MOST

Media **O**riented **S**ystems **T**ransport

Multimedia and Control
Networking Technology

MAMAC Specification
Rev 1.1
12/2003
Version 1.1-00

Legal Notice

SUPPORT AND FURTHER INFORMATION

For more information on the MOST technology, please contact:

MOST Cooperation
Administration
Bannwaldallee 48
D-76185 Karlsruhe
Germany

Tel: (+49) (0) 721 966 50 00
Fax: (+49) (0) 721 966 50 01
E-mail: contact@mostcooperation.com
Web: www.mostcooperation.com

Contents

References

Number	Document
[1]	MOST Specification v2.2

Document History

Changes MAMAC Specification 1V1-00 to MAMAC Specification 1V0-01

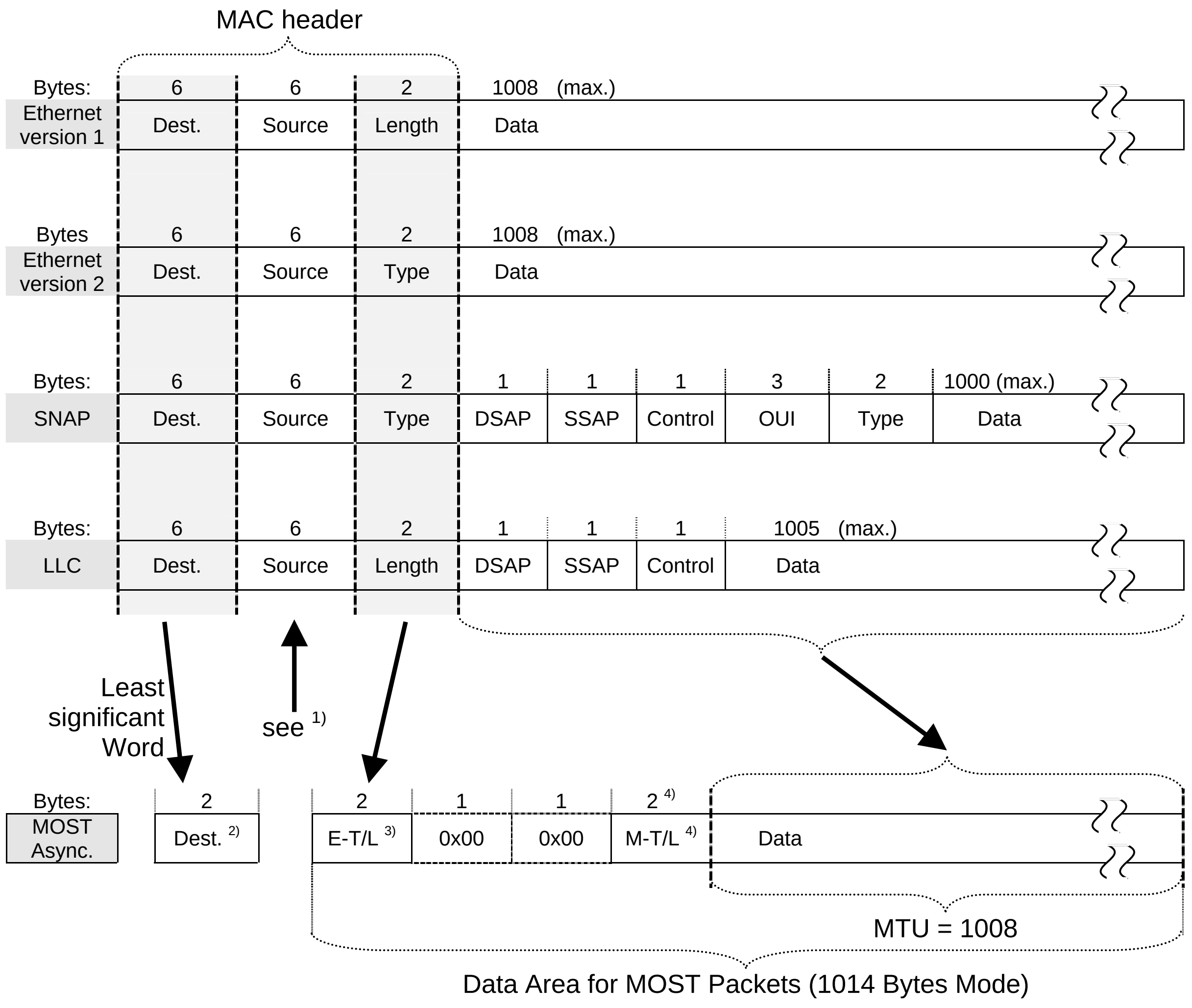
Change Ref.	Section	Changes
1V1-001	1.2.1	Removed references to OS8104
1V1-002	1.2.2	Replaced packet address with Logical Node Address (Rx/TxLog)
1V1-003	2.1	Removed rule description. Changed rule nr. 2 (only nodes that have to communicate with a node in internal packet mode have to support MAMAC48
1V1-004	2.2	Added a sentence to explain that it's not allowed for a sender to simultaneously transmit several segmented messages to the same receiver.
1V1-005	2.4	Added a sentence to explain that the number of segments in a message must not exceed 64.
1V1-006	2.4	Changed "Mode-TypeLength" to "MOST Telegram/Length"
1V1-007		Replaced Bibliography section with References. Removed chapter "Introduction"

1 MAMAC

To be able to run commonly used network protocols like TCP/IP (including IPX, NetBEUI and ARP) through the Asynchronous (Packet Data) channel of MOST, MOST Asynchronous Medium Access Control (MAMAC) was defined. MAMAC can be used simultaneously with Most High Protocol. The following sections show, how the structures of Ethernet are transported through MOST, and how addressing is done.

1.1 Packaging Frames

Due to the structures shown below, it is possible to transport Ethernet version 1, version 2, SNAP, and LLC type frames through MOST. The MTU value for MOST is 1008 bytes per frame.



Notes:
1. The Source Specifier is handled by the MOST Network automatically. No special treatment required.
2. The least two bytes of the Ethernet Destination address are used for addressing on the MOST network.
3. Ethernet Length or Type specifier.
4. MOST Telegram ID (4bits) and Length specifier (12 bits).

Figure 1-1: Comparison of Ethernet frames versus MOST Packets.

In Figure 1-1, Preamble, Start-of-frame and Checksum fields have been removed from the original Ethernet frame. This is appropriate, since they do not need to be transported through MOST.
The value of the MOST Telegram ID is 0xA or 0xB. The Length specifier value is the number of data bytes following the MOST Telegram/Length specifier (max. 1008).

1.2 Addressing Scheme

1.2.1 Address Generation

There are several possible ways to stipulate unique addresses for the MOST asynchronous packet transfer. Some mechanisms are described in this MOST specification. The mechanisms described below, are examples only. Every address configuration is possible, as long as the addresses are unique within the MOST network. Available approaches are:

- Using the Logical Node Address (Rx/TxLog) of the MOST device for address generation (a common approach, as described in section 1.2.2 on page 8).
- Using factory-set addresses.
- Manual or automatic address assignment after the system has been assembled.

Please note:
Since the address ranges 0x0300..0x03FF and 0x04000x04FF are used for Group-, Broadcast- and Node-position-Addressing in MOST networks, they should be omitted.

1.2.2 MAC Address Generation

Some Network stacks expect a 48bit Network Address (MAC Address) for Ethernet networking. This MAC address is derived from the Logical Node Address (Rx/TxLog) in the following way:

```
00:00:00:00:12:34 (If the node position address of the device is 0x1234)
```

All unused digits shall stay at 0x0. As an alternative approach, they could contain the Organizationally Unique Identifier or 'Company_ID' of the MOST system integrator, which is assigned by IEEE.

1.2.3 Handling Broadcast

The Ethernet broadcast address is 0xFFFFFFFFFFFF. This address can be mapped to 0xFFFF in MOST, to allow broadcast of packets over the MOST packet channel. On the receiving side, 0xFFFF is finally extended to Ethernet broadcast address 0xFFFFFFFFFFFF again.

2 MAMAC48

MAMAC as described above requires all nodes to run in combined mode. To be able to communicate with nodes working in internal packet mode (48 byte) a segmentation flow (MAMAC48) is added to the transport layer.
This allows differentiating the Telegram ID between:

0x8	MOST High
0x9	MOST High
0xA	MAMAC unsegmented (only if MOST in combined mode)
0xB	MAMAC segmented (needed by MOST in internal packet mode)

2.1 Rules

The following statements define some rules for MOST nodes using MAMAC:

1. If all MOST nodes using MAMAC are running in parallel-combined mode, only unsegmented messages are sent.
2. If there is, at least, one MOST node in internal packet mode (48 bytes) using MAMAC, all its communication partners must support MAMAC48 segmentation flow. Other MOST nodes in the network are not required to support MAMAC48.
3. The maximum packet size (MTU) for the protocol layer is always 1008 bytes (6 bytes needed for MOST header information).

2.2 Segment Flow

If a target signals the 48 byte packet mode, the sender splits the requested packet in segments of max 48 bytes. These packets use MOST TelegramID 0xB. The flow control consists of ACK packets that are sent back by the target to avoid loss of segments (Tx handshake). If an ACK is not received within a given time interval (default 1 sec.), the segmented transmission is aborted and the next requested packet is sent. There is no segment retries and the last segment is not acknowledged.
If the receiver detects a wrong segment sequence, it replies with an ERR packet. The sender will stop transmitting and prepare for the next packet request. It's not allowed for a sender to simultaneously transmit several segmented messages to the same receiver, i.e. all segments from the previous message must be sent before the next message can be sent.

The receiver uses a dynamic pool of receive buffers to allow multiple segments coming in from different sources. This dynamic pool should automatically free any unfinished buffers (due to lost segments) after a time-out has occurred.
The segmented receive process is started by the so called 0-Packet (segment count 0). When the last segment is received, the packet is copied up to the protocol layer in the same way as it is received in 1014 packet mode.

2.3 Mode detection

The ARP mechanism is used for dynamic recognition of MOST nodes running in 48 byte or 1014 byte mode. ARP telegrams are small enough to be transported in one 48 byte packet. To get the target's device address, ARP packets are broadcasted and the correct target replies with a packet containing its MAC address. Here, every MOST node in 48 byte packet mode signals its need for segmentation by setting bit 16 of the MAC address. Thus, the needed information is automatically stored by the protocol layer as a part of the MAC address. This single bit is used later during data flow to switch to segmented transmission mode for targets that only support the 48 bytes packet mode.

A typical ARP telegram looks like this:

ARP: **XXXX**08060000**Y**01C ... 0**MZZZZ**

Where **XXXX** is the target address (FFFF for broadcast). 0806 is the Ethernet ARP identifier. **Y** is either A for 1014 byte mode or B for 48 byte mode. **M** is the mode bit (bit 16), telling whether the node need segmentation (bit16 = 1) or not. **ZZZZ** is the address of the sender.

Mode detection is shown in the following table:

FFFF	Broadcast
0806	Ethernet ARP identifier
SSSS	Address of Sender
RRRR	Address of Receiver

Mode	Sender		Receiver
1014:1014	**1014**		**1014**
	ARP: FFFF 08060000**A**01C ... 0**0**SSSS Normal ARP as defined in MAMAC	➔ 🡐	ARP: SSSS 08060000**A**01C ... 0**0**RRRR Normal ARP reply as defined in MAMAC
1014:48	**1014**		**48**
	ARP: FFFF 08060000**A**01C ... 0**0**SSSS Normal ARP as defined in MAMAC	➔ 🡐	ARP: SSSS 08060000**B**01C ... 0**1**RRRR The receiver signals, that it can handle only 48 byte packets (MAC Address with Bit16=1). *If the Sender doesn't support 'B', ARP will fail (no response, same as target not exits)*
48:1014	**48**		**1014**
	ARP: FFFF 8060000**B**01C ... 0**1**SSSS The Sender signals that it is only using 48 byte packets. (MAC-Address with Bit16=1	➔ 🡐	ARP: SSSS 08060000**B**01C ... 0**0**RRRR The Receiver replies a 'B' to signal, that it can handle 48 byte packets. *If the Receiver doesn't support 'B' packets, ARP will fail.*
48:48	**48**		**48**
	ARP: FFFF 08060000**B**01C ... 0**1**SSSS *The Sender signals, that it is only using 48 byte packets*. (MAC-Address with Bit16=1)	➔ 🡐	ARP: SSSS 08060000**B**01C ... 0**1**RRRR Because of the Receiver only supports 48 bytes, a 'B' and MAC-Address Bit16 = 1 is replied.

Table 2-1: Mode Detection procedure.

2.4 Segment Frame Format

This is a description of how the different packets look:

E-T/L:	Ethernet-Type/Length specifier
SC:	Number of needed segments – 1
CSC:	Current segment count (= 0 for the fist packet)
M-T/L:	MOST Telegram/Length specifier (0xTLLL). T is a Telegram ID. 0xB indicates segmented packet mode. LLL is the length specifier of unsegmented MAMAC packets.

The length value in the M-T/L specifier is calculated without any header information. Each segment contains the real unsegmented packet-length.

The two most significant bits of the SC byte is reserved for the Ack and Err packet type, therefore the number of segments in a message must not exceed 64.

First Packet

E-T/L (msb)	E-T/L (lsb)	SC	CSC (= 0)	M-T/L (msb)	M-T/L (lsb)	D_0	D_1	D_2	…	D_n

Next Packets

0x00	0x00	SC	CSC	M-T/L (msb)	M-T/L (lsb)	D_{n+1}	D_{n+2}	D_{n+3}	...	D_m

Ack Packet

0xFF	0xFF	SC | 0x80	CSC	0xB0	0x00

Err Packet

0xFF	0xFF	SC | 0x40	CSC	0xB0	0x00

2.5 Data flow

The following section describes the frame contents and how the packets are segmented over the MOST packet channel. The Tx handling describes the functionality needed to split the messages on the TX side, while the RX handling section describes the receiving and merging of segments.

The following table gives detailed information about the segmented data flow.
For test, a ping packet was sent (size 64):
Device 1: called the pinger
Device 2: sends the reply (signals 48 byte mode with ARP)

Device 1		Device 2
Length: 48 08 00 02 00 B0 5C 45 00 00 5C 5F 05 00 00 20 01 … 68 05 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E	→ ←	Length:6 FF FF 82 00 B0 00
Length:48 00 00 02 01 B0 5C 6F 70 … 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69 6A	→ ←	Length:6 FF FF 82 01 B0 00
Length: 14 00 00 02 02 B0 5C 6B 6C 6D 6E 6F 70 71 72	→	
Length: 6 FF FF 82 00 B0 00	← →	Length: 48 08 00 02 00 B0 5C 45 00 00 5C 65 05 00 00 80 01 ... 68 05 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E
Length: 6 FF FF 82 01 B0 00	← →	Length: 48 00 00 02 01 B0 5C 6F 70 71 72 73 74 75 76 77 61 … 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69 6A
	←	Length: 14 00 00 02 02 B0 5C 6B 6C 6D 6E 6F 70 71 72

Table 2-2: Example of Data Flow.

2.6 Flow Charts

The following flow-charts gives a detailed information how to extend the MAMAC to support MAMAC48.

2.6.1 Variables

The flow-charts on the next pages uses the following variables for implementation of the MAMAC48:

Variable Type and Name	Description
BYTE *p	just a helper pointer to start of packet e.g. p[2] is byte2 in header (TxSegCount)
int TxSegState	State variable to identify a pending segmented transmission 0: Unsegmented transfer 1: Segmented transfer
int TxSegPending	Contains the number of current segments transmitted. Wait for ACK before sending next segment
int TxSegCount	Number of needed segments to transmit –1 filled at p[2]
int TxActSegCount	Number of current segment, started with 0, filled at p[3]
WORD TxSegBytesToTransmit	Left number of bytes to transmit
int RxSegBufIndex	Current finished RxSegBuffer or –1 for Rx pending
BYTE *RxSegBuffers[30]	RxSegBuffer pool for parallel open connections

Table 2-3: Variables used in the flow charts.

2.6.2 Init and TX Handling

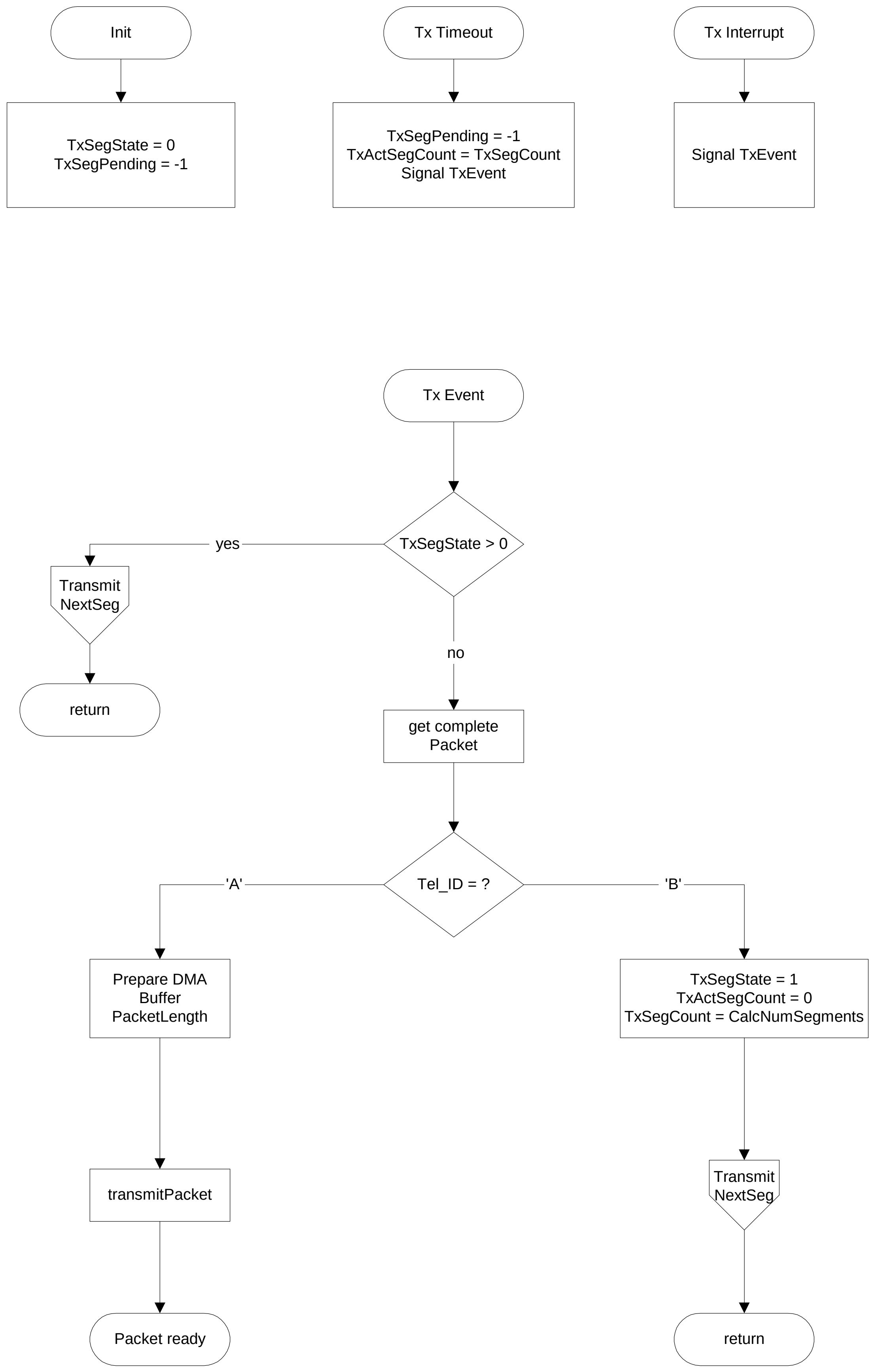


Figure 2-1: Init and TX Handling flow chart.

2.6.3 Transmit Next Segment

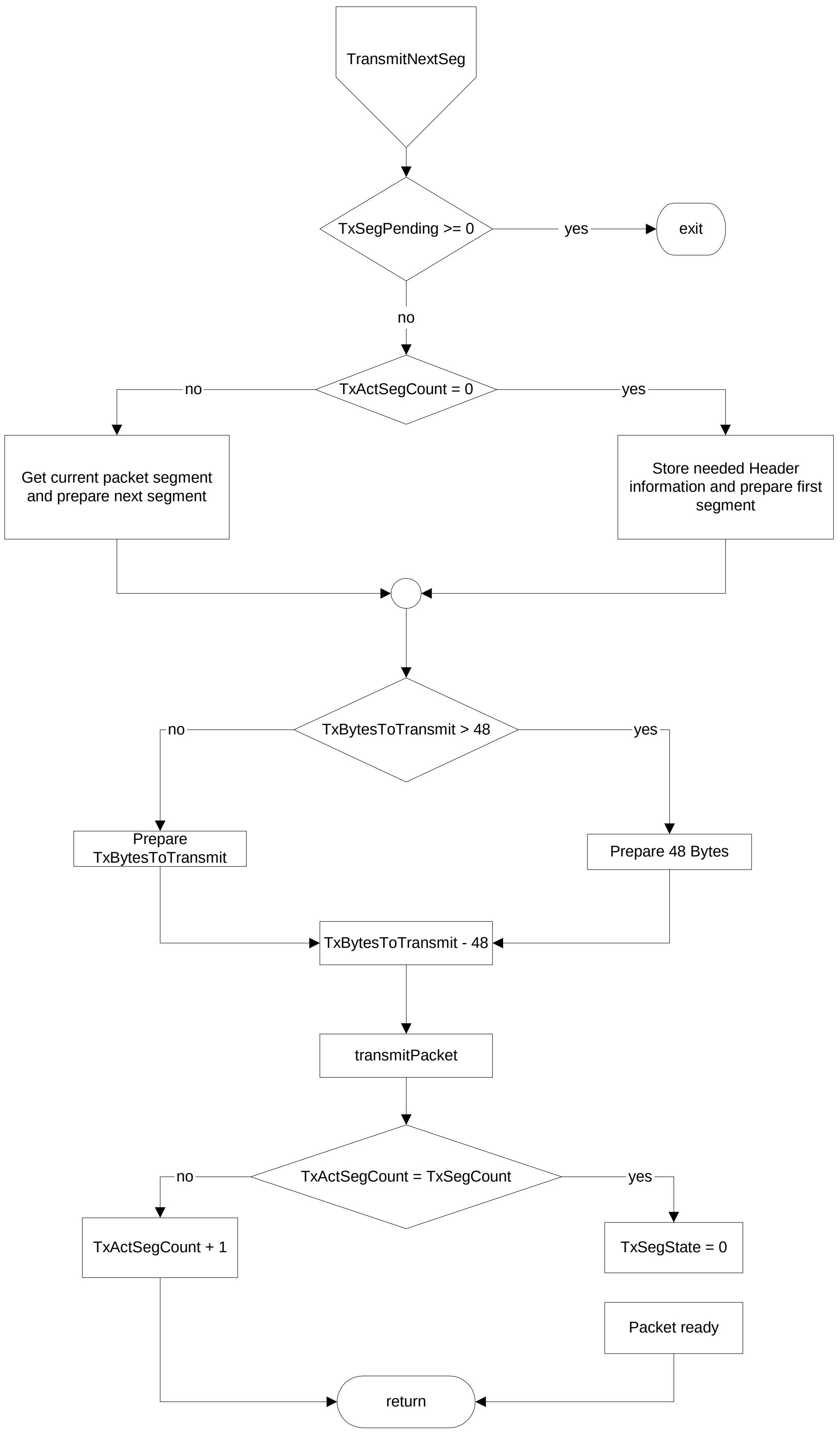


Figure 2-2: Transmit Next Segment flow chart.

2.6.4 Rx Interrupt

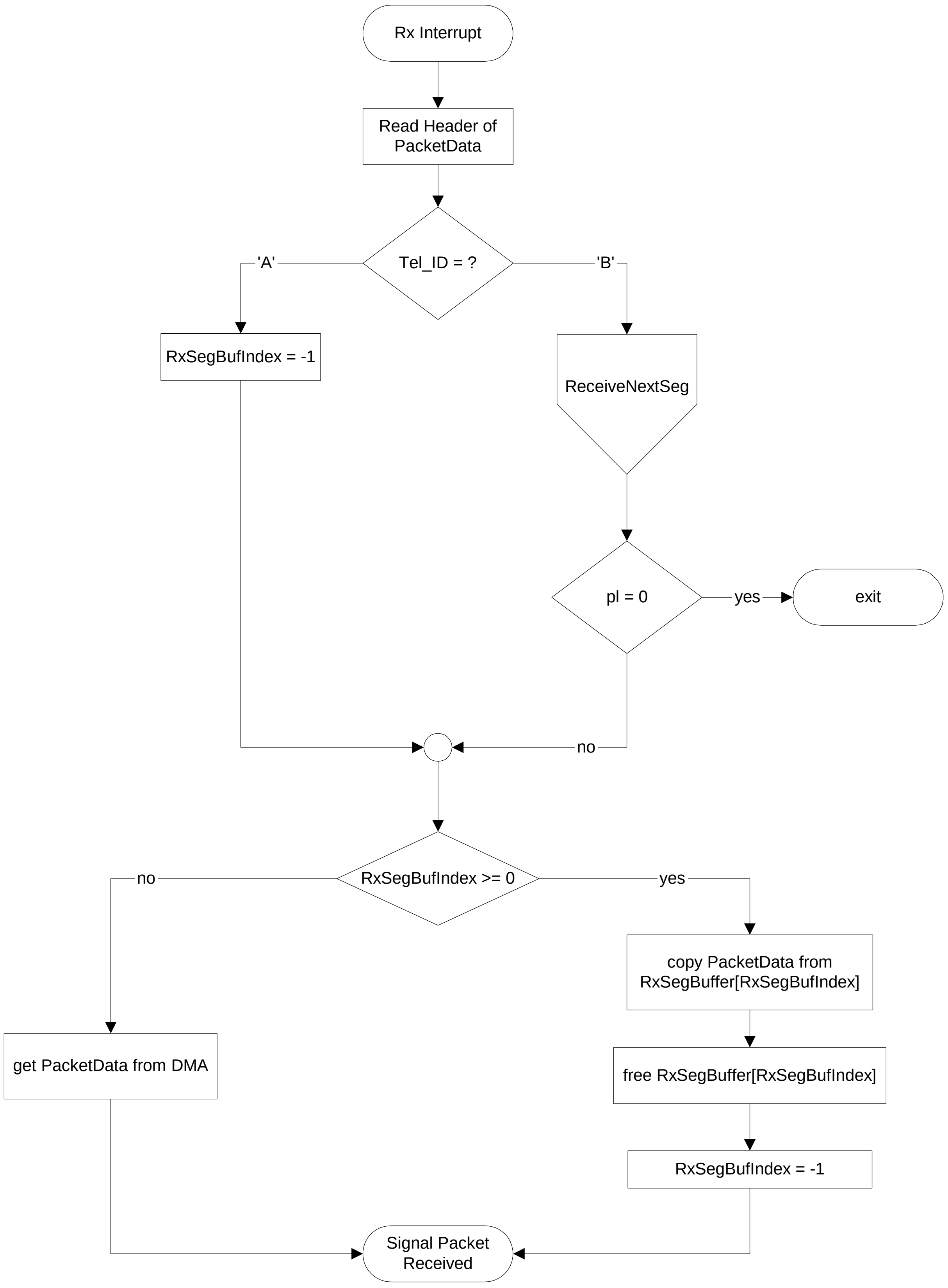


Figure 2-3: RX Interrupt flow chart.

2.6.5 Receive Next Segment

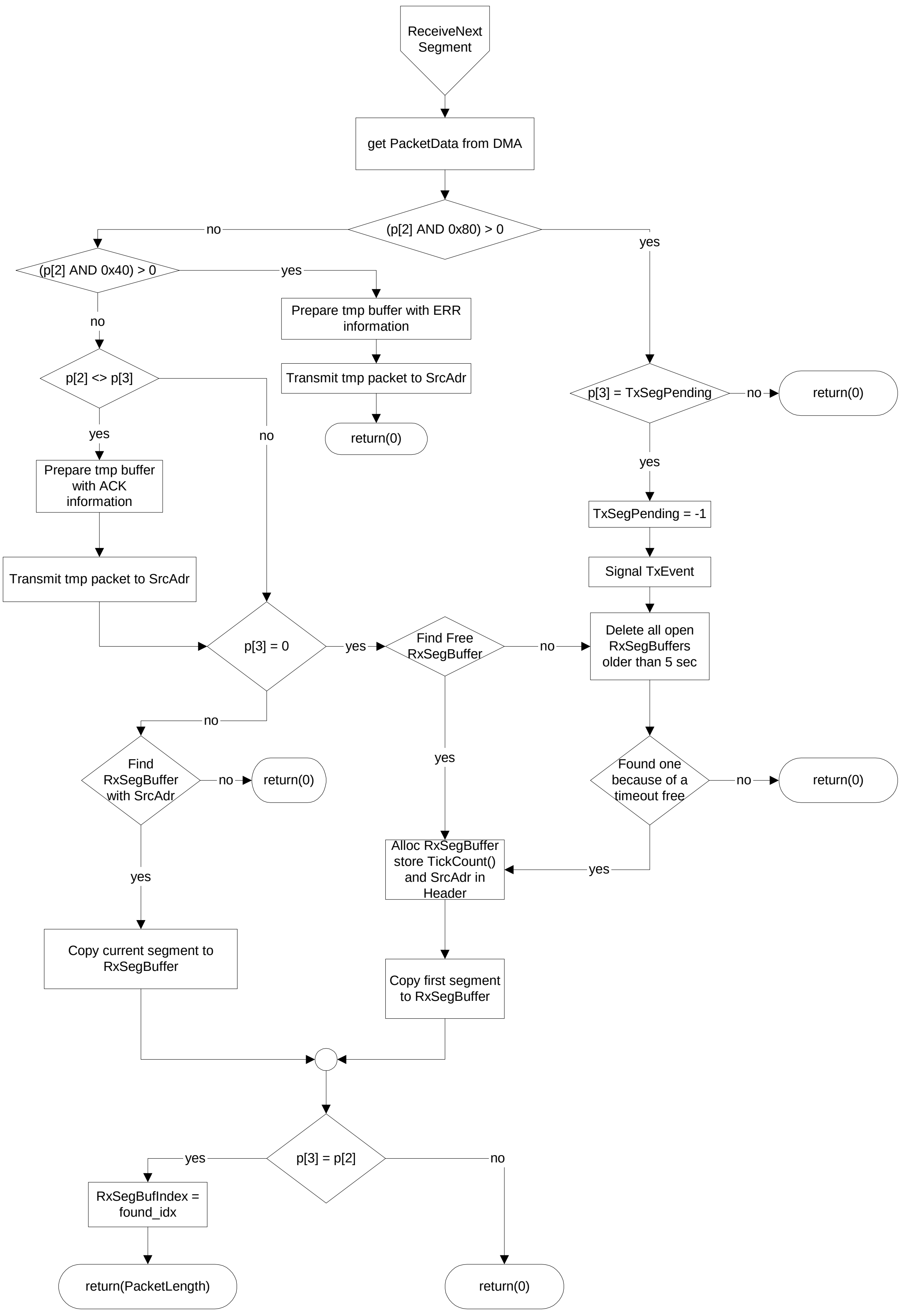

ReceiveNext Segment
get PacketData from DMA
(p[2] AND 0x80) > 0
no
yes
(p[2] AND 0x40) > 0
yes
no
Prepare tmp buffer with ERR information
Transmit tmp packet to SrcAdr
return(0)
p[2] <> p[3]
yes
no
Prepare tmp buffer with ACK information
Transmit tmp packet to SrcAdr
p[3] = TxSegPending
no
return(0)
yes
TxSegPending = -1
Signal TxEvent
Delete all open RxSegBuffers older than 5 sec
p[3] = 0
yes
Find Free RxSegBuffer
no
yes
no
Find RxSegBuffer with SrcAdr
no
return(0)
yes
Found one because of a timeout free
no
return(0)
yes
Alloc RxSegBuffer store TickCount() and SrcAdr in Header
Copy current segment to RxSegBuffer
Copy first segment to RxSegBuffer
p[3] = p[2]
yes
no
RxSegBufIndex = found_idx
return(PacketLength)
return(0)

Figure 2-4: Receive Next Segment flow chart.

3 Appendix A: Reference Timings

3.1 Ping

The following values are direct outputs of different ping packets of 2 Windows CE-PC's (Pentium I 133 MHz). These values were achieved for reference purposes only. It is not mandatory to follow these timings.

Please note:

A packet size of 980 Bytes will result in 1014 bytes (including IP and MOST header) to transmit (no segmentation of IP is needed).
The packet is transmitted in exactly 24 segments. In the timing table below, this entry gives the best performance with real packets.

A packet size of 14 Bytes can be transmitted without segmentation (ping only uses IP header). This results in a very short reply time. The value itself is useful to compare the low-level segment transmit time. For TCP/IP packets, this could not be used. The tests were made with two different settings of the Synchronous Bandwidth Control register (for more information see [7]). The first setting gives 28 bytes of asynchronous data per frame while the other gives 12 bytes per frame. Also the tests are done in both 48 byte mode and in 1014 byte mode.

SBC: 0x08

Packet size	48 - Byte Mode	
	Time in ms	**kByte / s**
100	4	50,0
200	6	66,6
400	11	72,7
800	20	80,0
1000	26	76,9
2000	50	80,0
4000	100	80,0
8000	196	81,6

Table 3-1: Ping in 48 byte mode with 28 bytes/frame

SBC: 0x0C

Packet size	48 - Byte Mode	
	Time in ms	**kByte / s**
100	5	40,0
200	7	57,1
400	13	61,5
800	23	69,5
1000	30	66,6
2000	60	66,6
4000	114	70,1
8000	225	71,1

Table 3-2: Ping in 48 byte mode with 12 bytes/frame

SBC: 0x08

Packet size	1014 - Byte Mode	
	Time in ms	kByte / s
100	1	200,0
200	1	400,0
400	2	400,0
800	3	533,3
1000	3	666,6
2000	6	666,6
4000	11	727,2
8000	20	800,0

Table 3-3: Ping in 1014 byte mode with 28 bytes/frame

SBC: 0x0C

Packet size	1014 - Byte Mode	
	Time in ms	kByte / s
100	2	100,0
200	3	133,3
400	4	200,0
800	6	266,6
1000	7	285,7
2000	13	307,7
4000	21	380,9
8000	39	410,2

Table 3-4: Ping in 1014 byte mode with 12 bytes/frame

4 Appendix B: Index of Figures

5 Appendix C: Index of Tables

6 Appendix D: INDEX

Notes:

Notes: