcurrmac	ctrl	NSR	udi_nd_ctrl_req, command = 6	indicator=<addr_size> data=<mac_address>	
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood
getfactmac	ctrl	NSR	udi_nd_ctrl_req, command = 7		
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood indicator=<addr_size> data=<mac_address>
promiscon	ctrl	NSR	udi_nd_ctrl_req, command = 8		
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood
promiscoff	ctrl	NSR	udi_nd_ctrl_req, command = 9		
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood
hwreset	ctrl	NSR	udi_nd_ctrl_req, command = 10		
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood
badrxpkt	ctrl	NSR	udi_nd_ctrl_req, command = 11	indicator=<frame_size>	
	ctrl	ND	udi_nsr_ctrl_ack		status=ok/not_understood
status	ctrl	ND	udi_nsr_status_ind	manually [un]plug NIC *link state initiates ind*	event=down/up/reset
info	ctrl	NSR	udi_nd_info_req	reset_stats=true/false timeout=<seconds> *-1 for no timeout*	
	ctrl	ND	udi_nsr_info_ack		interface_is_active=true/false ***OR*** timeout triggered
tx	tx	ND	udi_nsr_tx_rdy	*tx_buf must be NULL*	
	tx	NSR	udi_nd_tx_req	chain=<num_frames> data=<packet_headers> size=<frame_size> urgent=true/false	
tx-exp	tx	ND	udi_nsr_tx_rdy		
	tx	NSR	udi_nd_exp_tx_req	chain=<num_frames> data=<packet_headers> size=<frame_size> urgent=true/false	
rx	rx	NSR	udi_nd_rx_rdy	chain=<num_frames> data=<packet_headers> size=<frame_size> timeout=<seconds> *-1 for no timeout*	

	rx	ND	udi_nd_rx_ind		timeout triggered ***OR*** rx_status=0 actual_frame_size=size actual_frame_header=data frame_data=<testpattern>
rx-exp	rx	NSR	udi_nd_rx_rdy	chain=<num_frames> data=<packet_headers> size=<expected_size> timeout=<seconds> *-1 for no timeout*	
	rx	ND	udi_nsr_exp_rx_ind	*TBD – can we test this?*	timeout triggered ***OR*** rx_status=0 actual_frame_size=size actual_frame_header=data frame_data=<testpattern>

3 NIC Driver Test Case Definitions

Test cases are a series of functional tests. Functional tests are formed by combining one or more test purposes to test specific functionality. Each functional test generates a PASS or FAIL result. If one or more of the test purposes returns a FAIL result, the functional test returns a FAIL result. If one or more functional tests within a test case returns a FAIL result, the test case returns a FAIL result.

NOTE: Some of the functional tests require at least two machines to interact. These functional tests require the machines to use some type of synchronization and message exchange protocol. Such a protocol is not defined in this document. Driver configuration, initialization, and synchronization need to be described in the UDI NIC Test Case Design document.

3.1 NIC Driver Unit Test Case

The NIC driver unit test case performs functional testing of all channel operations defined in the UDI NIC Driver Specification. This test case verifies basic functionality of the driver under test. This test case also exercises all of the valid state transitions defined in the UDI NIC Driver Specification to verify driver compliance.

This test does not verify driver behavior under heavy load or stress, nor does it attempt to verify driver behavior when presented with invalid state transitions. This test case does not attempt operations with invalid parameters.

In the Functional Test column, state transitions are delineated with arrows (->) between two states. Operations attempted in a given state are marked <state>, <operation>.

Expected results refer to the final test purpose result in the Expected Results column.

NOTE: Two state transitions (enabled->active and active->enabled) occur when the driver detects link activation or loss, respectively. It may be desirable to break this test case into two separate test cases, one that is run with the adapter plugged into the media and another that is run with the adapter unplugged. This would reduce the need for manual intervention, making the test suite as a whole more automated.

Functional Test	Test Purpose Sequence	Input Parameters/*Notes*	Expected Results
unbound->bound	bind		status=ok
bound->unbound	bind, unbind		status=ok
bound->enabled	bind, enable	*adapter unplugged*	status=ok

enabled->bound	bind, enable, disable	*adapter unplugged*	
enabled->unbound	bind, enable, unbind	*adapter unplugged*	status=ok
enabled->active	bind, enable, *plugin NIC*, status	*requires manual intervention*	event=up
active->enabled	bind, enable, *unplug NIC*, status	*requires manual intervention*	event=down
active->unbound	bind, enable, unbind		status=ok
active, transmit	bind, enable, tx	chain=1 size=<min to max pdu> data=<valid_header> urgent=false	*receiver verifies frame header matches expected header (data) and frame contents are test pattern and generates pass/fail*
active, tx-cb_ret	bind, enable, tx	chain=1 size=<min to max pdu> data=<valid_header> urgent=true *NSR verifies tx cbs returned*	*receiver verifies frame header matches expected header (data) and frame contents are test pattern and generates pass/fail*
active, tx-chain	bind, enable, tx	chain=<num_tx_cbs> size=<min to max pdu data=<valid_header> urgent=false	*receiver verifies frame header matches expected header (data) and frame contents are test pattern and generates pass/fail*
active, tx-exp	bind, enable, tx-exp	chain=1 size=<min to max pdu> data=<valid_header> urgent=false	*receiver verifies frame header matches expected header (data) and frame contents are test pattern and generates pass/fail*
active, rx	bind, enable, rx	timeout=<tbd> chain=1 size=<min to max pdu> data=<*both unicast and broadcast MAC addr hdr*>	rx_status=0 actual_frame_size=size actual_frame_header=data frame_data=<testpattern>
active, addmulti	bind, enable, addmulti, rx *(rx to verify receive on each valid multicast addresses)*	indicator=1, 16, 1024 data=<test multicast addrs> timeout=100 chain=1 size=<min pdu> data=<test multicast headers>	status=ok *rx <indicator> frames: verify frame headers match expected headers (data) and frame contents are test pattern*
active, delmulti	bind, enable, addmulti, delmulti, rx *(rx to verify receive timeout on invalid multicast addrs)*	indicator=1, 16, 1024 data=<test multicast addrs> timeout=5 chain=1 size=<min pdu> data=<test multicast headers>	status=ok timeout triggered
active, allmultion	bind, enable, allmultion, rx *(rx to verify receive on multiple multicast addresses)*	 timeout=5 chain=1 size=<min pdu> data=<test multicast headers>	status=ok *verify actual frame headers match expected headers (data) and frame contents are test pattern*

active, allmultioff	bind, enable, allmultion, allmultioff, rx *(rx to verify receive timeout on invalid multicast addrs)*	status=ok indicator=0 timeout=5 chain=1 size=<min pdu> data=<test multicast headers>	status=ok timeout triggered
active, getcurrmac	bind, enable, getcurrmac		status=ok
active, setcurrmac	bind, enable, setcurrmac, rx *(rx to verify receive on current unicast address)* *(getcurrmac verify addr)*	data=<test_unimac_addr>	status=ok *verify receive on test_unimac_addr* data=<test_mac_addr>
active, getfactmac	bind, enable, getfactmac		status=ok data=<valid_mac_addr>
active, promiscon	bind, enable, promiscon, rx *(rx to verify receive on test unicast address)*		status=ok *verify receive on test unicast mac address*
active, promiscoff	bind, enable, promiscon, promiscoff, rx *(rx to verify no receive on test unicast address)*		status=ok *verify receive timeout triggered*
active, hwreset	bind, enable, hwreset		status=ok
active, badrxpkt	bind, enable, badrxpkt	indicator=0, *repeat:* indicator=<max_pdu>	status=ok
bound, info	bind, info	reset_statistics=false timeout=<tbd> *repeat, reset_statistics=true*	interface_is_active=true *verify info returns zero'd stats*
enabled, info	bind, enable, info	reset_statistics=false timeout=<tbd> *adapter unplugged*	interface_is_active=true
active, info	bind, enable, info	reset_statistics=false timeout=<tbd>	interface_is_active=true
unbound, info	info, bind	reset_statistics=false timeout=<tbd_long_time>	interface_is_active=true *verify info_ack returned after bind_ack*
enabled, status	bind, enable *unplug NIC*, status *plugin NIC*, status		 event=down event=up

3.2 NIC Driver Invalid Operations Test Case

The NIC driver invalid operations test case verifies driver behavior when presented with state transitions that are not valid according to the UDI NIC Driver Specification. This test case also verifies driver behavior when operations with invalid parameters are attempted.

In the Functional Test column, state transitions are delineated with arrows (->) between two states. Operations attempted in a given state are marked <state>, <operation>.

The Test Purpose Sequences assume the driver has been initialized by the Management Agent and received a udi_channel_event_ind(UDI_CHANNEL_BOUND) on the NSR's parent channel.

Functional Test	Test Purpose Sequence	Input Parameters/*Notes*	Expected Results
unbound->transmit	tx	chain=1 size=<min to max pdu> data=<valid_header> urgent=false	*rx machine verifies no frames sent on wire*
unbound->receive	rx	timeout=<tbd> chain=1 size=<min to max pdu> data=<valid_header>	timeout triggered *tx machine sends frames to broadcast address*
bound->transmit	bind, tx	chain=1 size=<min to max pdu> data=<valid_header> urgent=false	*rx machine verifies no frames sent on wire*
bound->receive	bind, rx	timeout=<tbd> chain=1 size=<min to max pdu> data=<valid_header>	timeout triggered *tx machine sends frames to broadcast address*
enabled->transmit->bound	bind, enable, tx, disable	*adapter unplugged*	*rx machine verifies no frames sent on wire*
bound, bind	bind, bind		status=invalid_state
enabled, bind	bind, enable, bind	*adapter unplugged*	status=invalid_state
active, bind	bind, enable, bind	status=invalid_state	status=invalid_state
unbound, unbind	unbind		status=invalid_state
unbound, enable	enable		status=hardware_problem
enabled, enable	bind, enable, enable	*adapter unplugged*	status=hardware_problem
active, enable	bind, enable, enable		status=hardware_problem
active, tx	bind, enable, tx	chain=1 size=<max pdu + 1> data=<valid_header> urgent=false	*rx machine verifies no frames sent on wire*
active, tx	bind, enable, tx	chain=1 size=<min pdu - 1> data=<invalid_header> urgent=false	*rx machine verifies no frames sent on wire*
unbound, disable	disable		
bound, disable	bind, disable		
bound, ctrl	bind, getfactmac		status=not_understood
unbound, info	info	reset_stats=false timeout=<tbd>	timeout triggered
active->unbound, info	bind, enable, unbind, info	reset_stats=false timeout=<tbd>	timeout triggered

3.3 NIC Driver Stress Test Case

The NIC driver stress test case verifies driver behavior under heavy load.
This test case measures and reports %media saturation and %cpu utilization.

Operation	Test Purpose Sequence	Input Parameters	Expected Result

transmit	bind, enable, tx (loop)	chain=<all_tx_cbs> size=<min to max pdu> data=<valid_header> urgent=false	*receiver verifies frame headers matchs expected headers (data) and frame contents are test pattern and generates pass/fail*
receive	bind, enable, rx (loop)	timeout=<tbd> chain=1 size=<min to max pdu> *data*=<valid_header>	rx_status=0 actual_frame_size=size actual_frame_header=data frame_data=<testpattern> *(for all frames)*

3.4 NIC Driver Multiple Adapter Test Case

The NIC driver multiple adapter test case verifies the driver is able to utilize more than one adapter for both send and receive. Unlike the stress test case, this test case does not attempt to put the driver under extreme load.

Test	Test Purpose Sequence	Input Parameters/*Notes*	Expected Results
transmit	bind, enable, tx (on all supported adatpers)	chain=<1> size=<min to max pdu> data=<valid_header> urgent=false *TBD - timed test?*	*receiver verifies frame header matches expected header (data) and frame contents are test pattern and generates pass/fail*
receive	bind, enable, rx (on all supported adapters)	timeout=<tbd> chain=1 size=<min to max pdu> *data*=<valid_header> *TBD - timed test?*	rx_status=0 actual_frame_size=size actual_frame_header=data frame_data=<testpattern>