4

MSM/TSM Data Structures and Variables

Introduction 4-3

MSM Equates 4-4
MSMVirtualBoardLink 4-4
MSMStatusFlags 4-5
MSMTxFreeCount 4-6
MSMMaxFrameHeaderSize 4-7
MSMPhysNodeAddress 4-8

Public Variables 4-9
LogicalToPhysical 4-9
PhysicalToLogical 4-9

Data Structures 4-10
Receive Control Blocks 4-11
Fragmented RCB 4-12
Non-Fragmented RCB 4-14
Transmit Control Blocks 4-16
TCB for Ethernet, Token-Ring, PCN2, and FDDI 4-17
TCB for RX-Net 4-18
Fragment Structure 4-21
Event Control Blocks 4-22
Receive ECBs vs RCBs 4-24
Transmit ECBs vs TCBs 4-25
ECB Field Descriptions 4-26

Introduction

This chapter describes the data structures, variables, and constants defined by the MSM and TSM. Some of the variables and structures in this chapter are required to control processes and must be initialized, updated, or managed by the driver. Others are made available as optional support for the developer and may be used accordingly.

MSM Equates

- *MSMVirtualBoardLink*
- *MSMStatusFlags*
- *MSMTxFreeCount*
- *MSMMaxFrameHeaderSize*
- *MSMPhysNodeAddress*

Public Variables

- *LogicalToPhysical*
- *PhysicalToLogical*

Data Structures

- *Receive Control Blocks (RCBs)*
- *Transmit Control Blocks (TCBs)*
- *Event Control Blocks (ECBs)*

MSM Equates

The HSM must access several variables located in the MSM's Data Space. This section describes the MSM defined equates which enable the HSM to access these variables. The equates represent negative offsets which are used in conjunction with EBP, the pointer to the Adapter Data Space.

MSMVirtualBoardLink

The MSM maintains a separate configuration table for each frame type supported by the driver. *MSMVirtualBoardLink* is used to access a list of pointers to the configuration tables.

The list contains 4 pointers for Ethernet, 2 for Token-Ring, PCNII, and FDDI, and 1 for RX-Net. If a particular frame has not been loaded, the pointer to the corresponding configuration table will be zero. The lists are accessed as follows.

Ethernet
```
[ebp].MSMVirtualBoardLink + 00h    ;ETHERNET 802.3
[ebp].MSMVirtualBoardLink + 04h    ;ETHERNET II
[ebp].MSMVirtualBoardLink + 08h    ;ETHERNET 802.2
[ebp].MSMVirtualBoardLink + 0Ch    ;ETHERNET SNAP
```

Token-Ring
```
[ebp].MSMVirtualBoardLink + 00h    ;TOKEN 802.2
[ebp].MSMVirtualBoardLink + 04h    ;TOKEN SNAP
```

PCN2
```
[ebp].MSMVirtualBoardLink + 00h    ;PCN2 802.2
[ebp].MSMVirtualBoardLink + 04h    ;PCN2 SNAP
```

FDDI
```
[ebp].MSMVirtualBoardLink + 00h    ;FDDI 802.2
[ebp].MSMVirtualBoardLink + 04h    ;FDDI SNAP
```

RX-Net
```
[ebp].MSMVirtualBoardLink + 00h    ;RX-Net
```

Example
```
mov   ebx, [ebp].MSMVirtualBoardLink+08h  ; Ptr to E_802.2 config table
or    ebx, ebx                            ; Check if valid pointer?
jz    Frame8022NotRegistered              ; Jump if not
mov   eax, [ebx].MLIDSlot                 ; EAX = Our slot number
```

MSMStatusFlags

The MSM maintains a dword variable which provides certain adapter status information. This status information enables the driver to determine if the adapter is shutdown or if the MSM has any packets waiting in its transmit queue. The *MSMStatusFlags* equate represents a negative offset which is used in conjunction with EBP, the pointer to the Adapter Data Space, to access the status variable. It is defined in the MSM.INC file as follows:

```
MSMStatusFlags equ   DriverAdapterStart -(2*4)

SHUTDOWN       equ   01h   ; Bit #0 = Shutdown Status
TXQUEUED       equ   02h   ; Bit #1 = Tx Queue Status
```

MSMStatusFlags can be used by the HSM to determine whether the adapter is partially shutdown. If bit #0 is set, the adapter is partially shutdown and should not be serviced. Likewise, the MSM will not call *DriverSend* to transmit a packet if the adapter is partially shutdown.

```
test     [ebp].MSMStatusFlags,SHUTDOWN
jnz      DoNotServiceAdapter
```

The status flags can also be used to determine if the TSM has any send TCBs queued, thus saving a call to *<TSM>GetNextSend.* If bit #1 is set, the TSM has at least one packet queued for transmission.

Note: RX-Net drivers cannot use this test since additional fragments of a split packet are not detected.

```
test     [ebp].MSMStatusFlags,TXQUEUED
jz       NoSendsQueued
```

Example

```
DriverISR   proc
     ⋮
TransmitComplete:                     ; EBP = Ptr to Adapter Data Space

   inc   [ebp].MSMTxFreeCount          ; Free adapter’s transmit resource
   mov   [ebp].TxInProgress, 0         ; Clear transmit in progress flag

   ;*** Transmit Next Packet ***

   test  [ebp].MSMStatusFlags,TXQUEUED ; Anything in send queue?
   jz    NoSendsQueued                 ; Jump if nothing to send
   call  <TSM>GetNextSend              ; Otherwise get the next TCB from
   call  DriverSend                    ;   the queue and send it
     ⋮
   MSMServiceEventsAndReturn

DriverISR   endp
```

MSMTxFreeCount

During initialization, the HSM must specify the number of hardware resources available on the adapter for handling pending packet transmissions. The MSM uses this value to determine if the adapter is ready to accept another packet for transmission. The count is also used to determine how many TCB structures the MSM will allocate. The *MSMTxFreeCount* equate represents a negative offset which is used in conjunction with EBP, the pointer to the Adapter Data Space, to access the count. It is defined in the MSM.INC file as follows:

```
MSMTxFreeCount    equ    DriverAdapterStart -(1*4)
```

For example, if the adapter has a second transmit buffer that can accept another packet before the current transmission is complete, the driver should set *MSMTxFreeCount* to a value of 2. Some adapters support hardware queuing. In this case, the count should represent the number of transmissions that the adapter can efficiently process. If the adapter has no additional resources available other than those used to transmit the current packet, set *MSMTxFreeCount* to 1.

The TSM decrements this count before it calls *DriverSend*. The count is also decremented during a successful call to *<TSM>GetNextSend*. The TSM assumes that the adapter is not ready for another packet if this count reaches zero.

The driver is responsible for incrementing the count each time one of the adapter's transmit resources becomes available. The count must be incremented not only when the adapter successfully completes a transmission, but also when a transmission is aborted due to timeout errors, maximum retry errors, ...etc.

Example

```
DriverInit  proc                        ; EBP = Ptr to Adapter Data Space
     ⋮
   mov   [ebp].MSMTxFreeCount,2         ; Adapter has 2 transmit resources
     ⋮
DriverInit  endp



DriverISR   proc
     ⋮
TransmitComplete:
   inc   [ebp].MSMTxFreeCount           ; Free adapter's transmit resource
     ⋮
DriverISR   endp
```

MSMMaxFrameHeaderSize

The *<TSM>GetRCB* procedure, which may be used during packet reception, employs a "LookAhead" process in which the header information of a received packet is transferred into a buffer and previewed by the TSM. This way, the TSM can first verify that it wants the packet, before the entire packet is read from the adapter.

The TSM sets the *MSMMaxFrameHeaderSize* value to the number of bytes the driver must transfer to that LookAhead buffer. Its value is equal to the *MLIDLookAheadSize* value from the configuration table plus the maximum media header size. It can be up to 128 bytes, the maximum *MLIDLookAheadSize*, plus the maximum media header size.

For example:

```
MLIDLookAheadSize = 128
Ethernet Maximum Media Header Size = 22
MSMMaxFrameHeaderSize = 128 + 22 = 150
```

To access the size, the *MSMMaxFrameHeaderSize* equate is used in conjunction with EBP, the pointer to the Adapter Data Space.

```
   mov   ecx, [ebp].MSMMaxFrameHeaderSize
```

Your driver must read the size each time before calling *<TSM>GetRCB* since it may dynamically change. The driver may optionally implement the *DriverRxLookAheadChange* routine to allow HSMs for intelligent adapters to be informed when the size changes rather than constantly checking.

For more information on the LookAhead process, see the *Packet Reception* section in Chapter 5 and the *<TSM>GetRCB* procedure in Chapter 6. Refer to the *DriverRxLookAheadChangePtr* field description of the *DriverParameterBlock* in Chapter 3 for more information on implementing this control procedure for intelligent adapters.

Example

```
DriverISR   proc                         ; ebp = Ptr to Adapter Data Space
     ⋮

ReceiveEvent:

   mov   ecx, [ebp].MSMMaxFrameHeaderSize
   lea   edi, [ebp].LookAheadBuffer
   rep   insb
   lea   esi, [ebp].LookAheadBuffer      ; Ptr to LookAhead Buffer
   mov   ecx, ReceivePacketSize          ; Get Packet Size
   call  <TSM>GetRCB                     ; Get an RCB
   jnz   PacketNotAccepted
     ⋮
```

MSMPhysNodeAddress

The *MSMPhysNodeAddress* equate is a negative offset that is used in conjunction with EBP, the pointer to the Adapter Data Space, to access the physical layer format of the node address. It is defined in the MSM.INC file as follows:

```
MSMPhysNodeAddress    equ    DriverAdapterStart –(16*4)
```

If bit 15 of the *MLIDModeFlags* is set, the driver must use *MSMPhysNodeAddress* instead of the configuration table *MLIDNodeAddress* to obtain the physical layer format of the node address. The MSM sets the *MSMPhysNodeAddress* value when the driver's initialization routine calls *MSMRegisterMLID*.

For additional information, refer to the configuration table *MLIDNodeAddress* and *MLIDModeFlags* descriptions in Chapter 3 and the canonical/non-canonical format discussion in Appendix G.

Example

```
DriverReset    proc                    ; ebp = Ptr to Adapter Data Space

   lea   esi,[ebp].MSMPhysNodeAddress
   lea   edi,[ebp].OpenAdapterNode
   movsd
   movsw
      •
      •
      •
DriverReset    endp
```

Public Variables

The following public variables are provided by the MSM and may be used by the developer as needed. The variables are available to the HSM using the "extern" statement and must be included in the HSM's linker definition file using the "import" keyword. (see Appendix A)

LogicalToPhysical / PhysicalToLogical

These variables are dword values that can be added to an address in order to convert it from a logical to physical or a physical to logical address. This may be needed by bus-master adapters designed with controllers that require absolute addresses.

For example, the HSM for a bus-master adapter may need to use *LogicalToPhysical* during *DriverSend* to convert the TCB address to an absolute address, then use *PhysicalToLogical* in *DriverISR* when the transmission is complete before giving the TCB back to the TSM.

Note: Do NOT use these variables to convert shared RAM addresses. You should use the *MLIDMemoryDecode* and *MLIDLinearMemory* fields of the configuration table for the physical and logical shared RAM addresses.

Example

```
DriverSend  proc
      ⋮
   mov   eax, esi                    ; EAX -> TCB
   add   eax, LogicalToPhysical      ; EAX = physical address
   out   dx,eax                      ; Pass address to adapter
   mov   [ebp].TCBInProcess,eax
      ⋮
DriverSend  endp


DriverISR   proc
      ⋮
TransmitComplete:
   mov   esi, [ebp].TCBInProcess     ; ESI -> TCB
   add   esi, PhysicalToLogical      ; ESI = logical address
   call  <TSM>SendComplete
      ⋮
DriverISR   endp
```

Data Structures

The structures used to transfer data between the layers of the ODI model are called Event Control Blocks (ECBs). The MSM defines two specific forms of the ECB structure.

- Receive Control Blocks (RCBs)
- Transmit Control Blocks (TCBs)

These streamlined forms of the general ECB structure are provided by the MSM to simplify driver development. Only the fields relevant to the specific packet transaction in progress are visible to the driver.

The following section describes the RCB and TCB structures. The HSM must refer to these structures during packet reception and transmission. The relationship of these MSM structures with the general ECB structure is also discussed.

Specific reception and transmission methods and related MSM/TSM support routines are described in Chapter 5.

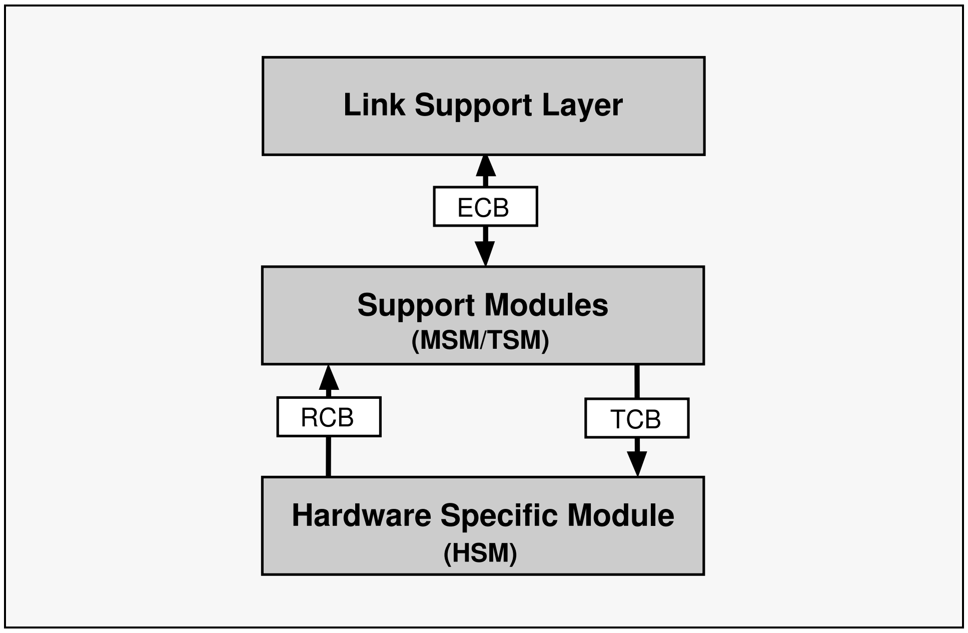


Figure 4.1 Packet transfer in the MSM/ODI Model

Receive Control Blocks

Receive Control Blocks are the structures used to transfer data from the HSM to the TSM.

Usually, when the adapter receives a packet, the HSM obtains a Receive Control Block from the TSM and copies the packet into the RCB's data fragment buffer(s). The RCB is passed back to the TSM where it is processed and transferred to the Link Support Layer. The Link Support Layer then directs it to the proper protocol stack.

On a server, there will normally be only one fragment buffer into which the received data must be copied, therefore drivers should be optimized for one fragment receives. However, the driver's receive routine should be designed to handle multiple fragment buffers if possible. Bit 10 of the *MLIDModeFlags* field in the configuration table must be set if the driver can handle fragmented receive buffers.

The following support routines are available to obtain RCBs.

- *MSMAllocateRCB*
- *<TSM>GetRCB*
- *<TSM>ProcessGetRCB*
- *<TSM>FastProcessGetRCB*

The *<TSM>GetRCB* routine provides fragmented RCBs. Drivers that cannot handle fragmented receive buffers should use *MSMAllocateRCB, <TSM>ProcessGetRCB,* or *<TSM>FastProcessGetRCB* to obtain RCBs. Chapter 5 describes specific reception methods and illustrates the use of these support routines.

The following section describes the RCB structures and fields. The structures are defined in the MSM.INC file.

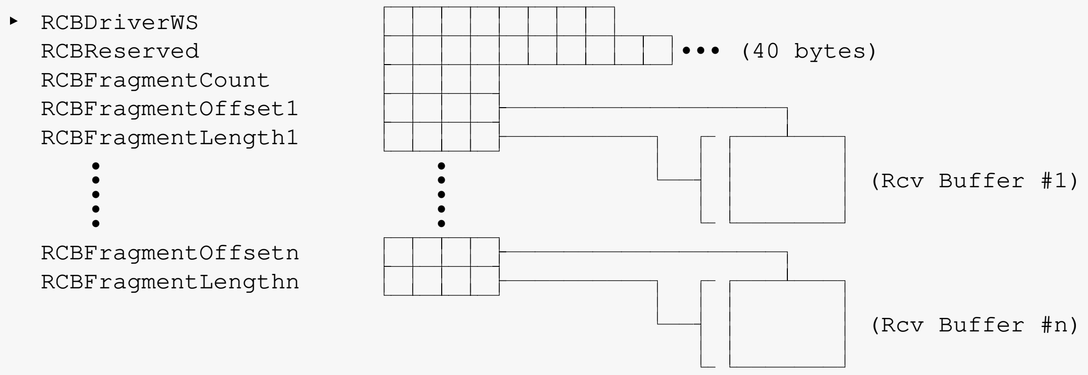
Fragmented RCB

```
RCBStructure  struc

 ‣ RCBDriverWS             db  8   dup (0)    ; Driver Workspace
   RCBReserved             db  40  dup (0)    ; Reserved for MSM use
   RCBFragmentCount        dd  ?              ; Number of Fragments
   RCBFragmentOffset1      dd  ?              ; Pointer to the 1st Fragment Buffer
   RCBFragmentLength1      dd  ?              ; Length of the 1st Fragment Buffer

RCBStructure  ends
       ⋮                                      ;*** Additional Fragment Descriptors ***

;;  RCBFragmentOffsetn     dd  ?              ; Pointer to the nth Fragment Buffer
;;  RCBFragmentLengthn     dd  ?              ; Length of the nth Fragment Buffer
```



```
 ‣ RCBDriverWS cannot be used by RX-Net drivers.
```

Figure 4.2   Fragmented Receive Control Block

Fragmented RCB Field Descriptions

Offset	Name	Bytes	Description
00h	RCBDriverWS	8	The HSM may use this field for any purpose as long as it controls the RCB. (RX-Net drivers cannot use this field)
08h	RCBReserved	40	This field should not be modified by the HSM. It contains status indicators, protocol information, and additional data maintained by the MSM and Link Support Layer.
30h	RCBFragmentCount	4	This field contains the number of data fragment descriptors to follow. Each descriptor consists of a pointer to a fragment buffer and the size of that buffer. The HSM will copy the received packet into these buffers.
34h	RCBFragmentOffset1	4	Pointer to the 1st fragment buffer.
38h	RCBFragmentLength1	4	Length of the 1st fragment buffer.
⋮	⋮	⋮	
	RCBFragmentOffsetn		Immediately following the RCB in memory are additional fragment descriptors.
	RCBFragmentLengthn		

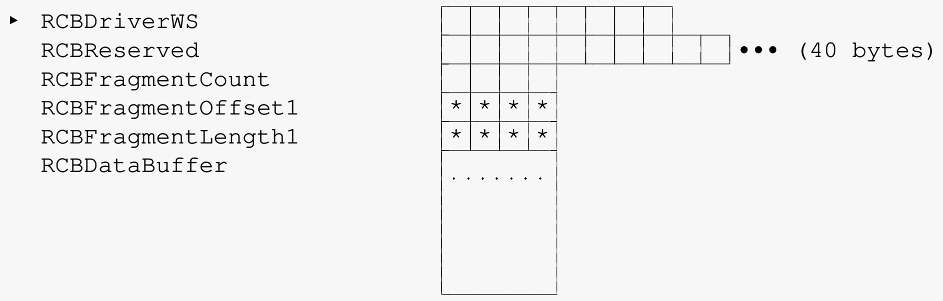
Non-Fragmented RCB

```
RCBStructure  struc

 ▸ RCBDriverWS             db  8   dup (0)                  ; Driver Workspace
   RCBReserved             db  40  dup (0)                  ; Reserved for MSM
   RCBFragmentCount        dd  1
   RCBFragmentOffset1      dd  ?
   RCBFragmentLength1      dd  ?

RCBStructure  ends

;; RCBDataBuffer           equ   RCBFragmentLength1 + 4     ; Buffer for Packet
```



▸ RCBDriverWS cannot be used by RX-Net drivers.

Figure 4.3   Non-Fragmented Receive Control Block

Non-Fragmented RCB Field Descriptions

Offset	Name	Bytes	Description
00h	RCBDriverWS	8	The HSM may use this field for any purpose as long as it controls the RCB. (RX-Net drivers cannot use this field)
08h	RCBReserved	40	This field should not be modified by the HSM. It contains status indicators, protocol information, and additional data maintained by the MSM and Link Support Layer.
30h	RCBFragmentCount	4	This field contains the number of data fragment descriptors to follow. This field will always be 1 for non-fragmented receives.
34h	RCBFragmentOffset1	4	The HSM should NOT use this field. The TSM determines this value *after* the HSM returns the RCB for processing. (It will contain a pointer to the "data" portion of the received packet in the RCBDataBuffer.)
38h	RCBFragmentLength1	4	The HSM should NOT use this field. The TSM determines this value *after* the HSM returns the RCB for processing. (It will contain the length of the "data" portion of the received packet in the RCBDataBuffer.)
3Ch	RCBDataBuffer	?	Immediately following the RCB in memory is a buffer for the received packet. The HSM copies the received packet into this buffer. For some frame types this data buffer contains MAC layer headers. (Refer to the *MSMAllocateRCB* routine for information on using a non-fragmented RCB)

Transmit Control Blocks

Transmit Control Blocks are the structures used to transfer data from the TSM to the HSM.

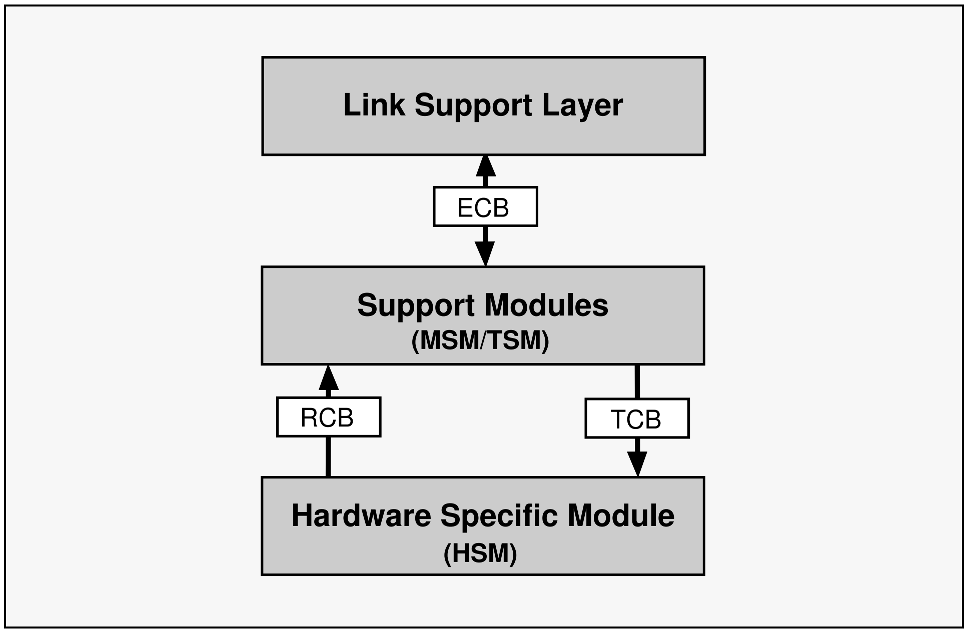


Figure 4.4 Packet transfer in the MSM/ODI Model

When sending a packet, a protocol stack assembles a list of fragment pointers in a transmit ECB and passes it to the Link Support Layer. The ECB is then transferred to the TSM where the information is processed and a TCB is constructed. The TCB structure consists of the assembled packet header and data fragment information. The TSM directs the TCB to the appropriate driver which collects the header and packet fragments and transmits the packet.

The following section describes the TCB structures used during packet transmission. The structures are defined in the MSM.INC file.

TCB for Ethernet, Token-Ring, PCN2, and FDDI

```
TCBStructure  struc

   TCBDriverWS          dd  3 dup (0)     ; Driver  Workspace
   TCBDataLen           dd  ?             ; Total Fragment + Media Header Length
   TCBFragStrucPtr      dd  ?             ; Pointer to Fragment Structure
   TCBMediaHeaderLen    dd  ?             ; Length of Media Header

TCBStructure  ends

;;  TCBMediaHeader      equ  TCBMediaHeaderLen + 4    ; Media Header Buffer
```

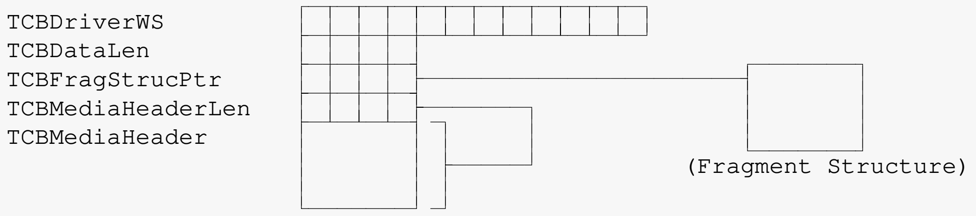


Figure 4.5 Ethernet, Token-Ring, and FDDI Transmit Control Block

TCB Field Descriptions

Offset	Name	Bytes	Description
00h	TCBDriverWS	12	The HSM may use this field for any purpose as long as it controls the TCB.
0Ch	TCBDataLen	4	This field contains the length of the packet described by the data fragments plus the media header. This value will never be 0.
10h	TCBFragStrucPtr	4	This field contains a pointer to a list of fragments defined by the *FragmentStructure* (described following the TCB section).
14h	TCBMediaHeaderLen	4	This field contains the length of the Media Header that immediately follows the TCB in memory. This value may be odd, even, or zero. A value of zero indicates a raw send. If the HSM is handed a raw send, the originating protocol stack has already included the media header in the first data fragment.
18h	TCBMediaHeader	?	Immediately following the TCB in memory is a buffer containing the Media Header that was assembled by the MSM.

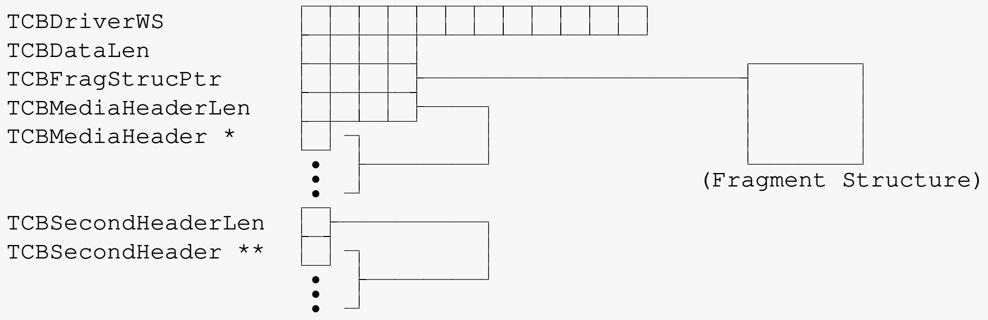
TCB for RX-Net

```
TCBStructure  struc

   TCBDriverWS         dd  3 dup (0)     ; Driver  Workspace
   TCBDataLen          dd  ?             ; Total Fragment + Media Header Length
   TCBFragStrucPtr     dd  ?             ; Pointer to Fragment Structure
   TCBMediaHeaderLen   dd  ?             ; Length of First Media Header

TCBStructure  ends

;;  TCBMediaHeader      db  3 or 4  dup (?)     ; First Media Header *
;;  TCBSecondHeaderLen  db  ?                   ; Length of Second Media Header
;;  TCBSecondHeader     db  4 or 8  dup (?)     ; Second Media Header **
```



```
 *  TCBMediaHeader is 3 bytes for Short Packets, and 4 bytes for Long or
    Exception Packets.

**  TCBSecondHeader is 4 bytes for Short or Long Packets, and 8 bytes for
    Exception Packets.
```

Figure 4.6   RX-Net Transmit Control Block

TCB Field Descriptions (RX-Net)

Offset	Name	Bytes	Description
00h	TCBDriverWS	12	This field is used by the MSM to link the TCBs
0Ch	TCBDataLen	4	This field contains the length of the packet described by the data fragments plus the media header. This value will never be 0.
10h	TCBFragStrucPtr	4	This field contains a pointer to a list of fragments defined by the *FragmentStructure* (described following this section).
14h	TCBMediaHeaderLen	4	This field contains the length of the first media header.
Immediately following the TCB in memory is a buffer containing the media header information.			
18h	TCBMediaHeader	3 or 4	This field contains the first media header. The header is 3 bytes for Short Packets and 4 bytes for Long or Exception Packets.
?	TCBSecondHeaderLen	1	This field contains the length of the second media header.
?	TCBSecondHeader	4 or 8	This field contains the second media header. The header is 4 bytes for Short or Long Packets, and 8 bytes for Exception Packets.

Note: Several fields of the above table reference the different types of RX-Net packets. Figure 4.7 on the following page shows the three RX-Net packet formats. A full description of each is included in Appendix D.

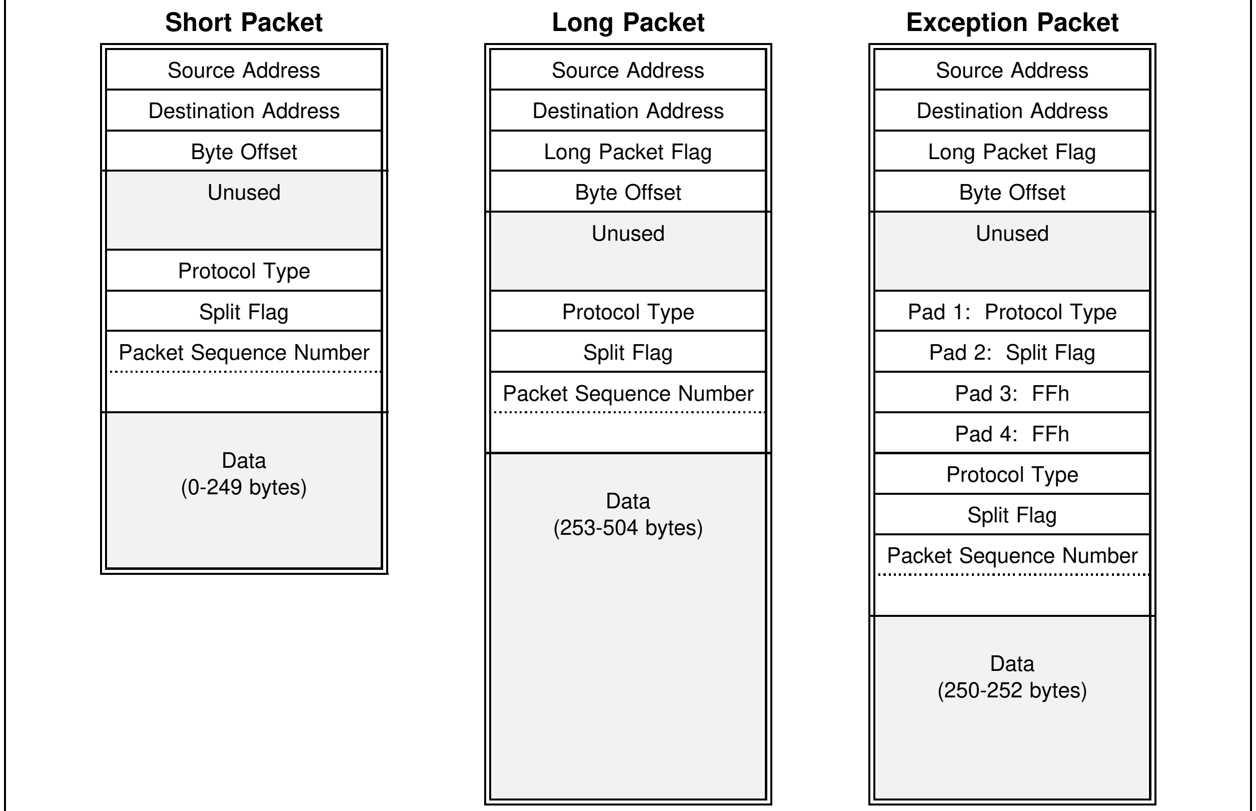


Figure 4.7 RX-Net Packet Format

Fragment Structure

The following section describes the format of the fragment structure pointed to by the *TCBFragStrucPtr* field of the Transmit Control Block.

```
FragmentCount       dd  ?         ; Number of Fragments
FragmentOffset1     dd  ?         ; Pointer to the 1st Data Fragment
FragmentLength1     dd  ?         ; Length of the 1st Data Fragment
    •
    •
    •
    •
    •
FragmentOffsetn     dd  ?         ; Pointer to the nth Data Fragment
FragmentLengthn     dd  ?         ; Length of the nth Data Fragment
```

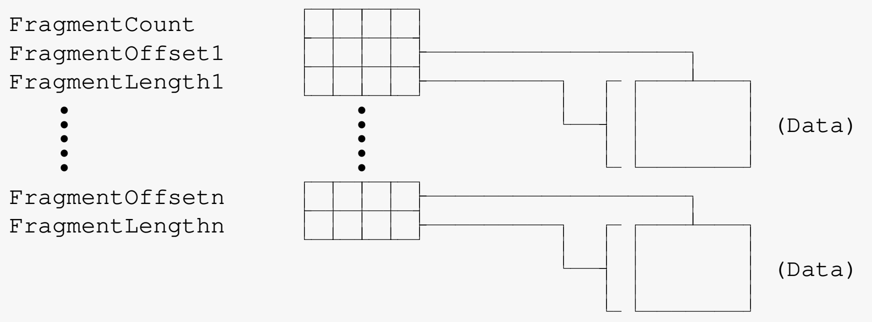


Figure 4.8 TCB Fragment Structure

Offset	Name	Bytes	Description
00h	FragmentCount	4	This field contains the number of data fragment descriptors to follow. Each descriptor consists of a pointer to a fragment buffer and the size of that buffer. The HSM collects the data from these buffers when forming the packet for transmission.
04h 08h ⋮	FragmentOffset1 FragmentLength1 ⋮ FragmentOffsetn FragmentLengthn	4 4 ⋮	Pointer to the buffer containing the first data fragment. Length of the buffer pointed to by FragmentOffset1. (These fields contain additional fragment descriptors)

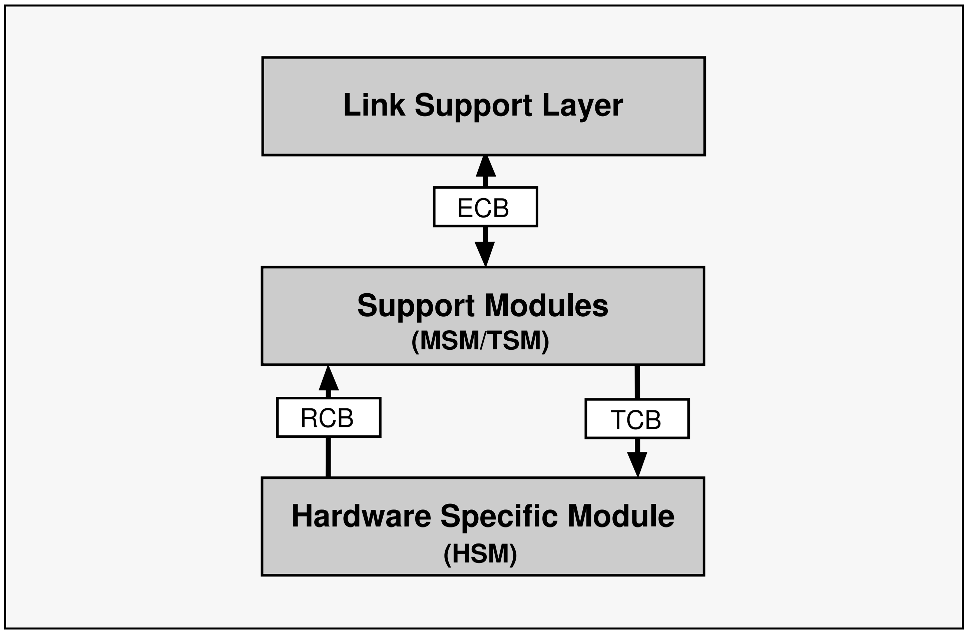
Event Control Blocks

This section defines the general Event Control Block (ECB) structure and illustrates its relationship to the RCB and TCB. This section does not apply to most drivers written with the MSM / TSM interface.

Drivers written using the MSM / TSM interface typically interact with RCBs and TCBs during packet transactions as shown in the figure below. However, some drivers may need to bypass these MSM provided structures in order to work directly with the underlying general ECB structure. This is typically the case for intelligent adapters that are designed to be *ECB aware*.

An *ECB aware* adapter/driver will completely fill in and manage all fields of the ECB during packet transactions. This shifts much of the overhead involved in packet reception and transmission to the adapter giving the processor more time to perform other tasks.

Note: This section only applies to *ECB aware* adapters/drivers.



The format of the ECB structure is shown in Figure 4.9. The same structure is used for both receiving and transmitting packets.

```
ECBStructure  struc

   Link                dd  ?          ; Forward Link used for Queuing ECBs
   BLink               dd  ?          ; Backward Link used for Queuing ECBs
   Status              dw  ?          ; Current ECB Status
   ESRAddress          dd  ?          ; Event Service Handler
   LogicalID           dw  ?          ; Protocol Logical ID
 ▸ ProtocolID          db  6 dup (?)  ; Protocol ID **
 ▸ BoardNumber         dd  ?          ; Logical Board # from Configuration Table
 ▸ ImmediateAddress    db  6 dup (?)  ; Rx...Source Addr / Tx...Destination Addr
 ▸ DriverWorkSpace     dd  ?          ; Driver Workspace / Dest and Frame Type **
   ProtocolWorkSpace   db  8 dup (?)  ; Protocol Stack Workspace
 ▸ PacketLength        dd  ?          ; Length of the Packet Data
 ▸ FragmentCount       dd  ?          ; Number of Fragments
 ▸ FragmentOffset1     dd  ?          ; Pointer to the 1st Fragment Buffer
 ▸ FragmentLength1     dd  ?          ; Length of the 1st Fragment Buffer

ECBStructure  ends
       •                              ; Additional Fields follow for both
       •                              ;   Receive and Transmit ECBs
       •



 ▸ During packet reception, these fields must be filled in by the ECB Aware
   Adapter/Driver before passing the ECB to the upper layers.  During packet
   transmission, all fields are filled in by the upper layers before passing
   the ECB to the driver.

** 802.2 frame types require special handling of the ProtocolID and
   DriverWorkSpace fields in the ECB during packet reception and transmission
   (refer to Appendix D).
```

Figure 4.9 Event Control Block

Receive ECBs vs RCBs

The general Receive ECB and the MSM's RCB essentially form a union. That is, both structures occupy the same memory space.

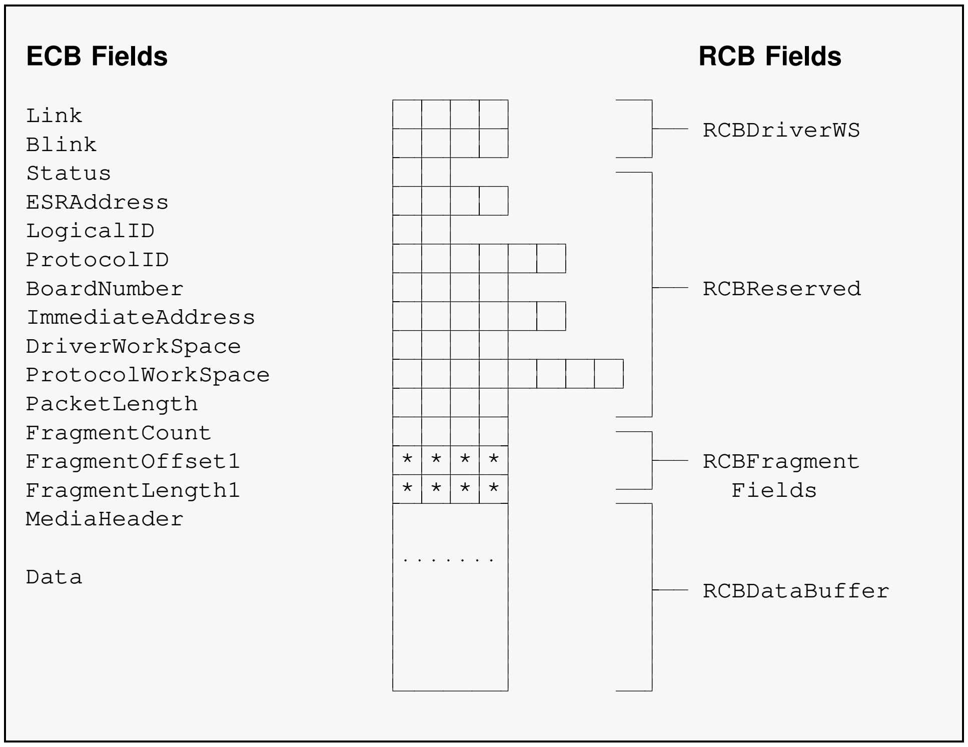


Figure 4.10 Receive ECBs vs RCBs

The ECB fields that correspond to *RCBReserved* are normally managed by the TSM. However, if an adapter is ECB aware, it can simply treat the structure as an ECB and take over the management of these fields.

Drivers written for ECB aware adapters must obtain control blocks by calling *MSMAllocateRCB*. This routine allows the driver to preallocate RCBs without the MSM initializing the fields. When a packet is received, the adapter copies it into the *RCBDataBuffer*, fills in the required fields (see Figure 4.9), and returns the structure using either the *<TSM>RcvComplete* / *MSMServiceEvents* combination or the function *<TSM>FastRcvComplete*.

Transmit ECBs vs TCBs

The general Transmit ECB and the TSM's TCB are totally separate structures. The *TCBFragStrucPtr* field of the TCB, however, points to the *FragmentCount* field of the ECB. Knowing this, it is possible to work directly with the underlying ECB by using both negative and positive offsets from this pointer.

The MSM provides another more efficient way for ECB Aware adapters to work directly with ECBs. By setting the *DriverSendWantsECBs* variable of the *DriverParameterBlock* to any non-zero value (see Chapter 3), the HSM's *DriverSend* routine will be given ECBs rather than TCBs for packet transmission. The HSM will then be responsible for building the proper media header depending on the board number.

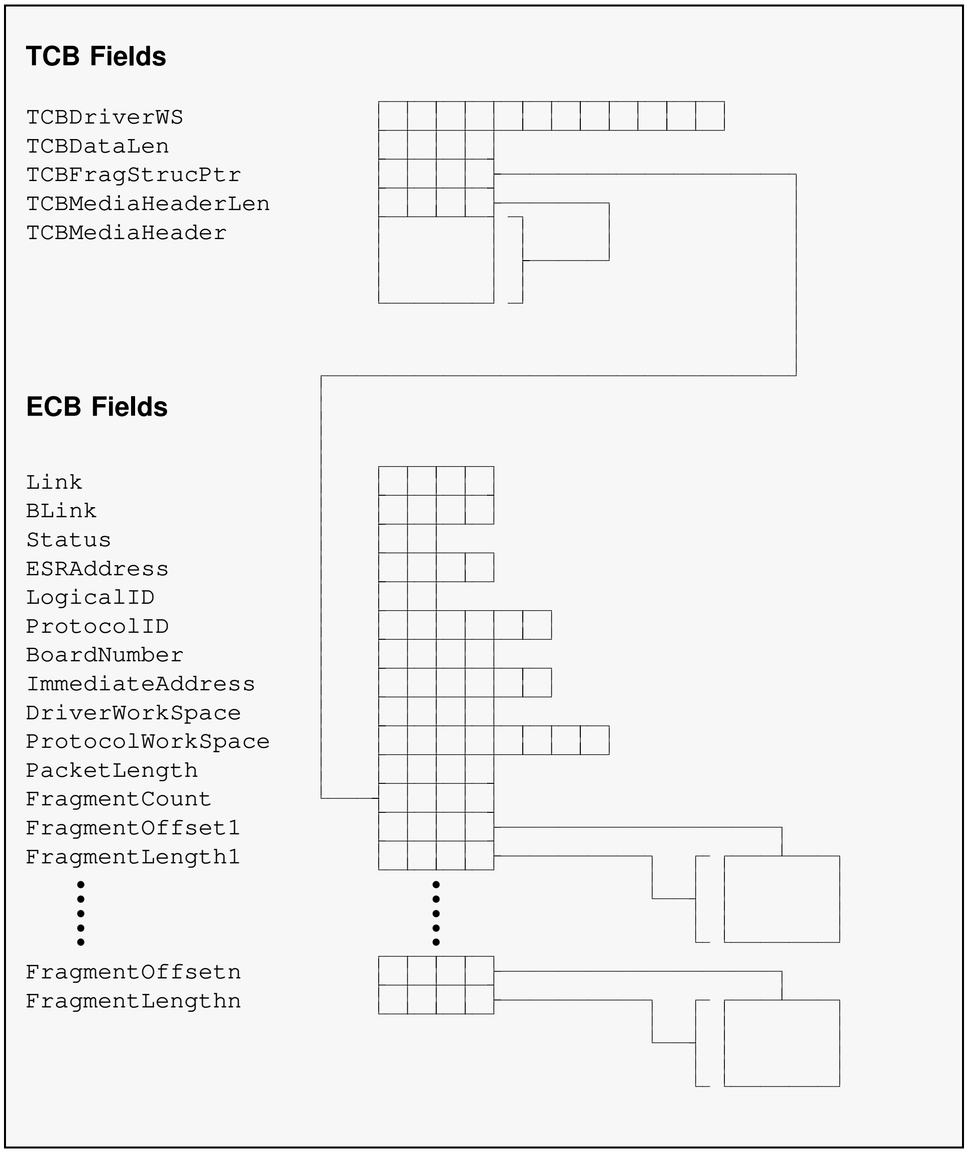


Figure 4.11 Transmit ECBs vs TCBs

ECB Field Descriptions

Offset	Name	Bytes	Description
00h	Link	4	This field contains a forward link to another ECB. The LSL uses this field for queuing ECBs. The HSM may use this field for any purpose as long as it controls the ECB.
04h	BLink	4	This field contains a backward link to another ECB. The LSL uses this field for queuing ECBs. The HSM may use this field for any purpose as long as it controls the ECB. However, if the HSM uses *DriverSendWantsECBs* to get ECBs instead of TCBs, it must **NOT** modify the transmit ECB's *BLink* field.
08h	Status	2	This field should not be modified by the HSM. The LSL uses the Status field to indicate the current state of the ECB. (i.e., currently unused, queued for sending, etc.).
0Ah	ESRAddress	4	This field should not be modified by the HSM. In receive ECBs, the LSL places a pointer to the target protocol stack's receive handler in this field and then queues the receive ECB on a hold queue. Later, the LSL polls the hold queue and routes the ECB to the proper protocol stack by calling the address in this field.
0Eh	LogicalID	2	This field should not be modified by the HSM. When a protocol stack registers with the LSL, it is assigned a logical number (0...15). This field contains that logical number or, if the packet is a raw send, the field contains the value FFFFh. On sends, the protocol stack places its own logical number in this field. On receives, the LSL places the target stack's logical number in this field.
10h	ProtocolID	6	This field contains the protocol ID (PID) value on both sends and receives. This value is stored in High-Low order. For the 802.2 frame type, on sends, this field also contains the 802.2 frame type information (Type I or II) required to build the media header (See Appendix D for an explanation of 802.2 Type I and Type II use of this field). On receives, this field always contains the DSAP value of the 802.2 header.
16h	BoardNumber	4	When a driver registers with the LSL, it is given a logical board number. The *MLIDBoardNumber* field of the configuration table contains that number (see Chapter 3). Logical board 0 is used internally in the operating system. Drivers are assigned logical board numbers 1 through 255. On receives, the HSM must fill in this field to indicate which logical board received the packet. On sends, a protocol stack fills in this field to indicate the target logical board.

ECB Field Descriptions
-(continued)-

1Ah	ImmediateAddress	6	On receives, the immediate address represents either the packet's source node address or the address of the last router that passed the packet if the packet was routed from another network. On sends, the immediate address represents either the destination node address or the destination router address. The address is stored in High-Low order. If the node address is less than six bytes, the most significant byte(s) must be padded with 0. The MSM fills in this field on receives. Addresses passed to the upper layers may be in canonical or non-canonical format depending upon whether the driver bit-swaps MSB format addresses. The stack fills in this field on sends. All addresses passed down to the MLID are in canonical format if the driver is configured to be LSB. (Refer to *MSMPhysNodeAddress* description)
20h	DriverWorkspace	4	The HSM can use this field for any purpose. The LSL will not modify the field. However, before passing a completed receive ECB to the LSL, fill in the first byte of the field (offset 20h) with the destination address type of the received packet: 00h = Direct 08h = Remote Multicast 01h = Multicast 10h = No Source Route 03h = Broadcast 20h = Error Packet 04h = Remote Unicast Set the second byte of this field (offset 21h) to indicate whether the MAC header contains one or two 802.2 control bytes: 0 = All frame types other than 802.2 1 = 802.2 header has only Ctrl0 byte (Type I) 2 = 802.2 header has Ctrl0 and Ctrl1 (Type II) See Appendix D for an explanation of 802.2 Type I and Type II.
24h	ProtocolWorkspace	8	This field is reserved for use by the protocol stack.
2Ch	PacketLength	4	This field contains the total length of the packet in bytes. This is the length of the **data portion** of the packet (not including media or SAP headers). On receives, this value is equal to the *FragmentLength1* (length may be 0). The HSM for ECB aware adapters must fill in this field. On sends, this value may be zero. The protocol stack fills in this field.

ECB Field Descriptions
-(continued)-

30h	FragmentCount	4	This field contains the number of data fragment descriptors to follow. Each descriptor consists of a pointer to a fragment buffer and the size of that buffer. On receives, the fragment count is always one or greater. On sends, the fragment count can be zero.
34h 38h	FragmentOffset1 FragmentLength1	4 4	On receives, immediately following the ECB in memory is a buffer where the HSM copies the received packet. After the packet is copied into the buffer the HSM must set the *FragmentOffset* to point around any media headers to the data portion of the packet. The HSM must also set the *FragmentLength* field to the total length of the data portion of the packet (see Figure 4.10) On sends, the *FragmentOffset* field points to the first fragment buffer containing packet data. The *FragmentLength* field specifies the length of that buffer. This value can be zero. Immediately following the ECB in memory there may be additional fragment descriptors. The HSM collects the data from these fragment buffers to form the packet for transmission (see Figure 4.11).

On receives, the memory immediately following the ECB contains:

3Ch	MediaHeader	varies	The media header of a packet is placed in this field. This field varies in length and appears only in Receive ECBs. This field is not used or present if the LAN media splits the data of a packet and transmits it in more than one frame (for example RX-Net).
??h	Data	varies	Immediately following the *MediaHeader* is the data portion of the packet.

On sends, the memory immediately following the ECB contains:

3Ch 40h . . .	FragmentOffset2 FragmentLength2 . . .	4 4 . . .	These fields contain additional fragment descriptors when the *FragmentCount* field is greater than 1.