

# ICs for Communications

Multichannel Network Interface Controller for HDLC  
MUNICH32

PEB 20320 Version 3.4

User's Manual 01.2000

DS3

**PEB 20320****Revision History:**                      **Current Version: 01.2000**

Previous Version:                      User's Manual 1998-06-01 DS2 (V3.4)

Page (in previous Version)	Page (in current Version)	Subjects (major changes since last revision)
		Package P-TQFP-176-1 removed from User's Manual.

For questions on technology, delivery and prices please contact the Infineon Technologies Offices in Germany or the Infineon Technologies Companies and Representatives worldwide:  
see our webpage at <http://www.infineon.com>

ABM®, AOP®, ARCOFI®, ARCOFI®-BA, ARCOFI®-SP, DigiTape®, EPIC®-1, EPIC®-S, ELIC®, FALC®54, FALC®56, FALC®-E1, FALC®-LH, IDEC®, IOM®, IOM®-1, IOM®-2, IPAT®-2, ISAC®-P, ISAC®-S, ISAC®-S TE, ISAC®-P TE, ITAC®, IWE®, MUSAC®-A, OCTAT®-P, QUAT®-S, SICAT®, SICOFI®, SICOFI®-2, SICOFI®-4, SICOFI®-4μC, SLICOFI® are registered trademarks of Infineon Technologies AG.

ACE™, ASM™, ASP™, POTSWIRE™, QuadFALC™, SCOUT™ are trademarks of Infineon Technologies AG.

**Edition 01.2000**

**Published by Infineon Technologies AG,  
SC,  
Balanstraße 73,  
81541 München**

© Infineon Technologies AG 2000.  
All Rights Reserved.

**Attention please!**

As far as patents or other rights of third parties are concerned, liability is only assumed for components, not for applications, processes and circuits implemented within components or assemblies.

The information describes the type of component and shall not be considered as assured characteristics.

Terms of delivery and rights to change design reserved.

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies AG is an approved CECC manufacturer.

**Packing**

Please use the recycling operators known to you. We can also help you – get in touch with your nearest sales office. By agreement we will take packing material back, if it is sorted. You must bear the costs of transport.

For packing material that is returned to us unsorted or which we are not obliged to accept, we shall have to invoice you for any costs incurred.

**Components used in life-support devices or systems must be expressly authorized for such purpose!**

Critical components<sup>1</sup> of the Infineon Technologies AG, may only be used in life-support devices or systems<sup>2</sup> with the express written approval of the Infineon Technologies AG.

1 A critical component is a component used in a life-support device or system whose failure can reasonably be expected to cause the failure of that life-support device or system, or to affect its safety or effectiveness of that device or system.

2 Life support devices or systems are intended (a) to be implanted in the human body, or (b) to support and/or maintain and sustain human life. If they fail, it is reasonable to assume that the health of the user may be endangered.

## Preface

The Multichannel Network Interface Controller for HDLC (MUNICH32) is a Multichannel Protocol Controller for a wide area of telecommunication and data communication applications.

### Organization of this Document

This User's Manual is divided into 9 chapters. It is organized as follows:

- Chapter 1, Introduction  
Gives a general description of the product and its family, lists the key features, and presents some typical applications.
- Chapter 2, Functional Description  
This chapter provides a detailed description of the interfaces and the protocol modes.
- Chapter 3, Operational Description  
Provides a description of MUNICH32 reset procedure and initialization.
- Chapter 4, Detailed Register Description  
Gives a detailed description of the shared memory organization.
- Chapter 5, Application Notes
- Chapter 6, Application Hints
- Chapter 7, Electrical Characteristics  
Gives a detailed description of all electrical DC and AC characteristics and provides timing diagrams and values for all interfaces.
- Chapter 8, Package Outlines
- Chapter 9, Appendix  
This chapter provides source code examples.

### Your Comments

We welcome your comments on this document as we are continuously aiming at improving our documentation. Please send your remarks and suggestions by e-mail to [sc.docu\\_comments@infineon.com](mailto:sc.docu_comments@infineon.com)

Please provide in the subject of your e-mail:

device name (MUNICH32), device number (PEB 20320), device version (Version 3.4),

and in the body of your e-mail:

document type (User's Manual), issue date (01.2000) and document revision number (DS3).



Table of Contents	Page
<b>1 Introduction</b>	<b>7</b>
1.1 Features	8
1.2 Pin Configuration	11
1.3 Pin Definitions and Functions	12
1.4 Logic Symbol	22
1.5 Functional Block Diagram	23
1.6 System Integration	25
<b>2 Functional Description</b>	<b>32</b>
2.1 Serial Interface	32
2.2 Microprocessor Interface	38
2.2.1 Intel Mode	39
2.2.2 Motorola Mode	43
2.2.3 DMA Priorities	46
2.3 Basic Functional Principles	47
2.4 Detailed Protocol Description	76
2.5 Boundary Scan Unit	126
<b>3 Operational Description</b>	<b>131</b>
3.1 Reset State	131
3.2 Initialization Procedure	132
<b>4 Detailed Register Description</b>	<b>134</b>
4.1 Organization of the Shared Memory	134
4.2 Control and Configuration Section	136
4.2.1 Action Specification (Read Once After Each Action Request Pulse)	136
4.2.2 Interrupt Queue Specification	140
4.2.3 Interrupt Information	141
4.2.4 Time Slot Assignment	148
4.2.5 Channel Specification	149
4.2.6 Current Receive and Transmit Descriptor Address	161
4.3 Transmit Descriptor	162
4.4 Receive Descriptor	168
<b>5 Application Notes</b>	<b>173</b>
5.1 Test Loops	173
5.1.1 Test Loop Definitions for the MUNICH32	173
5.1.1.1 Internal Complete Test Loop	173
5.1.1.2 Internal Channelwise Test Loop	174
5.1.1.3 External Complete Test Loop	174
5.1.1.4 External Channelwise Test Loop	175
5.1.2 Test Loop Activation	176
5.1.3 Test Loop Deactivation and Switching	176
5.1.3.1 Software Operations	177

Table of Contents	Page
5.1.3.2 Hardware Reset Operations .....	177
5.1.4 Test Loop Examples .....	178
5.1.4.1 Internal Channelwise Test Loop .....	178
5.1.4.2 External Channelwise Test Loop .....	180
5.2 MUNICH32 in a LAN/WAN Router .....	182
5.2.1 Introduction .....	182
5.2.2 Hardware .....	183
5.2.3 Software .....	188
5.2.3.1 Device Driver Module MUNICH32 .....	191
5.2.3.2 Application Module MROUTE .....	194
5.2.4 Performance Considerations .....	197
5.2.5 Final Remarks .....	201
5.2.6 Adaption of the 68040 $\mu$ P Signals .....	203
5.2.7 Schematics .....	205
5.3 Memory Bus Occupancy for a Single MUNICH32 .....	214
5.3.1 Bus Occupancy Calculations .....	217
5.3.2 Bus Occupancy for Idle Tx Channels .....	218
<b>6 Application Hints .....</b>	<b>220</b>
6.1 Frequency Adaption in an Intel 368 Common Bus System .....	220
6.2 MUNICH32 Memory Space Requirement .....	223
6.3 Serial Interface to different PCM Systems .....	224
6.3.1 MUNICH32 for SIEMENS Primary Access Interface .....	224
6.3.2 MUNICH32 in Systems with MITEL ST BUS .....	227
<b>7 Electrical Characteristics .....</b>	<b>229</b>
7.1 Absolute Maximum Ratings .....	229
7.2 DC Characteristics .....	230
7.3 Capacitances .....	231
7.4 AC Characteristics .....	231
7.5 Microprocessor Interface Intel Bus Mode .....	232
7.6 Microprocessor Interface Motorola Bus Mode .....	235
<b>8 Package Outlines .....</b>	<b>242</b>
<b>9 Appendix .....</b>	<b>243</b>
9.1 Source Code Extract MUNICH32 .....	243
9.2 Source Code .....	245

## 1 Introduction

The Multichannel Network Interface Controller for HDLC (MUNICH32) is a Multichannel Protocol Controller, which handles up to 32 data channels of a full duplex PCM highway. It performs layer 2 HDLC formatting/deformatting or V.110 and X.30 protocols up to a network data rate of 38.4 Kbit/s as well as transparent transmission for the DMI mode 0, 1 and 2. The processed data is passed on to an external memory shared with one or more host processors.

MUNICH32 is compatible with the LAPD ISDN (Integrated Services Digital Network) protocol specified by CCITT as well as with HDLC, SDLC, LAPB DMI protocols. It provides any rate adaption for time slot transmission data rate from 64 Kbit/s down to 8 Kbit/s and the concatenation of any time slots to data channels, supporting the ISDN H0, H11, H12 superchannels.

Due to these functions the MUNICH32 can be used in a wide area of telecommunication and data communication applications, e.g. in central office switches, for the connection of a digital PABX to a host computer, as a central D-channel controller to 32 ISDN basic access D-channels or as a multiplexer for terminals and other peripherals. Up to 4 MUNICH32s can be connected to one PCM highway, so a D-channel controller with 128 channels can be achieved.

# Multichannel Network Interface Controller for HDLC MUNICH32

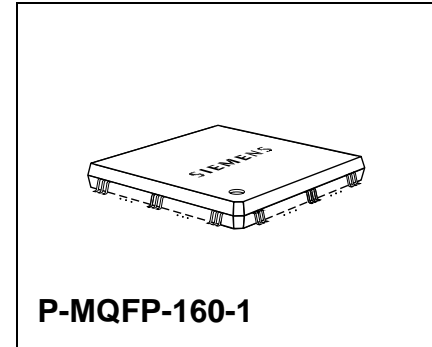
PEB 20320

Version 3.4

CMOS

## 1.1 Features

- Serial Interface
  - Up to 32 independent communication channels.
  - Serial multiplexed (full duplex) input/output for 2048-, 4096-, 1544- or 1536-Kbit/s PCM highways.
- Dynamic Programmable Channel Allocation
  - Compatible with T1/DS1 24-channel and CEPT 32-channel PCM byte format.
- Concatenation of any, not necessarily consecutive, time slots to superchannels independently for receive and transmit direction.
- Support of H0, H11, H12 ISDN-channels.
- Subchanneling on each time slot possible.
- Bit Processor Functions (adjustable for each channel)
  - HDLC Protocol
    - Automatic flag detection and transmission
    - Shared opening and closing flag
    - Detection of interframe-time-fill change, generation of interframe-time-fill '1's or flags
    - Zero bit insertion
    - Flag stuffing and flag adjustment for rate adaption
    - CRC generation and checking (16 or 32 bits)
    - Transparent CRC option per channel and/or per message
    - Error detection (abort, long frame, CRC error, 2 categories of short frames, non-octet frame content)
    - Special short frame mode to allow reception of 'frames' with a least on byte length
    - ABORT/IDLE generation



Type	Package
PEB 20320	P-MQFP-160-1



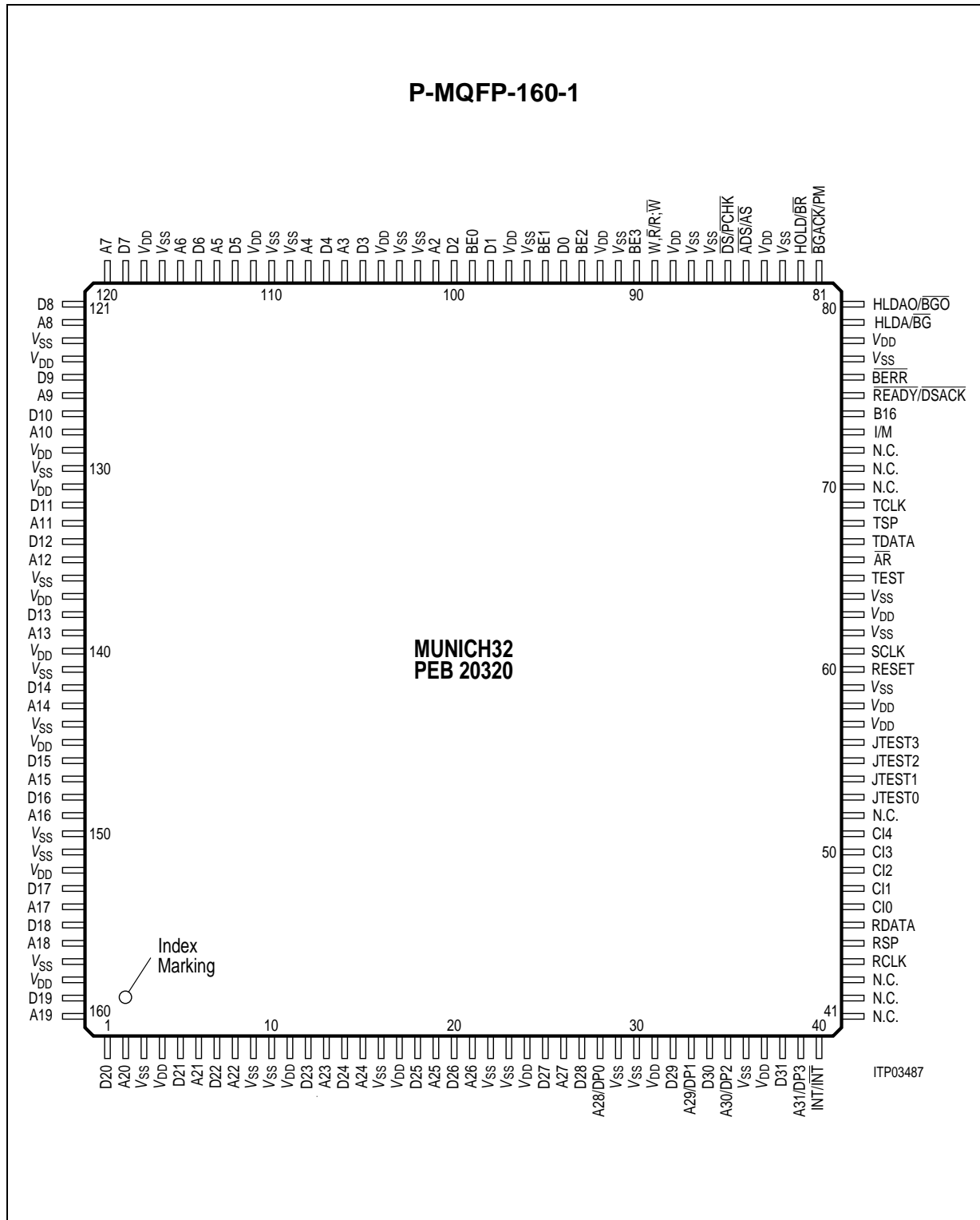
- V.110/X.30 Protocol
  - Automatic synchronization in receive direction, automatic generation of the synchronization pattern in transmit direction
  - E / S / X bits freely programmable in transmit direction, can be changed during transmission; changes monitored and reported in receive direction
  - Generation/detection of loss of synchronism
  - Bit framing with network data rates from 600 bit/s up to 38.4 Kbit/s
- Transparent Mode A
  - Slot synchronous transparent transmission/reception without frame structure
  - Bit-overwrite with fill/mask bits
  - Flag generation, flag stuffing, flag extraction, flag generation in the abort case with programmable flag
- Transparent Mode B
  - Transparent transmission/reception in frames delimited by 00<sub>H</sub> flags
  - Shared opening and closing flag
  - Flag stuffing, flag detection, flag generation in the abort case
  - Error detection (non octet frame content, short frame, long frame)
- Transparent Mode R
  - Transparent transmission/reception with GSM 08.60 frame structure
  - Automatic 0000<sub>H</sub> flag generation/detection
  - Support of 40, 39<sup>1</sup>/<sub>2</sub>, 40<sup>1</sup>/<sub>2</sub> octet frames
  - Error detection (non octet frame content, short frame, long frame)
- Protocol Independent
  - Channel inversion (data, flags, IDLE code)
  - Format conventions as in CCITT Q.921 § 2.8
  - Data over- and underflow detected
- Processor Interface
  - ON-CHIP 64-channel DMA controller with buffer chaining capability.
  - Compatible with Motorola 68020 processor family and Intel 32-bit processor (80386).
  - 32 bit data bus and 32 bit address bus (4 Gbyte RAM addressable, Motorola and Intel non-parity) or 28 bit address bus (256 Mbyte RAM addressable, Intel parity)
  - Intel parity mode with data byte parity (4 parity bits)
    - Parity check for read accesses
    - Parity generation for write accesses
  - Interrupt-circular buffer with variable size
  - Maskable interrupts for each channel
  - $\mu$ P interface buffer of depth 16 long words for adaptive bus occupation

---

**Introduction**

- General
  - Connection of up to four MUNICH32 supporting a 128-channel basic access D-channel controller.
  - ON-CHIP receive and transmit data buffer; the buffer size is 256 bytes each.
  - HDLC protocol or transparent mode, support of ECMA 102, CCITT I4.63 RA2, V.110, X.30, DMI mode 0, 1, 2 (bit rate adaption), GSM 08.60 TRAU frames.
  - LOOP mode, complete loop as well as single channel loop
  - JTAG boundary scan test
  - Advanced low-power CMOS technology
  - TTL-compatible inputs/outputs
  - 160 pin P-MQFP package

## 1.2 Pin Configuration (top view)



**Figure 1**

### 1.3 Pin Definitions and Functions

#### Pin Definitions and Functions

Pin No. P-MQFP-160-1	Symbol	Input (I) Output (O)	Function
83, 87, 88, 92, 97, 103, 104, 110, 111, 117, 123, 130, 136, 141, 144, 150, 151, 157, 3, 9, 10, 16, 22, 23, 29, 30, 36, 59, 62, 64, 77	$V_{SS}$	I	<b>Ground (0 V)</b> All pins must have the same level.
73	I/M	I	<b>Intel Bus Mode or Motorola Bus Mode</b> By connecting this pin to either $V_{SS}$ or $V_{DD}$ the bus interface can be adapted to either Intel or Motorola environment. The data is interpreted either in Intel or Motorola manner; i.e. little or big endian convention. I/M = low: Intel bus mode I/M = high: Motorola bus mode
39	A31  DP3	O  I/O	<b>Address Bit 31</b> (Intel non-parity/Motorola) tristate when unused. <b>Data Parity 3</b> (Intel parity mode), bidirectional tristate line containing/ expecting parity bit of D(31:24).
35	A30  DP2	O  I/O	<b>Address Bit 30</b> (Intel non-parity/Motorola) tristate when unused. <b>Data Parity 2</b> (Intel parity mode), bidirectional tristate line containing/ expecting parity bit of D(23:16).

*Note: Input pins that are unused in a specific configuration must be strapped to  $V_{SS}$ .  
I/O or output pins that are unused in a specific configuration must be left open!*

## Pin Definitions and Functions (cont'd)

Pin No. P-MQFP-160-1	Symbol	Input (I) Output (O)	Function
33	A29	O	<b>Address Bit 29</b> (Intel non-parity/Motorola) tristate when unused.
	DP1	I/O	<b>Data Parity 1</b> (Intel parity mode), bidirectional tristate line containing/ expecting parity bit of D(15:8)
28	A28	O	<b>Address Bit 28</b> (Intel non-parity/Motorola) tristate when unused
	DP0	I/O	<b>Data Parity 0</b> (Intel parity mode), bidirectional tristate line containing/ expecting parity bit of D(7:0)
26, 21, 19, 15, 13, 8, 6, 2, 160, 156, 154, 149, 147, 143, 139, 135, 133, 128, 126, 122, 120, 116, 114, 109, 107, 102	A(27:2)	O	<b>Address Bus</b> tristate when unused.
91, 94, 96, 100	BE(3:0)	O	<b>Byte Enable</b> (Intel bus mode) The MUNICH32 provides word and long word transfer. The byte enables determine the address offset to the address A31 ... A2, the actual word has been stored to. <b>Address Offset Size</b> (Motorola mode) Indicates the number of bytes remaining to be transferred for this access. These signals define the active sections of the data bus. In both cases these signals are tristate when unused. See <b>Chapter 2.2</b> for details.

## Introduction

### Pin Definitions and Functions (cont'd)

Pin No. P-MQFP-160-1	Symbol	Input (I) Output (O)	Function
38, 34, 32, 27, 25, 20, 18, 14, 12, 7, 5, 1, 159, 153, 148, 146, 142, 138, 134, 132, 127, 125, 121, 119, 115, 113, 108, 106, 101, 99, 95	D(31:0)	I/O	<b>Data Bus</b> The data bus lines are bidirectional tristate lines which interface with the system's data bus.
86	$\overline{DS}$	O	<b>Data Strobe</b> (Motorola mode) This signal indicates that valid data is to be placed on the data bus (read cycle) or has been placed on the data bus by the MUNICH32 (write cycle).
	$\overline{PCHK}$	O	<b>Parity Check</b> (Intel parity mode) This signal indicates, whether the parity bits of a read cycle are valid ( $\overline{PCHK}$ high) or invalid ( $\overline{PCHK}$ low). See <b>Chapter 2.2.1</b> for details.
84, 93, 89, 98, 105, 112, 118, 124, 129, 131, 137, 140, 145, 152, 158, 4, 11, 17, 24, 31, 37, 57, 58, 63, 78	$V_{DD}$	I	<b>Supply voltage</b> 5 V $\pm$ 5% All pins must have the same level.
85	$\overline{ADS}$	O	<b>Address Status</b> (Intel bus mode) This signal indicates that a valid bus cycle definition and address are being driven at the pins.
	$\overline{AS}$	O	<b>Address Strobe</b> (Motorola bus mode) A valid address is transmitted on the address bus at the falling edge of $\overline{AS}$ .  In both cases this signal is active low and tristate when unused.

## Introduction

### Pin Definitions and Functions (cont'd)

Pin No. P-MQFP-160-1	Symbol	Input (I) Output (O)	Function
90	$W/\overline{R}$	O	<b>Write/Read</b> (Intel bus mode) This signal distinguishes write from read operations.
	$R/\overline{W}$	O	<b>Read/Write</b> (Motorola bus mode) This signal distinguishes between read and write operations.  In both cases this signal is tristate when unused.
75	$\overline{READY}$	I	<b>Ready</b> (Intel bus mode) This signal indicates that the current bus cycle is complete. When $\overline{READY}$ is asserted during a read cycle the MUNICH32 latches the input data and terminates the cycle. When $\overline{READY}$ is asserted during a write cycle the MUNICH32 terminates the cycle.
	$\overline{DSACK}$	I	<b>Data Transfer Acknowledge</b> (Motorola bus mode) This active low input indicates that a data transfer may be performed. During a read cycle data becomes valid at the falling edge of $\overline{DSACK}$ . The data is latched internally and the bus cycle is terminated. During a write cycle the falling edge of $\overline{DSACK}$ marks the latching of data and the bus cycle is terminated.

## Introduction

### Pin Definitions and Functions (cont'd)

Pin No. P-MQFP-160-1	Symbol	Input (I) Output (O)	Function
76	$\overline{\text{BERR}}$	I	<p><b>Bus Error</b> (Intel and Motorola bus mode) This active low signal informs the MUNICH32 that a bus cycle error has occurred. The MUNICH32 terminates the bus cycle.</p> <p>In case of an erroneous read cycle in the control and configuration section an 'Action Request Fail' interrupt is generated and the action is suspended. In case of an erroneous read cycle in the transmit data section the corresponding frame is aborted and a FO interrupt is generated. In all other cases of read or write cycles terminated with an error condition no further actions are performed by the MUNICH32. Please see <b>Chapter 2.2</b>, 'Microprocessor Interface', first paragraph and <b>Figure 18</b>.</p> <p>As bus cycles are executed without time limit this signal prevents a hang-up situation of the MUNICH32.</p>
74	B16	I	<p><b>Word Operation</b> Setting this bit to <math>V_{DD}</math> causes the MUNICH32 to perform 32-bit long word accesses to the shared memory, setting it to <math>V_{SS}</math> causes the MUNICH32 to perform 16-bit word accesses on the data lines D(15:0) only. In 16-bit word access mode the data lines D(31:16) should be left open.</p> <p>This bit is <b>not</b> dynamic and should be set to <math>V_{DD}</math> in Intel parity mode.</p>



## Introduction

### Pin Definitions and Functions (cont'd)

Pin No. P-MQFP-160-1	Symbol	Input (I) Output (O)	Function
82	HOLD	O	<b>Bus Hold Request</b> (Intel bus mode) This signal is driven high when the MUNICH32 requests the control of the bus.
	$\overline{\text{BR}}$	I/O	<b>Bus Request</b> (Motorola bus mode) This signal is driven low when the MUNICH32 requests the control of the bus and is interpreted when another MUNICH32 wants to be the bus master.
79	HLDA	I	<b>Bus Hold Acknowledge</b> (Intel bus mode) This active high signal indicates that the processor has released the control of the bus. The MUNICH32 starts the bus cycles.
	$\overline{\text{BG}}$	I	<b>Bus Grant</b> (Motorola bus mode) This active low signal indicates that the MUNICH32 may assume the bus mastership.
81	$\overline{\text{BGACK}}$	I/O	<b>Bus Grant Acknowledge</b> (Motorola bus mode) This signal is driven low by the device, when it has become the bus master. It also informs the MUNICH32 whether another device is bus master.
	PM	I	<b>Parity Mode</b> (Intel bus mode) This signal has to be strapped to $V_{DD}$ before reset to enable the Intel parity mode or to $V_{SS}$ before reset to enable the Intel non-parity mode. It has to be left strapped during reset and operation.

## Introduction

## Pin Definitions and Functions (cont'd)

Pin No. P-MQFP-160-1	Symbol	Input (I) Output (O)	Function
80	HLDAO	O	<b>Bus Hold Acknowledge Passing ON</b> (Intel bus mode) If another MUNICH32 has initiated a HOLD REQUEST the HOLD ACKNOWLEDGE is passed on via HLDAO. The MUNICH32 does not give another HOLD REQUEST before the HOLD ACKNOWLEDGE has been deactivated in order to prevent blocking in the case of continuous request by one MUNICH32.
	$\overline{\text{BGO}}$	O	<b>Bus Grant Acknowledge</b> (Motorola bus mode) If the MUNICH32 has not requested the bus mastership it passes on the BUS GRANT. The MUNICH32 does not give another BUS REQUEST before the BUS REQUEST and the BUS GRANT ACKNOWLEDGE have been deactivated in order to prevent blocking in the case of continuous request by one MUNICH32.
66	$\overline{\text{AR}}$	I	<b>Action Request</b> $\overline{\text{AR}}$ must be pulsed low to cause an action of the MUNICH32. The $\overline{\text{AR}}$ is activated for updating the mode and channel configurations, setting a test loop, or initializing the interrupt queue. The min. time between Reset and first $\overline{\text{AR}}$ is 500 $\mu\text{s}$ .

## Introduction

### Pin Definitions and Functions (cont'd)

Pin No. P-MQFP-160-1	Symbol	Input (I) Output (O)	Function
40	INT/ $\overline{\text{INT}}$	O	<b>Interrupt Request</b> An interrupt is given when a transmission/reception error is detected, frames are received or transmitted, or a host initiated action is performed. The interrupt pulse signal interacts with a write cycle to the shared memory. The data written into the interrupt queue contains the interrupt specification. The interrupt is active high for Intel bus mode and active low for Motorola bus mode.
44	RCLK	I	<b>Receive Clock</b> This clock provides the data clock for RDA T1/DS1 24-channel 1.544 MHz 24-channel 1.536 MHz CEPT 32-channel 2.048 MHz 32-channel 4.096 MHz
45	RSP	I	<b>Receive Synchronization Pulse</b> This signal provides the reference for the receive PCM frame synchronization. It marks the first bit in the PCM frame.
46	RDATA	I	<b>Receive Data</b> Serial data is received at this PCM input port. The MUNICH32 supports the T1/DS1 24-channel PCM format, the CEPT 32-channel PCM format as well as a 32-channel PCM format with 4.096-Mbit/s bit rate.
61	SCLK	I	<b>System Clock</b> PCM highway system clock highway frequency 32-channel 16.384 MHz 2.048 or 4.096 MHz 24-channel 12.288 MHz 1.536 MHz 24-channel 12.352 MHz 1.544 MHz

## Introduction

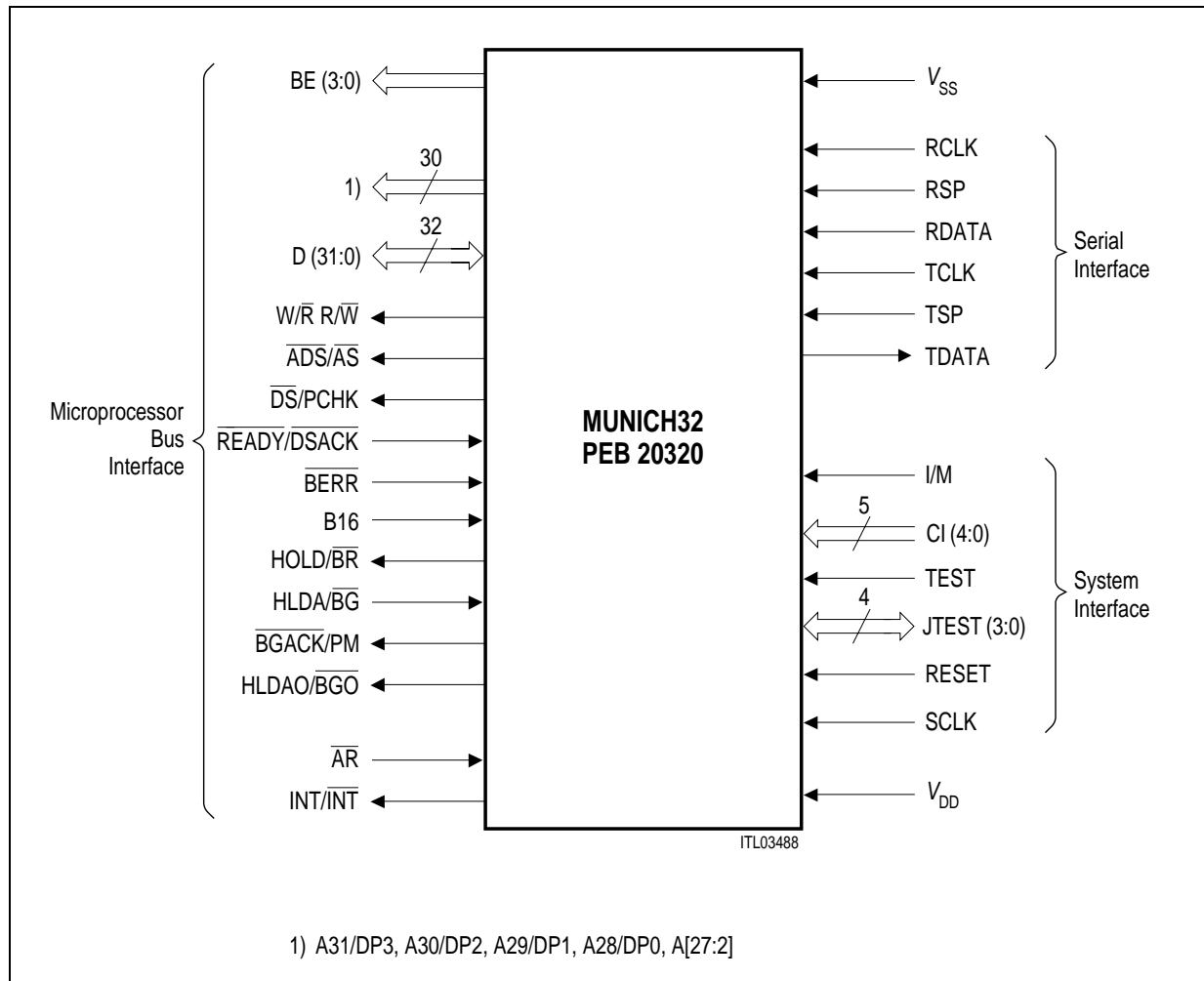
### Pin Definitions and Functions (cont'd)

Pin No. P-MQFP-160-1	Symbol	Input (I) Output (O)	Function
51 ... 47	CI(4:0)	I	<b>Chip Identification</b> Up to four MUNICH32 can be connected to the PCM highway. These inputs define the start address of the control section pointer in the shared memory. CI4 is the polarity of A31 ... A22 CI3 is the polarity of A21 ... A16 CI2 is the polarity of A15 ... A4 CI1 is the polarity of A3 CI0 is the polarity of A2 A1, A0 are always '00'
56 ... 53	JTEST (3:0)	I/O	<b>Test Pins</b> The MUNICH32 supports the JTAG boundary scan test and the JTAG test standards.
65	TEST	I	<b>Test</b> If this bit is set to $V_{DD}$ MUNICH32 works in a test mode. For the functional working mode this bit must be set to $V_{SS}$ .
67	TDATA	O	<b>Transmit Data</b> Serial data is sent by this PCM output port is push-pull for active bits in the PCM frame and tristate for inactive bits.
68	TSP	I	<b>Transmit Synchronization Pulse</b> This signal provides the reference for the transmit frame synchronization. It marks the last bit in the PCM frame.
69	TCLK	I	<b>Transmit Clock</b> This clock provides the data clock for TDATA T1/DS1 24-channel 1.544 MHz 24-channel 1.536 MHz CEPT 32-channel 2.048 MHz 32-channel 4.096 MHz

## Pin Definitions and Functions (cont'd)

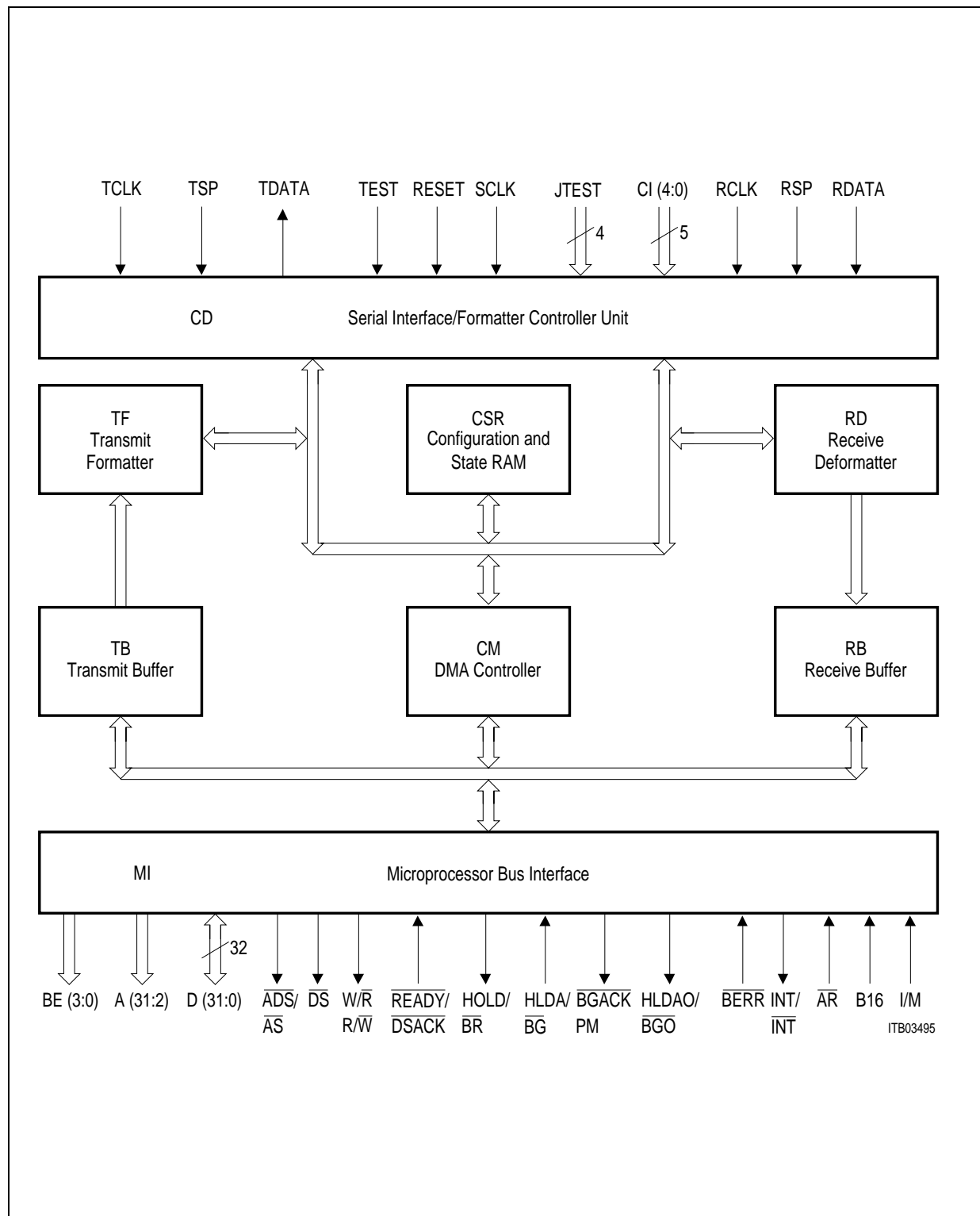
Pin No. P-MQFP-160-1	Symbol	Input (I) Output (O)	Function
60	RESET	I	<b>Reset</b>
41, 42, 43, 52, 70, 71, 72	N.C.	-	<b>No Connect</b> These pins are reserved and should not be connected

## 1.4 Logic Symbol



**Figure 2**  
**MUNICH32 Logic Symbol**

## 1.5 Functional Block Diagram



**Figure 3**  
**Block Diagram of MUNICH32**

---

## Introduction

The internal functions of MUNICH32 are partitioned into 8 major blocks.

1. Serial Interface, Formatter Control Unit CD
  - Parallel-Serial conversion, PCM timing, switching of the test loops, controlling of the multiplex procedure.
2. Transmit Formatter TF
  - HDLC frame, bit stuffing, flag generation, flag stuffing and adjustment, CRC generation, transparent mode transmission and V.110, X.30 80 bit framing.
3. Transmit Buffer TB
  - Buffer size of 64 long words allocated to the channels, i.e. eight PCM frames can be stored before transmission, individual channel capacity programmable.
4. Receive Deformatter RD
  - HDLC frame, zero-bit deletion, flag detection, CRC checking, transparent mode reception and V.110, X.30 80 bit framing.
5. Receive Buffer RB
  - Buffer size of 64 long words allocated to the channels, i.e. eight PCM frames can be stored, individual long words are freely accessible by each channel.
6. Configuration and State RAM CSR
  - Since the Transmit Formatter, Receive Deformatter are used in a multiplex manner, the state and configuration information of each channel has to be stored.
7. DMA Controller CM
  - Interrupt processing, memory address calculation, chaining list handling, chip configuration.
8.  $\mu$ P interface MI
  - Motorola/Intel microprocessor interface.

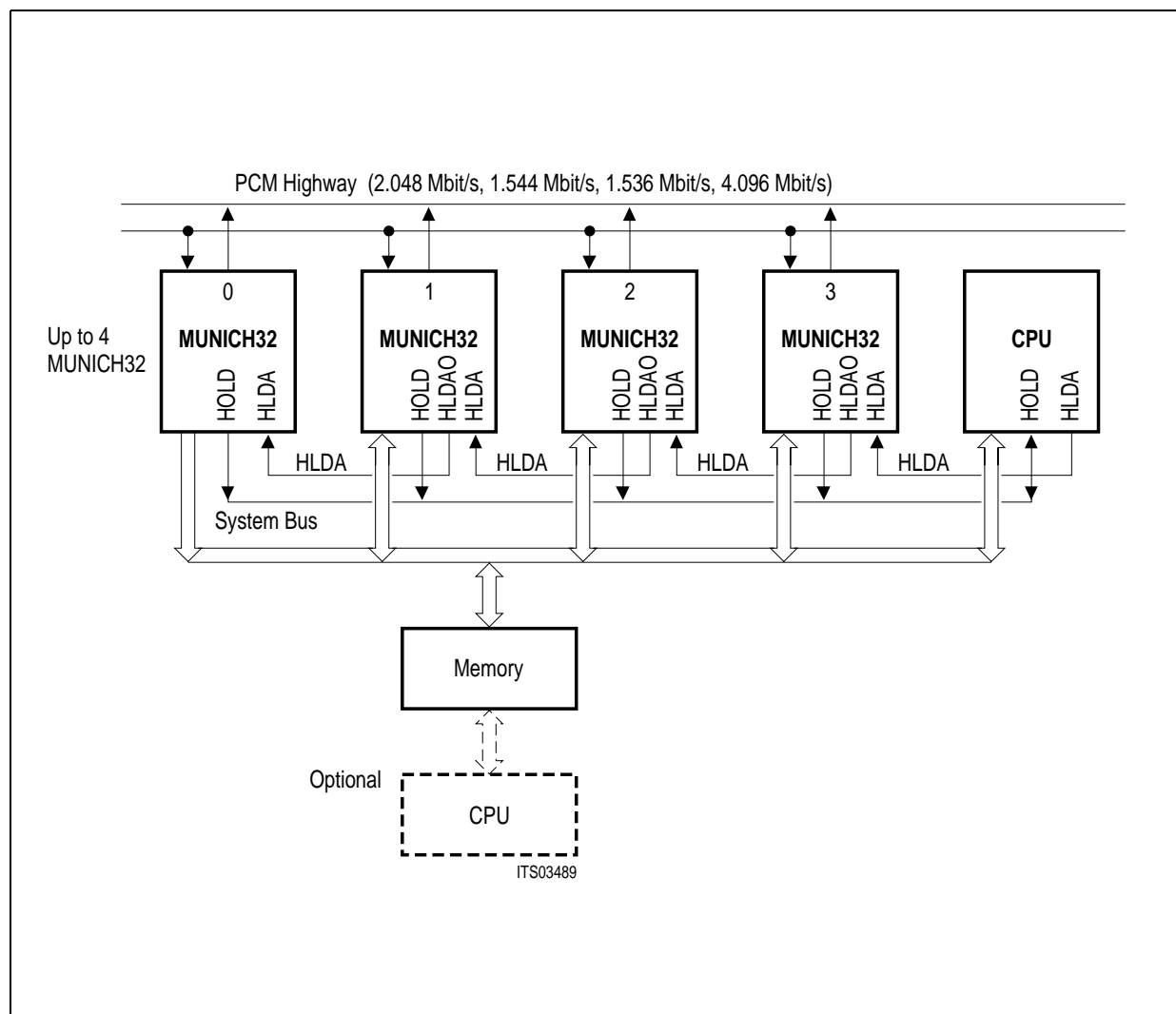


## 1.6 System Integration

The MUNICH32 is designed to handle up to 32 data channels of a PCM highway. It transfers the data between the PCM highway and a memory shared with a host processor via a 32-bit  $\mu$ P interface. At the same time it performs protocol formatting and deformatting as well as rate adaption for each channel independently. The host sets the operating mode, bit rate adaption method and time slot allocation of each channel by writing the information into the shared memory.

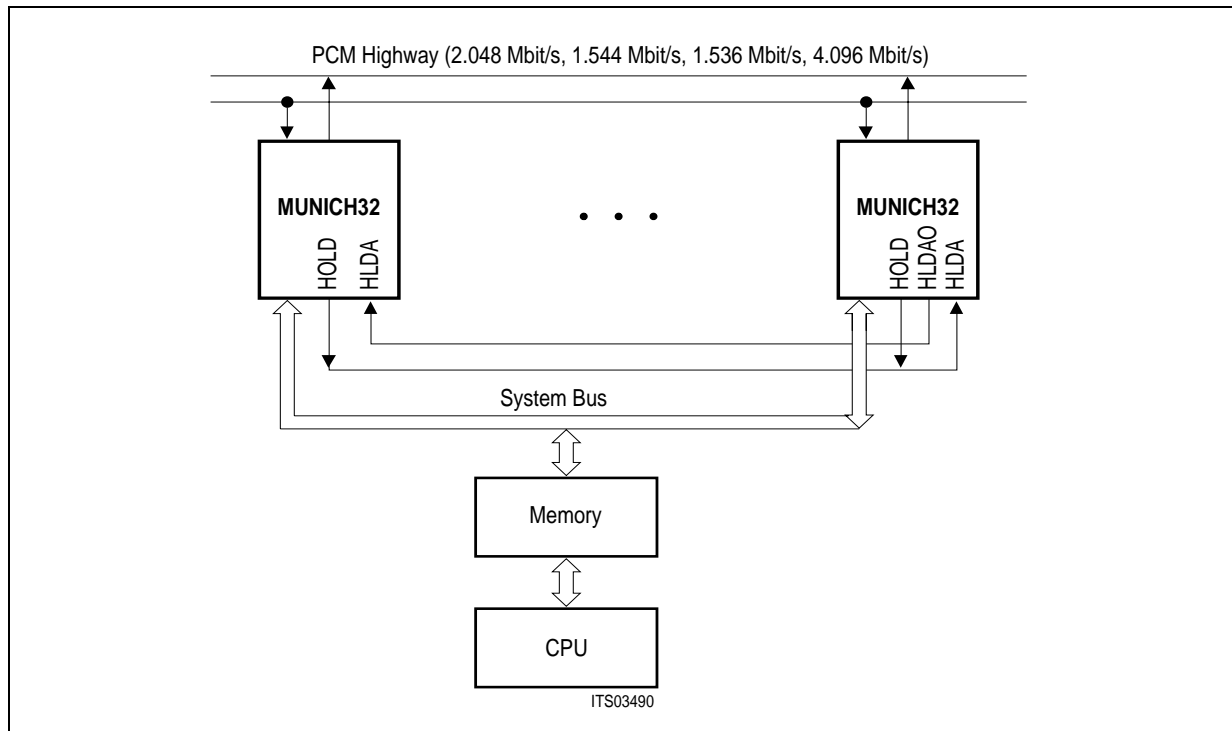
Using subchanneling each time slot can be shared between up to four MUNICH32s; so that in one single time slot four different D-channels can be handled by four MUNICH32s.

**Figure 4, Figure 5 and Figure 6** give a general overview of system integration of the MUNICH32.

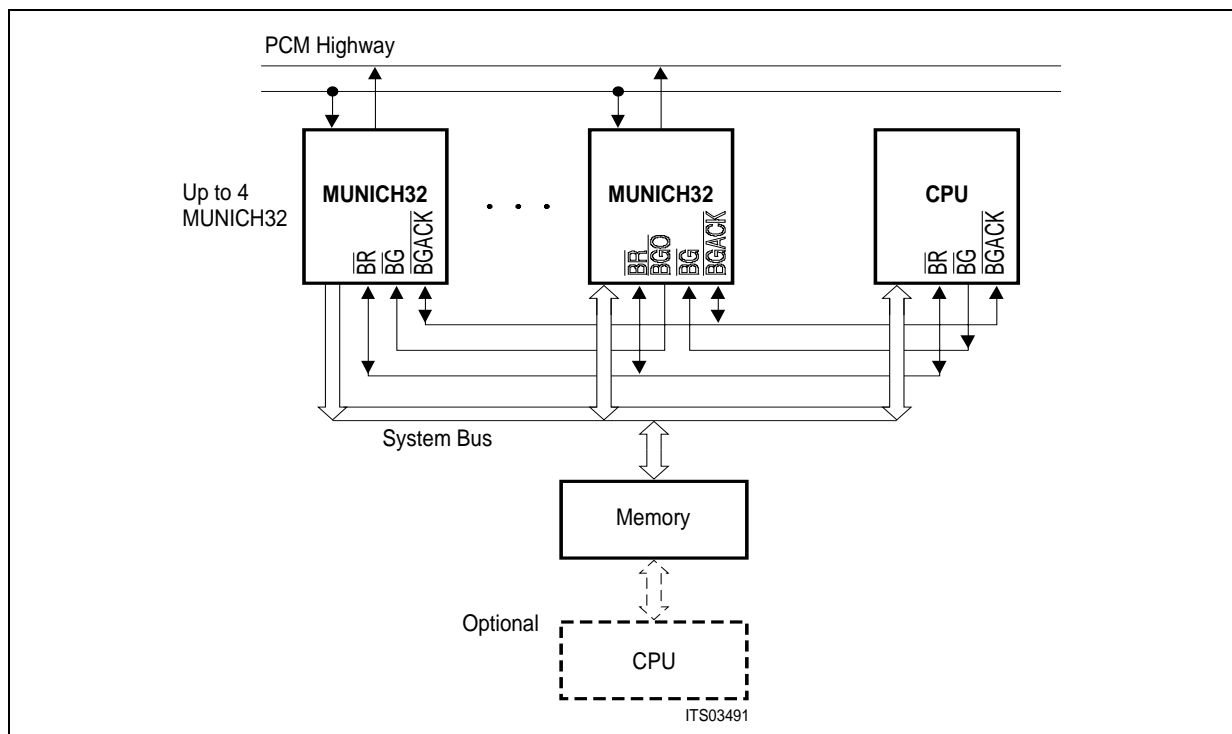


**Figure 4**  
**General System Integration (Intel Bus Mode)**

## Introduction



**Figure 5**  
**General System Interface (Intel Bus Mode)**



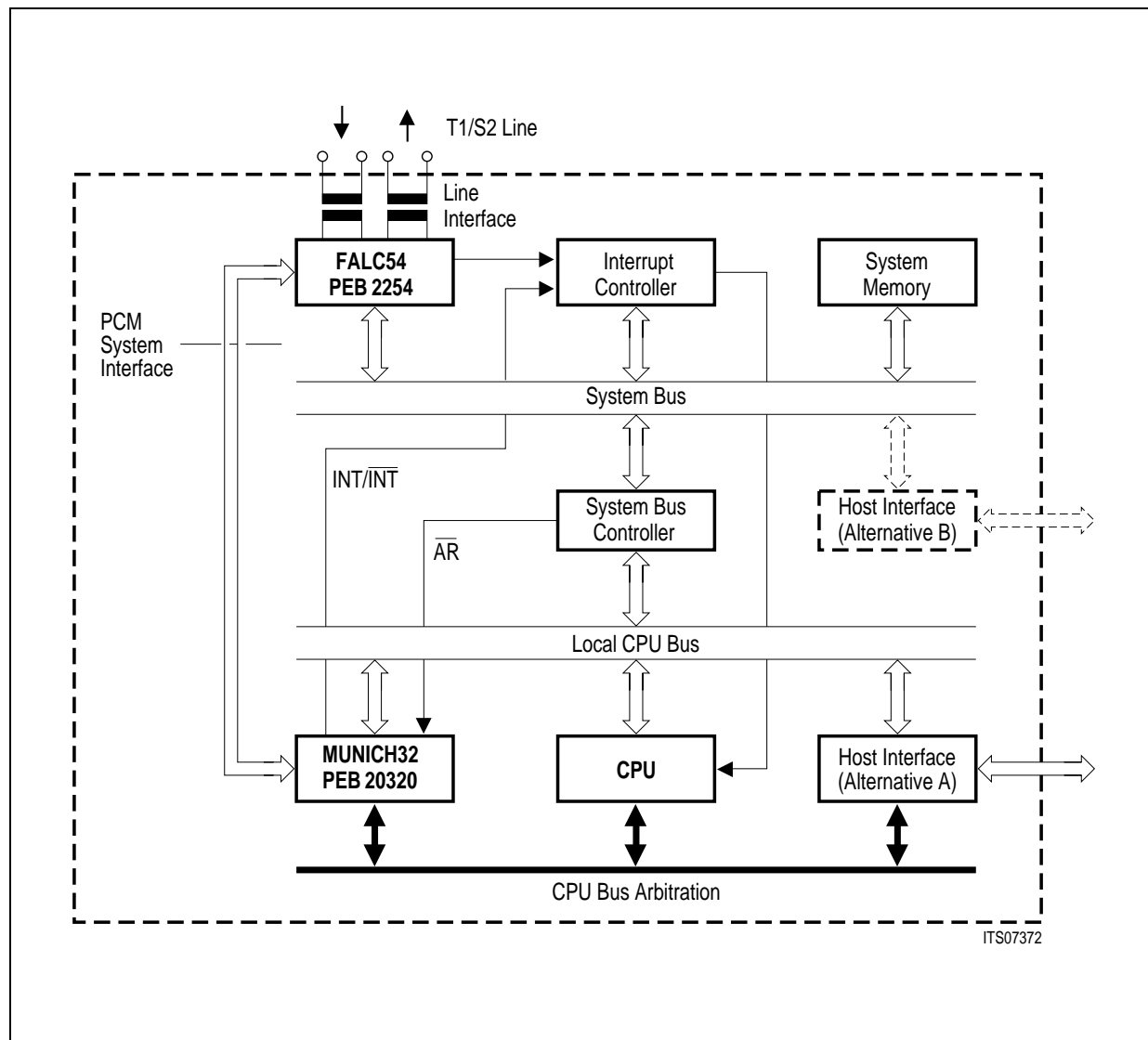
**Figure 6**  
**General System Interface (Motorola Bus Mode)**

## Introduction

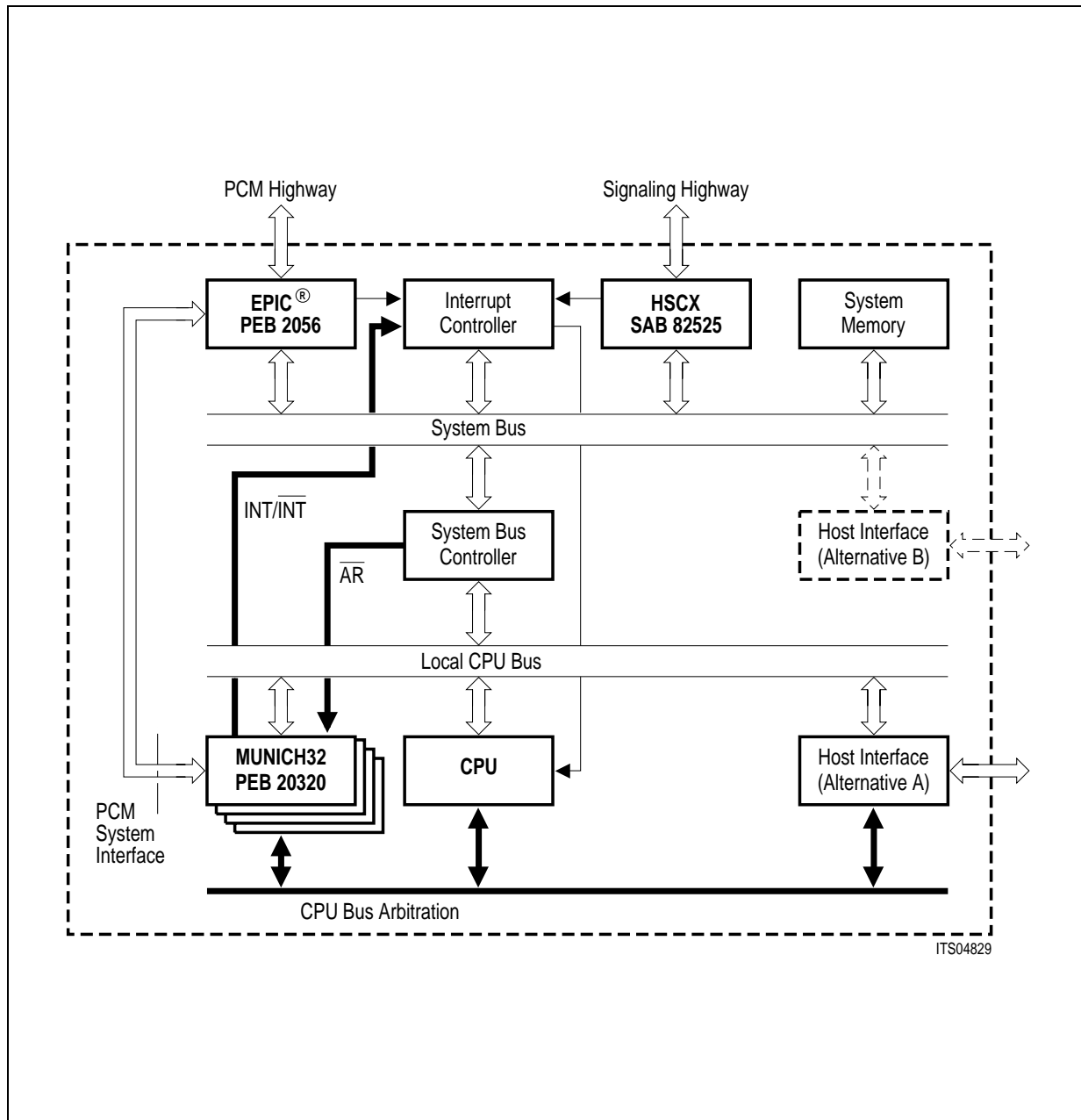
MUNICH32's bus interface consists of a 32 bit bidirectional data bus (D31 ... D0), 32/28 Address lines (A31 ... A2, BE3 ... BE0) or (A27 ... A2, BE3 ... BE0), four data byte parity lines DP(3:0), five lines ( $\overline{W/R/R/W}$ ,  $\overline{ADS/AS}$ ,  $\overline{DS/PCHK}$ ,  $\overline{BERR}$  READY/ $\overline{DSACK}$ ) to control and monitor the bus cycle, one action request and one Interrupt line.

The system bus allocation is controlled by the four signals ( $\overline{HOLD/BR}$ ,  $\overline{HLDA/BG}$ ,  $\overline{BGACK}$ ,  $\overline{HLDAO/BGO}$ ). A mode pin allows the bus interface to be configured for either Intel or Motorola mode. An operation mode pin B16 enables the transfer of a 32 bit long word in two consecutive 16 bit word operations.

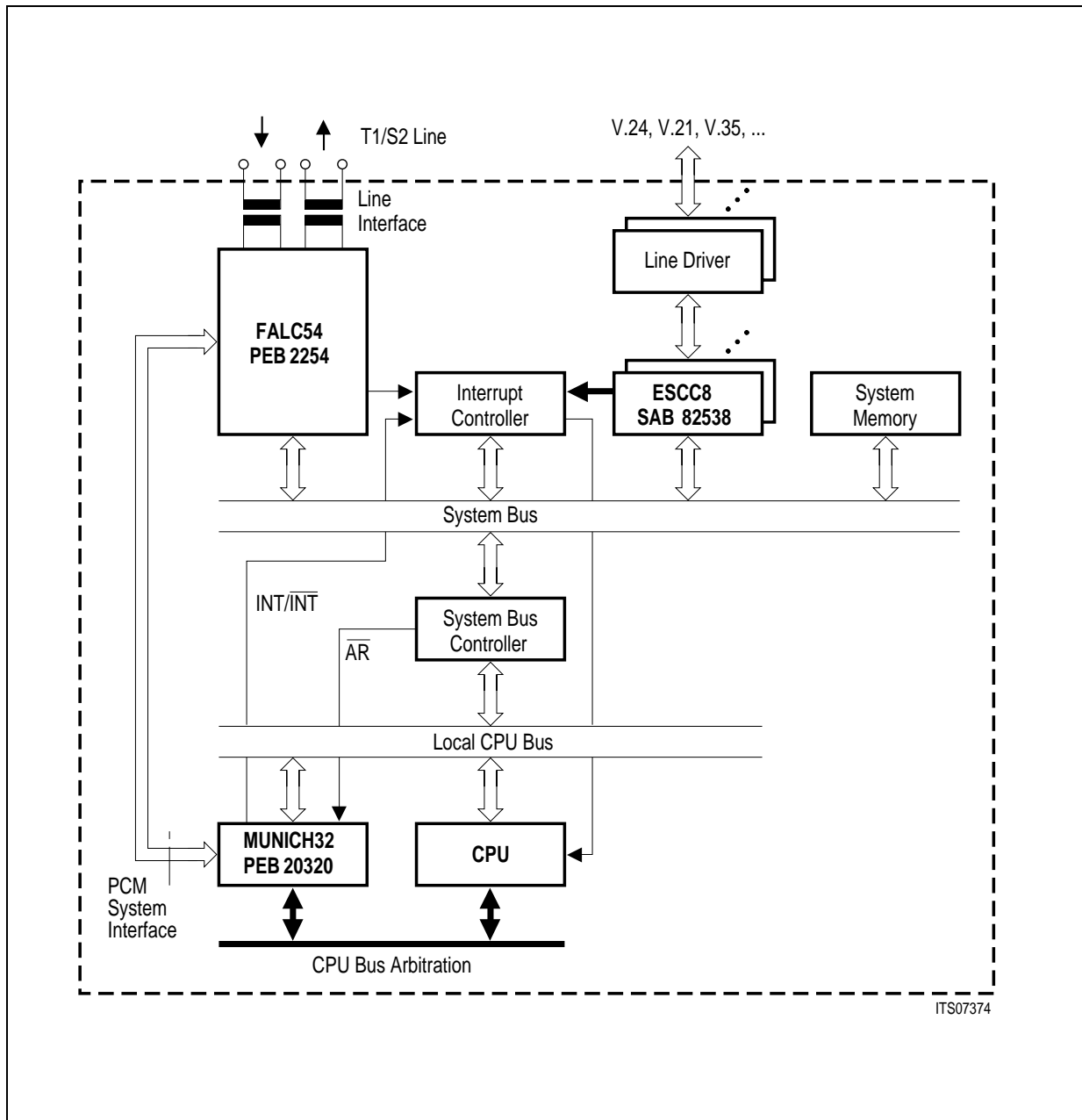
**Figure 7, Figure 8, Figure 9, Figure 10 and Figure 11** illustrate how the MUNICH32 may be used in different applications, like in a Primary Rate Interface, a Router, a Packet Switch and a Central D-Channel Handler, as part of an ISDN switching system.



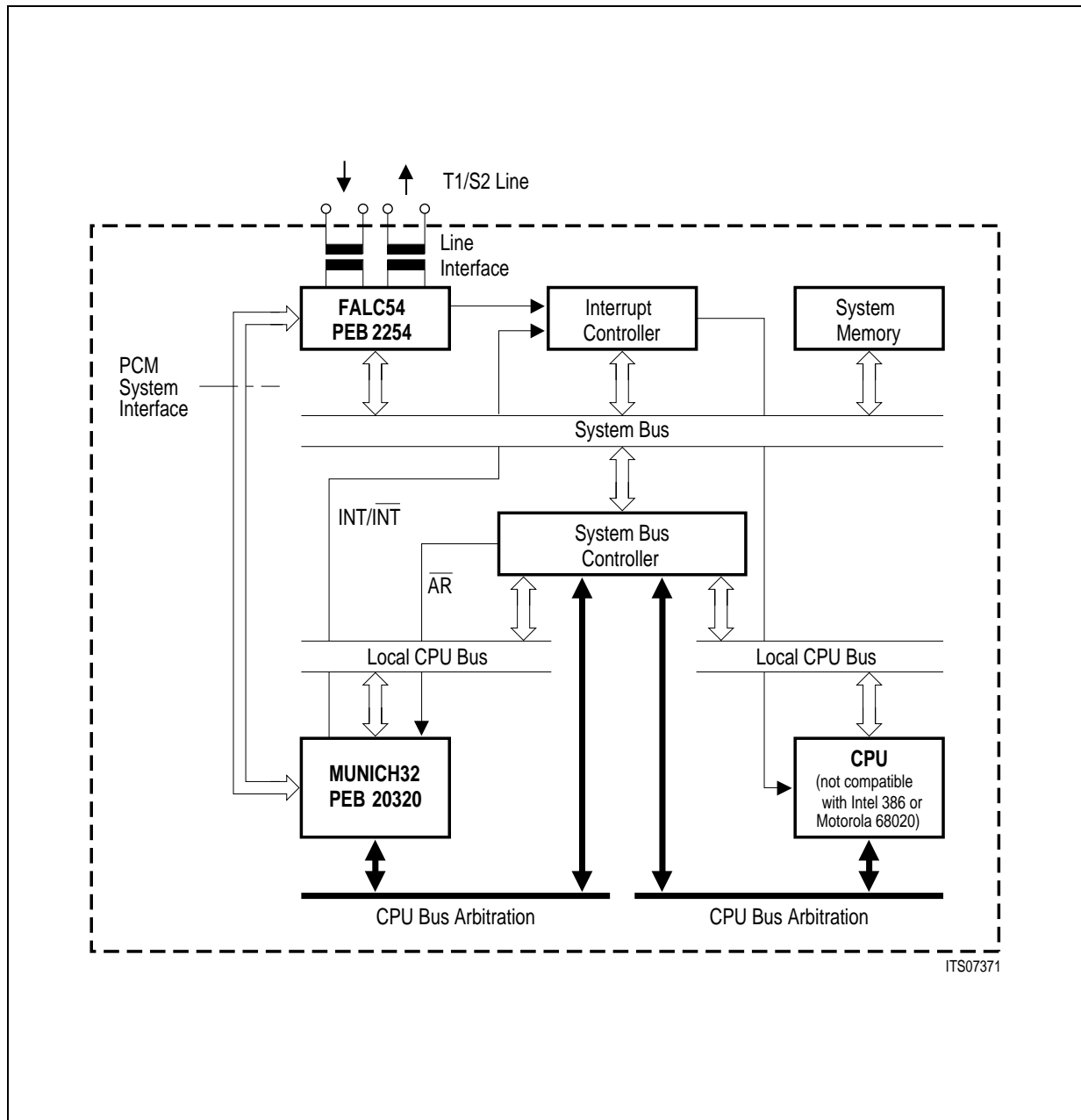
**Figure 7**  
**Architecture of a Primary Access Board**



**Figure 8**  
**Architecture of a Central D-Channel Handler**

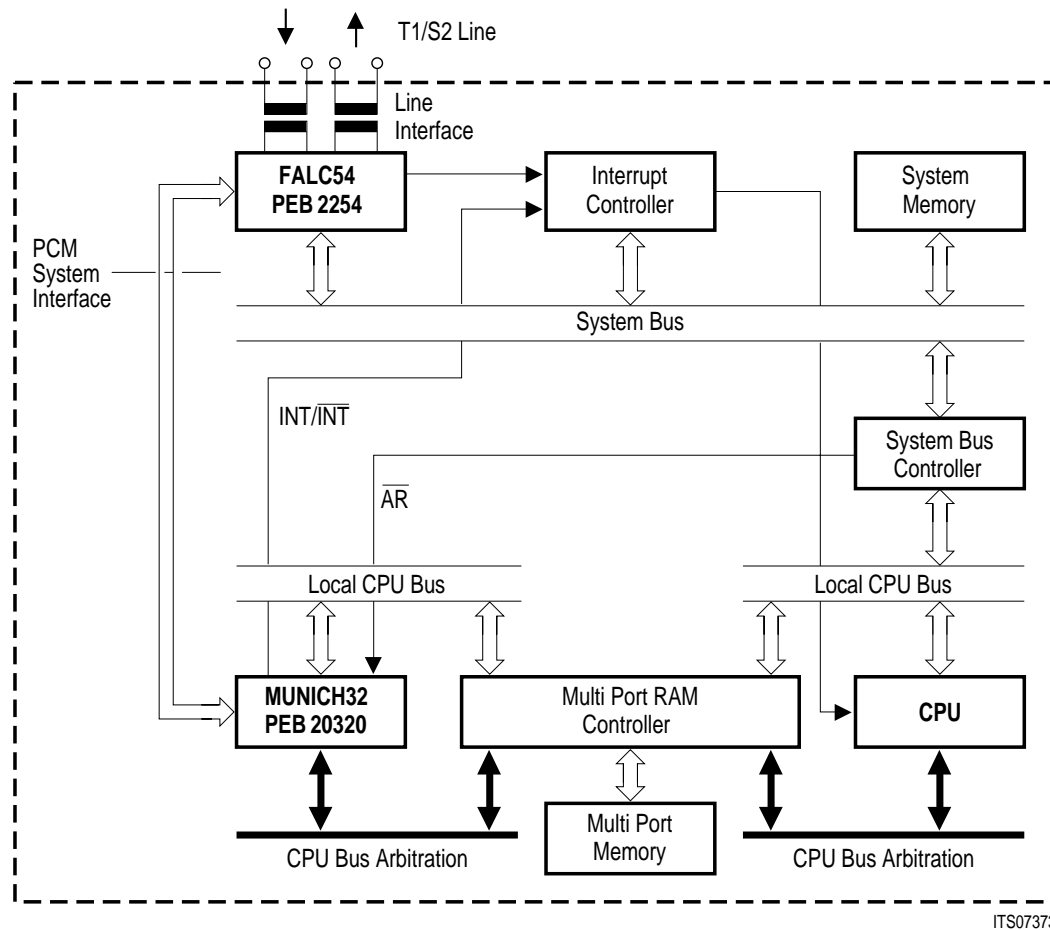


**Figure 9**  
**Architecture of a Packet Switch/Router**



**Figure 10**  
**MUNICH32 in a System with a RISC CPU**

*Note: To reduce complexity the host interface is not explicitly shown here.*



**Figure 11**  
**MUNICH32 in a System using Multiport Memory**

*Note: To reduce complexity the host interface is not explicitly shown here.*

## 2 Functional Description

### 2.1 Serial Interface

The serial interface of MUNICH32 includes a data receive (RDATA) and a data transmit line (TDATA) as well as the accompanying control signals (RCLK = Receive Clock, RSP = Receive Synchronization Pulse, TCLK = Transmit Clock, TSP = Transmit Synchronization Pulse). The timings of the receive and transmit PCM highway are independent of each other, i.e. the frame positions and clock phases are not correlated. Data is transmitted and received either at a rate of 2.048 Mbit/s for the CEPT 32-Channel European PCM format (**Figure 14**) or 1.544 Mbit/s or 1.536 Mbit/s for the T1/DS1 24-Channel American PCM format (**Figure 12** and **Figure 13**). MUNICH32 may also be connected to a 4.096-Mbit/s PCM system (**Figure 15**), where it handles either the even- or odd-numbered time slots, so all 64 time slots can be covered by connecting two MUNICH32s to the PCM highway.

The actual bit rate of a time slot can be varied from 64 Kbit/s down to 8 Kbit/s for the receive and transmit direction. A fill mask code specified in the time slot assignment determines the bit rate and which bits of a time slot should be ignored. Any of these time slots can be combined to a data channel allowing transmission rates from 8 Kbit/s up to 2.048 Mbit/s.

The frame alignment is established by the transmit and receive synchronization pulse (TSP, RSP), respectively. The sampled rising edge of TSP identifies the current bit on the serial line (TDATA) as the last bit of a PCM frame. The sampled rising edge of RSP indicates that the current bit on the serial line (RDATA) is the first bit of a PCM frame.

The F-bit for the 1.544 MHz T1/DS1 24-channel PCM format is ignored in receive direction, the corresponding bit is tristate in transmit direction. It is therefore assumed that this channel is handled by a different device.

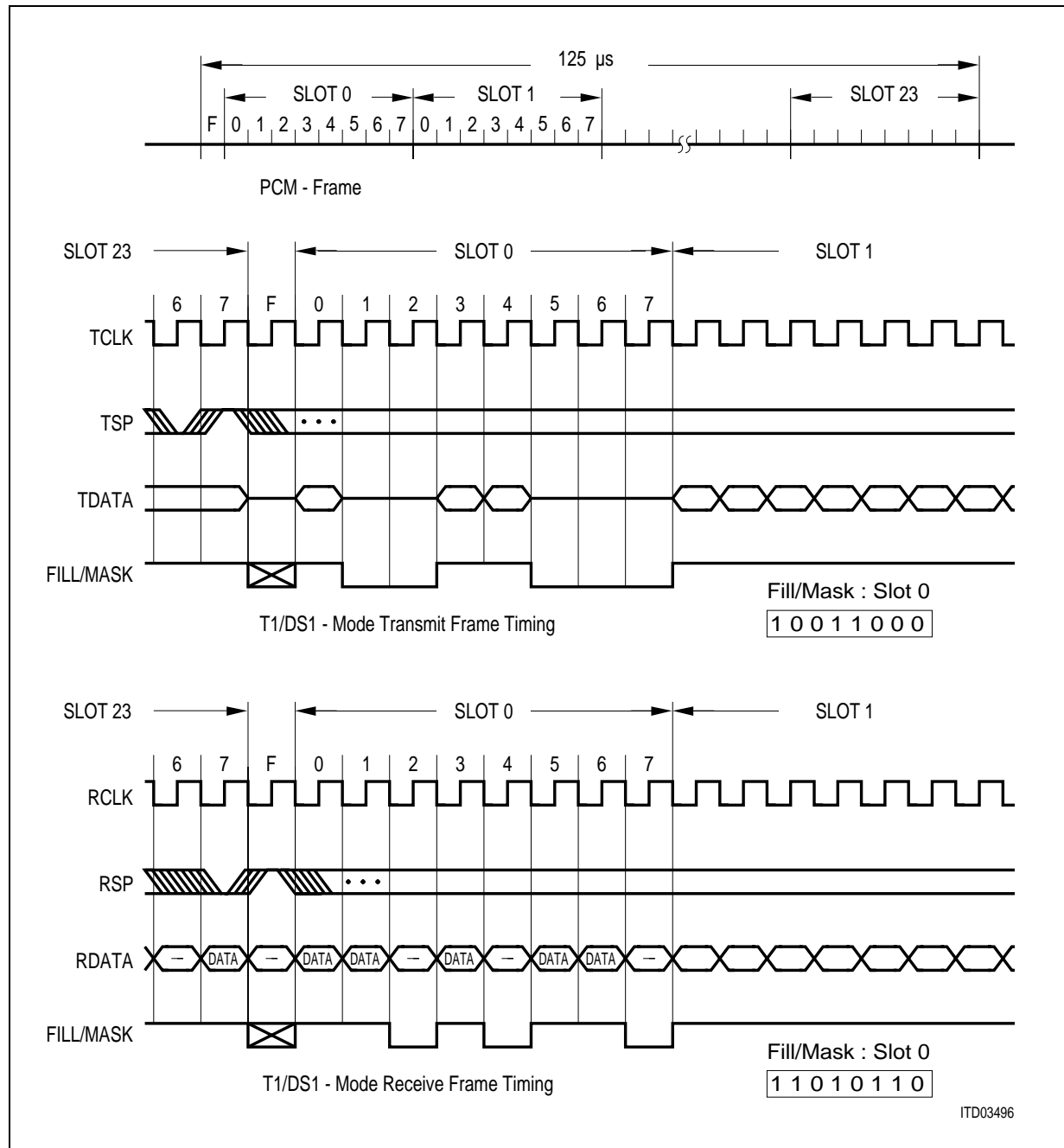
For test purposes four different test loops can be switched. In a complete loop all logical channels are mirrored either from serial data output to input (internal loop) or vice versa (external loop).

In a channelwise loop one single logical channel is logically mirrored either from serial data output to input (internal loop) or vice versa (external loop).


A detailed description of the different loops is found in **Chapter 4.2.1** and **Chapter 5.1**.



## Functional Description



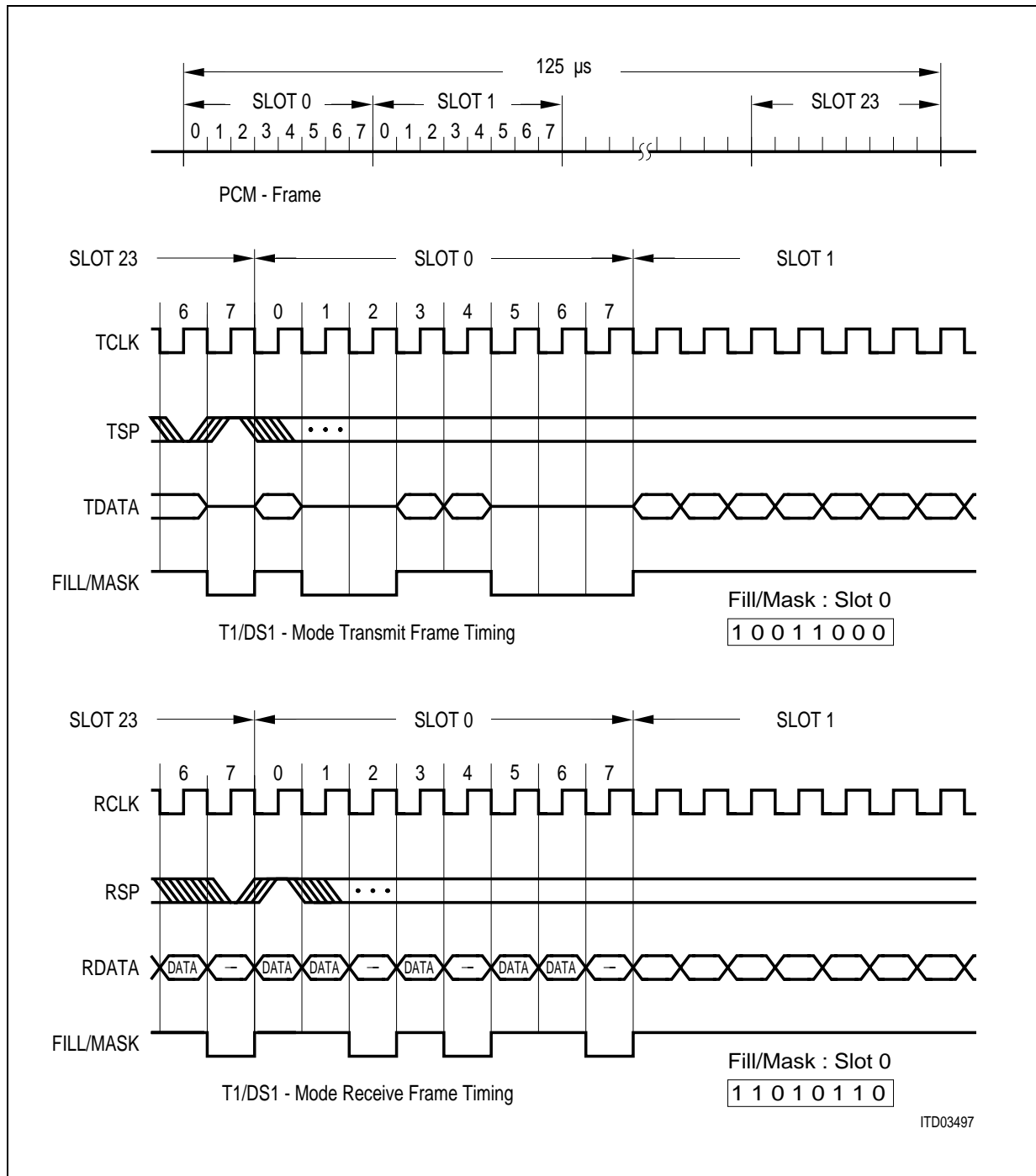
**Figure 12**  
**T1/DS1 Mode PCM Frame Timing 1.544 MHz**

**Note 1:** A  box in a bit of the RDATA line means that this bit is ignored (HDLC, TMB, TMR, V.110/X.30) or received as '1'-bit (TMA; one overwrite).


**Note 2:** The fill/mask bit for the F-bit is not defined. TDATA is tristate for the F-bit, and the F-bit is ignored in the receive direction.

**Note 3:** TSP and RSP must have one single rising and falling edge during a 125 μs PCM frame.

## Functional Description

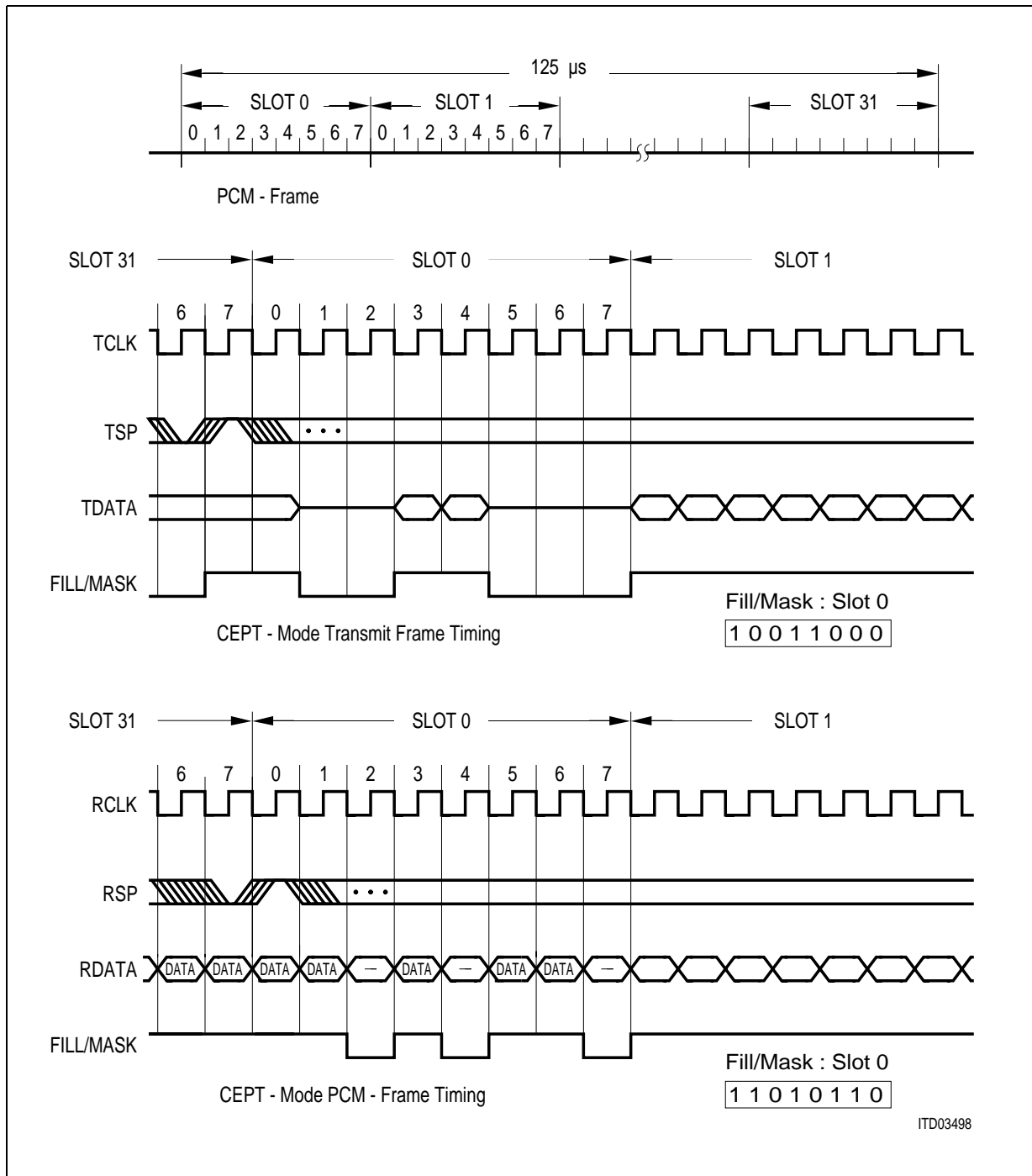


**Figure 13**  
**T1/DS1 Mode PCM Frame Timing 1.536 MHz**


**Note 1:** A  box in a bit of the RDATA line means that this bit is ignored (HDLC, TMB, TMR, V.110/X.30) or received as '1'-bit (TMA; one overwrite).

**Note 2:** TSP and RSP must have one single rising and falling edge during a 125 µs PCM frame.

## Functional Description

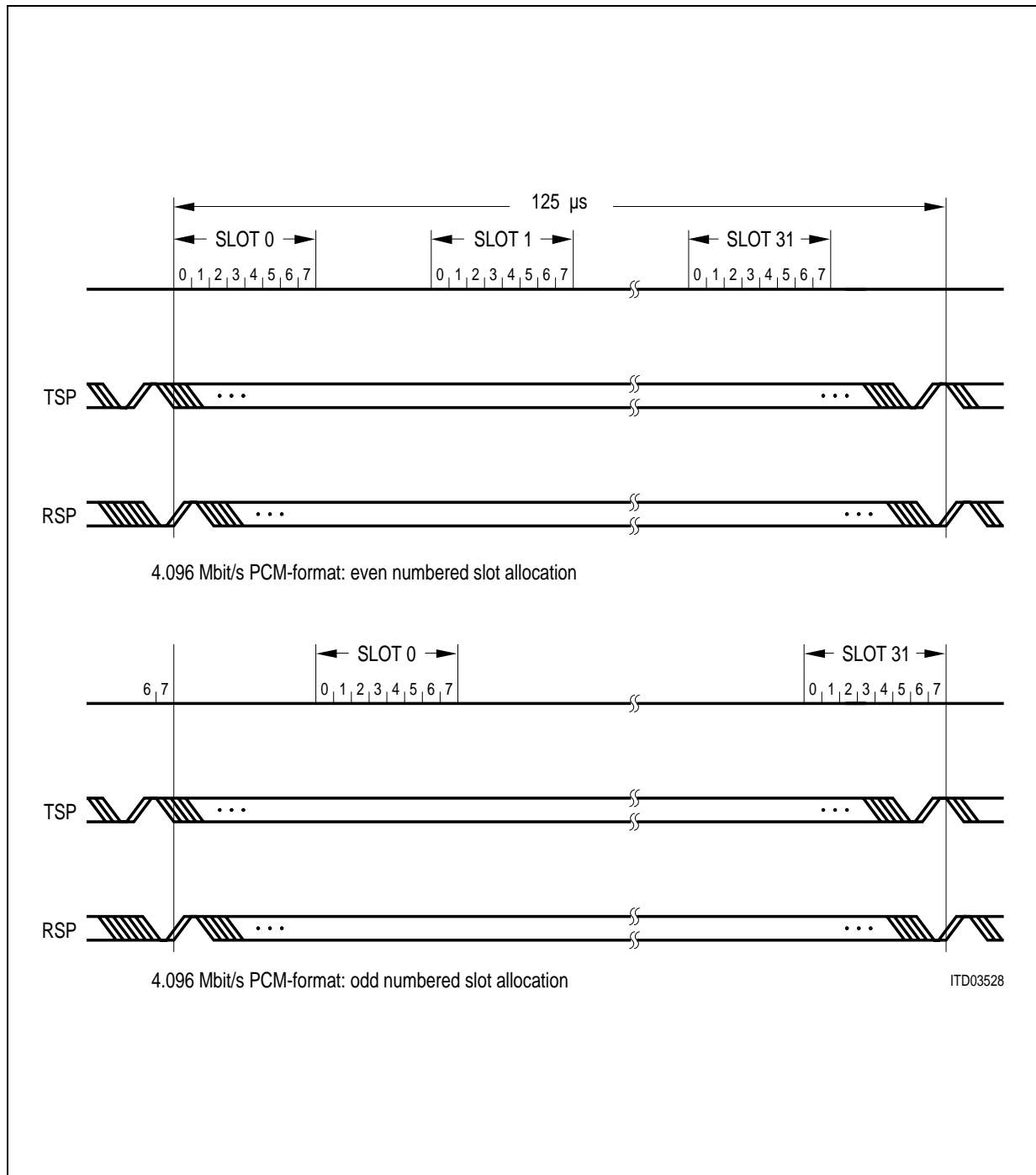


**Figure 14**  
**CEPT Mode PCM Frame Timing**

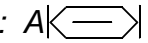
Note 1: A  box in a bit of the RDATA line means that this bit is ignored (HDLC, TMB, TMR, V.110/X.30) or received as '1'-bit (TMA; one overwrite).

Note 2: TSP and RSP must have one single rising and falling edge during a 125 μs PCM frame.

## Functional Description

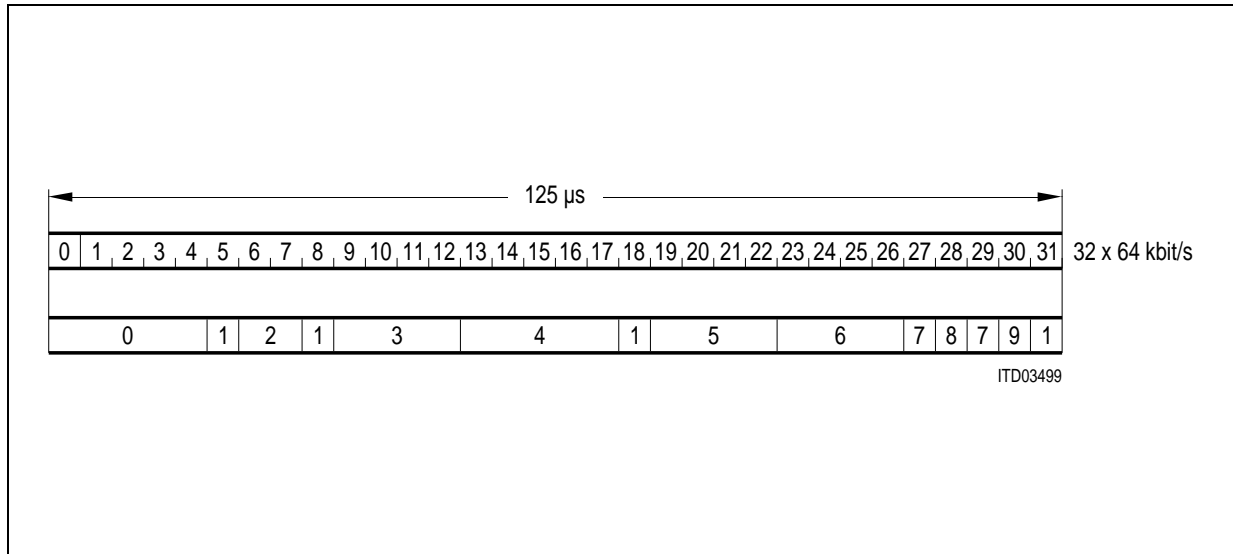


**Figure 15**  
**4.096 Mbit/s PCM Frame Timing**

Note 1: A  box in a bit of the RDATA line means that this bit is ignored (HDLC, TMB, TMR, V.110/X.30) or received as '1'-bit (TMA; one overwrite).

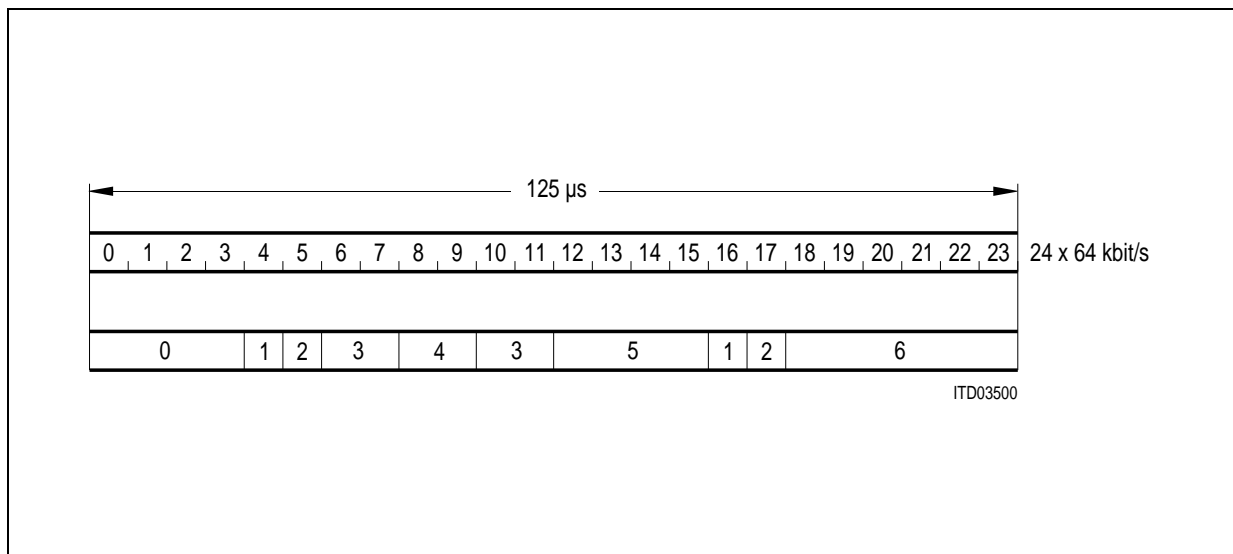
Note 2: TSP and RSP must have one single rising and falling edge during a 125 μs PCM frame.

## Functional Description



**Figure 16**

**Example: Programmable Channel Allocation for 32 Time Slots**



**Figure 17**

**Example: Programmable Channel Allocation for 24 Time Slots**

## Functional Description

### 2.2 Microprocessor Interface

A 64-channel DMA controller (32 channels in receive direction and 32 channels in transmit direction) with buffer chaining capability is integrated in the MUNICH32. It provides DMA functions for up to 32 full duplex channels and allows data transfer between the serial interface and an external memory. The MUNICH32 performs long word by long word transfers on a 32-bit bidirectional data bus (D(31:0)) and addresses up to 4 GByte of RAM with a 30-bit address bus (A(31:2)). The chip always works as a system bus master and can be operated in either a Intel or Motorola environment. MUNICH32 receives commands and data from the host processor via the shared memory. The host stores the action specification containing configuration initialization and monitor commands in the memory. Afterwards the host informs the MUNICH32 by generating an action request pulse ( $\overline{AR}$  line). The MUNICH32 reacts by reading the action specification and informs the microprocessor by appending the respective interrupt information to the interrupt queue. In addition, the INT/ $\overline{INT}$  line is activated during the write access belonging to the interrupt specification.

The timing of the microprocessor interface is established according to the Intel 80386 or Motorola 68020 processor. The system clock (SCLK) provides the fundamental timing for the  $\mu$ P interface and is the internal device clock. Each bus cycle performs a long word (B16 = 1) or a word (B16 = 0) transfer and takes four system clock periods in the fastest case, any number of wait clock cycles can be inserted.

MUNICH32's architecture is based on a 32-bit data structure. Therefore MUNICH32 performs long word operations preferably. While the word operation mode is selected the long word operation is divided into two consecutive word operations. In the case of a read access the data of the two words are connected together to build a 32-bit long word before processing.

Mode	Operation Mode B16	BE(3–0)	Access
Intel	1	0 <sub>H</sub>	long word
	0	3 <sub>H</sub>	MSB word
	0	C <sub>H</sub>	LSB word
Motorola	1	0 <sub>H</sub>	long word
	0	8 <sub>H</sub>	MSB word
	0	A <sub>H</sub>	LSB word

For a read access first the MSB bytes of a long word will be transferred and then the LSB bytes via D(15:0).

For a write access first the LSB-bytes of a long word will be transferred and then the MSB bytes via D(15:0).

The signal B16 cannot be changed dynamically and should be set to '1' in Intel parity mode (parity mode is not available in 16-bit word Intel mode).

## Functional Description

### 2.2.1 Intel Mode

The Intel mode has two submodes – parity mode (even parity) and non parity mode – to be chosen by strapping PM to ‘1’ or ‘0’ respectively.

In Intel mode the lower (higher) ordered byte of a long word (D31 ... D0) is assigned to the lower (higher) ordered physical address.

The read or write bus cycle is controlled by the signals  $\overline{W/R}$ ,  $\overline{ADS}$  and  $\overline{READY}$  as shown in **Figure 18**, **Figure 19**. Each bus cycle consists of two bus states (S1, S2). During state S1 the address signals and bus cycle definition signals are driven valid. Simultaneously, the address status  $\overline{ADS}$  is asserted to indicate their availability. The bus cycles are terminated by asserting  $\overline{READY}$ .  $\overline{READY}$  is ignored on the first bus state S1 and sampled at the end of the following state S2. If  $\overline{READY}$  is not asserted in S2 then wait cycles SW are inserted until a bus cycle end is detected. During a read cycle the MUNICH32 floats its data signals to allow external memory to drive the data bus.

The input data and parity bits DP3–0 (if parity mode is selected) is latched when  $\overline{READY}$  is asserted. During a write cycle MUNICH32 drives the data signals and parity bits DP3–0 (if parity mode is selected) beginning in the second clock period of S1 until the first clock period following the cycle acknowledgment  $\overline{READY}$ . If a bus cycle error indicated by  $\overline{BERR}$  has occurred, the MUNICH32 terminates the bus cycle. In case of a read cycle in the control and configuration section an action request fail interrupt is generated and the action is suspended. In case of a read cycle in the transmit data section the corresponding frame is aborted and a FO interrupt is generated. In all other cases of read or write cycles terminated with an error condition no actions are performed.

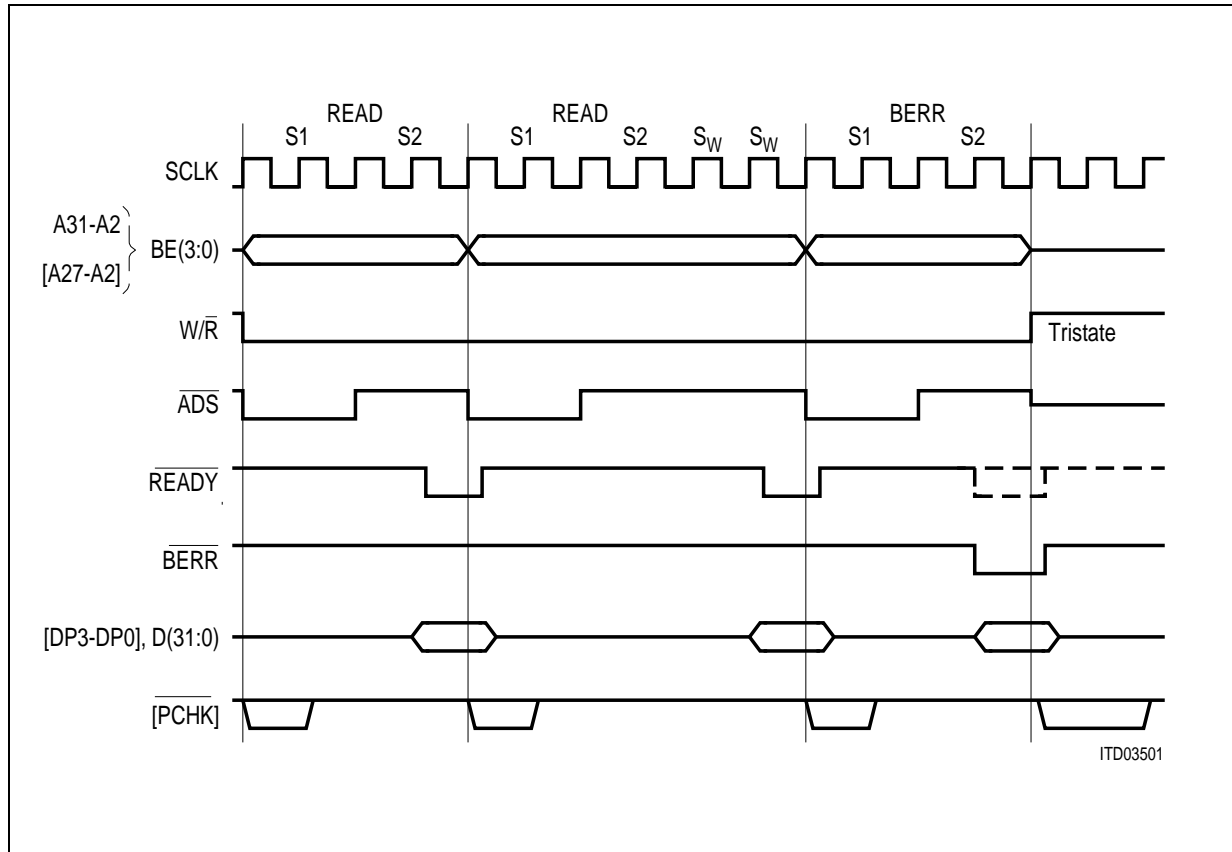
A 4-bit data byte parity bus DP3–0 is used in Intel mode if parity mode is selected by strapping PM to ‘1’. During a read access DP3–0 is supposed to contain the parity of D(31:24), D(23:16), D(15:8) and D(7:0) respectively. A low active output  $\overline{PCHK}$  indicates whether the parity was correct ( $\overline{PCHK} = 1$ ) or wrong ( $\overline{PCHK} = 0$ ) in the clock cycle after the data/parity is latched.  $\overline{PCHK}$  stays low 1 or 2 clock cycles. No further action is taken as consequence to a parity fail.

As the memory access is performed by using one common system bus, bus management is done with the signals HOLD, HLDA and HLDAO as shown in **Figure 20**.

The wired or HOLD line is driven high whenever one of the MUNICH32s has to perform a bus transfer. The activated HOLD ACKNOWLEDGE indicates that the bus control will be released. If the specific device has activated the HOLD itself, it will start the memory access. Otherwise it will pass the signal to the next cascaded device. Several memory accesses may be required if the MUNICH32 has not been granted access recently. In this example of four MUNICH32 devices sharing the same bus, each device will generate four memory cycles, giving a total of 16 cycles per HOLD/HLDA/HLDAO tenure. In order to prevent blocking in the case of continuous request by one device, the MUNICH32 does not generate another HOLD REQUEST before the HOLD ACKNOWLEDGE has been deactivated.

## Functional Description

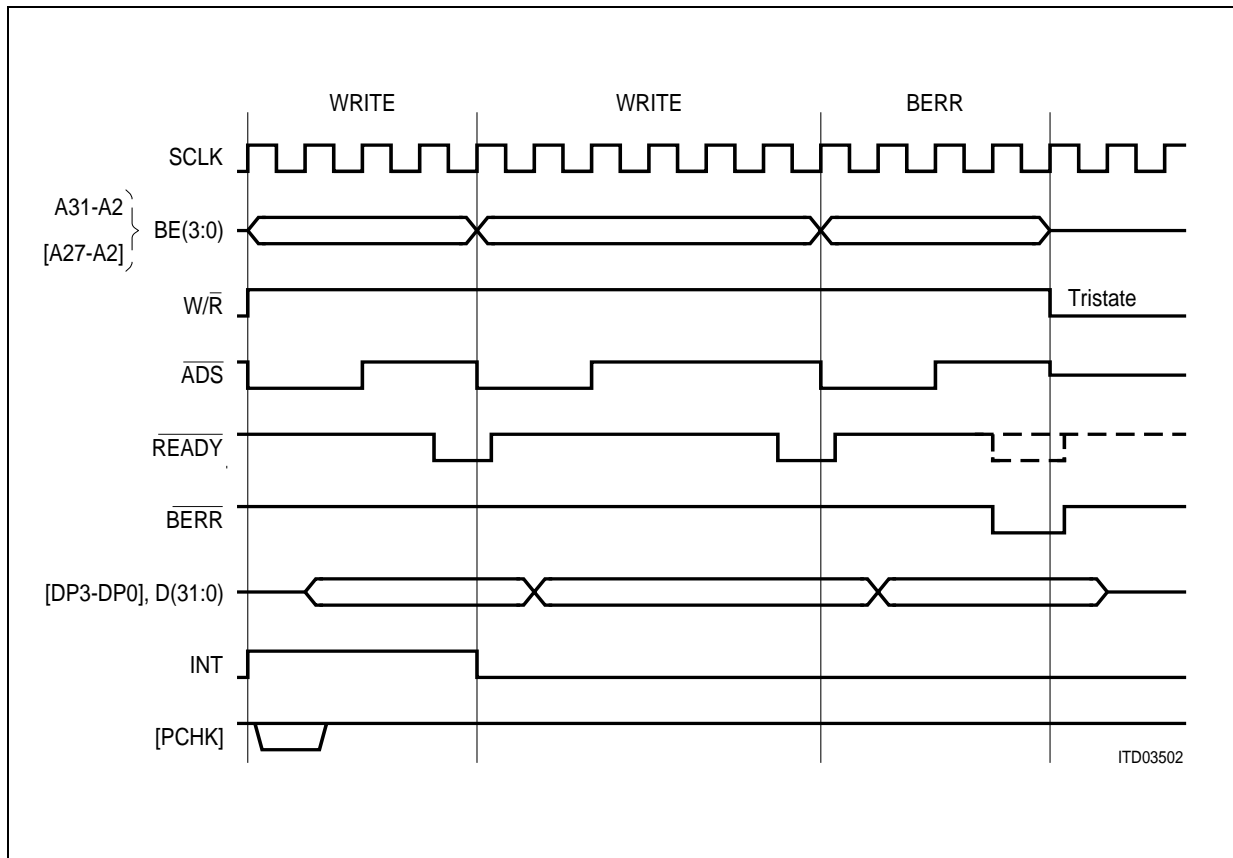
If the HOLD ACKNOWLEDGE is driven low while the MUNICH32 is performing a bus cycle, the bus is released later than two clock periods after de-assertion of HOLD ACKNOWLEDGE. The current bus cycle is finished with a bus cycle error. This action should be followed by an ASP.RES as described in **Chapter 4.2.1**.



**Figure 18**  
**Read Cycle Timing Diagram (Intel mode)**

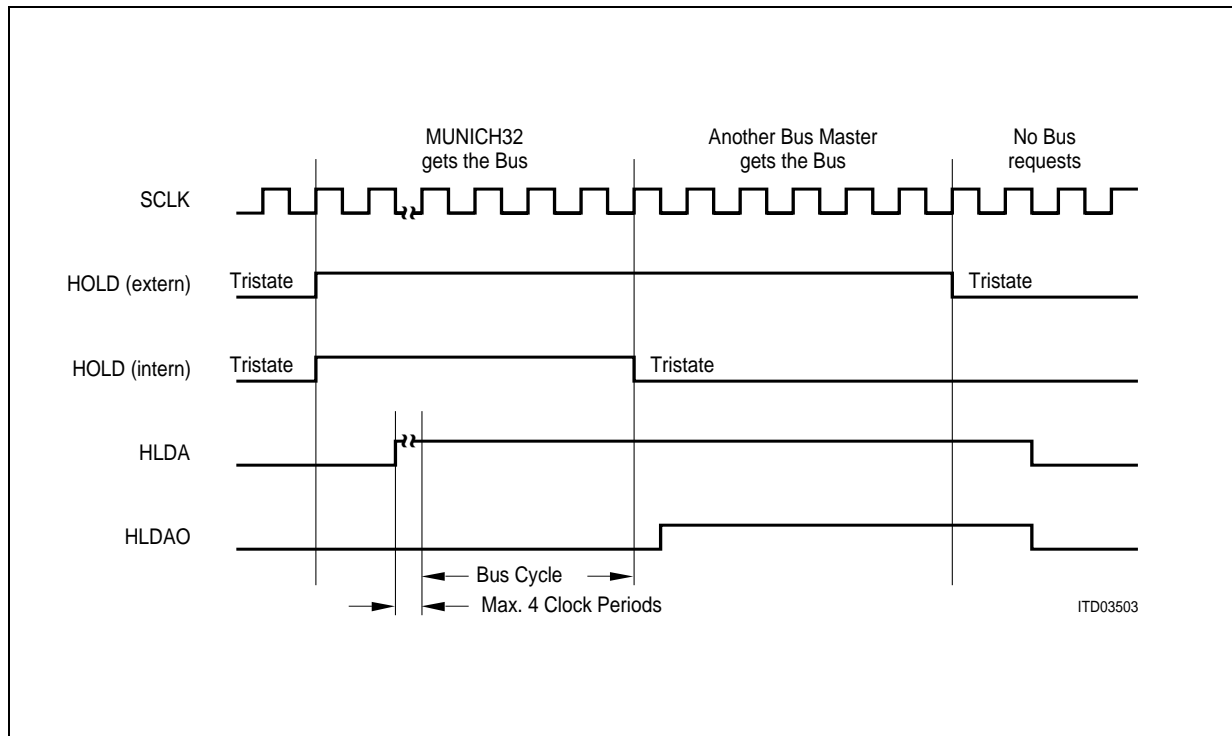


## Functional Description



**Figure 19**  
**Write Cycle Timing Diagram (Intel mode)**

## Functional Description



**Figure 20**  
**Bus Management for Intel Bus Mode**

- Note 1: Bus Cycle means, that the MUNICH32 under consideration starts a read or write access at most 4 clock periods after HLDA is asserted after its HOLD. The MUNICH32 terminates the cycle typically two clock periods after the last bus cycle.*
- Note 2: In the Bus Management example it is assumed that the MUNICH32 under consideration has a higher priority than the other bus master. HOLD (internal) is therefore the internal request generated by the MUNICH32, HOLD (external) the signal on the external HOLD line, being the OR combination of the HOLD signal generated by the MUNICH32 and the other bus master(s).*
- Note 3: A typical configuration example for a system with several bus masters is given in **Figure 4** and **Figure 5**.*

## Functional Description

### 2.2.2 Motorola Mode

In Motorola mode the bus is used in an asynchronous manner. The bus operation uses the handshake lines ( $\overline{AS}$ ,  $\overline{DS}$ ,  $\overline{DSACK}$  and  $\overline{BERR}$ ) to control data transfer as shown in **Figure 21**, **Figure 22**. Address strobe  $\overline{AS}$  indicates the validity of an address on the address bus (A31 ... A2) and of the bus definition R/W (Read or Write cycle). It is asserted half a clock cycle after the beginning of a bus cycle. The data strobe  $\overline{DS}$  signal is used as a condition for valid data of a write cycle. MUNICH32 asserts  $\overline{DS}$  one full clock cycle after the assertion of  $\overline{AS}$  during a write cycle. The data is placed on the bidirectional data bus (D31 ... D0) half a clock cycle after  $\overline{AS}$  is driven low. For a read cycle, MUNICH32 asserts  $\overline{DS}$  to signal the external memory to drive the data on the bus.  $\overline{DS}$  is asserted at the same time as  $\overline{AS}$  during a read cycle. The data is latched with the last falling edge of the clock for that cycle.

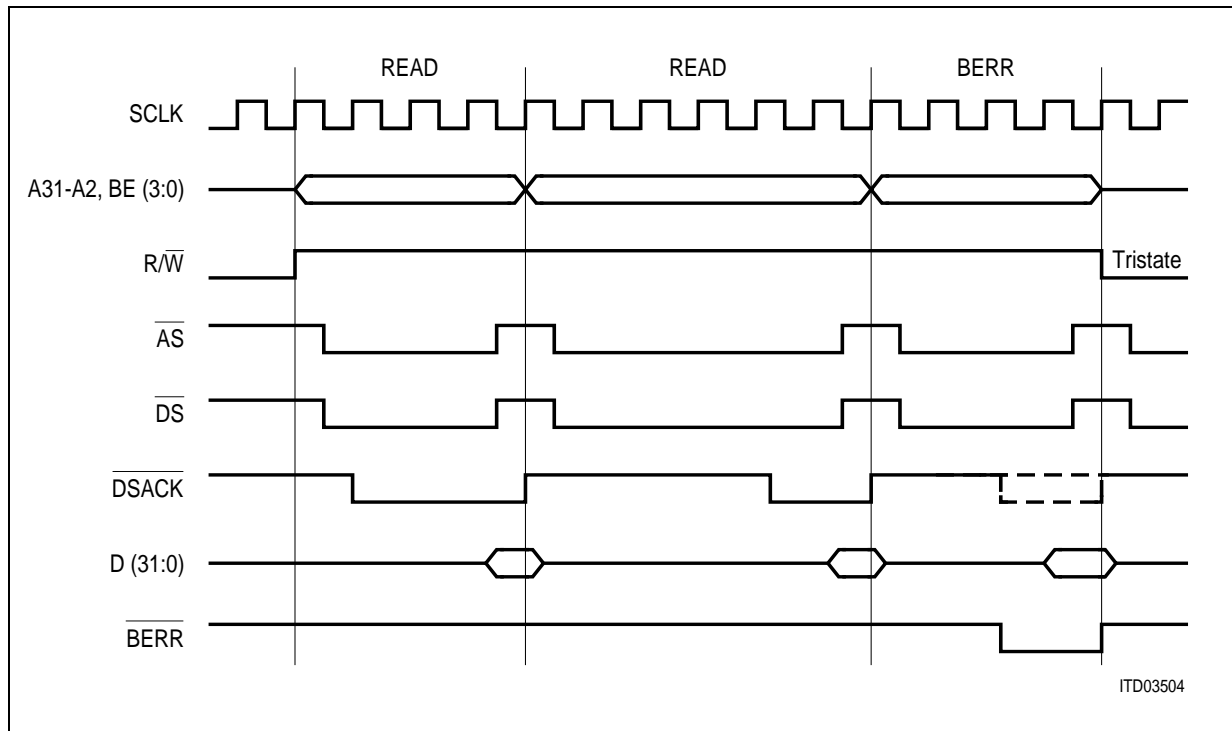
The bus cycle is terminated if the data transfer acknowledge ( $\overline{DSACK}$ ) is asserted with the falling edge of the third clock period. Otherwise MUNICH32 inserts wait cycles until  $\overline{DSACK}$  is recognized.  $\overline{AS}$  and  $\overline{DS}$  are driven high half a clock period before bus cycle end.

The bus error  $\overline{BERR}$  is also a bus cycle termination indicator. It can be used in the absence as well as in conjunction with  $\overline{DSACK}$ . If an abnormal termination has occurred during a read cycle, MUNICH32 generates an interrupt and aborts the corresponding transmit channel. For a write cycle no further action is performed.

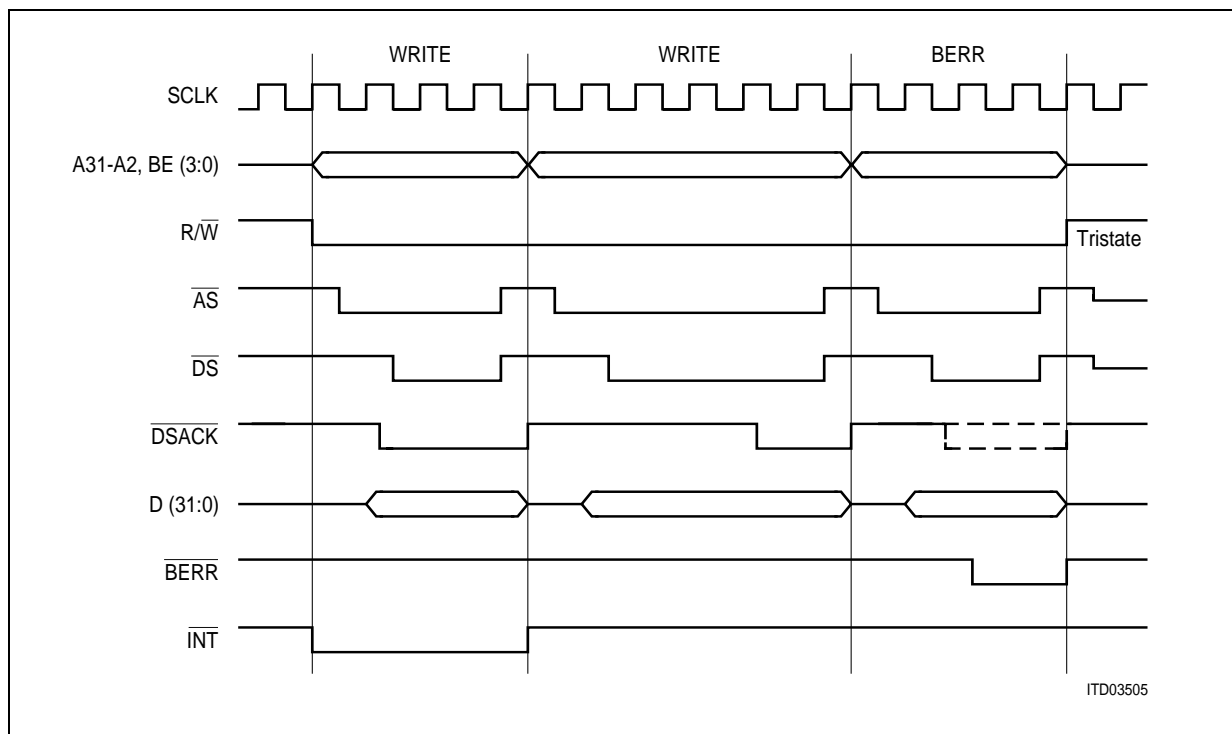
As the MUNICH32 is used in a multi-bus-master application, bus arbitration has to be done to avoid simultaneous system bus access by more than one master. In Motorola mode the bus arbitration protocol of the 68020 is established using the signals  $\overline{BR}$ ,  $\overline{BG}$ ,  $\overline{BGACK}$  and  $\overline{BGO}$  as shown in **Figure 23**. The wired-or Bus Request ( $\overline{BR}$ ) is driven low to indicate to the processor that one of the MUNICH32s requires control of the bus. The activated Bus Grant ( $\overline{BG}$ ) signals the availability of the system bus. If the MUNICH32 has activated the bus request itself, it asserts the wired-or Bus Grant Acknowledge to indicate that it has assumed bus mastership. Otherwise it will pass the BUS GRANT signal to the device cascaded next ( $\overline{BGO}$ ). At the same time it releases the Bus Request. After finishing the last bus cycle, the Bus Grant Acknowledge is deactivated and the Bus Grant is passed on. In order to prevent blocking in the case of continuous request by one device, MUNICH32 does not generate another Bus Request before the external Bus Request and Bus Grant Acknowledge have been deactivated.

After getting the bus mastership MUNICH32 drives the bus and starts the first bus cycle one clock after assertion of  $\overline{BGACK}$ . After finishing the memory access it releases the bus and de-asserts  $\overline{BGACK}$  at the same time.

## Functional Description

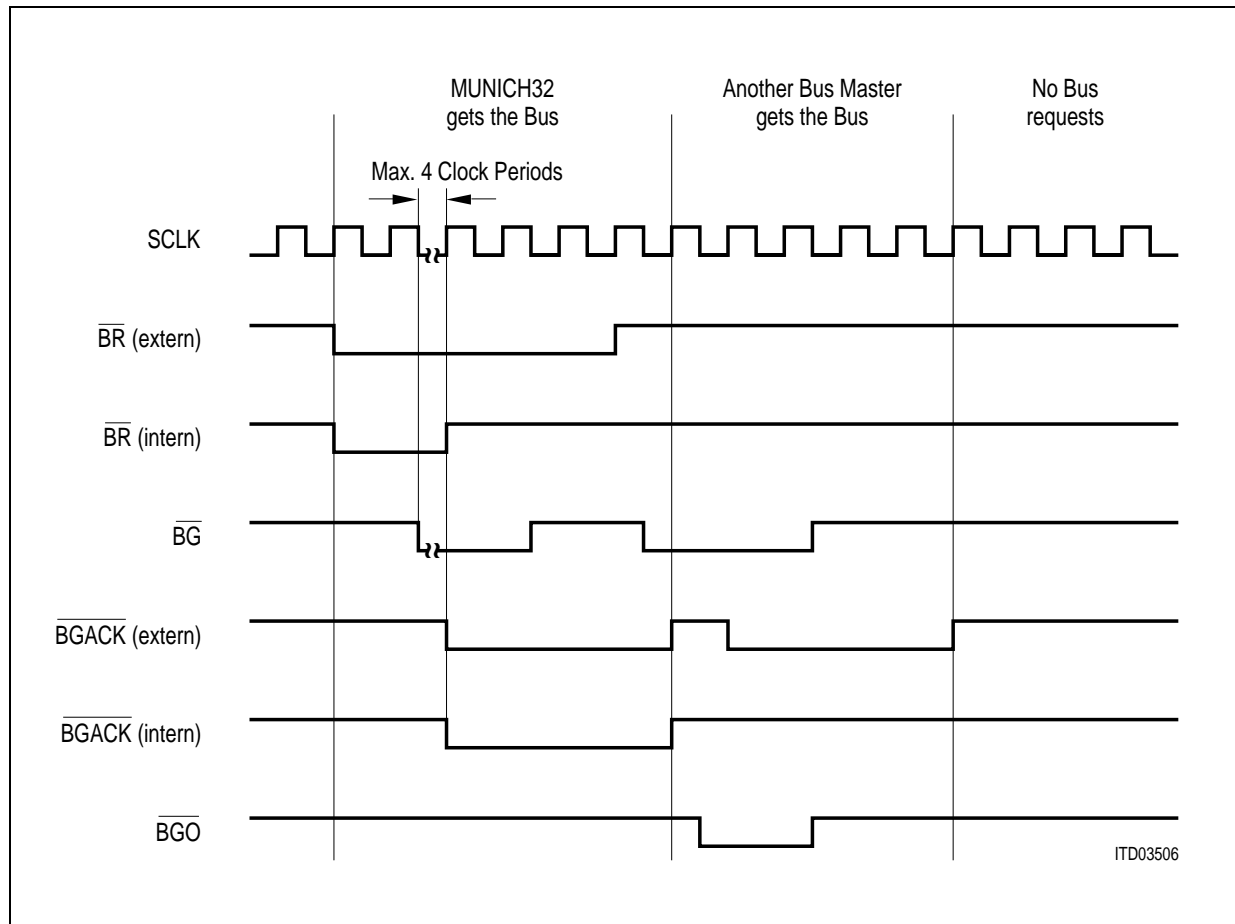


**Figure 21**  
**Read Bus Cycle Timing Diagram for Motorola Bus Mode**



**Figure 22**  
**Write Bus Cycle Timing Diagram for Motorola Bus Mode**

## Functional Description



### Figure 23 Bus Management for Motorola Mode

*Note: 1. In the Bus Management example it is assumed that the MUNICH32 under consideration has a higher priority than the other bus master. BR and BGACK are wired AND lines to be pulled to '1' by an external signal.*

2. A typical configuration example for a system with several bus masters is given in **Figure 6**.

---

**Functional Description****2.2.3 DMA Priorities****Prioritization of Queueing DMA Cycles**

Priority	Interrupt
Highest priority	Receive link list including accesses to the descriptors
	Transmit link list including accesses to the descriptors
Lowest priority	Configuration of a channel (action requests)

The MUNICH32 will perform all pending accesses on the same bus tenure.

*Note: Several bus transactions may be required if the MUNICH32 has not been given access to the system bus for a long period of time. This is often seen in multi-master systems where several MUNICH32 devices share the system bus.*

---

## Functional Description

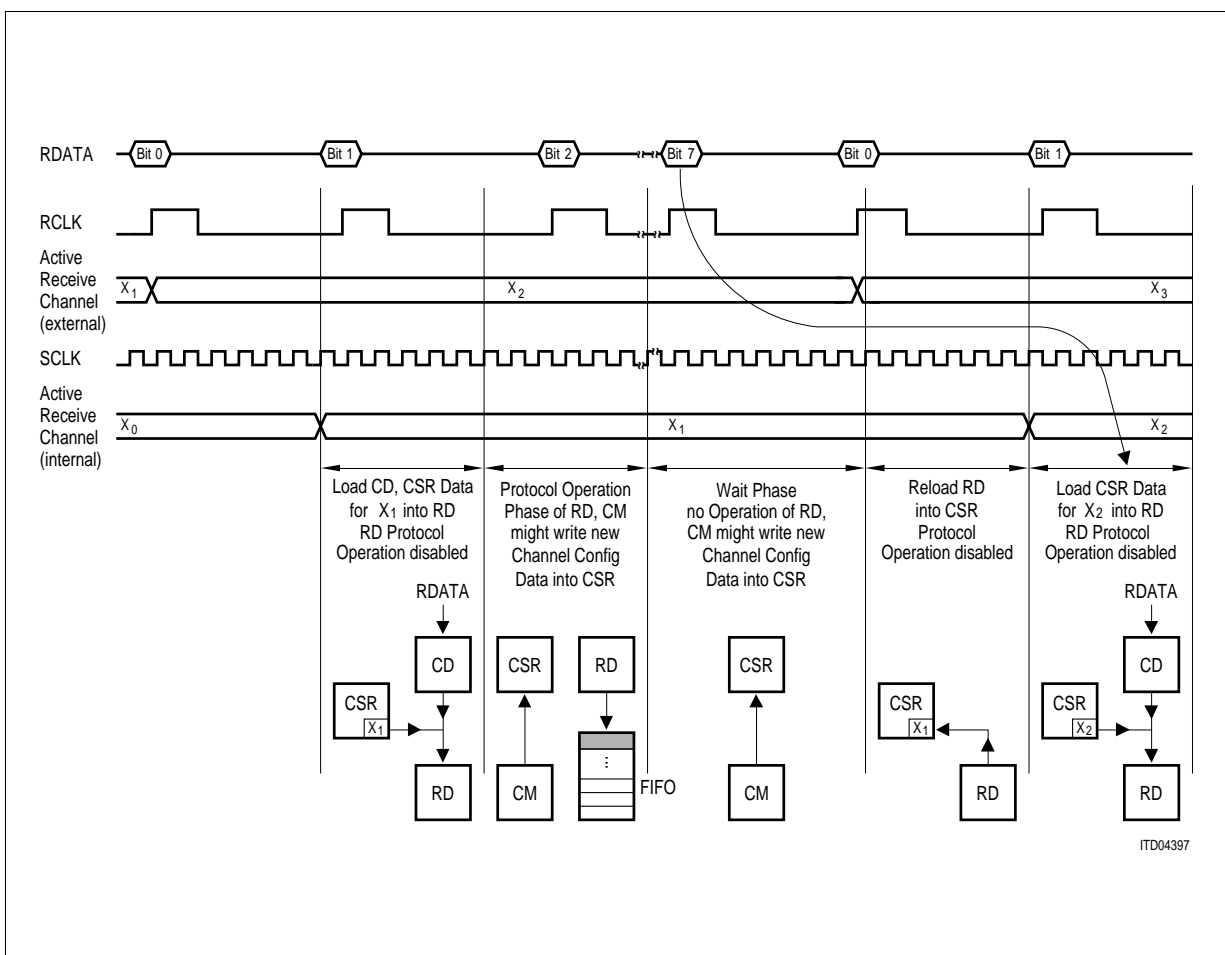
### 2.3 Basic Functional Principles

MUNICH32 is a Multichannel Network Interface Controller for HDLC, offering a variety of additional features like subchanneling, data channels comprising of one or more time slots, DMI 0, 1, 2 transparent or V.110/X.30 transmission and programmable rate adaption. MUNICH32 performs formatting and deformatting operations in any network configuration, where it implements, together with a microprocessor and a shared memory, the bit oriented part (flag, bit stuffing, CRC check) of the layer 2 (data link protocol level) functions of the OSI reference model.

The block diagram is shown in **Figure 3**. MUNICH32 is designed to handle up to 32 data channels of a 1.536/1.544 Mbit/s T1/DS1 24-channel, 2.048-Mbit/s CEPT 32-channel or a 4.096-Mbit/s 32-channel PCM highway. The device provides transmission for all bit rates from 8 Kbit/s up to 2.048 Mbit/s of packed data in HDLC format or of data in a transparent format supporting the DMI mode (0, 1, 2) or V.110/X.30 mode. Tristating of the transmission line as well as switching a channelwise or complete loop are also possible. An on-chip 64-channel DMA generator controls the exchange of data and channel control information between the MUNICH32 and the external memory.

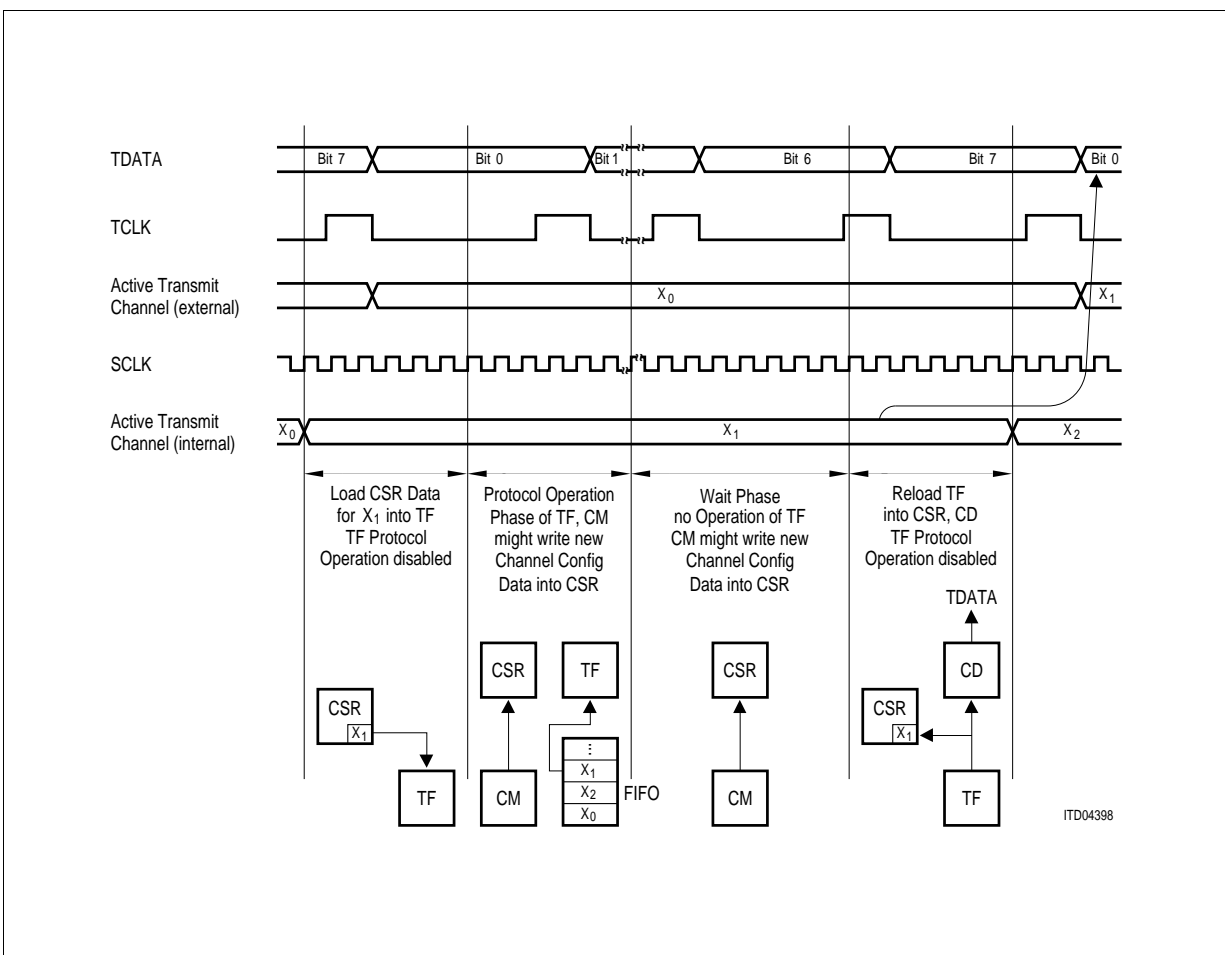
The MUNICH32 processes receive and transmit data independently for each time slot and transmission direction respectively (blocks TF = Transmit Formatter, RD = Receive Deformatter). The frame counters are reset by the rising edges of the RSP or TSP line. The processing units TF and RD work with a multiplex management, i.e. there exists only one protocol handler, which is used by all channels in a time sharing manner (see **Figure 24** and **Figure 25**). The actual configuration, e.g. transmission mode, channel assignment, fill/mask code or state of the protocol handlers is retrieved from the Configuration and State RAM (CSR) at the beginning of the time slot and reloaded to the CSR at the end. The control unit (CD) controls the access to the CSR and allows writing of reconfiguration information only if the continuous transfer of the configuration information between the CSR and the formatters (TF and RD) will not be disturbed. In receive direction, 32 unpacked data bits are first accumulated and then stored into an on-chip receive buffer (RB) for transfer to the shared memory. As soon as the RB receives 32 bits for a channel it requests access to the parallel microprocessor bus. The on-chip transmit buffer (TB) is always kept full of data ready for transmission. The TB will request more data when 32 bits become available in the ITBS. These buffers allow a flexible access to the shared memory in order to prevent data underflow (Tx) and data overflow (Rx).

The transmit buffer (TB) has a size of 64 long words (= 256 bytes). In this buffer, data of 8 PCM frames can be stored for each data channel. In this case, there are max. 1 ms between access to the shared memory and data supply to the Transmit Formatter. In order to meet these requirements a variable and programmable part of the buffer (ITBS) must be allocated to each data channel (see **Figure 26**).



**Figure 24**  
**Multiplex Management Receive Direction**





ITD04398

**Figure 25**  
**Multiplex Management Transmit Direction**

## Functional Description

For example:

a) 2.048-Mbit/s PCM highway

$32 \times 64$ -Kbit/s data channels (8 bits are sent with each PCM frame). Two long words of the buffer are allocated to each data channel.

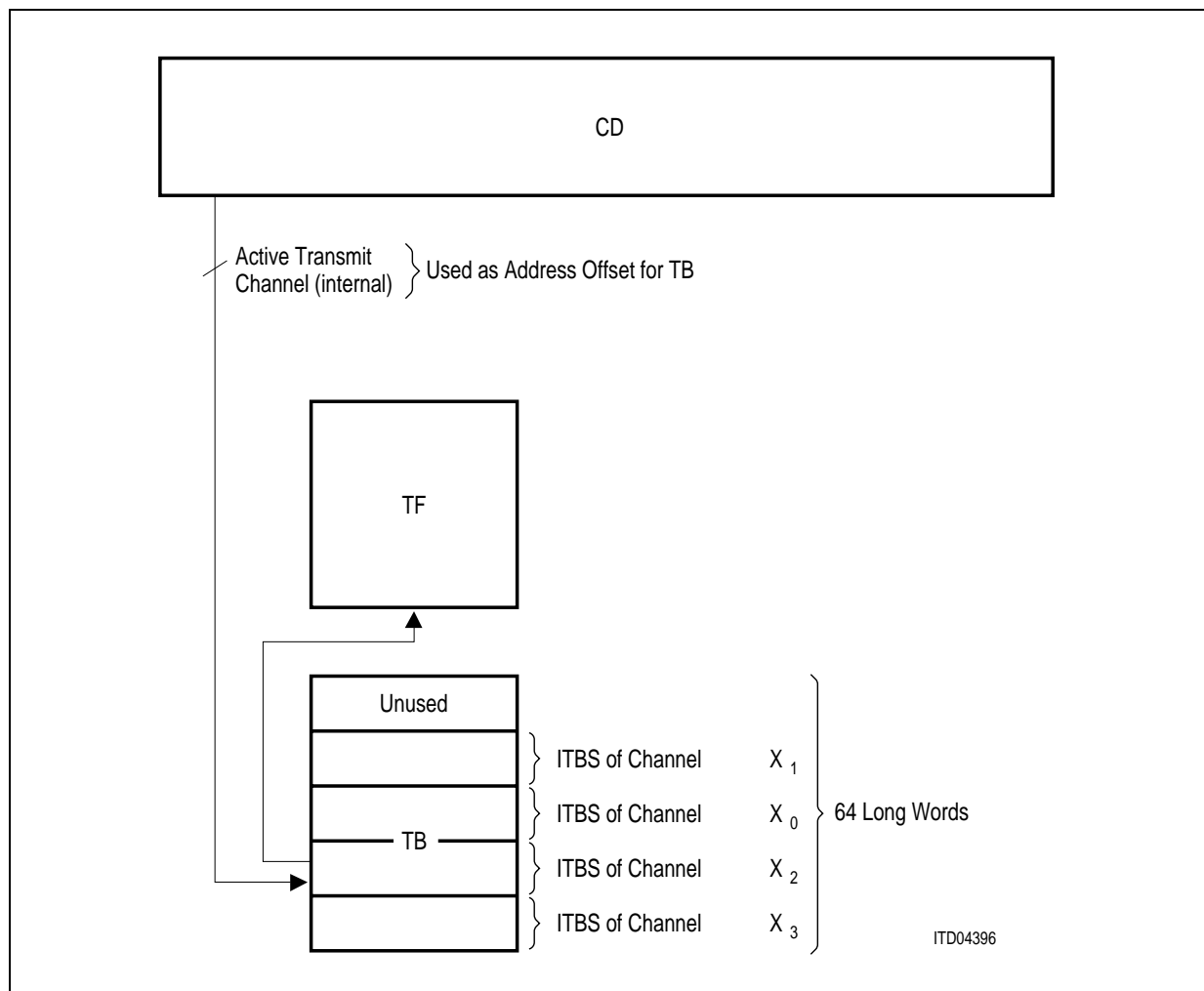
b)  $1 \times 2.048$ -Kbit/s data channel

The maximum buffer size for one channel (63 long words) is allocated to this data channel.

c)  $6 \times 256$ -Kbit/s and  $8 \times 64$  Kbit/s data channels.

Eight long words of the buffer are allocated to each of the 6 data channels with 256 Kbit/s and two long words are assigned to each of the 8 data channels with a transmission rate of 64 Kbit/s.

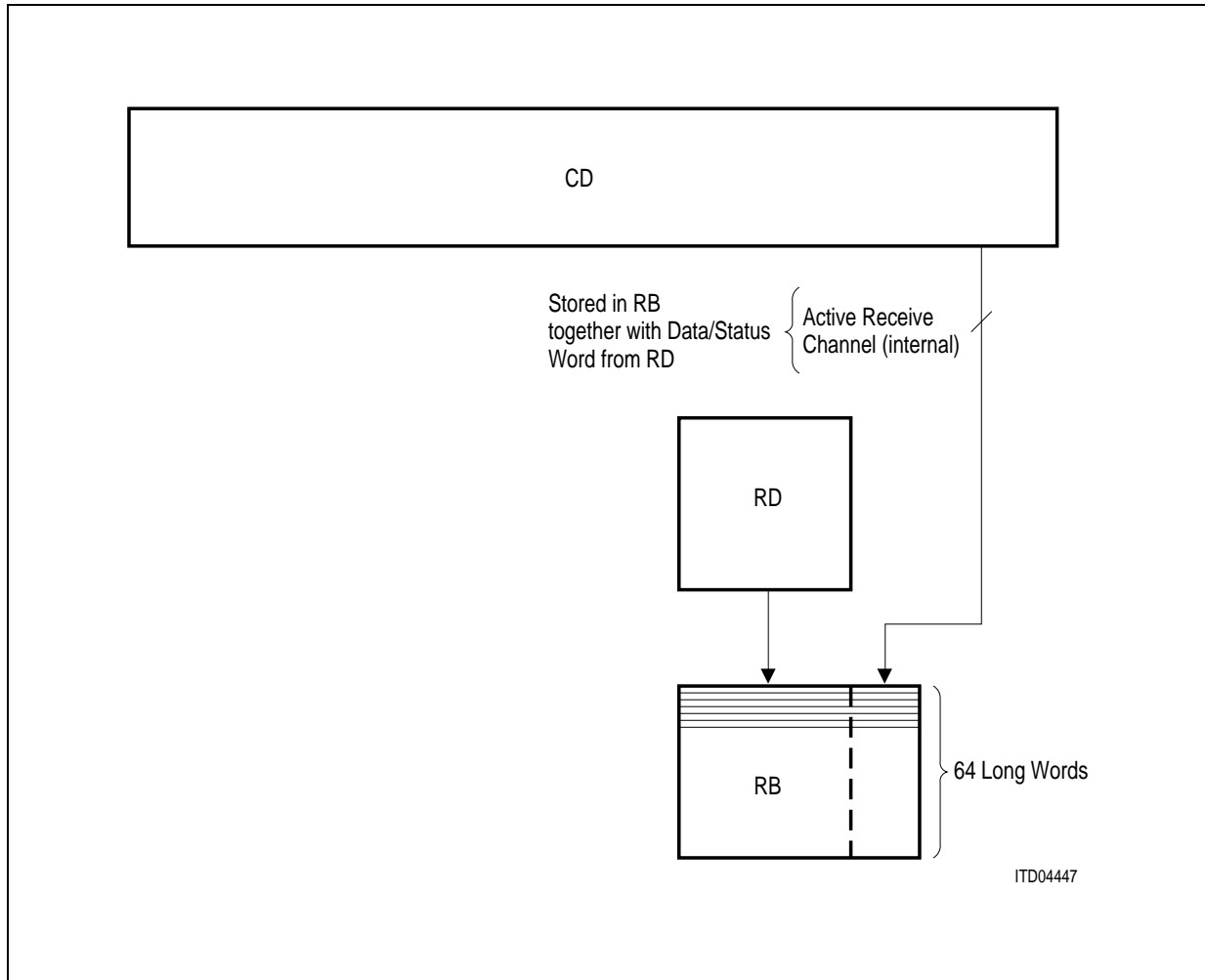
The choice of the individual buffer size of each data channel can be made in the channel specification (shared memory). The buffer size of one channel is changeable without disturbing the transmission of the other channels.



**Figure 26**  
**Partitioning of TB**

## Functional Description

The receive buffer (RB) is a FIFO buffer and also has a size of 64 long words, which allows storing the data of eight complete PCM frames before transferring to the shared memory.

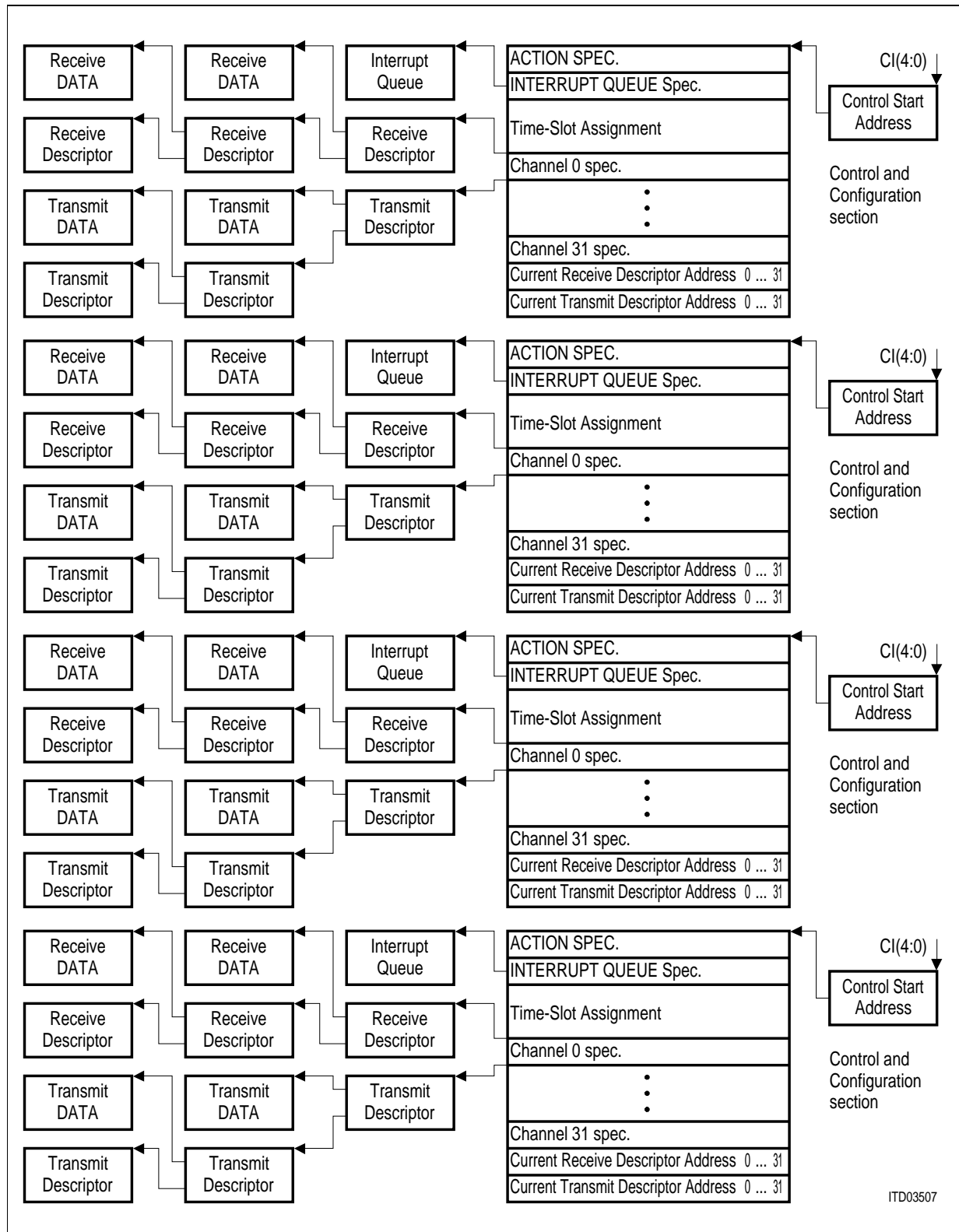


**Figure 27**  
**Partitioning of RB**

The data transfer to the shared memory is performed via a 32-bit microprocessor interface working either in SIEMENS/Intel or Motorola bus mode. **Figure 28** shows the division of the shared memory required for each MUNICH32:

- Configuration start address located at a programmable address
- Control and configuration section
- An interrupt circular queue with variable size
- Descriptor and data sections for each channel.

## Functional Description



**Figure 28**  
**Memory Division for up to four MUNICH32**

---

## Functional Description

The shared memory allocated for each transmit and receive channel is organized as a chaining list of buffers set up by the host. Each chaining list is composed of descriptors and data sections. The descriptor contains the pointer to the next descriptor, the start address and the size of a data section. It also includes control information like frame end indication, transmission hold and rate adaption with interframe time-fill.

In the transmit direction the MUNICH32 reads a transmit descriptor, calculates the data address, writes the current transmit descriptor address into the CCS, and fills the on-chip transmit buffer. When the data transfer of the specified section is completed, the MUNICH32 releases the buffer, and branches to the next transmit descriptor. If a frame end is indicated the HDLC, TMB or TMR frame will be terminated and a specified number of the interframe time-fill byte will be sent in order to perform rate adaption. If frame end is found in a transmit descriptor TMA channel the specified number of programmable TMA flags is appended to the data in the descriptor. If frame end is found in a transmit descriptor of a V.110/X.30 channel the frame is aborted (after the data in the descriptor are sent) by finishing the current 10-octet frame with 'zeros' and sending 2 more 10-octet frames with 'zeros' which leads to a loss of synchronism on the peer side. An adjustment for the inserted zeros in HDLC is programmable, which leads to a reduction of the specified number of interframe time-fill by  $\frac{1}{8}$ <sup>th</sup> of the number of zero insertions. This can be used to send long HDLC frames with a more or less fixed data rate in spite of the zero insertions. A maskable interrupt is generated before transmission is started again.

---

## Functional Description

The following Sections give Examples of Typical Transmit Situations for the Individual Modes

### Variable Size Frame Oriented Protocols (HDLC, TMB, TMR)

Normal operation, handling of frame end (FE) indication and hold (H) indication.

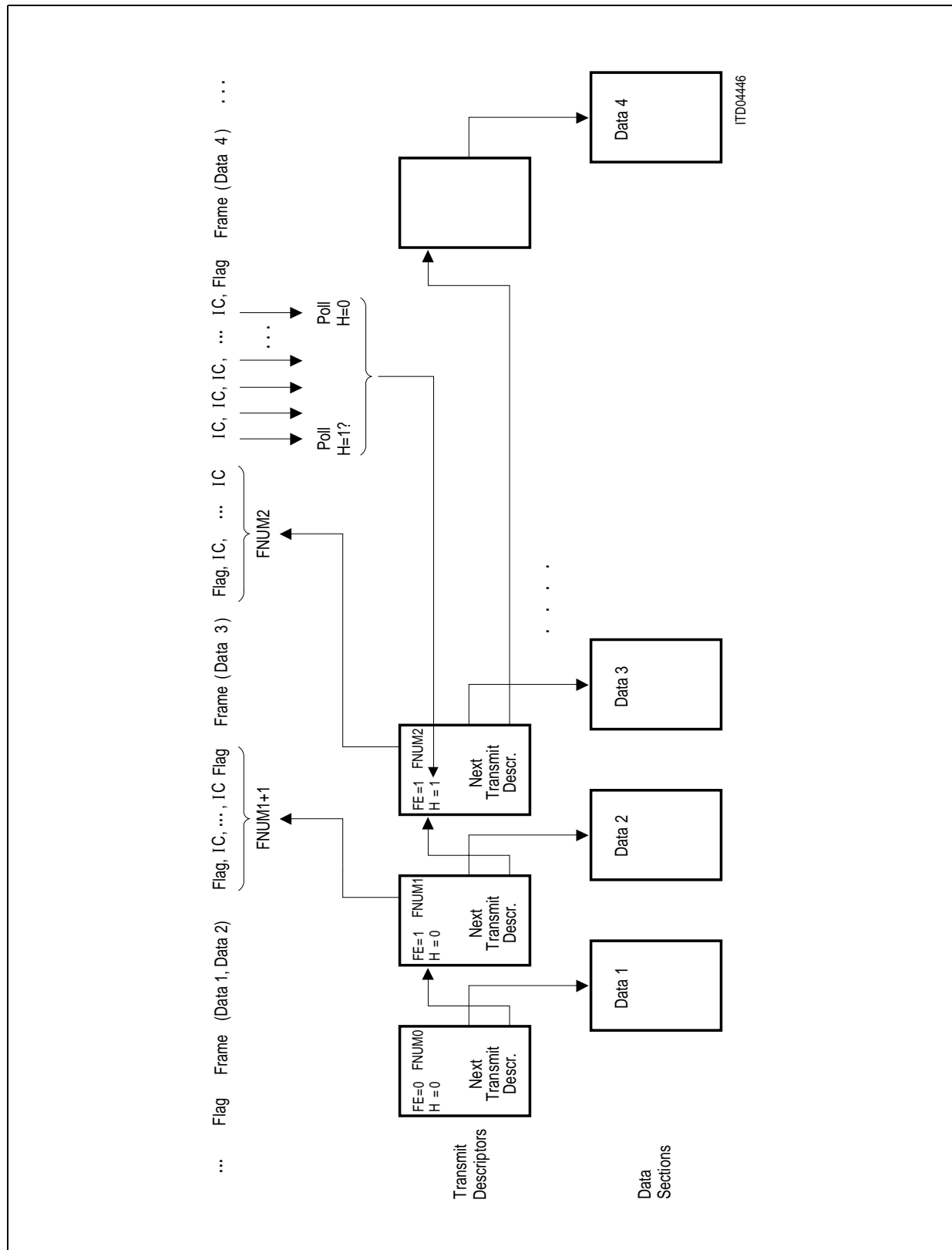
Note: 1. *FNUM0 must be set to zero.*

2. *Flag = 7E<sub>H</sub> for HDLC  
00<sub>H</sub> for TMB, TMR*

*IC = 7E<sub>H</sub> for HDLC and IFTF = 0  
FF<sub>H</sub> for HDLC and IFTF = 1  
00<sub>H</sub> for TMB, TMR*

3. *After sending the FNUM2 – 1 IC characters the device starts polling the hold bit in the transmit descriptor once for each further sent IC character. It also rereads the pointer to the next transmit descriptor once with each poll of the hold indication. The pointer to the next transmit descriptor can be changed while HOLD = 1 is set. The value of the pointer, (read in the poll where HOLD = 0) is used as the next descriptor address. If more than 6 IC characters will be sent, the use of the Transmit Hold (TH) should be considered as an alternative to using the descriptor hold bit. See **Chapter 5.3.2**.*

## Functional Description



### Figure 29

---

**Functional Description****Fixed Size Frame Oriented Protocols (V110/X.30)**

Normal operation, E, S, X change (indicated by the V.110-bit in the transmit descriptor)

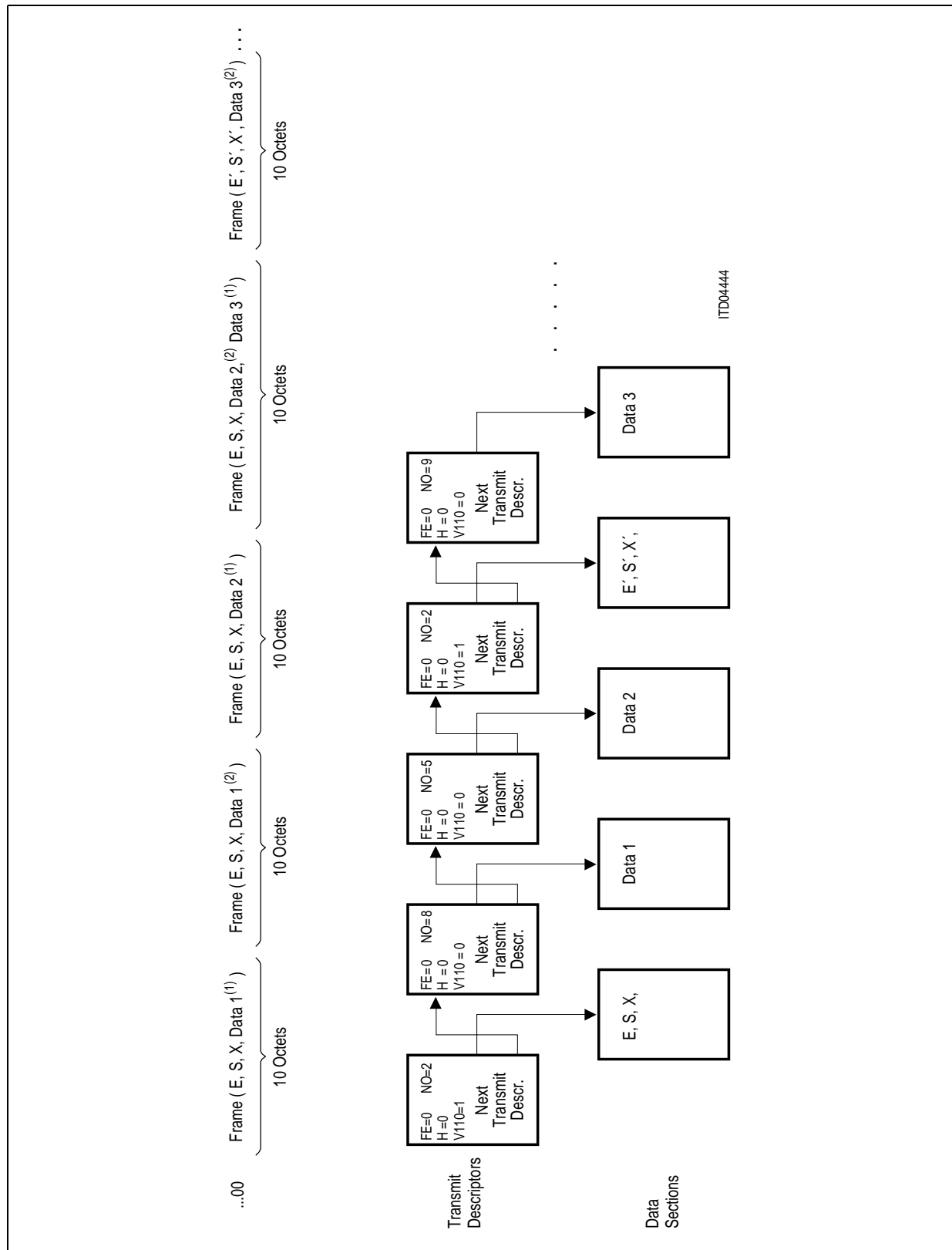
Example for TRV = '11'

*Note: 1. FNUM must be 0 for all transmit descriptors.*

- 2. The actual E-, S-, X-bits have to be in the first transmit descriptor after reset.*
- 3. As shown in the example the contiguous parts of a data section belonging to one descriptor are sent in contiguous frames (DATA 1<sup>(1)</sup> are the bytes 0 – 3 of DATA 1, DATA 1<sup>(2)</sup> are the bytes 4 – 7 of DATA 1). If the end of a data section is reached within a frame, the frame is continued with data from the next data section belonging to a transmit descriptor with the bit V.110 = 0 (DATA 2<sup>(2)</sup> = byte 4 of DATA 2, DATA 3<sup>(1)</sup> = byte 0 – 2 of DATA 3).*
- 4. The E-, S-, X-bits are only changed from one frame to the next not within a frame. The change occurs in the first frame which does not contain data of the previous data section.*
- 5. Neither FE nor H may be set to 1 during a normal operation of the mode. They both lead to an abort of the serial interface.*



## Functional Description



**Figure 30**

---

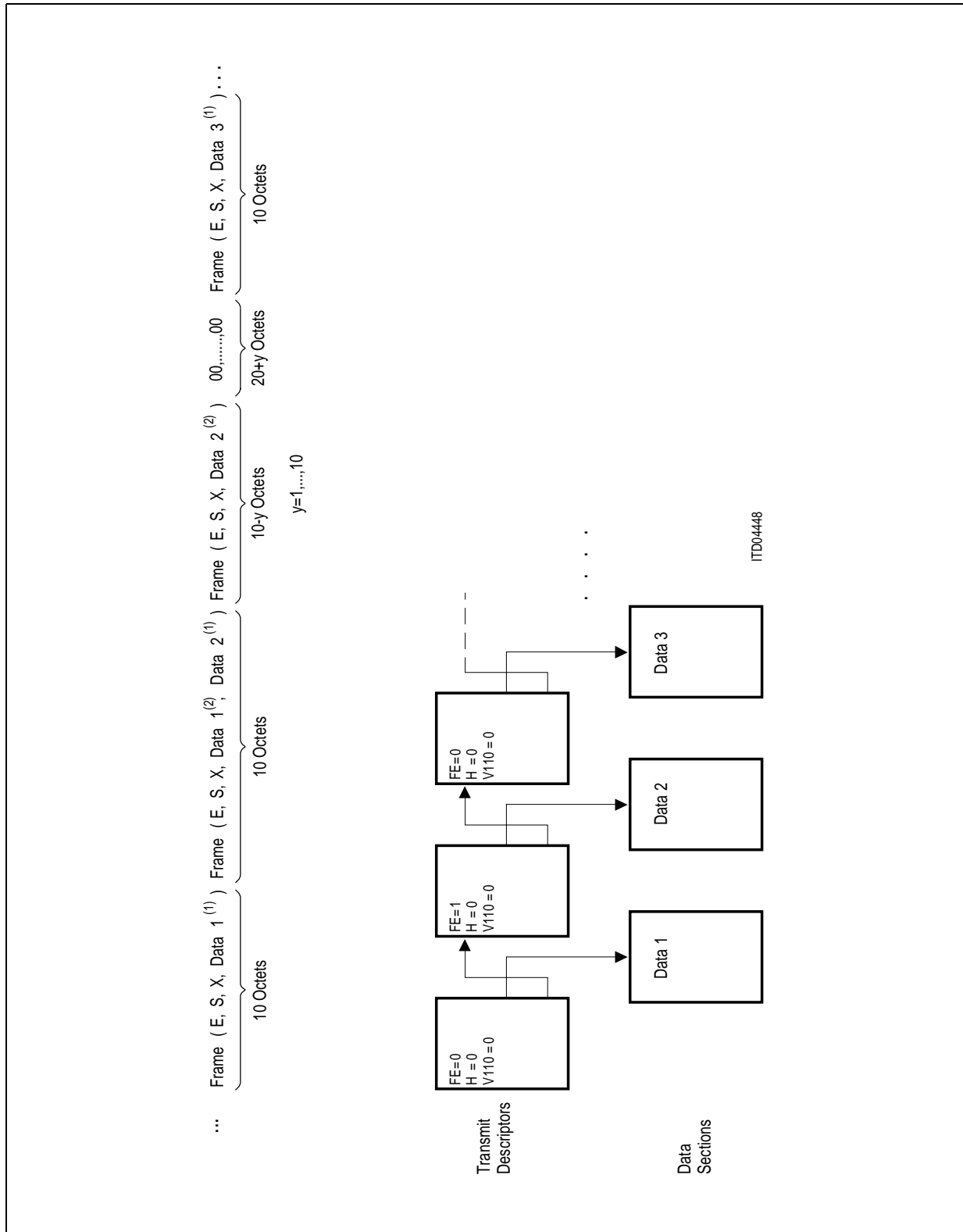
**Functional Description****Fixed Size Frame Oriented Protocols (V.110/X.30)**

Handling of frame end (FE) indication

*Note: 1. FNUM must be '0' for all transmit descriptors.*

- 2. The frame (E, S, X, DATA 2<sup>(2)</sup>) is the beginning of a 10-octet frame. It stops with the octet no. y, containing the last data bit of DATA 2 to be sent.*
- 3. Since y = 1, ..., 10 the 20 + y times 00<sub>H</sub> characters sent afterwards cause the peer station to recognize 3 consecutive 10-octet frames with frame error which leads to a loss of synchronism in the peer station.*
- 4. For y = 10 DATA 2 is identical to DATA 2<sup>(1)</sup> and 30 times 00<sub>H</sub> characters are sent after frame (E, S, X, DATA 1<sup>(2)</sup>, DATA 2<sup>(1)</sup>).*
- 5. The E-, S-, X-bits are supposed to be loaded by an earlier transmit descriptor in the example. A descriptor changing them (with V.110-bit set) can be put between, before or after the descriptors in the example. It will change these bits according to the rules discussed previously.*

## Functional Description

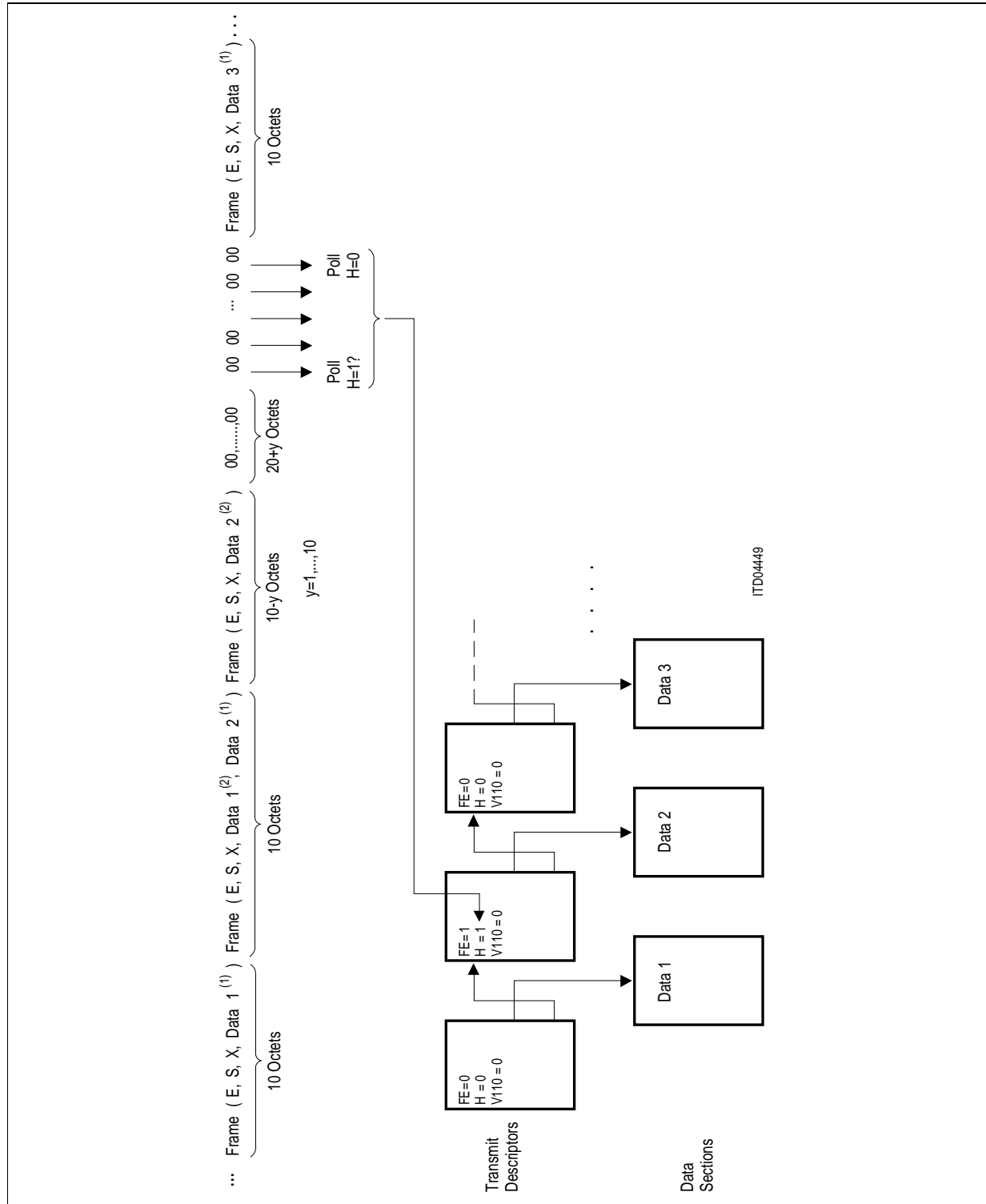


**Figure 31**

## Functional Description

### Fixed Size Frame Oriented Protocols (V110/X.30)

#### Handling of hold (H) indication



**Figure 32**

---

## Functional Description

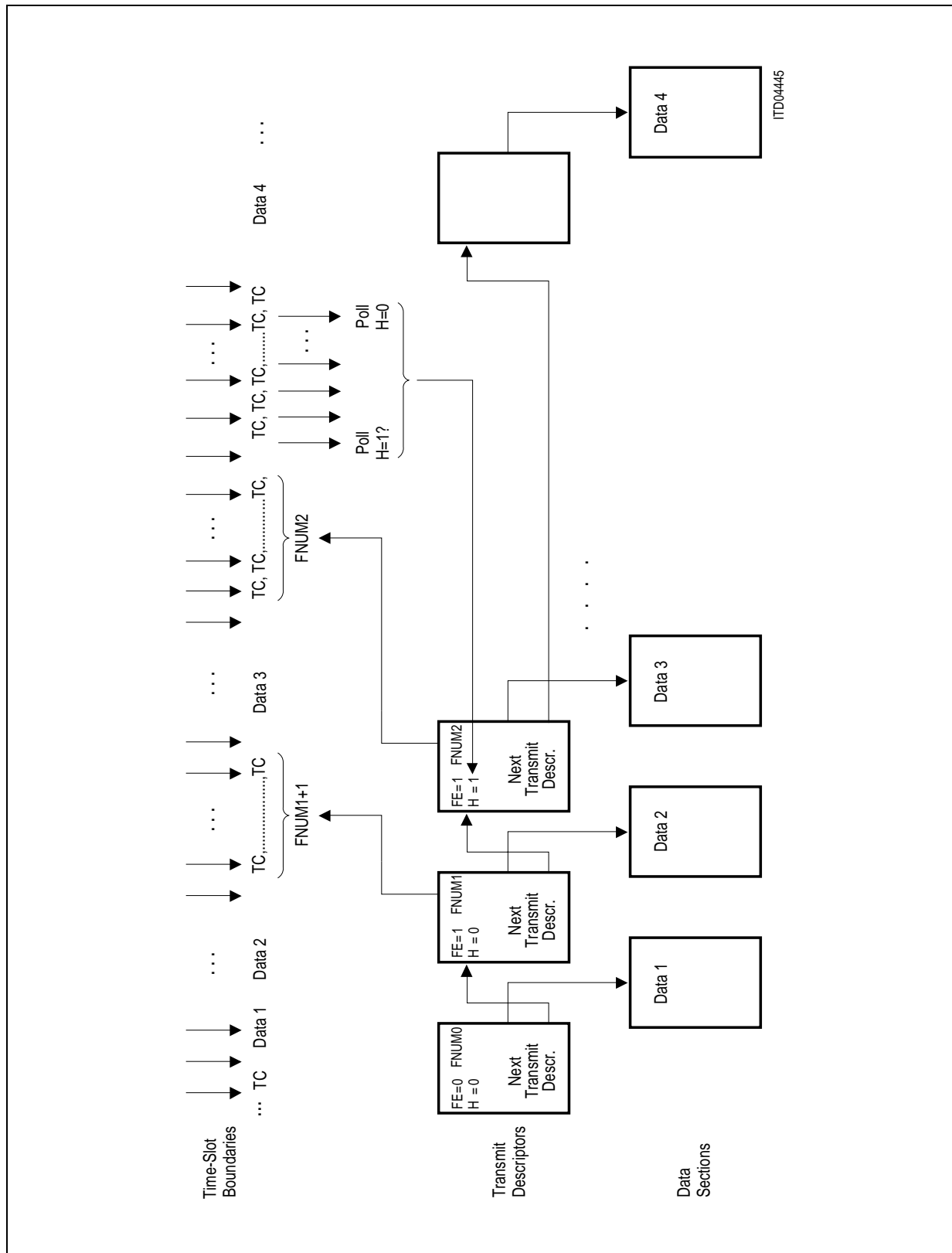
### Time Slot Oriented Protocol (TMA)

Normal operation, handling of frame end (FE) indication and hold (H) indication.

*Note:* 1. *FNUM must be set to zero.*

2.  $TC = FF_H$  for TMA and  $FA = 0$   
the programmed flag with TMA and  $FA = 1$
3. *After sending the  $FNUM2 - 1$  IC characters the device starts polling the hold bit in the transmit descriptor once for each further sent IC character. It also rereads the pointer to the next transmit descriptor once with each poll of the hold indication. The pointer to the next transmit descriptor can be changed while  $HOLD = 1$  is set. The value of the pointer, (read in the poll where  $HOLD = 0$ ) is used as the next descriptor address. If more than 6 IC characters will be sent, the use of the Transmit Hold (TH) should be considered as an alternative to using the descriptor hold bit. See **Chapter 5.3.2**.*

## Functional Description



### Figure 33

## Functional Description

An activated transmission hold (hold bit in descriptor) prevents the MUNICH32 from sending more data. If a frame end has not occurred just before, the current frame will be aborted and an interrupt generated. Afterwards, the interframe time-fill bytes will be issued until the transmission hold indication is cleared. There is a further transmit hold (TH) bit in the Channel Specification (CCS) in addition to the hold bit in the descriptor. Setting the transmit hold (TH) bit by issuing a channel command will prevent further polling of the transmit descriptor (see **Chapter 5.3.2**).

This hold bit (CCS.TH) is interpreted in the CD; it causes the transmit formatter to stay in the idle state and to send interframe time-fill after finishing the current frame. In the case of a very short frame ( $< ITBS$ ), this frame will stay in the TF and not be sent until CCS.TH is removed. (In case of X.30/V.110 the current frame is aborted).

This means that the buffer TB is not emptied from the TF side after the current frame, but still requests further data from the shared memory until it is filled. In the case of the descriptor hold on the other hand, the TF empties the TB and there are no further data requests from the shared memory until the descriptor hold is withdrawn. Then TB is filled again and the TF is activated only after enough data are provided in the TB to prevent a data underrun.

### The Reaction to the Transmit Hold Bit is now Discussed for the Different Modes in the Following Sections

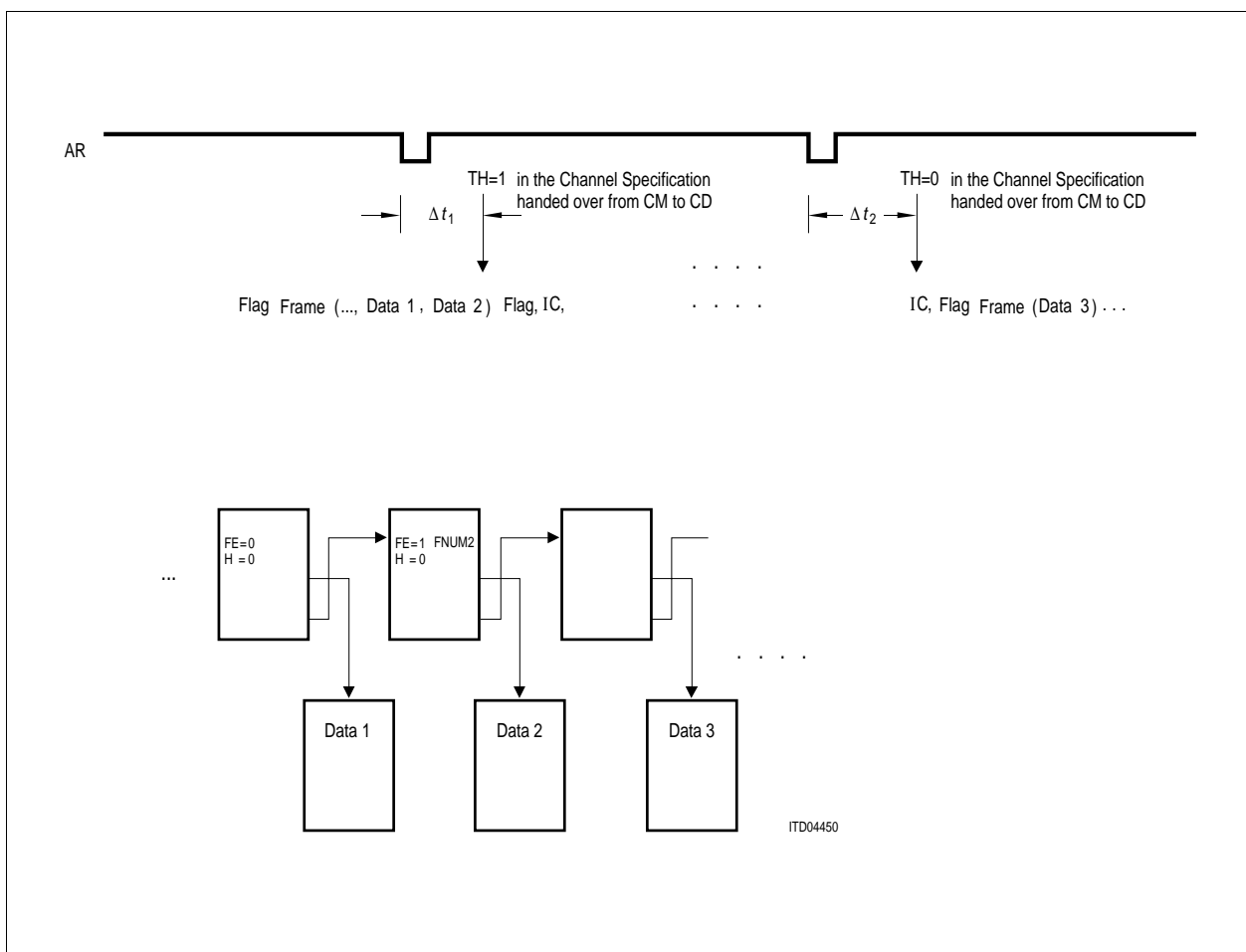
#### Variable Size Frame Oriented Protocols (HDLC, TMB, TMR)

Reaction to a channel specification containing  $TH = 1$

Normal operation

*Note:* 1.  $IC = 7E_H$  for HDLC and  $IFTF = 1$   
 $FF_H$  for HDLC and  $IFTF = 0$   
 $00_H$  for TMB or TMR

2.  $flag = 7E_H$  for HDLC  
 $00_H$  for TMB or TMR
3.  $FNUM2$  is ignored. The number of interframe time-fills sent between the first frame and the second frame solely depends on the AR low pulse leading to the action with the channel with  $TH = 0$ .
4. The times  $\Delta t_1$  and  $\Delta t_2$  are statistical but typically only a few clock cycles.
5. The TH bit (as all channel commands) is **not** synchronized with TB! (as opposed to the H-bit in the descriptor) TH acts on the frame currently being sent, not necessarily on the last frame currently stored in the TB. In the example, TB may or may not have stored DATA 3 before the action request with  $TH = 1$  was issued. See **Chapter 4.2.5** for a further discussion of this issue.
6. If TH is handed over to CD outside of a frame,  $TH = 1$  prevents the MUNICH32 from sending the next frame.





---

## Functional Description

### Fixed Size Frame Oriented Protocol (V.110/X.30)

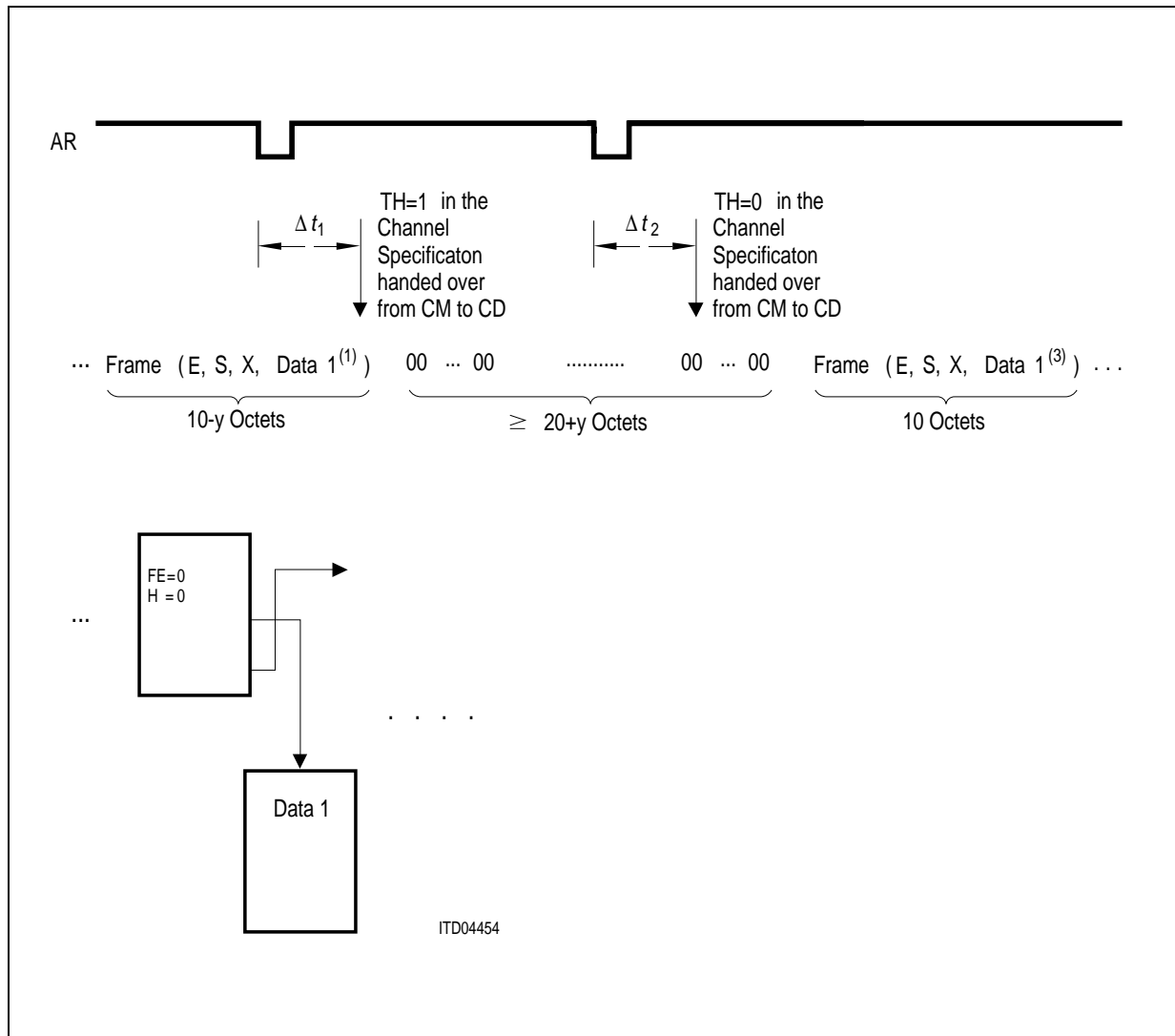
Reaction to a channel specification containing TH = 1

Normal operation

*Note: 1. The times  $\Delta t_1$  and  $\Delta t_2$  are statistical but typically only a few clock cycles.*

- 2. The current frame processed, when TH = 1 is handed over to CD is aborted, only  $10 - y$ , ( $y = 1, \dots, 10$ ) octets of it are sent. The device then starts to send  $20 + y 00_H$  characters no matter how fast the TH bit is withdrawn. This ensures, that the peer site is informed about the abort with a loss of synchronism*
- 3. The data section DATA 1 is split in the example; DATA 1<sup>(1)</sup> is sent in the aborted frame, all bits that were read into the MUNICH32 with the same access are discarded (they would have been sent in the next frame(s) if TH = 1 was not issued) and the device starts the next frame with the bits DATA 1<sup>(3)</sup> of the access to DATA 1 that follows the one getting the bits of DATA 1<sup>(1)</sup>.*
- 4. The TH (as all channel commands) is **not** synchronized with TB. TH acts on the frame currently sent, not necessarily on the last stored data.*
- 5. If TH is handed over to CD before a frame has started after an abort or after reset no frame will start.*

## Functional Description



**Figure 35**

### Time Slot Oriented Protocol (TMA)

Reaction to a channel specification containing TH = 1

*Note:* 1. TC is the programmed TFLAG for FA = 1  
FF<sub>H</sub> for FA = 0

2. The times  $\Delta t_1$  and  $\Delta t_2$  are statistical but typically only a few clock cycles.
3. The TH bit (as all channel commands) is **not** synchronized with the TB! (as opposed to the H-bit in the descriptor) TH acts to the data stream currently sent.

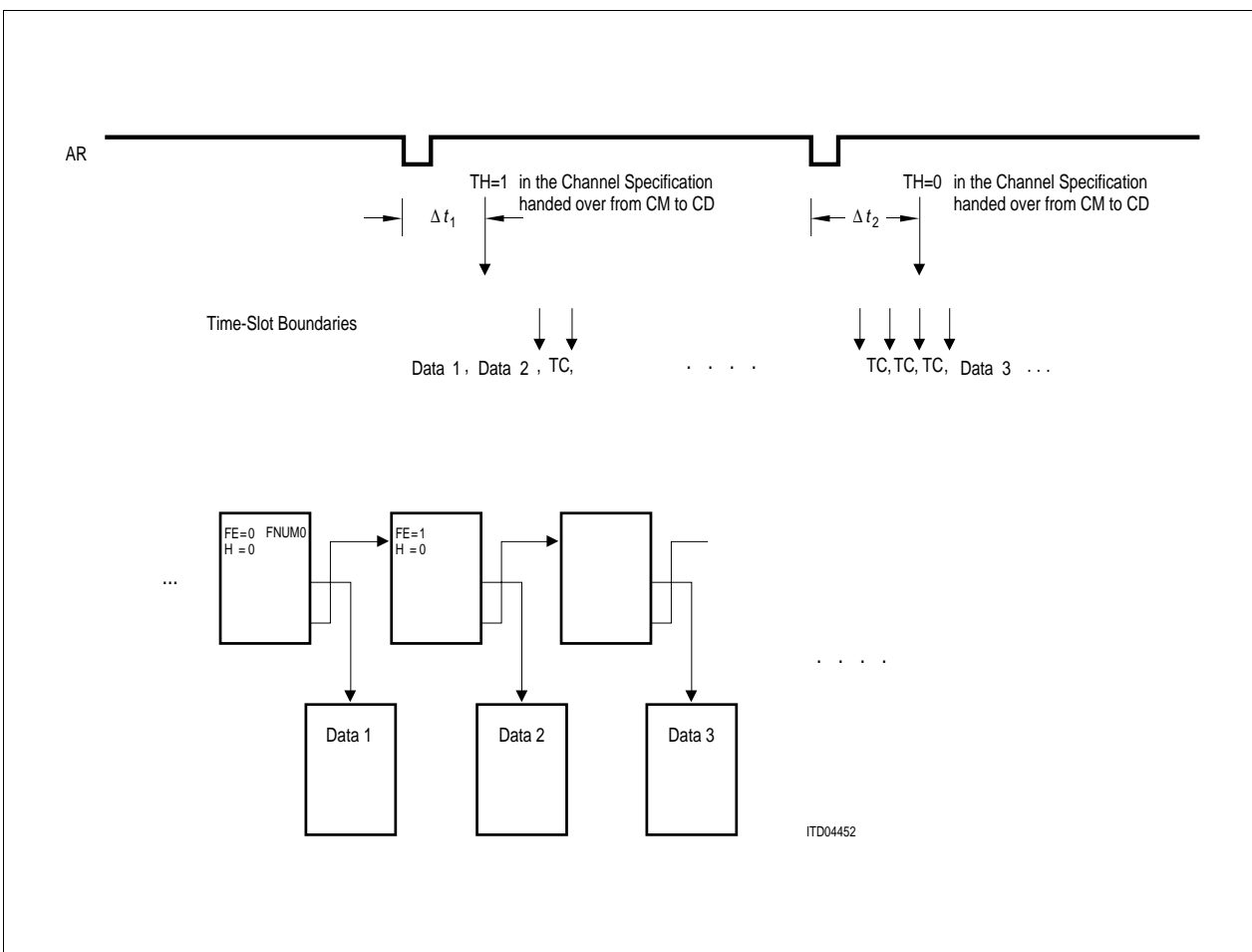


Figure 36

---

**Functional Description****Variable Size Frame Oriented Modes (HDLC, TMB, TMR)**

Reaction to a channel specification containing  $TH = 1$

Silencing of poll cycles for hold.

*Note: An AR pulse for an action specification leading to  $TH = 1$  should be issued after  $(ITBS + 2)$  polls of the MUNICH32, where  $ITBS$  is the previously programmed number of long words in the TB reserved for this channel.*

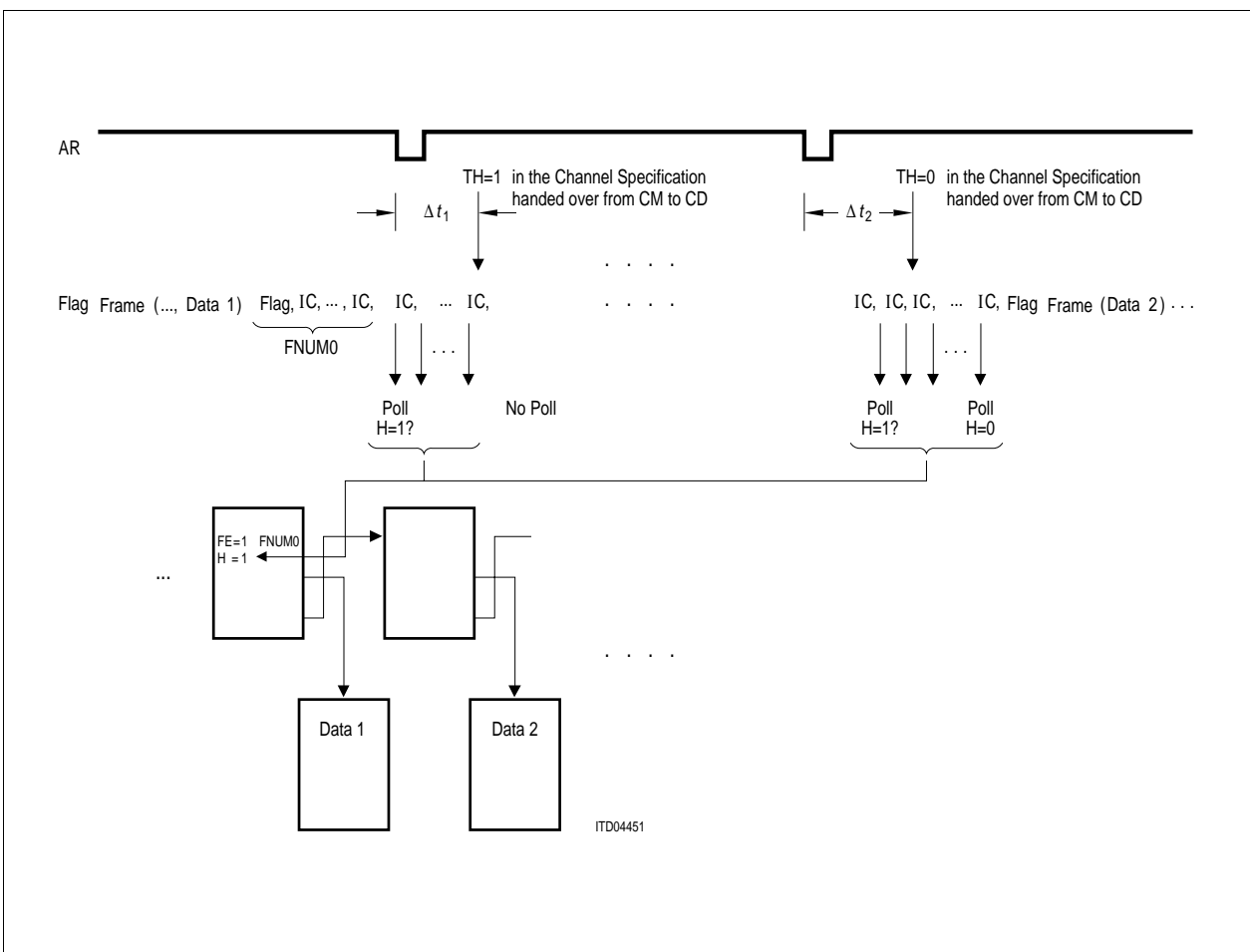


Figure 37

---

## Functional Description

### Fixed Size Frame Oriented Protocol (V110/.30)

Silencing of poll cycles by  $TH = 1$

*Note: 1. The times  $\Delta t_1$  and  $\Delta t_2$  are statistical but typically only a few clock cycles.*

- 2. The  $TH$  bit (as all channel commands) is **not** synchronized with  $TB$ ! (as opposed to the  $H$ -bit in the descriptor)  $TH$  acts to the data stream currently sent.*
- 3. In the example the proper use to silence a channel polling the  $HOLD$  bit of the transmit descriptor is illustrated. The  $AR$  pulse is issued **after** the polling has started and the  $H$ -bit is not reset before polling has stopped by the  $TH$  bit.*
- 4. An  $AR$  pulse for an action specification leading to  $TH = 1$  should be issued after  $(ITBS + 2)$  polls of the  $MUNICH32$ , where  $ITBS$  is previously programmed number of long words in the  $TB$  reserved for this channel.*

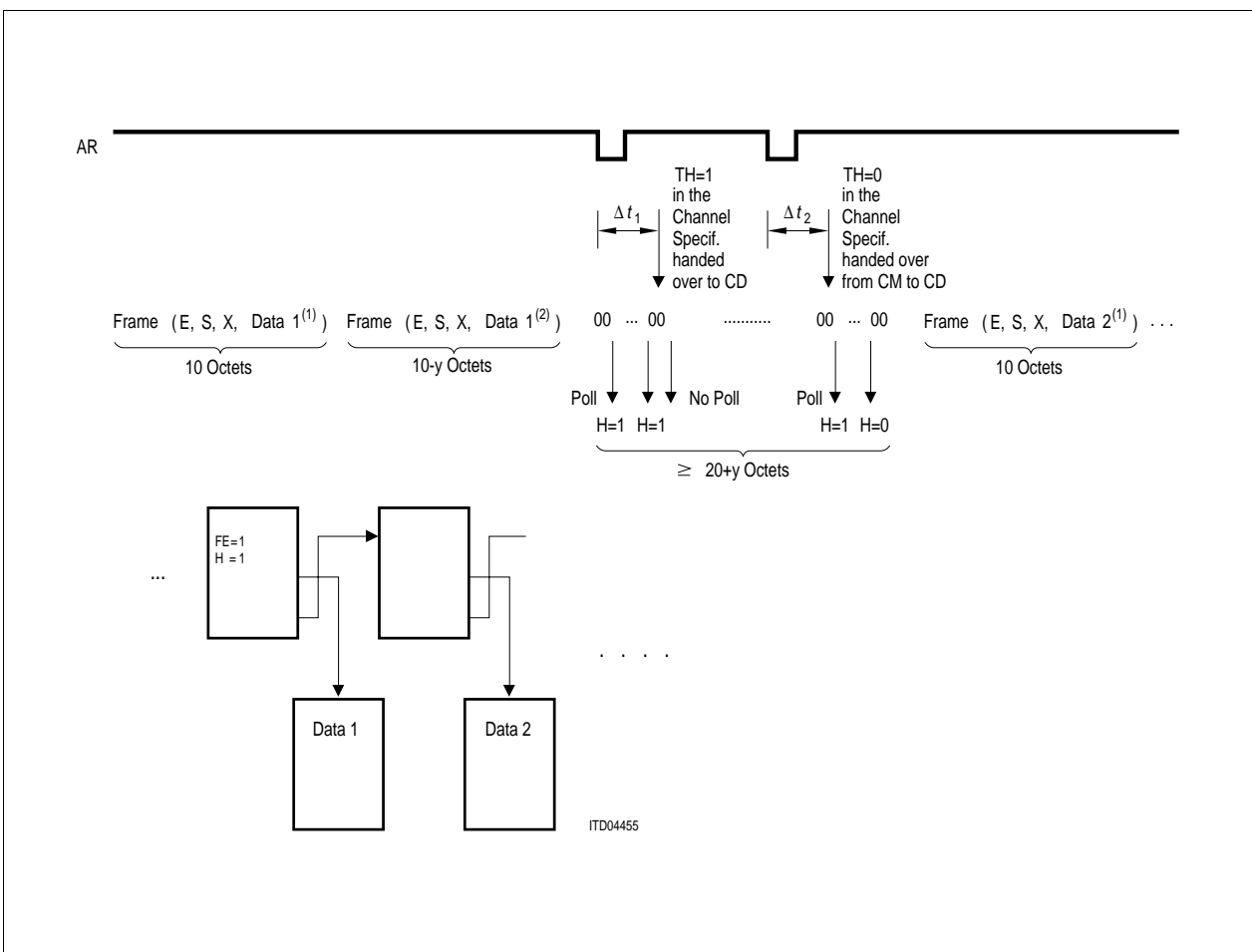


Figure 38

---

## Functional Description

### Time Slot Oriented Protocol (TMA)

Reaction to a channel specification containing  $TH = 1$

Note: 1.  $TC = FF_H$  for TMA and  $FA = 0$

*the programmed flag for TMA and  $FA = 1$*

2.  *$FNUM2$  is ignored. The number of interframe time-fills between the first frame and the second frame solely depends on the AR low pulse leading to the action with the channel with  $TH = 0$ .*
3. *The times  $\Delta t_1$  and  $\Delta t_2$  are statistical but typically only a few clock cycles.*
4. *The  $TH$  bit (as all channel commands) is **not** synchronized with  $TB$  (as opposed to the  $H$ -bit in the descriptor)  $TH$  acts on the data stream currently sent not necessarily on the last data stored in  $TB$ . In the example  $TB$  may or may not have stored DATA 3 before action request with  $TH = 1$  was issued.*
5. *The data stream is stopped and  $TC$  sent after the last byte of DATA 2 is sent. The stopping is triggered by the  $FE = 1$  bit in the descriptor.*
6. *If  $TH$  is bonded over to  $CD$  during interframe time-fill ( $TC$ ) it prevents the MUNICH32 from sending further data afterwards.*
7. *An AR pulse for an action specification leading to  $TH = 1$  should be issued after  $(ITBS + 2)$  polls of the MUNICH32, where  $ITBS$  is previously programmed number of long words in the  $TB$  reserved for this channel.*



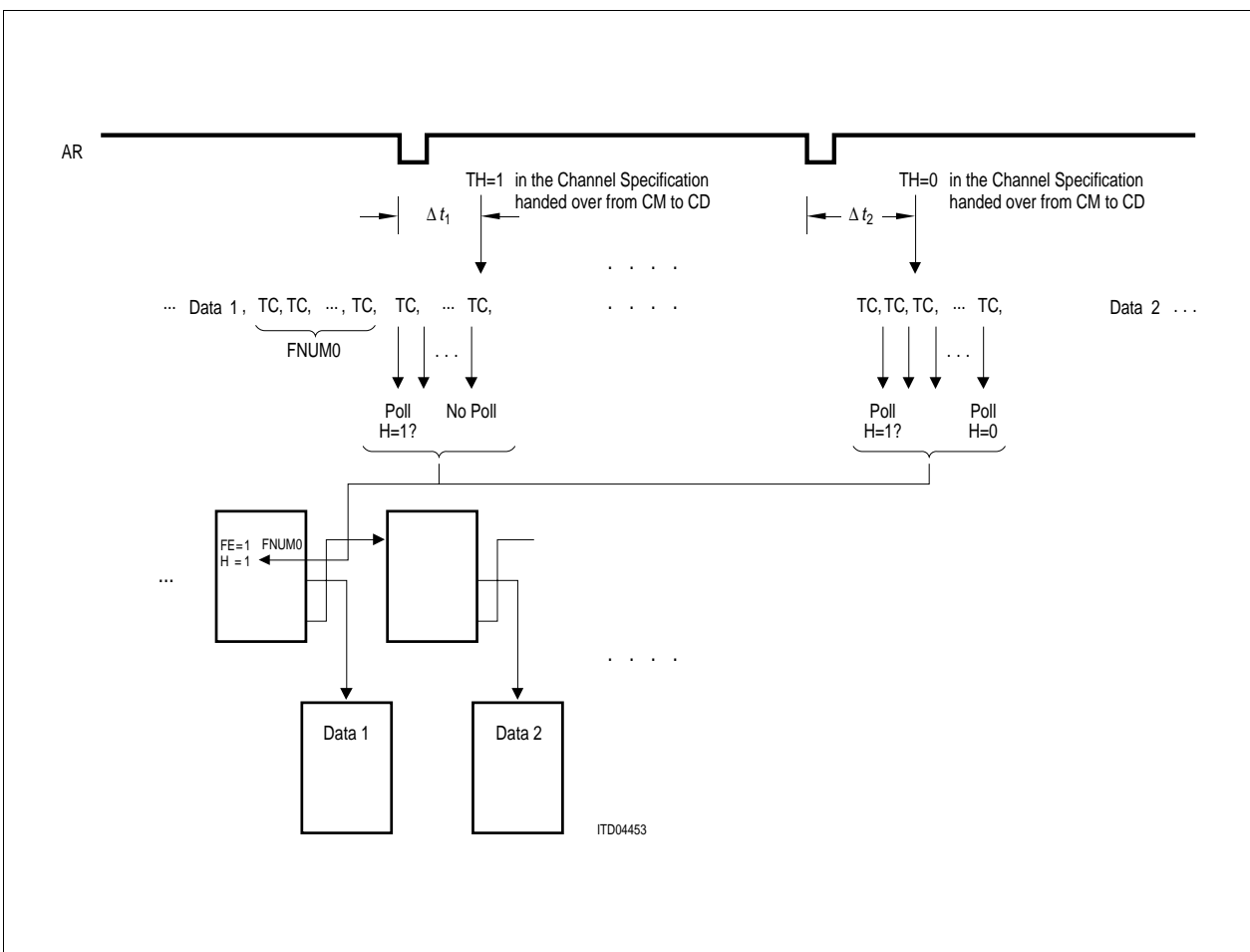


Figure 39

---

## Functional Description

In receive direction the MUNICH32 reads a receive descriptor, calculates the data address, writes the current receive descriptor address into the CCS, and exchanges data between the on-chip receive buffer and the external memory. After the data section has been filled, the MUNICH32 writes the number of stored bytes (BNO) into the descriptor. If a frame end has occurred the frame status is written into the descriptor and an interrupt is generated. The frame status includes the CRC check results and transmission error information like 'non octet of bits', 'aborted frame', 'data overflow', 'maximum frame length exceeded' and 'frames with less than or equal to two data bytes'. An activated reception-hold in the descriptor prevents the MUNICH32 from processing the receive data. The incoming frames are discarded until the hold is deactivated.

Because the MUNICH32 is divided into two non-synchronized parts by the on-chip buffers, two different kinds of aborting a channel transmission are implemented.

- Normal abort: This abort of a receive or transmit channel is processed in the formatters of the serial interface. The interframe time-fill code is sent after aborting the current issued frame. No accesses to the on-chip buffers are carried out, until the abort is withdrawn. The handling of the link lists and the processing of the buffers by the DMA controller are not affected by normal abort.
- Fast abort: A fast abort is performed by the DMA controller and does not disturb the transmission on the serial interface. If this abort is detected the current descriptor is suspended with an abort status immediately followed by a branching to the new descriptor defined in the channel specification of the CCS.

For initialization and control the host sets up a Control and Configuration Section (CCS), including the action specification, interrupt queue specification, time slot assignment and the channel specification. The host initiates an action, e.g. reconfiguration, change of the channel mode, reset or switching of a test loop by updating the CCS and issuing an action request pulse. When the action request pulse is detected by the MUNICH32 it reads the control start address, then the action specification and, if necessary, additional information from the CCS. After execution, the action request is acknowledged by an interrupt.

MUNICH32 indicates an interrupt by activating the interrupt line and storing the interrupt information including the corresponding channel number in the interrupt queue. The interrupt queue is a circular buffer; MUNICH32 starts to write the interrupt queue specification and fills it successively in a circular manner. The host has to allocate sufficient buffer size and to empty the buffer fast enough in order to prevent overflow of the queue.

---

## Functional Description

Monitoring functions are implemented in MUNICH32 to discover errors or condition changes, i.e.

- Receive frame end
- Receive frame abort by overflow of the receive buffer or hold condition or recognized ABORT flag
- Frame overflow, if a frame has to be discarded because of pending inaccessibility of the chip memory
- Transmit frame end
- Transmit frame abort (data underrun) by underrun of the transmit buffer or hold condition or bus cycle error
- Change of the interframe time-fill.
- Loss of synchronism or change of framing bits (V.110, X.30).
- Short frame with no data content detected.

An error or condition change is indicated by an interrupt. The host may react to the interrupt by either aborting or tristating the specific channel or with a channel reconfiguration. To prevent underrun of the transmit buffer sufficient buffer size has to be allocated to the channel.

A more detailed discussion of the receive procedure with examples is provided under the detailed protocol description in **Chapter 2.4**.

## 2.4 Detailed Protocol Description

In the following sections the protocol support of the MUNICH32 is described in detail for transmit and receive direction separately.

Each section starts with a discussion of the general features proceeds with protocol variants and options from the channel specification and closes with a description of the interrupts and special topics.

### HDLC

#### Transmit Direction General Features

In transmit direction

- the starting and ending flag ( $7E_H$  before and after a frame)
- the interframe time-fill between frames
- the zero insertions (a '0'-bit after 5 consecutive '1's inserted within a frame)
- (optional) the Frame Check Sequence (FCS) at the end of a frame

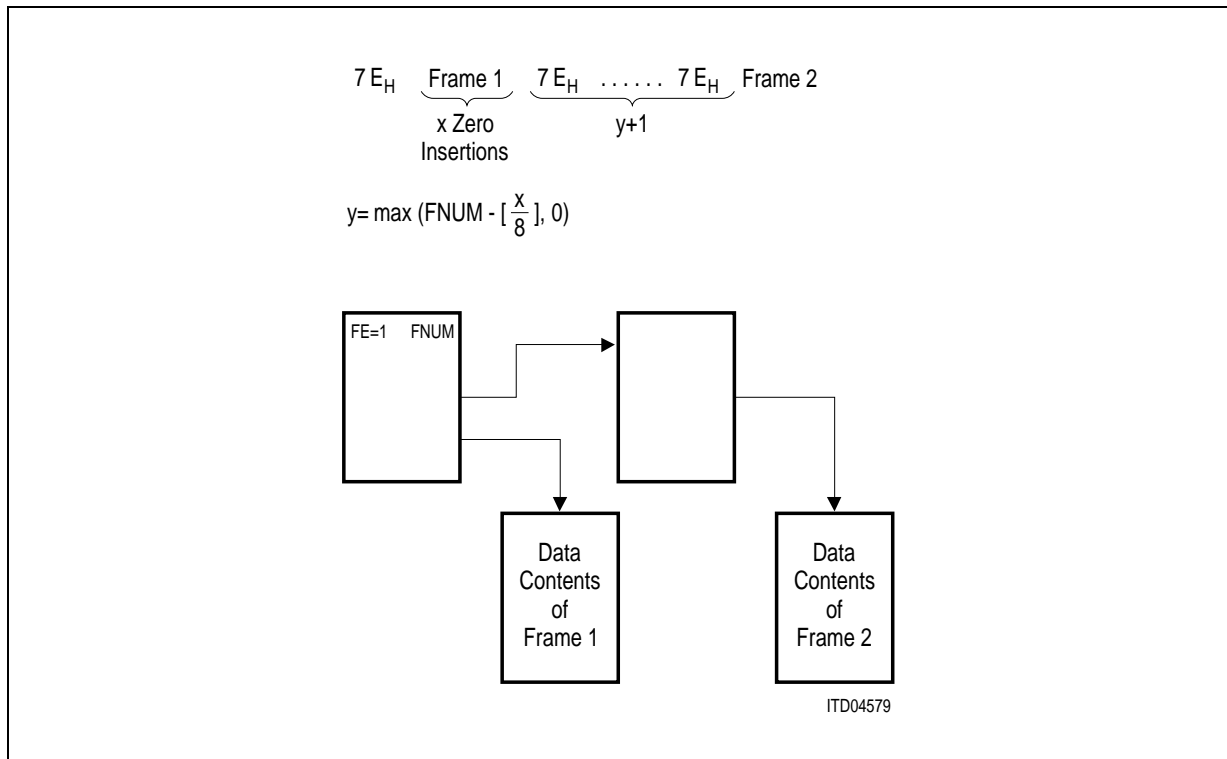
is generated automatically.

#### Options

The different options for this mode are

- the kind of the interframe time-fill character in the channel specification
  - $7E_H$  for IFTF = 0
  - $FF_H$  for IFTF = 1
- the number of interframe time-fill characters as FNUM in the transmit descriptor. For the values FNUM = 0, 1, 2 we have
  - FNUM = 0    frame 1,  $7E_H$ , frame 2 (start flag = end flag)
  - FNUM = 1    frame 1,  $7E_H$ ,  $7E_H$ , frame 2
  - FNUM = 2    frame 1,  $7E_H$ , IC,  $7E_H$ , frame 2
- the correction of the number of interframe time-fill characters by  $\frac{1}{8}$  of the number of zero insertions by programming FA in the channel specification.
  - FA = 0:        FNUM from the transmit descriptor is taken directly to determine the number of interframe time-fill characters as shown in **Figure 39**.
  - FA = 1:        FNUM from the transmit descriptor is reduced by  $\frac{1}{8}$  of the number of the zero insertions of the frame corresponding to the transmit descriptor as shown in **Figure 40**. This allows for a more or less constant bit rate transmission for long HDLC frames.

## Functional Description



**Figure 40**

Note: 1.  $\left\lfloor \frac{x}{8} \right\rfloor$  is the biggest integer smaller than  $\frac{x}{8}$ .

1. For  $FNUM - \left\lfloor \frac{x}{8} \right\rfloor < 0$ ,  $y = 0$

- the kind of Frame Check Sequence (FCS)  
two kinds of frame check sequences are implemented by the CRC bit in the channel specification  
 CRC = 0: the generator polynomial  $x^{16} + x^{12} + x^5 + 1$  is used  
 (2 byte FCS of CCITT Q.921)  
 CRC = 1: the generator polynomial  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + \dots$   
 $\dots x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$   
 (4 byte FCS) is used
- the suppression of the automatic generation of the FCS is programmable in the channel specification:
- CS = 0: FCS generated automatically  
 CS = 1: FCS generation suppressed  
 and in the transmit descriptor  
 CSM = 0: FCS generated automatically if CS = 0 in the channel specification  
 CSM = 1: FCS generation suppressed

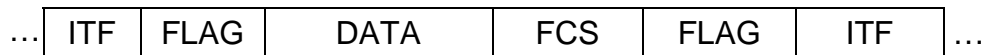
## Functional Description

### Interrupts

The possible interrupts for the mode in transmit direction are:

- HI: issued if the HI bit is detected in the transmit descriptor (not maskable)
- FI: issued if the FE bit is detected in the transmit descriptor (maskable by FIT in the channel specification)
- ERR: one of the following transmit errors has occurred:
  - the last descriptor had H = 1 and FE = 0
  - the last descriptor had NO = 0 and FE = 0 (maskable by TE in the channel specification)
- FO: one of the following transmit errors have occurred
  - a BERR = '0' was detected during a read access to a transmit data section for this channel
  - the MUNICH32 was unable to access the shared memory in time either for new data to be sent or for a new transmit descriptor. (maskable by TE in the channel specification)

typical data stream has the form

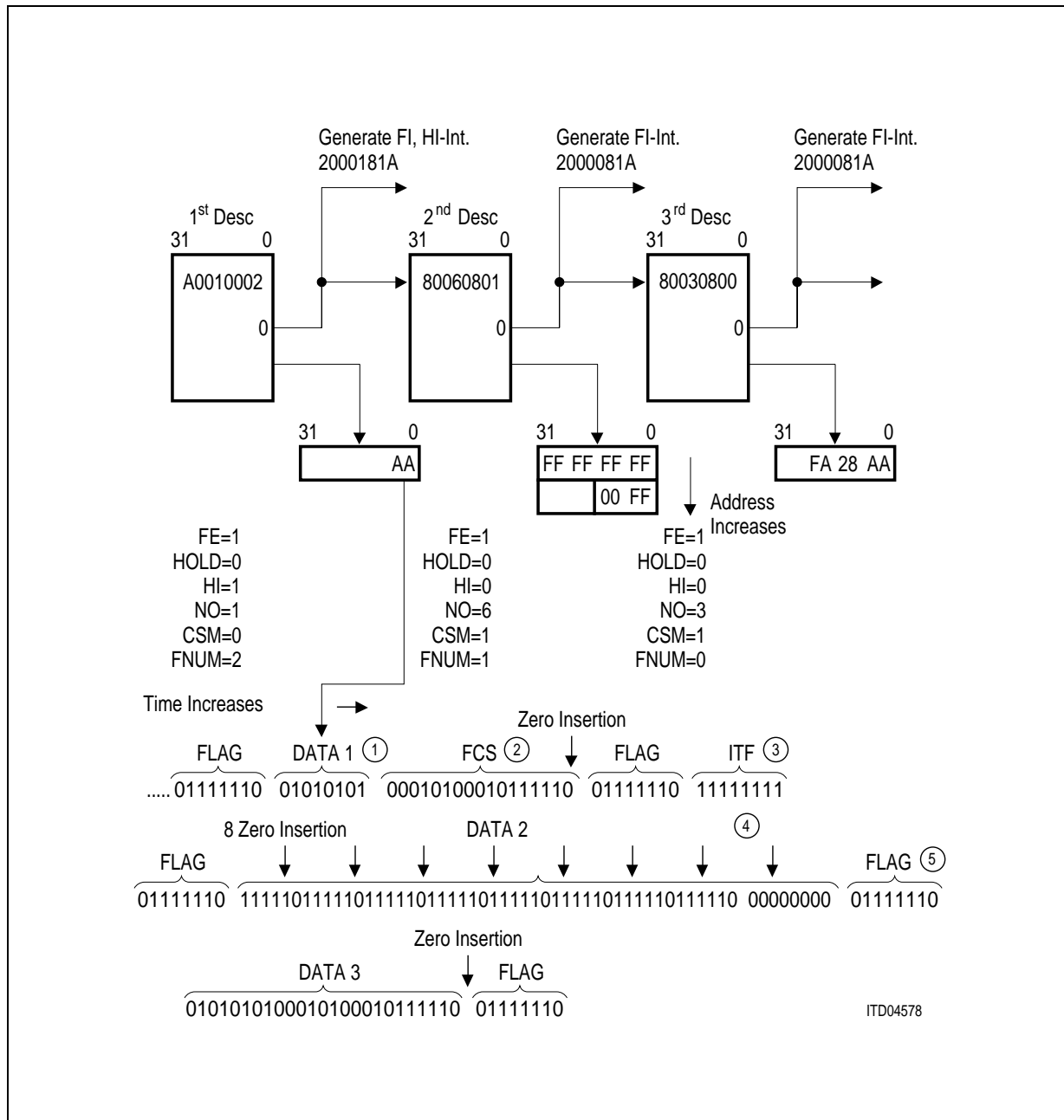


### Example:

HDLC channel with

- CS = 0) (FCS generated automatically)
- INV = 0 (no inversion)
- CRC = 0 (CRC16)
- TRV = 00 (required as unused in HDLC mode)
- FA = 1 (flag adjustment)
- MODE = 11 (HDLC)
- IFTF = 1 (interframe time-fill '1's)
- INTEL interface
- Channel number 1A

## Functional Description



### Figure 41

*Note: 1. Data is transmitted according to §2.8 of CCITT recommendation Q.921*

2. Note: FCS in the data section is formatted as ordinary data!!!

3. *FCS* is generated here automatically as  $CS=0$  and  $CSM=0$  for the 1<sup>st</sup> descriptor.

4. There was 1 zero insertion in the 1<sup>st</sup> frame, so  $FNUM - \left\lceil \frac{1}{8} \right\rceil = FNUM = 2$ . Therefore between the first and the second frame we have  $FLAG \text{ ITF } FLAG$  and  $ITF = FFH$  because  $ITF = 1$ .

## Functional Description

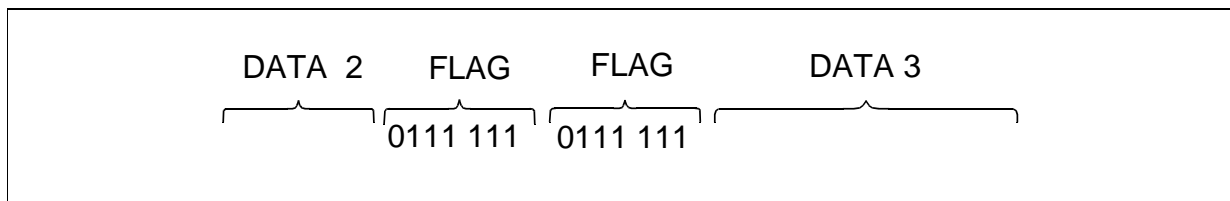
5. No FCS is generated here as CSM is '1' for the second and third transmit descriptor. The FCS is supposed to be the last 2 bytes to be transmitted in this case, their validity is not checked internally.
6. There was 8 zero insertions in the 2<sup>nd</sup> frame, so  $FNUM - \left\lceil \frac{8}{8} \right\rceil = FNUM - 1 = 0$ . Therefore between the second and the third frame we have a shared FLAG.

For CS = 1 (CRC select) the transmitted data stream would differ at FCS, FCS would just be omitted.

For INV = 1 (channel inversion) all bits of the data stream (including FLAG, DATA, FCS, ITF) would be inverted.

For CRC = 1 (CRC 32) the transmitted data stream would only differ in the FCS, the FCS would be 1101 0111 1010 0101 1000 0000 0010 0111.

For FA = 0 (no flag adjustment) the transmitted data stream would change only after DATA 2. The value FNUM = 1 in the second descriptor would alone determine the number of interframe time-fill characters, the scenario would look like



**Figure 42**

For ITF = 0 (ITF flags) the transmitted data stream would only differ at ITF, the 8 ones would be replaced by 0111 1110.

For Motorola interface the only difference is in the data section

For the first descriptor it ought to be

31	0
AA	

and for the second

31	0
FF	FF
FF	FF
FF	00

and for the third

31	0
AA	28
FA	



## HDLC

### Receive Direction

## General Features

In receive direction:

1. The starting and ending flag ( $7E_H$  before and after a frame) is recognized and extracted.
2. A change of the interframe time-fill is recognized and reported by an interrupt.
3. The zero insertions (a '0'-bit after five '1's within a frame) are extracted.
4. The FCS at the end of a frame is checked, it is (optionally) transferred to the shared memory together with the data.
5. The number of the bits within a frame (without zero insertions) is checked to be divisible by 8.
6. The number of bytes within a frame is checked to be smaller than  $MFL + 1$  (after extraction of '0' insertions).
7. The number of bits within a frame after extraction of '0' insertions is checked to be greater than (case NSF = 0 only)

[illegible][illegible]

(case NSF = 1 & CS = 1 only) check a') 8 for CRC = x (ignored)

8. The occurrence of an abort flag ( $7F_H$ ) ending a frame is checked.

More detailed description of the individual features:

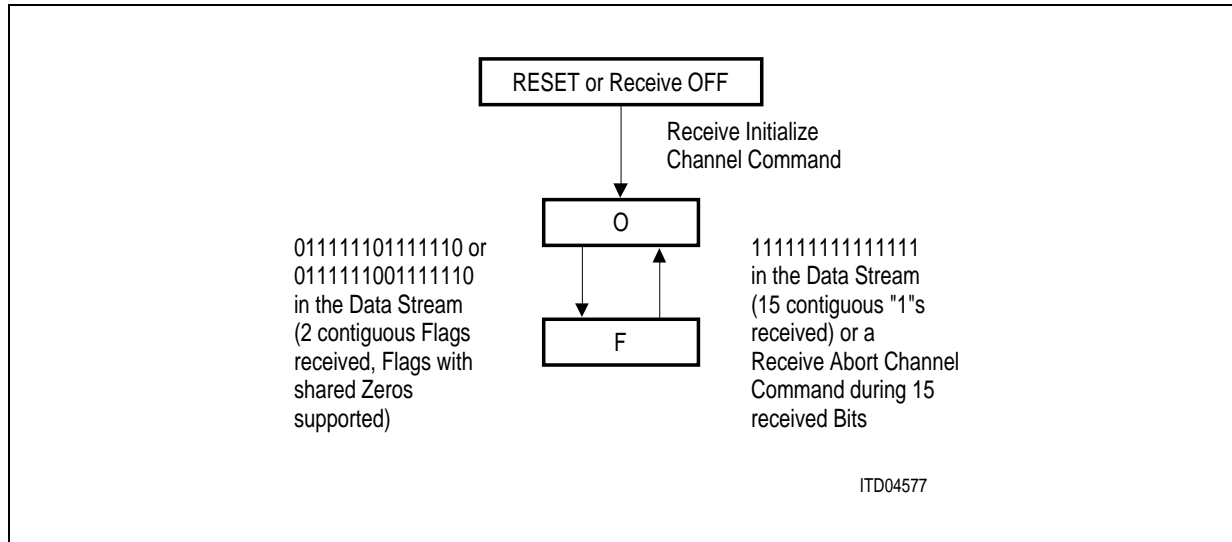
1. a. A frame is supposed to have started if after a sequence of 0111 1110 in the receive data stream neither  $FC_H$  nor  $FD_H$  nor  $7E_H$  has occurred. The frame is supposed to have started with the first bit after the closing '0' of the sequence.
- b. A frame is supposed to have stopped if a sequence of 0111 1110 or 0111 1111 is found in the data stream after the frame has started. The last bit of the frame is supposed to be the bit preceding the '0' in the above sequences. The cases of sequences 0111 1110 1111 111 and 0111 1110 0111 1111 are also supposed to be frames of bit length  $-1$  and  $0$  respectively.  
A frame is also supposed to have stopped if more than MFL bytes were received since the start of the frame and it wasn't stopped yet.
- c. The ending flag of a frame may be the starting flag of the next frame (shared flags supported).

## Functional Description

2. The receiver is always in one of two possible interframe time-fill states: to be called F and O.

The following diagram explains them.

A change from F to O and from O to F is reported by an IFC interrupt.



**Figure 43**

3. The '0' extraction is also carried out for the last 6 bits before the stopping sequence.
4. The last 16 (CRC = 0) or 32 (CRC = 1) bits of a frame (after extraction of the zero insertions are supposed to be the FCS of the remaining bits of the frame. (For the case of a frame with less than or equal to 16 or 32 bits respectively see discussion of 7). The FCS is always checked, the check is reported in the CRCO bit of the last receive descriptor of the frame  
CRCO = 1: FCS was incorrect  
CRCO = 0: FCS was correct.
5. The check is reported in the NOB bit in the last receive descriptor of the frame  
NOB = 1: The bit length of the frame was not divisible by 8.  
NOB = 0: The bit length of the frame was divisible by 8.  
If NOB = 1: The last access to a receive data section of the frame may contain erroneous bits and shouldn't be evaluated.
6. The check is reported in the LFD bit in the last receive descriptor of the frame.  
LFD = 1: The number of bytes was greater than MFL.  
LFD = 0: The number of bytes was smaller or equal to MFL.  
Only the bytes up to the  
MFL + 1<sup>st</sup> one for CS = 1  
MFL - 1<sup>st</sup> one for CS = 0, CRC = 0  
MFL - 3<sup>rd</sup> one for CS = 0, CRC = 1  
are transferred to be stored memory. The bytes of the last access may be erroneous and shouldn't be evaluated.

## Functional Description

7. For frames not fulfilling check a) no data are transferred to the shared memory irrespective of CS.  
Only an interrupt with the bit FI, SF and (possibly) ERR is generated.  
For frames fulfilling check a) but not check b) data is transferred to the shared memory but the SF bit in the last receive descriptor is set.
8. The check is reported in the RA bit in the last receive descriptor of the frame  
RA = 1: The frame was stopped by the sequence  $7F_H$   
RA = 0: The frame was not stopped by the sequence  $7F_H$ .  
*Note: A receive descriptor with RA = 1 may also result from a fast receive abort or a receive abort channel command or from a receive descriptor with set HOLD bit.*

## Options

The different options for this mode are:

- The kind of Frame Check Sequence (FCS)  
Two kinds of FCS are implemented and can be chosen by CRC bit.  
CRC = 0: the generator polynomial  $x^{16} + x^{12} + x^5 + 1$  is used (2 byte FCS of CCITT Q.921)  
CRC = 1: the generator polynomial  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$  (4 byte FCS) is used.
- the transfer of the FCS together with the received data is programmable by the CS bit.  
CS = 0: FCS is not transferred to the data section  
CS = 1: FCS is transferred to the data section.  
*Note: FCS is **always** checked irrespective of the CS bit.*

## Interrupts

The possible interrupts for the mode in receive direction are:

- HI: issued if the HI bit is detected in the receive descriptor (not maskable)
- FI: issued if a received frame has been finished as discussed in 1.b of the protocol features (also for frames which do not lead to data transfer as discussed in 7. of the protocol features)  
(maskable by FIR in the channel spec.)
- IFC: issued if a change of the interframe time-fill state as discussed in 2. has occurred.  
(maskable by IFC in the channel spec.)
- SF: a frame not fulfilling check a) has been detected (maskable by SFE in the channel spec.)

---

**Functional Description**

ERR: issued if one of the following error conditions has occurred:

- FCS was incorrect
- the bit length was greater than MFL
- the frame was stopped by 7F<sub>H</sub>
- the frame could only be partly stored because of internal buffer overflow of RB
- a fast receive abort channel command was issued
- a receive abort channel command was detected during reception of a frame
- a frame could only be partly transferred to the shared memory because of a receive descriptor with HOLD bit set (maskable by RE in the channel spec.)

FO: issued if due to inaccessibility of internal buffer RB

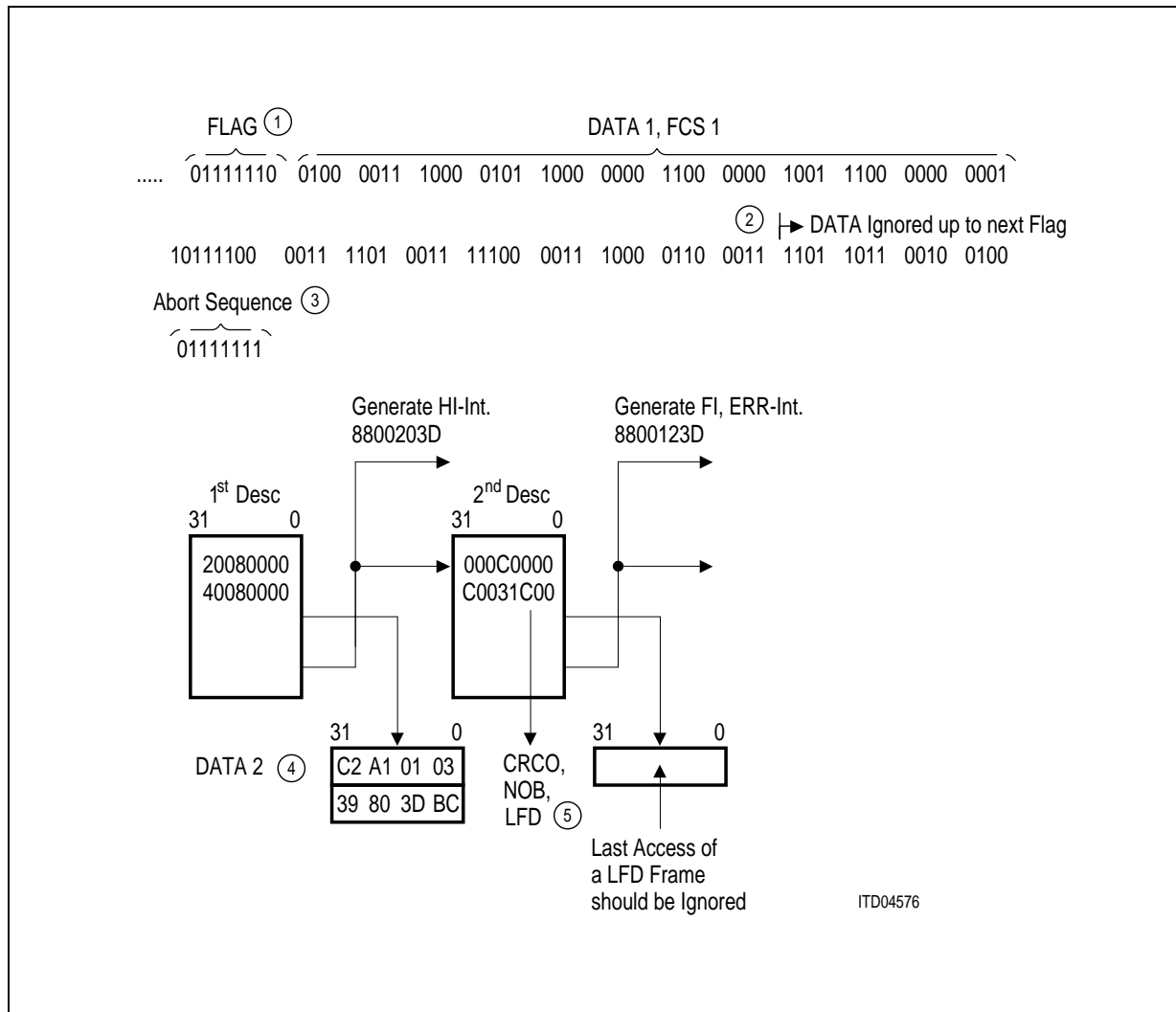
- one or more complete frames have been lost
- one or more changes of interframe time-fill state were lost (maskable by RE in the channel spec.)

**Example:**

HDLC channel with

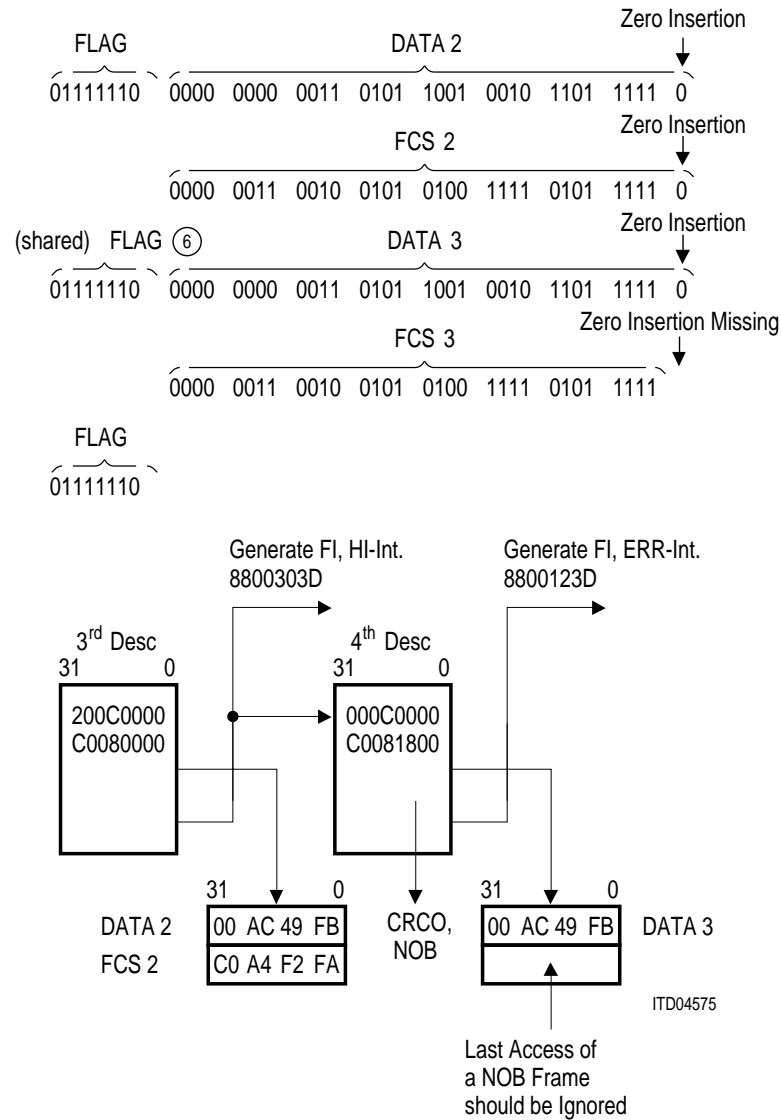
CS = 1           (FCS transferred to shared memory)  
INV = 0          (no inversion)  
CRC = 1          (CRC 32)  
TRV = 00       (required as unused in HDLC mode)  
FA = x           (irrelevant)  
MODE = 11       (HDLC)  
IFTF = x         (irrelevant)  
Motorola interface  
Channel No. 1D  
MFL = 10

## Functional Description



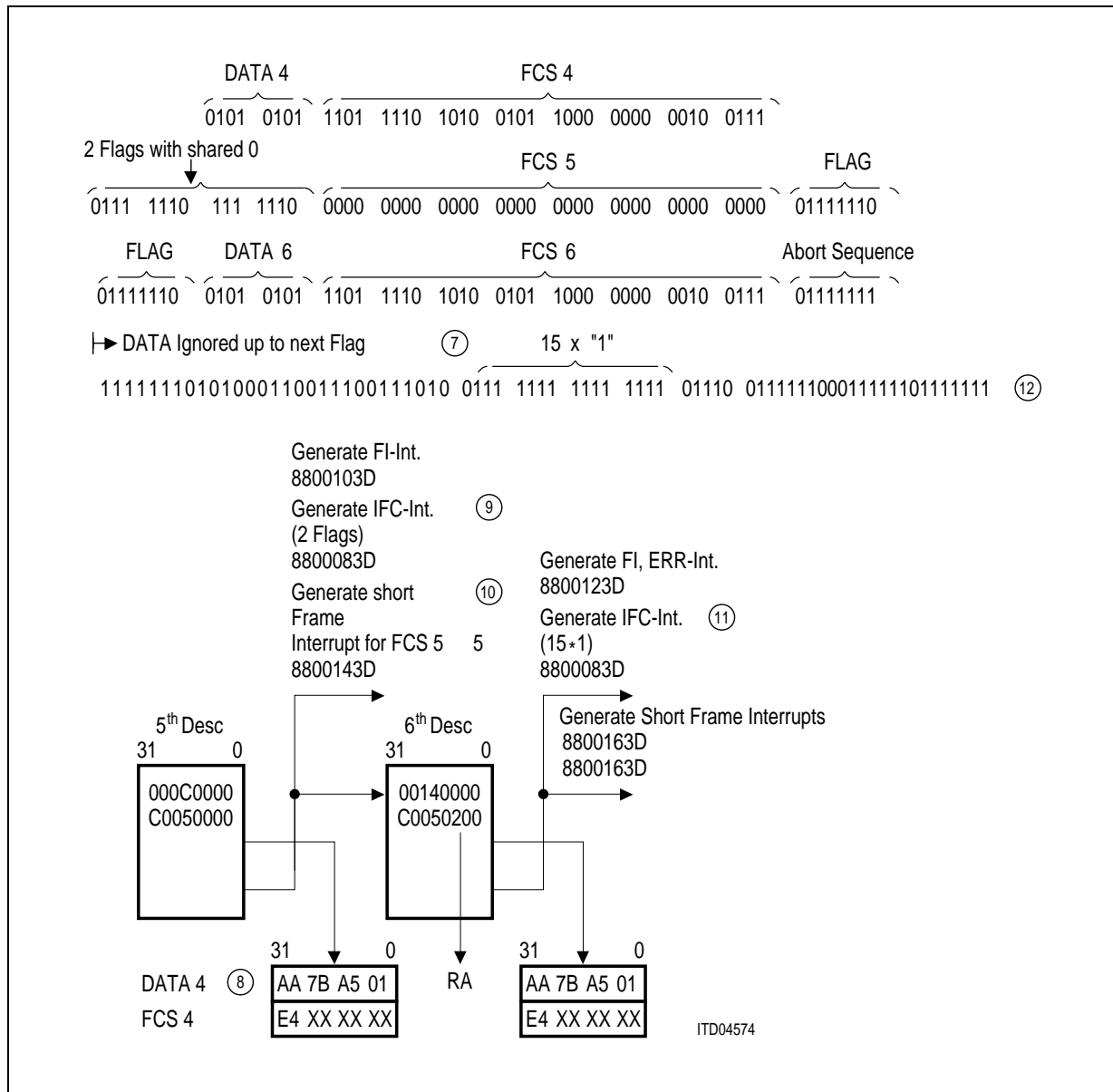
**Figure 44**

## Functional Description



**Figure 45**

## Functional Description



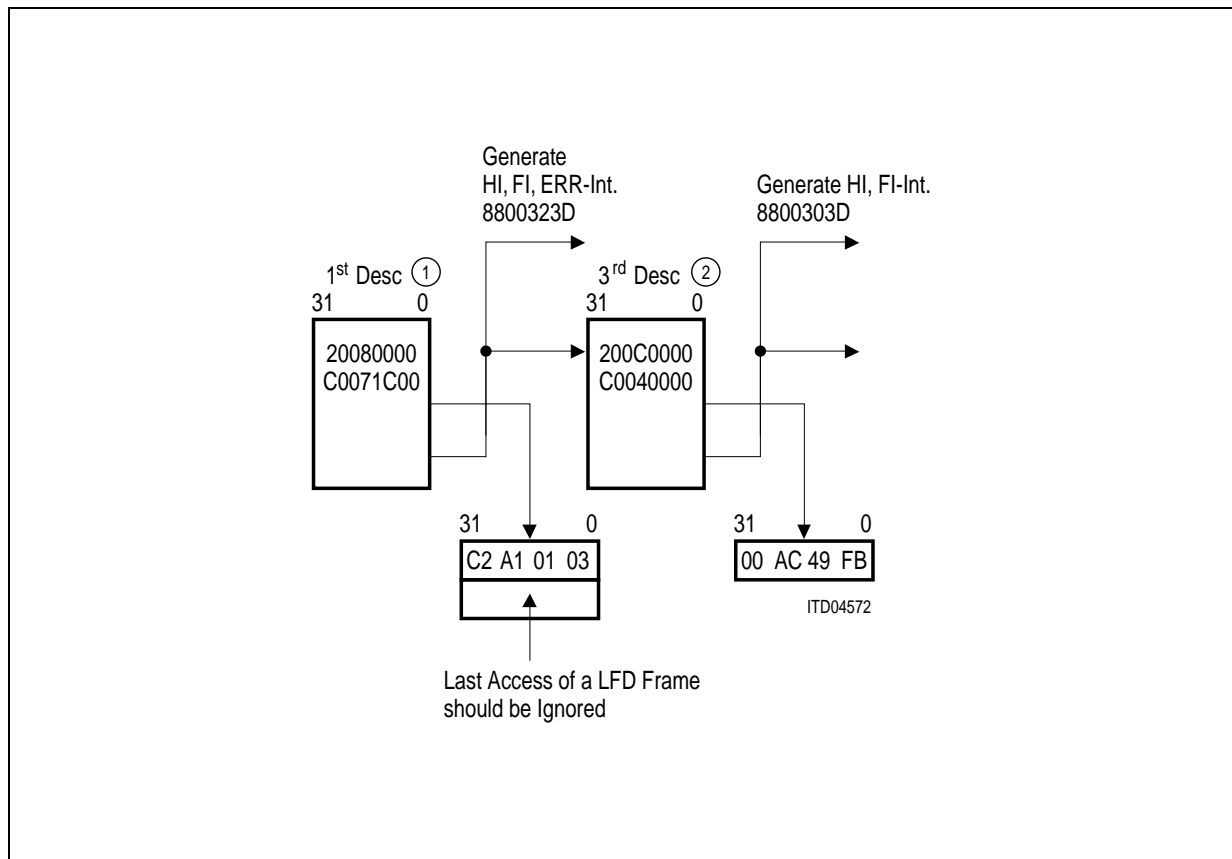
**Figure 46**

- Note: 1. After Receive Initialization is detected all data are ignored until a flag is received. The receiver is in the interframe time-fill state '0'.*
- 2. After MFL + 1 data bytes are received the further data are ignored (except for a change of the interframe time-fill state) and are neither stored in the RB nor reported to the shared memory. The receiver waits for the next flag.*
- 3. Even the abort sequence at the end of the frame will not lead to the RA bit in the descriptor to be set.*
- 4. Data are formatted according to §2.8 of CCITT Q.921.*
- 5. The FCS is formatted as ordinary data!!!*

## Functional Description

6. LFD is issued and always accompanied by NOB.  
CRCO shouldn't be interpreted for a LFD frame.
7. Here the ending flag of the second frame is the starting flag of the third frame.
8. After an abort sequence data is ignored until a flag is found (except for a change of the interframe time-fill state). They are neither stored in the RB nor reported to the shared memory.
9. The last 3 bytes in the last write access to the receive data section of the 5th descriptor have to be ignored.
10. The 2 flags with a shared 0 in the middle change the original interframe time-fill state '0' of the receiver to 'F'. The 2 flags following FCS 5 on the other hand do not change the interframe time-fill state, as it already was 'F'.
11. The frame consisting only of 32 times 0 between 2 flags does not pass check a). It only leads to an interrupt.
12. The  $15 \times '1'$  leads to a change of the interframe time-fill state from 'F' to '0' even through it is in a data ignored zone.
13. This frame of length - 1 leads to an interrupt.

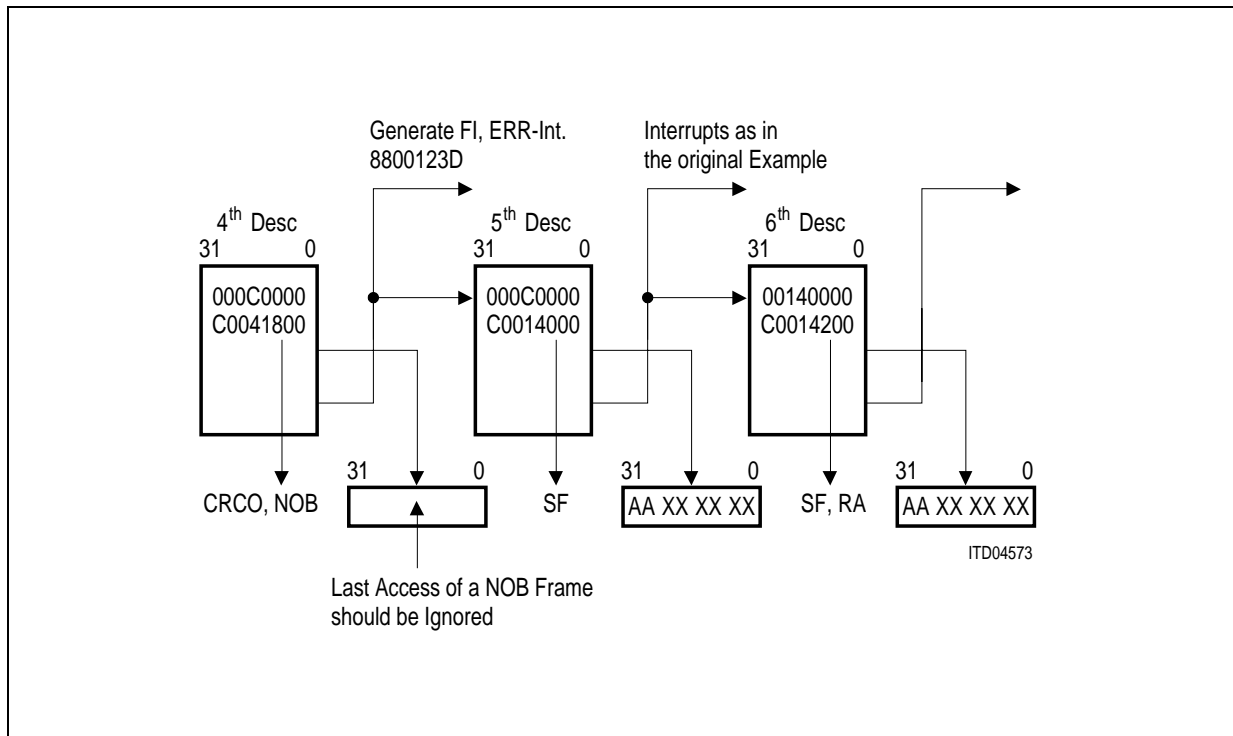
For CS = 0 (CRC not select) the descriptor would have looked like



**Figure 47**



## Functional Description



**Figure 48**

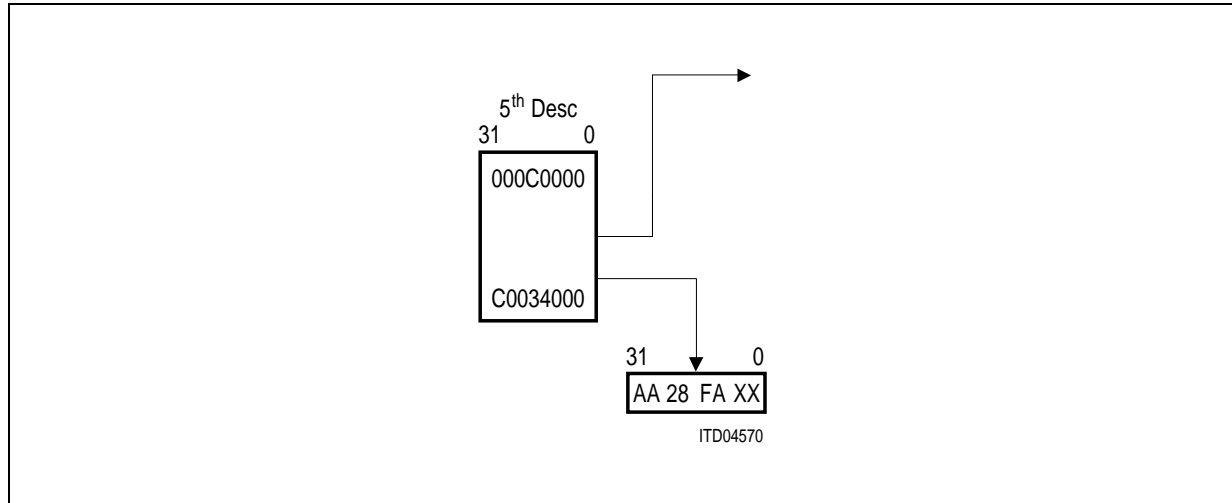
*Note: 1. Only the 7 leading bytes are reported (the last 4 are supposed to be the FCS even in this case).*

*2. It is assumed here for convenience that the first descriptor points to the third and not to the second descriptor as in the original example.*

For INV = 1 (channel inversion) all bits of the data stream (including DATA, FCS, flag, abort sequence  $15 \times '1'$ ) are interpreted inversely. e.g. '1000 0001' would be interpreted as flag  $15 \times '0'$  would lead to a change from interframe time-fill state 'F' to '0' etc.

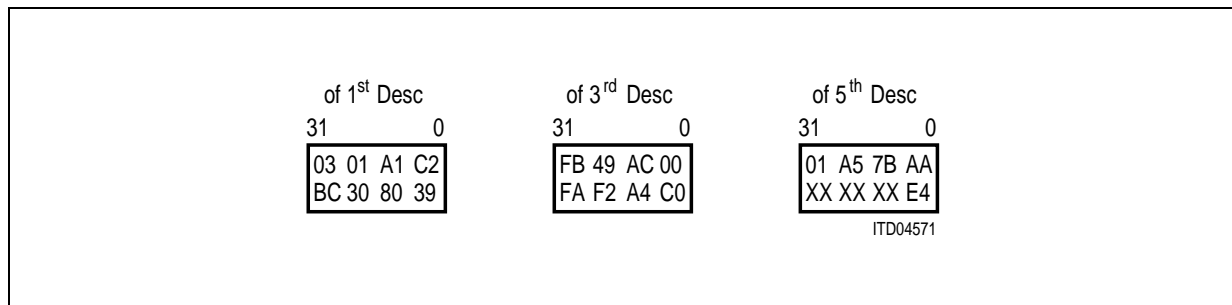
## Functional Description

For CRC = 0 (CRC 16) the correct FCS e.g. zeros for DATA 4 would be 00001 0100 0101 1110 the 5<sup>th</sup> descriptor would then be



**Figure 49**

For Intel interface the only difference is in the receive data sections. They would be



**Figure 50**

## TMB

### Transmit Direction

#### General Features

In transmit direction:

- The starting and ending flag (00<sub>H</sub> before and after a frame)
- The interframe time-fill between frames

is generated automatically.

#### Options

The different options for this mode are:

- The number of interframe time-fill characters as shown in **Figure 26** by choosing FNUM in the transmit descriptor. For the values FNUM = 0, 1, 2 we have

FNUM = 0    ... frame 1, 00<sub>H</sub>, frame 2 ... (start = end flag)

FNUM = 1    ... frame 1, 00<sub>H</sub>, 00<sub>H</sub>, frame 2 ...

FNUM = 2    ... frame 1, 00<sub>H</sub>, 00<sub>H</sub>, 00<sub>H</sub>, frame 2 ...

#### Interrupts

The possible interrupts for the mode in transmit direction are identical to those of HDLC.

A typical data stream has the form

ITF DATA ITF DATA

#### Example

TMB channel with

INV = 0        (no inversion)

CRC = 0        (required)

TRV = 00       (required)

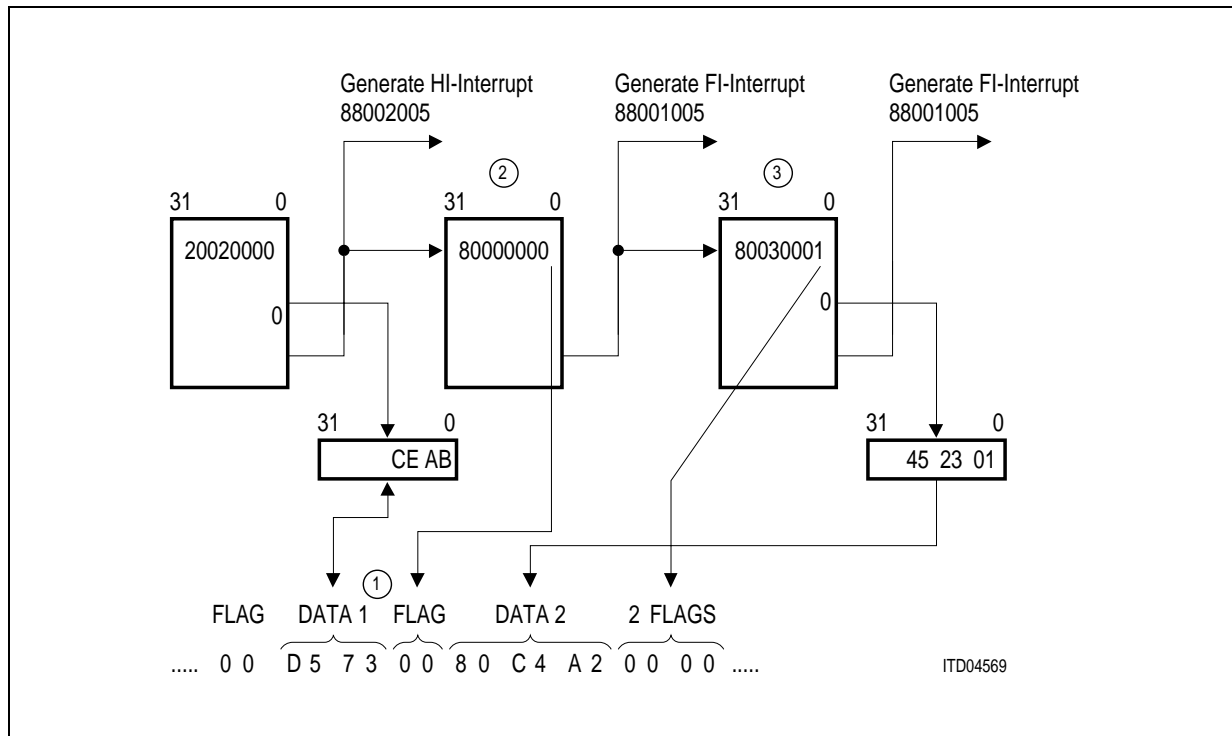
FA = 0         (required)

MODE = 01     (TMB)

IFTF = 0       (required)

Intel interface

Channel number 5



### Figure 51

*Note: 1. Data is transmitted according to Q.921 §2.8 and fully transparent.*

2. A transmit descriptor with  $NO = 0$  and  $FE = 1$  is allowed, one with  $NO = 0$  and  $FE = 0$  is forbidden.
3.  $FNUM = 1$  leads to 2  $FLAGS$  after  $DATA\ 2$ .

## TMB

### Receive Direction

#### General Features

1. The starting and ending flag ( $00_H$  before and after a frame) as well as interframe time-fill is recognized and extracted.
2. The number of bits within a frame is checked to be divisible by 8.
3. The number of bytes within a frame is checked to be smaller than  $MFL + 1$ .
4. A frame containing less than 8 bits may be ignored completely by the receiver.

More detailed description of the individual features:

1. a. A frame is supposed to have started if after a sequence '0000 0000' a '1'-bit is recognized. The frame is supposed to have this '1'-bit as first bit.  
b. A frame is supposed to have stopped if
  - either a sequence 0000 0000 1 is found in the data stream after the frame has started
  - or a sequence 0000 0000 is found octet synchronous (i.e. the first bit of the sequence  $00_H$  is the  $8m + 1^{st}$  bit since the starting '1'-bit of 1.a. for an integer  $m$ ).

In both cases the last bit before the sequence  $00_H$  is supposed to be the last bit of the frame.

2. The check is reported in the NOB bit in the last receive descriptor of the frame.  
NOB = 1: The bit length of the frame was not divisible by 8.  
NOB = 0: The bit length of the frame was divisible by 8.
3. The check is reported in the LFD bit in the last receive descriptor of the frame.  
LFD = 1: The number of bytes was greater than MFL.  
LFD = 0: The number of bytes was smaller or equal to MFL.  
Only the bytes up to the  $MFL + 1^{st}$  one are transferred to the shared memory. The bytes of the last access to the receive data section of the frame may contain erroneous bits and shouldn't be evaluated. LFD is always accompanied by NOB.

### Options

There are no options in receive direction for this mode.

---

## Functional Description

### Interrupts

The possible interrupts for the mode in receive direction are:

HI: issued if HI bit is detected in the receive descriptor (not maskable).

FI: issued if a received frame has been finished as discussed in 1b) of the protocol features or a receive abort channel command was detected during reception of a frame.

(maskable by FIR in the channel spec.)

ERR: issued if one of the following error conditions has occurred

- the bit length of the frame was not divisible by 8
- the byte length was greater than MFL
- the frame could only be partly stored because of internal buffer overflow of RB
- a fast receive abort channel command was issued
- the frame could only be partly transferred due to a receive descriptor with set HOLD bit.

(maskable by RE in the channel specification)

FO: issued if due to inaccessibility of the internal buffer RB one or more complete frames have been lost. (maskable by RE in the channel spec.)

### Example:

TMB channel with

INV = 0 (no inversion)

CRC = 0 (required)

TRV = 00 (required)

FA = 0 (required)

MODE = 01 (TMB)

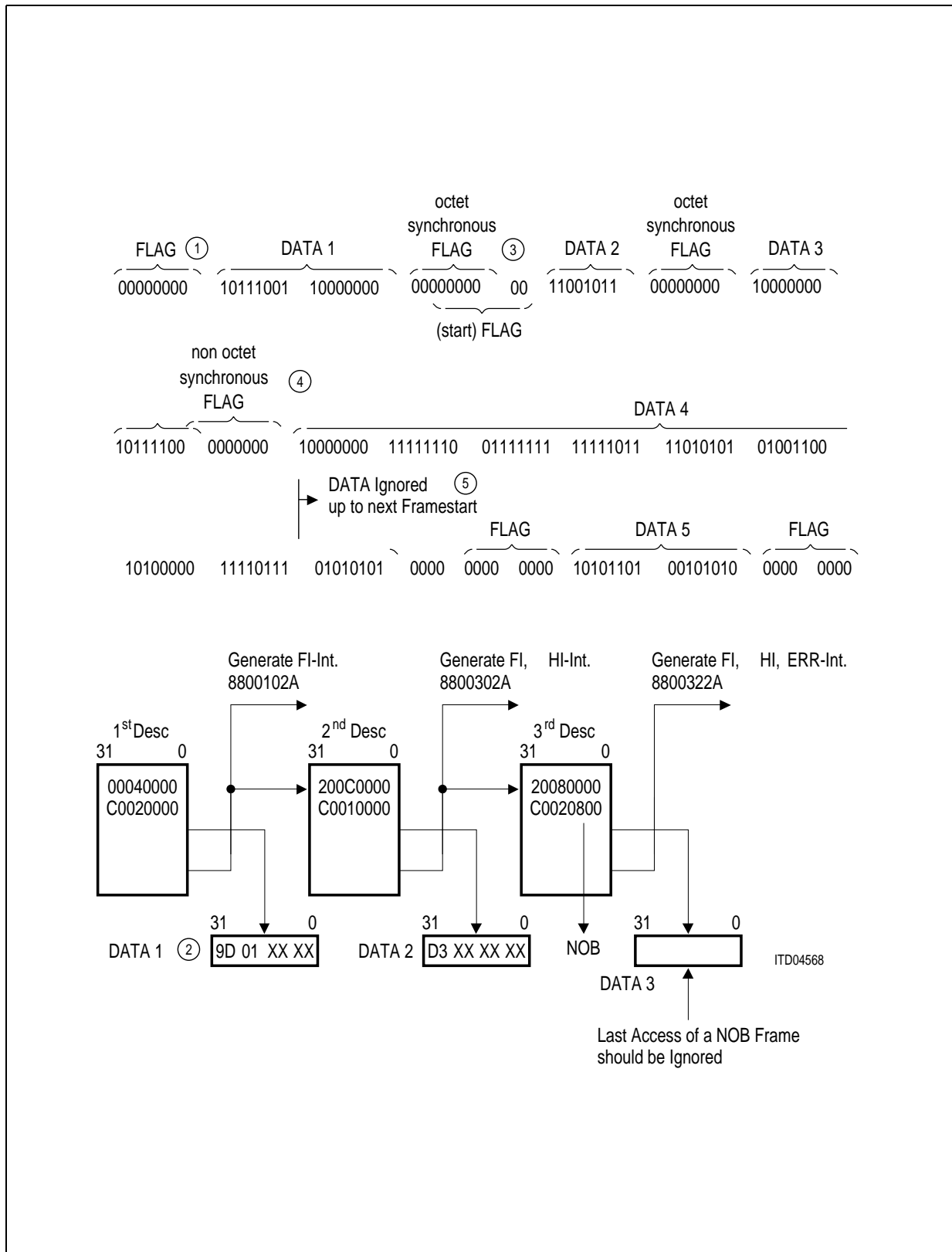
IFTF = 0 (required)

MFL = 7

Motorola interface

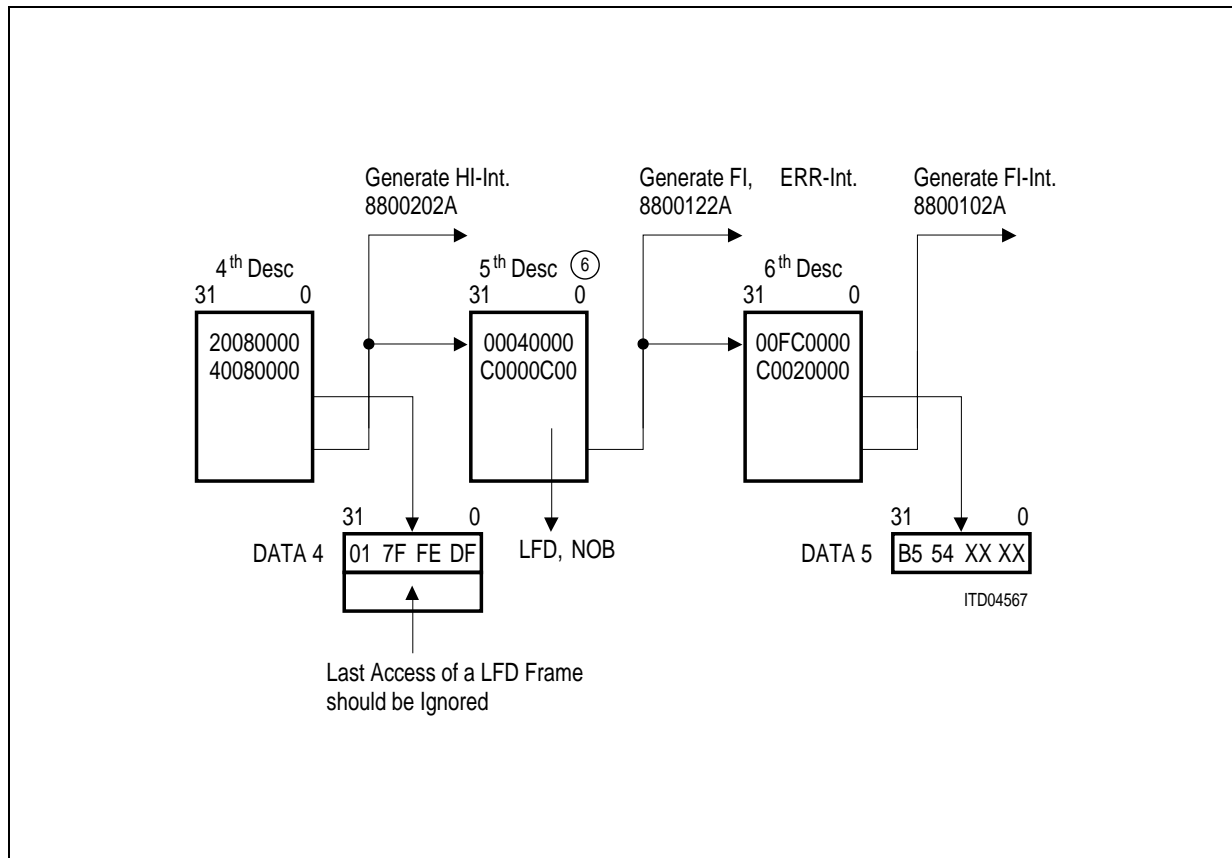
Channel No. A

## Functional Description



**Figure 52**

## Functional Description



**Figure 53**

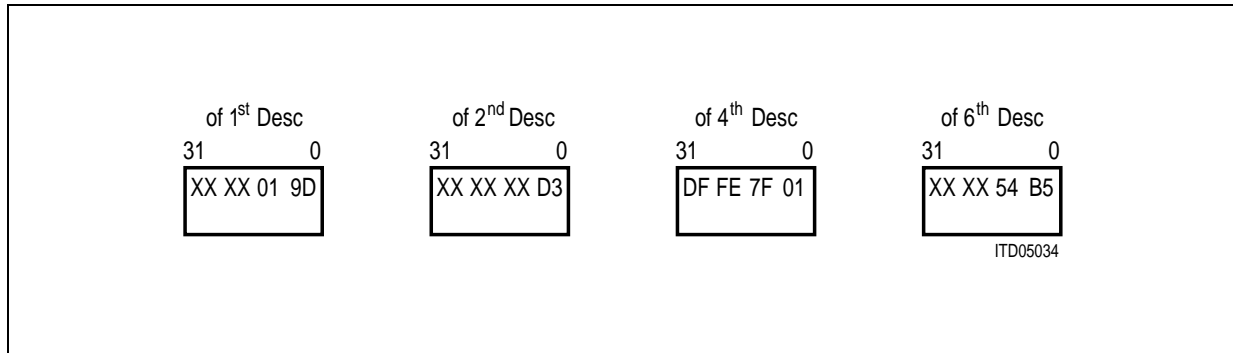
- Note:* 1. After Receive Initialization is detected all data are ignored until the starting sequence 0000 0000 1 is detected.
2. Data are formatted according to §2.8 of CCITT Q.921.
3. The octet synchronous (end) flag of one frame can be part of the (start) flag of the next frame. Between DATA 1 and DATA 3 they are identical (shared flags supported).
4. Here the sequence 0000 0000 1 is detected non-octet synchronously. Therefore the frame belonging to DATA 3 is supposed to have ended non-octet synchronously (NOB set in the 3<sup>rd</sup> descriptor).
5. After MFL + 1 data bytes the further data are ignored and are neither stored in the RB nor reported to the shared memory. The receiver waits for the next sequence 0000 0000 1 to come.
6. If a receive descriptor is full (4<sup>th</sup> desc.) the MUNICH32 branches to the next receive descriptor (5<sup>th</sup> desc.) even if no further data are to be given to the shared memory.



## Functional Description

For INV = 1 (channel inversion) all bits of the data stream (including DATA, FLAG) are interpreted inversely e.g. 1111 1111 0 would be interpreted as starting sequence then.

For Intel interface the only difference is in the receive data sections. They would be



**Figure 54**

---

## Functional Description

### TMR

#### Transmit Direction

##### General Features

In transmit direction

- the starting and ending flag (00 00<sub>H</sub> or 0 00<sub>H</sub> between frames) is generated automatically.

##### Options

The different options for this mode are

- the number of interframe time-fill characters as shown in **Figure 29** by choosing FNUM in the transmit descriptor. For the values 0, 1, 2 we have

FNUM = 0    ... frame 1, 000<sub>H</sub>, frame 2 ...

FNUM = 1    ... frame 1, 00<sub>H</sub>, 00<sub>H</sub>, frame 2 ...

FNUM = 2    ... frame 1, 00<sub>H</sub>, 00<sub>H</sub>, 00<sub>H</sub>, frame 2 ...

By choosing FNUM = 0 and setting the last transmitted nibble in the transmit data section to 0<sub>H</sub> frames of effective length  $n + 1/2$  bytes can be sent as required by GSM 08.60.

##### Interrupts

The possible interrupts for the mode in the transmit direction are identical to those of HDLC.

A typical data stream has the form

ITF DATA ITF DATA

##### Example:

TMR channel with

INV = 0            (no inversion)

CRC = 1            (required)

TRV = 00          (required)

FA = 0            (required)

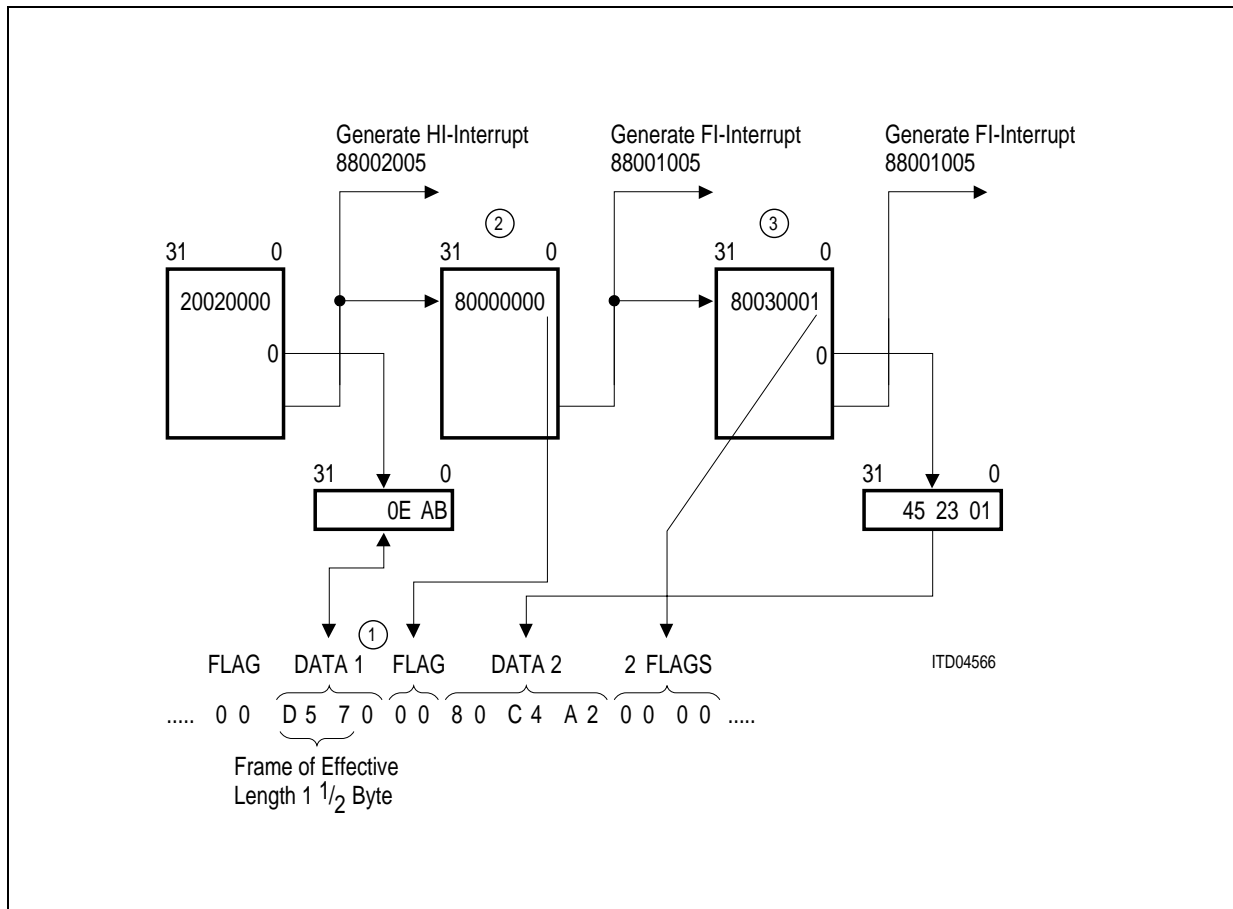
MODE = 01        (TMR)

IFTF = 0          (required)

Intel interface

Channel No. 5

## Functional Description



### Figure 55

*Note: 1. Data is transmitted according to Q.921 §2.8 and fully transparent.*

2. A transmit descriptor with  $NO = 0$  and  $FE = 1$  is allowed, one with  $NO = 0$  and  $FE = 0$  is forbidden.

3. *FNUM* = 1 leads to 2 *FLAGS* after *DATA* 2.

## TMR

### Receive Direction

#### General Features

1. The starting and the ending flag (00 00<sub>H</sub>) is recognized. Interframe time-fill, both characters of the starting flag and the last character of the ending flag is extracted.
2. The number of bits within a frame is checked to be divisible by 8.
3. The number of bytes within a frame is checked to be smaller than MFL.

More detailed description of the individual features

1. a. A frame is supposed to have started after a sequence of 16 zeros a '1'-bit is recognized. The frame is supposed to have this '1'-bit as first bit.  
b. A frame is supposed to have stopped if
  - either a sequence of 16 'zeros' and a 'one' is found in the data stream after the frame has started
  - or a sequence of 16 zeros is found octet synchronous (i.e. the first bit of the sequence 00 00<sub>H</sub> is the  $8m + 1^{\text{st}}$  bit since the starting '1'-bit of 1.a. for an integer m).

In both cases the eighth bit of the sequence 00 00<sub>H</sub> is supposed to be the last bit of the frame.

2. The check is reported in the NOB bit in the last receive descriptor of the frame.  
NOB = 1 the bit length of the frame was not divisible by 8.  
NOB = 0 the bit length of the frame was divisible by 8.  
If NOB = 1 the last byte of the last access to a receive data section of the frame may contain erroneous bits and shouldn't be evaluated. This does **not** affect the reception of frames with  $n + 1/2$  octets
3. The check is reported in the LFD bit in the last receive descriptor of the frame.  
LFD = 1 the number of bytes was greater than MFL.  
LFD = 0 the number of bytes was smaller or equal to MFL.  
MFL + 1<sup>st</sup> one are transferred to the shared memory. The bytes of the last access to the receive data section of the frame may contain erroneous bits and shouldn't be evaluated.  
LFD is always accompanied by NOB.

---

## Functional Description

### Options

There are no options in receive direction for this mode.

### Interrupts

The possible interrupts for the mode in receive direction are identical to those of TMB.

### Example:

TMR channel with

INV = 0 (no inversion)

CRC = 1 (required)

TRV = 00

FA = 0

MODE = 01 (TMR)

IFTF = 0 (required)

MFL = 7

Motorola interface

Channel No. 15

## Functional Description

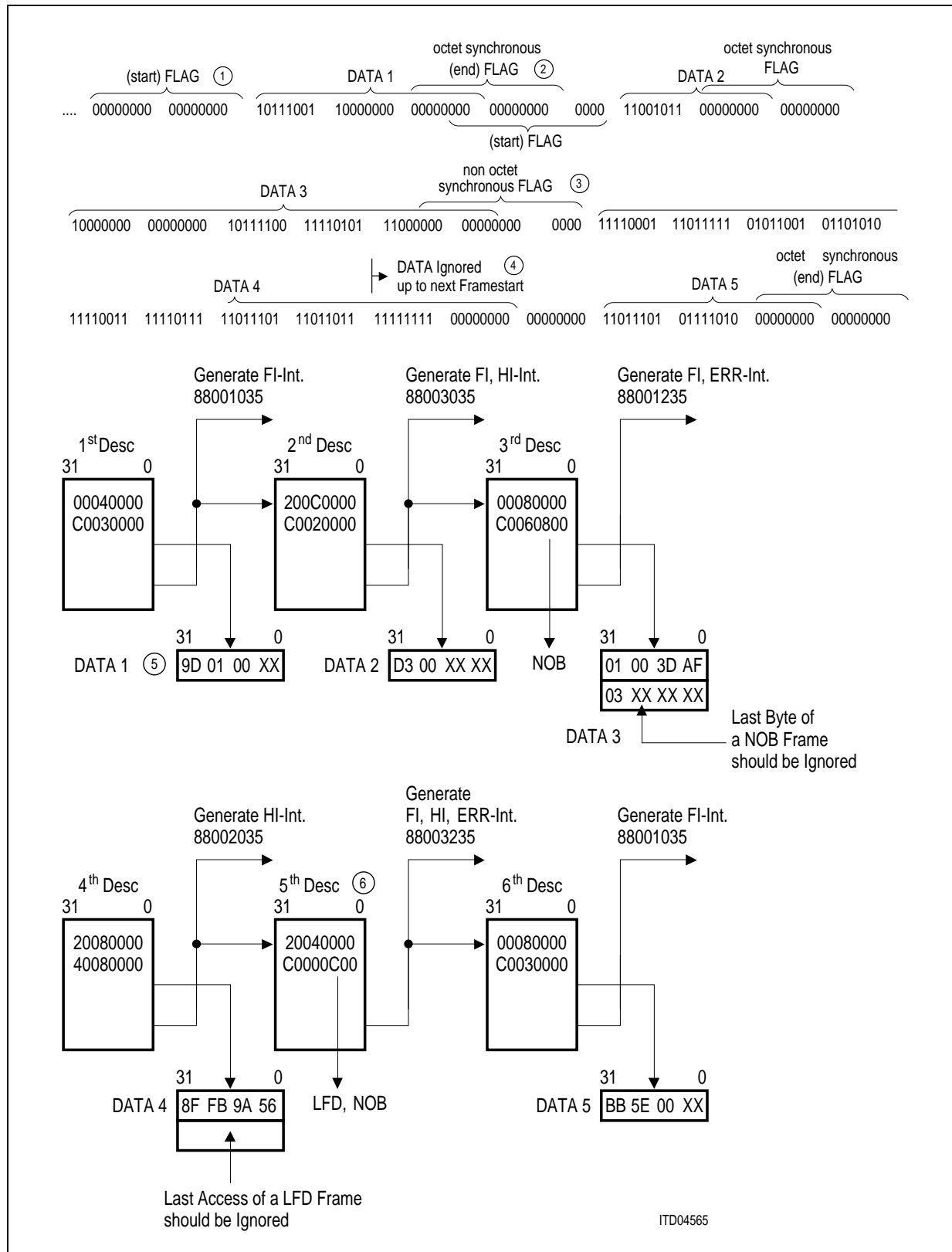


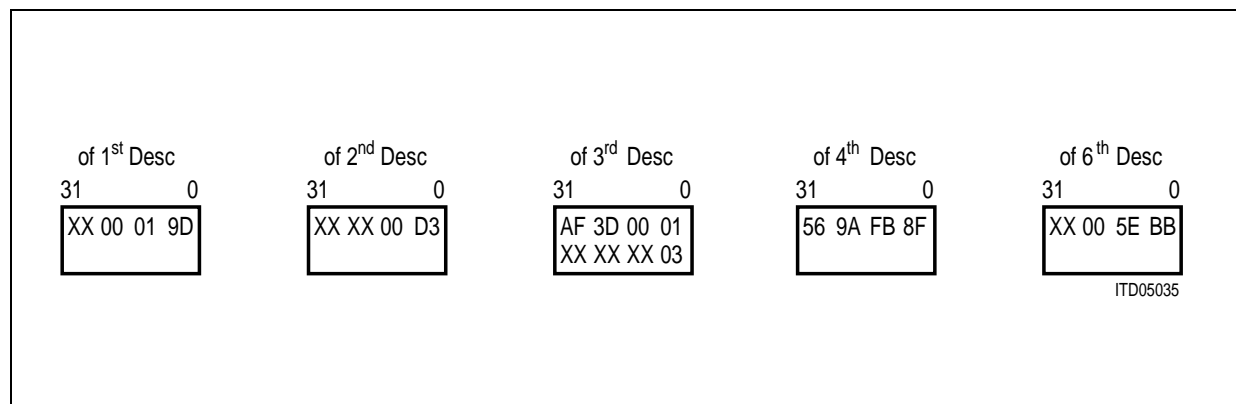
Figure 56

## Functional Description

1. After receive initialization is detected all data are ignored until a starting sequence (16 'zeros', 'one') is detected.
2. The octet synchronous (end) flag of one frame can be part of the (start) flag of the next frame.  
Note, that the first 00<sub>H</sub> character of the end flag is stored in the receive data section as ordinary data and is included in BNO.  
Between DATA 2 and DATA 3 the start and end flag are identical (shared flags supported).
3. Here the start sequence is detected non-octet synchronously within a frame. Therefore the frame belonging to DATA 3 is supposed to have ended non-octet synchronously (NOB set in the 3<sup>rd</sup> descriptor).
4. After MFL + 1 data bytes the further data are ignored and are neither stored in the RB nor reported to the shared memory.
5. Data are formatted according to §2.8 of CCITT Q.921.
6. If a receive descriptor is full (4<sup>th</sup> descriptor) the MUNICH32 branches to the next receive descriptor (5<sup>th</sup> descriptor) even if no further data are to be given to the shared memory.

For INV = 1 (channel inversion) all bits of the data stream (including DATA, FLAG) are interpreted inversely e.g. 16 'ones', 'zero' is interpreted as starting sequence then.

For Intel interface the only difference is in the receive data sections. They would be



**Figure 57**

## TMA

### Transmit Direction

#### General Features

In the transmit direction

- a slot-synchronous transparent data transmission
- a high impedance overwrite for the masked bits in the slot
- a programmable number of programmable fill characters between data (also slot synchronous)

is generated automatically.

#### Options

The different options for this mode are

- The value of the fill-character can be programmed for  $FA = 1$  in the channel specification. The fill-character (TC) is then programmed in the TFLAG. For  $FA = 0$  the fill character is  $FF_H$  and TFLAG has to be set to  $00_H$ . If subchanneling is chosen (not all fill/mask bits of the channel are '1') FA must be set to '0'.
- The number of inter-data time-fill characters as shown in **Figure 33** by choosing  $FNUM = 0, 1, 2$  we have

$FNUM = 0$     ... DATA 1, TC, DATA 2 ...  
 $FNUM = 1$     ... DATA 1, TC, TC, DATA 2 ...  
 $FNUM = 2$     ... DATA 1, TC, TC, TC, DATA 2 ...

#### Interrupts

The possible interrupts for this mode in transmit direction are identical to those of HDLC.



## Functional Description

### Example 1:

(no subchanneling by fill/mask bits)

TMA channel with

TFLAG = B2<sub>H</sub>

INV = 0 (no data inversion)

CRC = 0 (required)

TRV = 00 (required)

FA = 1 (flag filtering)

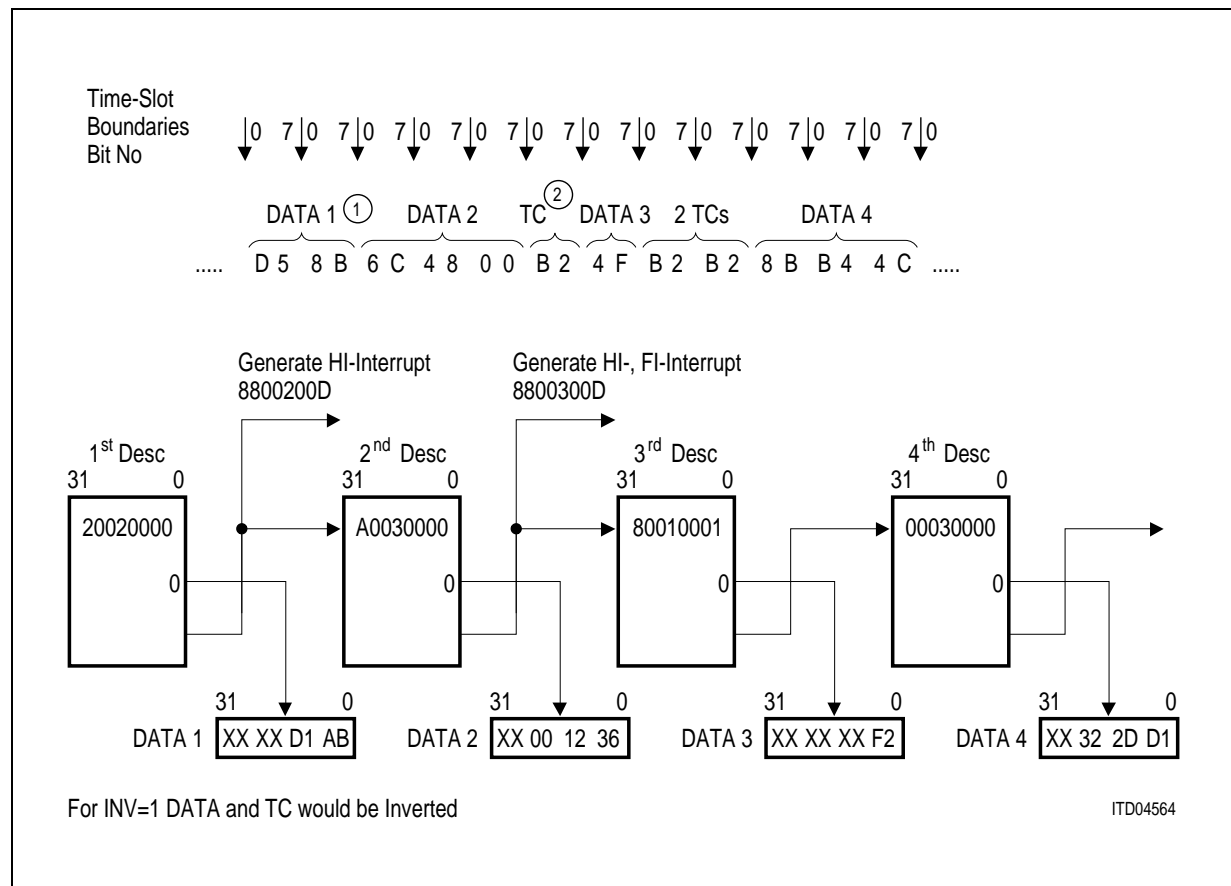
MODE = 00 (TMA)

IFTF = 0 (required)

All fill-mask bits are '1' for this channel (no high impedance overwrite)

Intel interface

Channel no. D



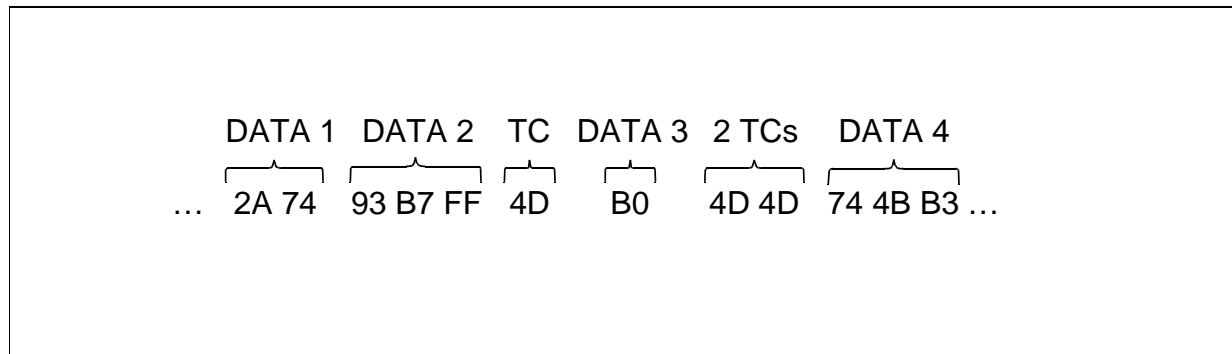
**Figure 58**

*Note: 1. Data are formatted according to §2.8 of Q.921. The TC is transmitted MSB (bit 15) first though!!!*

*2. FNUM = 0 in the second descriptor leads to the insertion of the TC after DATA 2, FNUM = 1 in the third descriptor to the insertion of 2 TCs.*

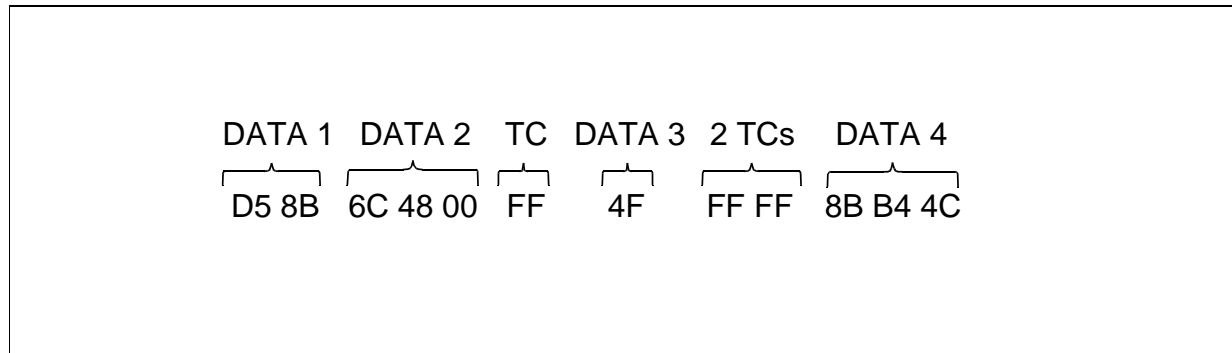
## Functional Description

For INV = 1 the data stream would be inverted completely



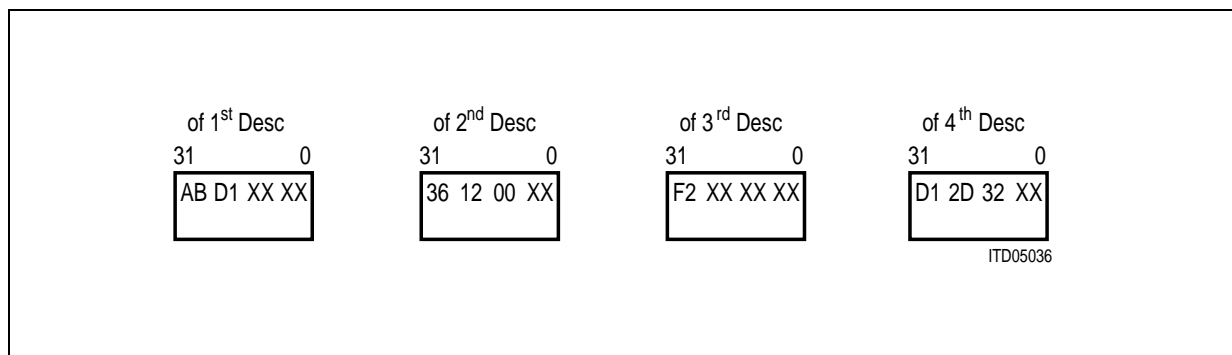
**Figure 59**

For FA = 0 TFLAG has to be programmed to 00<sub>H</sub> and the data stream would be



**Figure 60**

For Motorola mode the data sections leading to the same data stream would have been



**Figure 61**

## Functional Description

### Example 2:

(subchanneling by fill/mask bits)

TMA channel with

TFLAG = 00<sub>H</sub> (required for this case)  
 INV = 0 (no data inversion)  
 CRC = 0 (required)  
 TRV = 00 (required)  
 FA = 0 (required for subchanneling)  
 MODE = 00 (TMA)  
 IFTF = 0 (required)

Intel interface

Channel no. D

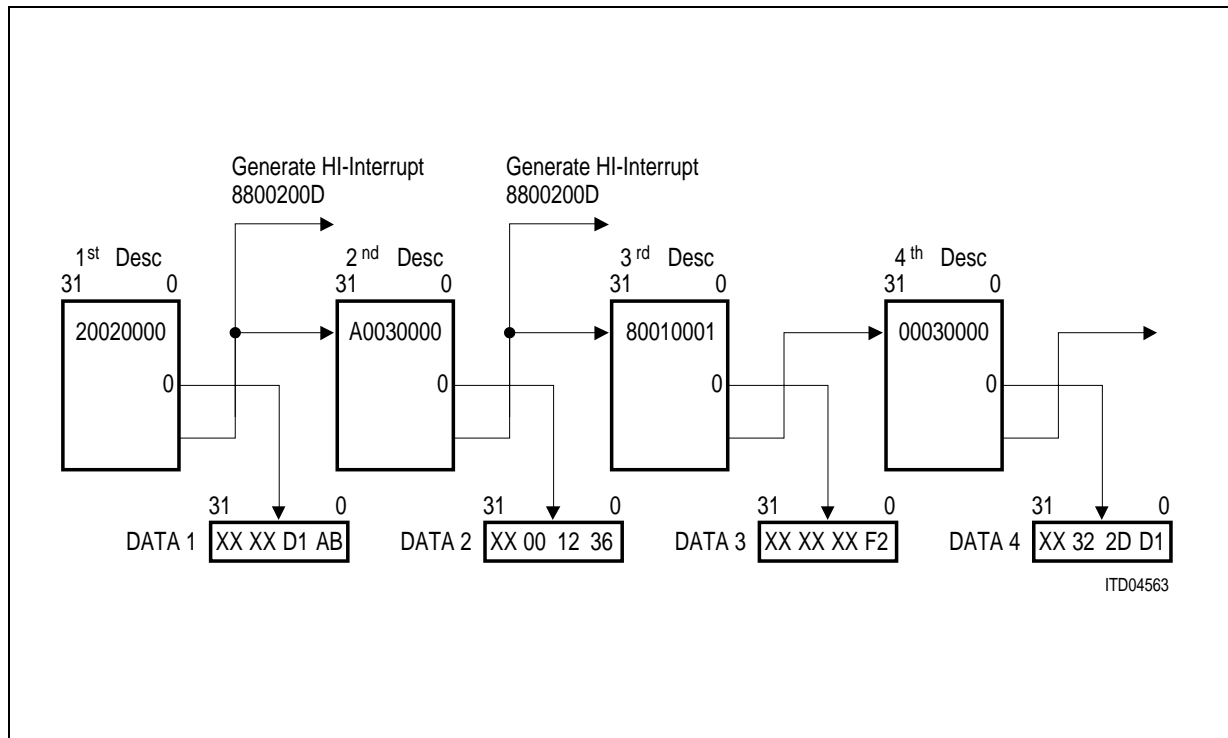
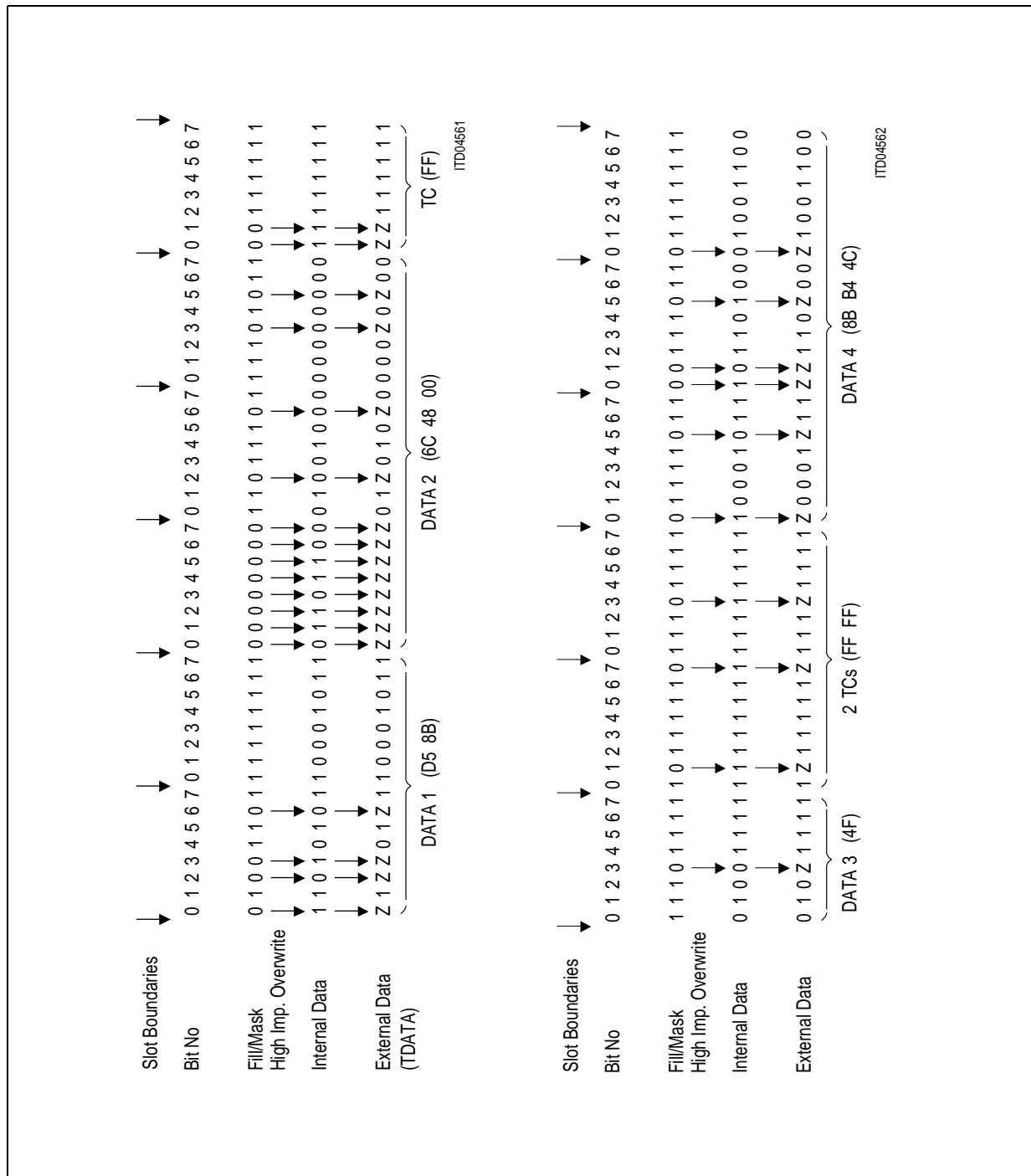


Figure 62



### Figure 63

*Note: Example 2 uses the same descriptors as example 1. Those bits in the data stream that are at places where fill/mask is 'zero' are overwritten by 'Z' i.e. high impedance. In all other protocols bits of the data stream are not overwritten by fill/mask zero bits.*

*Instead the whole data stream is sent at fill/mask one bits for all other protocols.*

## TMA

### Receive Direction

#### General Features

In the receive direction

- a slot synchronous transparent data reception
- a '1' overwrite for masked bits in the slot
- for FA = '1' a slot synchronous programmable flag extraction

is performed automatically.

#### Options

The different options for this mode are:

- the programmable character TC to be extracted for FA = '1' is TFLAG. For FA = '0' nothing is extracted. If subchanneling is chosen (not all fill/mask bits of the channel are '1') FA must be set to '0'.

#### Interrupts

The possible interrupts for the mode in receive direction are:

HI: issued if the HI bit is detected in the receive descriptor (not maskable).

ERR: issued if a fast receive abort channel command was issued.  
(maskable by RE in the channel spec.)

FO: issued if data could only partially stored due to internal buffer overflow of RB.  
(maskable by RE in the channel spec.)

#### Example 1:

(no subchanneling)

TMA channel with

TFLAG = D7

INV = 0 (no channel inversion)

CRC = 0 (required)

TRV = 00 (required)

FA = 1

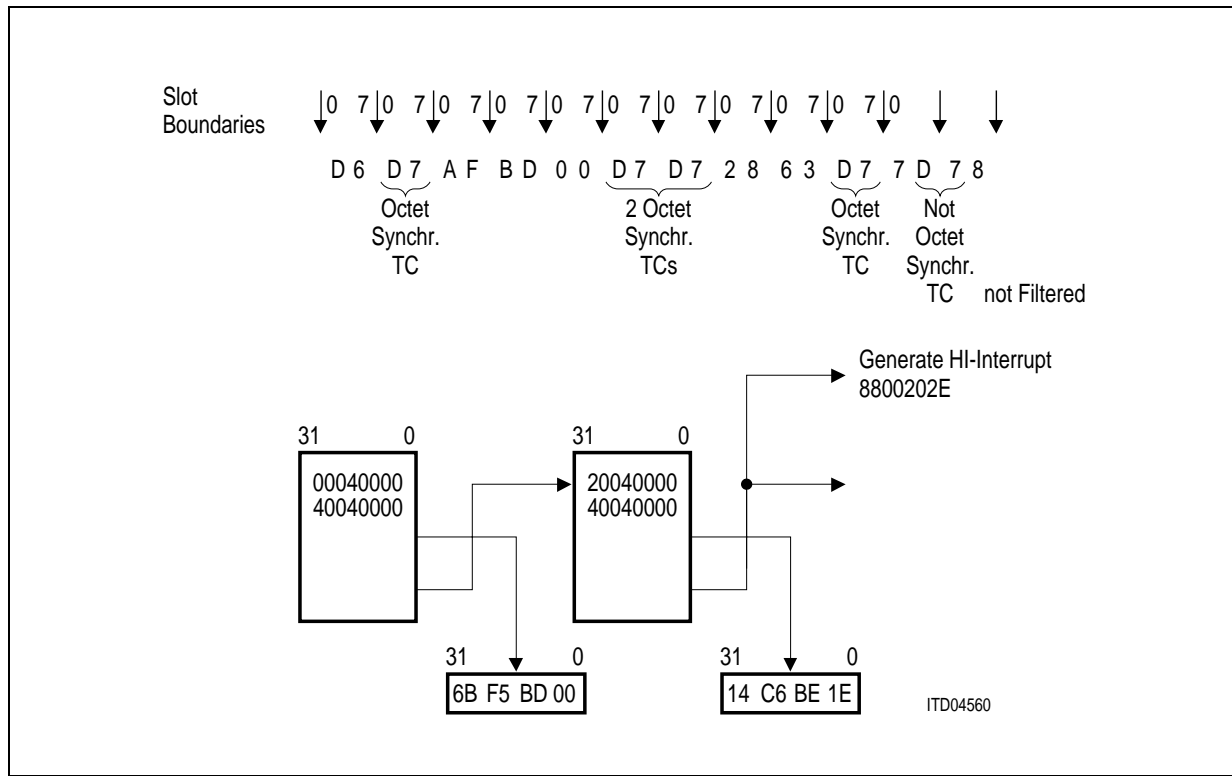
MODE = 00 (TMA)

IFTF = 0

Motorola interface

Channel No. E

## Functional Description

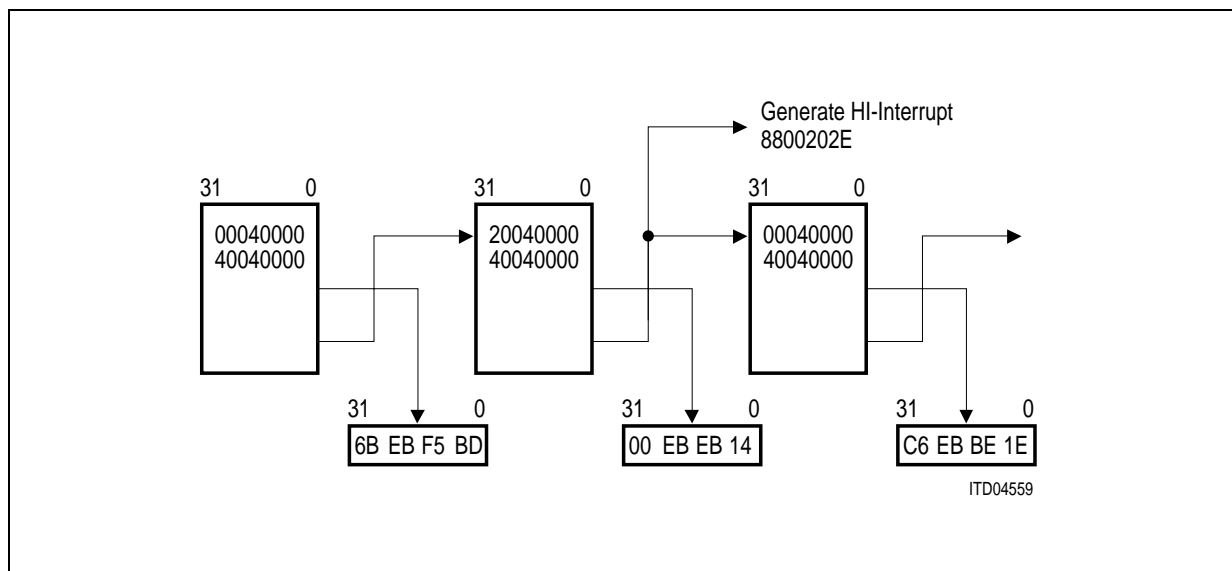


**Figure 64**

*Note: The FE bit is **never** set in a receive descriptor.  
The data are formatted according to §2.8 Q.921.*

For FA = 0 (and therefore TFLAG = 00<sub>H</sub>)

The descriptor would be



**Figure 65**

---

**Functional Description**

For INV = 1 the receiver filters the inverse of the TFLAG as TC out of the data stream and inverts the data (only the octet synchronous 28<sub>H</sub> would be filtered).

For Intel interface the data sections would be

00	BD	F5	6B
----	----	----	----

for the first descriptor and

1E	BE	C6	14
----	----	----	----

for the second.

**Example 2:**

(with subchanneling)

TMA channel with

TFLAG = 00<sub>H</sub> (required because of subchanneling)

INV = 0 (no channel inversion)

CRC = 0 (required)

TRV = 00 (required)

FA = 0 (required because of subchanneling)

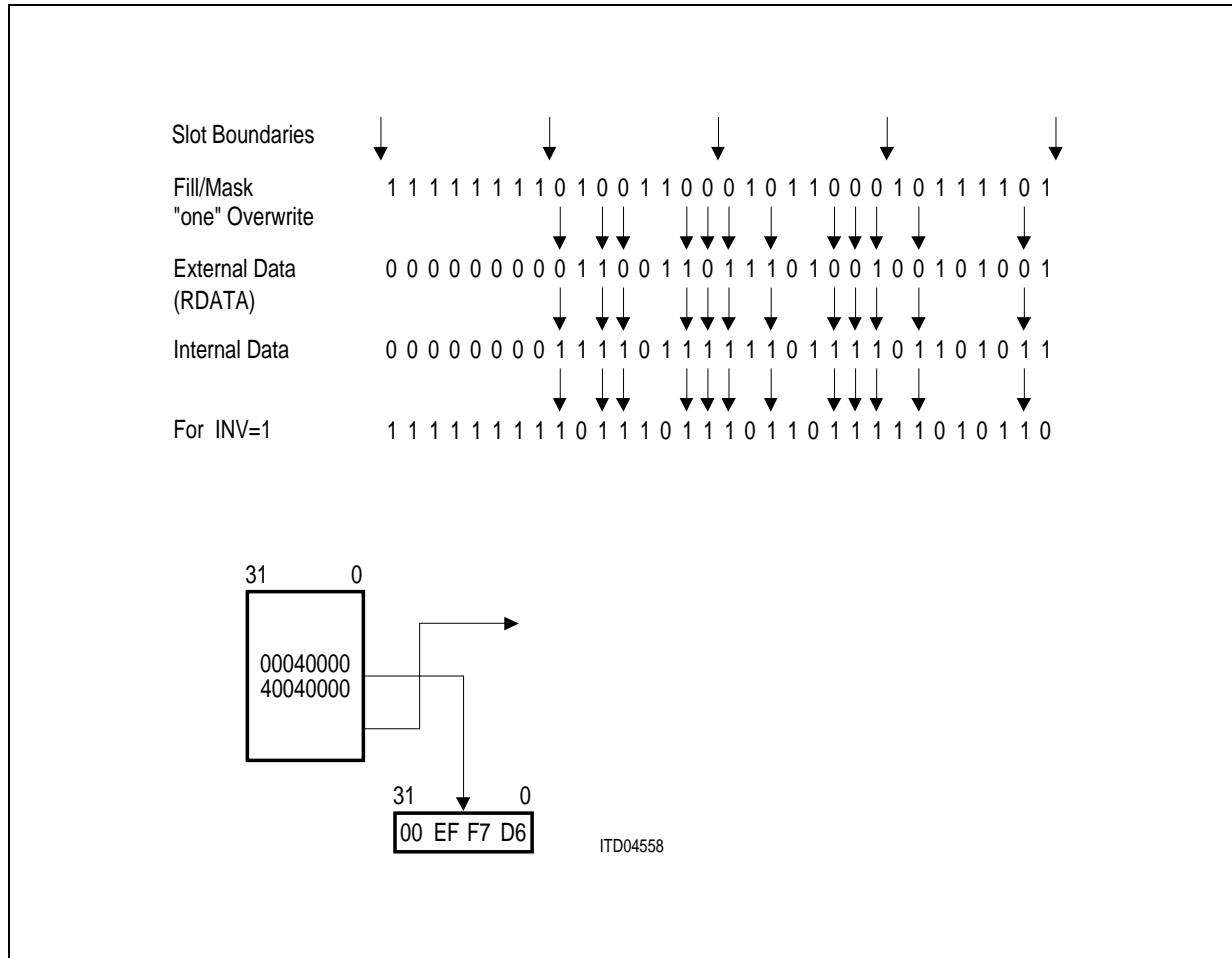
MODE = 00 (TMA)

IFTF = 0

Motorola interface

Channel No. E

## Functional Description



**Figure 66**



## Functional Description

### V.110/X.30

#### Transmit Direction

##### General Features

In transmit direction

- the synchronization pattern for V.110/X.30 frame as shown in **Table 1**.
- the framing for the different data rates with programmable E-, S-, X-bits
- sending '0' before all frames

is performed automatically.

**Table 1**  
**Synchronization Pattern for V.110/X.30-Frames**

Octet No.	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	1							
3	1							
4	1							
5	1							
6	1							
7	1							
8	1							
9	1							
10	1							

The E-, S-, X-bits are fed into the data stream by special transmit descriptor (as shown in **Figure 30**), they can only change from one 10-octet frame to the next, not within a 10-octet frame.

The data from the data sections are supposed to come in the form:

31 0  
1 1 B6 B5 B4 B3 B2 B1 1 1 B12 B11 B10 B9 B8 B7 1 1 B18 B17 B16 B15 B14 B13 1 1 B24 B23 B22 B21 B20 B19  
(for Motorola mode),

31 0  
1 1 B24 B23 B22 B21 B20 B19 1 1 B18 B17 B16 B15 B14 B13 1 1 B12 B11 B10 B9 B8 B7 1 1 B6 B5 B4 B3 B2 B1  
(for Intel mode).

where for 600 bit/s e.g. B1 to B6 belong to the first 10-octet frame, B7 to B12 belong to the second 10-octet frame, etc.

---

## Functional Description

### Options

The different options for this mode are:

- the framing pattern, as shown in **Table 2** to **Table 5**, is programmed by the bits TRV.

### Interrupts

HI: issued if the HI bit is detected in the transmit descriptor (not maskable)

ERR: if one of the following transmit errors has occurred

- the last descriptor had FE = 1 (leads to an abort of the transmit data, see **Figure 31**)
- the last descriptor had H = 1 (see **Figure 29**)
- the last descriptor had NO = 0  
(maskable by TE in the channel spec.)

FO: one of the following transmit errors has occurred

- a BERR = '0' was detected during a read access to a transmit data section for this channel
- the MUNICH32 was unable to access the shared memory in time either for new data to be sent or for a new descriptor.  
(maskable by TE in the channel spec.)

## Functional Description

### Example

X.30/V.110 channel with

CS = 0 (required)

INV = 0

CRC = 0

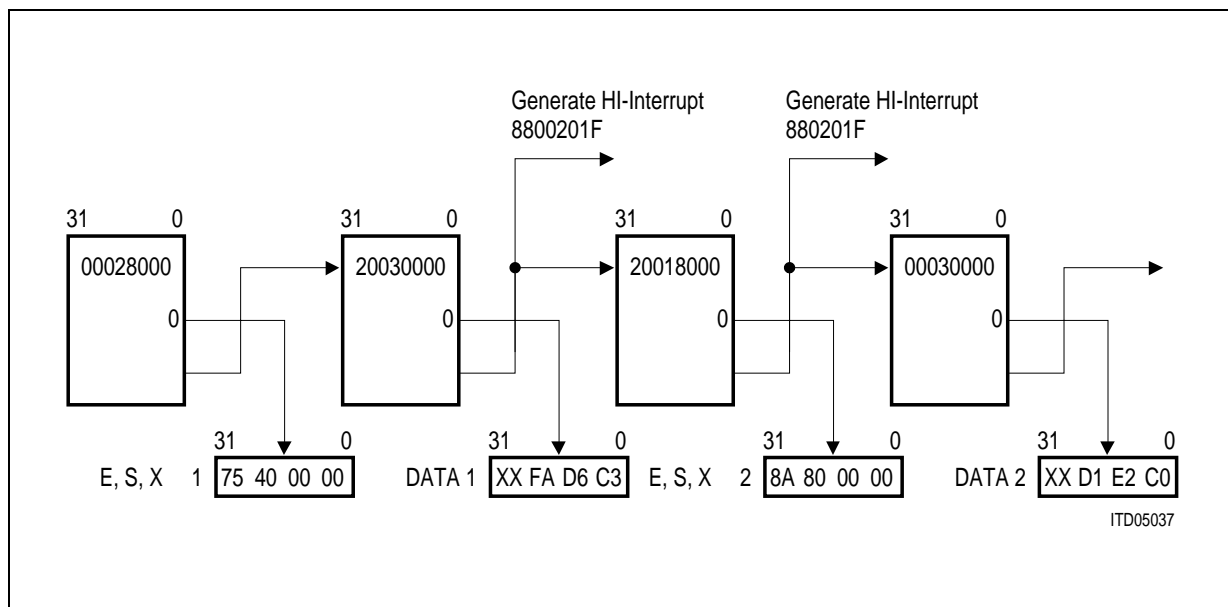
TRV variable (all values shown in examples)

FA = 0 (required)

MODE = 10 (V.110/X.30)

Intel interface

Channel No. 1F

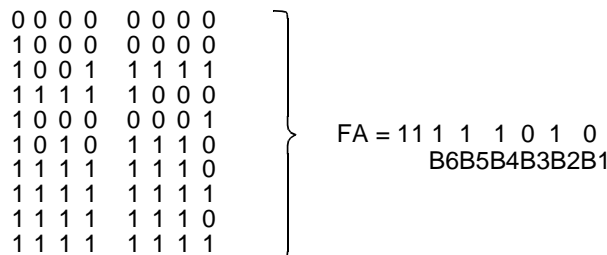
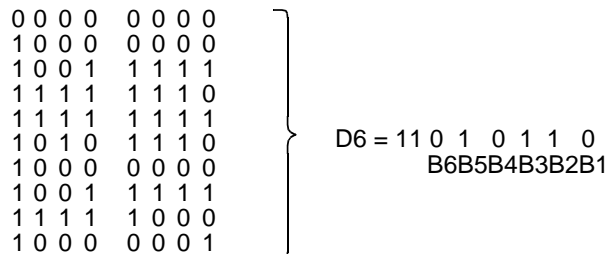
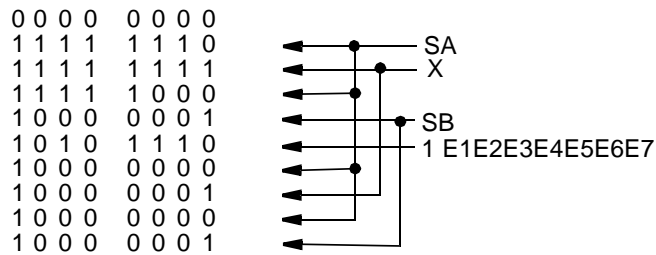


**Figure 67**

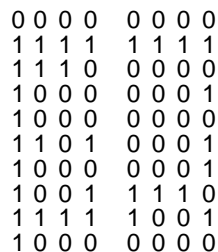
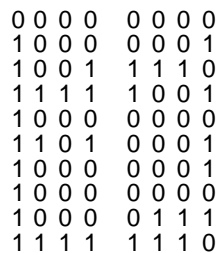
*Note: The first transmit descriptor **must** have the V.110-bit set.*

## Functional Description

TRV = 00



← Change of E-, S-, X-bits



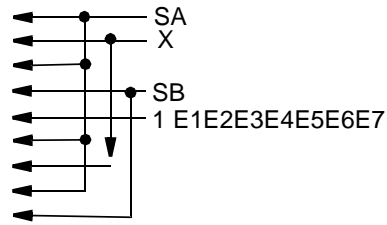
## Functional Description

### TRV = 01

```

0000 0000
1111 1110
1110 0001
1000 0000
1000 0001
1010 1110
1000 0110
1111 1111
1000 0110
1110 0001

```



```

0000 0000
1000 0110
1110 0001
1111 1110
1111 1111
1010 1110
1000 0000
1000 0001
1000 0000
1000 0001

```

FA (last byte of DATA 1) → 1 1 1 1 1 0 1 0  
B6B5B4B3B2B1

C0 (first byte of DATA 2) → 1 1 0 0 0 0 0 0  
B6B5B4B3B2B1

```

0000 0000
1000 0111
1110 0000
1000 0001
1001 1110
1101 0001
1111 1001
1000 0000
1000 0111
1110 0000

```

← Change of E-, S-, X-bits

E2 = 1 1 1 0 0 0 1 0  
B6B5B4B3B2B1

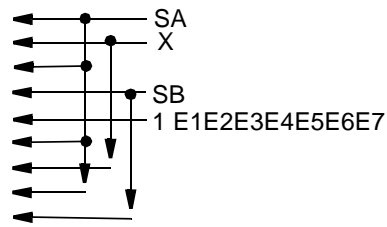
D1 = 1 1 0 1 0 0 0 1  
B6B5B4B3B2B1

### TRV = 10

```

0000 0000
1111 1000
1000 0001
1001 1110
1001 1001
1010 1110
1001 1000
1111 1111
1000 0000
1000 0001

```



← Change of E-, S-, X-bits

```

0000 0000
1001 1001
1000 0110
1110 0001
1001 1000
1101 0001
1
1
1
1

```

E2 = 1 1 1 0 0 0 1 0  
B6B5B4B3B2B1

D1 = 1 1 0 1 0 0 0 1  
B6B5B4B3B2B1

## Functional Description

TRV = 11

```

0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0
1 0 1 1 0 1 0 1
1 0 1 0 1 1 1 0
1 0 0 0 0 0 0 1
1 0 1 0 1 1 1 0
1 0 1 0 0 0 1 0
1 1 0 0 0 1 0 1
1
1

```

← Change of E-, S-, X-bits

For INV = 1 (channel inversion) all bits are inverted. For Motorola mode the data sections would have to have the form to yield the same output data.

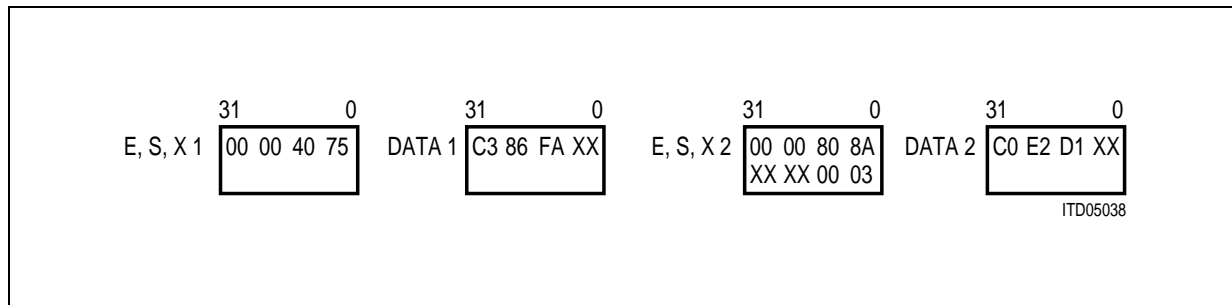


Figure 68

## Functional Description

### V.110/X.30

#### Receive Direction

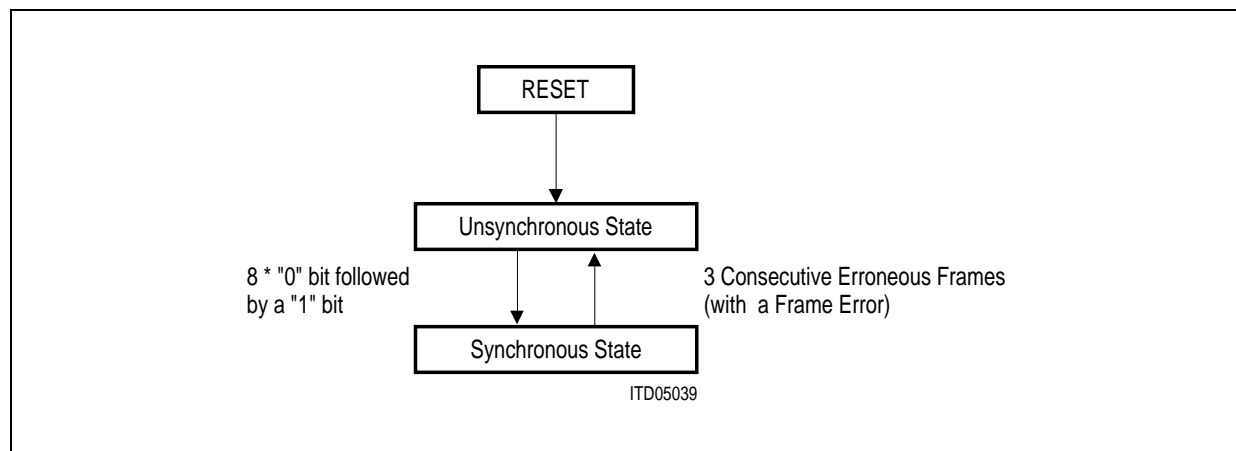
##### General Features

In receive direction

- the starting sequence (00<sub>H</sub> followed by a '1'-bit) after initialization of loss of synchronism is detected.
- the synchronization pattern is monitored, after 3 consecutive erroneous frames a loss of synchronism is detected.
- a change of E-, S-, X-bits is monitored and reported by an interrupt.
- the data bits are extracted and written into the data section.

More detailed description of the individual features:

1. and 2. the receiver can be in one of 2 states:



**Figure 69**

Data extraction and monitoring of a change of E-, S-, X-bits and synchronization pattern is only performed in synchronized state.

In the asynchronous state the receiver waits for the synchronization pattern. The '1'-bit is then interpreted as bit 1 of octet 2.

3. During the synchronized state a change of E, S, X-bits from one frame to the next and even within a frame (for SA, SB bits) is monitored. Only one interrupt per frame is reported even if SA e.g. changes 3 times within the frame. The E-, S-, X-bits reported in the interrupt are S9 for SB and S8 for SA and the second occurrence of X for X.
4. The bits written into the data section are marked by **○** in **Table 2** to **Table 4**. As shown, bits repeated in the serial data are only strobed than at their last instance.

## Functional Description

**Table 2**  
**Framing for Networks with 600-bit/s Data Rate**  
**Intermediate Rate = 8 Kbit/s, i.e. Subchannelling with Only 1 Fill/Mask Bit Set**

Octet No.	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	1	B1	(B1)	B1	B1	B1	B1	S1
3	1	B1	B1	B2	(B2)	B2	B2	X
4	1	B2	B2	B2	B2	B3	(B3)	S3
5	1	B3	B3	B3	B3	B3	B3	S4
6	1	E1	E2	E3	E4	E5	E6	E7
7	1	B4	(B4)	B4	B4	B4	B4	S6
8	1	B4	B4	B5	(B5)	B5	B5	X
9	1	B5	B5	B5	B5	B6	(B6)	S8
10	1	B6	B6	B6	B6	B6	B6	S9

**Table 3**  
**Framing for Networks with 1200-bit/s Data Rate**  
**Intermediate Rate = 8 Kbit/s, i.e. Subchannelling with Only 1 Fill/Mask Bit Set**

Octet No.	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	1	B1	B1	B1	(B1)	B2	B2	S1
3	1	B2	(B2)	B3	B3	B3	(B3)	X
4	1	B4	B4	B4	(B4)	B5	B5	S3
5	1	B5	(B5)	B6	B6	B6	(B6)	S4
6	1	E1	E2	E3	E4	E5	E6	E7
7	1	B7	B7	B7	(B7)	B8	B8	S6
8	1	B8	(B8)	B9	B9	B9	(B9)	X
9	1	B10	B10	B10	(B10)	B11	B11	S8
10	1	B11	(B11)	B12	B12	B12	(B12)	S9



## Functional Description

**Table 4**  
**Framing for Networks with 2400-bit/s Data Rate**  
**Intermediate Rate = 8 Kbit/s, i.e. Subchannelling with Only 1 Fill/Mask Bit Set**

Octet No.	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	1	B1	(B1)	B2	(B2)	B3	(B3)	S1
3	1	B4	(B4)	B5	(B5)	B6	(B6)	X
4	1	B7	(B7)	B8	(B8)	B9	(B9)	S3
5	1	B10	(B10)	B11	(B11)	B12	(B12)	S4
6	1	E1	E2	E3	E4	E5	E6	E7
7	1	B13	(B13)	B14	(B14)	B15	(B15)	S6
8	1	B16	(B16)	B17	(B17)	B18	(B18)	X
9	1	B19	(B19)	B20	(B20)	B21	(B21)	S8
10	1	B22	(B22)	B23	(B23)	B24	(B24)	S9

**Table 5**  
**Framing for Networks with 4800-, 9600-, 19200-, 38400-bit/s Data Rate**  
**Intermediate Rate = 8, 16, 32, 64 Kbit/s, i.e. Subchannelling with 1, 2, 4, 8 Fill/Mask Bit Set**

Octet No.	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	1	B1	B2	B3	B4	B5	B6	S1
3	1	B7	B8	B9	B10	B11	B12	X
4	1	B13	B14	B15	B16	B17	B18	S3
5	1	B19	B20	B21	B22	B23	B24	S4
6	1	E1	E2	E3	E4	E5	E6	E7
7	1	B25	B25	B27	B29	B29	B30	S6
8	1	B31	B32	B33	B35	B35	B36	X
9	1	B37	B36	B39	B41	B41	B42	S8
10	1	B43	B44	B45	B47	B47	B48	S9

They are grouped together in the form:

31 0  
1 1 B6 B5 B4 B3 B2 B1 1 1 B12 B11 B10 B9 B8 B7 1 1 B18 B17 B16 B15 B14 B13 1 1 B24 B23 B22 B21 B20 B19  
(for Motorola mode)

31 0  
1 1 B24 B23 B22 B21 B20 B19 1 1 B18 B17 B16 B15 B14 B13 1 1 B12 B11 B10 B9 B8 B7 1 1 B6 B5 B4 B3 B2 B1  
(for Intel mode)

where for the 600 bit/s e.g. B1 to B6 belong to the first 10-octet frame, B7 to B12 belong to the second 10-octet frame etc.

---

## Functional Description

### Options

The different options for this mode are the framing pattern as shown in **Table 2** to **Table 5** is programmed by the bits TRV.

### Interrupts

The possible interrupts for this mode are

**FRC:** issued if the receiver has detected a change of S-, X-, E-bits; the value of the bits E7, ..., E1, S8 for SA and S9 for SB and the second occurrence of X within the 10-octet frame is reported within the same interrupt.  
(maskable by CH in the channel specification)

**HI:** issued if the HI bit is detected in the transmit descriptor (not maskable).

**ERR:** issued if one of the following receive errors has occurred:

- a fast receive abort channel command was issued (this leads to a setting of the RA bit in the status byte)
- data could only partly be stored due to internal buffer overflow of RB
- 3 consecutive frames had an error in the synchronization pattern (loss of synchronism)
- the HOLD bit in the receive descriptor was detected (this leads to a setting of the RA bit in status in the receive descriptor).  
(maskable by RE in the channel specification)

**FO:** issued if due to inaccessibility of the internal buffer (RB) one or more changes of E-, S-, X-bits and/or loss of synchronism information have been lost.  
(maskable by RE in the channel specification)

### Example

V.110/X.30 channel with

CS = 0           (required)  
INV = 0  
CRC = 0  
TRV = 00       (600 bit/s)  
FA = 0  
MODE = 10   (V.110/X.30)  
Motorola interface  
Channel No. D

## Functional Description

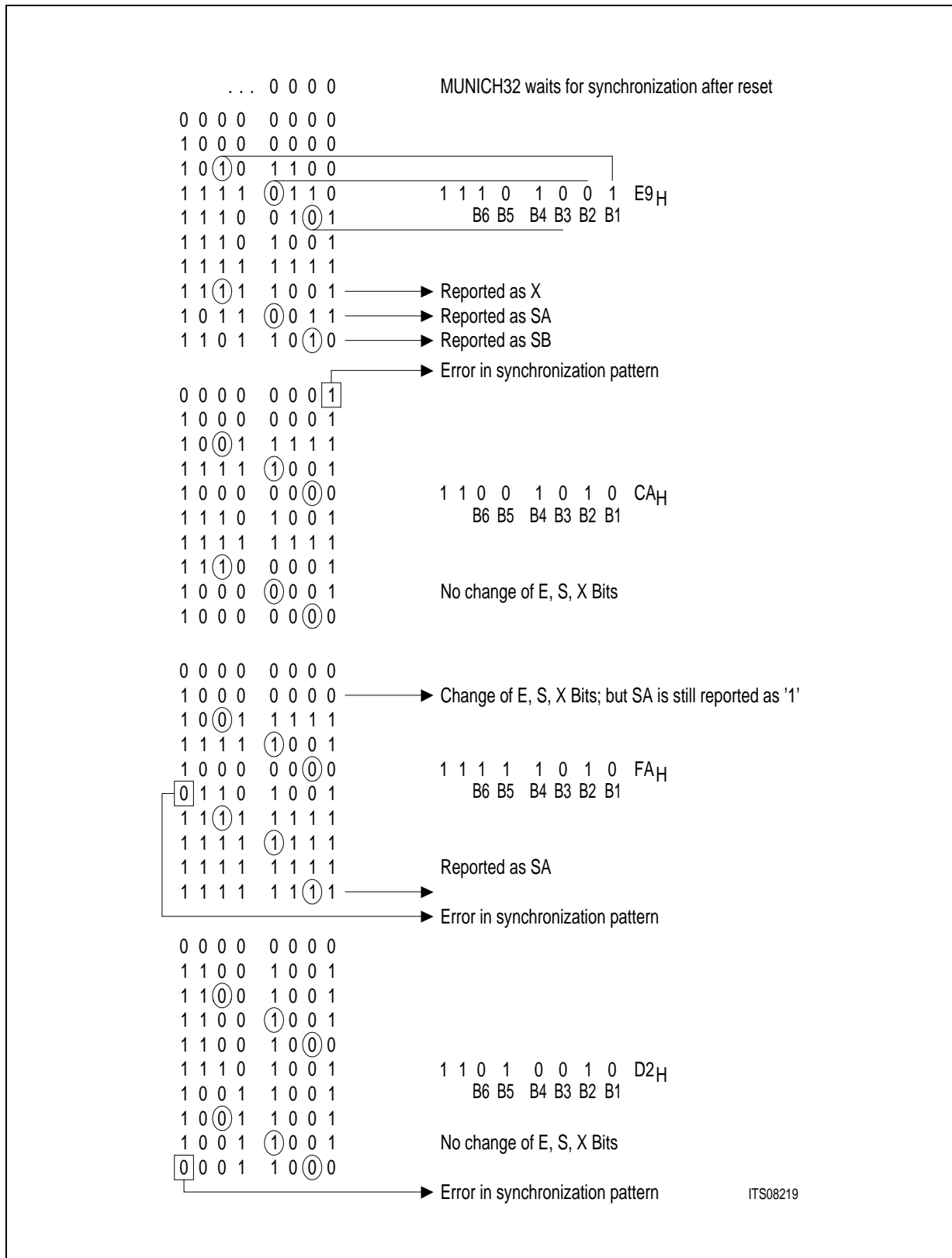
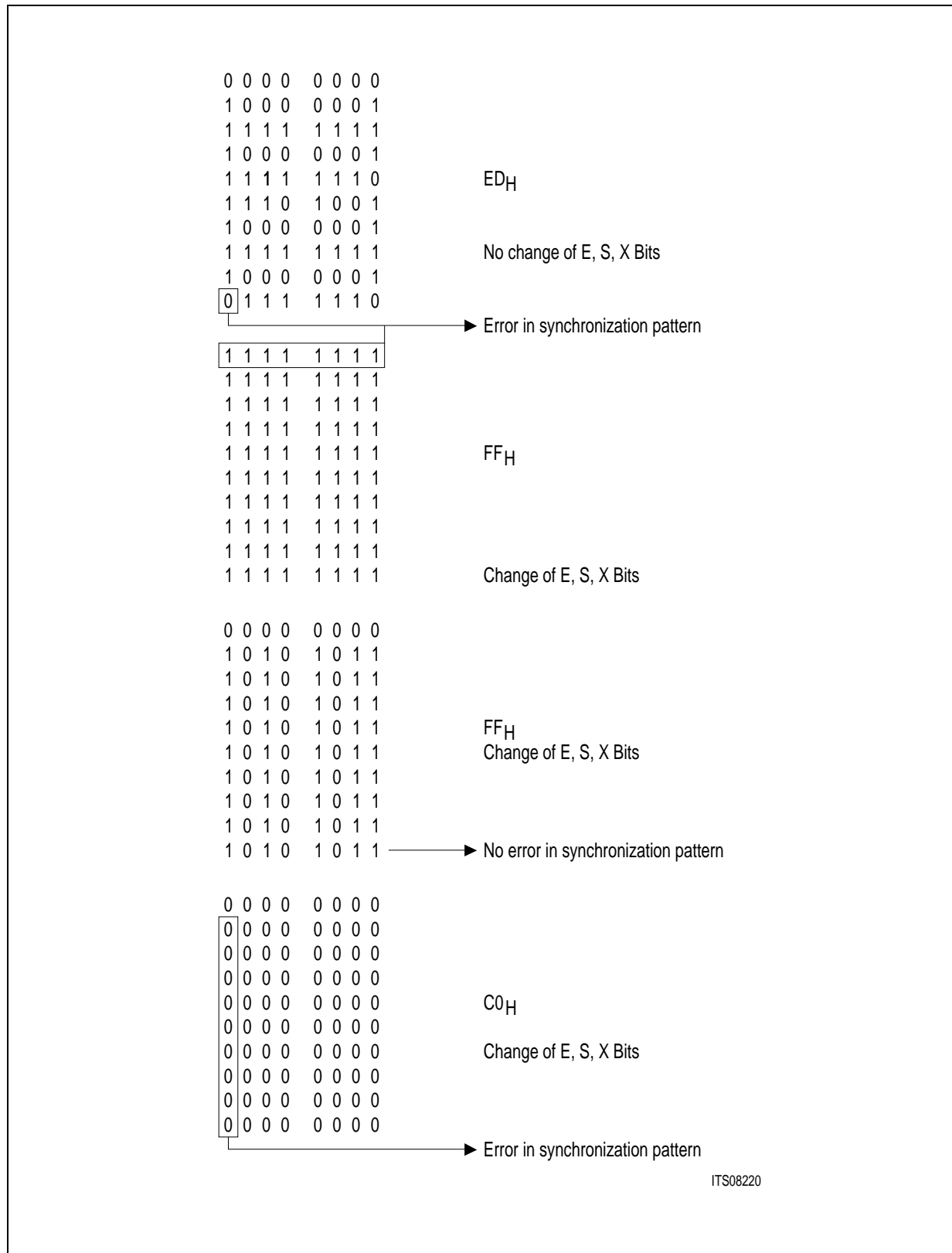


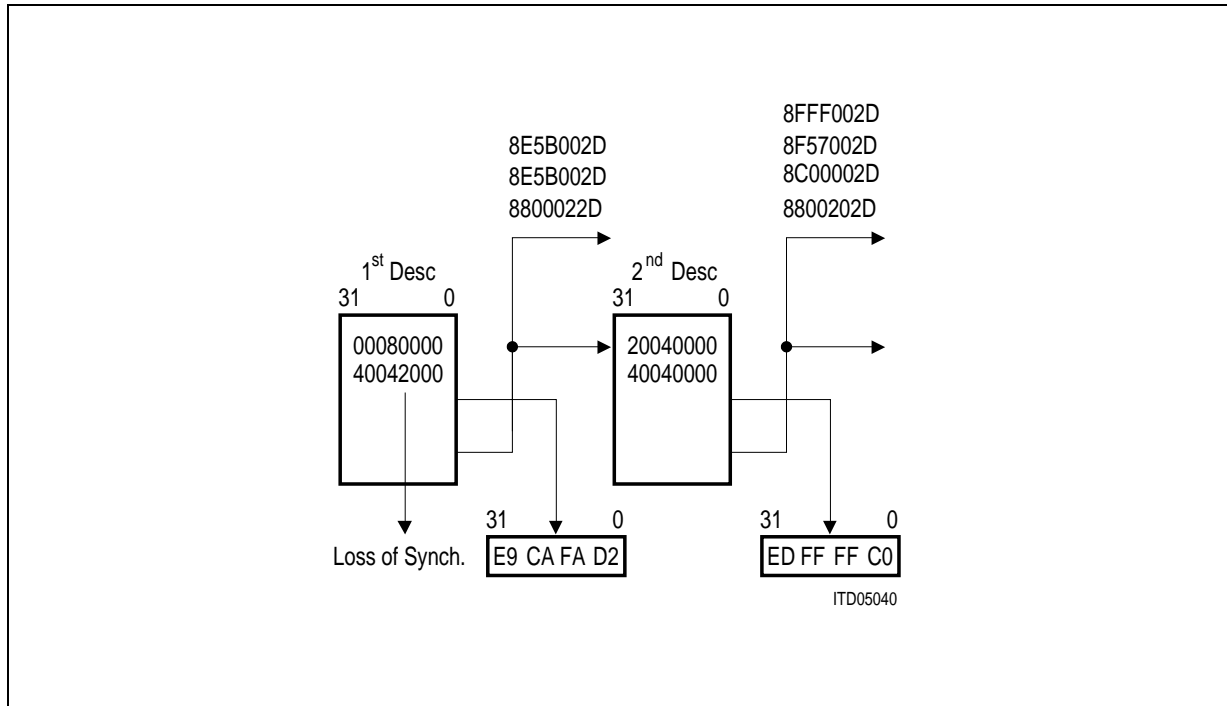
Figure 70a

## Functional Description



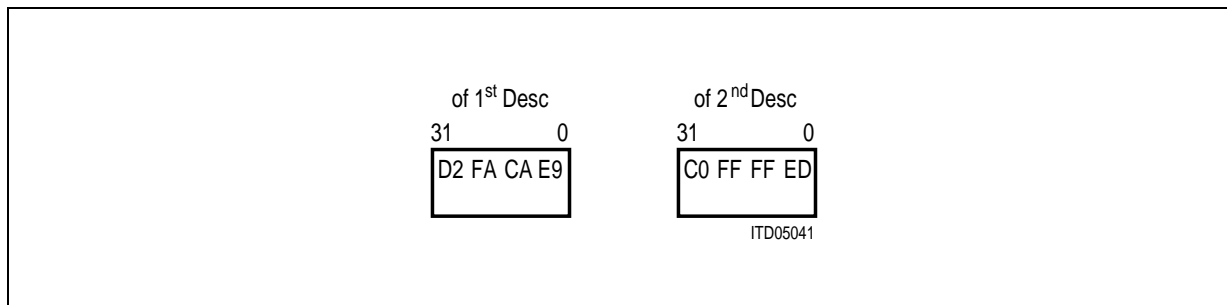
**Figure 70b**

## Functional Description



**Figure 71**

For Intel mode the data sections have the form:

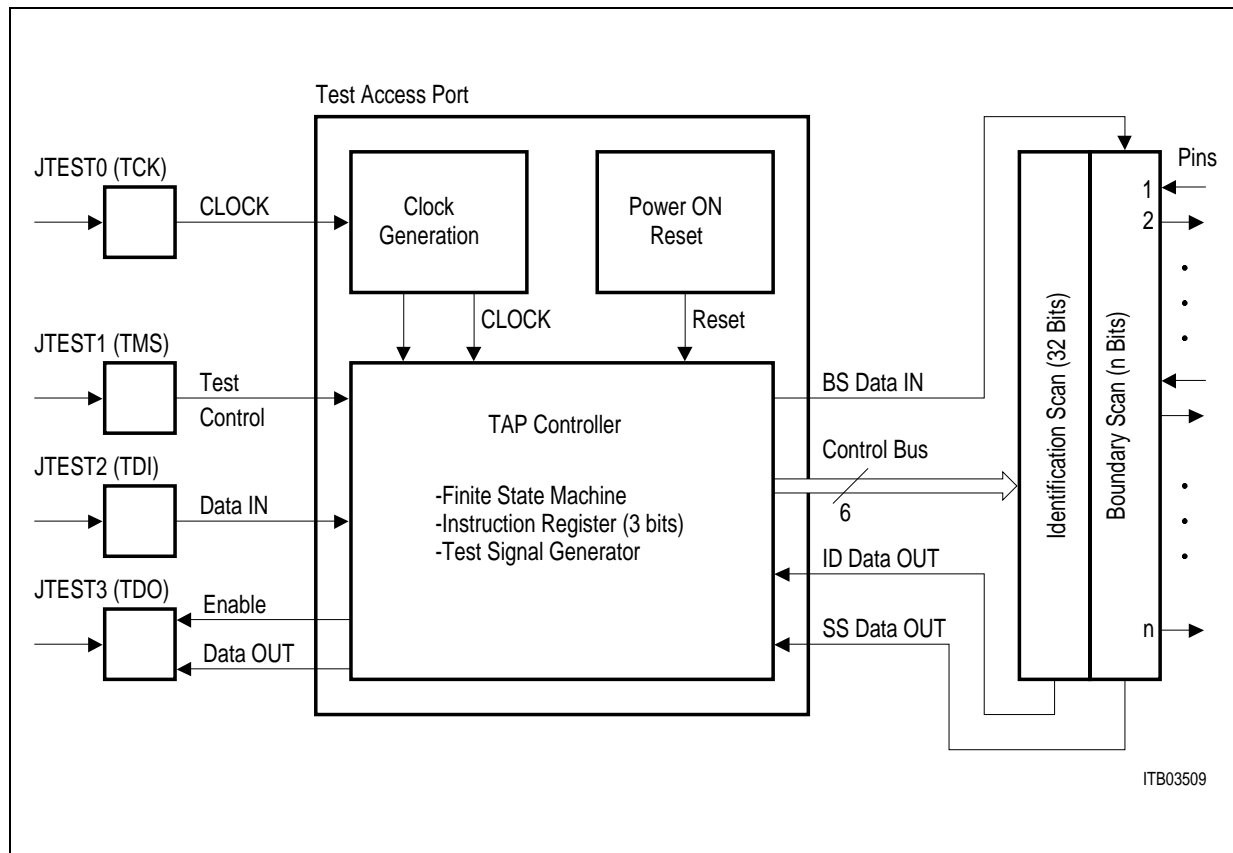


**Figure 72**

## Functional Description

### 2.5 Boundary Scan Unit

In MUNICH32 a Test Access Port (TAP) controller is implemented. The essential part of the TAP is a finite state machine (16 states) controlling the different operational modes of the boundary scan. Both, TAP controller and boundary scan, meet the requirements given by the JTAG standard: IEEE Std. 1149.1. **Figure 73** gives an overview.



**Figure 73**  
**Block Diagram of Test Access Port and Boundary Scan**

Test handling is performed via the pins JTEST0 (TCK), JTEST1 (TMS), JTEST2 (TDI) and JTEST3 (TDO). Test data at JTEST2 (TDI) are loaded with a 4-MHz clock signal connected to JTEST0 (TCK). '1' or '0' on JTEST1 (TMS) causes a transition from one controller state to an other; constant '1' on JTEST1 (TMS) leads to normal operation of the chip.

If no boundary scan testing is planned JTEST1 (TMS) and JTEST2 (TDI) do not need to be connected since pull-up transistors ensure high input levels in this case. Nevertheless it would be a good practice to put the unused inputs to defined levels. In this case, if the JTAG is not used:

JTEST1 = JTEST0 = '1'.

After switching on the device ( $V_{DD} = 0$  to 5 V) a power-on reset is generated which forces the TAP controller into test logic reset state.

## Functional Description

**Table 6**  
**Boundary Scan Sequence in PEB 20320**

JTEST2 (TDI) →

Pin No.	Pin	I/O	Number of Boundary Scan Cells	Constant Value
1	Reset	I	1	0
2	SCLK	I	1	1
3	TEST	I	1	1
4	AR	I	1	0
5	TDATA	O	2	00
6	TSP	I	1	0
7	TCLK	I	1	0
8	I/M	I	1	0
9	B16	I	1	0
10	Ready/DSACK	I	1	0
11	BERR	I	1	0
12	HLDA/BG	I	1	0
13	HLDAO/BGO	O	2	00
14	BGACK	I/O	3	001
15	HOLD/BR	I/O	3	010
16	ADS/AS	O	2	00
17	DS	O	2	01
18	WR/RW	O	2	00
19	BE3	O	2	00
20	BE2	O	2	01
21	D0	I/O	3	100
22	BE1	O	2	00
23	D1	I/O	3	000
24	BE0	O	2	00
25	D2	I/O	3	000
26	A2	O	2	00
27	D3	I/O	3	000
28	A3	O	2	00
29	D4	I/O	3	000
30	A4	O	2	00
31	D5	I/O	3	000
32	A5	O	2	00
33	D6	I/O	3	000
34	A6	O	2	00
35	D7	I/O	3	000

**Functional Description**

**Table 6**  
**Boundary Scan Sequence in PEB 20320 (cont'd)**

JTEST2 (TDI) →

	Pin	I/O	Number of Boundary Scan Cells	Constant Value
36	A7	O	2	00
37	D8	I/O	3	000
38	A8	O	2	00
39	D9	I/O	3	000
40	A9	O	2	00
41	D10	I/O	3	000
42	A10	O	2	00
43	D11	I/O	3	000
44	A11	O	2	00
45	D12	I/O	3	000
46	A12	O	2	00
47	D13	I/O	3	000
48	A13	O	2	00
49	D14	I/O	3	000
50	A14	O	2	00
51	D15	I/O	3	000
52	A15	O	2	00
53	D16	I/O	3	000
54	A16	O	2	00
55	D17	I/O	3	000
56	A17	O	2	00
57	D18	I/O	3	000
58	A18	O	2	00
59	D19	I/O	3	000
60	A19	O	2	00
61	D20	I/O	3	000
62	A20	O	2	00
63	D21	I/O	3	000
64	A21	O	2	00
65	D22	I/O	3	000
66	A22	O	2	00
67	D23	I/O	3	000
68	A23	O	2	00
69	D24	I/O	3	000
70	A24	O	2	00



## Functional Description

**Table 6**  
**Boundary Scan Sequence in PEB 20320 (cont'd)**

JTEST2 (TDI) →

	Pin	I/O	Number of Boundary Scan Cells	Constant Value
71	D25	I/O	3	000
72	A25	O	2	00
73	D26	I/O	3	000
74	A26	O	2	00
75	D27	I/O	3	000
76	A27	O	2	00
77	D28	I/O	3	000
78	A28/DP0	I/O	3	000
79	D29	I/O	3	000
80	A29/DP1	I/O	3	000
81	D30	I/O	3	000
82	A30/DP2	I/O	3	000
83	D31	I/O	3	000
84	A31/DP3	I/O	3	000
85	INT/INT	O	2	00
86	RCLK	I	1	0
87	RSP	I	1	0
88	RDATA	I	1	0
89	CI0	I	1	0
90	CI1	I	1	0
91	CI2	I	1	0
92	CI3	I	1	0
93	CI4	I	1	0

→ JTEST3 (TDO)

An input pin (I) uses one boundary scan cell (data in), an output pin (O) uses two cells (data out, enable) and an I/O-pin (IO) uses three cells (data in, data out, enable). Therefore the boundary scan of the MUNICH32 contains a total of  $n = 205$  scan cells.

The right column of **Table 6** gives the initialization values of the cells.

The desired test mode is selected by serially loading a 3-bit instruction code into the instruction register via JTEST2 (TDI) (LSB first); see **Table 3**.

## Functional Description

**Table 7**  
**Boundary Scan Test Modes**

Instruction (Bit 2 ... 0)	Test Mode
000	EXTEST (external testing)
001	INTEST (internal testing)
010	SAMPLE/PRELOAD (snap-shot testing)
011	IDCODE (reading ID code)
111	BYPASS (bypass operation)
others	handled like BYPASS

**EXTEST** is used to examine the interconnection of the devices on the board. In this test mode at first all input pins **capture** the current level on the corresponding external interconnection line, whereas all output pins are held at constant values ('0' or '1', according to **Table 6**). Then the content of the boundary scan is **shifted** to JTEST3 (TDO). At the same time the next scan vector is loaded from JTEST2 (TDI). Subsequently all output pins are **updated** according to the new boundary scan contents and all input pins again capture the current external level afterwards, and so on.

**INTEST** supports internal testing of the chip, i.e. the output pins **capture** the current level on the corresponding internal line whereas all input pins are held on constant values ('0' or '1', according to **Table 6**). The resulting boundary scan vector is **shifted** to JTEST3 (TDO). The next test vector is serially loaded via JTEST2 (TDI). Then all input pins are **updated** for the following test cycle.

*Note: In capture IR-state the code '001' is automatically loaded into the instruction register, i.e. if INTEST is wanted the shift IR-state does not need to be passed.*

**SAMPLE/PRELOAD** is a test mode which provides a snap-shot of pin levels during normal operation.

**IDCODE**: A 32-bit identification register is serially read out via JTEST3 (TDO). It contains the version number (4 bits), the device code (16 bits) and the manufacturer code (11 bits). The LSB is fixed to '1'.

JTEST2 (TDI) →	0110	0000 0000 0000 0101	0000 1000 001	1	→ JTEST3 (TDO)
----------------	------	---------------------	---------------	---	----------------

IDCODE for old versions:

0001 for version 2.1
0010 for version 2.2
0100 for version 3.2

*Note: As in test logic reset state the code '011' is automatically loaded into the instruction register the ID code can easily be read out in shift DR state which is reached by JTEST1 (TMS) = 0, 1, 0, 0.*

**BYPASS**: A bit entering JTEST2 (TDI) is shifted to JTEST3 (TDO) after one JTEST0 (TCK) clock cycle.

### 3 Operational Description

#### 3.1 Reset State

Upon reset MUNICH32 is set to its initial state. The active high system reset clears the internal logic and causes MUNICH32 to tristate all output lines. Channel processing is deactivated. After reset all buffers are empty and no buffer size of TB is allocated to the channels. The DMA controller state is set to the hold condition for all link lists. The descriptor and data pointers remain at a random value.

The bits RO and TO are set to '1' and RA and TA are set to '0' for all logical channels by reset. All time slots are connected to the logical channel 0 and the following configuration is set:

#### Action Specification

LOC = LOOP = LOOPI = 0

PCM = T1/DS1 × 24-channel 1.536 Mbit/s (000)

MFL = 0

#### Time Slot Assignment

fill/mask = 00<sub>H</sub>, i.e. all bits masked/set to '1'

RTI, TTI = 0

channel number = 00<sub>H</sub>

#### Channel Specification

MODE = 00, i.e. TMA

FA = 0

IFTF = 0

CRC = 0

INV = 0

TRV = 00,

RO = 1
--------

RA = 0

TO = 1
--------

TA = 0

TH = 0

## Transmit Descriptor

FNUM = 00<sub>H</sub>, i.e. shared flags in HDLC, only eight zero bits between sent frames for TMB.

The E-, S-, X-bits are all set to zero internally by the reset. The receiver is set into the ITF/IDLE state for all channels, i.e. it assumes that on the line there are '1's as interframe time-fill for HDLC.

## 3.2 Initialization Procedure

After reset MUNICH32 remains in the initial state until the microprocessor generates an action request. In the action specification the initialization sequence is defined. The sequence can be split up into individual procedures of each channel or in one single procedure to initialize all channels at the same time. For all procedures the time slot assignment and the selected channel specifications are loaded into the CSR-RAM. To prevent malfunction the initialization of the link lists and the allocation of the buffer size to the channels has to be specified before the transmission can be started. The interrupt queue must be established as well. MUNICH32 assumes that time slot 0 starts on the receive and transmit lines. They can be resynchronized by 2 rising edges of TSP and RSP respectively. The first rising edge of TSP/RSP should not take place within the first 1000 SCLK clock cycles after deassertion of the reset pin.

Before this resynchronization the host should neither remove RO = 1, TO = 1 nor set LOOP or LOOPI to '1' for any logical channel. During this time any incoming data is ignored, the transmit data line tristated.

For each action service the device first reads the control start address in the control and configuration section which is located under a fixed address determined by the input signals (CI(4:0)).

The values of CI(4:0) can be changed during operation. The values are used after the next falling edge of  $\overline{AR}$ .

CI4 is the polarity of A31 ... A22

CI3 is the polarity of A21 ... A16

CI2 is the polarity of A15 ... A4

CI1 is the polarity of A3

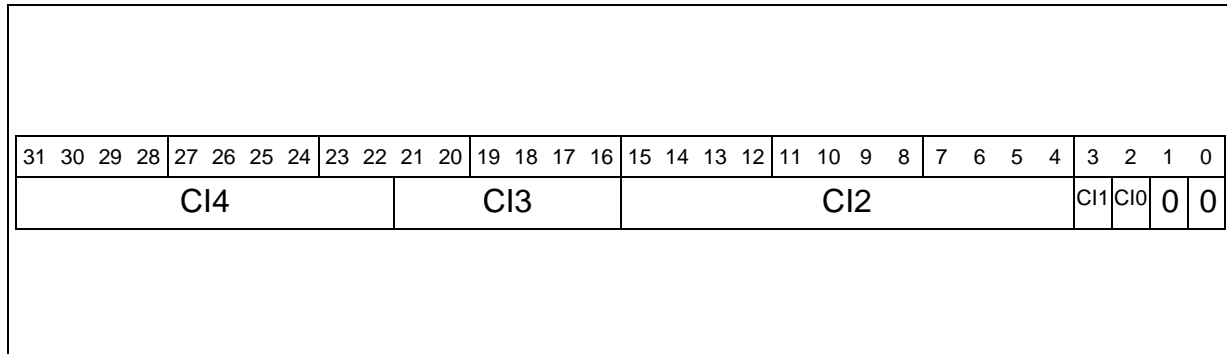
CI0 is the polarity of A2

A0, A1 = 0

for example CI(4:0) = 10101

ADDRESS = 1111.1111.1100.0000.1111.1111.1111.0100

## Operational Description



**Figure 74**

CI4	CI3	CI2	CI1	CI0	Loc. of Ctrl. Start Addr.	CI4	CI3	CI2	CI1	CI0	Loc. of Ctrl. Start Addr.
0	0	0	0	0	0000 0000	1	0	0	0	0	FFC0 0000
0	0	0	0	1	0000 0004	1	0	0	0	1	FFC0 0004
0	0	0	1	0	0000 0008	1	0	0	1	0	FFC0 0008
0	0	0	1	1	0000 000C	1	0	0	1	1	FFC0 000C
0	0	1	0	0	0000 FFF0	1	0	1	0	0	FFC0 FFF0
0	0	1	0	1	0000 FFF4	1	0	1	0	1	FFC0 FFF4
0	0	1	1	0	0000 FFF8	1	0	1	1	0	FFC0 FFF8
0	0	1	1	1	0000 FFFC	1	0	1	1	1	FFC0 FFFC
0	1	0	0	0	003F 0000	1	1	0	0	0	FFFF 0000
0	1	0	0	1	003F 0004	1	1	0	0	1	FFFF 0004
0	1	0	1	0	003F 0008	1	1	0	1	0	FFFF 0008
0	1	0	1	1	003F 000C	1	1	0	1	1	FFFF 000C
0	1	1	0	0	003F FFF0	1	1	1	0	0	FFFF FFF0
0	1	1	0	1	003F FFF4	1	1	1	0	1	FFFF FFF4
0	1	1	1	0	003F FFF8	1	1	1	1	0	FFFF FFF8
0	1	1	1	1	003F FFFC	1	1	1	1	1	FFFF FFFC

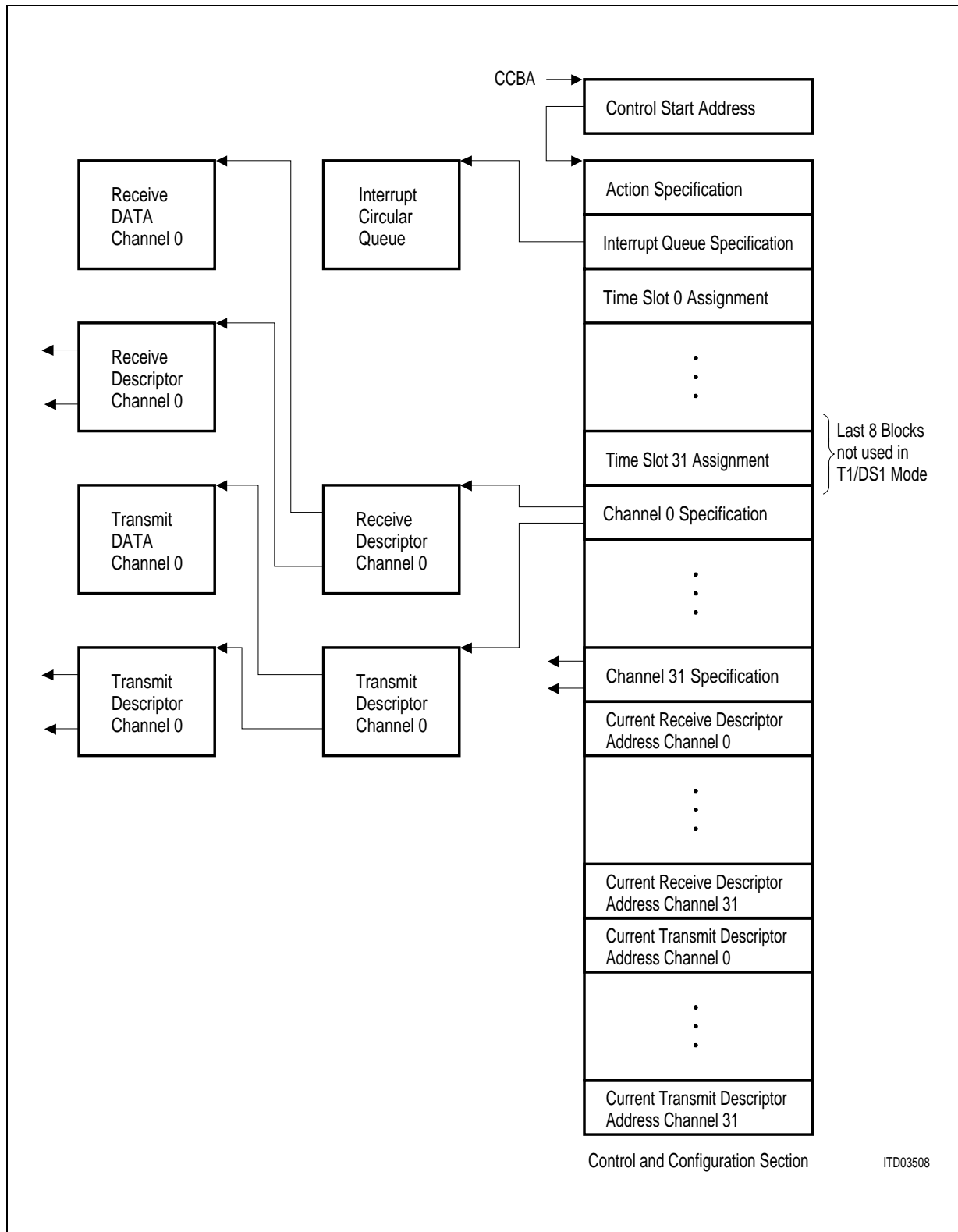
**Figure 75**  
**CI-Pin Decoding**

## 4 Detailed Register Description

### 4.1 Organization of the Shared Memory

Because the MUNICH32 reads only long words, all addresses of the link lists, interrupt queue and the CCS must be a multiple of four; i.e. the two least significant bits of the address must be '00'. **Figure 76** depicts the organization of the shared memory for one MUNICH32.

## Detailed Register Description



**Figure 76**  
**Organization of the Shared Memory**

## Detailed Register Description

### 4.2 Control and Configuration Section

**Table 8**  
**Buffer Size of the Control and Configuration Section**

Control and Configuration Section	Number of Long Words
Action specification	1
Interrupt queue specification	2
Time slot assignment	32
Channel specification	128
Current descriptor address	64

#### 4.2.1 Action Specification (Read Once After Each Action Request Pulse)

All actions are selected by setting the corresponding bits to '1'.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PCM			MFL												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IN	IC0	0	Channel Number					IM	RES	LOC	LOOP	LOOPI	IA	0	0

PCM: These three bits determine the PCM highway format.

000: T1/DS1 24-channel 1.536 Mbit/s

100: T1/DS1 24-channel 1.544 Mbit/s

101: CEPT 32-channel

110: 4.096-Mbit/s PCM format and even numbered time slots

111: 4.096-Mbit/s PCM format and odd numbered time slots

MFL: Maximum Frame Length (up to 8191 bytes); MUNICH32 monitors the frame length of the incoming HDLC, TMB or TMR frames. If the maximum frame length is exceeded an interrupt is generated and the current frame aborted. The length check is active in all modes except transparent mode A and V.110/X.30. Therefore in all other modes one has to write a reasonable value to MFL after reset. MFL is the same for all logical channels.



## Detailed Register Description

- IN:** Initialization procedure; setting this bit to one causes MUNICH32 to fetch all the time slot assignments and the channel specification of the selected channel (channel number). To avoid collision all time slots being reinitialized should be in a deactivated mode, i.e. the receive and transmit channels must be switched off.
- ICO:** Initialize Channel Only; only the channel specification of the selected channel (channel number) is read and reconfigured.
- IM:** Interrupt Mask; MUNICH32 suppresses the interrupt normally generated in order to acknowledge the action request.
- RES:** RESET; a single initialization procedure is performed. The time slot assignment and all channel specifications are written into the CSR. All time slots are reinitialized.

*Note 1: The bits IN, ICO, RES are mutually exclusive within one action specification. They establish different ways of initializing, configuring and reconfiguring the channels and time slots of the MUNICH32.*

For test purposes four different loops can be switched at the serial interface with aid of LOC, LOOP, LOOPI according to the following table

LOC	LOOP	LOOPI	Interpretation
0	0	0	no loop
1	0	0	not allowed
0	0	1	complete internal loop
1	0	1	channelwise internal loop
0	1	0	complete external loop
1	1	0	channelwise external loop
0	1	1	not allowed
1	1	1	not allowed

The loops have the following functions:

- Complete external loop  
The serial data input is physically mirrored back to the serial data output. The time and strobe signals for receive and transmit direction have to be identical.
- Complete internal loop  
The serial data output is physically mirrored back to the serial data input. The data on the external input line are ignored. The logical channels have to be programmed identically. The time and strobe signals for receive and transmit direction have to be identical.

## Detailed Register Description

- Channelwise external loop  
One single logical channel is mirrored logically from serial data input to serial data output. The other channels are not affected by this operation. The data rate for this single logical channel has to be identical for receive and transmit direction.
- Channelwise internal loop  
One single logical channel is mirrored logically from serial data output to serial data input. The other channels are not affected by this operation. The data rate for this single logical channel has to be the same for receive and transmit direction.

See **Chapter 5.1** and **Chapter 5.3.2** for a more detailed discussion of test loops.

All loops of the MUNICH32 V3.2 are under complete software control. Loops can be closed and opened via software.

Handling of the MUNICH32 V3.2 loops:

Switch loops on:

RES = IN = ICO = '0'

LOC, LOOP, LOOPI

PCM, MFL, IM, IA

CHANNEL NUMBER

for selected loop type

don't change the previous values

in case of channelwise loops use the selected channel number

in case of complete loops use channel number of an active channel.

Switch loops off:

RES = IN = ICO = '0'

LOC = '0', LOOP = LOOPI = '1'

PCM, MFL, IM, IA

CHANNEL NUMBER

don't change the previous values

use channel number used with the 'switch loop on'.

IA: Interrupt Attention; a new interrupt queue is defined by the host. MUNICH32 reads the interrupt queue specification and writes the interrupt information into the new interrupt queue.

## Detailed Register Description

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
PCM			MFL													IN	ICO	0	Channel Number			IM	RES	LOC	LOOP	LOOPI	IA	0	0				
			<p><b>Initialization Procedure</b> Read the complete time-slot assignment and the channel spec. of the specified channel (channel number).</p> <hr/> <p><b>Initialize Channel Only</b> Only the channel spec. of the selected channel (channel number) is read and reconfigured.</p> <hr/> <p><b>Maximum Frame Length</b> Maximum size of a received frame in HDLC, TMB and TMR mode (up to 8192 bytes). A received frame is aborted and an interrupt is generated if the size of a received frame exceeds the MFL value. MFL applies to all channels.</p> <hr/> <p><b>PCM Highway Format</b>            0 0 0 T1/DS1 24 Time-Slots, 1.536 Mbit/s            0 0 1 Not Allowed            0 1 0 Not Allowed            0 1 1 Not Allowed            1 0 0 T1/DS1 24 Time-Slots, 1.544 Mbit/s            1 0 1 CEPT 32 Time-Slots, 2.048 Mbit/s            1 1 0 4.096 Mbit/s PCM Format, even Time-Slots            1 1 1 4.096 Mbit/s PCM Format, odd Time-Slots         </p> <hr/>													<p><b>Channel No.</b> Used in conjunction with IN and ICO</p> <hr/>						<p><b>Interrupt Attention</b> A new interrupt queue has been defined. Read the interrupt queue specification.</p> <hr/> <p><b>Loops</b>            0 0 0 No Loop            0 0 1 Complete Internal Loop            0 1 0 Complete Internal Loop            0 1 1 Not Allowed            1 0 0 Not Allowed            1 0 1 Channelwise int. Loop            1 1 0 Channelwise ext. Loop            1 1 1 Not Allowed         </p> <hr/> <p><b>Reset</b>            Read the complete time-slot assignment            Read all channel specifications            Reinitialize all time-slots         </p> <hr/> <p><b>Interrupt Mask</b>            Do not generate the ARACK &amp; ARF interrupt         </p> <hr/>											

ITS08221

ITS08221

**Figure 77**  
**Action Specification**

## Detailed Register Description

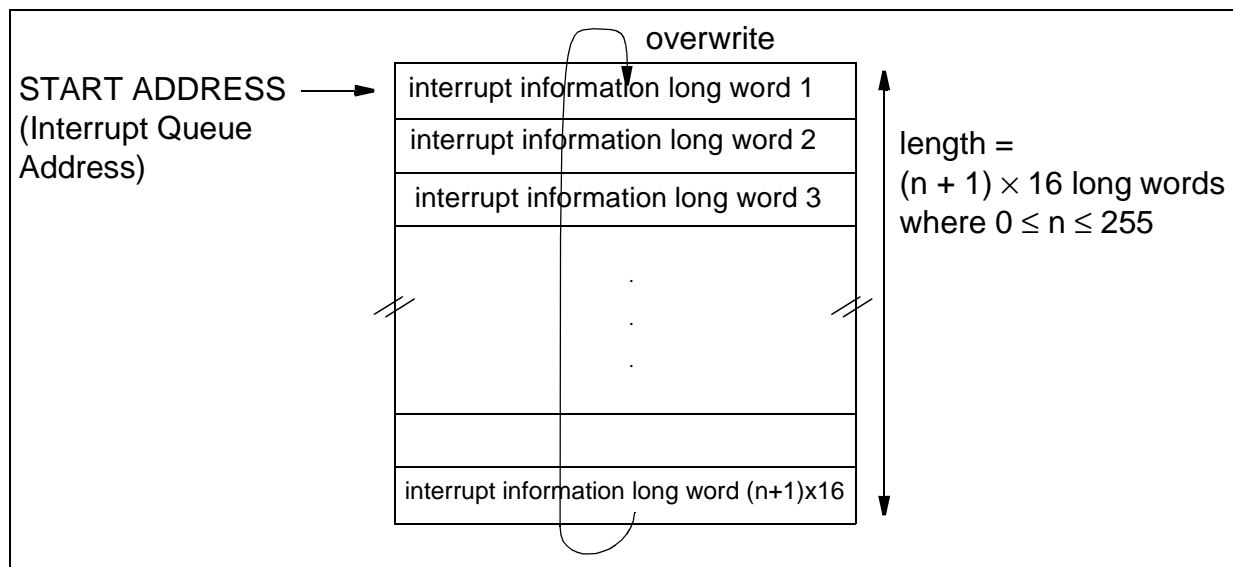
### 4.2.2 Interrupt Queue Specification

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Interrupt Queue Address															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Interrupt Queue Address															
0	0	0	0	0	0	0	0	n							

The interrupt queue is specified as a kind of block (queue), starting on a start address (programmable) with a defined length (programmable). Both, the start address and the queue length are programmable in the Interrupt Queue Specification of the Control and Configuration Section.



**Figure 78**

The minimum queue size is 16 long words; the maximum queue size is 4096 long words. For each interrupt arising, the MUNICH32 writes the interrupt information into the interrupt queue, will increment the pointer to the next address in this block automatically and will generate an interrupt pulse at each interrupt occasion. It is up to the processor to read the interrupt informations out of the interrupt queue. If the MUNICH32 arrives at the end of the interrupt queue, it will jump to the start address of the interrupt block again (cyclic queue) and completely overwrite the previous information.

Therefore the length of the interrupt queue should be calculated so, that the MUNICH32 will not overwrite information which was not yet read by the processor.

## Detailed Register Description

### 4.2.3 Interrupt Information

The next table shows the bit assignments for the interrupt information long word.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INT	0	Interrupt Information													

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Interrupt Information										Channel Number/Direction					

When an interrupt occurs MUNICH32 sets the INT bit and writes the interrupt information and the channel number into the interrupt circular buffer. At the same time it generates an interrupt pulse. The classes of error (for example host initiated interrupt or CRC error) of a channel in one direction are treated independently of each other. If several interrupt events coincide they will be indicated to the host with one shared interrupt.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INT	0	VN3	VN2	VN1	FRC	E7	E6	E5	E4	E3	E2	E1	SB	SA	X

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARACK	ARF	HI	FI	IFC	SF	ERR	FO	0	0	RT	Channel Number				

Bit assignment for interrupt queue

There are 3 classes of bits in the interrupt:

1. Bits present in each interrupt:

INT: this bit is always set to '1'

VN(3:1): these bits are '000' for version 1.1  
'001' for version 2.1  
'010' for version 2.2  
'100' for version 3.2  
'110' for version 3.4

## Detailed Register Description

### 2. Action request interrupts

**ARACK:** Action Request Acknowledge; MUNICH32 sets the ARACK bit to indicate that an action request has been serviced.

**ARF:** Action Request Fail; MUNICH32 aborts an ACTION REQUEST, if the required configuration cannot be performed. An action request fail can occur either when the TB buffer is initialized incorrectly or a bus cycle error ( $\overline{\text{BERR}} = 0$ ) is detected during a configuration access.

If ARACK or ARF is set, all bits except INT and VN(3:1) are set to 0.

*Note: An action request is forbidden during the time a preceding action has not been finished by an ARACK or ARF interrupt or a pulse at the reset pin.*

### 3. Channel specific interrupts

These interrupts indicate specific events in the channel indicated by 'Channel Number' and receive or transmit direction indicated by RT (RT = '1': receive direction; RT = '1': transmit direction).

The interpretation of these interrupts depends on the specification of the channel in which they occur.

The following table shows which interrupts can occur in which mode (unused bits are always 0).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INT	0	VN3	VN2	VN1	FRC	E7	E6	E5	E4	E3	E2	E1	SB	SA	X

#### HDLC

1 0 F F F 0 0 0 0 0 0 0 0 0 0 0

#### V.110/X.30

1 0 F F F R R R R R R R R R R R

#### TMA

1 0 F F F 0 0 0 0 0 0 0 0 0 0 0

#### TMB/TMR

1 0 F F F 0 0 0 0 0 0 0 0 0 0 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARACK	ARF	HI	FI	IFC	SF	ERR	FO	0	0	RT	Channel Number				

#### HDLC

G G TR TR R R TR TR 0 0 X X X X X X

#### V.110/X.30

G G TR 0 0 0 TR TR 0 0 X X X X X X

#### TMA

G G TR T 0 0 TR TR 0 0 X X X X X X

#### TMB/TMR

G G TR TR 0 0 TR TR 0 0 X X X X X X

## Detailed Register Description

Where '1' means that the bit is always '1' for this mode  
 '0' means that the bit is always '0' for this mode  
 'F' means the bit is fixed by the version number  
 'R' means a bit that can only be set in the receive direction, i.e. may only be '1' if RT is '1'  
 'T' means a bit that can only occur in transmit direction, i.e. may only be '1' if RT is '0'  
 'TR' means a bit that can occur in receive or transmit direction  
 'G' means a bit of an activation request interrupt which cannot be 'G' in a channel specific interrupt  
 'X' means a bit fixed by the channel and direction (receive, transmit) of the event it belongs to.

The meaning of the interrupt bits depend on the mode. We therefore will discuss them bit for bit and indicate the different meanings in the different modes.

**FRC:** (V.110/X.30 mode, receive direction only)  
 Change of the framing (E, S, X) bits of the V.110/X.30 frame detected. This interrupt is generated whenever a change in the E-, S-, X-bits is detected, but at most one time within one frame of 10 octets, even if there is more than one change within the frame. After detecting a receive abort channel command for one 10-octet frame FRC is also issued.

**Ex, Sx, X:** (V.110/X.30 mode, receive direction only, only in conjunction with FRC)  
 The value of the bits Ex, Sx, X in the received V.110/X.30 frame. If a value changes, e.g. 2 times within the same frame only the final change is reported.  
 If the change was caused by a receive abort channel command all bits are 0.

**HI:** (all modes, all direction)  
 Host initiated Interrupt; this bit is set when the MUNICH32 detects the HI bit in the receive or transmit descriptor and branches to the next descriptor, or starts polling the hold bit if set.

**FI:** 1.1 HDLC, TMB, TMR Receive Direction:  
 FI = 1 indicates, that a frame has been received completely or was stopped by a receive abort channel command or fast receive abort or a HOLD in a receive descriptor. It is set when the MUNICH32 branches from the last descriptor belonging to the frame to the first descriptor of a new frame. It is also set when the descriptor in which the frame finished contained a hold bit, the interrupt is then issued when the MUNICH32 starts polling the hold bit.

1.2 HDLC, TMB, TMR, TMA Transmit Direction:  
 issued if the FE bit is detected in the transmit descriptor. It is set when the MUNICH32 branches to the next transmit descriptor, belonging to a

## Detailed Register Description

new frame, or when it starts polling the hold bit if set in conjunction with the FE bit; ERR and FI are set if a transmit descriptor contains a HOLD bit no FE bit

**IFC:** (HDLC mode, receive direction only)  
Idle/Flag Change; an interrupt is generated in HDLC if the device changes the interframe time-fill (ITF) state. After reset the device is in the ITF idle state. It changes to the ITF flag state if it receives 2 consecutive flags with or without shared zeroes. It changes back to the ITF idle state upon reception of 15 contiguous '1'-bits or when a receive abort channel command is active during 15 received bits.

**SF:** (HDLC mode, receive direction only, always in conjunction with FI)  
Short frame detected  
A frame with  $\leq 16$  bits between start flag and end flag or end abort flag for CRC16  
 $\leq 32$  bits between start flag and end flag or end abort flag for CRC32  
has been detected. The sequences  $7E\ 7F_H$  and  $7E\ FE_H$  and  $7E\ FF_H$  are also short frames.  
SF is always in conjunction with ERR except for the frames  
 $7E00\ 007E_H$  for CRC16  
 $7E00\ 0000\ 007E_H$  for CRC32

**ERR:** always in conjunction with FI = 1

1.1 HDLC mode                      Receive Direction  
One of the following receive errors occurred

- FCS of the frame was incorrect
- the bit length of the frame was not divisible by 8
- the byte length exceeded MFL
- the frame was stopped by  $7F_H$
- the frame could only be partly stored due to internal buffer overflow of RB
- the frame was ended by a receive abort channel command
- the frame could not be transferred to the shared memory completely because of a hold bit set in a receive descriptor not providing enough bytes for the frame.
- the frame was aborted by a fast receive abort channel command

A more detailed error analysis can be done by the status information in the receive descriptor.

1.2 HDLC mode                      Transmit Direction  
one of the following transmit errors occurred:

- the last descriptor had HOLD = 1 and FE = 0
- the last descriptor had NO = 0 and FE = 0



## Detailed Register Description

### 2.1 V.110/X.30 mode Receive Direction

one of the following receive errors occurred:

- data could only partly stored due to internal buffer overflow of RB
- 3 consecutive frames had an error in the synchronization pattern (loss of synchronism)
- a fast receive abort channel command was issued
- the data could not be transferred to the shared memory completely because of a hold bit set in a receive descriptor not providing enough bytes for the data
- a receive abort channel command was active for at least 3 consecutive frames

A more detailed error analysis can be done by the status information in the receive descriptor.

### 2.2 V.110/X.30 mode Transmit Direction

one of the following transmit errors occurred

- the last descriptor had a HOLD = 1 or FE = 1
- the last descriptor had FE = 0 and NO = 0

### 3.1 TMA mode Receive Direction

one of the following errors occurred

- the data could not be transferred to the shared memory completely because of a hold bit set in a receive descriptor not providing enough bytes for the data
- a fast receive abort channel command was issued

### 3.2 TMA mode Transmit Direction

see **Chapter 1.2**

### 4.1 TMB/TMR mode Receive Direction

always in conjunction with FI = 1

one of the following receive errors occurred

- the bit length of the frame was not divisible by 8
- the frame could only be partly stored due to internal buffer overflow of RB
- the frame could not be transferred to the shared memory completely because of a hold bit set in a receive descriptor not providing enough bytes for the frame
- the frame was aborted by a fast receive abort channel command

A more detailed error analysis can be done by the status information in the receive descriptor.

### 4.2 TMB/TMR mode Transmit Direction

see **1.2**

### FO: 1.1 HDLC, TMB, TMR Receive Direction

The MUNICH32 has discarded one or more whole frames or short

## Detailed Register Description

frames or change of interframe time-fill informations due to inaccessibility of the internal buffer RB.

### 1.2 HDLC, TMB, TMR Transmit Direction

The MUNICH32 is unable to access the shared memory in time or has detected a bus cycle error ( $\overline{\text{BERR}} = 0$ ) during a read access on the transmit data section. The current erroneous frame is aborted with a '0' and 14 '1' for HDLC, with 00 for TMB and 0000 for TMR; afterwards interframe time fill is sent until the MUNICH32 can access again the shared memory. The MUNICH32 will read the transmit data from the location which should be accessed before the Tx-FO or  $\overline{\text{BERR}}$  happened and transmit the rest of the erroneous frame.

### 2.1 V.110/X.30 Receive Direction

The MUNICH32 has discarded a loss of synchronism information or a change of a E-, S-, X-bits information due to inaccessibility of the internal buffer RB.

### 2.2 V.110/X.30 Transmit Direction

The MUNICH32 is unable to access the shared memory in time or has detected a bus cycle error ( $\overline{\text{BERR}} = 0$ ) during a read access on the transmit data section. It generates 3 10-octet frames with framing errors and restarts with the next error-free transmit data.

### 3.1 TMA Receive Direction

The MUNICH32 has discarded data due to inaccessibility of the internal buffer RB.

### 3.2 see Chapter 1.2

The following table shows which interrupt bits are masked by which bits in the channel specification.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INT	Ø	VN3	VN2	VN1	FRC	E7	E6	E5	E4	E3	E2	E1	SB	SA	X

Receive

CH

Transmit

–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARACK	ARF	HI	FI	IFC	SF	ERR	FO	0	0	RT	Channel Number				

Receive

FIR IFC SFE RE RE

Transmit

FIT – – TE TE

## Detailed Register Description

### General

IM IM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INT	0	VN(3...1)			FRC	E7	E6	E5	E4	E3	E2	E1	SB	SA	X	ARACK	ARF	HI	FI	IFC	SF	ERR	F0	0	0	RT	Channel Number				
					<b>Framing Bits Changed</b> V.110/X30 mode received E, S, X Bits changed											<b>Channel Number</b> Identifies the channel where the interrupt occurred.															
					<b>Action Request Acknowledge</b> Action request has been completed successfully.											<b>Direction</b> 0 Transmit Interrupt 1 Receive Interrupt															
					<b>Action Request Failed</b> Action request could not be completed successfully.											<b>Overflow/Underflow</b> Internal buffer not available															
<b>Silicon Version Number</b> 0 0 0 V1.1 0 0 1 V2.1 0 1 0 V2.2 1 0 0 V3.2																<b>Protocol Error</b> e.g. CRC error, frame aborted, loss of sync. MFL exceeded Internal buffer overflow/underflow															
					<b>Host Initiated Interrupt</b> HI Bit in the Rcv./Xmt. descriptor was set											<b>Short Frame</b> HDLC mode, in conjunction with FI (empty HDLC frame or incorrect HDLC frame, nothing stored in memory)															
					<b>Frame Indication</b> End of receive or transmit frame indication											<b>Interframe Timefill Change</b> HDLC receiver detected change in ITF state															
<b>Valid Interrupt Entry</b> MUNICH32 sets this Bit with every entry to the interrupt circular queue Software should clear this Bit after reading																															

ITS08222

ITS08222

**Figure 79**  
**Interrupt Information**

## Detailed Register Description

### 4.2.4 Time Slot Assignment

(Read only once after each action request pulse with an action specification with set IN or RES bit)

The time slot assignment provides the cross reference between the 32 (24) time slots of the PCM highway and the data channels (up to a maximum number of 32). The data channels can be composed of different receive and transmit time slots, which have individual bit rates. With the concept of subchanneling, MUNICH32 can realize flexible transmission from 8 kbit/s up to 2.048 Mbit/s per channel.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
0	0	TTI	Transmit Channel Number					Transmit Fill Mask								time slot 0
0	0	TTI	Transmit Channel Number					Transmit Fill Mask								time slot 1
0	0	TTI	Transmit Channel Number					Transmit Fill Mask								time slot 31

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	RTI	Receive Channel Number					Receive Fill Mask								time slot 0
0	0	RTI	Receive Channel Number					Receive Fill Mask								time slot 1
0	0	RTI	Receive Channel Number					Receive Fill Mask								time slot 31

**Fill/Mask Code:** For bit rate adaption the fill/mask code determines the number of bits and the position of these bits within the time slot. For all modes except TMA the bits selected by Fill/Mask = 1 in the slots of a channel are concatenated, those with Fill/Mask = 0 are ignored/tristated in receive/transmit direction. For TMA the bits with Fill/Mask = 0 are received as '1'-bits, in transmit direction these bits are overwritten with 'Z' (see **Chapter 2.4** for more details).

## Detailed Register Description

**Channel Number:** The channel number identifies the data channel. Its transmission mode is described in the respective channel specification.

**TTI:** Transmit Time slot Inhibit; setting this bit to '1' causes MUNICH32 to tristate the transmit time slot. The data is not destroyed but sent in the next not tristated time slot allocated to this channel.

**RTI:** Receive time Slot Inhibit; setting this bit to '1' causes MUNICH32 to ignore the received data in the time slot. The channel is not processed in this time slot.

### 4.2.5 Channel Specification

(Read only once after each activation request pulse with an action specification with set IN, RES or ICO bit; RES: the channel specifications of all channels; IN, ICO: the channel specification of the channel indicated in the action specification)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Interrupt Mask								NITBS	RI	TI	TO	TA	TH	RO	RA
FRDA															
FTDA															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TFLAG						TFLAG /NSF	TFLAG /CS	INV	CR C	TRV		FA	Mode		IFT F
FRDA															
FTDA															
0	0	0	0	0	0	0	0	0	0	ITBS					

**Interrupt Mask:**

7	6	5	4	3	2	1	0
–	SFE	IFC	CH	TE	RE	FIR	FIT

These bits mask the bits in the interrupt information long word according to the table at the end of **Chapter 4.2.3** (interrupt information).

## Detailed Register Description

If an event leads to an interrupt with several bits set (e.g. FI and ERR) masking only a proper subset of them (e.g. ERR) will lead to an interrupt with the nonmasked bits set (e.g. FI). If all bits of an event are masked, the interrupt is suppressed. The interrupt mask is therefore bit specific and not event specific.

**NITBS:** New ITBS value; if this bit is set the individual transmit buffer size ITBS is valid and a new buffer field of TB is assigned to the channel. In this process first the occupied buffer locations of the channel are released and then according to ITBS a new buffer area is allocated. If there is not enough buffer size in TB (occupied by other channels) the process will be aborted and an action request failure interrupt is generated. After aborting no buffer size is allocated to the channel. For preventing action request failure enough buffer locations must be available. This can be done by reducing the buffer size of the other channels. To avoid transmission errors all channels to be newly configured must be deactivated before processing.

*Note: ITBS has to be set to '0' if NITBS = '0'.*

*NITBS should be set to '0' in conjunction with a transmit abort channel command.*

The bits RI, TI, TO, TA, TH, RO, RA are the so called channel command bits. They allow the channel to be initialized, aborted or reconfigured at the serial side as well as at the  $\mu$ P side.

These bits can be decomposed in 3 independent command groups:

	RI, RO, RA	form the receive command group
	TO, TI, TA	the first transmit command group
and	TH	is the second transmit command group.

**We will discuss these bits according to the groups.**

### 1. Receive command group (6 commands)

#### – receive clear

RI = 0, RO = 0, RA = 0 (clears a previous receive abort or receive off condition, affects only the serial interface)

The effect of this command depends on the previous history of the channel

- if the channel was never initialized by a receive initialization command it has no effect
- if it was initialized previously it clears a receive off or receive abort condition set by a previous channel command
- if no receive off or receive abort condition is set it has no effect.

#### – fast receive abort

RI = 0, RO = 0, RA = 1 (clears a previous receive abort or receive off condition, affects only the DMA interface)

This abort is performed in the DMA controller and does not interfere with the reception on the serial interface and the transfer of the data into the receive buffer. If this abort is detected the current receive descriptor is suspended with an abort status (RA bit set

## Detailed Register Description

to '1') followed by a branching to the new descriptor (FRDA) defined in the channel specification of the CCS.

For HDLC, TMB, TMR the rest of a frame which was only partially transferred before suspension of the receive descriptor is aborted, the new descriptor is related to the next frame. An interrupt with FI, ERR is issued. For V.110/X.30 and TMA data bits might get lost. An interrupt with ERR is issued.

### – receive off

RI = 0, RO = 1, RA = 0 (clears a previous receive abort condition, sets off condition, affects only the serial interface)

This channel command sets the receiver into the receive off condition. The receive channel is disabled completely at the serial interface, i.e. the receive deformatter RD is reset and the receive buffer RB is not accessed for this channel. A currently processed frame (HDLC, TMB, TMR mode) is not properly finished with any status information. The data stored in the RB at that time is still transferred to the shared memory.

After the receive off condition is cleared by another channel command:

- in HDLC, TMB, TMR (V.110/X.30, TMA) mode the device waits for a new frame (10-octet frame, nothing) to begin and then starts filling RB again. If the receive off command lead to an improper finishing of a frame (data, data), the new frame (data, data) is concatenated with the finished one. To avoid this problem there are two suggestions:
  - a) issue a receive abort channel command and wait for 32 (240, 8) bits for this channel to be processed before issuing the receive off command.
  - b) wait in the receive off condition until the RB is emptied for this channel (i.e. for at most 8 PCM frames if the MUNICH32 has sufficient access to the shared memory) and leave the receive off condition by a receive initialization command.

The receive off channel command is ignored in case of any kind of loop.

### – receive abort

RI = 0, RO = 1, RA = 1 (clears a previous receive off condition, sets a receive abort condition, affects only the serial interface)

This receive channel command sets the receiver into the receive abort condition. In this condition it receives (instead of the normally received bits)

logical '1' bits for HDLC

logical '0' bits for V.110/X.30, TMB, TMR

logical '0' bits for unmasked bits in TMA mode

logical '1' bits for masked bits in TMA mode

irrespective of the INV bit.

This leads to

- For HDLC: a currently processed frame is aborted after  $\leq 7$  received bits for this channel, leading to a RA set in the status of the frame and an interrupt with set FI and ERR bits only or to an interrupt with set SF, FI and ERR bits. If the receiver was

## Detailed Register Description

in the flag interframe time-fill state it will lead to an interrupt with set IFC bit after  $\leq 15$  received bits.

- For V.110/X.30: if the receiver was in the synchronized frame state it will go to the unsynchronized state after  $\leq 240$  bits and issue a LOSS bit in the status of the current receive descriptor. It will also issue an interrupt with set ERR bit and (unless all E-, S-, X-bits were 0 previously) issue one or two interrupts with FRC set and having all E-, S-, X-bits at 0 in the last one.
- For TMB: a currently processed frame is aborted after  $\leq 15$  received bits for this channel, leading to an interrupt with FI set but ERR on 0, the status of this frame is always 00<sub>H</sub>.
- For TMR: a currently processed frame is aborted after  $\leq 31$  received bits for this channel, leading to an interrupt with FI set but ERR on 0, the status of this frame is always 00<sub>H</sub>.
- For TMA: the device receives the inverse of the fill/mask bits programmed for this channels.

*Note 1: It is advisable to clear the receive abort condition via a receive off command for V.110/X.30 mode, the TMB and the TMR mode.*

- 2. After issuing a receive abort channel command it is advisable to stay in this condition during at least 16, 240, 16, 32, 8 bits of the channel for HDLC, V.110/X.30, TMB, TMR, TMA respectively.*

### – receive jump

RI = 1, RO = 0, RA = 0 (clears a previous receive abort or receive off condition, affects only the DMA interface)

During normal operation branching to a new descriptor (FRDA) is possible without interrupting the current descriptor and aborting the received frame (HDLC, TMB, TMR) or received data (V.110/X.30, TMA).

The DMA controller will proceed finishing the current receive descriptor as usual either with a frame end condition or with the corresponding data buffer completely filled and afterwards branch to the new descriptor specified by FRDA. Thus a received frame may be splitted on 'old' and 'new' descriptors.

### – receive initialization

RI = 1, RO = 0, RA = 1 (clears a previous receive abort or receive off condition, affects the DMA and serial interface)

Before the MUNICH32 has got a receive initialization command it will not receive anything properly in a channel. This command should therefore be the first channel command after a pulse at the reset pin for a channel to be used. FRDA is then the address of the starting point of the receive descriptor chaining list.

If the command is issued during normal operation it only affects the DMA interface. The current receive descriptor is suspended without writing the second long word with the status, no interrupt is generated. For HDLC, TMB, TMR the rest of a frame which was only partially transferred before the suspension of the receive descriptor is



---

## Detailed Register Description

aborted, the new descriptor (FRDA) is related to the next frame.  
For V.110/X.30 and TMA data bits might get lost.

### General Notes to Receive Commands:

1. After a pulse at the reset pin a channel having a time slot with  $RTI = 0$  should be issued receive off commands until it is supposed to be used.
2. When it is supposed to be used it should be issued a receive initialize command before using any other receive channel command.
3. To shut down a channel in receive direction one should first set it into the receive abort condition for the time specified there and then set it into the receive off condition.
4. Before changing the MODE, CRC, CS, TRV, INV, TFLAG bits of a channel or its RTI or time slot assignment or its fill/mask bits it should have been shut down. The bits should be changed while issuing the receive off command.
5. To revive a channel after it has been shut down one should use the receive initialization command.
6. To switch to a new starting point of a receive descriptor chain one should preferably use the receive jump command, only exceptionally the fast receive abort command and never the receive initialize command.
7. To issue channel commands not affecting the receive side one should issue
  - a receive clear command if neither a receive off nor a receive abort condition is set
  - a receive off command if a receive off condition is set
  - a receive abort command if a receive abort condition is set.
8. Combinations of the bits RI, RO, RA not in this description are reserved and are not allowed to be used.

### 2. First Transmit Command Group

#### – transmit clear

TI = 0, TO = 0, TA = 0 (clears a previous transmit abort or transmit off condition, affects only the serial interface)

- if the channel was never initialized by a transmit initialization command it has no effect
- if it was initialized previously it clears a transmit off or transmit abort condition set by a previous channel command
- if no transmit off or transmit abort condition is set it has no effect

#### – fast transmit abort

TI = 0, TO = 0, TA = 1 (clears a previous transmit abort or transmit off condition, affects only the DMA interface)

This abort is performed in the DMA controller and does not interfere with the current transmission on the serial interface and the transfer between the TF and TB. If this abort is detected the current descriptor is suspended and the frame or data transferred to the TB is aborted. The next frame beginning in the transmit descriptor (FTDA) defined in the channel specification of the CCS will be started immediately.

## Detailed Register Description

For HDLC, TMB, TMR the first part of the frame of the suspended descriptor is sent and append by 011 1111 1111 111 for HDLC

at least 00<sub>H</sub> for TMB

at least 00 00<sub>H</sub> for TMR

Afterwards the next frame is started.

For V.110/X.30 three 10-octet frames with errors in the synchronization pattern are sent after the data of the suspended descriptor, afterward the next data are sent in correct frames.

For TMA a TFLAG (FA = 1) or FF<sub>H</sub> (FA = 0) is sent in at least one time slot after the data of the suspended descriptor, afterwards the next data are sent.

### – transmit off

TI = 0, TO = 1, TA = 0 (clears a previous transmit abort condition, sets a transmit off condition, effects only the serial interface)

The transmit channel is disabled immediately, i.e. the transmit formatter is reset and the transmit buffer is not accessed for this channel. The output time slots are tristated. Upon leaving the transmit off mode the transmit link list must be initialized by a transmit reinitialize command. Otherwise the transmission will be started with the remaining data still stored in TB and continue with the old link list. If a loop condition is set the transmit off does not reset the transmit formatter, it only tristates the serial output line.

After the transmit off condition is cleared by the transmit initialize command.

- In HDLC, TMB, TMR, V.110/X.30 the device starts with the interframe time-fill

7E for HDLC and IFTF = 0

FF for HDLC and IFTF = 1

00 for TMB, TMR, V.110/X.30

and then with the frame in the descriptor at FTDA. For V.110/X.30 this descriptor **must** have the V.110-bit set and point to the E-, S-, X-bits, the data are then at the next transmit descriptor.

- In TMA mode the device starts with the interframe time-fill

TFLAG for FA = 1

FF<sub>H</sub> for FA = 0

and then with the data in the descriptor at the FTDA.

## Detailed Register Description

### – transmit abort

TI = 0, TO = 1, TA = 1 (clears receive off condition, sets transmit abort condition, affects only the serial interface)

This abort is performed in the transmit formatter at the serial interface. The currently transmitted frame is aborted

by

- 011 1111 1111 1111 for HDLC
- 00<sub>H</sub> for TMB
- 0000<sub>H</sub> for TMR
- 3 frames with erroneous synchronization pattern for V.110/X.30
- TFLAG for TMA, FA = 1
- FF for TMA, FA = 0.

Afterwards or – if no frame is currently sent directly inter frame time fill:

- 7E for HDLC and IFTF = 0
- FF for HDLC and IFTF = 1
- 00 for TMB, TMR, V.110/X.30
- TFLAG for TMA, FA = 1
- FF for TMA, FA = 0

is sent.

During transmit abort the TF does not access the transmit buffer. The handling of the link list is not affected by the transmit abort, i.e. the device keeps the TB full. When the transmit abort is withdrawn the transmit formatter continues the transmission with the data stored in TB. In the case of HDLC or TMB or TMR mode the remaining data of the aborted HDLC or TMB frame is sent as a new independent frame. To avoid this problem the link list must be reinitialized by a transmit initialization command together with the revoking of the transmission abort.

Another proper use of the transmit abort command consists in setting the last descriptor of the last frame to be transmitted with HOLD = 1 and waiting for the device to poll the HOLD bit (ITBS + 2) times where ITBS is the number of long words assigned to this channel currently. Afterwards TB is empty and the transmit abort then issued does not abort a currently sent frame. The same procedure can also be used for the transmit off command.

### – transmit jump

TI = 1, TO = 0, TA = 0 (clears a transmit off and transmit abort condition, affects only the DMA interface)

This bit is set only during normal operation. Then MUNICH32 branches to the transmit descriptor (FTDA) specified in the CCS after finishing the current transmit descriptor without interrupting or aborting the transmitted frame.

The DMA controller will proceed finishing the current transmit descriptor as usual and afterwards branch to the new descriptor specified by FTDA. If the current descriptor does not include a frame end (FE = 0) (HDLC, TMB, TMR) the DMA controller will link the following data section(s) of the 'new' descriptor chain to the opened frame. This may generate unexpected frames.

## Detailed Register Description

### – transmit initialization

TI = 1, TO = 0, TA = 1 (clears a previous transmit abort condition, affects the DMA interface and the serial interface)

Before the MUNICH32 has got a transmit initialization command it will not transmit anything properly in the channel. This command should therefore be the first channel command after a pulse at the reset pin for a channel to be used.

FTDA is then the address of the starting point of the transmit descriptor for chaining list. In this case the transmit initialize command should be accompanied by the NITBS bit set and a reasonable value for ITBS ( $0 < ITBS < 64$ ).

If the command is issued during normal operation it only affects the DMA. The MUNICH32 stops processing of the current link list and branches to the transmit descriptor at the FTDA address. The data stored in the TB are discarded and the TB is filled with the data of the new descriptor.

## 3. Second Transmit Command Group

### – Transmit HOLD

TH; setting this bit causes the device to finish transmission of the current frame (HDLC, TMB, TMR mode) the current data (TMA -mode) or leads to an abort with 3 frames with '0'-bits (V.110/X.30-mode). Afterwards

for	HDLC mode and IFTF = 1	FF <sub>H</sub> fill characters
	HDLC mode and IFTF = 0	7E <sub>H</sub> fill characters
	V.110/X.30-mode	00 <sub>H</sub> fill characters
	TMA mode and FA = 1	TFLAG fill characters
	TMA mode and FA = 0	FF <sub>H</sub> fill characters
	TMB/TMR	00 <sub>H</sub> fill characters

are sent until TH is withdrawn by a further action specification affecting the channel specification of this channel.

Afterwards no further access to the TB from TF is done, therefore no further data are fetched from the shared memory and the polling of a possible hold bit in the transmit descriptor stops.

To send necessary frames/data before the transmit hold is active one should use the proper procedure described under the transmit abort command.

## General Notes to Transmit Commands:

1. After a pulse at the reset pin a channel having a time slot with TTI = 0 should be issued transmit off commands and TH = 1 until it is supposed to be used.
2. When it is supposed to be used it should be issued a transmit initialization command and TH = 0 before using any other transmit channel commands (together with NITBS = 1, ITBS  $\neq$  0).
3. To shut down a channel in transmit direction one should first set it into the transmit abort condition or use the TH bit with the proper procedure. One should leave it in

## Detailed Register Description

that condition for 32, 240, 32, 32, 8 bits for HDLC, V.110/X.30,TMB, TMR, TMA respectively and then set it into the transmit off condition.

4. Before changing the MODE, CRC, CS, TRV, INV, TFLAG bits or TTI or time slot assignment or the fill/mask bits or the ITBS the channel should be shut down. The bits should be changed while issuing the transmit off command.
5. To revive a channel after it has been shut down one should use the transmit initialization command.
6. For V.110/X.30-mode the first descriptor after reviving from shut down or initialization after reset **must** have the V.110-bit set and contain the E-, S-, X-bits.
7. To switch to a new starting point of a transmit descriptor chain one should preferably use the transmit jump command, only exceptionally the fast transmit abort command and never the transmit initialize command.
8. To issue channel commands not affecting the transmit side one should issue
  - TH with the last set value
  - a transmit clear command if neither a transmit off nor a transmit abort condition is set
  - a transmit off if a transmit off condition is set
  - a transmit abort if a transmit abort condition is set.
9. Bit combinations in the first transmit command group not described are reserved.
10. Set NITBS = 1 preferably in conjunction with a transmit initialize and transmit clear command if TB is to be newly configured, otherwise set NITBS = 0.

**TFLAG:** Transparent mode Flag; these bits are only used in the transparent mode A and constitute the fill code for flag stuffing and for flag filtering. These bits must be set to '0' if subchanneling is used in transparent mode A. Bit No. 15 is the first bit of the flag to be received/transmitted.

**NSF:** No Short Frame suppression; NSF = 1 is only allowed in combination with HDLC mode and CS = 1.

In this mode the MUNICH32 transfers all data to the shared memory even if only one byte (or more) per 'frame' is received. No short frame interrupt and no short frame status bit will be generated in this case.

*Note: CRC is still calculated and checked and e.g. a frame of 1 or 2 byte length (in CRC16 mode) will always cause an FI + ERR interrupt.*

---

**Detailed Register Description****Receive Frame Examples:**

a) 0x7E, data byte, 0x7E

- data byte copied to shared memory + frame end
- status SF-bit set
- no SF indication interrupt generated
- FI indication interrupt generated
- ERR interrupt generated due to wrong CRC'

b) 0x7E, data byte = 0xFC (or 0xFD or 0x7F), 0x7E

- no data byte copied to shared memory
- SF and FI interrupt generated

**CS:** CRC Select; only used in HDLC mode. Setting this bit to '1' causes the MUNICH32 to transfer the CRC bits to the data section in the shared memory. In receive direction the CRC check is carried out whereas in transmit direction the CRC generation is suppressed, see **Chapter 2.4** for more details.

**INV:** Inversion; If this bit is set, all data of the channel transmitted or received by the MUNICH32 is inverted.

**CRC:** Cyclic Redundancy Check; in HDLC mode this bit determines the CRC generator polynomial: When the CRC bit is set to '1' the 32-bit CRC is performed, otherwise the 16-bit CRC; for TMB/TMR mode this bit distinguishes:

TMB: CRC = '0'

TMR: CRC = '1'

for all other modes this bit has to be set to '0'.

## Detailed Register Description

**TRV:** Transmission Rate of V.110/X.30. These signals determine the number of repeated D-bits in a V.110/X.30 frame.

**Table 9**

TRV	No. of Repetitions	Transmission Rate
00	7	600 bit/s
01	3	1200 bit/s
10	1	2400 bit/s
11	0	4.8, 9.6, 19.2, 38.4 kbit/s

*Note: In the other modes these bits must be set to '00'.*

**FA:** Flag Adjustment selected (in HDLC mode) or flag filtering (selected in transparent mode A only if all fill/mask bits of the corresponding slots are '1'). In all other modes this bit must be set to '0'. If flag adjustment is selected in HDLC mode the number of interframe time-fill characters is FNUM minus one eighth of the number of zero insertions in the frame proceeding the interframe time-fill and belonging to the same transmit descriptor as FNUM. If flag filtering is selected and fills a physical time slot in transparent mode A the flag specified in TFLAG is recognized and extracted from the data stream. In transmit direction the flag TFLAG is sent in all exception conditions, i.e. abort, idle state etc.; if flag filtering is not selected '1'-bits are sent in this case. Flag filtering is only allowed if all fill/mask codes are set to '1', i.e. subchanneling is not allowed. If flag filtering is not selected the bits in TFLAG have to be set to 0 for TMA.

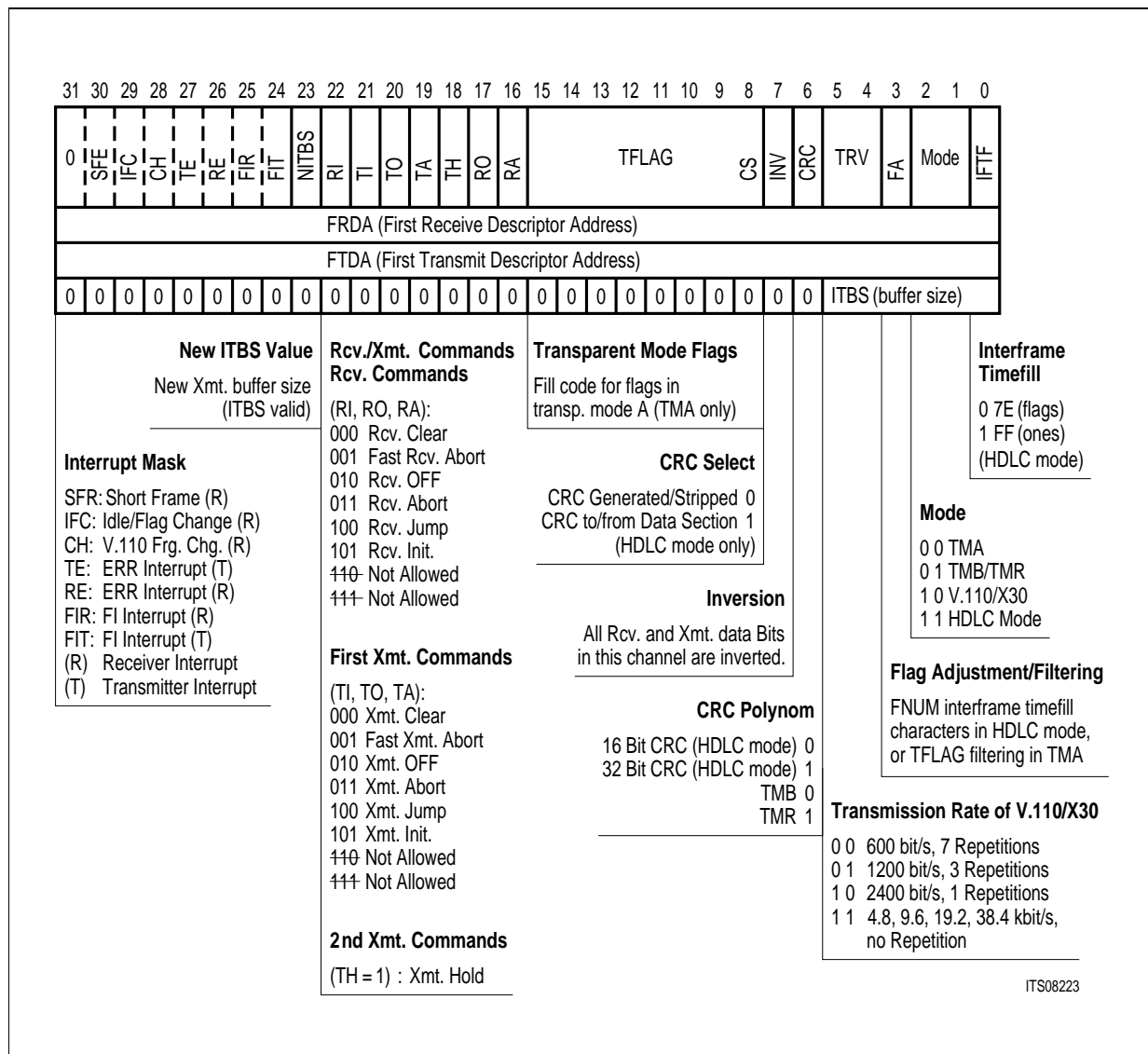
**MODE:** Defines the transmission mode:  
11: HDLC mode  
10: V.110/X.30 mode  
00: Transparent mode A  
01: Transparent mode B or transparent mode R.

**ITTF:** Interframe Time-Fill: this bit determines the interframe time-fill for HDLC mode:  
ITTF = 0: AE<sub>H</sub> characters are sent as interframe time-fill  
ITTF = 1: FF<sub>H</sub> characters are sent as interframe time-fill.

**FRDA:** First Receive Descriptor Address points to the beginning of the receive data chaining list.  
This descriptor is only interpreted with a fast receive abort or a receive jump or a receive initialization command. It is read but ignored with any other receive channel command.

## Detailed Register Description

- FTDA:** First Transmit Descriptor Address points to the beginning of the transmit data chaining list.  
This descriptor is only interpreted with a fast transmit abort or a transmit jump or a transmit initialization command. It is read but ignored with any other transmit channel command.
- ITBS:** Individual Transmit Buffer Size; for undisturbed transmission an on-chip transmit buffer with a total size of 64 long words stores the data before formatting and transmitting. The individual buffer size specifies the part of the on chip transmit buffer allocated to the channel. This allows a variable data buffer size if NITBS = 0, ITBS has to be set to 0 also; it is then read but ignored. (see **Chapter 2.3**).



**Figure 80**  
**Channel Specification**



## Detailed Register Description

### 4.2.6 Current Receive and Transmit Descriptor Address

31	16	15	0
Current Receive Descriptor Address Channel 0			
.			
.			
.			
Current Receive Descriptor Address Channel 31			
Current Transmit Descriptor Address Channel 0			
.			
.			
.			
Current Transmit Descriptor Address Channel 31			

For easier monitoring of the link lists the addresses of the just processed descriptors are written into the CCS. MUNICH32 changes the current descriptor address at the same time when it branches to the next descriptor.

## Detailed Register Description

### 4.3 Transmit Descriptor

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FE	HOLD	HI	NO												
Transmit Data Pointer															
Next Transmit Descriptor Pointer															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V.110	0	0	0	CSM	FNUM										
Transmit Data Pointer															
Next Transmit Descriptor Pointer															

- FE: Frame End; this bit is valid in all modes.
- It indicates that after sending the data in the transmit data section
- the device generates an interrupt with FI bit set for HDLC, TMB, TMR, TMA  
ERR bit set for V.110/X.30
  - the device then sends
    - $(FNUM + 1) \times 7E_H$  for HDLC, IFTF = 0
    - $7E, (FNUM - 1) \times FF_H, 7E$  for HDLC, IFTF = 1,  $FNUM \geq 1$
    - $7E$  for HDLC, IFTF = 1,  $FNUM = 0$
    - $(FNUM + 1) \times 00_H$  for TMB, TMR ( $FNUM \geq 1$ )
    - $000_H$  for TMR,  $FNUM = 0$
    - $(FNUM + 1) \times TFLAG$  for TMA, FA = 1
    - $(FNUM + 1) \times FF_H$  for TMA, FA = 0
    - three frames with synchronization errors for V.110/X.30
- before starting with the data of the next transmit descriptor. If the data of the next transmit descriptor are not available in time (e.g. because the descriptor has FE **and** HOLD set) the device sends the interframe time-fill indefinitely.

## Detailed Register Description

- HOLD:** If the MUNICH32 detects a hold bit it
- generates an interrupt with ERR bit set if FE = 0 or V.110/X.30 mode
  - sends the data in the current transmit data section
  - generates the FCS bits for HDLC and CS = 0 and CSM = 0
  - the device then sends at least
    - $(FNUM + 1) \times 7E_H$  for HDLC, IFTF = 0
    - $7E, FNUM \times FF_H$  for HDLC, IFTF = 1
    - $(FNUM + 1) \times 00_H$  for TMB, TMR ( $FNUM \geq 1$ )
    - $0000_H$  for TMR,  $FNUM = 0$
    - $(FNUM + 1) \times TFLAG$  for TMA, FA = 1
    - $(FNUM + 1) \times FF_H$  for TMA, FA = 0
    - three frames with synchronization errors for V.110/X.30.
  - It polls the HOLD bit and the next transmit descriptor address, but does not branch to a new descriptor until the HOLD bit is reset. The next transmit descriptor address is read but not interpreted as long as HOLD = 1. Therefore it can be changed together with setting HOLD = 0. The polling occurs at most every 8 valid clock cycles of the channel and corresponds with internal requests from TF to TB.
  - The device sends interframe time-fill until HOLD = 0 is polled. The HOLD condition is also discarded if a transmit jump, fast transmit abort or transmit initialization command is detected during the polling. The MUNICH32 then branches to the transmit descriptor determined by FTDA even though the HOLD bit of the current transmit descriptor may still be '1'.
- HI:** Host initiated Interrupt; if the HI bit is set, MUNICH32 generates an interrupt with set HI bit after transferring all data bytes.
- NO:** This byte number defines the number of bytes stored in the data section to be transmitted. A transmit descriptor and the corresponding data section must contain at least either one data byte or a frame end indication. Otherwise an interrupt with set ERR bit is generated.
- V.110:** This bit indicates that in the corresponding data section the E-, S- and X-bits of the following V.110/X.30 frame are stored. MUNICH32 reads these bits and inserts them into the next possible V.110/X.30 frame. The data section may contain only two bytes specified in the next figure. The first transmit descriptor after a transmit initialization channel command **must** have this bit set if it revives the channel from a transmit off condition or after a pulse at the reset pin.

## Detailed Register Description

### Intel Mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
E7	E6	E5	E4	E3	E2	E1	SB	SA	X	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

### Motorola Mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SA	X	0	0	0	0	0	0	E7	E6	E5	E4	E3	E2	E1	SB

**CSM:** CRC Select per Message: This bit is only valid in HDLC mode with CS = 0 and only in conjunction with the FE bit set. If set, it means that no FCS is generated automatically for the frame finished in this transmit descriptor.

**FNUM:** FNUM denotes the number of interframe time-fill characters between 2 HDLC or TMB frames. For X.30/V.110 these bits have to be set to '0'.

FNUM = 0 means that after the current frame only 1 character (7E<sub>H</sub> for HDLC and 00<sub>H</sub> for TMB, 000<sub>H</sub> for TMR, TFLAG, TFLAG for TMA, FA = 1; FF<sub>H</sub> for TMA, FA = 0) is sent before the following frame (shared flags).

FNUM = 1 means that after the current frame 2 characters (7E<sub>H</sub> 7E<sub>H</sub> for HDLC and 00<sub>H</sub> 00<sub>H</sub> for TMB and TMR, TFLAG, TFLAG for TMA, FA = 1; FF FF<sub>H</sub> for TMA, FA = 0) are sent before the following frame (non shared flags).

FNUM = 2 means that after the current frame 3 characters (7E<sub>H</sub> 7E<sub>H</sub> 7E<sub>H</sub> (IFTF = 0) or 7E<sub>H</sub> FF<sub>H</sub> 7E<sub>H</sub> (IFTF = 1) for HDLC and 00<sub>H</sub> 00<sub>H</sub> 00<sub>H</sub> for TMB and TMR, TFLAG, TFLAG, TFLAG for TMA, FA = 1; FF FF FF<sub>H</sub> for TMA, FA = 0) are sent.

## Detailed Register Description

FNUM = k means that after the current frame k + 1 characters are sent

(k + 1) times 7E<sub>H</sub> for ITFT = 0 and HDLC

7E<sub>H</sub>, (k – 1) times FF<sub>H</sub>, 7E<sub>H</sub> for ITFT = 1 and HDLC

(k + 1) times 00<sub>H</sub> for TMB, TMR

(k + 1) times TFLAG for TMA, FA = 1

(k + 1) times FF<sub>H</sub> for TMA, FA = 0.

For HDLC mode FNUM is reduced by one eighth of the number of zero insertions if FA is set. If the reduction would result in a negative number of interframe time-fill characters it is set to 0.

**Transmit Data Pointer:** This 32-bit pointer contains the start address of the transmit data section. Although MUNICH32 works only long word oriented, it is possible to begin a transmit data section at an uneven address. The two least significant bits (ADD) of the transmit data pointer determine the beginning of the data section and the number of data bytes in the first long word of the data section, respectively.

ADD:            00 = 4 bytes  
                   01 = 3 bytes  
                   10 = 2 bytes  
                   11 = 1 byte

MUNICH32 reads the first long word and discards the unused least significant bytes. The NO establishes (determines) the end of the data section, whereas the remainder of  $\lfloor (NO - ADD) \div 4 \rfloor$  defines the number of bytes in the last long word of the data section.

MUNICH32 reads the last long word and discards the unused most significant bytes of the last long word.

If the first access is the same as the last access, ADD specifies the beginning of the data section and NO the number of data bytes in the long word. All unused bytes are discarded.

## Detailed Register Description

For example (Intel mode):

1) ADD = 01, NO = 8

11	10	01	00	
byte 2	byte 1	byte 0	–	
byte 6	byte 5	byte 4	byte 3	3 long words are read
–	–	–	byte 7	

2) ADD = 00, NO = 8

11	10	01	00	
byte 3	byte 2	byte 1	byte 0	
byte 7	byte 6	byte 5	byte 4	2 long words are read
–	–	–	–	

3) ADD = 10, NO = 1

11	10	01	00	
–	byte 0	–	–	
–	–	–	–	1 long word is read!
–	–	–	–	

For example (Motorola-mode):

1) ADD = 01, NO = 8

11	10	01	00	
–	byte 0	byte 1	byte 2	
byte 3	byte 4	byte 5	byte 6	3 long words are read
byte 7	–	–	–	

2) ADD = 00, NO = 8

11	10	01	00	
byte 0	byte 1	byte 2	byte 3	
byte 4	byte 5	byte 6	byte 7	2 long words are read

3) ADD = 10, NO = 1

11	10	01	00	
–	–	byte 0	–	1 long word is read!

---

## Detailed Register Description

### Next Transmit Descriptor Pointer:

This 32-bit pointer contains the start address of the next transmit descriptor. After sending the indicated number of data bytes, MUNICH32 branches to the next transmit descriptor to continue transmission. The transmit descriptor is read entirely at the beginning of transmission and stored in an on-chip memory. Therefore all information in the next descriptor must be valid when MUNICH32 branches to this descriptor when  $HOLD = 0$ . For  $HOLD = 1$  the next transmit descriptor pointer is polled together with  $HOLD$ ; the next transmit descriptor must be valid, when  $HOLD = 0$  is polled.

This pointer is not used if a transmit jump, fast transmit abort or transmit initialization channel command is detected while the MUNICH32 still reads data from the current transmit descriptor or polls the  $HOLD$  bit. In this case FTDA is used as a pointer for the next transmit descriptor to be branched to.

## Detailed Register Description

### 4.4 Receive Descriptor

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	HOLD	HI	NO												
FE	C	0	BNO												
Receive Data Pointer															
Next Receive Descriptor Pointer															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Status								0	0	0	0	0	0	0	0
Receive Data Pointer															
Next Receive Descriptor Pointer															

The receive descriptor contains 4 long words; the first, third and fourth have to be written by the CPU, the second is written by the MUNICH32 when it branches to the next receive descriptor or when it starts polling the HOLD bit.

*Note: The MUNICH32 branches to a next descriptor **without** writing the second long word if the receive initialization command is used during normal operation (see **Chapter 4.2.4**)*

**HOLD:** Setting the HOLD bit by the host prevents the device from branching to the next descriptor. The current data section is still filled.

- Afterwards the second descriptor long word is written by the MUNICH32.  
For HDLC, TMB, TMR the FE and C-bit is set. If the frame could not completely be stored into the data section the RA bit is set in the status.  
An interrupt with set FI bit is generated, and in case the frame was aborted, the ERR bit is also set.  
For TMA, V.110/X.30 the C-bit and the RA bit is set and an interrupt with set ERR but with FI = 0 is generated.
- Afterwards the device starts polling the HOLD bit, received data, and received events normally leading to interrupts (with RT = 1) are discarded until HOLD = 0 is polled. Each 1 ... 4 byte data word or interrupt event normally leading to an access now results in a poll cycle.  
Whenever HOLD = 1 is polled the next receive descriptor address is read but ignored.
- When HOLD = 0 is polled
  - for HDLC, TMB, TMR the device continues to discard data until the end of a received frame or an event leading to an interrupt (with RT = 1) is



## Detailed Register Description

detected. Afterwards the next received frame is transferred into the next receive descriptor. Interrupts are also generated again.

- For V.110/X.30, TMA the device puts the next data into the next receive descriptor. Interrupts are also generated again.

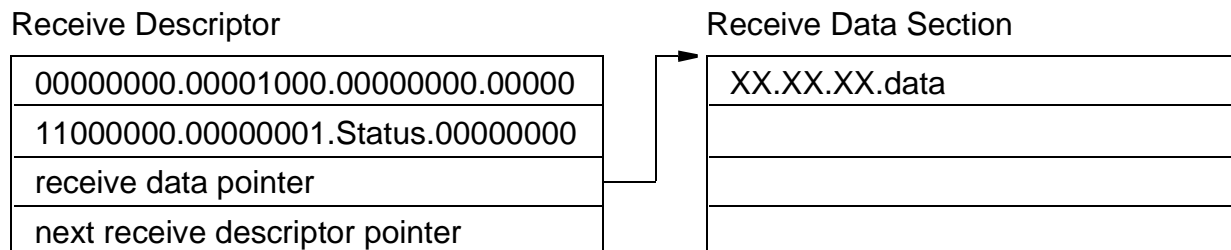
The HOLD condition is also discarded upon detection of a receive jump, fast receive abort or receive initialization command. The MUNICH32 then branches to the receive descriptor determined by FRDA even though the HOLD bit in the current receive descriptor may still be '1'.

**HI:** Host initiated interrupt; if the HI bit is set, MUNICH32 generates an interrupt with set HI bit after receiving all data bytes.

**NO:** This byte number defines the size of the receive data section allocated by the host. Because MUNICH32 always writes long words the number of bytes (data section size) must be a multiple of 4 and greater or equal to 4. The maximum data section size is 8188 bytes.

After reception of an HDLC frame with a data byte number not divisible by 4 MUNICH32 first transfers the greatest entire ([number of data bytes/4]) in long words. Then the remainder of the data bytes is transferred in another long word, where the non-significant bytes are filled with random values. They should not be interpreted.

For example a HDLC frame with one data byte is received:



The data bytes are stored into the receive data section according to the Little Endian convention (Intel mode) or Big Endian convention (Motorola mode).

**FE:** Frame End: The frame end bit is '1' only in HDLC, TMB, TMR mode and indicates that a receive frame has ended in this receive descriptor. For TMA, V.110/X.30 the bit is always '0'.

FE = 0 in HDLC, TME, TMR mode means that frame continues in the next receive descriptor or that it filled the current receive data section exactly (BNO = NO). In this case the next receive descriptor will have FE = 1, C = 1, BNO = 0 and no data bytes are stored in the corresponding data section.

**C:** This bit is set by MUNICH32 if

- it completes filling the data section normally (BNO = NO)  $\Rightarrow$  FE = 0, status = 00
- it was aborted by a fast receive abort channel command  $\Rightarrow$  status = 02

## Detailed Register Description

- for HDLC, TMB, TMR if the end of a frame was stored in the receive data section  $\Rightarrow$  FE = 1, status gives the receive status determined by RD (interrupt with set FI bit is generated)
- for V.110/X.30 mode if the 3 contiguous frames with errors in the synchronization pattern are received  $\Rightarrow$  FE = 0, status = 20 or status = 21 interrupt with set ERR bit
- for V.110/X.30 mode if the data could not be transferred to the shared memory due to RB buffer inaccessibility  $\Rightarrow$  FE = 0, status = 01 or status = 21 interrupt with set ERR bit.

C indicates that the second long word of the receive descriptor was written by the MUNICH32. Afterwards the MUNICH32 writes the next receive descriptor address into CCS. Then it branches to this descriptor immediately.

**BNO:** MUNICH32 writes the number of data bytes it has stored in the current data section into BNO.

**Status:** The MUNICH32 writes the status information into the status byte whenever it sets the C-bit. If the status information is not 00 or 40 an interrupt with ERR bit set is generated. The status is then a means to locate or analyze the receive error.

The following table gives a general overview over the different status bits in relation to the channel modes.

7	6	5	4	3	2	1	0
0	SF	LOSS	CRCO	NOB	LFD	RA	ROF

HDLC CS = 0

0	NI	0	ILN	IL	I	I	I
---	----	---	-----	----	---	---	---

HDLC CS = 1

0	0	0	ILN	IL	I	I	I
---	---	---	-----	----	---	---	---

V.110/X.30

0	0	I	0	0	0	IF	I
---	---	---	---	---	---	----	---

TMB

0	0	0	0	IL	I	IF	I
---	---	---	---	----	---	----	---

TMR

0	0	0	0	IL	I	IF	I
---	---	---	---	----	---	----	---

TMA

0	0	0	0	0	0	IF	0
---	---	---	---	---	---	----	---

Where '0' means that in the corresponding mode the bit is always '0'. It should not be interpreted though to be upward compatible to future versions.

## Detailed Register Description

NI	means the bit may be '1' or '0' but does not cause an interrupt with set ERR bit.
ILN	means that it may be '1' or '0' but should not be evaluated if LFD or NOB is also '1'.
IL	means that it may be '1' or '0' but should not be evaluated if LFD = 1.
I	means that it may '1' or '0'.
IF	means that it may be '1' only after a fast receive abort channel command or detection of a HOLD bit in the current receive descriptor.

I, IF, IL, ILN lead to an interrupt with ERR bit set.

*Note: For HDLC, TMB, TMR the status word is only valid if the FE bit is set.*

The meaning of the individual status bits is as follows:

SF = 1	(HDLC mode with CS = 0 only): The device has received a frame with ≤ 32 bit between start flag and end flag or end abort flag for CRC16 ≤ 48 bit between start flag and end flag or end abort flag for CRC32 i.e. BNO was 1 or 2.
LOSS = 1	Three contiguous frames with errors in the synchronization pattern were detected.
CRCO = 1	A frame with a CRC error was detected CRCO = 0 means the frame had no CRC error.
NOB = 1	A frame whose bit content was not divisible by 8 was detected. NOB = 0 means that the frame content was divisible by 8.
LFD = 1	Long frame detected. If this bit is set a frame whose bit content was > MFL was detected and aborted. The reception will be continued as soon as a flag is recognized.
RA = 1	Receive Abort; this bit indicates that for HDLC: the frame was ended by an abort flag (7F <sub>H</sub> ) or by a receive abort command or a fast receive channel command or by a HOLD bit in the current receive descriptor. for V.110/X.30, TMB, TMR, TMA that the frame or data were aborted by a fast receive abort channel command or a HOLD bit set in the current receive descriptor.
ROF = 1	An overflow of the internal buffer RB has occurred and lead to a loss of data.

*Note: If ROF without FO interrupt is generated for a channel*

- for HDLC, TMB, TMR only the last part of one frame has been lost.
- For V.110/X.30 only data but no status information (change E-, S-, X-bits, Loss) has been lost.

---

## Detailed Register Description

*Note: In case of multiple errors all relevant bits are set.*

*In case of  $ROF = 1$  only the error conditions of the frame within which the overflow occurred are reported. Later frames that are aborted do not change the status.*

**Receive Data Pointer:** This 32-bit pointer contains the start address of the receive data section.

**Receive Descriptor Pointer:** This 32-bit pointer contains the start of the next receive descriptor.

It is not used if a receive jump, fast receive abort or receive initialize command is detected while the MUNICH32 still writes data into the current receive descriptor or polls the HOLD bit. In this case FRDA is used as a pointer for the next receive descriptor to be branched to.

## 5 Application Notes

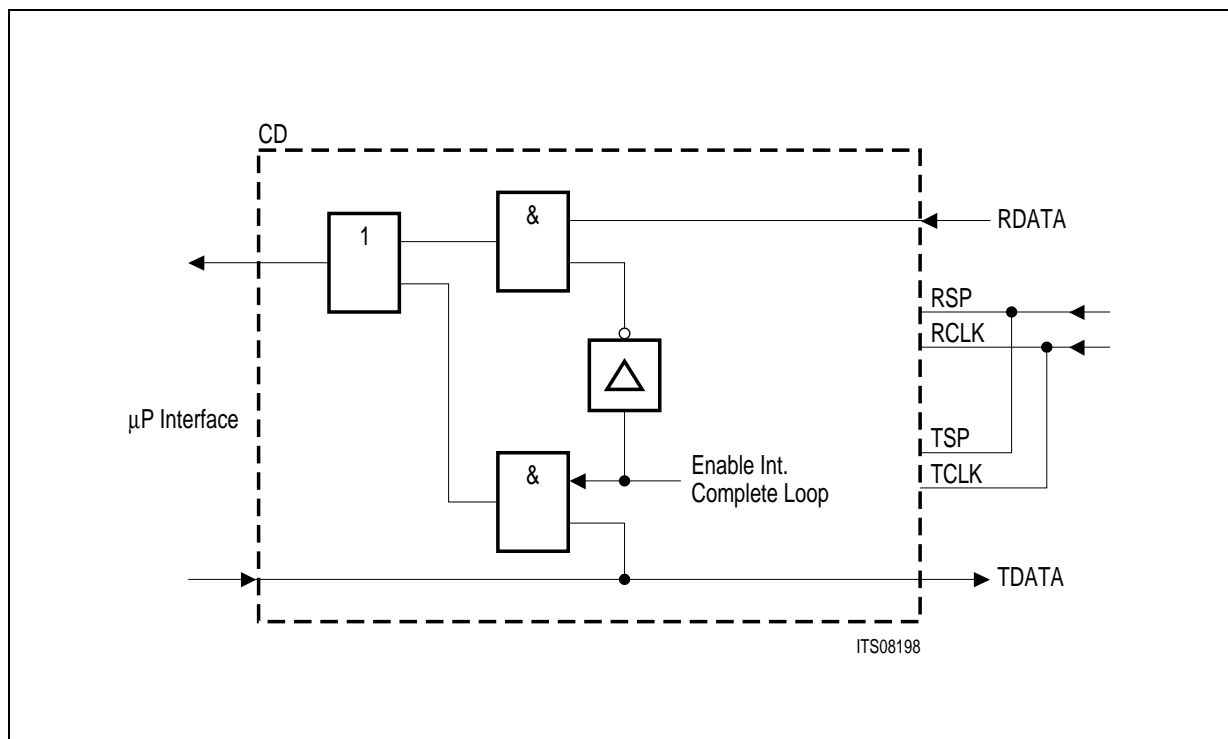
### 5.1 Test Loops

#### 5.1.1 Test Loop Definitions for the MUNICH32

Two basic types of test loops are provided by the MUNICH32, internal and external. Each of these types is further subdivided into channelwise and complete test loops thus providing four possible test loops.

##### 5.1.1.1 Internal Complete Test Loop

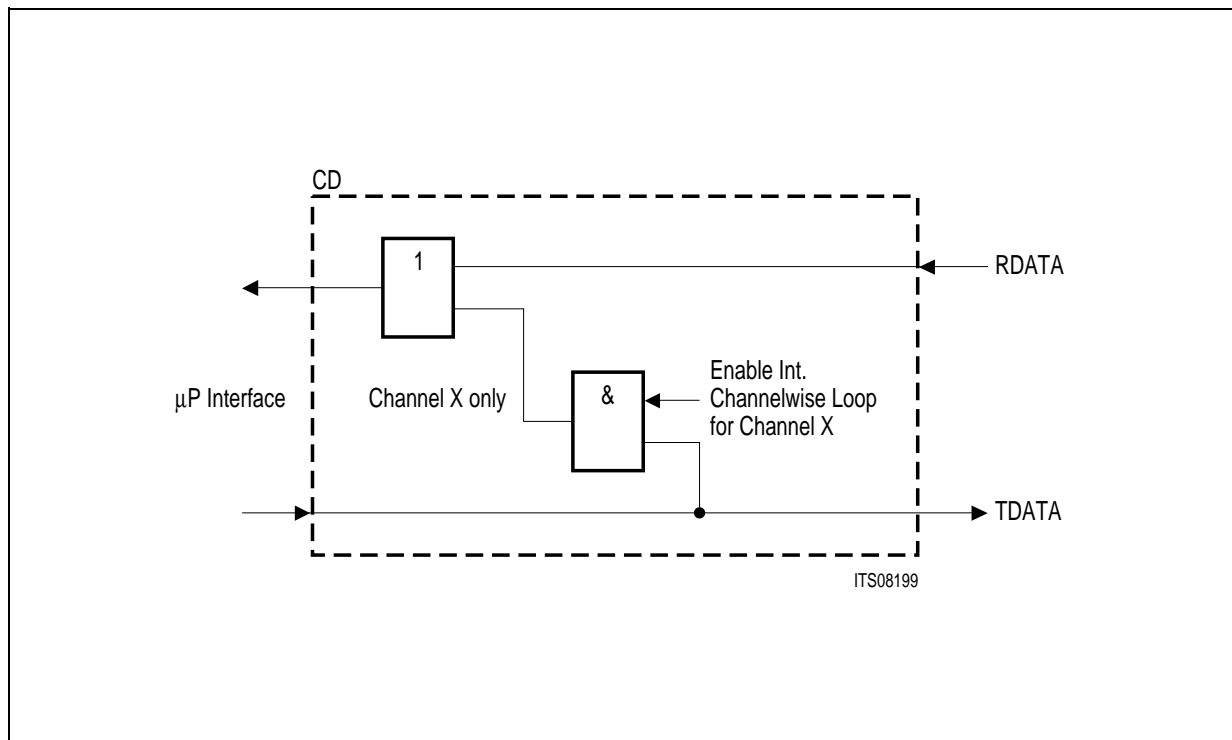
The serial data output is physically routed to the serial data input. The TX data appears on the TDATA output pin and the RDATA input pin is ignored. TCLK and RCLK have to be identical; TSP and RSP have to be identical. The logical Transmit and Receive channels have to be programmed identically.



**Figure 81**

### 5.1.1.2 Internal Channelwise Test Loop

One (and **only** one) logical channel is mirrored from the serial data output to the serial data input. The other logical channels are not affected by this operation. The transmit and receive data rates for this single logical channel must be identical. Normal TCLK, RCLK, TSP and RSP design rules apply. This test loop provides channelwise testing capabilities during idle channel time slots, without interfering with normal data transmission/reception.



**Figure 82**

### 5.1.1.3 External Complete Test Loop

The serial data input is physically routed to the serial data output. Data is received on the RDATA pin and routed to the TDATA pin. The received data can be stored in shared memory for additional diagnostic purposes. TCLK and RCLK have to be identical; TSP and RSP have to be identical.

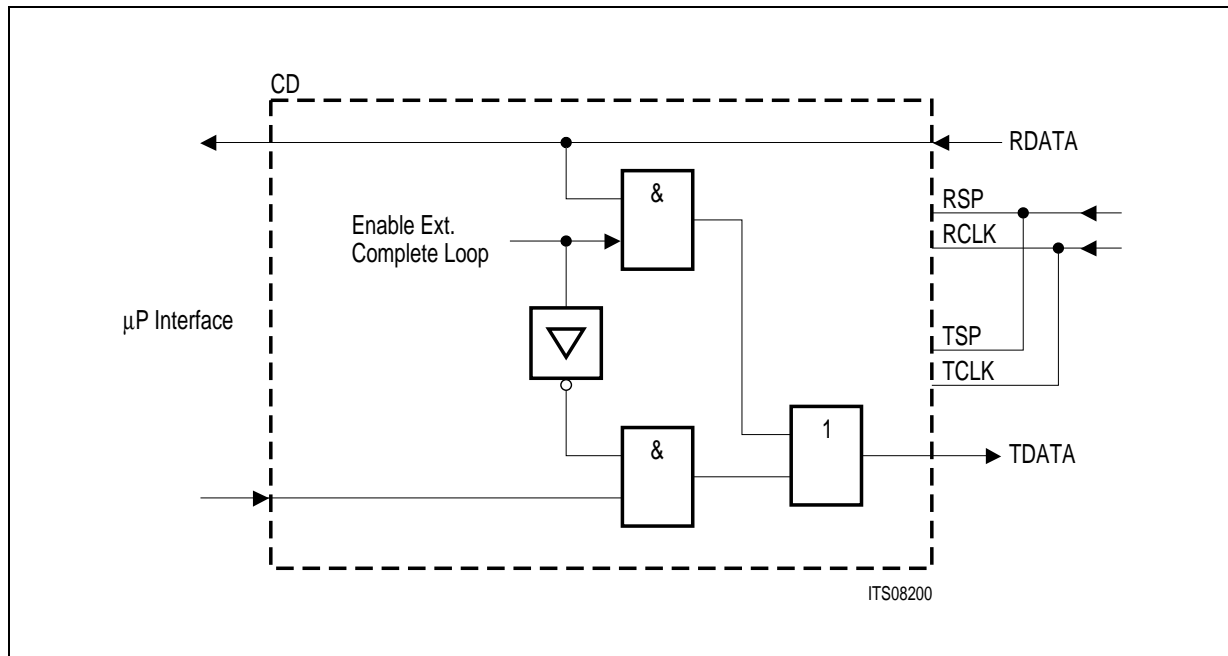


Figure 83

#### 5.1.1.4 External Channelwise Test Loop

One (and **only** one) logical channel is mirrored from the serial data input to the serial data output. The other logical channels are not affected by this operation. The receive and transmit data rates for this single logical channel must be identical. Normal TCLK, RCLK, TSP and RSP design rules apply. This test loop provides channelwise testing capabilities during idle channel time slots, without interfering with normal data reception/transmission.

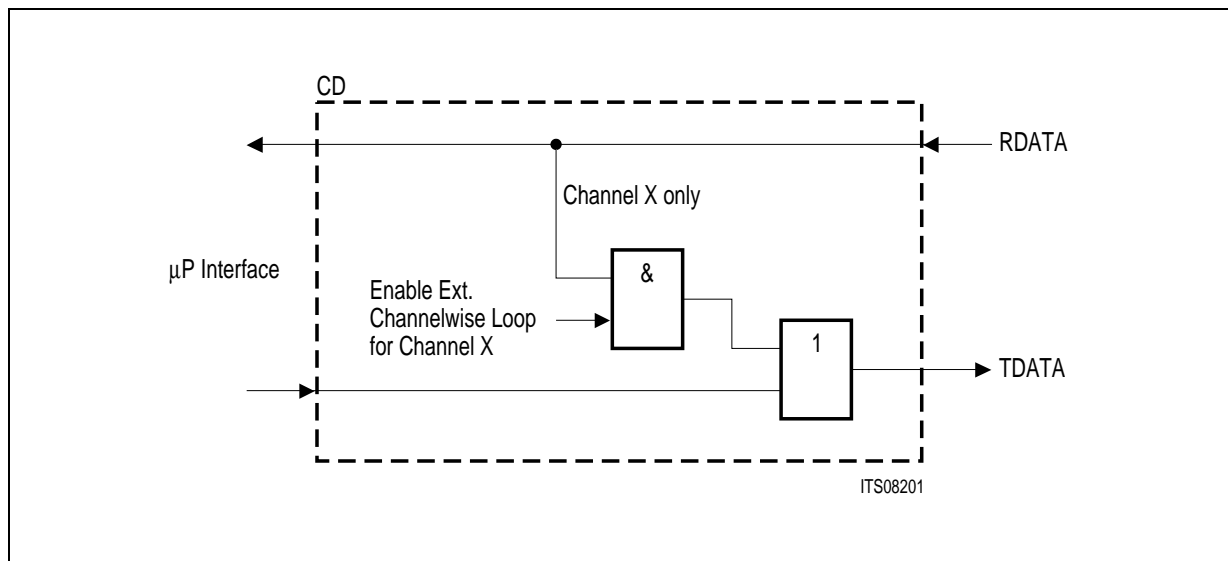


Figure 84

### 5.1.2 Test Loop Activation

All of the test loops are closed (activated) and opened (deactivated) by setting/resetting the appropriate combination of bits in the **Action Specification (Table 10)**. Any unlisted combination of LOC, LOOP and LOOPI is an invalid operation. Although the data sheet (Data Sheet 08.93) specifically states that loops must be left (opened) by issuing the reset pin to '1', there are exceptions to this rule. Generally, the test loops can be opened by software. There are several cases that must be examined and these will be discussed in the next section.

When closing (activating) a test loop, the IN, ICO, IM, RES, and IA bits should equal '0' and PCM and MFL should be set to the appropriate values.

**Table 10**  
**Test Loop Activation**

Test Loop	LOC	LOOP	LOOPI	ASP
Internal complete	0	0	1	xxxxxx08 <sub>H</sub>
Internal channelwise	1	0	1	xxxxxx28 <sub>H</sub>
External complete	0	1	0	xxxxxx10 <sub>H</sub>
External channelwise	1	1	0	xxxxxx30 <sub>H</sub>
No loop	0	0	0	xxxxxx00 <sub>H</sub>

The following recommended procedure for activating a test loop assumes that the MUNICH32 has been fully initialized and the user desires to activate a test loop on channel x:

- Initialize Rc and Tx channel as appropriate for type of test loop.
- Close (activate) the test loop.
- Perform test functions (transmit/receive data, check for interrupts, errors, etc.)
- Open (deactivate) the test loop.
- Perform Rc and Tx off function.

*Note: While the test loop is activated, do not execute the transmit off command. It will not have the effect of resetting the transmit formatter.*

### 5.1.3 Test Loop Deactivation and Switching

As mentioned previously, a test loop can be opened (deactivated) by software. To deactivate a test loop a new ASP should be issued with LOC, LOOP, and LOOPI = 0 and all other bits should be set to the previous values used during activation. Listed below are the possible test loop operations that can be activated with software and those requiring a hardware reset. **Table 11** is provided as a graphical representation of this information.



### 5.1.3.1 Software Operations

Close and open internal complete loop.  
 Close and open internal channelwise loop.  
 Close and open external complete loop.  
 Close and open external channelwise loop.  
 Change from internal complete loop to internal channelwise loop.  
 Change from external complete loop to external channelwise loop.

### 5.1.3.2 Hardware Reset Operations

Change between the internal complete loop and external complete loop.  
 Change between the internal channelwise loop and external channelwise loop.  
 Change between the internal channelwise loop and internal complete loop.  
 Change between the external channelwise loop and external complete loop.  
 Change between internal channelwise loop and external complete loop.  
 Change between internal complete loop and external channelwise loop.  
 Change between external channelwise loop and internal complete loop.  
 Change between external complete loop and internal channelwise loop.

**Table 11**  
**Allowed Operations**

	Change to			
	Internal Complete Loop	Internal Channelwise Loop	External Complete Loop	External Channelwise Loop
<b>Internal Complete Loop</b>	<b>X</b>	SFW	HDW Reset required	HDW Reset required
<b>Internal Channelwise Loop</b>	HDW Reset required	<b>X</b>	HDW Reset required	HDW Reset required
<b>External Complete Loop</b>	HDW Reset required	HDW Reset required	<b>X</b>	SFW
<b>External Channelwise Loop</b>	HDW Reset required	HDW Reset required	HDW Reset required	<b>X</b>

## 5.1.4 Test Loop Examples

### 5.1.4.1 Internal Channelwise Test Loop

Generate HW RESET, and hold off RSP/TSP for 1000 SCLK cycles.

```

ASP:          A104-8004          ;CEPT, MFL=260, IN, IA=1
IQS:          ICQ
              0000-001F
TSA[0]:       00FF-00FF          ;TS0 = CH0
TSA[1...31]:  0000-0000 (x31)
CSP[0]:       00E9-0006          ;Tx/Rc init, poll Tx Desc, HDLC
              FRDA
              FTDA
              0000-0002          ;ITBS = 2 long words
CSP[1...31]   0000-0000 0000-0000 0000-0000 0000-0000 (x31)
CRA[0...31]   0000-0000 (x32)
CTA[0...31]   0000-0000 (x32)

ICQ:          0000-0000 (x512)

FRDA:         0020-0000
              0000-0000
              RcvDtaPtr  → 32 byte
              NxtRDPtr   data block
              +-----+
              |
              +-----+
              | 0020-0000
              | 0000-0000
              | RcvDtaPtr  → 32 byte
              | NxtRDPtr   data block
              +-----+
              |
              +-----+
              | 0020-0000
              | 0000-0000
              | RcvDtaPtr  → 32 byte
              | NxtRDPtr   data block
              +-----+
              |
              +-----+
              | 4020-0000          ;HOLD = 1
              | 0000-0000
              | RcvDtaPtr  → 32 byte
              | 0000-0000   data block
              +-----+

FTDA:         C000-0000          ; HOLD, FE = 1 for dummy frame
              XmtDtaPtr
              NxtTDPtr
              +-----+
              |
              +-----+
              | 0020-0000
              | XmtDtaPtr  → abcdefghi jklmnop
              | NxtTDPtr   qrstuvwxyz012345
              +-----+
              |
              +-----+
              | C020-0000          ;FE, HOLD = 1
              | XmtDtaPtr  → ABCDEFGHIJKLMNOP
              | 0000-0000   QRSTUVWXYZ987654
  
```

## Application Notes

Generate AR Pulse and wait for INT signal (set up TS0 and CH0).

Read interrupt queue:

ICQ:	9000-8000	;Action Request Acknowledge
		;V2.2 (V2.1 = 8800-8000)
	9000-1000	;Polls HOLD bit of 1st Tx Desc.

Set ASP for Internal Channelwise Loop test

ASP:	A104-0028	;CEPT, MFL=260, Int. Chnl loop
------	-----------	--------------------------------

Generate AR Pulse and wait for INT signal.

Read interrupt queue:

ICQ:	9000-8000	;Action Request Acknowledge
	9000-1000	;Polls HOLD bit of 1st Tx Desc.
	9000-082020	;Rc ITF state change

Clear HOLD bit in FTDA (allow frame to Tx over Internal Chnl. Loop).

Read interrupt queue:

ICQ:	9000-1000	;End of Tx frame, polling HOLD bit of Tx desc.
	9000-1020	;Rc frame complete

Read receive descriptors:

FRDA:	0020-0000		
	4020-0000		
	RcvDtaPtr	→	abcdefghijklmnp
	NxtRDPtr		qrstuvwxyz012345
	+		
	→	0020-0000	
	4020-0000		
	RcvDtaPtr	→	ABCDEFGHIJKLMNP
	NxtRDPtr		QRSTUVWXYZ987654
	+		
	→	0020-0000	
	C000-0000		
	RcvDtaPtr	→	;FE, C = 1, BNO = 0
			;empty! (p. 139 User's Manual - FE description)
	+		
	NxtRDPtr		
	+		
	→	4020-0000	
	0000-0000		
	RcvDtaPtr	→	32 byte
	0000-0000		data block

### 5.1.4.2 External Channelwise Test Loop

Generate HW RESET, and hold off RSP/TSP for 1000 SCLK cycles.

```

ASP:                A104-8004                ;CEPT, MFL=260, IN, IA=1
IQS:                ICQ
                   0000-001F
TSA[0]:             00FF-00FF                ;TS0 = CH0
TSA[1...31]:        0000-0000 (x31)
CSP[0]:             00E9-0006                ;Tx/Rc init, poll Tx desc., HDLC
                   FRDA
                   FTDA
                   0000-0002                ;ITBS = 2 long words
CSP[1...31]         0000-0000 0000-0000 0000-0000 0000-0000 (x31)
CRA[0...31]         0000-0000 (x32)
CTA[0...31]         0000-0000 (x32)

ICQ:                0000-0000 (x512)

FRDA:              0020-0000
                   0000-0000
                   RcvDtaPtr    → 32 byte
                   +———|      data block
                   |
                   +——→ 0020-0000
                   0000-0000
                   RcvDtaPtr    → 32 byte
                   +———|      data block
                   |
                   +——→ 0020-0000
                   0000-0000
                   RcvDtaPtr    → 32 byte
                   +———|      data block
                   |
                   +——→ 4020-0000                ;HOLD = 1
                   0000-0000
                   RcvDtaPtr    → 32 byte
                   0000-0000      data block

FTDA:              +——→ C000-0000                ;HOLD, FE = 1 for dummy frame
                   |
                   XmtDtaPtr
                   +———|      NxtTDPtr

```

## Application Notes

Generate AR Pulse and wait for INT signal (set up TS0 and CH0).

Read interrupt queue:

```
ICQ:          9000-8000      ;Action Request Acknowledge
                                ;V2.2 (V2.1 = 8800-8000)
                                9000-1000    ;Polls HOLD bit of 1st Tx Desc.
                                9000-1000    ;FI Frame indication for the
                                                1st Tx Desc.
                                ;Now M32 starts polling HOLD
                                bit of 1st Desc.
```

Set ASP for External Channelwise test loop

```
ASP:          A104-0030    ;CEPT, MFL=260, Ext. Chnl loop
```

Generate AR Pulse and wait for INT signal.

Read interrupt queue:

```
ICQ:          9000-8000      ;Action Request Acknowledge
                                9000-0820    ;Only if other station uses
                                                idle code 7E
                                9000-1020    ;Received frame complete
```

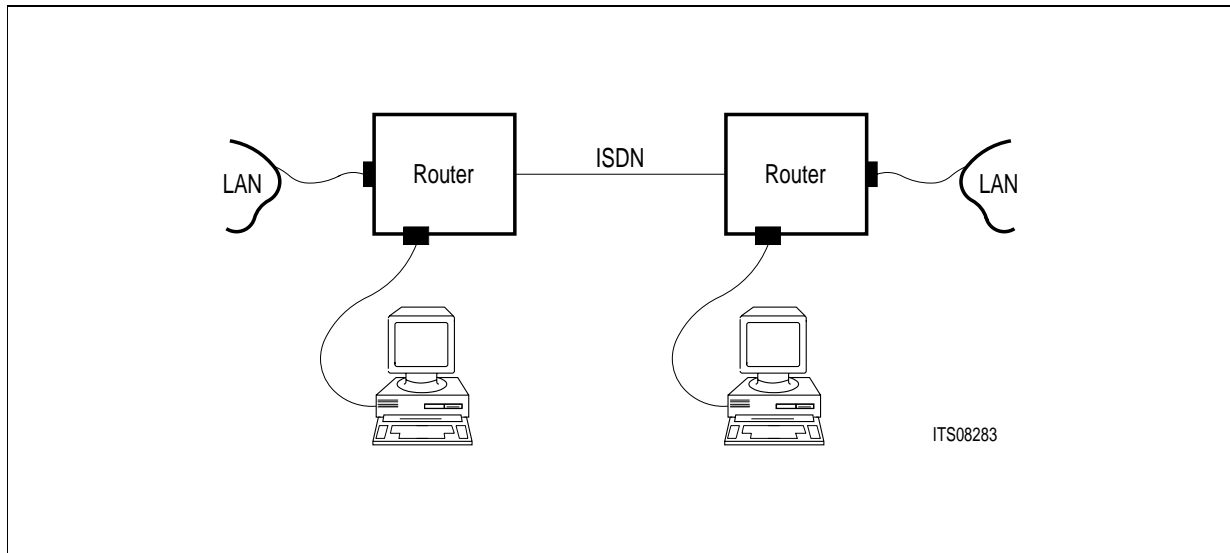
Read receive descriptors: ;assumes 64 byte frame externally looped

```
FRDA:          0020-0000      ;with proper HDLC framing
                                4020-0000      ;NO = BNO
                                RcvDtaPtr    —→  abcdefghijklmnop
                                +——— NxtRDPtr    qrstuvwxyz012345
                                |
                                +——→ 0020-0000
                                4020-0000      ;NO=BNO
                                RcvDtaPtr    —→  ABCDEFGHIJKLMNOP
                                +——— NxtRDPtr    QRSTUVWXYZ987654
                                |
                                +——→ 0020-0000
                                C000-0000      ;FE, C = 1, BNO = 0
                                RcvDtaPtr    —→  ;empty!
                                +——— NxtRDPtr
                                |
                                +——→ 4020-0000
                                0000-0000
                                RcvDtaPtr    —→  32 byte
                                0000-0000      data block
```

## 5.2 MUNICH32 in a LAN/WAN Router

### 5.2.1 Introduction

Subject of this application note is an ISDN/LAN Router, a communication system that enables two LANs to communicate via the ISDN.



**Figure 85**  
**ISDN/LAN Router**

The structure of the whole system is shown in **Figure 85**. The router itself is realized as a stand alone solution. It is connected to a standard PC for software download and maintenance control only. After the download the system works fully independent of the host PC.

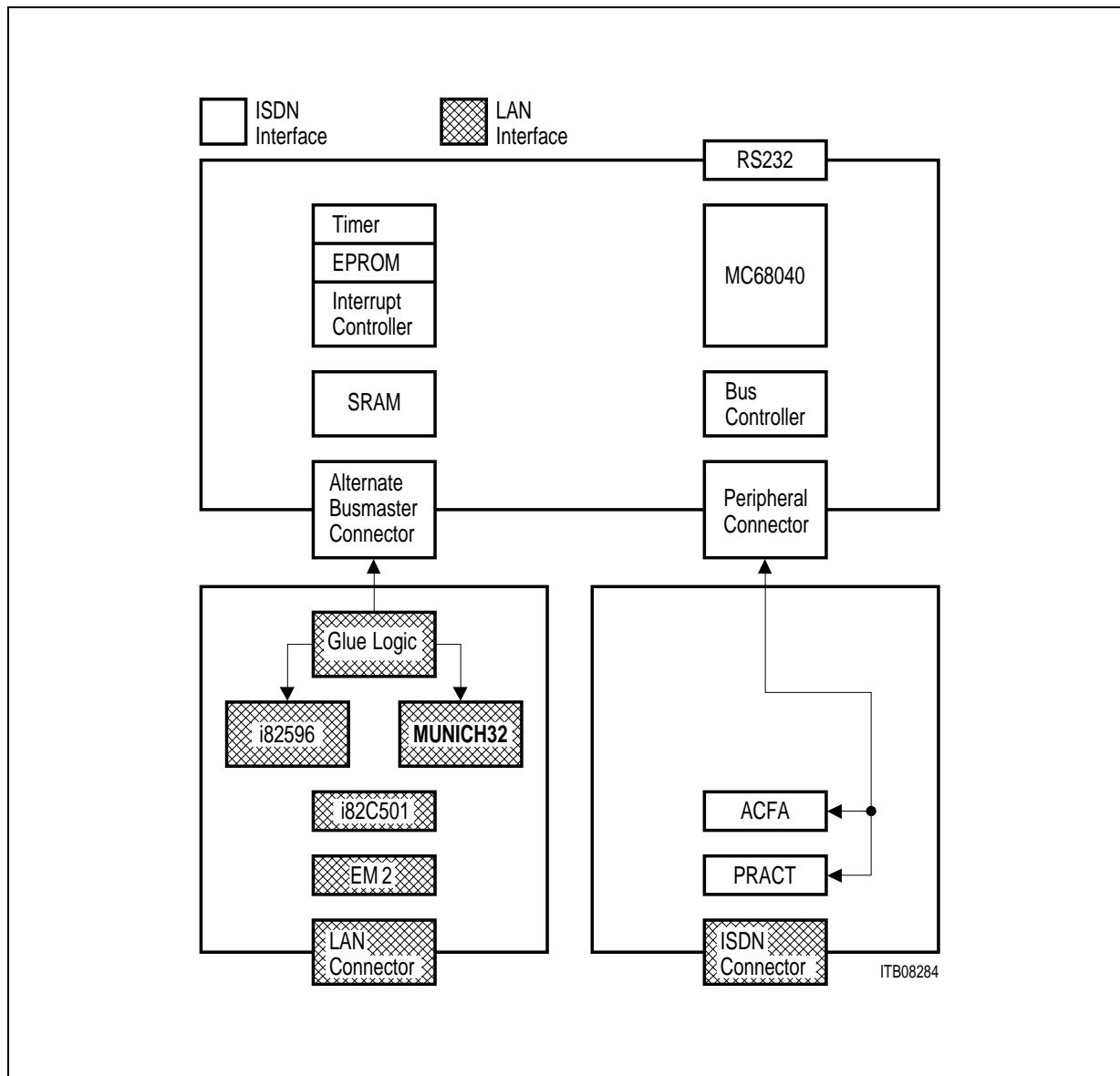
The hardware of the ISDN/LAN router consists of an application specific part and a processor system. The application specific hardware is mainly based on the SIEMENS Component MUNICH32 (Multi Channel Network Interface Controller for HDLC) and a standard LAN controller. Both devices are integrated in the same processor system.

The software of the ISDN/LAN router is formed by integrating the MUNICH32 Device Driver Module (DDM) and the corresponding LAN controller Device Driver Module in a Device Driver System (DDS). The device driver modules build a platform to implement the routing strategy in a separate application module.

The application specific hardware, the MUNICH32 Device Driver Module and the application module are the main aspects described in the following chapters. The structure of the processor system is briefly illustrated. The DDS service routines are explained as far as necessary to understand this special application. It is suggested that the reader has some knowledge about the MUNICH32 before reading this application note. Detailed information about the MUNICH32, its features and memory structures are given in the MUNICH32 PEB20320 Data Sheet.

### 5.2.2 Hardware

The processor system is based on a Motorola 68040 processor. It contains 512 KByte SRAM, a bus controller and peripherals like timer, EPROM and interrupt controller. The application specific hardware is integrated by using a Peripheral Connector and an Alternate Busmaster Connector. The Peripheral Connector allows the integration of external peripherals. The Alternate Busmaster Connector is used to connect external bus masters to the local bus. The system is provided with a RS232 serial interface to download executable software on the board.

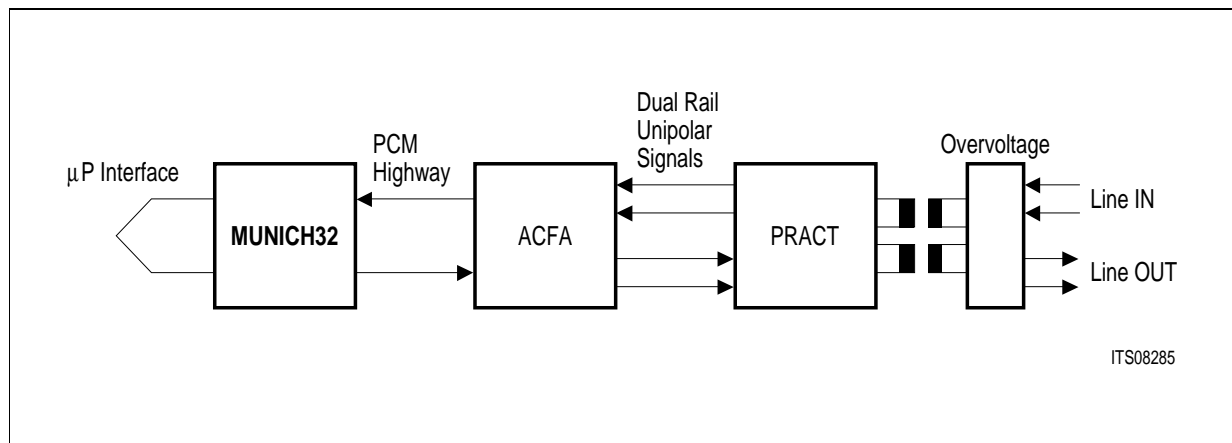


**Figure 86**  
**Hardware Block Diagram**

## Application Specific Hardware

The application specific hardware consists of an ISDN primary rate interface and an Ethernet interface. The MUNICH32 PEB 20320 in conjunction with the layer 1 SIEMENS components ACFA (Advanced CMOS Frame Aligner) PEB 2035 and PRACT (Primary Rate Access Clock Generator and Transceiver) PEB 22320 are used to build the primary rate interface. Incoming data from the ISDN is first processed from the PRACT. It translates the HDB3 coded line signals in dual rail signals. The PRACT also supplies ACFA and MUNICH32 with clock signals. Main task of the ACFA is the frame alignment. Besides, the ACFA translates the dual rail data in a single rail, unipolar bit stream which can be processed by the MUNICH32.

The MUNICH32 handles up to 32 channels of a full duplex PCM highway. All time-slots may have data rates between 8 Kbit/s and 64 Kbit/s. The MUNICH32 supports besides other protocols the HDLC formatting/deformatting. If programmed for HDLC mode, the MUNICH32 performs HDLC specific functions like framing, CRC check/generation, flag stuffing and zero bit insertion/deletion autonomously. An on-chip 64-channel DMA controller allows the device to store/read data into/from the SRAM. The DMA controller manages long word or word transfers via a 32-bit processor interface. The  $\mu$ P interface can be configured to be Motorola 68020 or intel 80386 compatible.



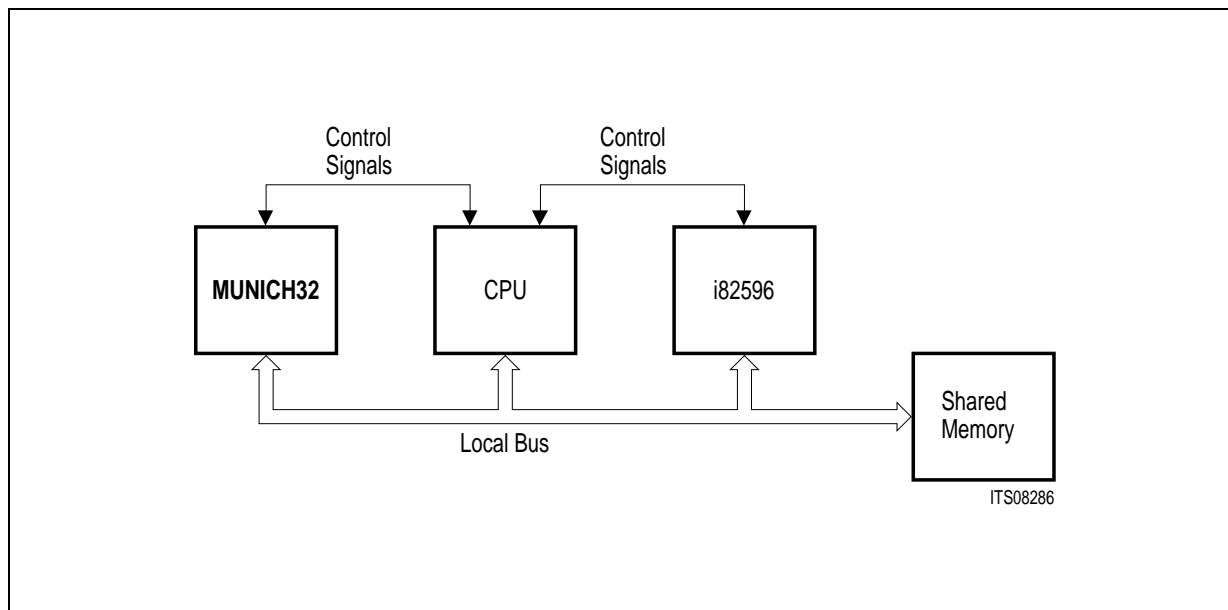
**Figure 87**  
**ISDN Interface**

The Ethernet interface is built with a LAN controller, a Manchester encoder/decoder and a transceiver. The LAN controller supports all IEEE 802.3 standards. The Ethernet framing: preamble generation, source address generation, destination address checking, short-frame detection, automatic length field handling is performed. After LAN controller processing the transmit data is Manchester encoded and forwarded to the transmission line, while receive data is Manchester decoded before being processed by the LAN controller.



## System Architecture

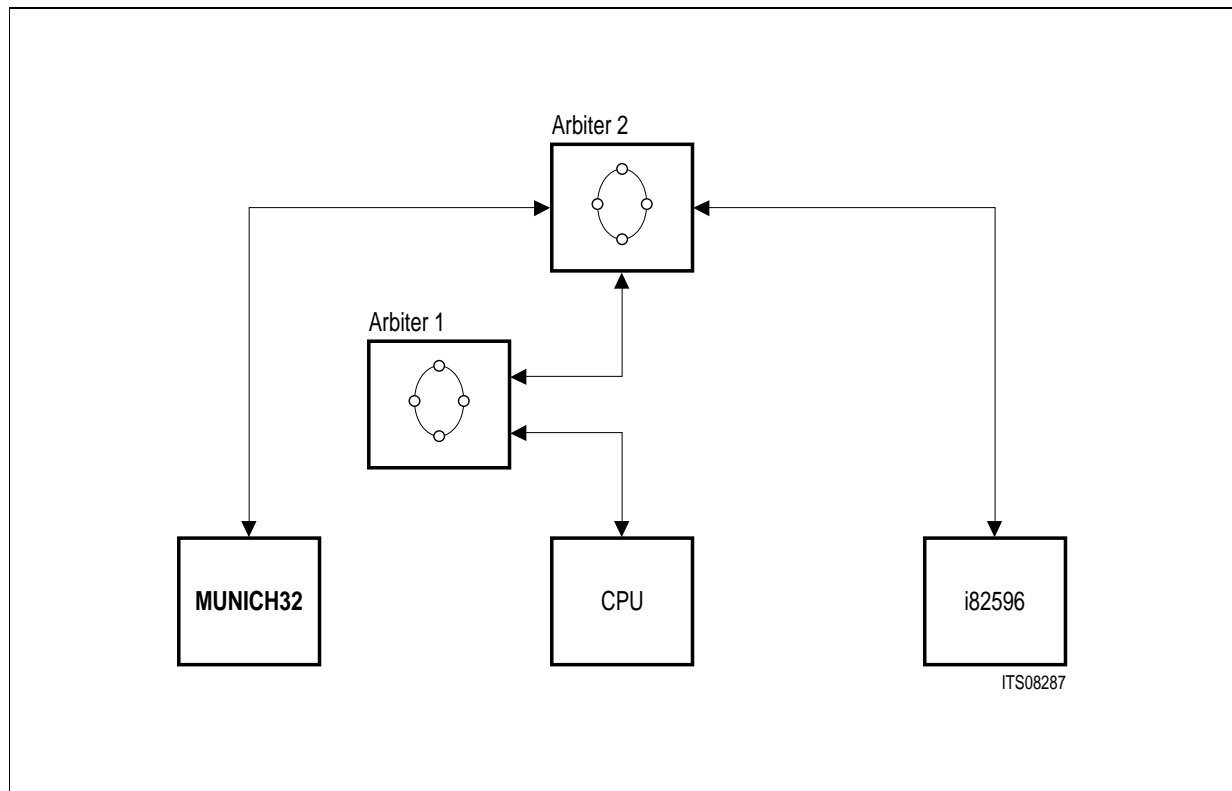
The system architecture is shown in **Figure 88**. The MUNICH32, the CPU and the LAN controller store data in the shared memory. The communication between CPU and alternate bus master is done via the shared memory. The CPU informs the alternate bus masters with help of control signals about changes in the shared memory and vice versa. The MUNICH32 input control signal is the Action Request pulse (ACTION REQUEST). It is generated by one CPU write cycle to a defined address and decoding the address lines. The MUNICH32 then responds by generating an interrupt pulse and writing the respective interrupt information in the SRAM.



**Figure 88**  
**System Architecture**

## Bus Arbitration

Since three devices are using the bus it is necessary to implement a bus arbitration. Each bus master requests bus mastership and awaits bus control given to it by the arbiter. The bus arbitration protocol is also Motorola specific. The intel specific signals of the LAN controller (i82596) are translated into Motorola specific signals. The bus arbitration is realized in two devices GAL16V8 (15 ns), both containing a Finite State Machine. Arbiter 1 gives bus mastership to the CPU whenever no other bus master requests bus mastership. If either the MUNICH32 or the LAN controller requests bus mastership the arbiter 2 gives a bus request to the arbiter 1. Arbiter 1 forces the CPU to release the bus and gives bus mastership to arbiter 2. Arbiter 2 then responds to MUNICH32 or LAN controller. In this solution the priority of the MUNICH32 is higher than that of the LAN controller. Consequently if both alternate bus masters request bus mastership at the same time, bus mastership will be given to the MUNICH32. The LAN controller has to wait until MUNICH32 has finished his accesses and arbiter 1 returns the bus to the CPU. It might happen, that some Ethernet frames get lost, because the LAN controller can not get access to the bus in time, but the loss of incoming data from the ISDN (where fees have to be paid) is minimized.



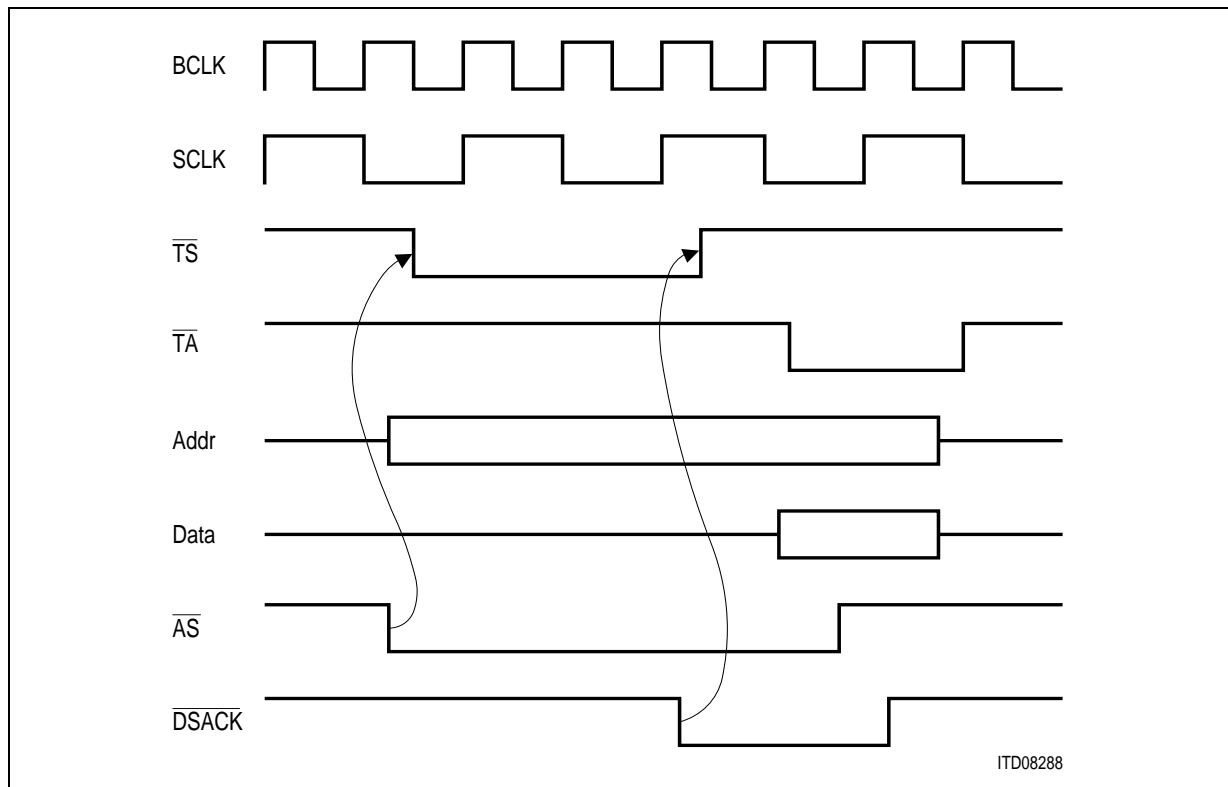
**Figure 89**  
**Bus Arbitration**

## Bus Timing Adaptation<sup>1)</sup>

The bus controller manages memory accesses of all bus masters (CPU, MUNICH32 or LAN controller). The bus controller timing is Motorola 68040 specific. The MUNICH32 bus interface is either Intel specific or Motorola 68020/030 specific. Therefore the MUNICH32 bus timing needs to be adapted by using simple glue logic. One Gate Array Logic (Gal16V8, 15 ns) contains all necessary logic.

The MUNICH32 Address Strobe ( $\overline{AS}$ ) signal determines valid addresses on the bus. The equivalent Motorola 68040 control signal is the Transfer Start ( $\overline{TS}$ ). During MUNICH32 write cycles valid data on the bus is indicated with the Data Strobe ( $\overline{DS}$ ) signal. MUNICH32 write and read bus cycles are terminated with the Data Transfer Acknowledge ( $\overline{DSACK}$ ) signal. For the Motorola 68040 the end of a bus cycle is indicated by the Transfer Acknowledge ( $\overline{TA}$ ) signal.

During MUNICH32 bus cycles the MUNICH32 output signal  $\overline{AS}$  is used to generate the bus controller input signal  $\overline{TS}$ . The  $\overline{TS}$  is deasserted with the MUNICH32 input  $\overline{DSACK}$  rising edge. Since all bus cycles have the same length the  $\overline{DSACK}$  signal is generated two bus clock cycles after  $\overline{AS}$  is detected low.  $\overline{TS}$  is tristated, if the MUNICH32 is not busmaster. This signal is driven by another bus master during that time.



**Figure 90**  
**MUNICH32 Timing Adaption**

<sup>1)</sup> See also **Chapter 5.2.6**.

## Application Notes

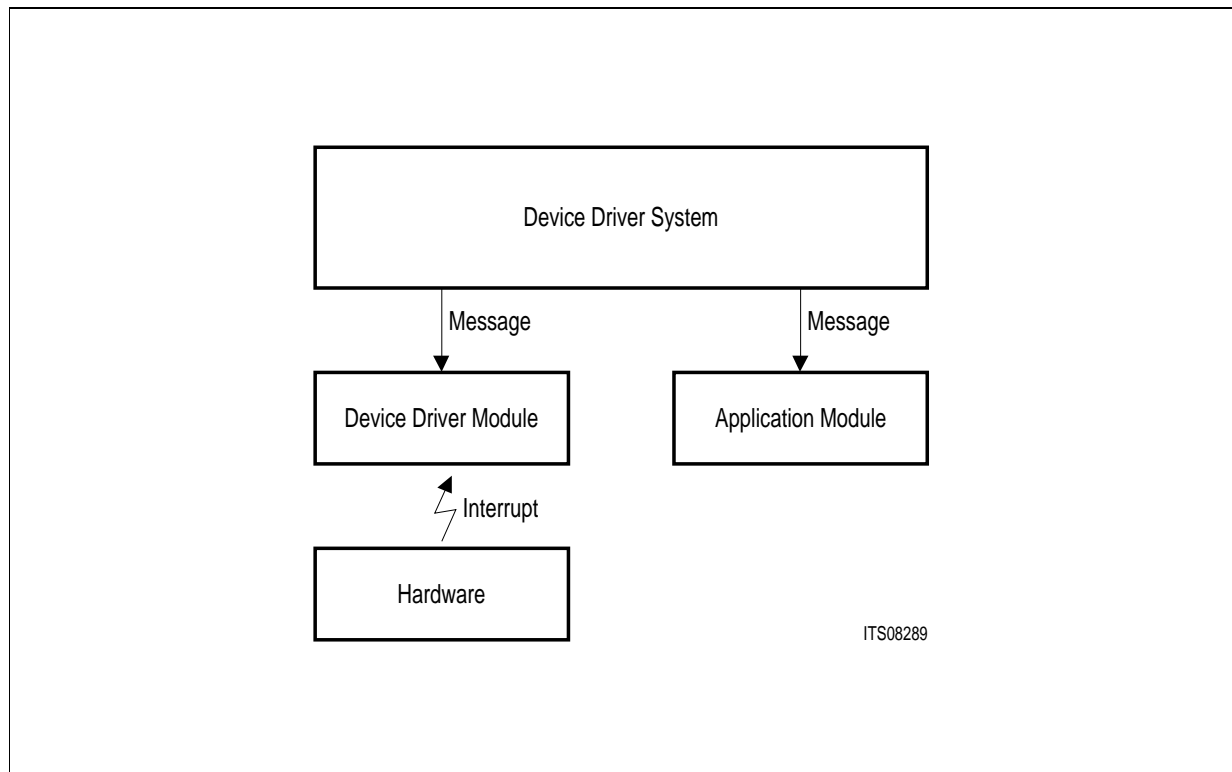
The LAN controller's (i82596) bus timing also needs to be adapted. The address lines A1, AO, Size 0 and Size 1 need to be generated, because the LAN controller performs 8 bit and 16 bit cycles as well as 32 bit cycles. There are also some non standard bus signals for the LAN controller, that have to be generated. Furthermore the System Clock and the Bus Clock have to be synchronized. All necessary glue logic for the LAN controller is realized in four devices Gal 16V8.

### 5.2.3 Software

The software is based on a message oriented device driver system. The device driver modules and application modules have a structure that allows to access them via defined entry points.

#### Module Entry Points

Two Entry points offer access to the DDMs. Messages can be sent to the DDM via the Message Entry Point. A hardware interrupt causes the program to branch to the Interrupt Entry Point. The APM also offers access via a Message Entry Point, but since the APM does not control any hardware, there does not exist any Interrupt Entry Point.



**Figure 91**  
**Module Entry Points**

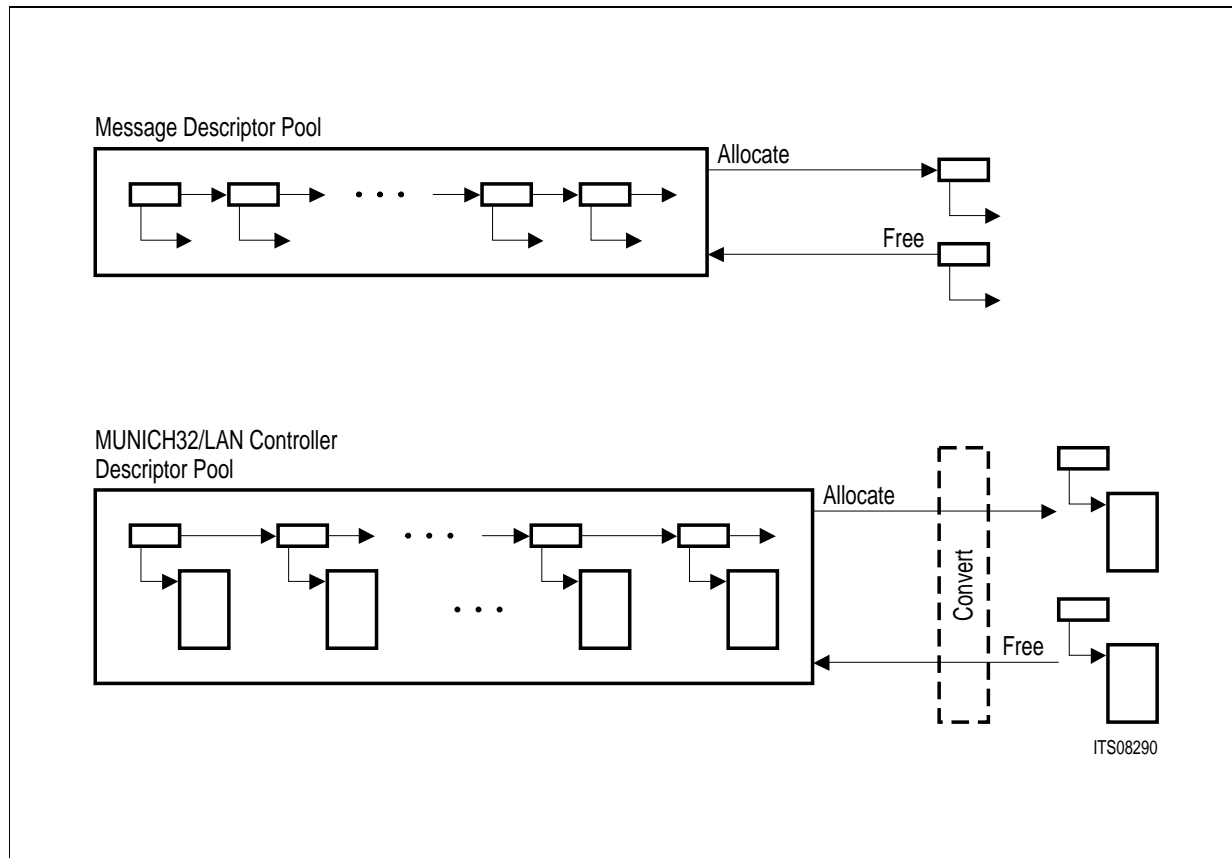
## DDS Tasks

The message transfer between the modules is the main task of the DDS, realized by some service routines. DDMs and APMs are integrated in the DDS by executing a Module Init Routine. The Module Init Routine is called by the DDS. Additionally the DDS offers service routines for memory management. All service routines can be used by all modules. Some memory management functions will be presented in more detail. For detailed information about the other DDS service routines please refer to the SIPB 7520 Primary Rate User Board or EASY532 Datacom Userboard Documentation.

## Memory Management

With the memory management functions the allocation of message descriptors, MUNICH32 receive/transmit descriptors<sup>1)</sup> or LAN controller receive/transmit descriptors is simplified. During initialization of the memory management module DDSM a pool of descriptors is prepared in a linked list. The memory management functions allow to allocate descriptors and to free descriptors. During initialization of the memory management module DDSM a pool of descriptors is prepared in a linked list. The memory management functions allow to allocate descriptors and to free descriptors. During allocation a descriptor is taken from the prepared list. After utilization the descriptor is given back to the descriptor pool. There is one pool for message descriptors and one pool for MUNICH32 receive/transmit **and** LAN controller receive/transmit descriptors. Because MUNICH32 transmit and receive descriptors differ and they both differ from the LAN controller transmit and receive descriptors, there are service functions available to convert the descriptor type.

<sup>1)</sup> Refer to MUNICH32 Data Sheet.



**Figure 92**  
**Memory Management**

### 5.2.3.1 Device Driver Module MUNICH32

#### Tasks

The MUNICH32 Device Driver Module has to prepare all memory structures for the MUNICH32. The ACTION REQUEST Pulse has to be generated. The device driver module also has to treat the MUNICH32 interrupts.

#### Message Entry Point

Every incoming message results in executing a function.

Function	Action
ResetMunich32	Action Specification Reset bit is set, All channels are initialized, all time-slots are initialized, ACTION REQUEST Pulse is generated.
ConfigMunich32	Sets PCM mode and maximum frame length, ACTION REQUEST Pulse is generated.
InitInterruptQueue	Interrupt Attention bit is set, A new interrupt queue is defined, ACTION REQUEST Pulse is generated.
InitChannel	Action Specification in-bit is set, Initializes receiver and transmitter of selected channel, ACTION REQUEST Pulse is generated.
InitTxChannel	Initializes transmitter of selected channel, ACTION REQUEST Pulse is generated.
InitRcChannel	Initializes receiver of selected channel, ACTION REQUEST Pulse is generated.
SendFrame	Adds tx descriptors to the transmit descriptor queue and clears hold bit of poll descriptor if the channel is active.
TxJump	If no poll descriptor is detected Initialize Channel Only bit is set, 'transmit jump' command is given, if the previous command was not 'receive abort' or 'off the receive clear' command is given, ACTION REQUEST Pulse is generated.
TxHold	Initialize Channel Only bit is set, turns channel on or off, turn channel on: if last command was 'transmit off' or 'transmit abort' 'transmit clear' is given and 'transmit hold' bit is cleared, turn channel off: if channel is active and 'transmit hold' bit is set, ACTION REQUEST Pulse is generated.

## Application Notes

Function	Action
TxShutDown	Initialize Channel Only bit is set, gives 'transmit off' command or 'transmit abort' command, ACTION REQUEST Pulse is generated.
RcJump	Initialize Channel Only bit is set, If last command was 'transmit off' or 'transmit abort' 'transmit clear' is given, 'receive jump' command is given, ACTION REQUEST Pulse is generated.
RcShutDown	'Initialize Channel Only' bit is set, Gives receive off command if receiver was aborted otherwise gives receive abort command, ACTION REQUEST Pulse is generated.
SwitchInternalChanLoop	Sets/clears Internal Channelwise Loop, ACTION REQUEST Pulse is generated.
SwitchInternalCompLoop	Sets/clears Complete Loop, ACTION REQUEST Pulse is generated.
ShowMunich32VersionNr	ACTION REQUEST Pulse is generated.
CheckActionRequestQueue	Looks for messages to be processed and branches to the Message Entry Point.

### Interrupt Entry Point

The information in the interrupt queue is read and a message containing that information is sent to the user.

In case of a received frame the written receive descriptors are linked to a message and sent to the user. The next available descriptor in the list is linked to the memory structures. An equivalent number of new receive descriptors is allocated and linked to the end of the receive descriptor queue.

In case of a transmit acknowledge interrupt the used transmit descriptors are released to the descriptor pool.



## Programming the MUNICH32 for this Application

The basic programming of the MUNICH32 for this application is realized in the Module Initialization Routine. Further programming is done by calling the function 'Init Channel' for each channel once. Transmit data is then added to the memory structures by passing a message with linked transmit descriptor(s) to the function 'Send Frame'.

### Module Initialization Routine

Here the IM-bit is cleared because the MUNICH32 DDM expects the action request acknowledge interrupt. The values for PCM and MFL are set. The PCM format is a 32-channel format according to CEPT. The maximum frame length is set to its maximum. Finally the address and length of a new interrupt queue are defined. Those values will not be changed anymore.

### Init Channel Routine

The function 'Init Channel' initializes the time-slot assignment and the channel specification for one channel. The channel number is set to the value of the variable 'channel'. The MUNICH32 is alerted to access all time-slot assignments and the channel specification by setting the in-bit.

The fillmask (transmit and receive) for the selected channel is written in the appropriate word of the time-slot assignment. All other channels and their fillmasks are not affected.

For this application all interrupts are enabled. Initialization of the selected channel comprises the definition of a new ITBS value and initialization of the receiver and the transmitter. The transmit hold bit is cleared. After initialization the MUNICH32 starts polling the hold bit of the current transmit descriptor. Therefore a transmit descriptor is allocated and connected to the memory structures. Its hold bit and fe-bit are set to one, its no-bits are set to zero. For that reason the MUNICH32 does not transmit anything but polls this descriptor. Since after the receiver's initialization the MUNICH32 is ready to receive data, a queue of receive descriptors is allocated and linked to the memory structures. The hold bit of the last descriptor in the list is set to indicate the end of the list. In all other descriptors the hold bit is cleared.

### Send Frame Routine

Calling 'SendFrame' after initialization of a channel results in executing 'AddHdlcFrame'. In that routine the transmit descriptors are disconnected from the message and linked to the memory structures. If the message source is the 'MROUTE Application Module' the hold bit and fe-bit indicating the end of a frame and the end of the list have already been set/cleared in the MROUTE module, they are not modified anymore. If the message source is any other module the fe-bit and hold bit are cleared in all descriptors except for the last one. There the hold bit has to be set, to prevent the MUNICH32 from branching to the next descriptor. Setting the fe-bit in the last descriptor only forces the MUNICH32 to send the data in one HDLC frame. The bits HI, V110 and CSM are cleared in both cases.

### Transmit/Receive Interrupt

A transmit acknowledge interrupt is treated by returning the transmit descriptor(s) to the descriptor pool.

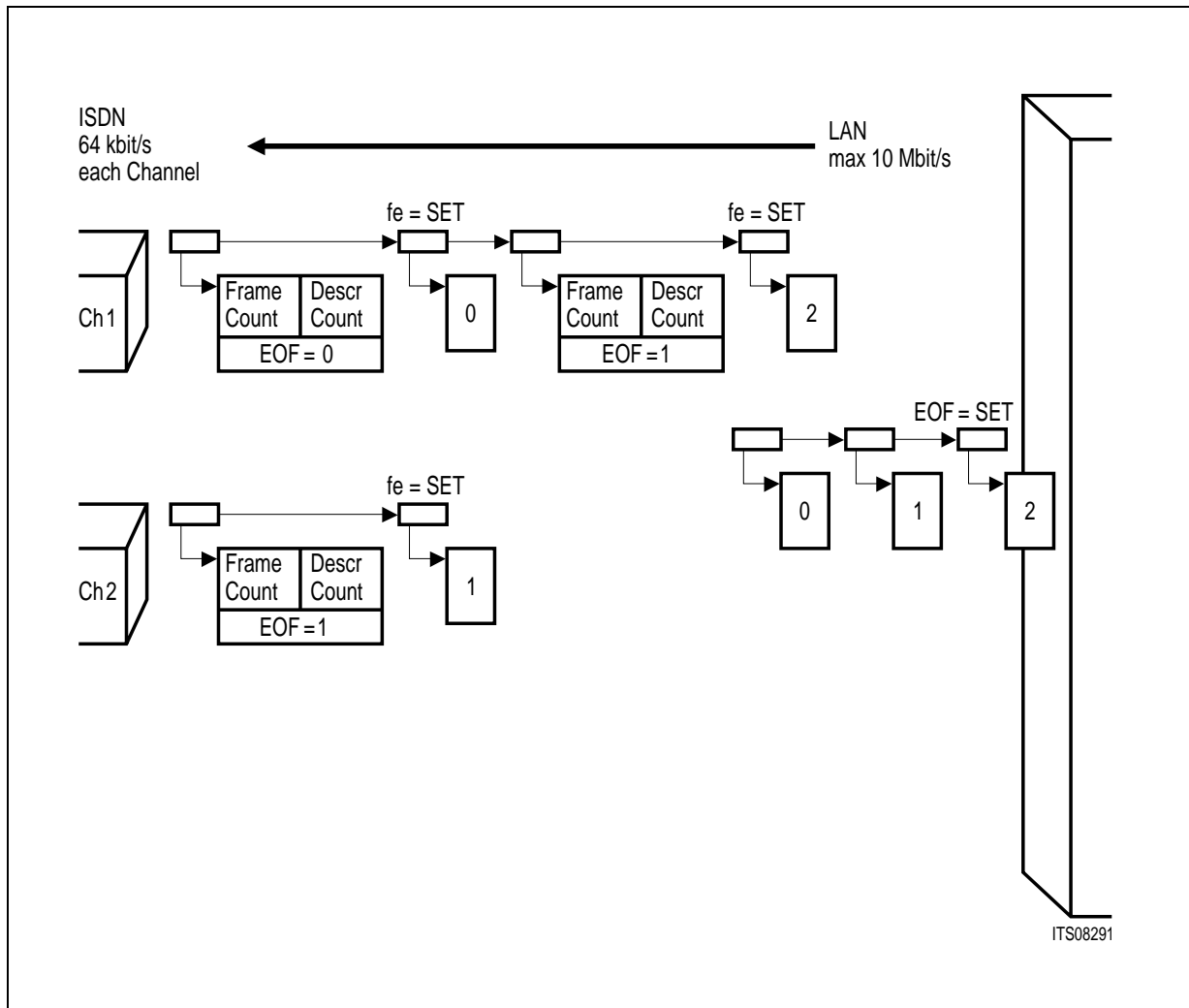
After a receive interrupt (FI bit set) the receive descriptors with c-bit set, are disconnected from the list of receive descriptors, linked to a message and sent to the MROUTE module. The next free receive descriptor in the list is linked to the memory structures. An equivalent number of new descriptors is allocated and linked to the end of the receive descriptor list.

#### 5.2.3.2 Application Module MROUTE

The application module MROUTE implements the routing strategy.

### Routing Strategy

Both devices the MUNICH32 and the LAN controller organize receive and transmit data in a linked list of receive descriptors and a linked list of transmit descriptors. The data is stored in data buffers of variable size. The receive/transmit descriptors contain the address of the data buffer. The basic idea behind the routing strategy is, to take the MUNICH32's receive descriptor and link it to the LAN controller's transmit descriptor queue. On the other hand to take the LAN controller's receive descriptor and link it to the MUNICH32's transmit descriptor queue.



**Figure 93**  
**Insertion of additional Information**

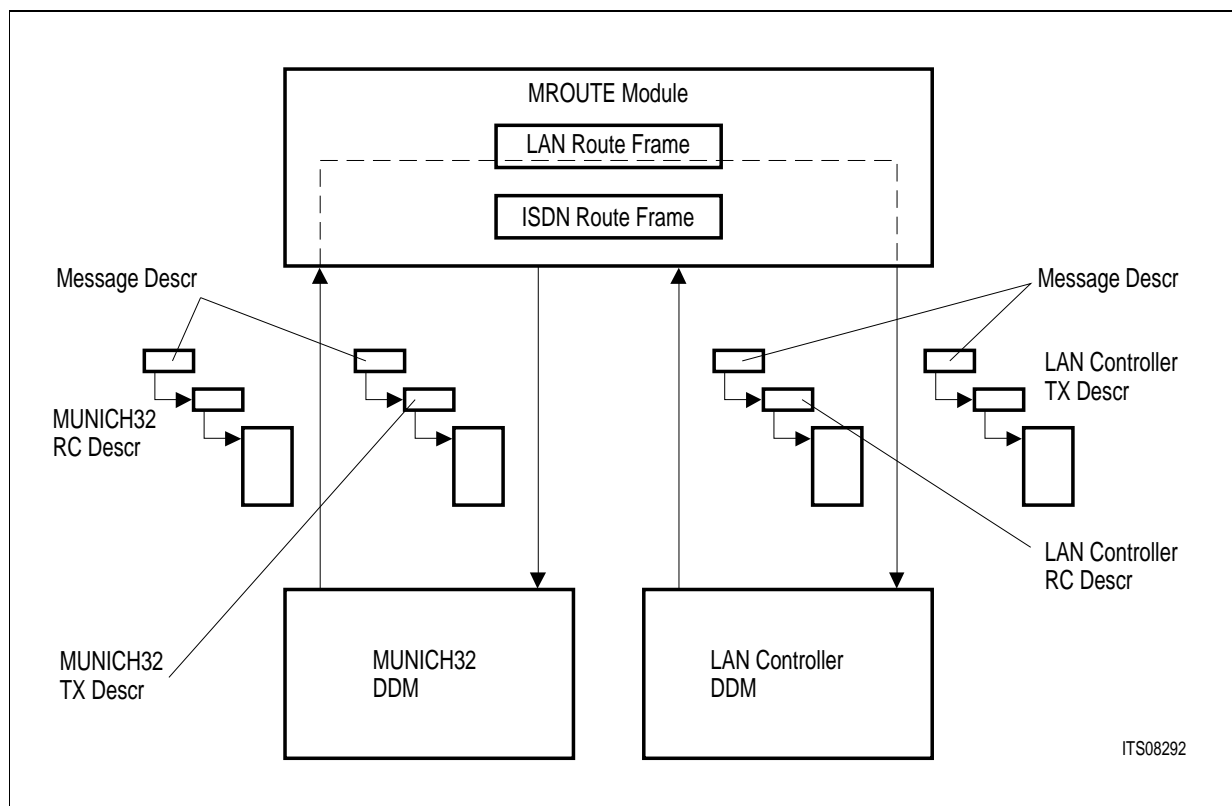
To make efficient use of the available bandwidth, the parallel use of several B-channels is one of the routing strategy's goals. Every Ethernet frame is divided into several parts because the LAN controller stores the received data in several receive descriptors, if necessary. The frame is then sent via the ISDN by using a separate B-channel for every LAN receive descriptor. To ensure that the parts of the Ethernet frame will be reassembled in correct order, every part of the Ethernet frame is supplied with additional information. That additional information has to be extracted before reassembling the frame. In **Figure 93** an example of one Ethernet frame consisting of three descriptors, spread over two B-channels, is shown. The additional information contains the frame number, the descriptor number and the information, whether the frame is completed. To simplify the extraction of the additional information every frame part and its additional information are sent in **one** HDLC frame.

## Application Notes

The fe-bit marks the end of one HDLC frame, the EOF bit marks the end of the Ethernet frame. The additional information comprises the 8-bit word descriptor count, 16-bit word frame count and EOF a 8-bit variable which indicates the last descriptor of the frame.

### Message Entry Point

The message entry point calls two functions: IsdnRouteFrame and LanRouteFrame. An Ethernet frame is processed by IsdnRouteFrame, an ISDN frame by LanRouteFrame. The MUNICH32 receive descriptors are converted to LAN controller transmit descriptors and those of the LAN controller are converted to MUNICH32 transmit descriptors.



**Figure 94**  
**Message Flow between DDMs and MROUTE Module**

Besides the IsdnRouteFrame realizes the insertion of additional information and splits an Ethernet frame on several B-channels. The additional information is stored in an extra allocated transmit descriptor which is placed before the descriptor containing the data. Every descriptor and the respective extra descriptor are connected to one message descriptor. This message with set hold bit and set fe-bit in the descriptor containing the data is further processed from the MUNICH32 DDM routine 'Send Frame'.

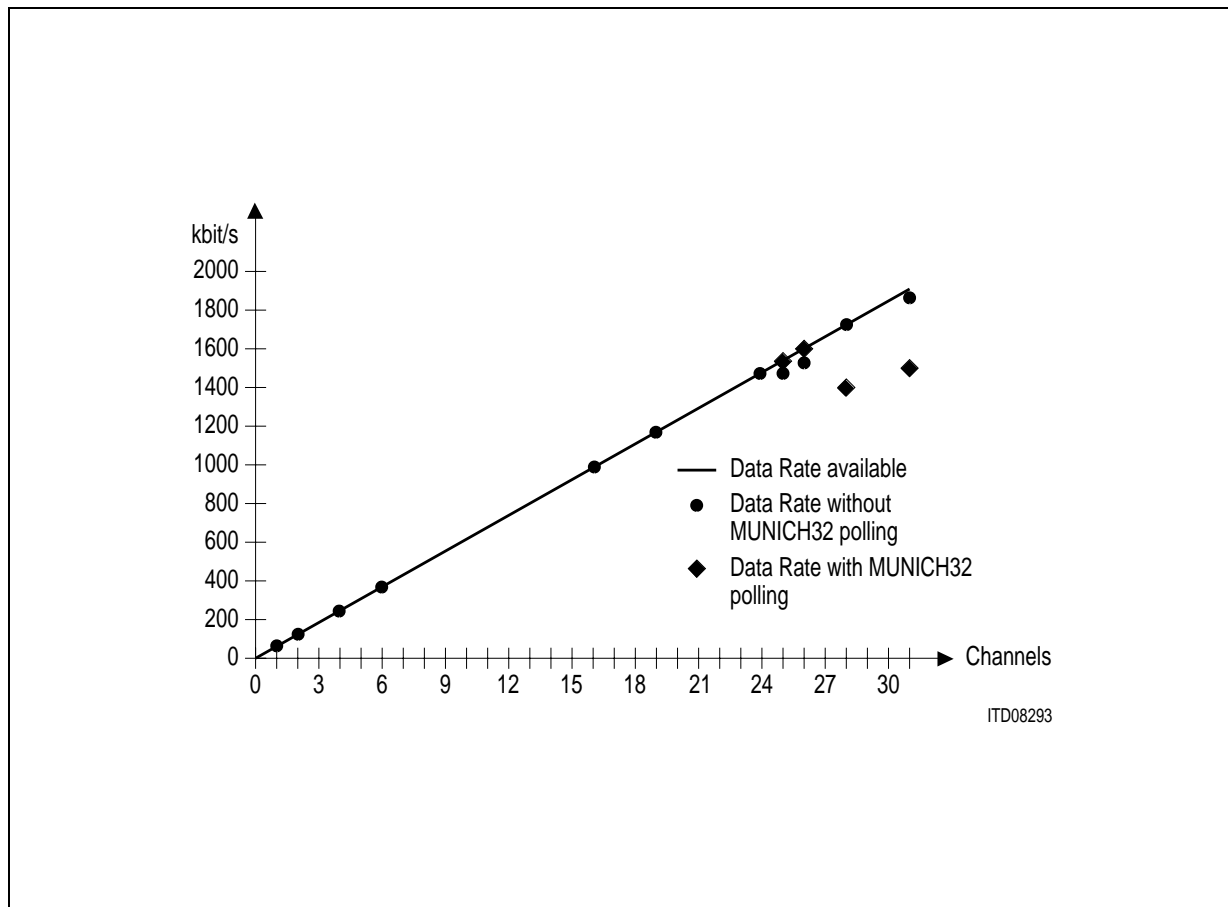
LanRouteFrame reassembles the Ethernet frames. It takes into account, that the parts might arrive with different delays. Every complete frame is connected to a message descriptor and then processed from the LAN controller DDM.

### 5.2.4 Performance Considerations

Some considerations about the performance are made by investigating the maximum data rate. Further investigations are made about the bus occupancy by all busmasters and the MUNICH32 poll access' influence on the data rate. Finally the processing of one frame is illustrated.

#### Data Rates

The data rate during transmission from the ISDN into the Ethernet was tested.



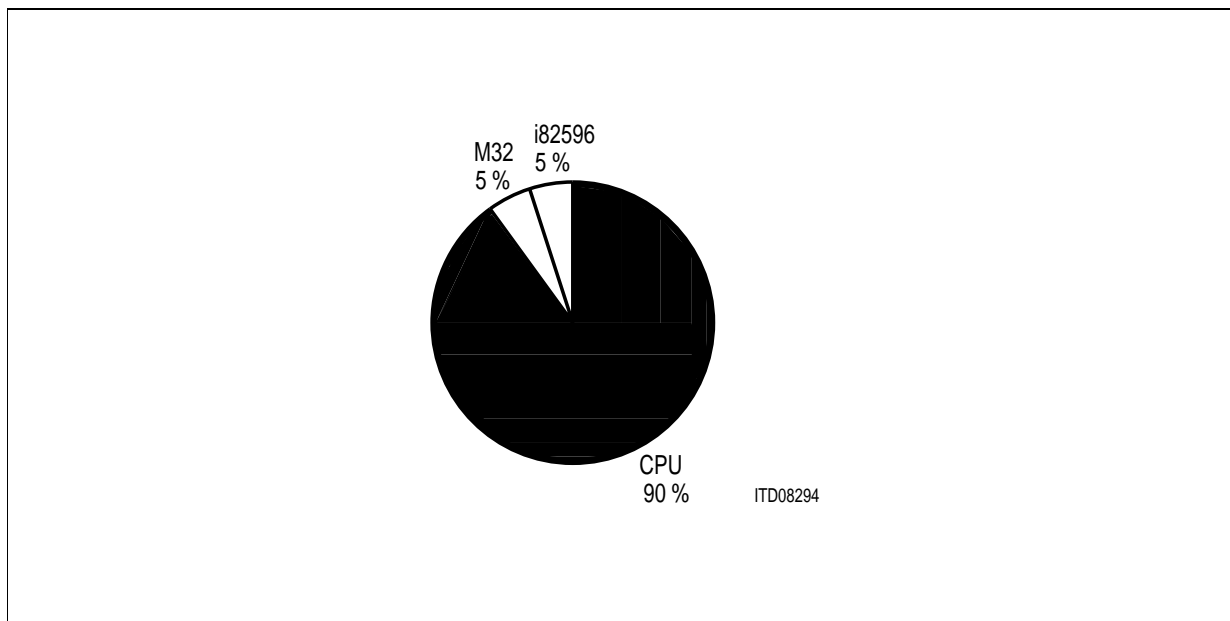
**Figure 95**  
**Data Rate**

The size of one data buffer is 128 Byte. If the number of channels exceeds 24 the data rate depends on the MUNICH32 transmitter. If the transmitter is initialized the data rate decreases. This shows the influence of the MUNICH32 polling the Hold bit.

## Bus Occupancy

The bus occupancy during normal operation is shown in **Figure 96**. In this case the data buffer size was 32 Byte. The CPU has busmastership during 90% of the time. The MUNICH32 as well as the LAN controller, each have busmastership 5% of the time.

The bus occupancy of the MUNICH32 is calculated to 2.5%<sup>1)</sup>. In this system it is higher because of inserted wait states in every bus cycle. Another reason is the bus controller's clock which is switched from 33 MHz to 40 MHz. This and the existence of two alternate bus masters results in a more time consuming arbitration protocol than that needed for a simpler architecture.

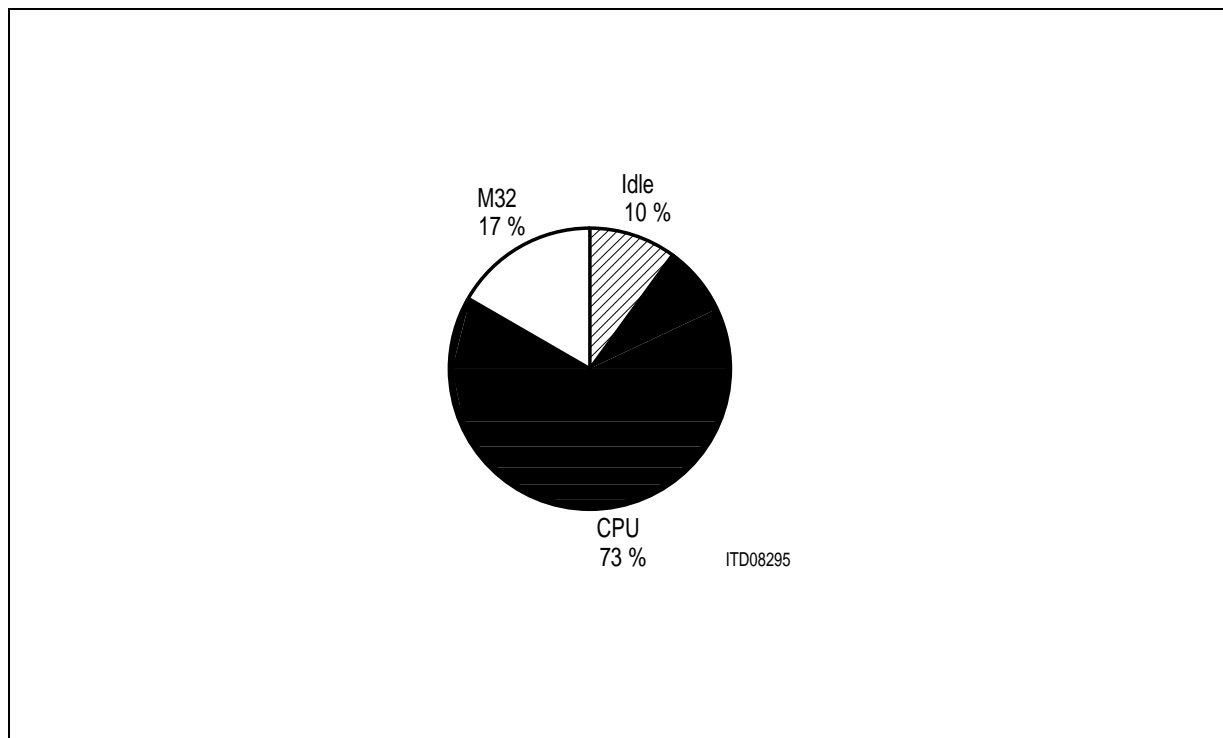


**Figure 96**  
**Bus Occupancy**

<sup>1)</sup> Compare Data Sheet.

## MUNICH32 Polling

The influence of the polling can be illustrated by showing the bus occupancy of MUNICH32 poll accesses only.

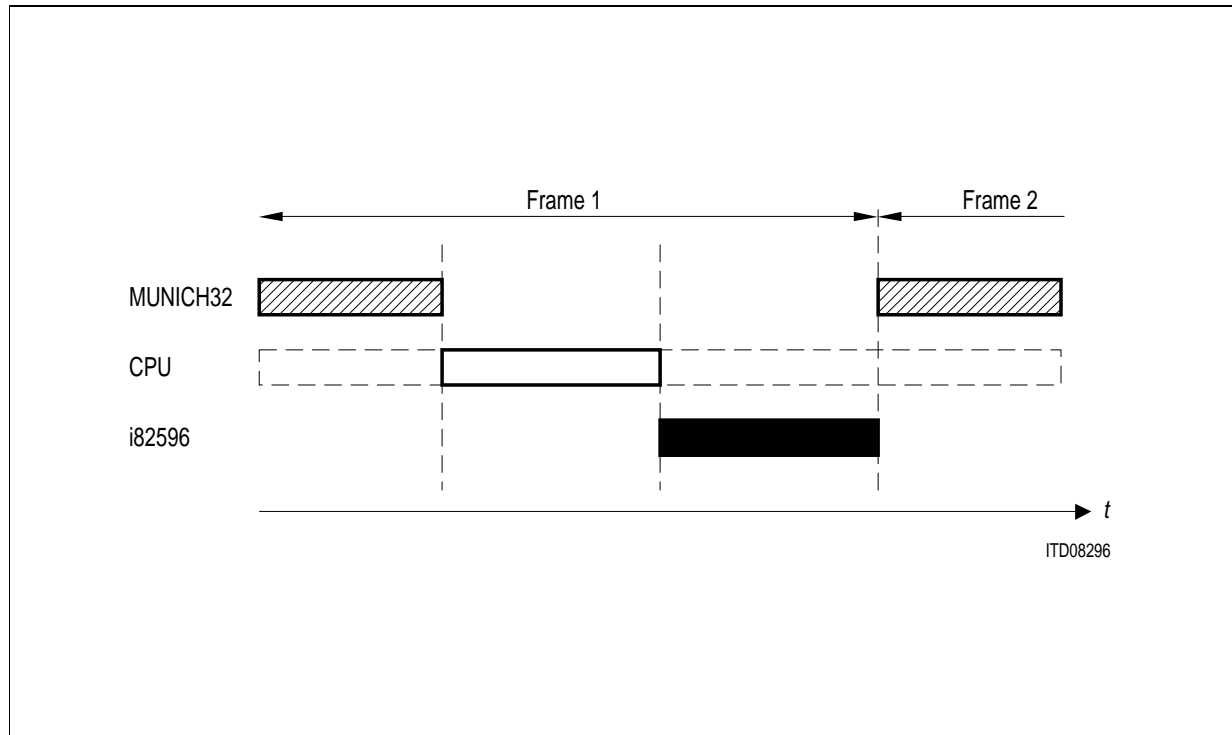


**Figure 97**  
**Bus Occupancy During Polling**

Here the MUNICH32 is polling 31 channels ( $= 31 \times 2$  read accesses during  $125 \mu\text{s}$ ). Every access is 5 clock cycles long, instead of the minimum length of 4 clock cycles. The time for the arbitration protocol needed during every access results in bus idle time.

## Frame Processing

During normal operation the processing of a frame comprises three consecutive parts. During transmission from ISDN to LAN the frame is first processed from the MUNICH32, then from the CPU and finally from the LAN controller.



**Figure 98**  
**Frame Processing**

Though the CPU is never idle, its part on frame processing is that between the MUNICH32 and the LAN controller are active. The time to process one frame is the minimum delay required between frames during continuous transmission.



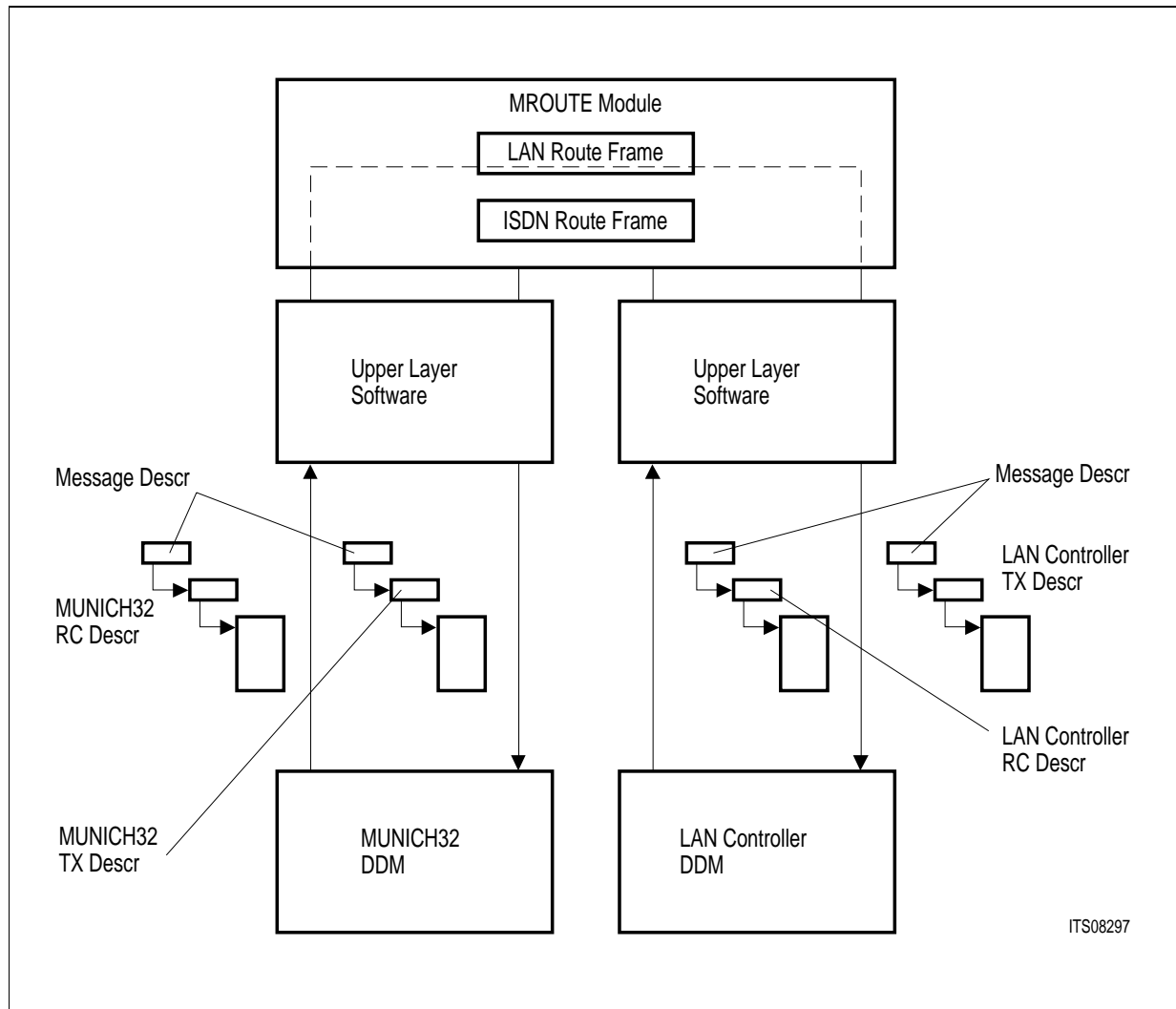
### 5.2.5 Final Remarks

This application note shows a design example for the MUNICH32 (PEB 20320). Though the design example is of reduced complexity it gives an idea of how to use the MUNICH32 in a system. The MUNICH32 is integrated in a 68040 processor system in conjunction with one more alternate bus master.

To achieve higher data rates the time to process the frames should be minimized. This includes minimization of bus idle time. The bus arbitration still has big improvement potential because of its modular structure. Additionally the existence of the alternate bus masters results in clocking the bus controller with two different frequencies. This also results in increased idle time for the bus should therefore be modified. Furthermore frame processing could be shortened by eliminating the wait states in every bus cycle.

The influence of MUNICH32 poll accesses is extremely high in this example, because of the bus arbitration architecture and the system architecture with one bus controller for all bus masters. But anyway it should always be kept in mind, that the bus occupancy during polling is higher than during transmission. During transmission it decreases rapidly.

No upper layer software is realized in this example so far. For 'real life' routing layer 2 and 3 software module(s) have to be integrated.



**Figure 99**  
**Integration of Upper Layer Software**

### 5.2.6 Adaption of the 68040 $\mu$ P Signals

begin header

This GAL is used to adapt the 68040  $\mu$ -processor signals to the MUNICH32. It is used in a system with a frequency relationship of 1/2 PCLK/SCLK.

end header

begin definition

```

device      gal1 6v8;                {Select the device to be Gal16V8}

input       bclk      = 1,
            M32ASQ     = 2,
            reset      = 3,
            M32BGACKQ  = 4,
            int         = 5,
            ACREQM68    = 6,
            RWQ         = 7,
            clk         = 8,        {= musclk}
            rclk        = 9;

feedback (com)  DSACKQ      = 19;

output (com)    TSQ         = 18,
            RESETQ        = 17,
            INTQ          = 16,
            ACREQM32      = 15,
            sclk          = 14;

statebits      sb2        = 13,
            sb1          = 12;

state_names    idle       = 2,
            one          = 1,
            two          = 0;

```

end definition

```
begin equations
    TSQ.OE          = /M32BGACKQ;
    TSQ             = /(M32ASQ × DSACKQ);
    RESETQ         = /reset;
    INTQ           = int;
    ACREQM32       = /(ACREQM68 × reset × /RWQ);
    sclk           = (/reset × rclk) + (reset × /clk);
end equations
```

```
begin state_diagram tktadaptor (sb2, sb1)
```

```
    state all:
    if (/reset + M32ASQ) then idle
    with DSACKQ = 1;
    endwith;

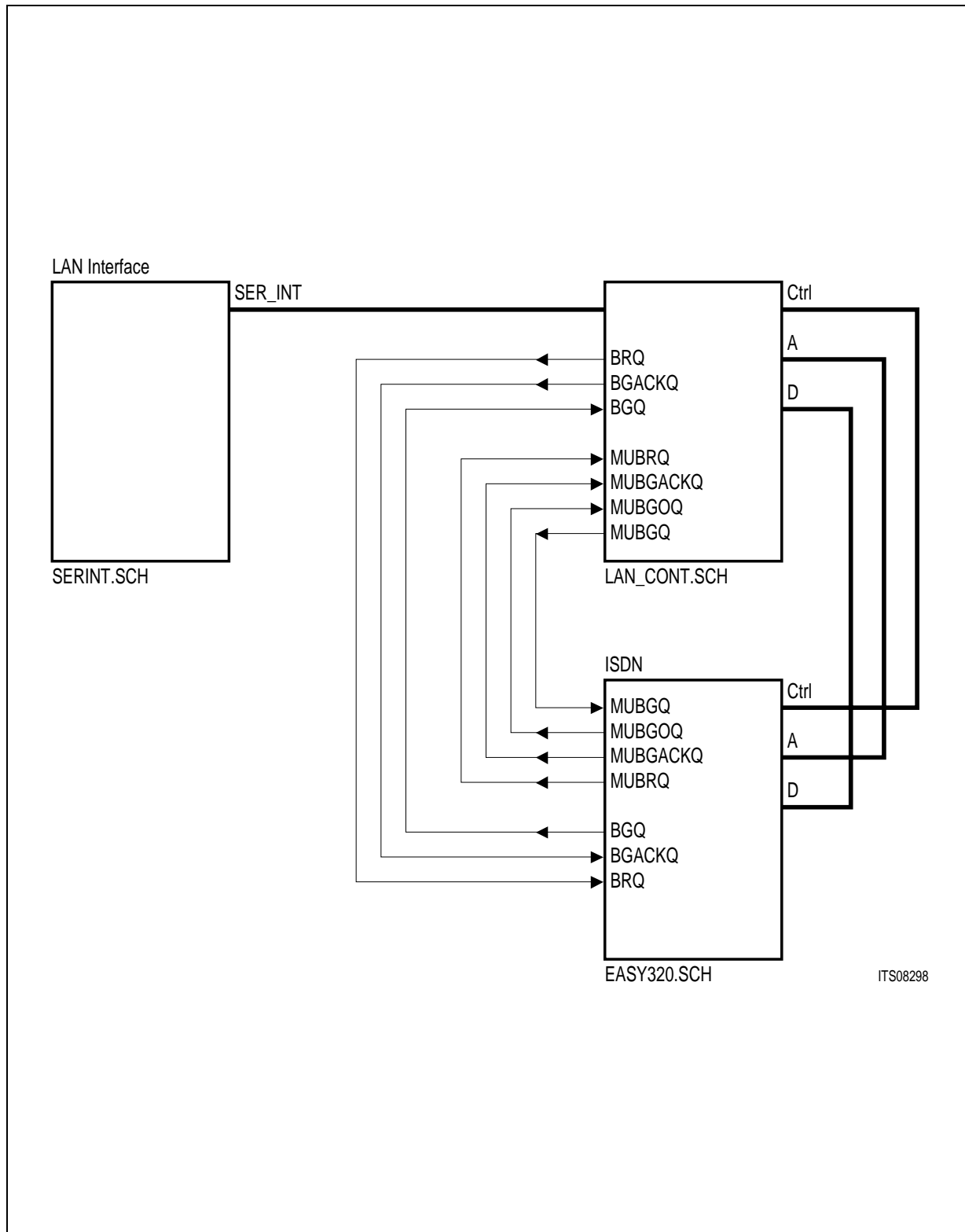
    state idle:
    DSACKQ = 1;
    if (/M32ASQ × reset) then one else idle;

    state one:
    DSACKQ = 1;
    go to two;

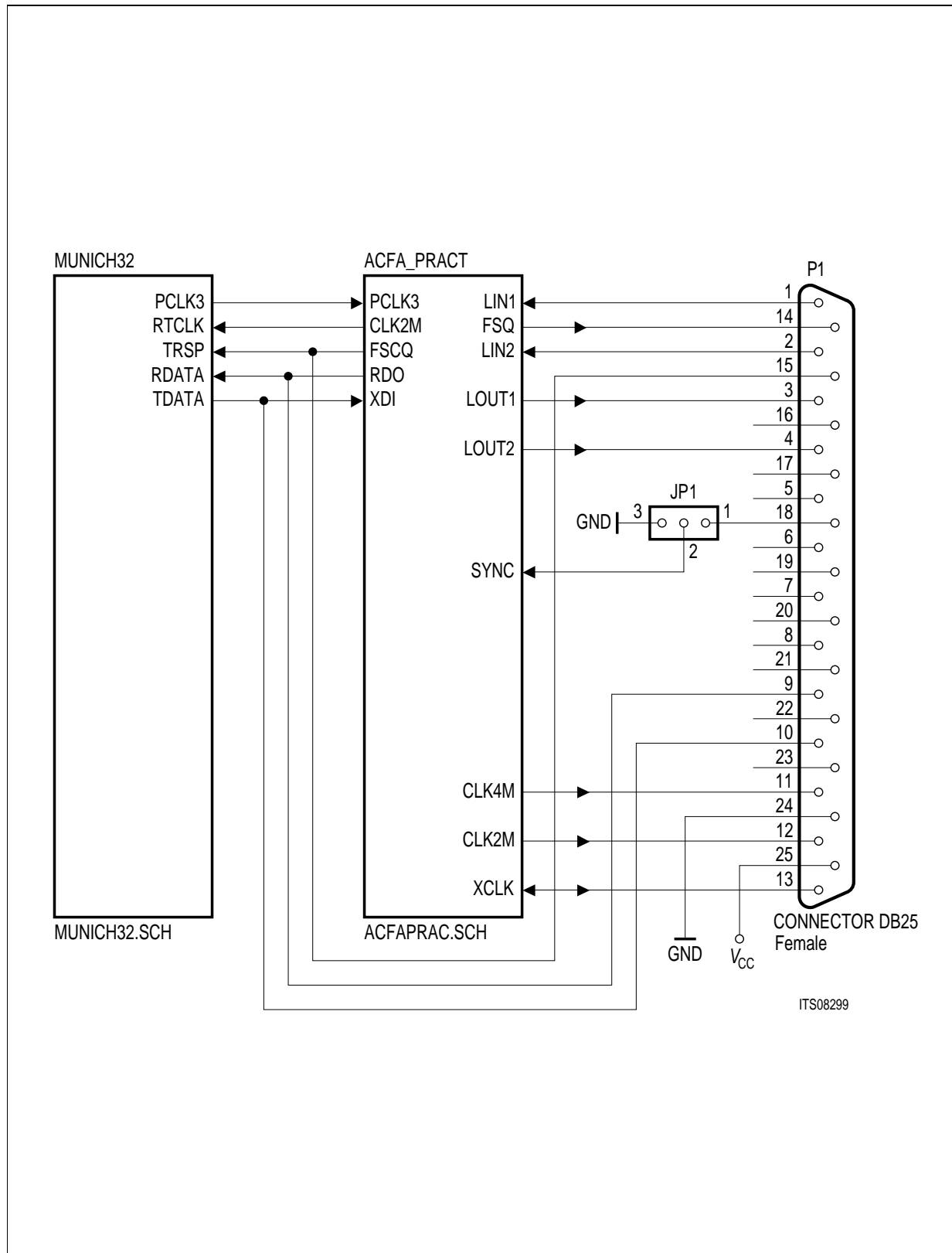
    state two:
    DSACKQ = 0;
    if M32ASQ then idle else two;
```

```
end state_diagram
```

## 5.2.7 Schematics



**Figure 100**



**Figure 101**

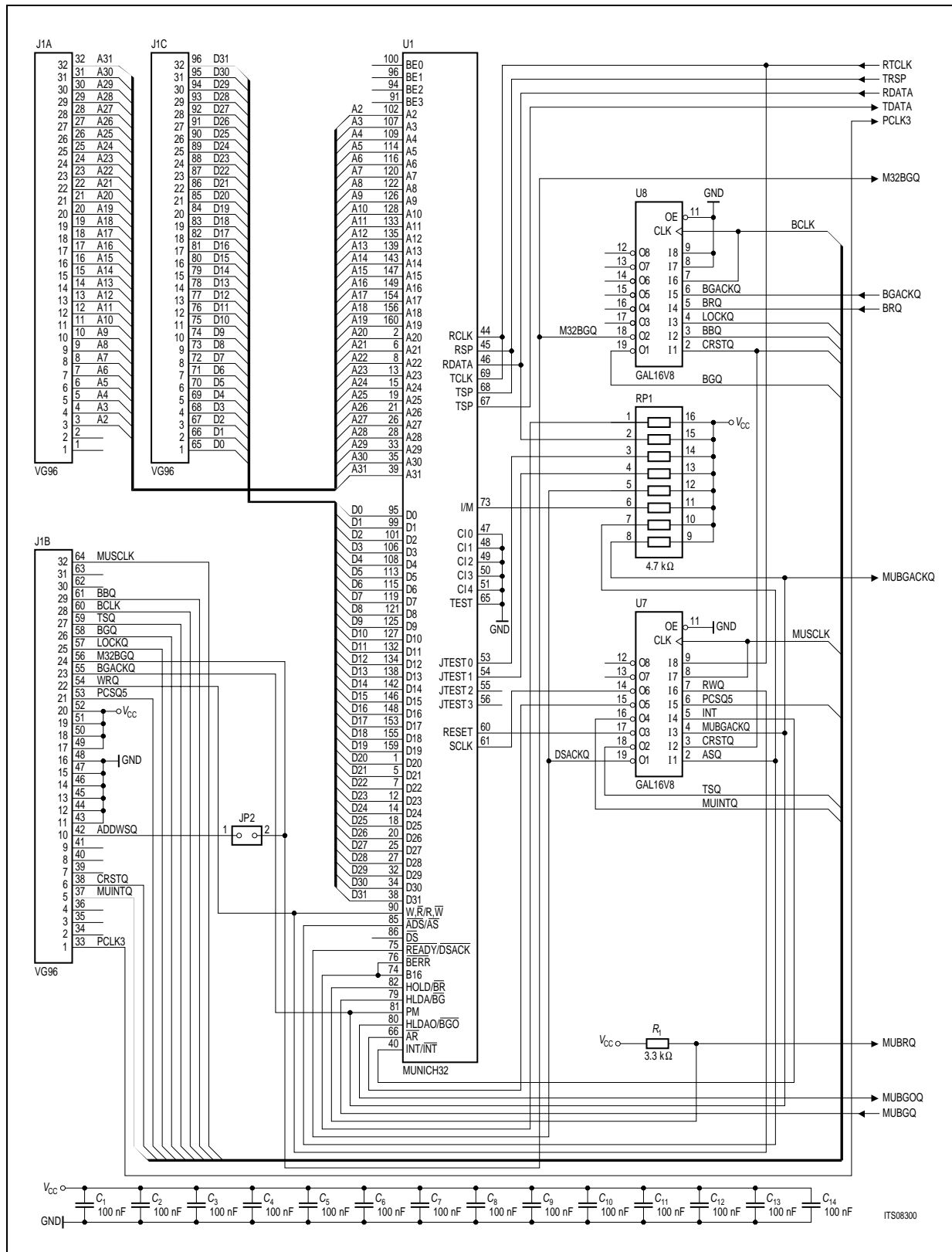
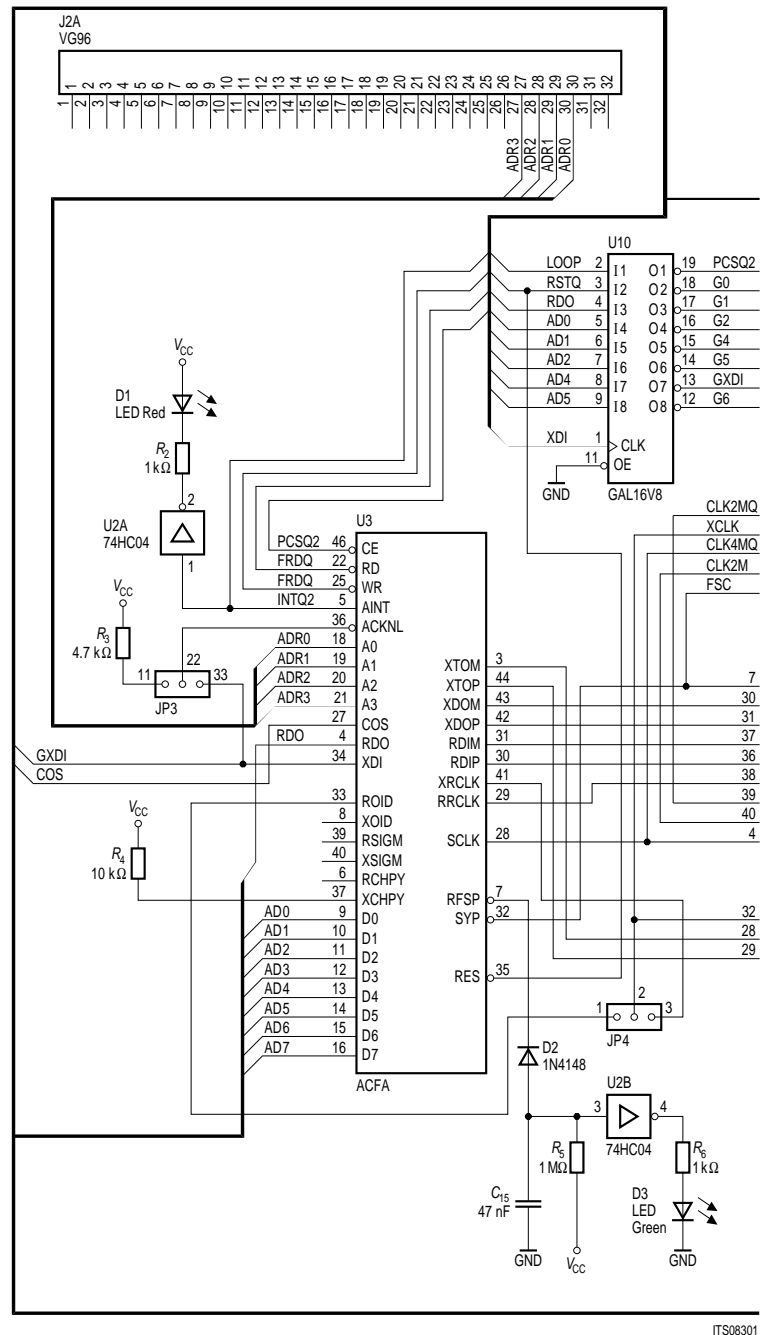
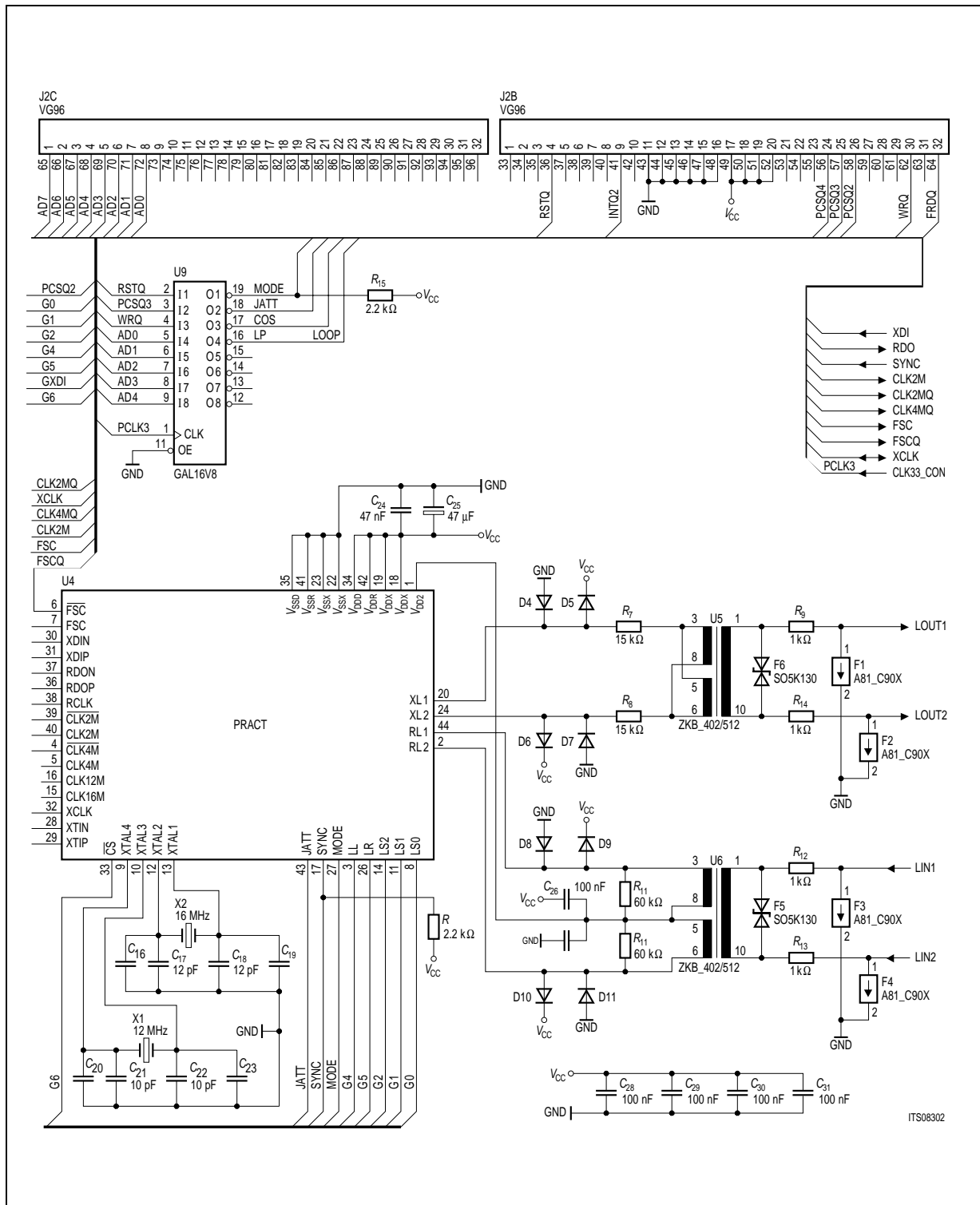


Figure 102



### Figure 103





**Figure 104**

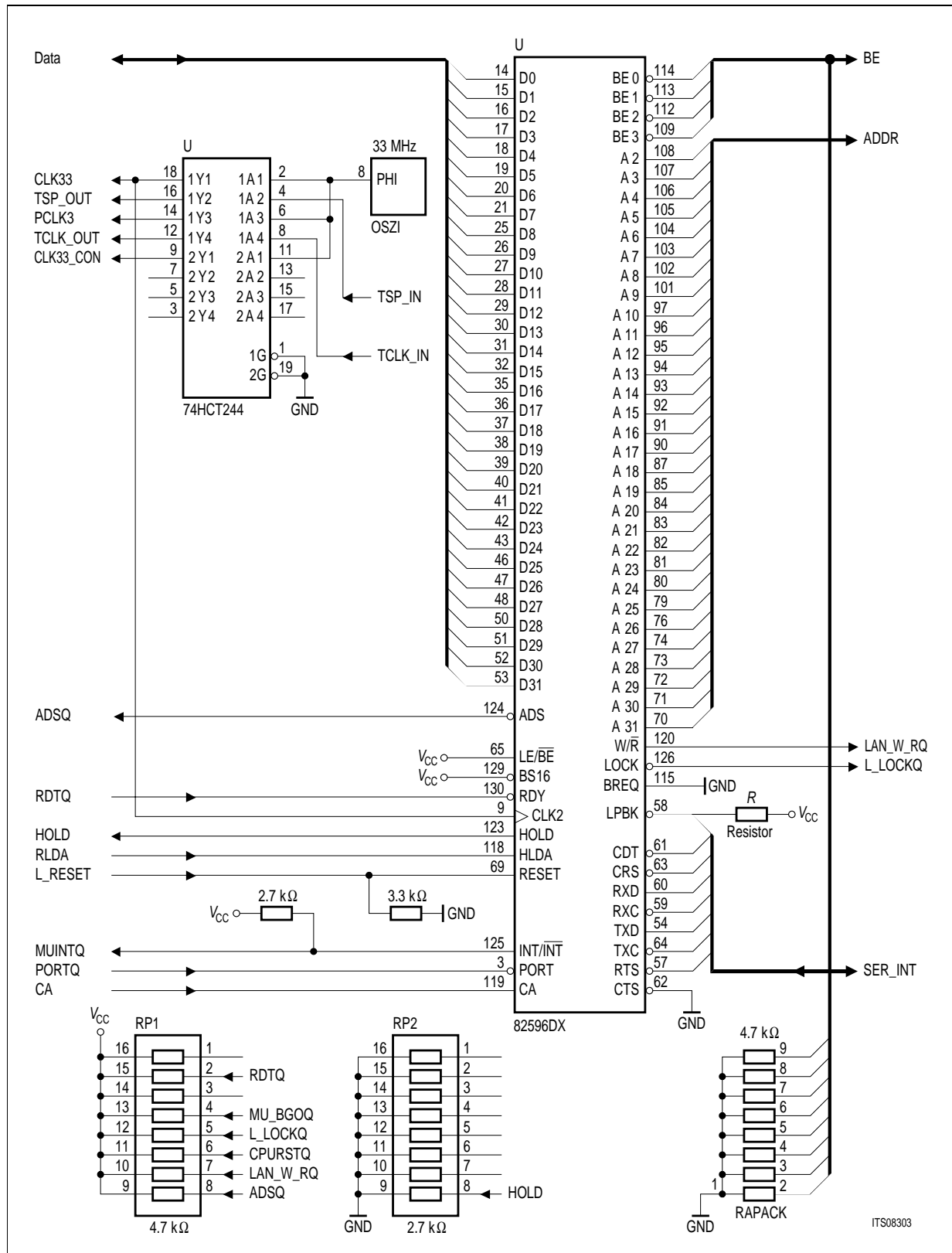
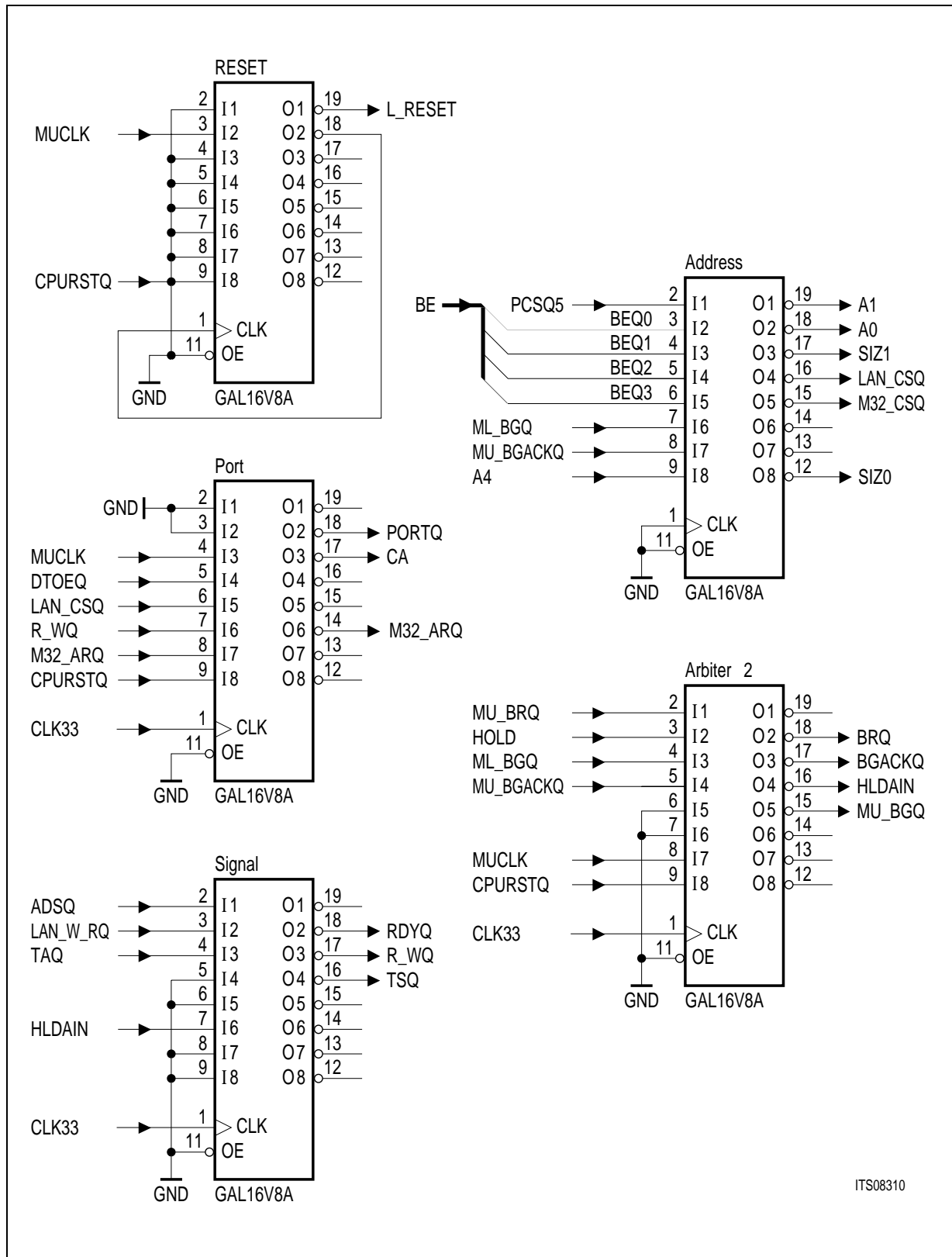


Figure 105



ITS08310

Figure 106

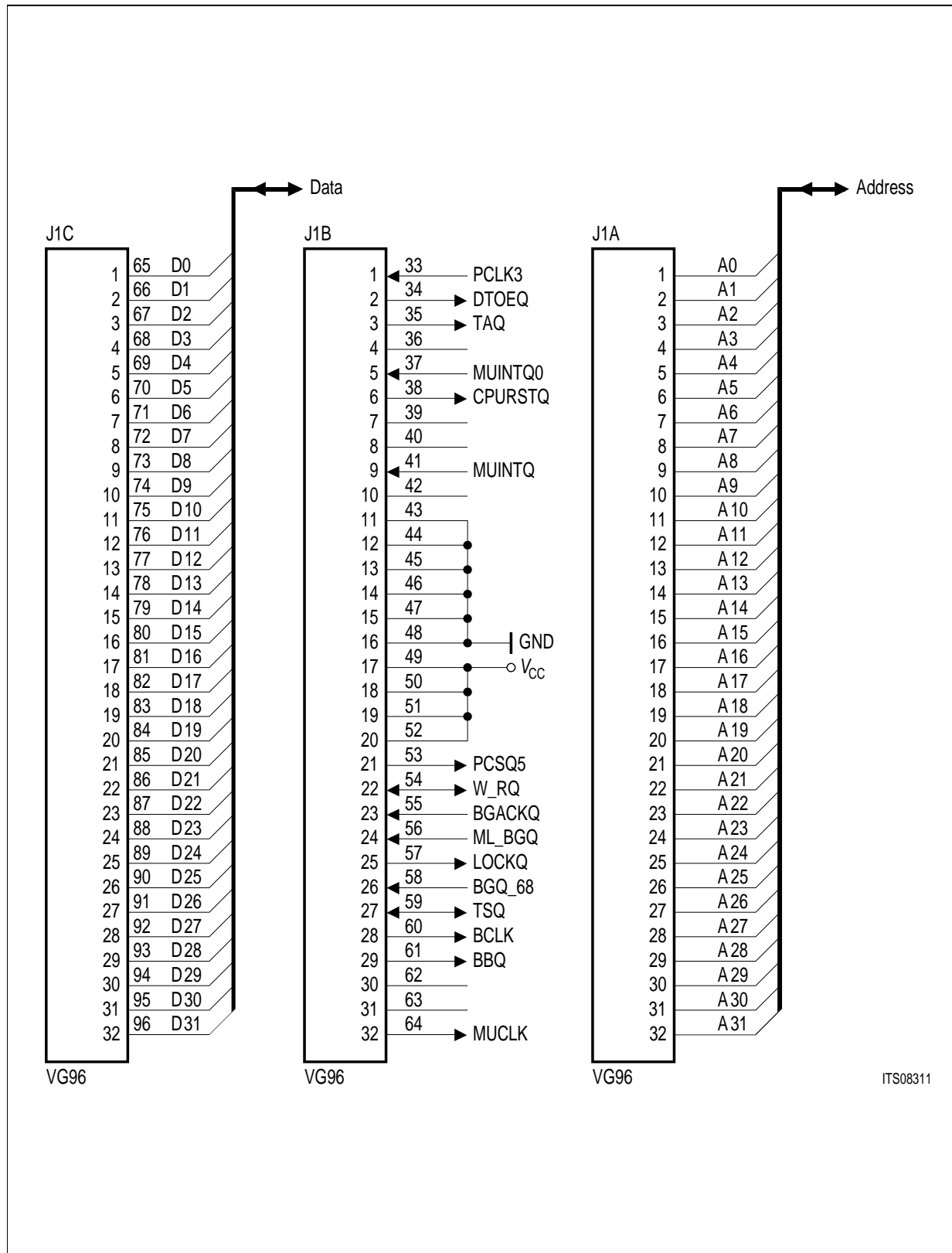


Figure 107

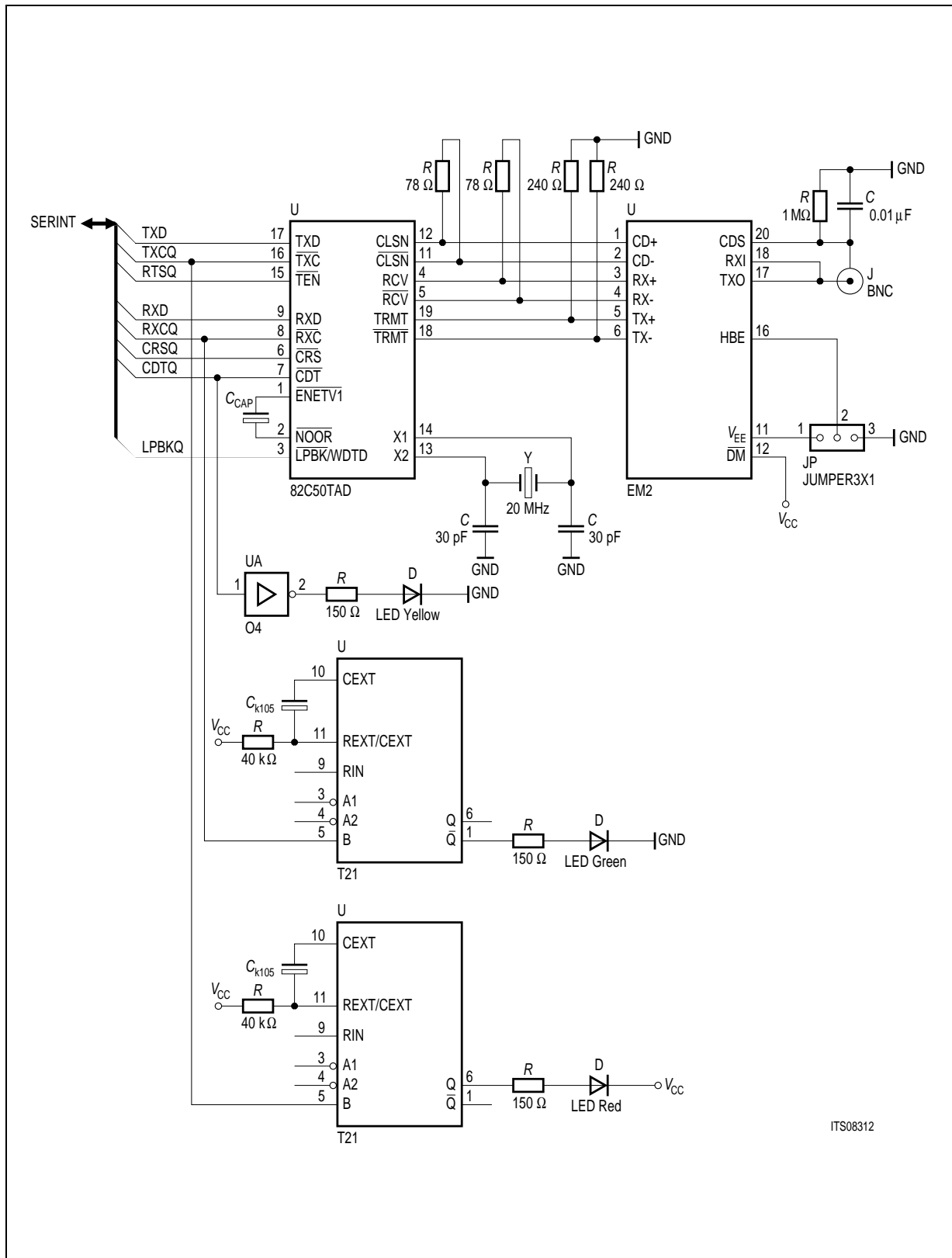


Figure 108

### 5.3 Memory Bus Occupancy for a Single MUNICH32

The MUNICH32 may be used in different system architectures depending mainly on how the data buffers are shared between the interacting bus masters. In the following the memory bus occupancy is calculated for a system, where the MUNICH32 is directly coupled with a 32-bit CPU (compatible to either Motorola 68020 or Intel 386) sharing one common local CPU bus and translated via an appropriate system bus controller sharing the system memory as well. This example system looks very similar to the one depicted in the **Figure 7** and **Figure 9** of **Chapter 1**. In this case it is easier to estimate the behavior of the complete system.

In addition to that, the following assumptions are made about the communication parameters:

- HDLC operating mode
- the MUNICH is clocked with SCLK = 16 MHz
- the bus arbitration time is estimated to be about 4 extra clock cycles (SCLK) for every 10 MUNICH32 memory accesses (typical is 10 to 16)
- the data buffer size allocated in the data buffer pool is 32 bytes for transmit and receive descriptors
- a full duplex connection with up to  $32 \times 64$  Kbit/s channels and heavy traffic load (shared flags)
- the data size per HDLC frame is defined to be without the shared flag and the two CRC bytes
- when the data size exceeds 32 bytes, more than one descriptor is needed for a single frame
- an interrupt information is generated for every descriptor.

The MUNICH32 needs the following 32-bit memory accesses (read or write):

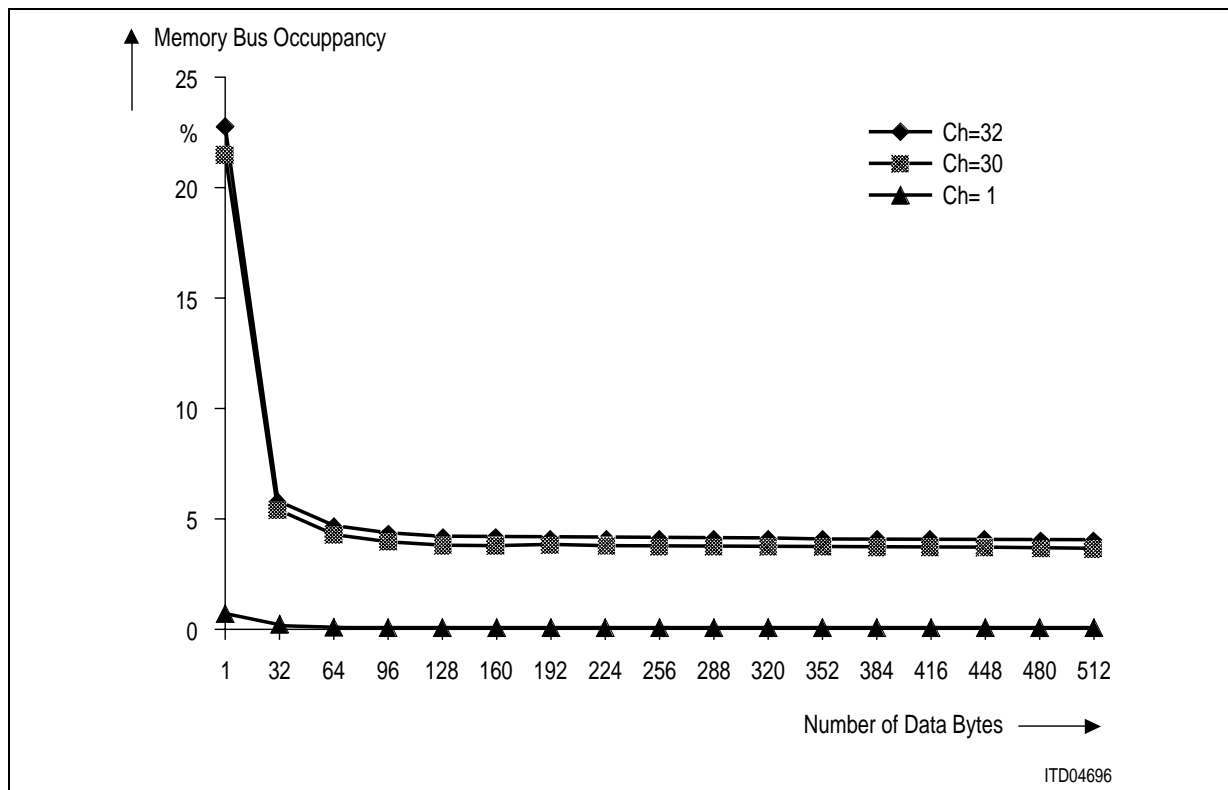
Receive:	read descriptor	→ 3	
	write current descriptor address	→ 1	
	write status	→ 1	
	write interrupt	→ 1	
	write data (size)	→ accesses	size
		1	1
		1	2
		1	3
		1	4
		2	5
		:	:

## Application Notes

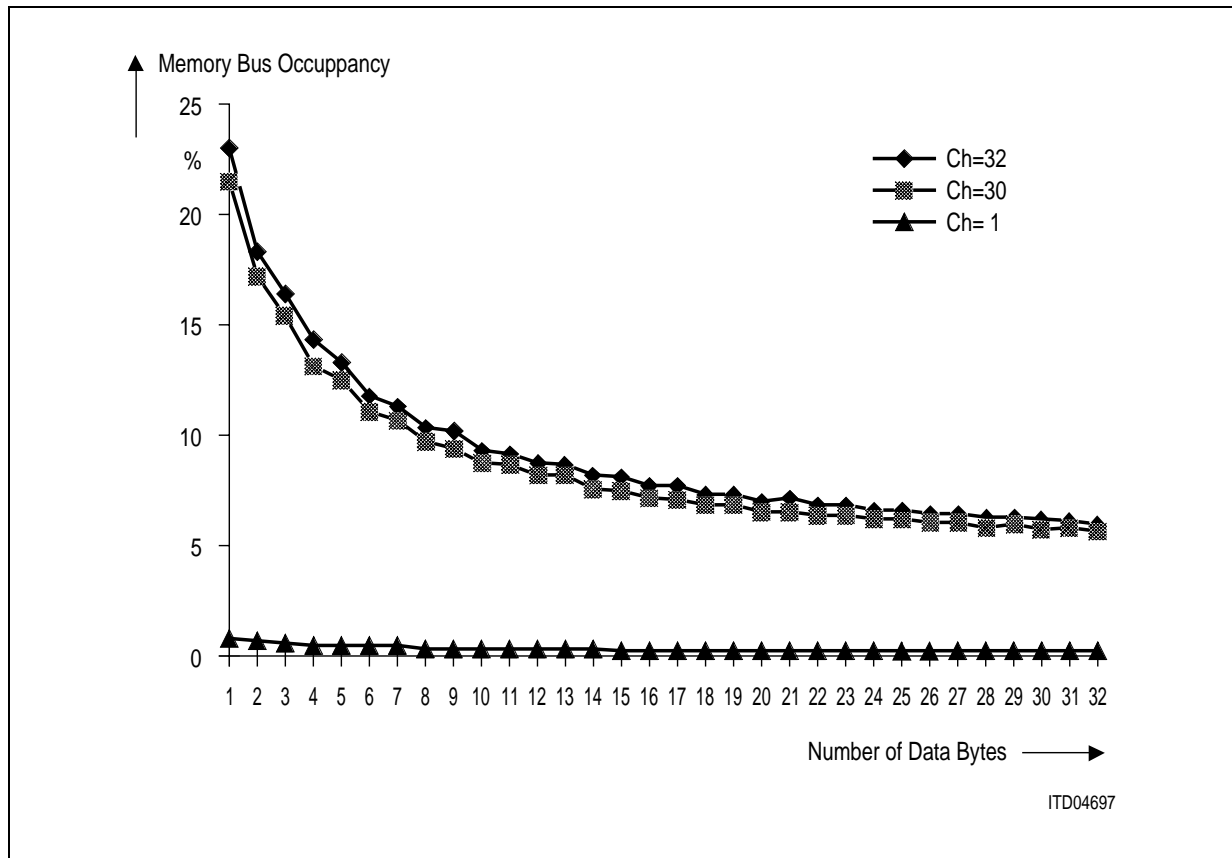
Transmit:	read descriptor	→ 3	
	write current descriptor address	→ 1	
	write interrupt	→ 1	
	read data (size)	→ accesses	size
		1	1
		1	2
		1	3
		1	4
		2	5
		:	:

The accumulated access time for a single MUNICH32 channel, depending on the actual frame size, is then related to the serial transfer time on a PCM system:  $(3 + \text{size}) \times 125 \mu\text{s}$ .

The following two diagrams illustrate the overall results for two different ranges and their corresponding resolution. As you can see, for frame size greater than 32 bytes the time needed for MUNICH32 memory accesses drops below 5%. That means in a simple communication subsystem (e.g. Primary Access Board) the CPU performance is also reduced by 5% only and it is therefore not necessary to use a complex multiport memory approach to reach a significant overall performance gain.



**Figure 109**  
**Frame Size 1 to 512**



**Figure 110**  
**Frame Size 1 to 32**



### 5.3.1 Bus Occupancy Calculations

As described in the previous section, the MUNICH32 in a steady state condition consumes approximately 5% of the system bus bandwidth. Based on the conditions previously described, a set of equations can be used to describe the MUNICH32 system bus behavior. Other MUNICH32 systems can be evaluated using these equations. The Bus occupancy is defined as the ratio of the time required for memory accesses for that data to the time used to send the data. The two equations are defines as follows:

Time used for memory accesses:

= Number of received bits plus transmitted bits multiplied by the time required to transfer this

information to/from memory.

$$= \{([6 + (1 + m)] \times rc) + (5 + (1 + m)] \times tc)\} \times (1 + 1/ba) \times NC \times sclk$$

6 for receive descriptor access

5 for transmit descriptor access

(1 + *m*) for data access where *m* is the largest integer smaller (*n* – 1)/4

(*n* is the number of transmitted data bytes).

*rc* is the number of receive channels.

*tc* is the number of transmit channels.

(1 + 1/*ba*) is the bus arbitration time

*sclk* is the system clock (61 ns for 16.384 MHz)

*NC* is the number of memory clocks per bus operation (0ws = 4, 1WS = 5, etc.).

Time used to send the data is the number of transmitted bits per time slot multiplied by the frame time:

$$= ((4 + n) \times 8/abtc) \times 125 \mu s$$

4 because shared flags are not used + 2 byte CRC

*n* is the number of octets to transmit

*abtc* = assigned bits to channel

e.g. a channel with one time slot of 1 bit would require 8/1 = 8 time slots to transmit a single octet.

From the previous example, the variables are assigned the following values:

Variable	n	m	rc	tc	(1 + 1/ba)	sclk	NC	abtc
Value	32 bytes	7	32	32	1.1	61 ns	4	8

Applying these values to the equations yields the following:

$$\begin{aligned}
 &\text{Time used to access memory} \\
 &= \{([6 + (1 + 7)] \times 32) + ([5 + (1 + 7)] \times 32)\} \times (1.1) \times 4 \times 61 \text{ ns} \\
 &= \{(14 \times 32) + (13 \times 32)\} \times 1.1 \times 244 \text{ ns} \\
 &= \{864\} \times 268.4 \text{ ns} \\
 &= 231.9 \mu\text{s}.
 \end{aligned}$$

$$\begin{aligned}
 &\text{Time used to send data} \\
 &= ((4 + 32) \times 8/8) \times 125 \mu\text{s}. \\
 &= 36 \times 125 \mu\text{s}. \\
 &= 4500 \mu\text{s}.
 \end{aligned}$$

$$\text{Bus occupancy} = 231.9 \mu\text{s} / 4500 \mu\text{s} = 5.1\%$$

When the packet size is much larger (256 bytes or larger), the bus occupancy decreases to less than 4%. Conversely, sending very small frames (4 bytes), causes bus occupancy to increase to over 11%. This is primarily due to the increased descriptor processing per packet.

### 5.3.2 Bus Occupancy for Idle Tx Channels

The previous discussion shows bus occupancy to be very low, even when a MUNICH32 is processing 32 channels of receive and transmit data. There is another system consideration of bus occupancy that must be examined. When a MUNICH32 channel has no data required for transmit, the channel must be temporarily (or permanently) stopped. There are several methods that may be used to stop the transmission.

1. The first method involves executing a channel command with TH = 1 (reactivation of the channel requires a new channel command with TH = 0). This method places the transmit channel on hold and prevents any further accesses of the memory for this channel.
2. A second method is based on statistical knowledge of the frequency of transmitted frames. If frames are transmitted without shared flags and if the average number of interframe time fill characters can be determined, the MUNICH32 can be programmed to suppress poll sequences. By setting FNUM in the Tx descriptor to a value (n) greater than 0, the MUNICH32 will transmit n + 1 idle characters after the end of the current frame. During this period of interframe time fill, the MUNICH32 will not poll the Tx descriptor. As an example, if it is determined that 5 idle characters typically occur between frames, FNUM can be set to 4. At the end of the current frame, 5 idle characters will be transmitted (625  $\mu\text{s}$ . on a DS0 channel) before the next frame is transmitted and no polls of the Tx descriptor will occur during that time.

---

**Application Notes**

3. The final method is to set the HOLD bit in the Tx descriptor. When the HOLD bit in the Tx descriptor is set, the MUNICH32 checks the status of the this bit for each time slot assigned to this channel. In this way, if the bit has been cleared, the MUNICH32 will immediately resume transmission. Although this method is simpler (in concept) for the software design, it causes the MUNICH32 to consume higher than normal bus bandwidth. For this reason, this is the least desirable of the three methods. In the previous example discussed, if all 32 channels were holding on the Tx descriptors, bus occupancy might rise as high as 17%. The reason bus occupancy rises this dramatically is due to the bus access once per time slot rather than once every four time slots (typical).

## 6 Application Hints

### 6.1 Frequency Adaption in an Intel 386 Common Bus System

If you use the i386 as host processor with the MUNICH32 in a common bus system you have to adapt the different frequencies of the devices. The MUNICH32 works e.g. with a fixed frequency of 16.384 MHz in CEPT 32 channel PCM highway format. The i386 works with frequencies from 16 up to more than 50 MHz. If you compare the timing diagrams you will see that a few glue logic is necessary to adapt the MUNICH32 to the i386 timing.

A possible adaption of the different frequencies is described below. For an example we use an i386 with a frequency of 16.384 MHz. The MUNICH32 is configured in the CEPT 32 channel PCM highway format with a SCLK of 16.384 MHz. The SCLK signal is build by dividing the 32.768 MHz CLK2 signal of the i386. That means that both clocks are synchronous. This is not necessary in general but selected in our example. The bus controller generates e.g. one wait state for the memory access. The falling edge of the ADS signal marks the beginning of a bus cycle which is completed with the sampled READY signal. A general bus controller should not see a difference between the two bus masters, so we have to delay the falling edge of the MUNICH32 ADS signal to that moment as the i386 would generate its ADS to get the READY signal at the same time. In the picture below you can see the relationship and the adaption of both timings as specified in our example. A second picture shows the adaption in an i386 24.576 MHz system. Again the clocks are synchronous.

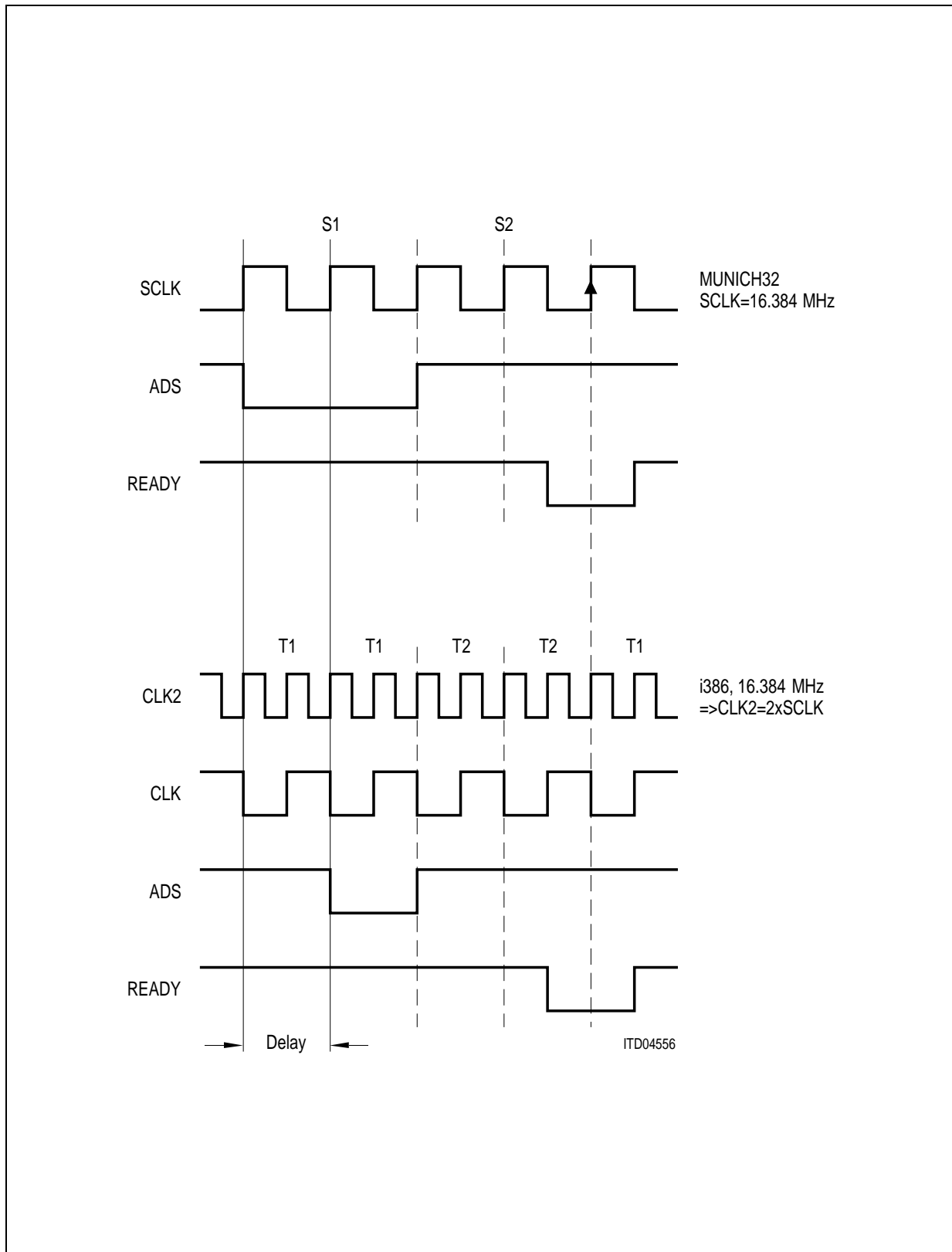


Figure 111

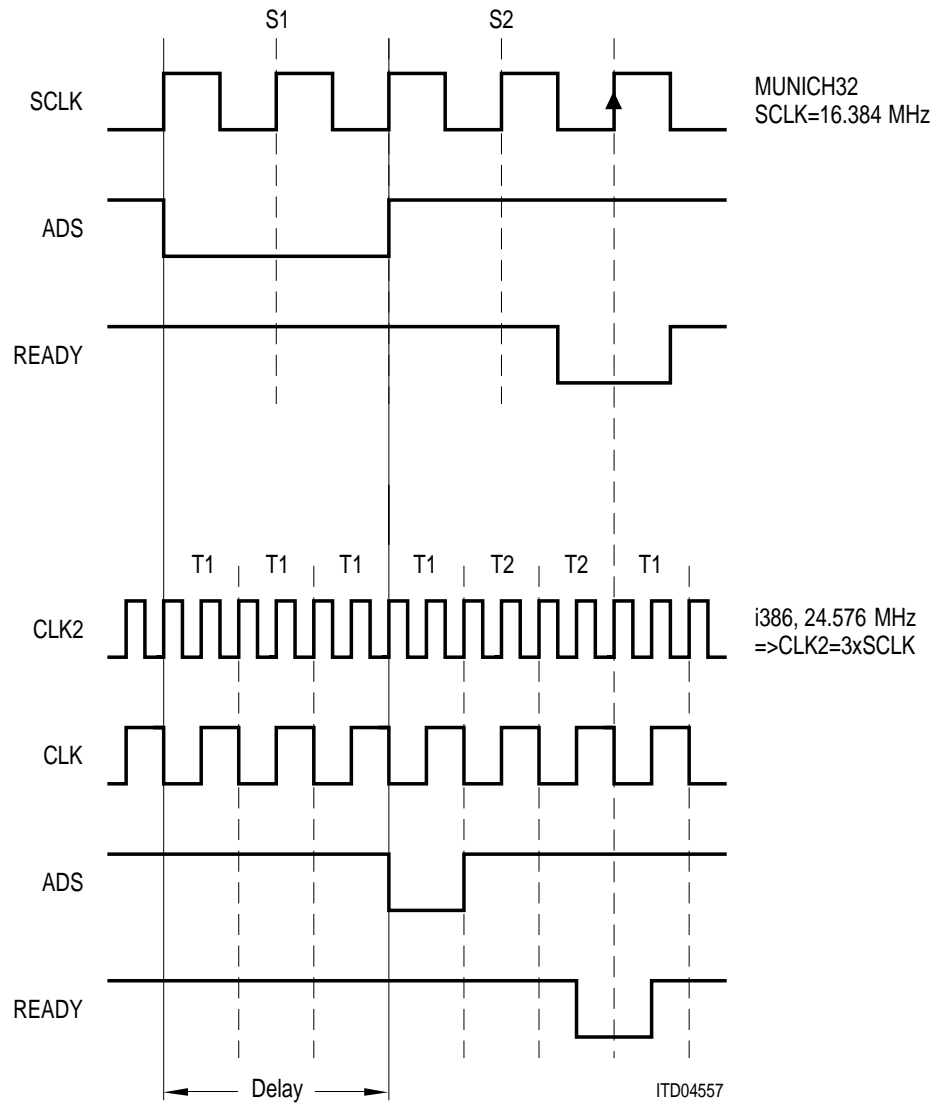


Figure 112

## 6.2 MUNICH32 Memory Space Requirement

Implementation independent:

- Start Address 4 byte
- Control & Configuration Section 908 byte
- Tx Descriptor Size 12 byte
- Rc Descriptor Size 16 byte

Implementation dependent:

- Interrupt Queue Size 64 byte < Interrupt Queue Size < 16384 byte
- Data Buffer Size Data Buffer Size
- Allocation of Tx and Rc descriptors per channel

In general the memory space requirement may be calculated the following way:

$$\begin{aligned} & \text{Start Address} \\ & + \text{Size of Control \& Configuration Section} \\ & + \text{Interrupt Queue Size} \\ & + \text{number of channels} \times [\text{number of Tx Descriptors} \times (\text{Tx Descriptor Size} + \text{Data Buffer Size})] + \\ & \text{number of channels} \times [\text{number of Rc Descriptors} \times (\text{Rc Descriptor Size} + \text{Data Buffer Size})] \\ & \hline & = \text{Total MUNICH32 Memory Space Requirement} \end{aligned}$$

### Example:

The MUNICH32 is used in a 31 channel ISDN Primary Access application, that means that 31 full duplex channels are active. The LAPD protocol is implemented. In this case a window size of 7 is specified, that means that 7 Rc Descriptors and in transmit direction 7 Tx Descriptors must be available for each channel. The Data Buffer Size is set to 260 byte according to the LAPD specification.

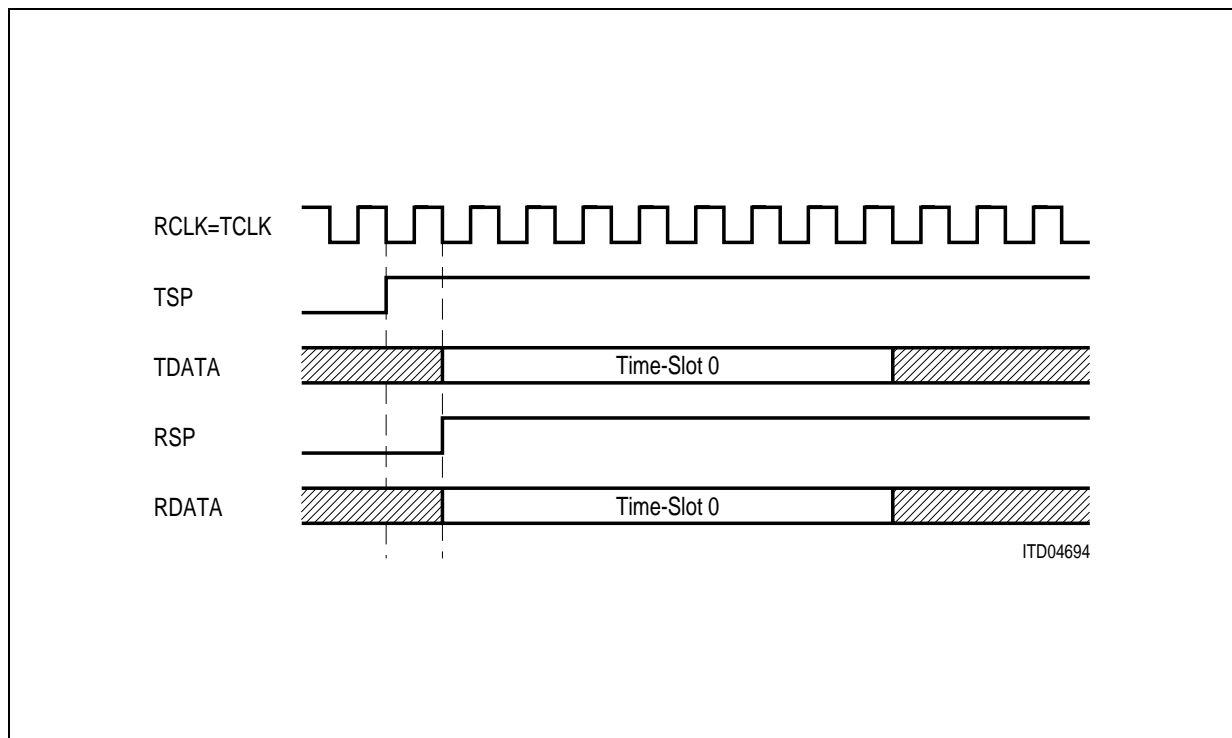
Summary:

- 31 channels;
- Interrupt Queue Size = 1024 byte;
- 7 Tx and 7 Rc Descriptors;
- Data Buffer Size = 260 byte;

In our example a memory space of 120 kbytes is required.

### 6.3 Serial Interface to different PCM Systems

The serial interface of the MUNICH32 is very general and comprises standard clock, PCM frame synchronization and data signals, which are independent for both directions. The following description explains typical applications integrating the MUNICH32 into 2.048 Mbps PCM systems, like SIEMENS System Interface for Primary Access and the MITEL ST BUS. In these systems the receive and transmit clocks are identical. The general timing is shown in **Figure 113** (see also **Chapter 2.1**).



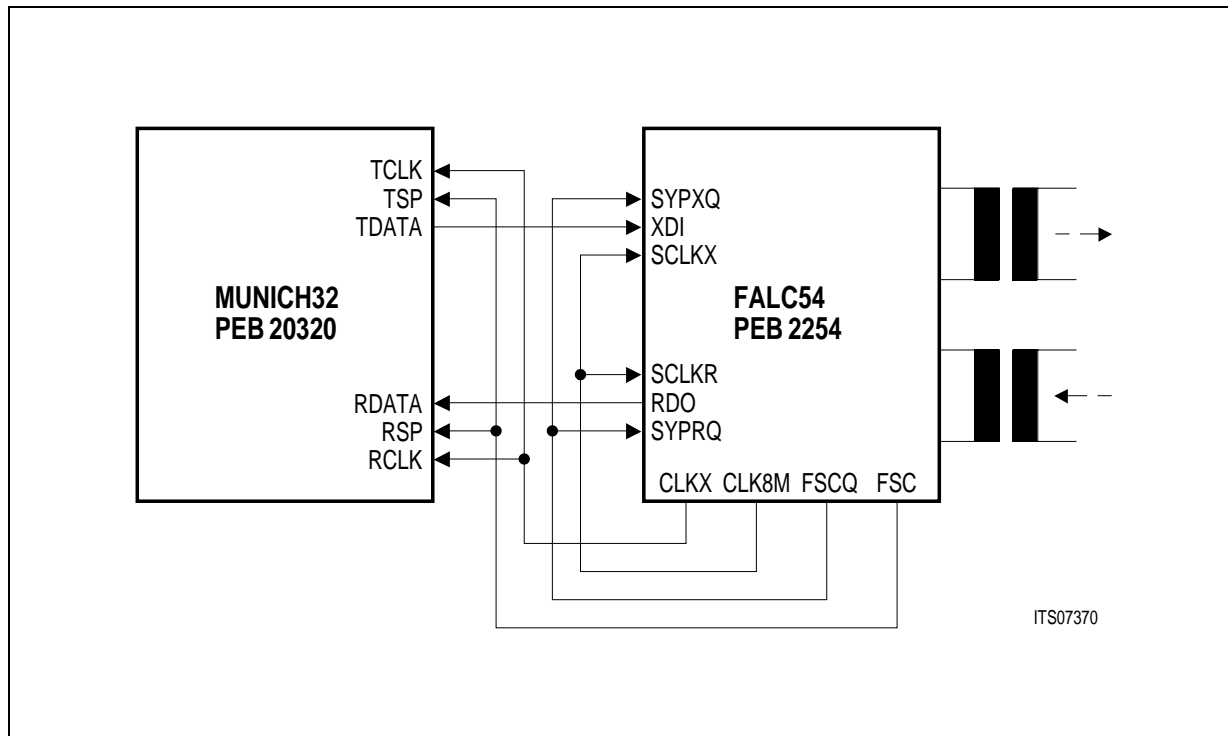
**Figure 113**

The RSP pulse is shifted by one clock period against the TSP pulse. The main task using this timing for different PCM systems is to adapt the TSP and RSP pulses appropriately, as described below.

#### 6.3.1 MUNICH32 for SIEMENS Primary Access Interface

The SIEMENS devices for the Primary Access Interface is the Frame and Line Interface Component (FALC54). This device can directly be connected to the MUNICH32 without any additional glue logic. In combination with the MUNICH32 this application is the most effective way to build a powerful and flexible Primary Access Interface, especially supporting different combined B channel paths over long distances (LAN-WAN Internetworking). The following block diagram illustrates how easy it is to integrate the MUNICH32 into a Primary Access application based on SIEMENS devices.




**Figure 114**

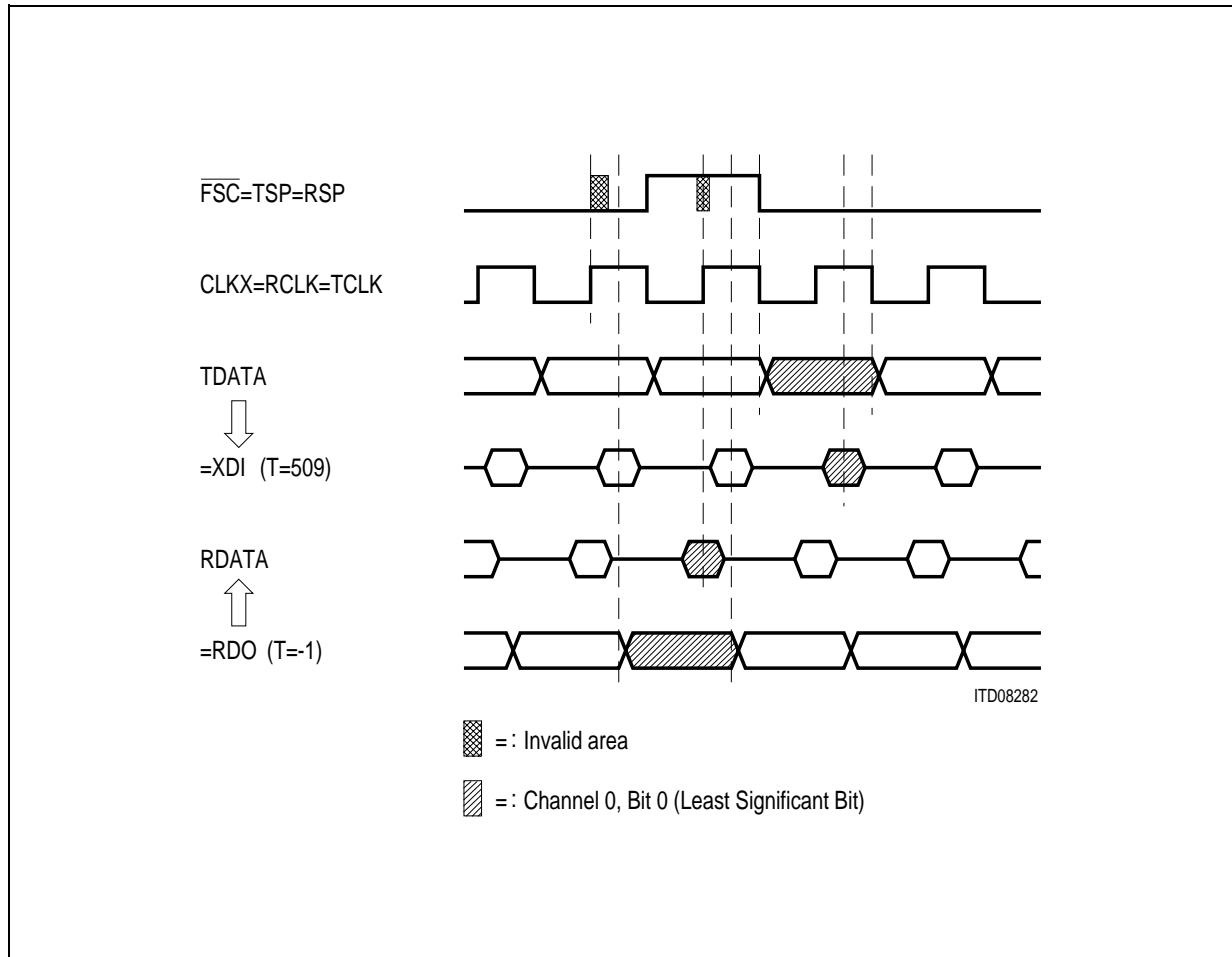
The adaption of the TSP and RSP pulses is solved by means of shifting the receive data and transmit data in the FALC54 device appropriately. In this case the TSP and RSP synchronization pulses are also identical. The FALC54 device contains special registers to control the bit shift of the serial bit streams at the system interface (see FALC54 Data Sheet). With the following register programming the bit shift selected is  $T = 509$  for the MUNICH32 transmit data and  $T = -1$  for the receive data respectively. The programming is as follows:

XDI:  $XC1.XTO = 3D_H \Rightarrow X = 494 \Rightarrow T = 509$   
 $XC0.XCO = 06_H$

RDO:  $RC1.RTO = 00_H \Rightarrow X = 5 \Rightarrow T = -1$   
 $RC0.RCO = 05_H$

## Application Hints

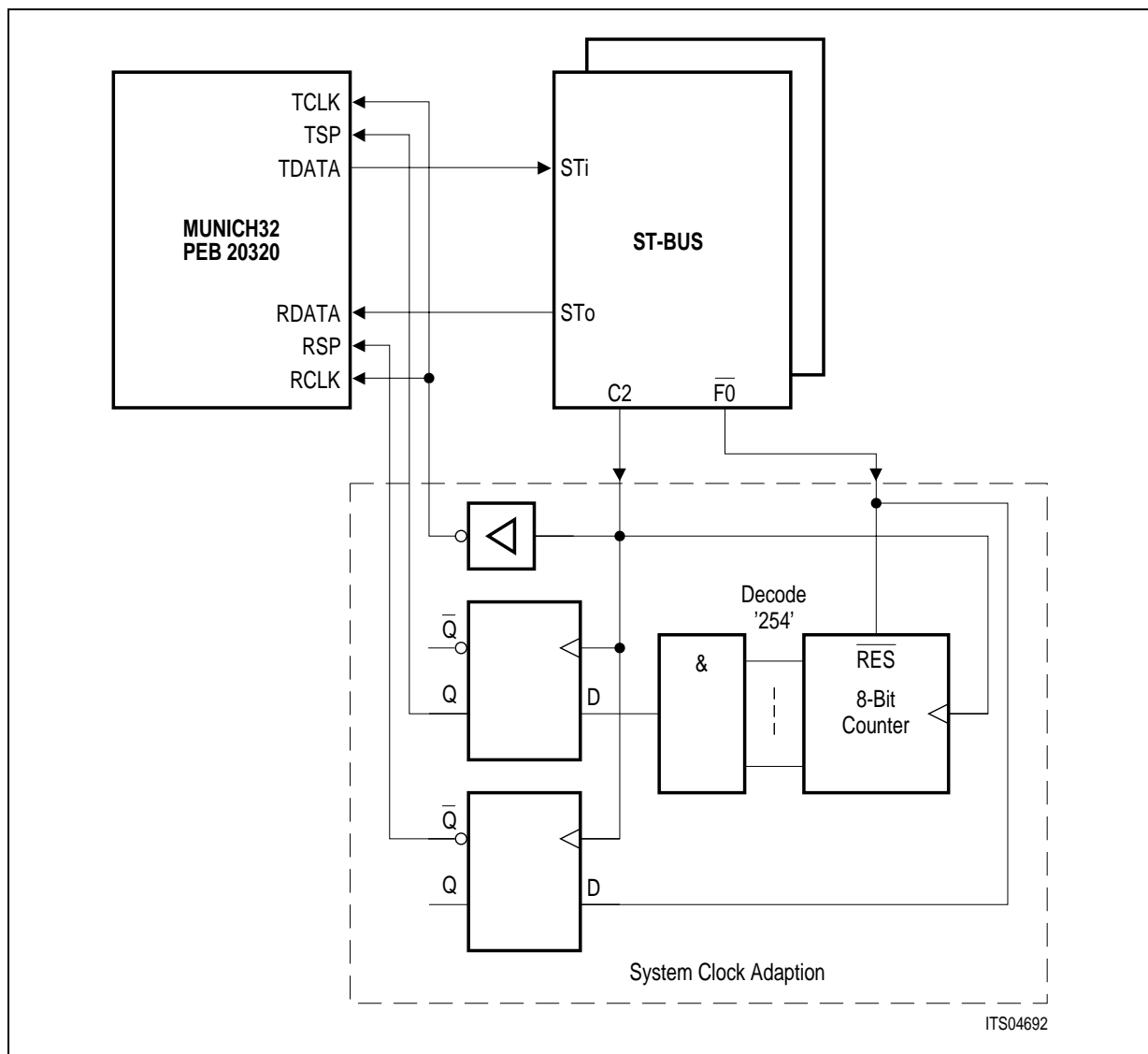
The timing in principle is depicted in the following diagram. Without all details of a typical electrical timing it illustrates how the different signals from MUNICH32, and FALC54 are mapped in such a Primary Access system.



**Figure 115**

### 6.3.2 MUNICH32 in Systems with MITEL ST BUS

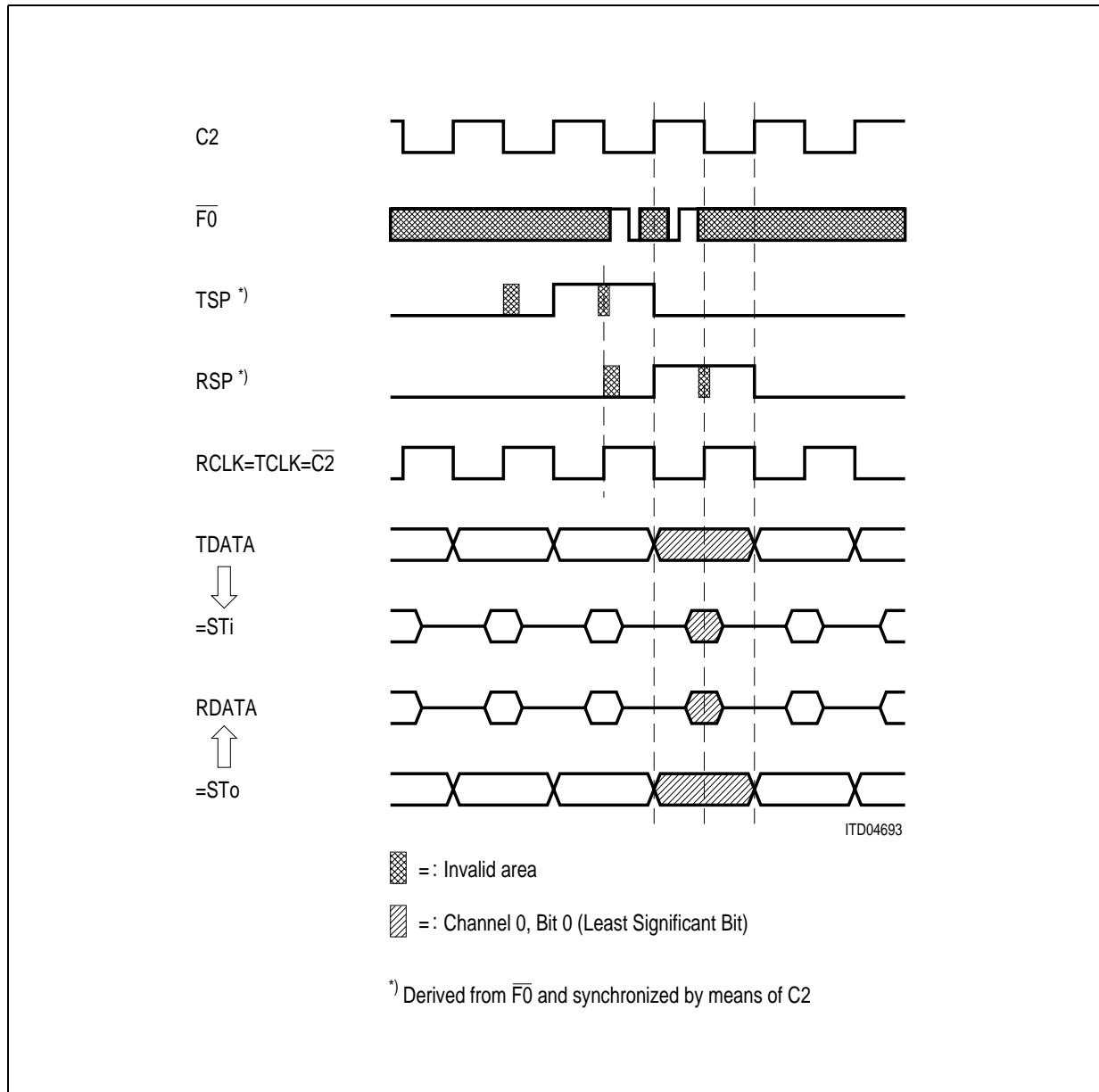
A few more effort is necessary to integrate the MUNICH32 into a ST BUS system from MITEL. The basic assumption made here is that the clock master is the ST BUS system. That means all signals derived from the ST BUS need to be adapted to match the MUNICH32 timing requirements. First of all the clock signal  $C2$  must be inverted before it can be used as the MUNICH32 clocks ( $TCLK = RCLK = \overline{C2}$ ). The next step is the generation of the TSP and RSP pulses out of the  $\overline{F0}$  signal, which is the ST BUS frame synchronization signal. The RSP pulse can be derived from the  $\overline{F0}$  signal by means of a simple D-Flip-Flop clocked with  $C2$ , as depicted in the following **Figure 116**. Due to the necessary phase relationship between the serial data streams and their corresponding TSP, RSP and  $\overline{F0}$  pulses, the effort to generate the TSP pulse is much higher than for RSP.



**Figure 116**

## Application Hints

The TSP pulse must be derived from the  $\overline{F0}$  signal with a phase shift by 255 clock cycles to be at the right position. The corresponding timing is illustrated in the following diagram.



**Figure 117**

## Electrical Characteristics

### 7 Electrical Characteristics

*Note: All specifications are for V3.4 unless otherwise specified. Version numbers are identified in the Interrupt Information bits VN(3:1):*

*these bits are*

- '0000' for version 1.1*
- '0001' for version 2.1*
- '0010' for version 2.2*
- '0100' for version 3.2*
- '0110' for version 3.4*

#### 7.1 Absolute Maximum Ratings

**Table 12**

Parameter	Symbol	Limit Values		Unit
		min.	max.	
Ambient temperature under bias: PEB PEF	$T_A$	0	70	°C
	$T_A$	– 40	85	
Storage temperature	$T_{stg}$	– 65	125	°C
Voltage at any pin with respect to ground	$V_S$	– 0.4	$V_{DD} + 0.4$	V

*Note: Stresses above those listed here may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

## Electrical Characteristics

### 7.2 DC Characteristics

**Table 13**
 $T_A = 0 \text{ to } +70 \text{ }^\circ\text{C}$ ;  $V_{DD} = 5 \text{ V} \pm 5\%$ ,  $V_{SS} = 0 \text{ V}$ 

Parameter		Symbol	Limit Values		Unit	Test Condition
			min.	max.		
L-input voltage		$V_{IL}$	-0.4	0.8	V	–
H-input voltage		$V_{IH}$	2.0	$V_{DD} + 0.4$	V	–
L-output voltage		$V_{QL}$	–	0.45	V	$I_{QL} = 7 \text{ mA}$ (pin TDATA) $I_{QL} = 2 \text{ mA}$ (all others)
H-output voltage		$V_{QH}$	$V_{DD} - 0.5$	–	V	$I_{QH} = -2 \text{ mA}$ (pin HOLD/BR) $I_{QH} = -100 \text{ } \mu\text{A}$ (all others)
H-output voltage		$V_{QH}$	2.4		V	$I_{QH} = -400 \text{ } \mu\text{A}$
Power supply current	operational	$I_{CC}$	–	< 100	mA	$V_{DD} = 5 \text{ V}$ inputs at 0 V/ $V_{DD}$ , no outputs loads
	power down (no clocks)	$I_{CC}$	–	< 2	mA	
Input leakage current		$I_{LI}$	–	10	$\mu\text{A}$	$0 \text{ V} < V_{IN} < V_{DD}$ to 0 V
Output leakage current		$I_{LQ}$				$0 \text{ V} < V_{OUT} < V_{DD}$ to 0 V

*Note: The listed characteristics are ensured over the operating range of the integrated circuit. Typical characteristics specify mean values expected over the production spread. If not otherwise specified, typical characteristics apply at  $T_A = 25 \text{ }^\circ\text{C}$  and the given supply voltage.*

## Electrical Characteristics

### 7.3 Capacitances

**Table 14**
 $T_A = 25\text{ }^{\circ}\text{C}; V_{DD} = 5\text{ V} \pm 5\%, V_{SS} = 0\text{ V}$ 

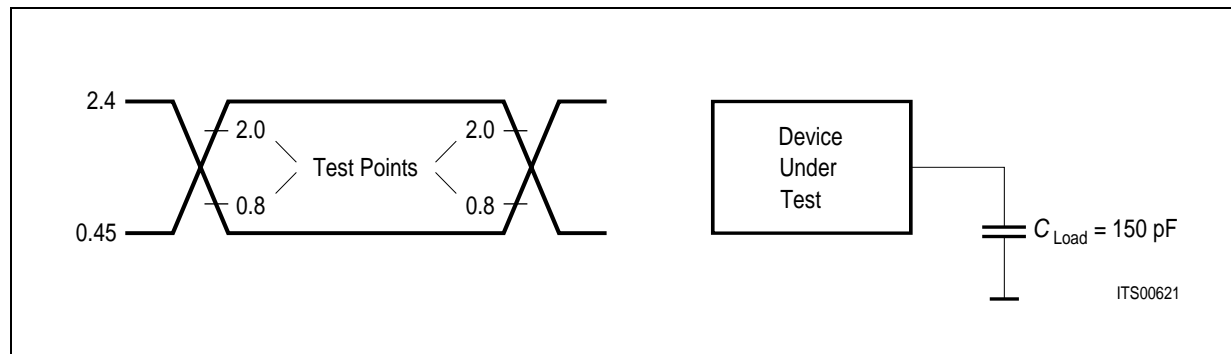
Parameter	Symbol	Limit Values		Unit	Test Condition
		min.	max.		
Input capacitance	$C_{IN}$	5	10	pF	–
Output capacitance	$C_{OUT}$	8	15	pF	–
I/O-capacitance	$C_{IO}$	10	20	pF	–

### 7.4 AC Characteristics

 $T_A = 0\text{ to }+70\text{ }^{\circ}\text{C}; V_{DD} = 5\text{ V} \pm 5\%$ 

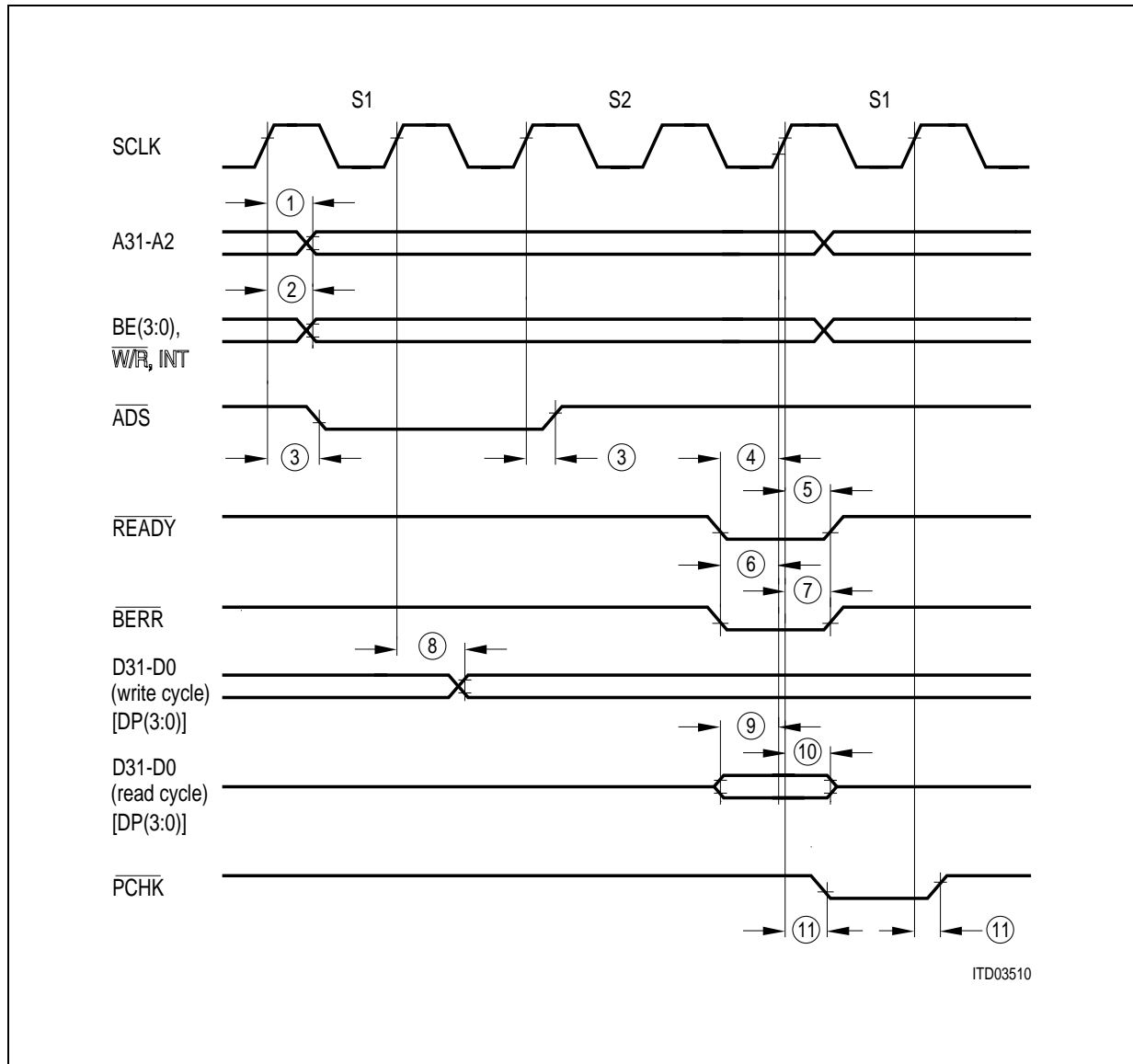
Inputs are driven to 2.4 V for a logical '1' and to 0.4 V for a logical '0'. Timing measurements are made at 2.0 V for a logical '1' and at 0.8 V for a logical '0'.

The AC testing input/output waveforms are shown below.



**Figure 118**  
**Input/Output Waveform for AC Tests**

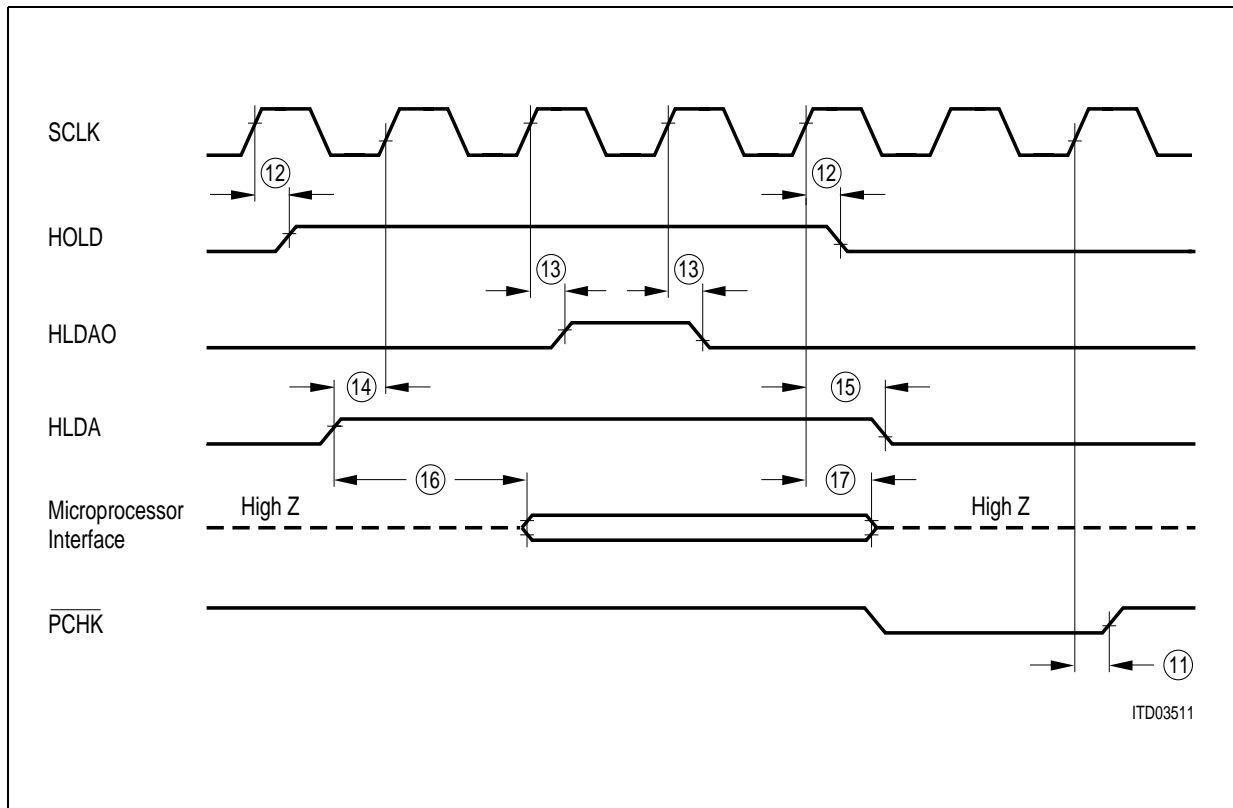
## 7.5 Microprocessor Interface Intel Bus Mode



**Figure 119**  
**Timing Diagram Intel Bus Mode**



## Electrical Characteristics



**Figure 120**  
**Bus Arbitration Timing Diagram Intel Bus Mode**

## Intel Bus Timing

**Table 15**

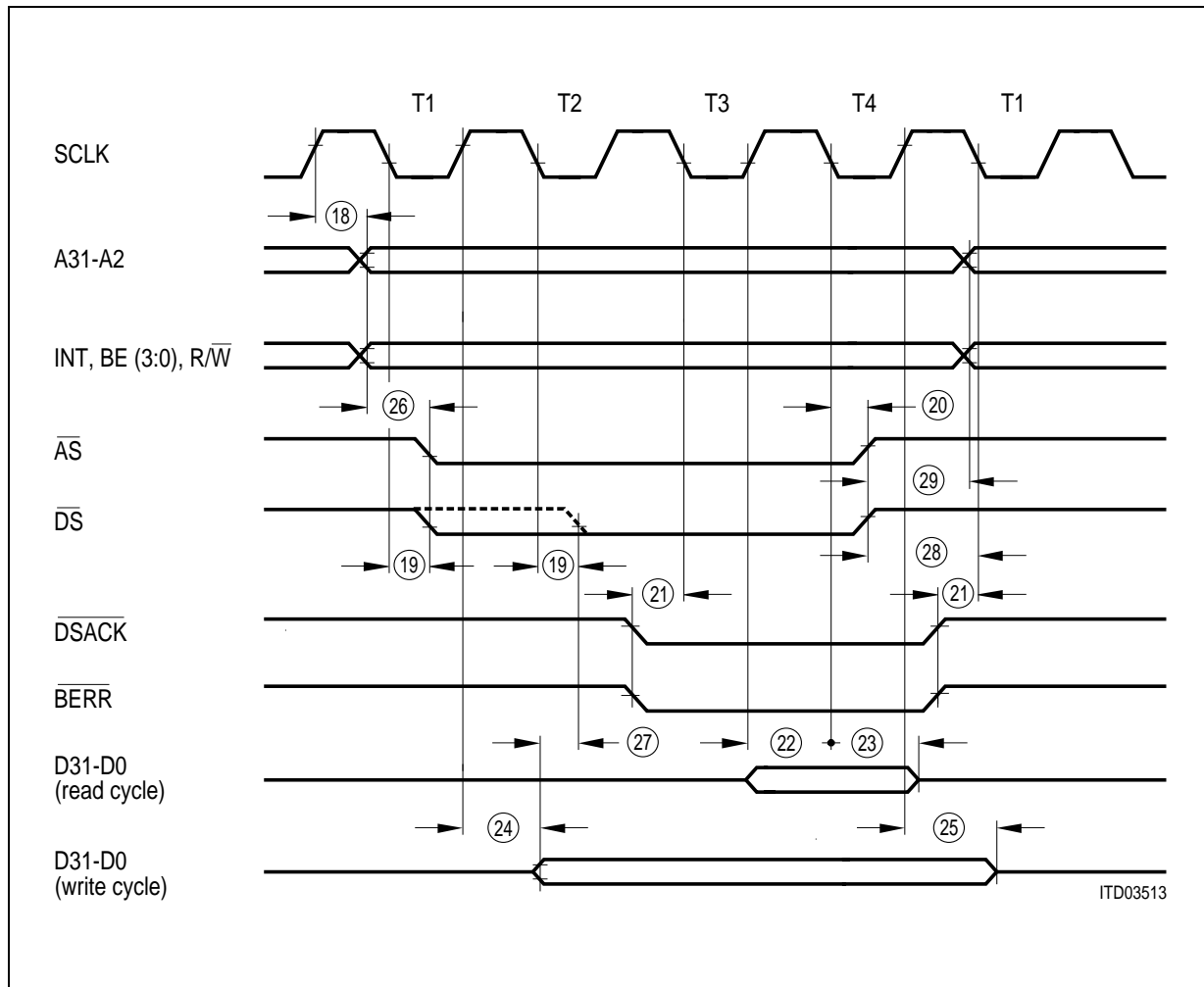
No.	Parameter	Limit Values		Unit
		min.	max.	
1	Address, valid delay	–	20	ns
2	BE, INT, W/R valid delay	–	20	ns
3	ADS valid delay	–	20	ns
4	READY setup time	10	–	ns
5	READY hold time	5	–	ns
6	BERR setup time	10	–	ns
7	BERR hold time	5	–	ns
8	Data valid delay (write)	–	35	ns
9	Data setup time (read)	5	–	ns
10	Data hold time (read)	8	–	ns

## Electrical Characteristics

**Table 15**

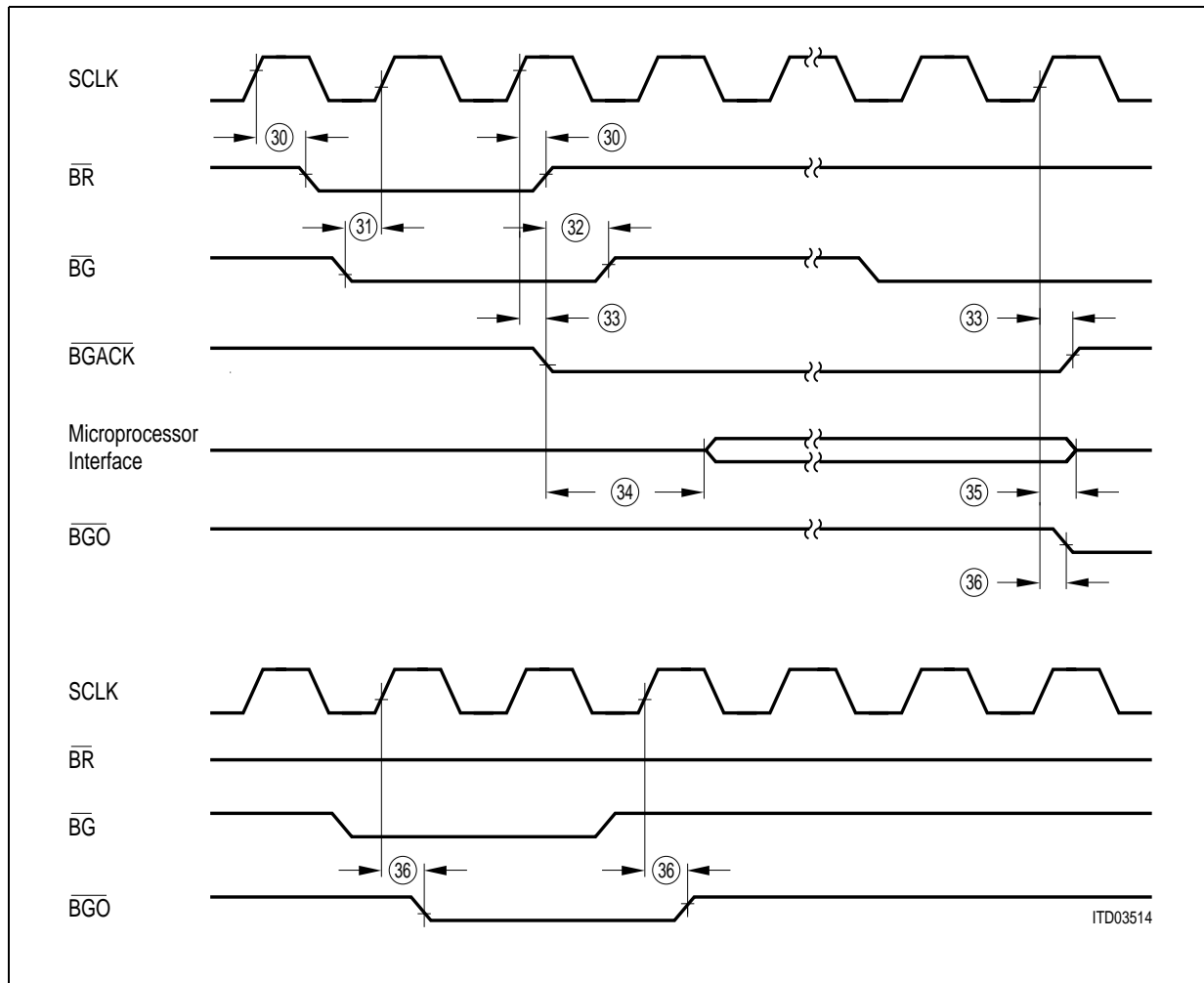
No.	Parameter	Limit Values		Unit
		min.	max.	
11	Parity check valid delay	–	50	ns
12	HOLD valid delay	–	20	ns
13	HLDAO valid delay	–	20	ns
14	HLDA setup time	10	–	ns
15	HLDA hold time	10	–	ns
16	Microprocessor Interface (MI) driven after HLDA set	2 SCLK cycles	–	–
17	MI tristated after bus accesses	–	40	ns

## 7.6 Microprocessor Interface Motorola Bus Mode



**Figure 121**  
**Timing Diagram Motorola Bus Mode**

## Electrical Characteristics



**Figure 122**  
**Bus Arbitration Timing Motorola Bus Mode**

### Motorola Bus Timing

**Table 16**

No.	Parameter	Limit Values		Unit
		min.	max.	
18	Address, BE, INT, R/W valid delay	—	20	ns
19	$\overline{AS}$ , $\overline{DS}$ asserted after clock low	—	20	ns
20	$\overline{AS}$ , $\overline{DS}$ negated after clock low	—	20	ns
21	$\overline{DSACK}$ , $\overline{BERR}$ setup time to clock low	5	—	ns
22	Data read setup time to clock low	5	—	ns
23	Data read hold time to clock low	8	—	ns

## Electrical Characteristics

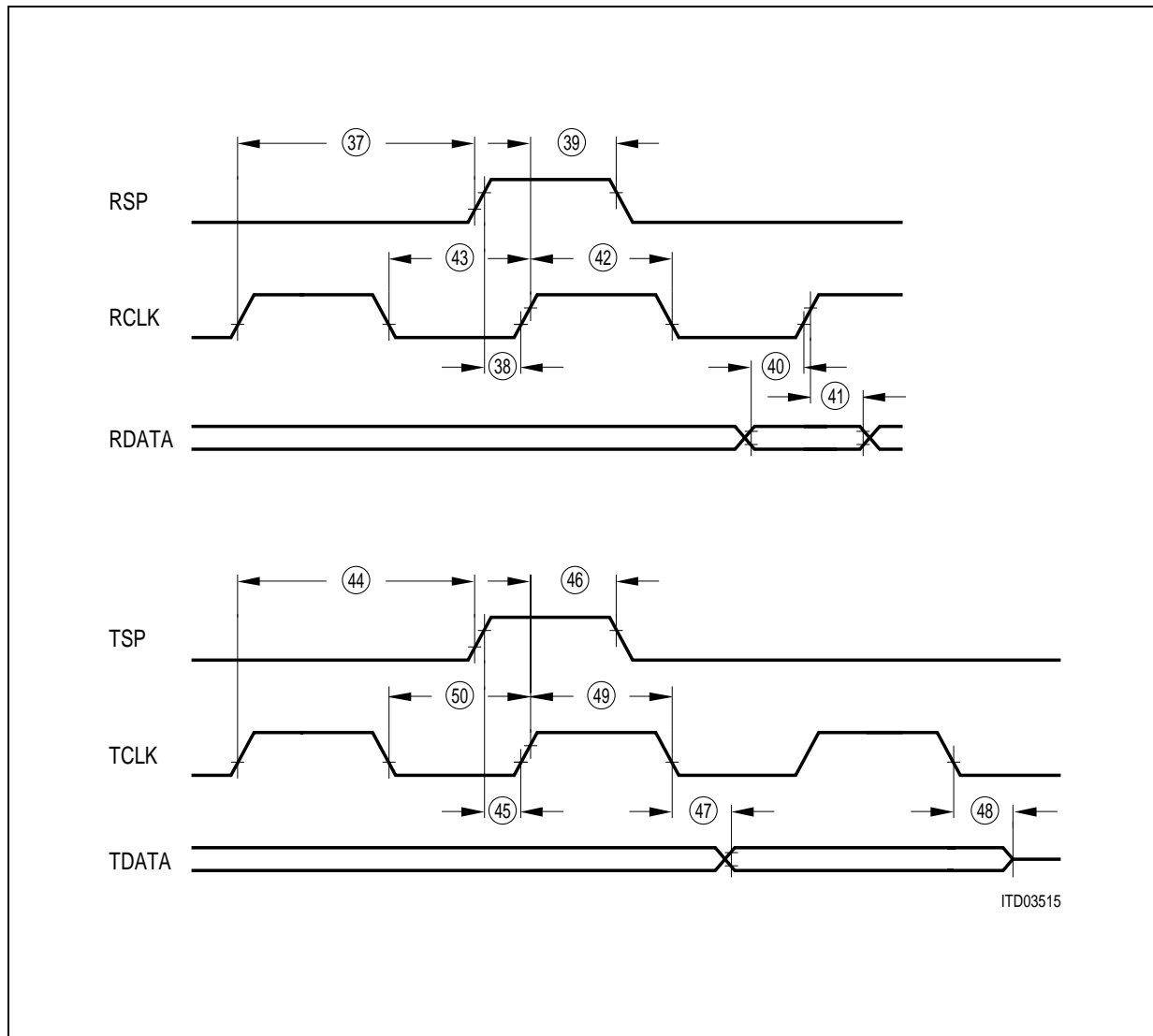
**Table 16**

No.	Parameter	Limit Values		Unit
		min.	max.	
24	Data write valid delay	–	35	ns
25	Data write hold from clock high	–	35	ns
26	Address valid to $\overline{AS}$ high	10	–	ns
27	Data valid to $\overline{DS}$ low	10	–	ns
28	$\overline{DS}$ high to data invalid	5	–	ns
29	$\overline{AS}$ high to address invalid	10	–	ns
30	$\overline{BR}$ valid delay	–	25	ns
31	$\overline{BG}$ setup time to clock high	5	–	ns
32	$\overline{BG}$ hold time after $\overline{BGACK}$	10	–	ns
33	$\overline{BGACK}$ valid delay	–	25	ns
34	Microprocessor Interface driven after $\overline{BGACK}$ asserted	1 SCLK cycle	–	–
35	Clock high to Microprocessor Interface tristated	–	40	ns
36	$\overline{BGO}$ valid delay from clock high	–	40	ns

<sup>1)</sup> Newly specified for V2.1 and V2.2. Not specified in Data Sheet 08.93.

## Electrical Characteristics

### Serial Interface Timing



**Figure 123**

**Table 17**

No.	Parameter	Limit Values		Unit
		min.	max.	
37	Receive strobe guard time	10	—	ns
38	Receive strobe setup	5	—	ns
39	Receive strobe hold	5	—	ns
40	Receive data setup	5	—	ns
41	Receive data hold	5	—	ns

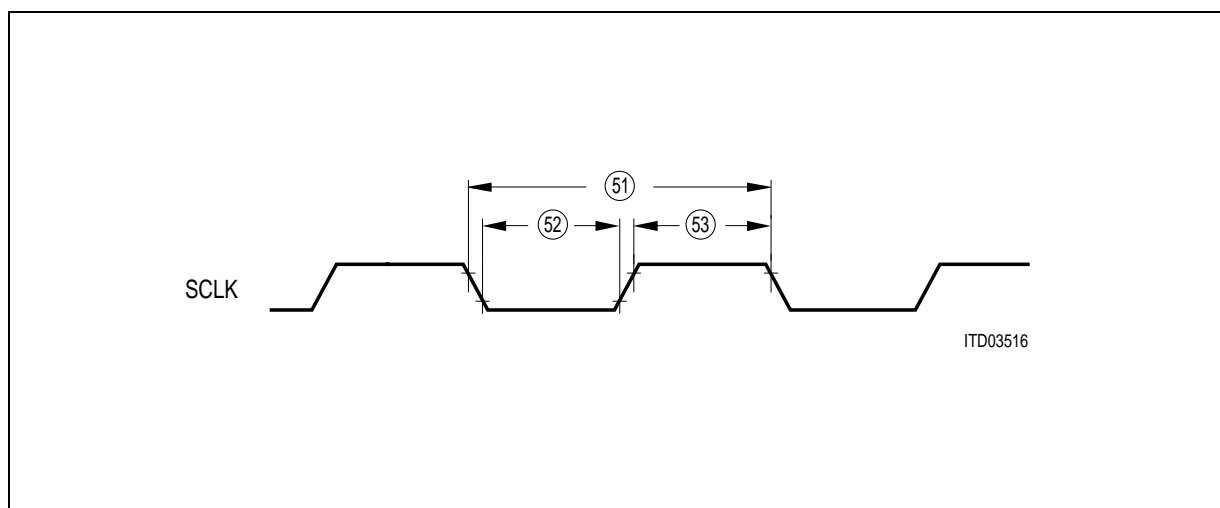
## Electrical Characteristics

**Table 17** (cont'd)

No.	Parameter	Limit Values		Unit
		min.	max.	
42	Receive clock high width	60	–	ns
43	Receive clock low width	60	–	ns
44	Transmit strobe guard time	20	–	ns
45	Transmit strobe setup	5	–	ns
46	Transmit strobe hold	5	–	ns
47	Transmit data delay	–	40	ns
48	Transmit clock to high impedance	–	50	ns
49	Transmit clock high width	60	–	ns
50	Transmit clock low width	60	–	ns

- Note:* 1. The frequency on the serial line **must** be smaller or equal to  $\frac{1}{8}^{th}$  of the frequency on the  $\mu P$  bus for 1.536 MHz, 1.544 MHz, 2.048 MHz  $\frac{1}{4}^{th}$  of the frequency on the  $\mu P$  bus for 4.096 MHz.
2. For complete internal or complete external loop  $t_{42}$  and  $t_{49}$  must be greater or equal to 3 times  $t_{51}$ .

## Clock Input Timing


**Figure 124**  
**Clock Timing**

## Electrical Characteristics

**Table 18**

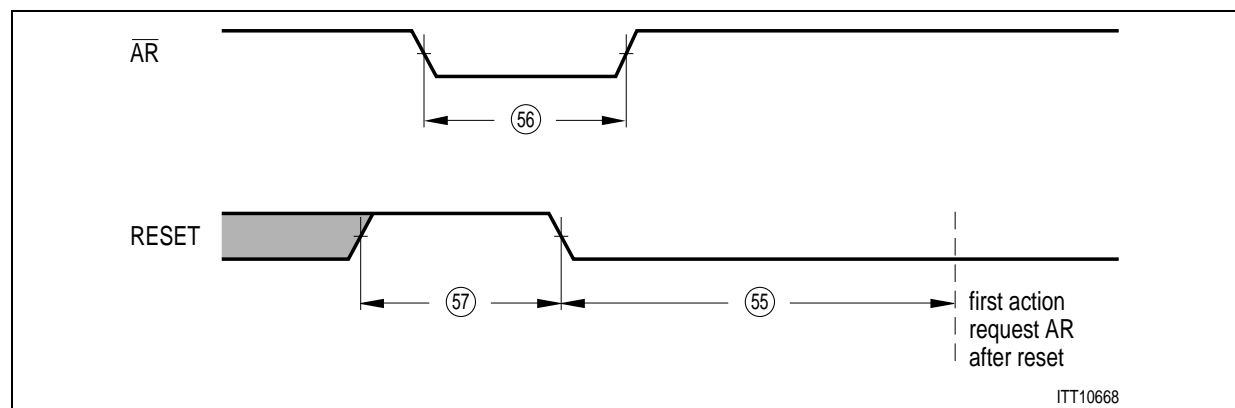
No.	Parameter	Limit Values		Unit
		min.	max	
51	Cycle period	50	–	ns
52	Clock low time	25	–	ns
53	Clock high time	25	–	ns

Note: If  $f_T$  is the frequency of the clock TCLK,  $f_R$  the frequency of the clock RCLK and  $f_S$  the frequency of the clock SCLK the equations

$7.996 \times \max(f_T, f_R) \leq f_S \leq 16.667 \text{ MHz}$  for CEPT, T1, E1 PCM mode  
and

$3.998 \times \max(f_T, f_R) \leq f_S \leq 16.667 \text{ MHz}$  for 4.096 MHz PCM mode  
describe the allowed range of frequencies for  $f_S$ .

## System Interface Timing


**Figure 125**
**Table 19**

No.	Parameter	Limit Values		Unit
		min.	max.	
55	Reset to first action request delay	12 SCLK cycles	–	–
56	AR# pulse width	2 SCLK cycles	5 SCLK cycles	–
57	Reset pulse width	2 SCLK cycles	–	–

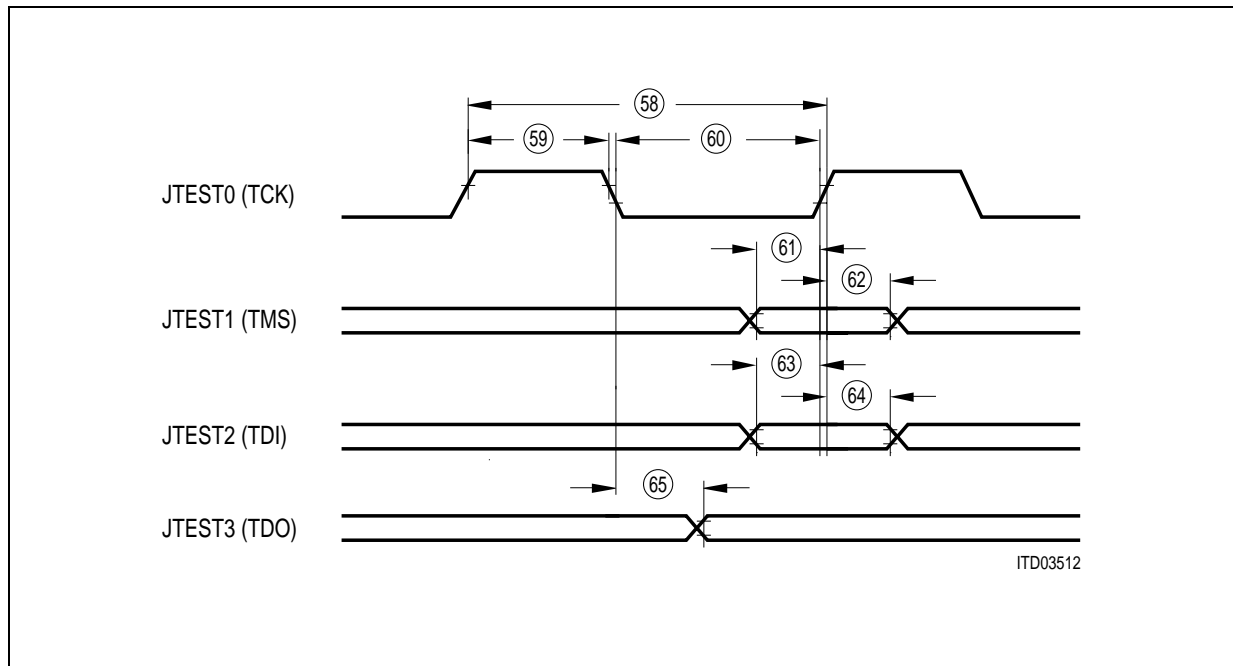
After power up a logical '1' at the reset pin of the MUNICH V3.4 sets the device into a reset state where the complete microprocessor bus interface is tristated and the internal reset sequence is started.



## Electrical Characteristics

The trailing edge of the reset starts the last part of the internal reset sequence and takes about 12 SCLK cycles. It is not allowed to give an action request (AR) during these first 12 SCLK cycles after the trailing edge of signal RESET.

### JTAG-Boundary Scan Timing



**Figure 126**  
**JTAG-Boundary Scan Timing**

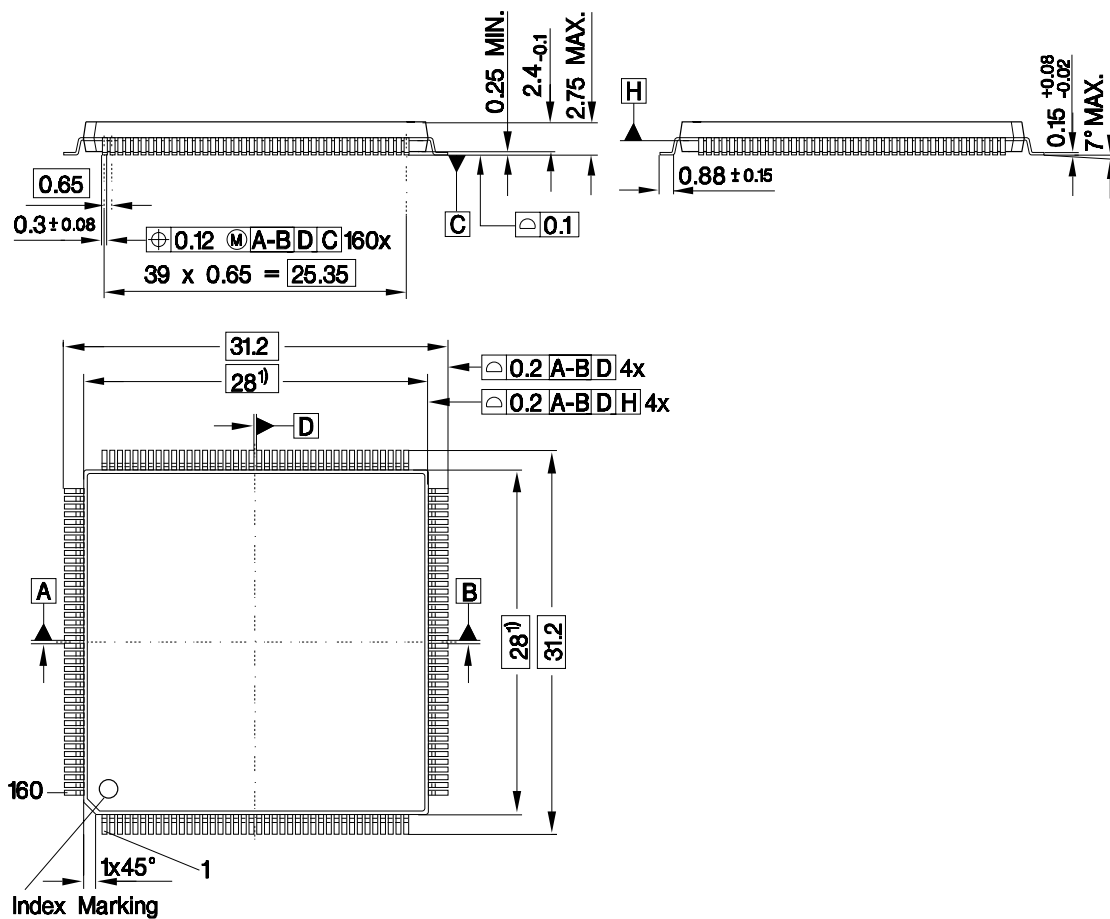
**Table 20**  
**Intel Bus Timing**

No.	Parameter	Limit Values		Unit
		min.	max.	
58	JTEST0 (TCK) period	166	inf	—
59	JTEST0 (TCK) high time	80	—	—
60	JTEST0 (TCK) low time	80	—	—
61	JTEST1 (TMS) setup time	15	—	—
62	JTEST1 (TMS) hold time	10	—	—
63	JTEST2 (TDI) setup time	15	—	—
64	JTEST2 (TDI) hold time	15	—	—
65	JTEST3 (TDO) valid delay	30	—	—

## 8 Package Outlines

### P-MQFP-160-1

(Plastic Metric Quad Flat Package)



1) Does not include plastic or metal protrusion of 0.25 max. per side

GPM05247

### Sorts of Packing

Package outlines for tubes, trays etc. are contained in our Data Book "Package Information".

SMD = Surface Mounted Device

Dimensions in mm

## 9 Appendix

### 9.1 Source Code Extract MUNICH32

The MUNICH32 code extract is taken from the low level device driver for the MUNICH32, which is written in 'C'. This extract gives you a brief impression how a MUNICH32 device driver could be programmed.

The munich control configuration (munichCtrlCfg) is a structure which consists of the following substructures:

Action Specification	actionSpec
Interrupt Queue Specification	intQueueSpec
Time-Slot Assignment	timeSlot[ ]
Channel Specification	channelSpec[ ]
Munich Receive Descriptor Pointer	currRcDescrAddr[ ]
Munich Transmit Descriptor Pointer	currTxDescrAddr[ ]

These substructures mainly consist of bit fields. The use of bit fields does not produce a speed optimized but a highly readable code, in our case to demonstrate the programming of the MUNICH32 very clearly.

The structures are directly memory mapped to the MUNICH32 structures and listed below.

In this short example we select the CEPT-32 PCM highway format and the HDLC mode. All time-slots are assigned to channel number 0. HDLC frames are send via channel0.

There are two functions:

`InitChannel0AndSendFirstFrame()`  
`TxHdlcFrame().`

The function `InitChannel0AndSendFirstFrame()` comprises the following initialization tasks:

- the MUNICH32 is configured for the CEPT32 channel format
- the interrupt queue is initialized and assigned
- each time-slot consists of 8 bit and all time-slots are assigned to channel 0
- the transmit outputs and the receive inputs are active
- here nine transmit buffers are assigned to channel0
- idle code flags.

---

**Appendix**

The second part of the function prepares the device to send the first HDLC frame:

- the linked list of frames to be send is registered
- in receive direction a linked list of 10 receive descriptors with 32 bytes data each is prepared and installed.
- the macro `MUNICH32_ACTION_REQUEST()` 'generates' an activation request pulse to the MUNICH32
- the device reads the initialization data and transmits the first transmit frame

The MUNICH32 then polls the hold bit of last transmit descriptor until this bit is cleared. If the hold bit is cleared the device sends the next data until it finds the next hold bit.

The function `TxHdlcFrame` connects the transmit descriptor of the next frame with the last transmit descriptor of the last send frame and clears the hold bit; the next frame is send.

## 9.2 Source Code

```
...
/*-----
- MUNICH32 Transmit Descriptor Structure -
-----*/

typedef struct      munichTxDescr
{
    unsigned                fnum : 11;
    unsigned                csm   : 1;
    unsigned                : 3;
    unsigned                v110  : 1;
    unsigned                no    : 13;
    unsigned                hi    : 1;
    unsigned                hold  : 1;
    unsigned                fe    : 1;
    WORD8                  _ptr  data;
    struct munichTxDescr   _ptr  next;
}
MUNICH_TRANSMIT_DESCRIPTOR;

typedef MUNICH_TRANSMIT_DESCRIPTOR _ptr MUNICH_TX_DESCR_PTR

/*-----
- MUNICH32 Receive Descriptor Structure -
-----*/

typedef struct      munichRcDescr
{
    unsigned                : 16;
    unsigned                no    : 13;
    unsigned                hi    : 1;
    unsigned                hold  : 1;
    unsigned                : 1;
    unsigned                : 8;
    unsigned                status : 8;
    unsigned                bno   : 13;
    unsigned                : 1;
    unsigned                c     : 1;
    unsigned                fe    : 1;
    WORD8                  _ptr  data;
    struct munichRcDescr   _ptr  next;
}
MUNICH_RECEIVE_DESCRIPTOR;
```

## Appendix

```
typedef MUNICH_RECEIVE_DESCRIPTOR _ptr MUNICH_RC_DESCR_PTR;
```

```
/*-----  
- MUNICH32 Structures  
-----  
*/
```

```
typedef struct  
{  
    unsigned channelNumber      : 5;  
    unsigned rt                 : 1;  
    unsigned                   : 2;  
    unsigned fo                 : 1;  
    unsigned err                : 1;  
    unsigned sf                 : 1;  
    unsigned ifc                : 1;  
    unsigned fi                 : 1;  
    unsigned hi                 : 1;  
    unsigned arf                : 1;  
    unsigned arack              : 1;  
    unsigned x                  : 1;  
    unsigned sa                 : 1;  
    unsigned sb                 : 1;  
    unsigned e1                 : 1;  
    unsigned e2                 : 1;  
    unsigned e3                 : 1;  
    unsigned e4                 : 1;  
    unsigned e5                 : 1;  
    unsigned e6                 : 1;  
    unsigned e7                 : 1;  
    unsigned frc                : 1;  
    unsigned                   : 4;  
    unsigned intFlag            : 1;  
}
```

```
MUNICH32_INTERRUPT_QUEUE;
```

```
typedef struct  
{  
    MUNICH32_INTERRUPT_QUEUE _ptr addr;  
    unsigned                  n : 8;  
    unsigned                   : 24;  
}
```

```
INTERRUPT_QUEUE_SPECIFICATION;
```

```
typedef struct  
{  
    unsigned rcFillMask      : 8;  
    unsigned rcChannelNumber : 5;
```

## Appendix

```

    unsigned    rti            : 1;
    unsigned    : 2;
    unsigned    txFillMask    : 8;
    unsigned    txChannelNumber : 5;
    unsigned    tti            : 1;
    unsigned    : 2;
}
TIME_SLOT_ASSIGNMENT;

typedef struct
{
    unsigned    iftf          : 1;
    unsigned    mode          : 2;
    unsigned    fa            : 1;
    unsigned    trv           : 2;
    unsigned    crc           : 1;
    unsigned    inv           : 1;
    unsigned    tflagCs       : 1;
    unsigned    tflag         : 7;
    unsigned    ra            : 1;
    unsigned    ro            : 1;
    unsigned    th            : 1;
    unsigned    ta            : 1;
    unsigned    to            : 1;
    unsigned    ti            : 1;
    unsigned    ri            : 1;
    unsigned    nitbs         : 1;
    unsigned    intMask       : 8;
    MUNICH_RC_DESCR_PTR      frda;
    MUNICH_TX_DESCR_PTR      ftdda;
    unsigned    itbs          : 6;
    unsigned    : 26;
}
CHANNEL_SPECIFICATION;

typedef struct
{
    WORD32                                *currentReceiveDescriptorAddrCh;
}
CURRENT_RC_DESCR_ADDR;

typedef struct
{
    WORD32                                *currentTransmitDescriptorAddrCh;
}
CURRENT_TX_DESCR_ADDR;

```

## Appendix

```
typedef struct
{
    unsigned                : 2;
    unsigned    ia          : 1;
    unsigned    loopi       : 1;
    unsigned    loop        : 1;
    unsigned    loc         : 1;
    unsigned    res         : 1;
    unsigned    im          : 1;
    unsigned    channelNumber : 5;
    unsigned                : 1;
    unsigned    ico         : 1;
    unsigned    in          : 1;
    unsigned    mfl         : 13;
    unsigned    pcm         : 3;
}
ACTION_SPECIFICATION;

/*-----
- MUNICH32 Control Block -
-----
*/

typedef struct
{
    ACTION_SPECIFICATION    actionSpec;
    INTERRUPT_QUEUE_SPECIFICATION    intQueueSpec;
    TIME_SLOT_ASSIGNMENT    timeSlot    32;
    CHANNEL_SPECIFICATION    channelSpec    32;
    MUNICH_RC_DESCR_PTR    currRcDescrAddr    32;
    MUNICH_TX_DESCR_PTR    currTxDescrAddr    32;
}
MUNICH32_CTRL_CFG_SECTION;
..
..
```



## Appendix

```

/*-----
- Function      : InitChannel0AndSendFirstFrame
-
- Description   : Initialization of channel 0.
-                 - PCM Highway format CEPT 32-channel
-                 - HDLC Mode
-                 - All timeslots are assigned to channel 0.
-                 - Send the first HDLC frame
-----*/

static void InitChannel0AndSendFirstFrame ( MUNICH_TX_DESCR_PTR m32TxDescr )
{
    ..
    ..
    -----
*/

    txDescr = m32TxDescr          /* store transmit descriptor pointer */

    /*== Action Specification =====*/

    munichCtrlCfg.actionSpec.in      = 1; /* initialization procedure */
    munichCtrlCfg.actionSpec.ico     = 0; /* initialize channel only */
    munichCtrlCfg.actionSpec.channelNumber = 0; /* - */
    munichCtrlCfg.actionSpec.im      = 0; /* interrupt mask */
    munichCtrlCfg.actionSpec.res      = 0; /* reset */
    munichCtrlCfg.actionSpec.loopi    = 0; /* loops for test purposes */
    munichCtrlCfg.actionSpec.loop     = 0; /* loops for test purposes */
    munichCtrlCfg.actionSpec.loc      = 0; /* loops for test purposes */
    munichCtrlCfg.actionSpec.ia       = 1; /* interrupt attention */
    munichCtrlCfg.actionSpec.pcm      = 5; /* PCM, CEPT 32 channel */
    munichCtrlCfg.actionSpec.mfl      = 256; /* maximum frame length */

    /*== Interrupt Queue Specification =====*/

                                /* interrupt queue address */
    munichCtrlCfg.intQueueSpec.addr  = &munichIntQueue [0];
                                /* interrupt queue size */
    munichCtrlCfg.intQueueSpec.n     = (INT_QUEUE_SIZE_MAX / 16 -1);

    for ( i = 0; i < INT_QUEUE_SIZE_MAX; i++ ) /* Reset interrupt queue */
    {
        munichIntQueue[i].intFlag = CLEAR;
    }

```

## Appendix

```

/*== Timeslot Assignment =====*/

for ( i = 0; i < 32; i++)
{
    munichCtrlCfg.timeSlot[i].rcChannelNumber = 0; /* assigned to */
    munichCtrlCfg.timeSlot[i].txChannelNumber = 0; /* channel 0 */
    munichCtrlCfg.timeSlot[i].rcFillMask      = 0xFF; /* all bits assigned */
    munichCtrlCfg.timeSlot[i].txFillMask      = 0xFF; /* per channel */
    munichCtrlCfg.timeSlot[i].tti             = 0; /* Tx output active */
    munichCtrlCfg.timeSlot[i].rti             = 0; /* Rc input active */
}

/*== Channel Specification =====*/

munichCtrlCfg.channelSpec[channel0].intMask = 0; /* interrupts enabled */
munichCtrlCfg.channelSpec[channel0].nitbs   = 1; /* new ITBS value */
munichCtrlCfg.channelSpec[channel0].to      = 0; /* transmit */
munichCtrlCfg.channelSpec[channel0].ta      = 1; /* initialization */
munichCtrlCfg.channelSpec[channel0].ti      = 1; /* */
munichCtrlCfg.channelSpec[channel0].ro      = 0; /* receive */
munichCtrlCfg.channelSpec[channel0].ra      = 1; /* initialization */
munichCtrlCfg.channelSpec[channel0].ri      = 1; /* */

munichCtrlCfg.channelSpec[channel0].th      = 0; /* no transmit hold */
munichCtrlCfg.channelSpec[channel0].fa      = 0; /* no flag adjustment */
munichCtrlCfg.channelSpec[channel0].tflag   = 0; /* only for TMA */
munichCtrlCfg.channelSpec[channel0].tflagCs = 0; /* CRC select */
munichCtrlCfg.channelSpec[channel0].inv     = 0; /* no bit inversion */
munichCtrlCfg.channelSpec[channel0].crc     = 0; /* 16-bit CRC */
munichCtrlCfg.channelSpec[channel0].trv     = 0; /* transmission rate */
munichCtrlCfg.channelSpec[channel0].mode    = 3; /* HDLC Mode */
munichCtrlCfg.channelSpec[channel0].iftf    = 0; /* idle code flags */
munichCtrlCfg.channelSpec[channel0].itbs    = 9; /* transmit buffer size */

munichCtrlCfg.channelSpec[channel0].ftda    = txDescr; /* first transmit */
                                           /* descriptor address */

```

## Appendix

```

/*== Transmit Descriptor =====*/

/* the next pointer of the last txDescr points to the zero pointer */

for ( ; txDescr ->next; txDescr = txDescr ->next )
{
    txDescr ->fnum = 3;          /* 3 interframe timefill char */
    txDescr ->hold = 0;          /* clear hold bit */
    txDescr ->hi = 0;            /* clear host initiated interrupt bit */
    txDescr ->fe = 0;            /* clear frame end bit */
    txDescr ->v110 = 0;          /* clear v110 bit */
}
txDescr ->fe = 1;                /* set frame end bit */
txDescr ->hold = 1;              /* set hold bit */

/*== Receive Descriptor =====*/

rcDescr = AllocReceiveDescriptor(10);          /* Alloc e.g. ten */
                                                /* receive descriptors */
                                                /* with 32 data byte each */

munichCtrlCfg.channelSpec[channel0].frda = rcDescr; /* first receive */
                                                /* descriptor address */

/*== Prepare Receive Descriptor 1 to 9 =====*/

for ( ; rcDescr ->next; rcDescr = rcDescr ->next )
{
    rcDescr ->hold = 0;          /* not the last descriptor */
    rcDescr ->hi = 0;            /* no host interrupt */
    rcDescr ->no = 32;           /* 32 data byte available */
    rcDescr ->fe = 0;            /* clear frame end bit */
    rcDescr ->c = 0;             /* clear data section complete bit */
}

/*== Prepare The Last Receive Descriptor, Number 10 =====*/

rcDescr ->hold = 1;              /* last available descriptor */
rcDescr ->hi = 1;                /* no host interrupt */
rcDescr ->no = 32;               /* 32 data byte available */
rcDescr ->fe = 0;                /* clear frame end bit */
rcDescr ->c = 0;                 /* clear data section complete bit */

channelControl[0].lasttxdescr = txDescr; /* store last transmit pointer */
channelControl[0].lastrcdscr = rcDescr; /* store last receive pointer */

MUNICH32_ACTION_REQUEST ();        /* generate MUNICH32 activation request */
}

```

## Appendix

```

/*-----
- Function      :   TxHdlcFrame                                     -
-----
- Description   :   Transmit an HDLC frame via channel 0           -
-----
*/

static void      TxHdlcFrame ( MUNICH_TX_DESCR_PTR m32TxDescr )
{
    ..
    ..
}

/*-----
*/

m32TxDescr = txDescr;                /* store transmit descriptor pointer */

channelControl[0].lasttxdescr ->next = txDescr; /* Add frame to existing */
                                           /* channel0 frame queue */

/*=== Transmit Descriptor =====*/

for ( ; txDescr ->next; txDescr = txDescr ->next )
{
    txDescr ->fnum     = 3;          /* 3 interframe timefill char */
    txDescr ->hold     = 0;          /* clear hold bit */
    txDescr ->hi       = 0;          /* clear host initiated interrupt bit */
    txDescr ->fe       = 0;          /* clear frame end bit */
    txDescr ->vll0     = 0;          /* clear vll0 bit */
}

txDescr ->fe          = 1;          /* set frame end bit */
txDescr ->hold        = 1;          /* set hold bit */

channelControl[0].lasttxdescr ->hold = 0;    /* the polling MUNICH32 */
                                              /* will then detect the */
                                              /* cleared hold bit and */
                                              /* send the following */
                                              /* frame */

channelControl[0].lasttxdescr = txDescr; /* store last transmit pointer */
}

```