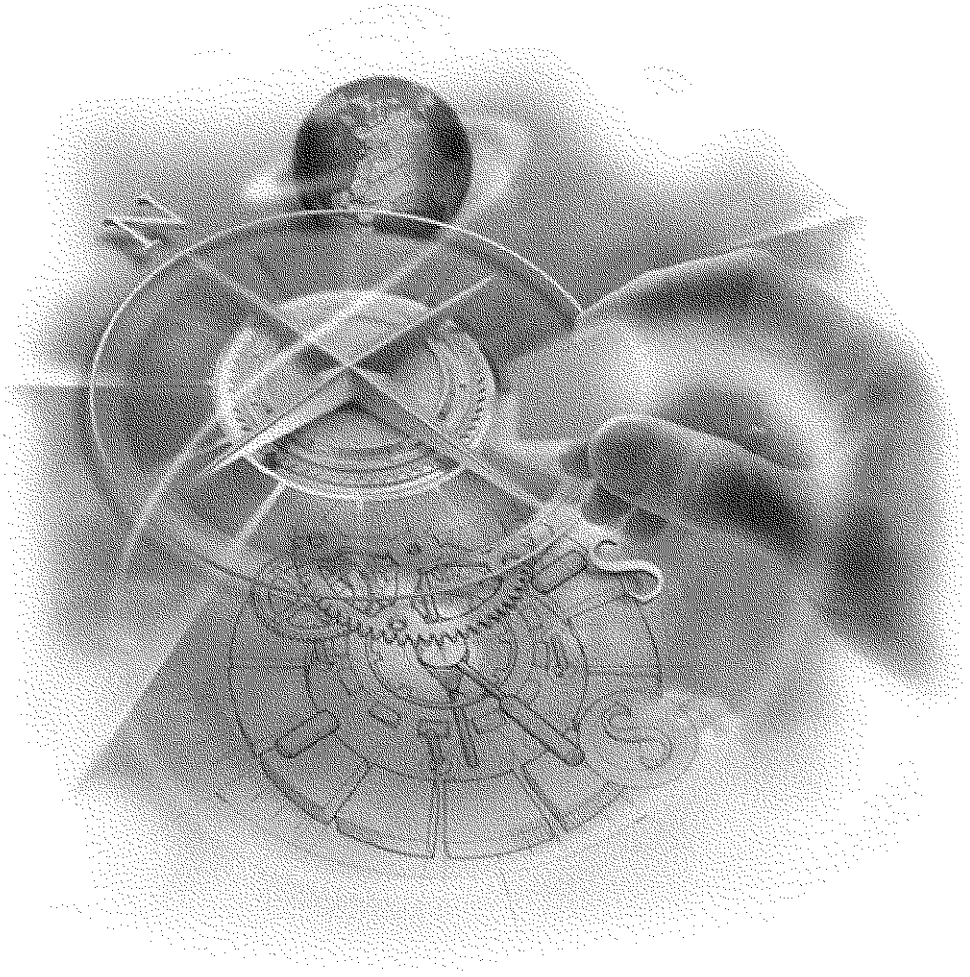
ODI™ SPECIFICATION SUPPLEMENT:

Hardware Checksumming



Novell Developer Kit

Novell®

Legal Notices

Novell, Inc.
122 East 1700 South
Provo, UT 84606
U.S.A.

www.novell.com

Hardware Checksumming
November 1999
104-000199-001

Online Documentation: To access the online documentation for this and other Novell developer products, and to get updates, see developer.novell.com/ndk. To access online documentation for Novell products, see www.novell.com/documentation.

Novell Trademarks

AppNotes is a registered trademark of Novell, Inc.
AppTester is a trademark of Novell, Inc., in the United States.
ArcNet 68 is a trademark of Novell, Inc.
BorderManager is a trademark of Novell, Inc.
C3PO is a trademark of Novell, Inc.
Client 32 is a trademark of Novell, Inc.
ConsoleOne is a trademark of Novell, Inc.
Controlled Access Printer is a trademark of Novell, Inc.
Custom 3rd-Party Object is a trademark of Novell, Inc.
DeveloperNet is a registered trademark of Novell, Inc.
DeveloperNet 2000 is a trademark of Novell, Inc.
GroupWise is a registered trademark of Novell, Inc., in the United States and other countries.
GroupWise 5 is a trademark of Novell, Inc.
Hardware Specific Module is a trademark of Novell, Inc.
HostPublisher is a trademark of Novell, Inc.
Hot Fix is a trademark of Novell, Inc.
HSM is a trademark of Novell, Inc.
InForms is a trademark of Novell, Inc.
Internetwork Packet Exchange is a trademark of Novell, Inc.
IPX is a trademark of Novell, Inc.
IPX/SPX is a trademark of Novell, Inc.
LANalyzer is a registered trademark of Novell, Inc., in the United States and other countries.
Link Support Layer is a trademark of Novell, Inc.
LSL is a trademark of Novell, Inc.
ManageWise is a registered trademark of Novell, Inc., in the United States and other countries.
Mirrored Server Link is a trademark of Novell, Inc.
MLI is a trademark of Novell, Inc.
MLID is a trademark of Novell, Inc.
MSL is a trademark of Novell, Inc.
Multiple Link Interface is a trademark of Novell, Inc.
Multiple Link Interface Driver is a trademark of Novell, Inc.
NControl is a trademark of Novell, Inc.
NCP is a trademark of Novell, Inc.
NDebug is a trademark of Novell, Inc.
NDPS is a registered trademark of Novell, Inc.
NDR is a trademark of Novell, Inc.
NDS is a trademark of Novell, Inc.
NDS Manager is a trademark of Novell, Inc.
NetWare is a registered trademark of Novell, Inc., in the United States and other countries.
NetWare 386 is a trademark of Novell, Inc.
NetWare Aware is a trademark of Novell, Inc.
NetWare Connect is a registered trademark of Novell, Inc, in the United States.
NetWare Core Protocol is a trademark of Novell, Inc.

TTS is a trademark of Novell, Inc.
Universal Component System is a trademark of Novell, Inc.
ViewMAX is a trademark of Novell, Inc.
ZENworks is a trademark of Novell, Inc.

Third-Party Trademarks

All third-party trademarks are the property of their respective owners.
Java is a trademark or registered trademark of Sun Microsystems, Inc., in the United States and other countries.

Contents

1 Introduction

Software checksumming has become so costly in terms of processor usage that it has become necessary to make use of hardware checksumming. This document explains how developers can develop protocols and MLIDs that make use of hardware checksumming that conforms to ODI Specifications: *Hardware Specific Modules (HSMs) (C Language), spec v1.11* and *Protocol Stacks and MLIDS (C Language), spec v1.11*.

Generating a checksum for a transmission packet uses many processor cycles and often flushes the processor cache, which slows down processor time considerably. Validating a checksum for a reception packet can also use many processor cycles and flush the processor cache. Protocols and MLIDs must be developed that rely on the LAN hardware to generate and validate checksums. This will keep processor usage and cache flushes to a minimum.

This document provides the basic information necessary to develop protocols and MLIDs or HSMs that rely on the LAN hardware for checksum generation and validation. Checksums can be generated and validated on hardware that services the following transport layer protocols:

- TCP
- UDP
- ICMP
- RSVP

Checksums can also be generated and validated on hardware that services the following network layer protocols regardless of the transport layer protocol it is servicing:

- IPv4

Other protocol specific checksums may be added in the future, but are not defined in this supplement.

References

The following APIs from ODI Specification:

Protocol Stacks and MLIDS (C Language), spec v1.11, doc v1.20, (May 27, 1997) are referenced in this document.

- CLSL_GetMLIDControlEntry (Index 18 (0x12))
- MLIDManagement (Index14 (0x0E))

The following structures from ODI Specification: *Protocol Stacks and MLIDS (C Language), spec v1.11, doc v1.20, (May 27, 1997)* are referenced in this document:

- ECB Structure
  - ECB.ECB_PreviousLink
  - ECB.ECB_StackID
  - ECB.ECB_ProtocolID
  - ECB.ECB_BoardNumber
- LookAhead
  - LookAhead.LkAhd_PktAttr

The following other items from ODI Specification: *Protocol Stacks and MLIDS (C Language), spec v1.11, doc v1.20, (May 27, 1997)* are referenced in this document:

- How to access MLID Control API (Chapter 18 , "MLID Control Routines").
- The MLID Management IOCTL (Chapter 18 , "MLID Control Routines").

- Protocol Stack Initialization (Chapter 4, "Protocol Stack Initialization").

2 Novell's MLID Capability Management ECB

Potocol stacks must call the MLID Management API containing the MLID Capability Management ECB.

The MLID Capability Management ECB structure is defined as follows:

```
typedef  struct _MAN_CAP_ECB_
{
    struct _MAN_CAP_ECB_      *MCECB_NextLink;
    struct _MAN_CAP_ECB_      *MCECB_PreviousLink;
    UINT16          MCECB_Status;
    void            (*MCECB_ESR) (struct _MAN_CAP_ECB_ *);
    UINT16          MCECB_STACKID;
    UINT8           MCECB_IDENT[6];
    PROT_ID         MCECB_Protocol_ID;
    UINT32          MCECB_Control;
    UINT32          MCECB_BoardNumber;
    UINT32          MCECB_HSMCapabilities;
    UINT32          MCECB_HSMCapabilitiesState;
    UINT32          MCECB_ProtWorkspace;
    UINT32          MCECB_DriverWorkspace;
    UINT32          MCECB_Reserved[2];
} MANAGEMENT_CAPABILITY_ECB;
```

MLID Capability Management ECB Structure Field Definitions

The fields for the MLID Capability Management ECB structure are filled in and defined as follows:

Table 1 MLID Capability Management ECB Structure Field Definitions

*MCECB_NextLink	*On Entry:* Initialized to 0 *On Exit:* Reserved, currently set to 0
*MCECB_PreviousLink	*On Entry:* Initialized to 0 *On Exit:* Reserved, currently set to 0
MCECB_Status	*On Entry:* Initialized to 0 *On Exit:* Completion Code: ◆ ODISTAT_SUCCESSFUL - All the control functions were successful. ◆ ODISTAT_ITEM_NOT_PRESENT - One or more of the requested capabilities are not supported by the MLID. The requested capabilities that are supported were activated and the currently active capabilities, indicated in the *MCECB_HSMCapabilitiesState* field, were set on return. ◆ ODISTAT_BAD_PARAMETER - None of the requested capabilities are supported by the MLID.
(*MCECB_ESR) (struct _MAN_CAP_ECB_ *)	*On Entry:* Pointer to the Event Service Routine (ESR) to call after the requested capabilities have been activated. This field may be set to NULL if an ESR is not needed. *On Exit:* Unchanged. (See section 2.3 - Management Capabilities Completion ESR).
MCECB_STACKID	*On Entry:* The protocol stack ID assigned by the LSL. *On Exit:* Unchanged.
MCECB_IDENT[6]	*On Entry:* MANCAP' *On Exit:* Unchanged.

MCECB_Protocol_ID	*On Entry:* The network protocol ID value that identifies the network protocol that is sending or receiving the packets. The MLID uses the network protocol ID and the board number to determine which checksum to generate or validate. The network protocol ID and the board number are specified in the transmission ECB (*ECB_StackId* field) and the reception ECB (*ECB_PreviousLink* field). *On Exit:* Unchanged.
MCECB_BoardNumber	*On Entry:* The board number to activate the capabilities on. *On Exit:* Unchanged.
MCECB_Control	*On Entry:* One of the following control functions: MCECB_CON_GET_CAPABILITIES 0 MCECB_CON_ENABLE_ACTIVE_CAP 1 MCECB_CON_DISABLE_ACTIVE_CAP 2 MCECB_CON_DISABLE_REMOVE_CAP 3 *On Exit:* Unchanged.
MCECB_HSMCapabilities	*On Entry:* Initialized to 0. *On Exit:* The bits set to indicate which capabilities are possible for the HSM. (See section 2.2 - Capability Bits Defined for the MLID Capability Management ECB.)
MCECB_HSMCapabilitiesState	*On Entry:* The bits set to indicate which capabilities the *MCECB_Control* field will act on for the specified network protocol ID and board number combination. This field is ignored on entry if the *MCECB_CON_GET_CAPABILITIES* control function is used. *On Exit:* The bits set to indicate which capabilities are active for the specified protocol ID and board number combination. (See section 2.2 - Capability Bits Defined for the MLID Capability Management ECB.)
MCECB_ProtWorkspace	*On Entry:* Protocol specific values. *On Exit:* Unchanged.

MCECB_DriverWorkspace	*On Entry:* Initialized to 0. *On Exit:* Driver sets to 0.
MCECB_Reserved	*On Entry:* Initialized to 0 *On Exit:* Reserved, currently set to 0

Capability Bits Defined for the MLID Capability Management ECB

Capability bits are to be set in the *MCECB_HSMCapabilities* and *MCECB_HSMCapabilitiesState* fields of the MLID Capability Management ECB.

The following bit values indicate which checksums can be generated for transmissions:

```
HSMCAP_IPv4_CHECKSUM_TX    0x0001
HSMCAP_TCP_CHECKSUM_TX     0x0002
HSMCAP_UDP_CHECKSUM_TX     0x0004
HSMCAP_RSVP_CHECKSUM_TX    0x0008
HSMCAP_ICMP_CHECKSUM_TX    0x0010
```

The following bit values indicate which checksums can be validated for receptions:

```
HSMCAP_IPv4_CHECKSUM_RX    0x0100
HSMCAP_TCP_CHECKSUM_RX     0x0200
HSMCAP_UDP_CHECKSUM_RX     0x0400
HSMCAP_RSVP_CHECKSUM_RX    0x0800
HSMCAP_ICMP_CHECKSUM_RX    0x1000
```

All other bits are reserved and set to 0.

NOTE: Hardware checksums can only be generated for *transport* layer protocols (TCP, UDP, ICMP, and RSVP). If the *network* layer protocol (IPv4) needs to fragment a packet, then the *transport* layer protocol must handle the checksuming generation and validation for the network layer protocol.

Management Capabilities Completion ESR

*void (*MCECB_ESR) (struct _MAN_CAP_ECB_ *);*

On Entry:

- *ecb* - Pointer to the MLID Capability Management ECB being completed.

On Exit:

- Unchanged.

Remarks:

If the requested function can be completed synchronously, the MLID will process the requested function to completion without calling the MCECB_ESR.

If a delay must occur to act on the MLID Capability Management ECB, the MLID Management API returns ODISTAT_RESPONSE_DELAYED, and the MLID calls the MCECB_ESR after processing the requested function to completion. The MCECB_ESR may be called before the MLID Management API has completed.

3

The Protocol's View of Checksumming

The protocol must be aware of whether the MLID is capable of generating or validating checksums for the protocol. If the MLID is capable of generating or validating checksums for the protocol, then the protocol must call the MLID Management API to inform the MLID that it must generate or validate the checksum of the subsequent packet. The protocol indicates this to the MLID in the ECB of the packet it is sending.

The Protocol must disable all capabilities that it enabled prior to shutting down. The protocol must set the MCECB_CON_DISABLE_REMOVE_CAP value in the *MCECB_Control* field and send the MLID Capability Management ECB to the MLID prior to shutting down the protocol.

The protocol stack must configure the C HSM to generate checksums as follows:

1. Determine the MLID's control Entry point.

2. Determine if the MLID supports checksum generation by sending a MLID Capability Management ECB.

3. If the MLID supports checksum generation, send another MLID Capability Management ECB indicating which checksums to generate.

4. Flag each ECB the MLID must generate a checksum for in the *StackId* field of a transmission ECB.

The protocol stack must configure the C HSM to validate checksums as follows:

1. Determine the MLID's control Entry point.

2. Determine if the MLID supports checksum validation by sending a MLID Capability Management ECB.

3. If the MLID supports checksum validation, send another MLID Capability Management ECB indicating which checksums to validate.

4. The C HSM will flag each ECB it attempts to validate in the *ECB_PreviousLink* field of the receive ECB and in the *LKAhd_PktAttr* field of the LookAhead structure associated with the received packet.

5. The C HSM will flag each ECB it fails to validate in the *ECB_PreviousLink* field of the receive ECB and in the *LKAhd_PktAttr* field of the LookAhead structure associated with the received packet.

Generating an MLID Capability Management ECB

The protocol creates an MLID Capability Management ECB to get a list of the capabilities of the MLID. The protocol sets the appropriate value, MCECB_CON_GET_CAPABILITIES, in the *MCECB_Control* field. The MLID then returns the active capabilities in the *MCECB_HSMCapabilitiesState* field. The protocol activates capabilities by setting MCECB_CON_ENABLE_ACTIVE_CAP in the *MCECB_Control* field, setting the appropriate bits in the *MCECB_HSMCapabilitiesState* field, and sending the MLID Capability Management ECB to the MLID. The MLID acts on the MLID Capability Management ECB and returns the active capabilities in the *MCECB_HSMCapabilitiesState* field.

Packet Reception

Hardware checksum validation is controlled by the C HSM via the *MCECB_HSMCapabilities* field of the MLID Capability Management ECB. The *ECB_PreviousLink* field of the receive ECB will indicate when the checksum was validated and whether or not the packet failed validation. This information will also be contained in the *LKAhd_PktAttr* field of the LookAhead structure associated with the incoming packet.

Packet Transmission

Protocol stacks are assigned protocol stack IDs by the LSL when they are registered. Typically, protocol stacks put their protocol stack IDs in the *ECB_StackID* field of the transmit packet ECB. A set of values that indicate such things as whether the packet is a raw send, a priority transmit packet, or whether it requires checksum generation can also be placed in the *ECB_StackID* field.

The *ECB_StackID* field values are as follows:

Table 2 ECB_StackID Field Values

0x0000-0x00FF (0-255)	Protocol Identification Number assigned by the LSL
0xFFFF	Raw Send
0xFFF0-0xFFFF	Priority Sends

0xzzFy	Check Sum Packets

zz is the checksum to generate. The bit pattern for *zz* is *10xx xxxx*.

0x80 = No Checksum generation on this packet.

0x90 = Generate Transport Layer Checksum (TCP, UDP, ICMP, RSVP)

0xA0 = Generate Network Layer Checksum (Ipv4)

Fy is the priority transmission support level.

[See "Priority Tx Support" in ODI Specification: *Hardware Specific Modules (HSMs) (C Language)* and the *ECB_StackID* definition in ODI Specification: *Protocol Stacks and MLIDS (C Language)*]

The *Fy* values are the same as the last byte of the priority send values. These values decode as follows:

(Raw Sends)

0xFF = CHK_RAW_SEND_PRIORITY_0 No Priority

0xFE = CHK_RAW_SEND_PRIORITY_1 Lowest Priority

0xFD = CHK_RAW_SEND_PRIORITY_2

0xFC = CHK_RAW_SEND_PRIORITY_3

0xFB = CHK_RAW_SEND_PRIORITY_4

0xFA = CHK_RAW_SEND_PRIORITY_5

0xF9 = CHK_RAW_SEND_PRIORITY_6

0xF8 = CHK_RAW_SEND_PRIORITY_7 Highest Priority

(Normal Sends)

0xF7 = CHK_SEND_PRIORITY_0 No Priority

0xF6 = CHK_SEND_PRIORITY_1 Lowest Priority

0xF5 = CHK_SEND_PRIORITY_2

0xF4 = CHK_SEND_PRIORITY_3

0xF3 = CHK_SEND_PRIORITY_4

0xF2 = CHK_SEND_PRIORITY_5

0xF1 = CHK_SEND_PRIORITY_6

0xF0 = CHK_SEND_PRIORITY_7 Highest Priority

Raw send packets may be checksummed. The raw send designation indicates that the protocol stack has generated the full MAC header. The MLID will not change the MAC header on a raw send.

All other bits are reserved and reset to zero.

Which checksum to generate is determined by the *ECB_StackID* and *ECB_ProtocolID* fields of the ECB. This combination is matched with the information obtained by the MLID Management API call.

For example:

The protocol ID for the IPv4 checksum for an Ethernet_II frame type is 800. When the HSM generates an IPv4 checksum for Ethernet_II and you have also indicated no priority, normal send, checksum IP, and TCP for the packet, the *ECB_ProtocolID* field will contain 800 and the *StackID* field will contain 0xB0F7 .

4 The MLID's View of Checksumming

Generally, MLIDs that support checksumming are comprised of the Media Specific Module (MSM) and the Topology Specific Module (TSM) provided by the Novell *LAN Driver ToolKit*, and a Hardware Specific Module (HSM) written to ODI Specifications: *Hardware Specicfic Modules (HSMs) spec v1.11+* and *Protocol Stacks and MLIDs, spec v1.11+*.

Typically, C HSMs are not ECB-aware and are not aware of network layer protocols or MAC layer protocols; however, C HSMs that support checksumming should generally be ECB-aware.

All C HSMs (ECB-aware or not) report their checksumming capabilities via the MLID Management API when queried.

Upon Receipt of the MLID Capability Management ECB

The C HSM receives the MLID Capability Management ECB via the Driver Management API. The C HSM acts on the bits in the *MCECB_HSMCapabilitiesState* field of the MLID Capability Management ECB as directed by the *MCECB_Control* field.

To enable any capabilities, the *MCECB_Control* field value must be set to MCECB_CON_ENABLE_ACTIVE_CAP. The C HSM will then enable any capabilities indicated by the bits in the *MCECB_HSMCapabilitiesState* field.

To disable any capabilities, the *MCECB_Control* field value must be set to MCECB_CON_DISABLE_ACTIVE_CAP. The C HSM will then disable any capabilities indicated by the bits in the *MCECB_HSMCapabilitiesState* field.

Upon completion of any MLID Capability Management ECB request, the CHSM always sets the *MCECB_HSMCapabilities* field and updates the *MCECB_HSMCapabilitiesState* field to reflect which checksums are currently enabled.

The status of any bits that were not set in the *MCECB_HSMCapabilitiesState* field is unchanged.

ECB-Aware C HSMs

Packet Reception

When a packet is received, and the checksum capabilities for the board number / protocol ID combination are active, the ECB-aware C HSM will provide checksum validation.

The ECB-aware C HSM will indicate whether a packet's checksum has been validated and whether the checksum was valid or invalid in the *ECB_PreviousLink* field of the received packet's ECB. The TSM will also indicate this information in the *LkAhd_PktAttr* field of the LookAhead structure associated with the packet.

NOTE: ECB-aware C HSMs do not fill in or prepare LookAhead structures. TSMs do this.

Table 3 ECB_PreviousLink Field Definitions - Currently Defined Bits

PAE_CRC_BIT
PAE_CRC_ALIGN_BIT
PAE_RUNT_PACKET_BIT
PAE_TOO_BIG_BIT
PAE_NOT_ENABLED_BIT
PAE_MALFORMED_BIT
PAE_NO_COMPRES_BIT
PAE_NONCAN_ADDR_BIT

Table 4 ECB_PreviousLink Field Definitions - New Defined Bits

PAE_TRANS_PROT_CHKSUM_ERR	Set if the transport layer protocol checksum failed validation: TCP, UDP, RSVP, ICMP.
PAE_NET_PROT_CHKSUM_ERR	Set if the network layer protocol checksum failed validation: IPv4.
PAE_TRANS_PROT_CHKSUM	Set if the transport layer protocol checksum validation was performed: TCP, UDP, RSVP, ICMP.
PAE_NET_PROT_CHKSUM	Set if the network layer protocol checksum validation was performed: Ipv4.

Packet Transmission

ECB-aware C HSMs generate checksums on all transmit packets that have the appropriate bit(s) set in the *ECB_StackID* field of the packet's ECB. C HSMs must be configured according to the information in an MLID Capability Management ECB before generating any checksums. The *StackID* field provides the ECB-aware C HSM with the information it needs to determine when to generate a checksum (See section 3 - The Protocol's View of Checksumming). The C HSM uses the board number / protocol ID combination to determine which checksum to generate.

ECB-aware C HSMs will not transmit packets with checksum generation codes that are not supported by the MLID. The C HSM will return the TCB to the protocol stack using *<CTSM >SendComplete()* with the *transmitStatus* parameter set to ODISTAT_PACKET_UNDELIVERABLE. The CTSM will set the *ECB_Status* field to ODISTAT_CANCELED in this case.

ECB-aware C HSMs track the capabilities of specific logical boards and only generate checksums when the checksum capabilities for a specific board number / protocol ID combination are activated.

TCB/RCB C HSMs

Packet Reception

When a packet is received, and the checksum capabilities for the board number / protocol ID combination are active, the C HSM will provide checksum validation.

The C HSM will indicate whether a packet's checksum has been validated and whether the checksum was valid or invalid in the *rcvStatus* parameter of *GetRCB* or *ProcessGetRCB*. The TSM will also indicate this information in the *LkAhd_PktAttr* field of the LookAhead structure associated with the packet and in the *ECB_PreviousLink* field of the received packet's ECB. See "ECB_PreviousLink Field Definitions - Currently Defined Bits".

NOTE: C HSMs do not fill in or prepare LookAhead structures. TSMs do this.

NOTE: On 3.12 and 4.11 servers, the LSL overwrites the *ECB.ECB_PreviousLink* field. Checksum validation is only valid if the LSL Configuration Table is x.xx or higher.

Packet Transmission

C HSMs generate checksums on all transmit packets that have the appropriate bit(s) set in the *ECB_StackID* field of the packet's ECB. The *ECB_StackID* field of the packet's ECB is accessable via the TCB_STACKID macro.

C HSMs must be configured according to the information in an MLID Capability Management ECB before generating any checksums. The *StackID* field provides the C HSM with the information it needs to determine when to generate a checksum (See section 3 - The Protocol's View of Checksumming). The C HSM uses the board number / protocol ID combination to determine which checksum to generate.

C HSMs will not transmit packets with checksum generation codes that are not supported by the MLID. The C HSM will return the ECB to the protocol stack using <CTSM >SendCompelete() with the *ECB_Status* field set to ODISTAT_BAD_PARAMETER and the *transmitStatus* parameter set to ODISTAT_PACKET_UNDELIVERABLE.

C HSMs track the capabilities of specific logical boards and only generate checksums when the checksum capabilities for a specific board number / protocol ID combination are activated.

5 Hwchksum.h File

```
/*---------------------------*
 *  Copyright Unpublished Work of Novell, Inc. All Rights Reserved
 *
 *  THIS WORK IS AN UNPUBLISHED WORK AND CONTAINS CONFIDENTIAL,
 *  PROPRIETARY AND TRADE SECRET INFORMATION OF NOVELL, INC.
 *  ACCESS TO THIS WORK IS RESTRICTED TO (I) NOVELL EMPLOYEES
 *  WHO HAVE A NEED TO KNOW TO PERFORM TASKS WITHIN THE SCOPE
 *  OF THEIR ASSIGNMENTS AND (ii) ENTITIES OTHER THAN NOVELL
 *  WHO HAVE ENTERED INTO APPROPRIATE LICENSE AGREEMENTS.
 *  NO PART OF THIS WORK MAY BE USED, PRACTICED, PERFORMED, COPIED,
 *  DISTRIBUTED, REVISED, MODIFIED, TRANSLATED, ABRIDGED,
 *  CONDENSED, EXPANDED, COLLECTED, COMPILED, LINKED, RECAST,
 *  TRANSFORMED OR ADAPTED WITHOUT THE PRIOR WRITTEN CONSENT OF
 *  NOVELL.  ANY USE OR EXPLOITATION OF THIS WORK WITHOUT AUTHORIZATION
 *  COULD SUBJECT THE PERPETRATOR TO CRIMINAL AND CIVIL LIABILITY.
 *---------------------------*
/*****************************************************************************
 *
 * Program Name:C ODI Hardware Checksumming Supplement Header File
 *
 * Filename:HWChkSum.H
 *
 * ODI Spec Ver:1.11
 *
 * Description:This file is the main source for the
 *                      C ODI SPECIFICATION: Hardware Checksumming
 *                      Supplement.
 * Structures needed by the MLI or MPI interface for
 *                      Hardware Checksumming are defined here.
 *
 * Modification History:
 * 971003       JWR   Added MCECB_CON_DISABLE_REMOVE_CAP value for the
 *                    MCECB_Control Field.
 *
```

```
 * 981001       LON   Added TCB_STATUS macro for use by TCB aware HSMs.
 *
 * 981105       LON   These defines had a typo:
 *                    PAE_TRANS_PROT_CHKSUM_ERR, PAE_NET_PROT_CHKSUM_ERR,
 *                    PAE_TRANS_PROT_CHKSUM and PAE_NET_PROT_CHKSUM,
 *                    - SPD 216043
 *
 ************************************************************************/

#ifndef         _ODI_HWChkSum_Include_
#define         _ODI_HWChkSum_Include_

/* C ODI Hardware Checksumming Specification Version Numbers */

#define ODI_HWChkSum_VER        02

/* Novell's MLID Capability Management ECB Defined */

typedef  struct _MAN_CAP_ECB_
      {
        struct _MAN_CAP_ECB_    *MCECB_NextLink;
        struct _MAN_CAP_ECB_    *MCECB_PreviousLink;
        UINT16                  MCECB_Status;
        void                    (*MCECB_ESR) (struct _MAN_CAP_ECB_ *);
        UINT16                  MCECB_STACKID;
        UINT8                   MCECB_IDENT[6];
        PROT_ID                 MCECB_Protocol_ID;
        UINT32                  MCECB_Control;
        UINT32                  MCECB_BoardNumber;
        UINT32                  MCECB_HSMCapabilities;
        UINT32                  MCECB_HSMCapabilitiesState;
        UINT32                  MCECB_ProtocolWorkSpace;
        UINT32                  MCECB_DriverWorkSpace;
        UINT32                  MCECB_Reserved[2];
      } MANAGEMENT_CAPABILITY_ECB;

/* MECB_IDENT Value Defined */
  #define       MCECB_IDENT_STR { 'M','A','N','C','A','P'} /* 'MANCAP' */

/* MECB_Control Values Defined */
  #define       MCECB_CON_GET_CAPABILITIES      0
  #define       MCECB_CON_ENABLE_ACTIVE_CAP     1
  #define       MCECB_CON_DISABLE_ACTIVE_CAP    2
  #define       MCECB_CON_DISABLE_REMOVE_CAP    3


/* MECB_HSMCapabilities defined. */
  #define       HSMCAP_IPv4_CHECKSUM_TX     0x0001
```

```
  #define       HSMCAP_TCP_CHECKSUM_TX      0x0002
  #define       HSMCAP_UDP_CHECKSUM_TX      0x0004
  #define       HSMCAP_RSVP_CHECKSUM_TX     0x0008
  #define       HSMCAP_ICMP_CHECKSUM_TX     0x0010
            /* Reserved 0x0020 */
            /* Reserved 0x0040 */
            /* Reserved 0x0080 */

  #define  HSMCAP_IPv4_CHECKSUM_RX          0x0100
  #define  HSMCAP_TCP_CHECKSUM_RX           0x0200
  #define  HSMCAP_UDP_CHECKSUM_RX           0x0400
  #define  HSMCAP_RSVP_CHECKSUM_RX          0x0800
  #define  HSMCAP_ICMP_CHECKSUM_RX          0x1000
            /* Reserved 0x2000 */
            /* Reserved 0x4000 */
            /* Reserved 0x8000 */

/* ECB_StackID and TCB_StackID values defined  0xzzFy */
/* (RAW Sends) */
  #define       CHK_RAW_SEND_PRIORITY_0     0x80FF /* No Priority */
  #define       CHK_RAW_SEND_PRIORITY_1     0x80FE /* Lowest Priority */
  #define       CHK_RAW_SEND_PRIORITY_2     0x80FD
  #define       CHK_RAW_SEND_PRIORITY_3     0x80FC
  #define       CHK_RAW_SEND_PRIORITY_4     0x80FB
  #define       CHK_RAW_SEND_PRIORITY_5     0x80FA
  #define       CHK_RAW_SEND_PRIORITY_6     0x80F9
  #define       CHK_RAW_SEND_PRIORITY_7     0x80F8 /* Highest Priority */

/* (Normal Sends) */
  #define       CHK_SEND_PRIORITY_0         0x80F7 /* No Priority */
  #define       CHK_SEND_PRIORITY_1         0x80F6 /* Lowest Priority */
  #define       CHK_SEND_PRIORITY_2         0x80F5
  #define       CHK_SEND_PRIORITY_3         0x80F4
  #define       CHK_SEND_PRIORITY_4         0x80F3
  #define       CHK_SEND_PRIORITY_5         0x80F2
  #define       CHK_SEND_PRIORITY_6         0x80F1
  #define       CHK_SEND_PRIORITY_7         0x80F0 /* Highest Priority */


/* zz bits defined for generation. */
  #define       CHK_Generate_Transport      0x1000
  #define       CHK_Generate_Network        0x2000

/* PreviousLink ECB Field definitions (and RcvStatus parameter) */
  #define       PAE_TRANS_PROT_CHKSUM_ERR   0x00001000
  #define       PAE_NET_PROT_CHKSUM_ERR     0x00002000
  #define       PAE_TRANS_PROT_CHKSUM       0x00010000
  #define       PAE_NET_PROT_CHKSUM         0x00020000
```

```
/* Access MACROS for the TCB aware Drivers to get ECB information */
  #define TCB_STACKID(t) \
          (UINT16 *) ((UINT8*)(t->TCB_FragBlockPtr)-\
                    (((UINT8) &(((ECB*)0)->ECB_FragmentCount)) - \
                     ((UINT8) &(((ECB*)0)->ECB_StackID))))

  #define TCB_PROTOCOLID(t)\
          (PROT_ID *)((UINT8*)(t->TCB_FragBlockPtr)-\
                   (((UINT8) &(((ECB*)0)->ECB_FragmentCount)) - \
                     ((UINT8) &(((ECB*)0)->ECB_ProtocolID))))

  #define TCB_BOARDNUMBER(t)\
          (UINT32 *)((UINT8*)(t->TCB_FragBlockPtr)-\
                   (((UINT8) &(((ECB*)0)->ECB_FragmentCount)) - \
                     ((UINT8) &(((ECB*)0)->ECB_BoardNumber))))

  #define TCB_STATUS(t)\
          (UINT16 *)((UINT8*)(t->TCB_FragBlockPtr)-\
                   (((UINT8) &(((ECB*)0)->ECB_FragmentCount)) - \
                     ((UINT8) &(((ECB*)0)->ECB_Status))))

#endif  /*_ODI_HWChkSum_Include_ */
```