Table of Contents

1.0 Introduction to the Point to Point Interface (PPI) Communication Protocol

This document defines the master/slave serial communication interface which is called Point to Point Interface (PPI). The communication protocol is based upon the PROFIBUS DIN Standard 19 245 using the seven layer OSI model. The PPI protocol will implement layers 1, 2, and 7. The remaining layers (layer 3, 4, 5, and 6) are null layers in the PPI implementation.

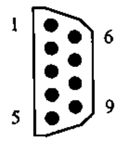
There are two basic types of PPI devices, masters and slaves. Master devices initiate all messages and hence must hold the token in order to initiate messages. A master device that does not currently hold the token may optionally be designed to respond to a PPI request as a slave. Slave devices are not token holders and therefore can never initiate a message.

2.0 Layer 1 PPI Definition

The electrical interface shall be provided through a nine pin D female style connector. The electrical signaling levels shall be RS485. The pinout of the nine pin D connector shall be as shown below.

Every PPI device shall provide a small internal bias on pins 3 and 8 of the nine pin D connector to prevent the line from chattering when the port is not being used. External bias and termination shall be provided by the user when cable lengths greater than 5 meters are required. The source for the external bias shall be provided by the PPI device on pin 6 of the D connector.

9 Pin D Female Connector
(Front View)



Pin #	Description
1	No connection
2	24V Return or Ground
3	RxD/TxD+
4	Request to Send (TTL Level) (Optional)
5	Ground (Logic or Communication Ground)
6	+5V (with 100 ohm series resistor)
7	Reserved for 24V power
8	RxD/TxD-
9	Reserved for test input (Optional)
Connector Shell tied to Logic Ground	

9 Pin Sub D Connector Pin Out

Each character of the PPI message shall have one start bit, eight data bits, an even parity bit, and one stop bit. No PPI device shall be required to detect and/or reject messages that are received with gaps between characters.

3.0 Layer 2 PPI Definition

After a TRDY time, PPI slave devices shall acknowledge, either negatively or positively, all incoming requests within a maximum station delay (TSDR). PPI devices shall detect a TSYN time and use this idle line time period to delineate the beginning of a new message. Once a TSYN time has been detected, the PPI devices shall copy the next message up to the destination address field. If the destination address field indicates that the message is for this station, the entire message will be copied and the appropriate response will be generated. If the destination address field indicates that the message is not for this station, the device shall discontinue copying the message and enter the state that searches for the next TSYN.

PPI devices are intended for point to point operation and shall require address initialization before being connected in a multi-drop configuration.

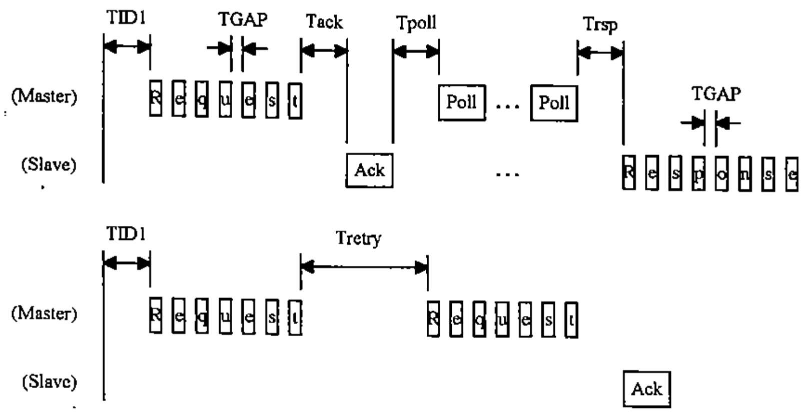
PPI devices do not support the address extensions defined in the PROFIBUS standard. Therefore, all requests are processed by way of the default service access point (SAP).

Network management including token initialization, recovery of a lost token and elimination of duplicate tokens shall be processed in accordance with the PROFIBUS standard.

Since all messages must be acknowledged within a station delay, the PPI slave will not be able to process the incoming message and generate the reply within the station delay time. Therefore, the requesting station shall be compelled to poll for the response after the request has been acknowledged. Each poll for the response shall be acknowledged and once a request has been processed and the response is ready, the PPI slave shall transmit the response in answer to the next poll.

The message and retry timing requirements are shown below.

Timing for PROFIBUS Compatibility



	Minimum	Maximum
TID1 (Idle time 1)	TSYN + TSM	TSYN + TSM or TSDI (which ever is larger)
TSYN (Synchronization Time)	33 bit times	33 bit times
TSM (Saftey Margin)	2 + 2(TSET) + TQUI	2 + 2(TSET) + TQUI
TSDI (Station Delay of Initiator)	22 bit times	33 bit times
TQUI (Quiet Line Time)	0 bit times	5 bit times
TSET(Set Up Time)	2 bit times	2 bit times
TRDY (Station Ready Time)	20 bit times	20 bit times
TSDR (Station Delay Time)	22 bit times	40 bit times
TGAP (Time Gap Between Characters)	0 bit times	0 bit times
Tack	22 bit times	30 bit times
Tpoll	TID1	less than 10 seconds
Trsp	TSDR	TSDR
Tretry	TSL	not defined
TSL (Slot Time)	not defined	2(TTD + TSM) + TSYN + 11 bit times
*TTD (Transmission Delay Time)	17 bit times	not defined

*Transmission Delay Time is a variable that depends upon cable length and number of repeaters. Using the minimum value of TTD, the slot time (TSL) is 100 bit times.

The following communication parameters shall be used by all PPI devices as default parameters.

Highest Station Address (HSA)	= 126
Re-try count	= 3
Gap Update Factor (GUF)	= 1

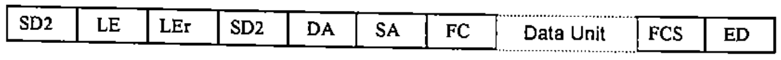
PPI slaves shall accept the following layer 2 message types:

1) SD2 frame for request data transfers (SRD low priority only)
2) SD1 frame for FDL_STATUS request
3) SD1 frame for polling
4) SD4 token message (the slave does not accept the token but recognizes the token message)

PPI masters shall accept the following layer 2 message types:

1) SD2 frame for response data transfers
2) SD1 frame for FDL_STATUS response
3) SD1 frame for negative acknowledge (RR for no resource and RS for no service activated)
4) SC frame for single character acknowledge with no data
5) SD4 token message

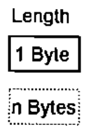
The format of each of these frame types is shown in the following diagrams.



SD2	- Start Delimiter 2 (068H)
LE	- Length Byte
LEr	- Length Byte repeated
DA	- Destination Address
SA	- Source Address
FC	- Frame Control
Data Unit	- Message Data and Control
FCS	- Frame Check Sequence
ED	- End Delimiter (016H)

FC Byte Values (Request)
SRD - 6C (first message cycle)
- 5C (alternating FCB)
- 7C (alternating FCB)

FC Byte Values (Response)
DL - 08



SD2 Message Format

Note that the FCS(Frame Check Sequence) is the bytewise addition(8 bits with no Carrry) of the message bytes starting with the DA and ending with the last byte of the data unit. LE is data unit length + 3(DA/SA/FC).

SD1	DA	SA	FC	FCS	ED

Length

1 Byte

SD1 - Start Delimiter 1 (010H)
DA - Destination Address
SA - Source Address
FC - Frame Control
FCS - Frame Check Sequence
= (DA+SA+FC)
ED - End Delimiter (016H)

FC Byte Values (Request)
- 5C, Poll (alternating FCB)
- 7C, Poll (alternating FCB)
- 49, FDL_STATUS

FC Byte Values (Response)
- 02, NAK (no resource, RR)
- 03, NAK (no service activated, RS)

For NAK, logical or RR/RS with FDL_STATUS:
- 00, FDL_STATUS (slave station)
- 10, FDL_STATUS (master station, not ready)
- 20, FDL_STATUS (master, ready to enter ring)
- 30, FLD_STATUS (master, already in ring)

NOTE: A Master which receives an SD2 frame but does not want to initiate an action should respond with an SD1 NAK(RS).

SD1 Message Format

SD4	DA	SA

Length

1 Byte

SD4 - Start Delimiter 4 (0DCH)
DA - Destination Address
SA - Source Address

SD4 Message Format

SC

Length

1 Byte

SC - Short Acknowledge (0E5H)

SC Single Character Acknowledge

3.1 Valid Request/Response Message Pairs and Exceptions

The following paragraphs provide general information about the handling of errors and re-tries.

The FC byte in the SD1 and SD2 frames is used to select the SRD (Send and Request Data) low priority message service and to control retries. When a station first attempts to initialize or restart communication with a slave, the requesting station shall set the value of the FC byte to 06CH in the first SD2 frame.

For subsequent polls and requests the value of the FC byte shall alternate from 05CH to 07CH. In the event that the response frame from the slave is received with an error or is not received at all, the master may re-try the message. A re-try shall be identified by repeating the message without changing the value of the FC byte. Likewise, when the slave recognizes a token pass after returning the SD2 response or receives a frame where the FC byte value has been changed from either 05CH to 07CH or from 07CH to 05CH, the slave can discard the previous data knowing that a retry for that data will not be required. The slave shall not detect protocol violations where the value of the FC byte does not alternate from one request to the next. Instead the slave shall respond appropriately to each request.

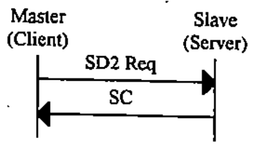
Messages with FC=06CH and FDL_STATUS may be repeated by the master, but there is no way for the slave to know if the message is being re-tried. The slave shall respond appropriately in these situations.

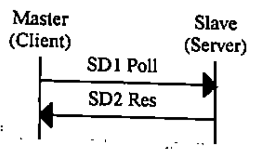
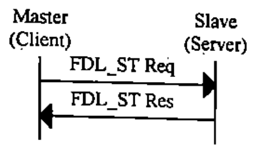
The FDL_Status request shall receive an immediate response containing the FDL_Status of the addressed unit. All SD2 requests will be acknowledged immediately with the SC acknowledge or a SD1 frame negative acknowledge (NAK). SD1 frame polls will be acknowledged with a SC acknowledge until the SD2 response frame is ready. When the SD2 response frame is ready, it shall be transmitted in response to the next poll. Checksum and/or parity errors shall cause the slave to discard the message without making any type of response. The slave device shall immediately begin monitoring the communication line for a *Tsync* time after detection of an error.

If the master detects an error in the slaves response, it may re-try the request. If after sending the prescribed number of retries without success, the master shall terminate the attempted message sequence.

The tables that follow describe the process by which the master and slave stations must exchange information. In all cases the issuance of a request, a poll or a response (in the case of the load sequence) by the master grants the slave the right to issue one message. The master shall not issue another request of any kind until the slave has responded or has been timed out by the master. Therefore, communication between the master and slave shall always be performed in message pairs that shall not be interrupted by other messages.

In between the message pairs exchanged by the master and the slave, the master is free to pass the token or issue requests to the same slave or to other stations. The master should be cognizant of the capabilities of the slave and should never issue requests that cannot be accepted by the slave.

Transaction	Description
	The master (token holder) issues a request to a slave. Upon receipt of a valid frame (no errors) that can be accepted, the slave acknowledges receipt of the frame by responding with a short character acknowledge. After receipt of the SC acknowledge, the master is responsible for polling for the response. After accepting the request, the slave shall process the request and generate an appropriate response. Upon receipt of the first poll from the requesting station after the response is available, the slave shall return the response. Exceptions: 1. The slave shall recognize retries of the same request and invoke the generation of a single response. 2. Once the request has been accepted, the slave shall start a message sequence timeout timer. If the requestor does not poll for the response within this timeout time(*currently 10 seconds for the S7-200*) or waits longer than this timeout time between polls, the slave shall discard the request thereby terminating the message sequence. Note that token passes may occur between polling bursts. 3. If the slave does not respond to the request, the master may retry the request. If after sending the prescribed number of retries without success, the master shall terminate the attempted message sequence. 4. If the slave returns a no resource available (RR) or a no service activated (RS) negative acknowledge (NAK), the master shall terminate the attempted message sequence and try again in a subsequent token hold period. 5. If the slave returns an outstanding response (result of a previous request) in answer to this request, the master shall be responsible for discarding the response to the previous request. The master shall also be responsible for polling for the response to this request.

Transaction	Description
 Master (Client) Slave (Server) SD1 Poll SD2 Res	After the master has issued a request to a slave and the slave has responded with an SC accepting the request, the master shall poll for the response. The slave shall return the SD2 response that corresponds to the request for which the master is polling. Exceptions: 1. If there is no response available to be returned to the master, the slave shall return a SC. Even if there is no outstanding request from the master which issued the poll, the slave shall respond with SC. 2. Each poll from the master which issued the request for which the slave is preparing the response shall re-start the message sequence timeout timer. If this timer expires before the next poll for the response is received, the message sequence shall be terminated by the slave. 3. If the slave does not respond to the poll or if the master detects an error in the slave's response, the master may re-try the poll. If after sending the prescribed number of re-tries without success, the master shall terminate the attempted message sequence.
 Master (Client) Slave (Server) FDL_ST Req FDL_ST Res	When the master issues a FDL Status request to a slave, the slave shall respond with a FDL Status response. Exceptions: 1. If the slave does not respond to the request, the master may repeat the FDL Status request or not, as is appropriate.

3.2 Token Handling and Polling at Low and High Baud Rates

For optimal network performance the method of polling should be dependent upon the baud rate of the network. For baud rates of 9.6KB and 19.2KB the master should poll for the response during the same token hold period in which the request was issued. If the slave does not return the response after the prescribed number of polls, the master shall pass the token.

For all baud rates higher than 19.2KB the master should not poll for the response during the same token hold period in which the request was issued.

4.0 Layer 7 PPI Definition - Communication Services

The communication services for the S7-200 constitute the data unit of the SD2 frame described in section 3.0.

Layer 7 defines the message or PDU (Protocol Data Unit) that is exchanged between the communication partners. Two types of PDU's are defined:

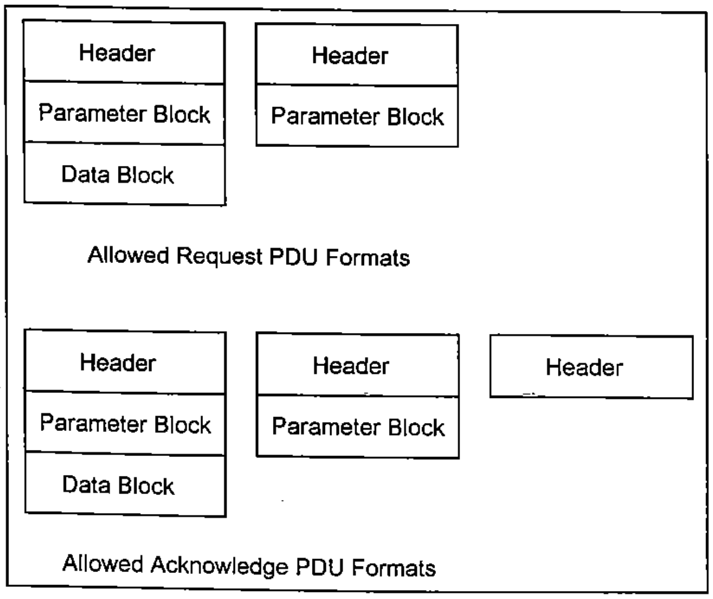
Request PDU's and
Acknowledgment PDU's

All PDU's have the same basic format which consists of three parts:

Header
Parameter block
Data block

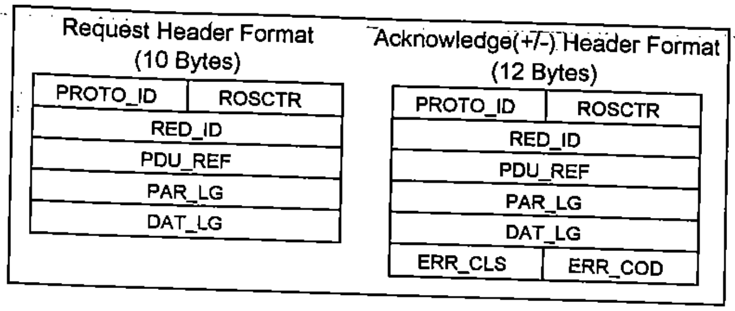
The header is required with every PDU and contains the control information and the lengths of both the parameter and data fields. The parameters vary from service to service and are explained with the description of the individual services.

Service request PDU's always include both the header and the parameter fields, and may or may not include the data field. Acknowledge PDU's sometimes omit both the parameter and data fields. The following PDU formats are allowed.



4.1 Header Definition Overview

The following diagram shows the format of request and acknowledge headers:



Parameter descriptions:

PROT_ID (UNSIGNED8) - Protocol Identification

32H for S7-200

ROSCTR (UNSIGNED8) - Remote Operating Services Control

Defined as follows:

01H - Acknowledged request
02H - Acknowledgement without the parameter and data fields
03H - Acknowledgement with either or both the parameter and data fields

RED_ID (UNSIGNED16) - Redundancy Identification

0000H - This parameter is intended for use with redundant concepts, and is not used at present.

PDU_REF (UNSIGNED16) - Protocol Data Unit Reference

Each request message is assigned a PDU_REF that is unique to the connection over which the request is transmitted. When the corresponding acknowledgment to this request is generated, it must include this same PDU_REF so that the response can be correlated with the request.

PAR_LG (UNSIGNED16) - Parameter Length

PAR_LG specifies the entire length of the parameter block in bytes.

DAT_LG (UNSIGNED16) - Data Length

DAT_LG specifies the entire length of the data block in bytes.

ERR_CLS (UNSIGNED8)/ERR_COD (UNSIGNED8) - Error Class/Error Code

Positive acknowledgments return a value of 00H in both fields while negative acknowledgments return a non-zero value in each field. These fields provide protocol or application specific error indications. The values for the error class field are given in the following table. The values for the error code field are given with the definition for the various request and response service formats.

ERR_CLS	Description of Protocol Specific Errors
00H	No error
81H	Error in the application ID of the request
82H	Error in the object definition (e.g. bad data type)
83H	No resources available
84H	Error in the structure of the service request
85H	Error in the communication equipment
87H	Access Error

ERR_CLS	Description of Application Specific Errors
D2H	OVS error
D4H	Diagnostic error
D6H	Protection system error
D8H	BuB error
EFH	Layer 2 specific error

Note: The unused error codes from D0H to EFH are reserved for future expansion.

4.2 Parameter Block Definition Overview

The construction of the parameter block is specified for each service request and response. The first parameter of the parameter block is always the service identifier and is found in all requests and positive acknowledgments (if the acknowledgment has a parameter block).

SERVICE_ID (UNSIGNED8) - Service Identification

The service identifications are generally classified as services with acknowledgment or services without acknowledgment. For those services which are acknowledged, the request and the response have the same service identification. The services in each group can be distinguished by the ROSCTR parameter in the header. The following table lists the various service ID's.

Service	SERVICE ID
Reading	04H
Writing	05H
General Services	
Virtual Device Status	00H
Application Associations Management	
Setting Up Application Associations	F0H

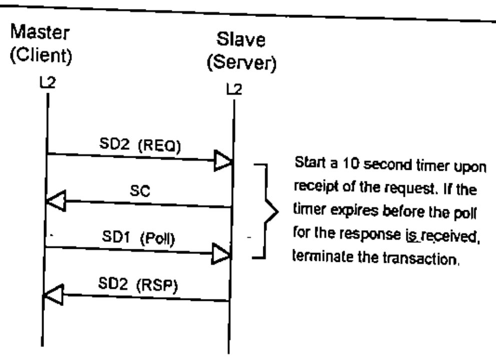
Note: There is no PDU for the termination of the applications association, instead the corresponding transport connection is broken (disconnected).

4.3 Data Block Definition Overview

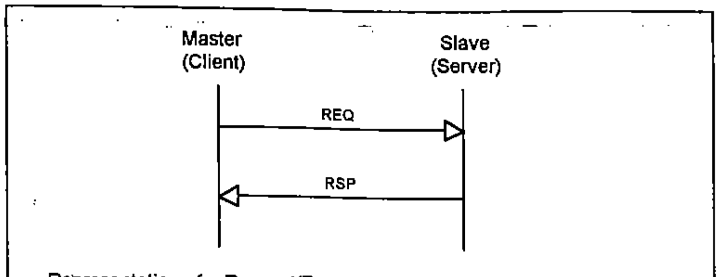
The format of the data block is specific to each service and is defined in the individual service descriptions.

4.4 Simplified Representation of the Communication Sequence

The following diagrams show the communication sequences and their representation in a simplified form that corresponds to the representations used in the description of each service. In general the master is the client and the slave is the server. Therefore, the request is issued by the master who is obliged to poll for the response.


Master
(Client)
L2
Slave
(Server)
L2
SD2 (REQ)
SC
SD1 (Poll)
SD2 (RSP)
Start a 10 second timer upon receipt of the request. If the timer expires before the poll for the response is received, terminate the transaction.

Actual Communication Request/Response Sequence Issued by the Master



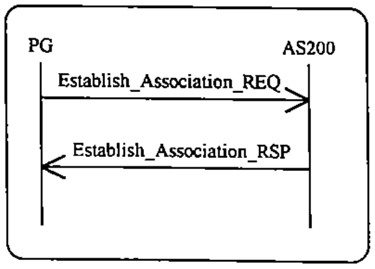
Representation of a Request/Response Sequence Issued by the Master

4.5 Establish an Association

The Establish Association service is sent in order to establish the Protocol Data Unit (PDU) size and the number of credits. The S7-200 supports a credit of 1. The number of credits available is the same as the number of outstanding requests/responses possible.

Once an association has been established, the remote system may send a keep awake message consisting of a SD1 poll that is acknowledged immediately with a SC. The S7200 does not require the keep awake message, but it will accept it and respond appropriately.

The following figures show the message sequence and format of the establish association request.



The master requests a number of credits of the slave and also suggests a PDU size. The slave will respond with the number of credits available to the master and the maximum PDU size supported by the slave.

Establish_Association_REQ

PROTO_ID=32H	ROSCTR=01H
RED_ID=0000H	
PDU_REF=xxxxH	
PAR_LG=0008H	
DAT_LG=0000H	
SERVICE_ID=F0H	Reserved
P1 (Max AmQ calling suggestion)	
P2 (Max AmQ called suggestion)	
P3 (Max PDU Size of caller)	

Establish_Association_RSP(+)

PROTO_ID=32H	ROSCTR=03H
RED_ID=0000H	
PDU_REF=xxxxH	
PAR_LG=0008H	
DAT_LG=0000H	
ERR_CLS=00H	ERR_COD=00H
SERVICE_ID=F0H	Reserved
P1 (Max AmQ calling suggestion)	
P2 (Max AmQ called suggestion)	
P3 (Max PDU Size of caller)	

Establish_Association_RSP(-)

PROTO_ID=32H	ROSCTR=03H
RED_ID=0000H	
PDU_REF=xxxxH	
PAR_LG=0001H	
DAT_LG=0000H	
ERR_CLS=xxH	ERR_COD=xxH
SERVICE_ID=F0H	

Request Parameter Block:

P1 (Max AmQ calling suggestion)
This parameter identifies the maximum number of outstanding requests w/responses that the calling partner can accept and process simultaneously.

P2 (Max AmQ called suggestion)
This parameter identifies the maximum number of outstanding requests w/responses that the calling partner will send before it waits for pending acknowledgments.

P3 (Max PDU Size of caller)
This parameter is used to establish the maximum PDU size in bytes that will be sent by either of the communication partners. The following PDU size values are allowed:

0070H (112 Bytes: 128 bytes minus header for layers 2 and 4)
00F0H (240 Bytes: 256 bytes minus header for layers 2 and 4)

Response Parameter Block:

P1 (Max AmQ calling negotiated)
This parameter identifies the maximum number of outstanding requests w/responses that the called partner will send before it waits for pending acknowledgments. This value is equal to or smaller than the suggested value. This prevents the calling partner from receiving more jobs with acknowledgment than it is able to process. For S7-200 the only allowed value is 1.

P2 (Max AmQ called negotiated)
This parameter identifies the maximum number of outstanding requests w/responses that the called partner can accept and process simultaneously. This value is equal to or smaller than the suggested value. This prevents the calling partner from sending more jobs with acknowledgment than the called partner is able to process. For S7-200 the only allowed value is 1.

P3 (Max PDU Size of caller)
Same as the request or maximum size supported by the S7-200.

Note: If the called partner returns a PDU size value that is not in the table of allowed values, the application association will be terminated.

ERR_CLS	ERR_COD	Description
83	04	Resouces not available; there are no more resources available for application associations to be established

4.6 Data Read/Write Using the ANY Pointer

The S7-200 will support reading and writing to all user data areas through read and write functions using variable addresses.

In order to provide complete clarification of what the S7-200 will support with regard to the variable address field (ANY-Pointer), the following description of the format and the definition of elements of the format are defined below.

ANY-Pointer (10 Bytes)

Syntax_ID = 10H	Type
Number_Elements	
Subarea = 0001H for V; 0000H otherwise	
Area	(msb)
Offset	(lsb)

Type	Type Code
BOOL	01H
BYTE	02H
WORD	04H
DWORD	06H
C (IEC)	1EH
T (IEC)	1FH
HC (IEC)	20H

Number_Elements

Type	Number_Elements
BIT	ONE: Bit addressing use 24 bit offset
BYTE WORD DWORD	Number of elements
C (IEC) T (IEC) HC (IEC)	Number of objects

Area	Area Code
S	04H
SM	05H
AI	06H
AQ	07H
C (IEC)	1EH
T (IEC)	1FH
HC (IEC)	20H
I	81H
Q	82H
M	83H
V	84H

Offset (Write access to a bit)

msb 23	03	02	01	lsb 00
Byte Address		Bit Address		

Offset (Read/Write access of bytes, words, and double word)

msb 23	03	02	01	lsb 00
Byte Address		0	0	0

Offset (Read/Write access of objects)

msb 23	lsb 00
Object Number	

Of the numerous PDU data types used with variable addressing (ANY-Pointer) formats, the S7-200 shall support only Boolean (03H) and bit string (04H).

The structure formats for the IEC timer, counter and high speed counter as used by the S7-200 are shown below:

Data Structures for S7-200 Communications Use

IEC Counter with 16 Bit CV

	7 (MSB)							0 (LSB)
0	0	0	0	C = CV≥ PR	0	0	0	0
1	CV (MSB)							
2	CV (LSB)							

IEC Timer with 32 Bit ET

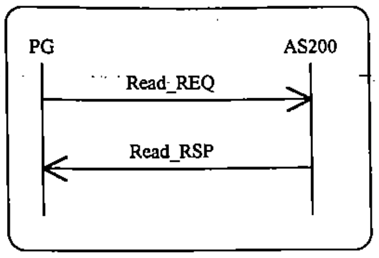
	7 (MSB)							0 (LSB)
0	0	0	0	0	0	0	T = CT≥ PR	0
1	CT (MSB; HH) = 0							
2	CT (HL) = 0							
3	CT (LH)							
4	CT (LSB; LL)							

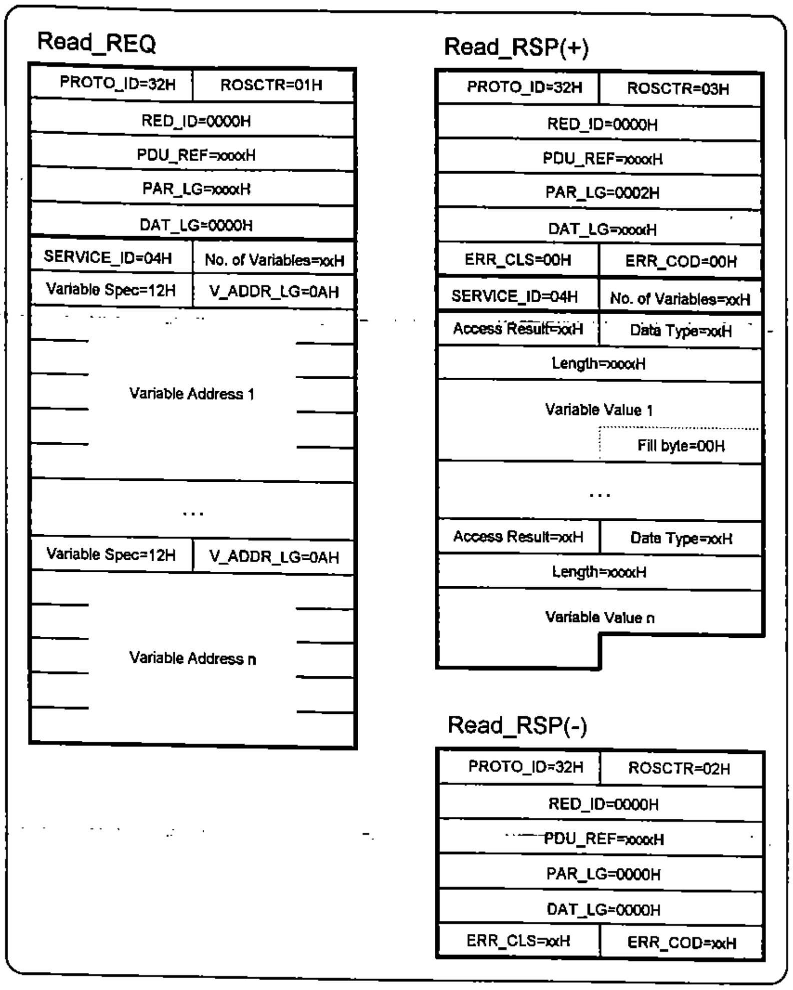
High Speed Counter

	7 (MSB)							0 (LSB)
0	0	0	0	0	0	0	0	0
1	CV (MSB; HH)							
2	CV (HL)							
3	CV (LH)							
4	CV (LSB; LL)							

4.6.1 Data Read

The following figures show the message sequence and the message formats.





Request Parameter Block:

Parameter	Type	Value
SERVICE_ID	UNSIGNED8	04H
# of Variables	UNSIGNED8	xxH
Variable Spec	UNSIGNED8	12H
V_ADDR_LG	UNSIGNED8	0AH

Number of Variables:
This parameter defines the number of variable addresses in the request.

Response Parameter Block:

Parameter	Type	Value
SERVICE_ID	UNSIGNED8	04H
# of Variables	UNSIGNED8	xxH

Number of Variables:
The number of variables specifies the number of values returned.

Response Data Block:

Parameter	Type	Value
ACC_RSLT	UNSIGNED8	xxH
D_TYPE	UNSIGNED8	xxH
VAR_LG	UNSIGNED16	xxxxH
Variable Value	UNSIGNED8	xxH
Fill Byte	UNSIGNED8	00H

ACC_RSLT:
Specifies the result of accessing each variable. The following results are possible:

FFH	No error
01H	Hardware fault
03H	Illegal object access
05H	Invalid address (incorrect variable address)
06H	Data type is not supported (currently, only octet string is supported)
0AH	Object does not exist or length error

D_TYPE:
Specifies the type of the data value returned. The following type values are possible:

00H	Returned if access result indicates an error
03H	Returned for bit access
04H	Returned for byte, word, double word, etc. access

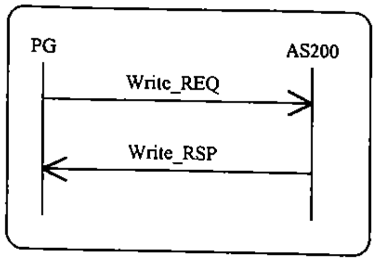
VAR_LG:
For successful accesses the VAR_LG specifies the number of bits(bytes * 8) in the variable value string. If there was an error, a value of 00H will be returned. The variable length does not include the fill byte(if present). Bit accesses return a length of 1.

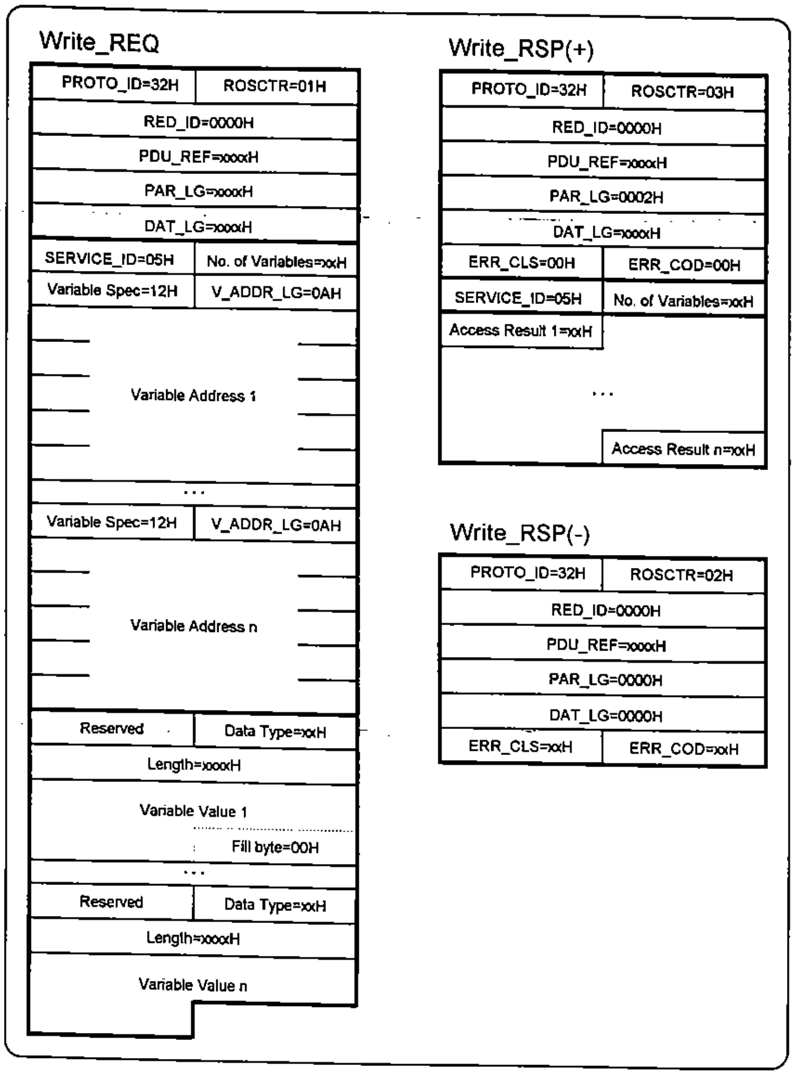
Fill Byte:
For variable value strings with an odd number of bytes, a fill byte is added to the end of the string to adjust the variable length to a word boundary. A fill byte is never appended to the last variable value string in the positive acknowledgment structure.

ERR_CLS	ERR_COD	Description
81	04	Context is not supported: -Error in PDU structure -Unknown service
84	04	Fatal error detected. Service or function aborted
85	00	PDU size error

4.6.2 Data Write

The following figures show the message sequence and the message formats.




Write_REQ
PROTO_ID=32H
ROSCTR=01H
RED_ID=0000H
PDU_REF=xxxxH
PAR_LG=xxxxH
DAT_LG=xxxxH
SERVICE_ID=05H
No. of Variables=xxH
Variable Spec=12H
V_ADDR_LG=0AH
Variable Address 1
...
Variable Spec=12H
V_ADDR_LG=0AH
Variable Address n
Reserved
Data Type=xxH
Length=xxxxH
Variable Value 1
Fill byte=00H
...
Reserved
Data Type=xxH
Length=xxxxH
Variable Value n
Write_RSP(+)
PROTO_ID=32H
ROSCTR=03H
RED_ID=0000H
PDU_REF=xxxxH
PAR_LG=0002H
DAT_LG=xxxxH
ERR_CLS=00H
ERR_COD=00H
SERVICE_ID=05H
No. of Variables=xxH
Access Result 1=xxH
...
Access Result n=xxH
Write_RSP(-)
PROTO_ID=32H
ROSCTR=02H
RED_ID=0000H
PDU_REF=xxxxH
PAR_LG=0000H
DAT_LG=0000H
ERR_CLS=xxH
ERR_COD=xxH

Request Parameter Block:

Parameter	Type	Value
SERVICE_ID	UNSIGNED8	05H
# of Variables	UNSIGNED8	xxH
Variable Spec	UNSIGNED8	12H
V_ADDR_LG	UNSIGNED8	0AH

Number of Variables:
This parameter defines the number of variable addresses in the request.

Request Data Block:

Parameter	Type	Value
Reserved	UNSIGNED8	00H
D_TYPE	UNSIGNED8	xxH
VAR_LG	UNSIGNED16	xxxxH
Variable Value	UNSIGNED8	xxH
Fill Byte	UNSIGNED8	00H

D_TYPE:
Specifies the type of the data value to be written. The following type values are possible:

03H	Specifies bit accesses
04H	Specifies byte, word, double word, etc. accesses

VAR_LG:
The VAR_LG specifies the number of bits(bytes * 8) in the variable value string. The fill byte(if present) is not included in the variable length. Bit values have a length of 1 even though a full byte is used.

Fill Byte:
For variable value strings with an odd number of bytes, a fill byte is added to the end of the string to adjust the variable length to a word boundary. A fill byte is never appended to the last variable value string in the request structure.

Response Parameter Block:

Parameter	Type	Value
SERVICE_ID	UNSIGNED8	04H
# of Variables	UNSIGNED8	xxH

Number of Variables:
The number of variables specifies the number of access results returned.

Response Data Block:

Parameter	Type	Value
ACC_RSLT	UNSIGNED8	xxH

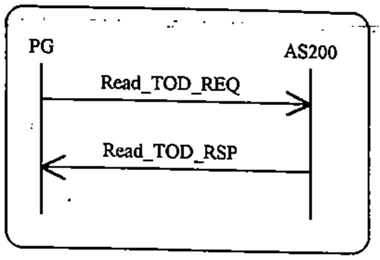
ACC_RSLT:
Specifies the result of accessing each variable. The following results are possible:

FFH	No error
01H	Hardware fault
03H	Illegal object access
05H	Invalid address (incorrect variable address)
06H	Data type is not supported (only data types 3 & 4 are supported)
0AH	Object does not exist or length error

ERR_CLS	ERR_COD	Description
81	04	Context is not supported: -Error in PDU structure -Unknown service
84	04	Fatal error detected. Service or function aborted
85	00	PDU size error

4.7 Read Time of Day Clock

The following figures show the message sequence and the message formats.



Read_TOD_REQ

PROTO_ID=32H	ROSCTR=07H
RED_ID=0000H	
PDU_REF=xxxxH	
PAR_LG=0008H	
DAT_LG=0004H	
SERVICE_ID=00H	NO._VAR=01H
VAR_SPC=12H	VADDR_LG=04H
SYN_ID=11H	CLASS=47H
ID1=01H	ID2=00H
ACC_RSLT=0AH	D_TYPE=00H
Length=0000H	

Weekday - 1 is Sunday

Read_TOD_RSP(+)

PROTO_ID=32H	ROSCTR=07H
RED_ID=0000H	
PDU_REF=xxxxH	
PAR_LG=000CH	
DAT_LG=000EH	
SERVICE_ID=00H	NO._VAR=01H
VAR_SPC=12H	VADDR_LG=08H
SYN_ID=12H	CLASS=87H
ID1=01H	ID2=xxH
SEG_ID=00H	MORE_SEG=00H
ERR_CLS=00H	ERR_COD=00H
ACC_RSLT=FFH	D_TYPE=09H
Length=000AH	
TOD_Clock_Status=xxxxH	
Year(2 BCD digits)	Month(2 BCD digitis)
Day(2 BCD digits)	Hour(2 BCD digits)
Minute(2 BCD digits)	Second(2 BCD digits)
1/10 sec | 1/100 sec	1/1000 sec | Weekday

Read_TOD_RSP(-)

PROTO_ID=32H	ROSCTR=07H
RED_ID=0000H	
PDU_REF=xxxxH	
PAR_LG=000CH	
DAT_LG=0004H	
SERVICE_ID=00H	NO._VAR=01H
VAR_SPC=12H	VADDR_LG=08H
SYN_ID=12H	CLASS=87H
ID1=01H	ID2=00H
SEG_ID=00H	MORE_SEG=00H
ERR_CLS=xxH	ERR_COD=xxH
ACC_RSLT=0AH	D_TYPE=00H
Length=0000H	

The TOD_clock_status is defined as :

KV	K4	K3	K2	K1	K0	r	r	r	r	ZNA	UA1	UA0	UZS	ESY	SYA

KV	Sign of corrective value (***0 in S7-200***)	
K(0..4)	Corrective value (0...31) in 30 minute steps (for summer/winter/world time; 1/2 hour) (***00000 in S7-200***)	
ZNA	Time value not up to date(***set if clock has stopped***)	
UA	Time resolution	00 = 1 mSec
		01 = 10 mSec
		10 = 100 mSec
		11 = 1 second(***only value possible for S7-200***)
UZS	Time skip	0 = no skip(***only value possible for S7-200***)
		1 = skip
ESY	Alternative synchronization	0 = none(***only value possible for S7-200***)
		1 = sync on LAN
SYA	Synchronization failure	
r	Reserved = 0	

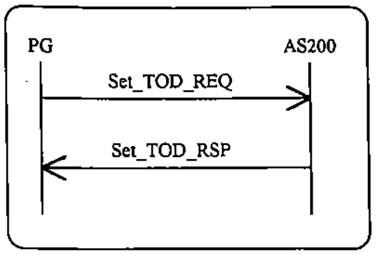
Only the last two digits of the year are used.

The day of the week is encoded as :

1H = Sunday
2H = Monday
:
7H = Saturday
0H = Day of week not set

4.8 Set Time of Day Clock

The following figures show the message sequence and the message formats.



Set_TOD_REQ

PROTO_ID=32H	ROSCTR=07H
RED_ID=0000H	
PDU_REF=xxxxH	
PAR_LG=0008H	
DAT_LG=000EH	
SERVICE_ID=00H	NO._VAR=01H
VAR_SPC=12H	VADDR_LG=04H
SYN_ID=11H	CLASS=47H
ID1=02H	ID2=00H
ACC_RSLT=FFH	D_TYPE=09H
Length=000AH	
TOD_Clock_Status=xxxxH	
Year(2 BCD digits)	Month(2 BCD digitis)
Day(2 BCD digits)	Hour(2 BCD digits)
Minute(2 BCD digits)	Second(2 BCD digits)
00H	0 | Weekday

Weekday - 1 is Sunday

Set_TOD_RSP(+)

PROTO_ID=32H	ROSCTR=07H
RED_ID=0000H	
PDU_REF=xxxxH	
PAR_LG=000CH	
DAT_LG=0004H	
SERVICE_ID=00H	NO._VAR=01H
VAR_SPC=12H	VADDR_LG=08H
SYN_ID=12H	CLASS=87H
ID1=02H	ID2=xxH
SEG_ID=00H	MORE_SEG=00H
ERR_CLS=00H	ERR_COD=00H
ACC_RSLT=0AH	D_TYPE=00H
Length=0000H	

Set_TOD_RSP(-)

PROTO_ID=32H	ROSCTR=07H
RED_ID=0000H	
PDU_REF=xxxxH	
PAR_LG=000CH	
DAT_LG=0004H	
SERVICE_ID=00H	NO._VAR=01H
VAR_SPC=12H	VADDR_LG=08H
SYN_ID=12H	CLASS=87H
ID1=02H	ID2=00H
SEG_ID=00H	MORE_SEG=00H
ERR_CLS=xxH	ERR_COD=xxH
ACC_RSLT=0AH	D_TYPE=00H
Length=0000H	

The TOD_clock_status is defined as :

KV	K4	K3	K2	K1	K0	r	r	r	r	ZNA	UA1	UA0	UZS	ESY	SYA

KV	Sign of corrective value(*0 in S7-200*)	
K(0..4)	Corrective value (0...31) in 30 minute steps (for summer/winter/world time; 1/2 hour) (*00000 in S7-200*)	
ZNA	Time value not up to date(*0 in S7-200*)	
UA	Time resolution	00 = 1 mSec 01 = 10 mSec 10 = 100 mSec 11 = 1 second(*only value accepted for S7-200*)
UZS	Time skip	0 = no skip(*only value possible for S7-200*) 1 = skip
ESY	Alternative synchronization	0 = none(*only value possible for S7-200*) 1 = sync on LAN
SYA	Synchronization failure	
r	Reserved = 0	

Only the last two digits of the year are used.

The day of the week is encoded as :

1H = Sunday
2H = Monday
:
7H = Saturday
0H = Day of week not set

ERR_CLS	ERR_COD	Description
DC	01	If clock set command and data is not BCD, data is out of range, etc.

Appendix A S7-200 Realization of PPI+ Protocol

A1 Overview

This Appendix describes the PPI+ protocol implementation used in the S7-200 CPUs. This protocol implementation is a simplification of that protocol defined in DIN 19 245 and for which an overview description is provided in Appendix C.

Profibus is the name given to an international standard for communications to field devices and is the German version of Field Bus. The S7-200 uses Profibus layer one and two specifications for the physical layer interfaces as are described in DIN Std. 19 245. The token passing functions have been added to the S7-214 in revision 1.0 but are not implemented in the S7-212.

The version of the protocol supported by the S7-200 is known as PPI (Point-to-Point Interface). The full protocol with all functions implemented is known as MPI (MultiPoint Interface). With the addition of the token handling, the functionality of PPI and MPI are nearly the same. The differences are baud rate (PPI is limited to 9600 and 19.2K baud), the polling of the slaves and the use of only the default SAP(Service Access Point) in PPI.

PPI slave support in the S7-200 provides the means for the programming units and operator panels to download ladder logic and read/write variable information.

The PPI+ master support in the S7-214 allows the 214 to read from and write data to other S7-200 devices. In this mode the 214 initializes itself as a token holding master which may then initiate requests to other devices. The requests are initiated by the user through use of the NETR and NETW ladder logic boxes. These boxes allow the user to read/write up to sixteen bytes per request from/to any other address.

A2 Background Communication Supervisor

When PPI+ is selected and the S7-200 CPU is in the run mode, the S7-200 CPU enters the PPI master mode. The S7-200 CPU will then attempt to enter the token ring or establish one if a ring does not exist. The interrupt service routines perform the handling of the FDL states. These states are described in a subsequent section. The main function of the communication supervisor is to process any request frames received by the interrupt routines. Whenever the interrupt routines receive a complete request frame in the Active Idle state, the background communication supervisor is notified. If there are no errors in the request, it is processed as described in the following section. When the response is complete, the interrupt routines are notified by the setting of a "response ready" flag and a 10-second timer is started. If the master does not pick up the response the S7-200 will discard the current response by resetting the response ready flag in the interface status byte (IFSTAT). This action is required since the S7-200 is a single buffer system and if there a request pending from a master, all other masters are locked out until the original master polls for the response frame. The timer ensures that if a master dies, the communication will reinitialize within ten seconds.

A3 Standard PPI functions

This section lists the allowed 3rd party PPI functions supported by the S7-200 with a description of how each is processed by the S7-200 CPU.

Establish Association - This is a master request for the PDU size of the secondary device and number of outstanding requests possible. The S7-212 will respond with a PDU size of 112 bytes and the S7-214 responds with a PDU size of 240 bytes. Both models respond that they allow only one outstanding request at a time.

Read Variables - This is the master request to read random variables from the S7-200. The PPI variable request format is converted to the internal data type codes and then error checked. Each variable is serviced before error checking the next. Once an error is found, that variable will respond with an access result error and then processing continues with the next variable. There is special handling for timers, counters and high speed counters because of the special formats required in the response for these data types.

Write Variables - This is the master request to write random variables to the S7-200. Like the read request, the Profibus request format is converted to internal data type codes and processed. If errors are found, all other accesses are aborted within the frame because the error may be with the data and may cause succeeding variables to write incorrect data to memory. There is special handling for timers and counters because of the special formats required for the data.

Read TOD Clock - This is a master request to read the time-of-day clock in the S7-214. This command is not implemented in the S7-212. The real time clock is read and a check is made to determine if it is running. If it is not running the status word of the response is set to show that the clock is not up to date. The status also shows that the resolution of the clock is one second. The time is returned to the master in BCD format.

Set TOD Clock - This is a master request to set the time-of-day clock in the S7-214. This command is not implemented in the S7-212. The clock data is passed to the routine which actually sets the clock. This routine will check the data and verify that all of the fields are valid BCD values and within the legal ranges for the particular fields. If all of the fields are legal, the clock is programmed.

A4 PPI Layer Seven Error Codes

The layer seven services will return error codes to the master when there are problems executing the request. The following is a list of the error codes supported by the S7-200s. There are many other error codes defined for the S5 protocol in general, but this is a subset supported by the S7-200 CPUs for the functions described in the previous section.

General errors :

8104	Command not supported - if 212, returned for read/set clock - returned for any other unsupported command
8404	Busy - for any command from a second master while an upload or download is in progress from a first master
8500	PDU length error - all commands if length fields do not agree with received lengths
8104	PDU structure error - all commands for misc syntax errors

Time of Day Clock Errors

DC01	time-of-day clock error - if clock set command and data is not BCD, out of range data, etc.

A5 PPI Layer One

The PPI layers one and two support is defined in DIN 19 245. This DIN standard defines the physical layer and the message structures, checksums, etc. for Profibus. The standard describes a token passing communication network, but only the S7-214 will hold the token only when configured to do so and the S7-200 CPU is in RUN mode. The S7-212 and the S7-214 in STOP are classified as a slave devices. The structures of PPI are based on Profibus as defined in this standard.

A5.1 Physical Implementation

PPI or Profibus (at this level) is an RS485 network. This means that it is automatically half duplex and contains no control signals such as RTS and CTS. The RS485 receiver is normally disabled when the transmitter is enabled so that we do not see the echo of the transmissions. The exception to this is when the S7-214 is acting as a token holding master and it must then monitor the token passes to verify operation of the transmitter.

The S7-212 provides minimal RAM for the PPI interface so the implementation described is rather limited. The PPI communication subsystem utilizes three large buffers of 128 bytes each for SD2 frame requests(2 buffers) and responses(1 buffer), and one small buffer (about 10 bytes) for SD1 poll responses. The frames are passed between layers one, two and seven via pointers to the buffers to minimize the time and RAM required.

The S7-214 provides more RAM for the PPI interface. In the S7-214, the buffers are 256 bytes long for the slave functions. The master functions allocate eight sets of master buffers. Each master buffer (Rx and Tx) is 64 bytes, for a total of 1K byte used for all of the master buffers. The master buffers are shorter than the slave buffers since the maximum length of the message is known.

Neither model S7-200 provides more than minimal hardware support for a PPI interface. The UART in the 8032 is used as the PPI port. The receiver has a one byte buffer meaning that the receive interrupt has one entire byte time to remove the data from the UART before the character is replaced by the next incoming character. The transmitter is not buffered and the transmitter empty interrupt is received as the UART begins transmitting the stop bit for the byte. This means that there is only one bit time (104 uSecs at 9600 baud) to reload the transmitter to transmit with no gaps between characters.

The baud rate for PPI is currently fixed on both units(S7-212 and S7-214) at 9600 baud with eight data bits, one start bit, one stop bit and even parity.

Profibus defines specific timings for synchronization of the communication link. A message start is denoted by the first byte received after a sync time (***Tsync***) of 33 bit times at the current baud rate. Expiration of a sync time period is determined by setting a timer for a time-out of approximately 41 bit times(30 bit times for a sync time period and 11 bit times for the first character). When attempting to determine expiration of a sync time period, the timer is reset to 41 bit times on each received byte. When the timer eventually expires, there will have been an idle time on the communication line for at least 30 bit times. This will signal the S7-200 that the next byte received will be the start of a new message.

Nearly all messages are acknowledged by the receiving device. This normally takes the form of a single character acknowledge (SC ack) or the response to the latest request. The time between the receiving of the last character of the request and the acknowledge is known as the ready time (Trdy) and is specified to be a minimum of 11 bit times. The S7-200 will wait 22 bit times.

The communication timer is also used to measure other times such as a hold timer to be sure the transmitter stays enabled until the stop bit is completely sent (just one bit time). It is also used to wake up the master functions periodically where there are no other communications present (and interrupting) on the line. The specific functions of the timer will be described later in the token handling state descriptions.

A5.2 UART Interrupt

The communication interrupt has four modes of operation. The four modes are :

1) No activity - UART interrupts are cleared and the port disabled.
2) Receive - Characters are checked for parity and stored in a buffer.
3) Transmit - Characters are transmitted out the UART.
4) Sync time search - A timer is reset for each character until it expires.

The "no activity" mode should never be used unless the port is disabled completely.

The "transmit" mode is initiated normally on a transmitter empty interrupt from the UART. When the last character is loaded on the last interrupt (no more characters to send), the transmit interrupt will set a timer for one bit time at the current baud rate. This last interrupt comes at the beginning of the stop bit of the last character and the timer is used to hold the RS-485 transmitter on for one bit time longer.

There is a special case in the transmit service routine to assist the monitoring of the token pass. When the token is passed, both the transmitter and receiver are enabled allowing the system to check for collisions on the line. If the receive interrupt is set during transmitting, the service routine will verify the received character matches the last transmitted character and increment a receive count. A mismatch flag and the receive count will be checked by the layer two handler after the token pass is complete to verify the token pass.

The "sync time search" mode will reset a timer to 33 bit times whenever a character is received. This will happen until the timer finally times out and the timer interrupt changes the communication mode to the receive mode.

The "receive" mode is the most complicated and longest of the interrupt service routines.

The receive interrupt routine manages a character/message timeout timer. When the system is on a real Profibus network, denoted by tokens being received, the timer is reset as each character is received to look for gaps between characters. The gap is currently allowed to be anything up to the sync time(33 bit times). When the system is not on a real Profibus network, denoted by no tokens being received, the times are lengthened so that a PC operating Windows will not have a problem when there are large gaps between the transmitted characters. In this mode the receive interrupt resets a very long timer on the first character. If this timer ever expires, the message is aborted. The timer is set for 4100 bit times which is one and one-half times the length of the longest possible message (249 chars * 1.5 * 11 bits/char). This mode is only used when a non-token-passing master is attached to an S7-200 CPU. The background communication supervisor provides a timer which ages the tokens. If there are no tokens received for one minute, the S7-200 CPU will go the non-Profibus mode. The system is initialized to the non-Profibus.

The receiver always receives the entire message so long as there are no errors. In case of errors, an error event is set and the layer two processor called. This will normally result in the system being reset to look for a sync time. Profibus messages are always aborted/ignored in case of errors. If the message is received without errors, a message complete event is set and the layer two processor called. The layer two processor examines all messages in order to monitor the token passes.

A5.3 Timer Interrupt

The timer interrupt operates similarly to the UART interrupt in that there are various modes. The modes of the timer interrupt are :

1) Disabled - There should be no timer interrupts so the timer is disabled.
2) Character/Message Timeout - The gap/message timer has expired.
3) Trdy Timeout - The turnaround timer has expired.
4) Thold - The transmitter hold timer has expired.
5) Tsyn - The message sync timer has expired.
6) Response Timeout - Master mode response timer expired.
7) Tpoll - Master mode poll delay has expired.
8) Twake - Master mode wakeup timer has expired.

The "disabled" mode should never be entered, but if it is, the timer is disabled and the interrupt is aborted.

The "character/message timeout" mode is set up by the UART receive Interrupt Service Routine after the first character of a message has been received. This timer monitors for gaps between bytes (character mode) or for a total message length (message mode). If the timer ever expires, an error event is set and the layer two processor called to handle the event.

The "Trdy timeout" is used when the S7-200 CPU has received a message and is waiting to initiate a response. The layer two processor will start the timer with the Trdy time. When the timer expires it sets a "Trdy complete" event and invokes the layer two processor to begin transmission of the response.

The "Thold timeout" is used to time one bit time so that the RS 485 transmitter can be kept on while the stop bit transmits. This interrupt mode will disable the 485 driver and send a "transmit complete" event to the layer two processor.

The "Tsyn" mode is used when the S7-200 is searching/waiting for a 33 bit time idle line to delineate the start of a message. This timer mode is initiated in various locations whenever one transaction is complete and the next message must be received. This interrupt will set the "sync complete" event and pass that to the layer two processor.

The "response timeout" mode is used when the master has transmitted a request and is waiting for the slave device to respond. If the slave responds, the response will cause the response timer to be disabled before it expires. If the slave does not respond, the response timer will expire and a "no response error" event will be passed to the layer two processor.

The "Tpoll" mode is used to provide a time delay between receiving an acknowledge from a slave and the sending of the poll request. This is only used by the master communication functions. When the timer expires, a "poll delay complete" event is passed to the layer two processor.

The "Twake" mode is used to provide a wake up interrupt to the layer two processor about every ten milliseconds if there have been no other events to wake up the layer two processor. This wake up is used to age an idle line and to initiate the sending of requests when the S7-214 is the sole master on the line. This mode will set a "system timer" event and pass that to the layer two processor.

A6 PPI Layer Two

The PPI layer two functions follow the Profibus standard DIN 19 245. The S7-200s are somewhat limited in the operations that they can perform due to the software emulation of some hardware functions. This will generally not affect the performance of the S7-200s on a Profibus network, but may affect the operation with some types of masters.

The S7-200s will accept requests from any master but will only be able to service one PPI layer seven request at a time. If a layer seven request is in process, the S7-200 will respond to other data requests (SD2 frames) with a NAK(RR or RS SD1 frame described in Section 3.0) response. The S7-200 will accept and respond to polls and requests for FDL status from any master while a layer seven request is in process, since these are layer two functions and are not affected by the layer seven processing.

The S7-200 does not support messages with extended addressing or broadcast messages. The S7-200 will also not respond to SD3 requests. When in PPI slave mode, the token pass request(SD4) is noted and the frame control count reset, but there is no other action taken and the S7-200 will not accept or respond to a SD4 frame. Only when the S7-200 is configured as a PPI master will the S7-200 CPU accept and act on SD4 messages.

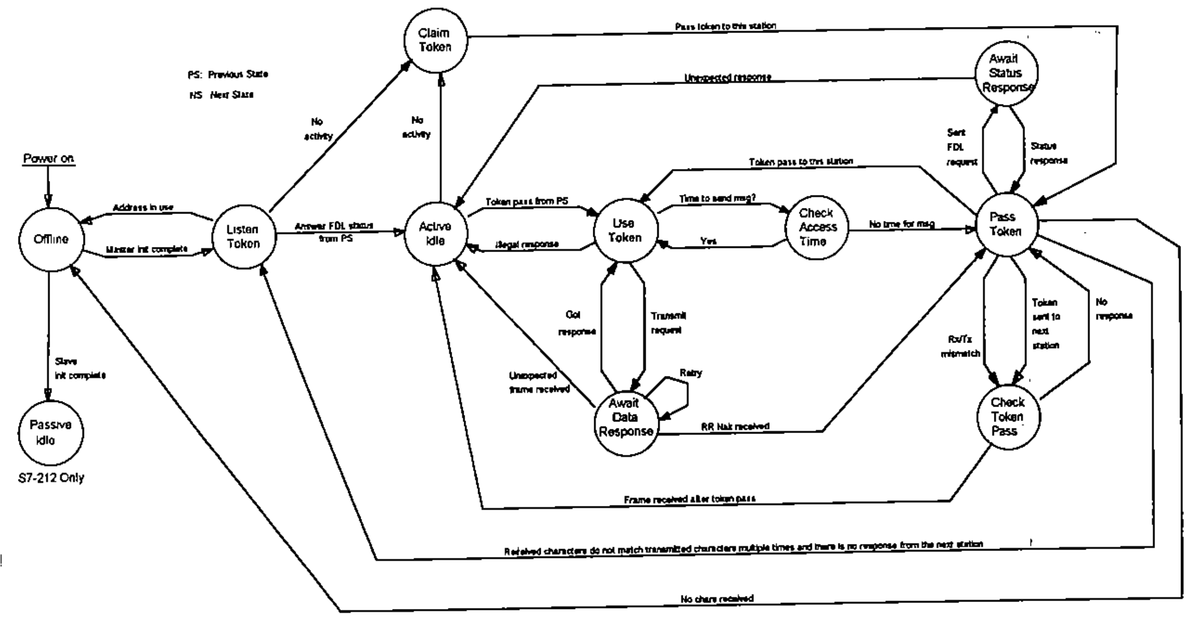
The default address of the S7-200 is two. Siemens reserves address zero for the programmer and address one for an operator panel.

The layer two functions are broken down into eleven states. These states encompass all of the processing needed to establish, acquire, monitor, accept, use, pass and maintain the token. The states are :

STATE	DESCRIPTION
OFFLINE	Startup or failure state.
PASSIVE IDLE	Slave state. The S7-212 utilizes only this state.
LISTEN	Listen to determine which masters are already on the network.
CLAIM TOKEN	Claim the token for this station(TS) if the line is idle.
ACTIVE IDLE	Slave state of the S7-214 when accepting requests from another master.
USE TOKEN	The master makes requests from other network devices.
AWAIT RESPONSE	The master waits for an ACK and polls for the response.
CHECK ACCESS TIME	Check the timers to see if there is time to make a request.
PASS TOKEN	Pass the token to another master.
AWAIT STATUS RESPONSE	Manage the gap between this station and the next master on the network.
CHECK TOKEN PASS	Verify that another master accepted and is using the token.

The following sections describe the implementation of the Profibus token states in the S7-200s.

The diagram on the following page shows the token states and the transitions between them.



Token Handling State Diagram

A6.1 Offline

This is the initial state that is entered upon power up. It is also the state that is entered if there is an unrecoverable error on the network. An unrecoverable error occurs when this station detects another station already using this station's address.

Once this state is entered, there will be no further activity by this station on the network. There is no escape from this state other than a reset of the communication.

A6.2 Passive Idle(S7-214)

The passive idle state is the slave device state and is not utilized on the S7-214. The slave communications are handled in the active idle state on this model. The passive idle state is similar to the offline state and takes the same actions. Communication should never enter this state on the S7-214.

A6.3 Passive Idle(S7-212)

The passive idle state is the slave state where the station acts only as a PPI slave. This is the case in the S7-212. The passive idle state is entered after a reset of the communication and is never exited.

The purpose of passive idle state is to provide an immediate response/acknowledgment to a master. This will normally be a single character acknowledge for a SD2 frame or for a poll for an outstanding frame if the response is not complete. Requests for FDL status will be processed in the passive idle state along with error messages for back-to-back SD2 frames or buffer overrun conditions. If a SD2 response frame is ready, the passive idle state will initiate the sending of the frame in response to a poll from the original requesting master.

The passive idle state consists of several event handlers to process the various events triggered by the layer one interrupt service routines.

A sync complete event will be triggered by the timer interrupt when a sync time expires. This means that the next character received on the line should be a start of frame character (SD1 = 10, SD2 = 68, SD3 = A2, SD4 = DC). The handler for this event sets the UART to receive a message.

A transmit complete event will be triggered whenever this station has transmitted some sort of response back to the master. The action in this case is to look for a sync time on the line to delineate a new message.

A transmit ready event is triggered when the Trdy timer expires. This timer is initialized whenever a complete message is received(see receive complete event) and a response to the master must be initiated. The Trdy timer allows the master and/or repeaters to turn around the communication drivers to the receive mode. This event will begin the transmission of the response to the master when the timer expires.

The receive complete event handler is the largest of the passive idle state handlers. This event is triggered after the last byte of the message has been received with no errors. Since a complete message was received without errors, there will be some sort of response generated to the master. The Trdy timer is started to trigger the transmission of the response after a turn around delay of 22 bit times. The delay of 22 bit times is used to allow any repeaters (specifically the PC/PPI cable) to turn around the RS 485 drivers and enter the receive mode.
The S7-200s monitor the frame control bits in the received message to determine if a request is for a re-transmission of the last response. In this case, the request will be an SD1 poll with the frame control valid bit set and the frame control bit will be in the same state as it was on the prior poll, all from the same master. If a retry is requested, layer two restores information about the previous response and initiates the retransmission.

The S7-200 will not monitor frame control bits for proper sequencing, nor store the status of the frame control bit as the master changes. Each master must execute any retries before passing the token to another master. Once the token has passed and another master accesses the S7-200, the last response frame will be discarded.

After checking the frame control bits, the frame type is determined. The layer two response will take one of several forms depending on the request and the current state of the higher level (layer seven) message processing. If the response is an ACK, NAK or for FDL status, the response will be built by the layer two processor as required. The conditions and responses defined for the S7-200 are :

1. There was an error due to parity, checksum or receive buffer overrun.

   When the receive interrupt service routine detects an error, an error event is generated which will cause the passive idle state handler to set up to search for a sync time. The Profibus standard states that all errors are handled by not responding to the master.

2. The S7-200 sees a token pass message.

   The S7-200 will discard the old *retry* response(also see items 4 & 5 below and item 2 in the SD2 request description in Section 3.1). Retries for the last response will now be answered with a SC ack. The background communication processor is notified that a token frame has been detected. This information is used to determine that the S7-200 is on a real Profibus network (one with tokens being passed) and causes the S7-200 CPU to check for gaps between characters of subsequent messages. The token pass itself is ignored by the S7-200 even if it is to this station.

3. The source (master) address changed.

   The S7-200 will reinitialize the frame control bit to that in the current request. Retries for the last response (from the last master) will now be answered with a SC ack. The message is then processed normally (as in 5 below).

4. The source address is the same but the frame control bit did not change.

   The S7-200 will resend the last response to the master.

5. The source address is the same but the frame control bit did change.

   The new frame control bit is stored and the last response discarded.

If the request is a poll and the response is ready, the response is sent.

If the request is a poll and the response is not ready, a single character acknowledge is sent.

If the request is an SD2 frame, a single character acknowledge is sent back to the master and the SD2 frame is then passed up to the background communication processor (layer seven handler) for processing. If an SD2 frame is already pending, an RS NAK response is returned to the master to reject the request.

6. The frame was an SD1 FDL status request.

   The FDL status response is built and sent to the master.

7. The frame was a SD1 but not a request for status or a Send and Receive Data low (SRD low)

   An SD1 NAK(RS) response will be built and sent to the master. The S7-200 will only support SRD low requests.

A6.4 Listen Token

After the power up and communication initialization is complete, the PPI master enters the listen token state to determine the current status of the network before attempting to enter or establish the network. Upon entering this state the LAS (List of Active Stations) is cleared, the NS (Next Station) and PS (Previous Station) are set to null addresses, and the FDL status response is set to show that TS (This Station) is a master not ready to enter the network.

The purpose of this state is to build a LAS so that this station knows its position in the network. To accomplish this, all of the messages on the network are received, watching for the token passes. When a token pass is detected, the source address, which must be another master, is added to the LAS. While this process is being performed, this station may receive a FDL status request from another master. The status request is answered as a master not ready to enter the network. Once the LAS is complete, we determine the NS and the PS and set the FDL status response to master ready to enter the network. At this point the listen token state is complete and the state changes to the active idle state to await the token.

The building of the LAS is done in three stages. In the first stage the 214 will monitor the token pass source addresses to find the lowest address. Once this address has been detected for the second time, the second stage is entered in which the token source addresses are added to the LAS. When the lowest address is detected again, the third stage is entered. This stage is just to verify that the token source addresses exactly match those in the LAS. This is just a confirmation stage to verify the LAS. When the lowest address is detected for the fourth time, the verification is complete and a transition is made to the active idle state.

When the listen token state is initialized, a system timer is started to interrupt and return to the listen token state every ten milliseconds. Every ten milliseconds an idle line timer is decremented. If this timer ever reaches to zero, this station will claim the token by changing to the claim token state.

The idle line timer is based on the address of this station and the slot time. The slot time is defined as the amount of time a master waits for a response from a secondary before determining that the secondary has failed to answer. The formula for the idle time is :

Idle time = (6 * slot_time) + (2 * my_address * slot_time)

This formula allows the network to be initialized by the lowest numbered address on the network, providing all devices assume the same slot time. The S7-214 and the TD200 allow 30 milliseconds as the slot time at 9600 baud.

A6.5 Claim Token

The claim token state is entered from two places: the listen token state and the active idle state. The entry condition into both states is due to the communication line having been idle for longer than the idle line time. The claim token state then transfers control to the pass token state so that the token will be passed to this station. The token is passed to this station so that other master devices will see an active line and note that another master device is establishing the network. ***The token is only passed to TS one time rather than the two times noted in the DIN 19 245 standard.***

A6.6 Active Idle

The active idle state is only implemented in the S7-214 and is very similar to the passive idle state in the S7-212. The only real difference in the two states is how the token frames are handled. In the passive idle state(S7-212), the tokens are ignored except for use in deciding to discard the last response message. In the active idle state(S7-214) the token frames are used to update the list of active stations (LAS) and to switch the state to the use token state so that this station may make requests on the network.

If a token frame is received for this station, a check is made to see if the source of the frame is the registered previous station (PS). If the source is the PS, then the token state is changed to the use token state and control transfers there. If the source of the token frame is not the PS, the Profibus standard states that this station should only accept the token from the new previous station on the second token pass from the new station. To meet this standard, if the 214 receives a token pass from a station which is not the current PS and the station number sending the token is less than the current PS, the LAS is modified to remove the current PS (since it must have dropped off of the network) and the new PS (the source of the token pass) is added. If the station number sending the token is greater than the current PS, the new PS is simply added to the LAS. After the LAS is modified, the 214 waits for the next message. The next token pass to this station will be accepted if the source is now the new PS. Thus, the token should be accepted from a new PS on the second attempt from that station.

The only other difference between the active idle state and the passive idle state is that the active idle state must monitor for an idle line. If the idle line time expires, this station is supposed to claim the token for itself and begin rebuilding the network. This is the same idle line time as discussed in the listen token state.

Each time the sync timer expires, i.e., whenever a transaction is competed and communication is waiting for a new message delimiter, the idle line timer is reset to the maximum value for this station. A 10 millisecond wake up timer is also started. Every 10 milliseconds the wake up timer will expire and trigger a system timer event. The system timer event handler for the active idle state will decrement the idle line timer and if the timer reaches zero, change the state to the claim token state. This will cause the system to claim the token for this station and pass the token to this station to notify other masters on the line that there is now an active master on the line. If the idle line timer has not expired, the wake up timer is reset for another 10 milliseconds and the communication goes to sleep until then or until some other event occurs.

See the passive idle state for information on how the individual frames are handled.

A6.7 Use Token

The use token state is entered from several states. The main and most important entry comes from the active idle state when that state determines that there has been a token pass to this station from the PS. In this case, the setup event for this state is entered which sets the FDL status response state to master in ring (since a token was just received) and resets the counter for the number of requests to be made during each token hold(see Check Access Time state). Control then passes to the system timer event handler.

The system timer event occurs every 10 milliseconds when there is nothing else going on and the token is held by this station. This event will check to see if there is time to send another request by changing the state to the check access time state. Control will return back to the use token state if there is time to send another request or control will transfer to the pass token state if there is not time for another request.

If the last time this station had the token it timed out polling the secondary, the state is changed to the await response state to continue the polling. If this station was unable to issue a request on the last token hold due to the resources of the secondary being held by another master (this station was NAK'd due to a frame pending), the last message is now resent. If there is no polling or request retries to do, a new request may be sent. The master request queue is checked and if there is no request, control is transferred to the pass token state. If there is a request in the queue, a timer is started to delay the transmission for a sync time.

Note that a master should set the sync time to 40 - 50 bit times.

The sync time event will be entered when the sync time has expired. This event handler will initiate the transmission of the master request. When the transmission is complete, a Tx complete event will be set.

The Tx complete event occurs when the message has been completely sent out. Control now transfers to the await response state to look for the ACK and poll the secondary for the response.

The use token state will also be entered (as a system timer event) from the await response state when a request/response cycle has completed. There is also an entry from the pass token state when this station has passed the token to itself. Again, the state is entered as a system timer event.

A6.8 Await Response

The await response should be a very simple state in which the master simply waits for the acknowledgment from the secondary that the message was received. In the S7-200 world, it is complicated by the fact that the secondary must be polled for the response.

The await response state is entered after sending a request to a secondary and then the use token state changes to the await response state for the next event. The next event will normally be a receive complete event generated when the acknowledge comes back from the secondary. The received message will either be a SC (single character) acknowledge, a NAK (negative acknowledge) or an SD2 response frame. Each of these responses is handled differently.

The SC ack will be returned in response to an SD2 request frame. The response is to wait a sync period and then send a poll to the secondary. A poll timer is started to signal when the next poll is allowed. The maximum number of polls which may be attempted within one token hold is 15. This is to force a token pass so that one master does not hold the token for a long period of time while the slave is busy. When this maximum is reached, a flag is set to indicate polling and the state is changed to the pass token state. On the next token hold, this flag is checked in the use token state and if set a transition is made to the await response state where the polling can continue.

A NAK(RR or RS SD1 frame described in Section 3.0) response will be returned when the S7-200 slave CPU already has a frame pending and cannot accept the new request. In this case a retry flag is set so that on the next token pass to this station, the use token state will resend the same request. After setting the retry flag, the state is changed to the pass token state.

An SD2 frame will be returned in response to one of the polls sent from this state. When an valid SD2 frame is received, a flag is set to notify the background that the response is complete and return control to the use token state to see if there is time to send another message during this token hold.

A6.9 Check Access Time

The check access time state is entered each time that the use token state wants to send a request. This state is intended to check how much token hold time is left for this station and return back to the use token state if there is some token hold time still available. If there is not time available, then the state changes to the pass token state.

In the S7-214 the check access time state is implemented to allow one request/response cycle per token hold. This was done to minimize the adjustments required when adding master devices to a network. Since each S7-214 master sends only one request, as S7-200 CPUs are added to the network the token cycle time increases linearly.

A6.10 Pass Token

The pass token state is responsible for providing gap management and passing the token. The gap management function sends an FDL status request to the stations between this station address and the next station address. This range of addresses is known as the gap. If the NS is the actual next address there is no gap and thus no gap management. The gap management function is used to look for additional masters wanting to enter the network.

There are two entry events to the pass token state. The normal setup event provides for gap management. The alternate setup state does no gap management and is used when an immediate token pass is desired as in the case where the idle line timer has expired and this station is attempting to establish the network by passing the token to itself.

The normal setup event will first start a timer to provide detection of a sync time period on the line. When the sync timer expires, the sync event handler will determine if there is a gap (the gap address is not null) and if so, send a FDL status request to the current gap address. The state is then changed to the await status response state to wait for either a time-out or the response to the status request. Either way, the await status response state will set an event and return to the pass token state.

When the response comes back from the await status response state, there will be either a no gap response event or a gap response event. Both of these events modify the gap list based on the new information about the secondary polled. After the address has been either logged in or out(if there was no response) then the token is passed to the next station. If there was a response, there is a sync delay initiated before the token is passed to the next station.

The gap list is implemented as just the address currently being checked within the gap. If there was no response to the FDL status request, the address is set to the next address to check. If there was a response to the status request, the response is checked to see if it was from another master wanting to enter the network. If this is the case, the new master address is added to the LAS and the new NS and PS addresses determined. The next gap address is determined and the gap list update is complete.

The token pass message should be monitored by the master passing the token to look for collisions on the line and to verify that the transmission hardware is working. When in the transmit interrupt, if the presence of a receive interrupt is detected, the received character is compared to the last transmitted character and a flag is set if there is a mismatch. The transmit interrupt will also maintain a count of the characters received. After the token pass is complete there will be a transmit complete event generated and this event will check the mismatch flag and the character count. If there were no characters received, indicating a failure of the transmitter, the token state is set to the offline state. If more than zero but less than three characters were received or the message was garbled, a retry of sending the token will be attempted if there is no response from the receiver of the token. If the NS begins transmitting, the problems are assumed to be some intermittent problem and no action is taken.

The token pass is verified by passing control to the check token pass state. If this was a token pass to this station (this station is the only master on the network, the NS is null), the token state is set back to the use token state and control passes to the setup event there.

A6.11 Check Token Pass

The check token pass state will wait for the next station to begin transmitting, thereby indicating that the token pass was successful. If the next station does not begin transmitting, there is some problem and the token pass is attempted again.

The check token pass state starts a timer which will wait one slot time (about 30 milliseconds at 9600 baud) before expiring. If the timer expires, control is transferred to the pass token state with a no response event. This event handler will check the receiver count and if no characters have been received, control is transferred back to the pass token state to retry the token pass. If some characters have been received, indicating that the next station accepted the token and has begun using it, control is transferred from the check token pass state to the active idle state. This will let the active idle state handle the communication in progress.

A receive complete event may occur while in this state. If this is the case (there was a very short message transmitted by the NS) the receive complete event is simply passed to the active idle state handler.

If characters are received but there are parity or other problems with the characters, a transition is made to the active idle state to handle the message and assume the token pass was successful.

A6.12 Await Status Response

The await status response state is entered after the pass token state has initiated the transmit of a FDL status request to an address in the gap. This state will monitor the link for a response and check the response or wait for the link to time out if there is no station at that address.

The pass token state will begin the transmission of the FDL status request and then change the token state to the await status response state. The first event the await status response state will see is the transmit complete event when the status request has been transmitted. The event handler enables the system to receive the response and starts a timer for expiration in one slot time (about 30 milliseconds at 9600 baud).

If there is no station at the requested address, the slot time will expire and a time-out event occurs. At this point, a no response event is generated and a transition is made back to the pass token state which handles the gap management.

If there is a response, a receive complete event occurs. In this case, the response is checked to be sure that it is a FDL status response from the requested address. If everything is correct, a gap response event is generated and control is transferred back to the pass token state which handles the gap management. If there are any errors, such as a response from a different station, a transition is made to the active idle state to handle the receive complete event. This error response assumes that there is another master in control of the network.

If there are receiver errors in the response, it is treated as a no response event and control is transferred back to the pass token state to manage the gap.

Appendix B S7-200 PPI+ Operating Parameters

B1 PPI+ Simplifications

The following simplifications characterize the PPI+ protocol implementation:

PPI+ utilizes only low priority messages(Frame Control Byte hex values of : 6C, 5C, 7C)

Only one message/request is processed by a master during a token hold. This simplifies the timekeeping requirements and makes network performance predictable as masters are added. ***Ttargr*** and ***Trealr*** are not utilized in the S7-214 masters.

Gap maintenance is performed each token hold cycle. One address is checked and then the token is passed.

The number of polls issued within one token hold is limited to 15. This forces the master to pass the token for the case of a slave device busy, e.g., compiling a downloaded program, for several seconds. The slave CPU may also be busy due to a long execution time for the ladder program. Layer seven responses are processed by the CPU at the end of each scan cycle.

The current S7-200 master devices allow a slot time(***Tslot***) of 30 milliseconds to accommodate slow programming devices, e.g., PG702.

When initializing a network, the current PPI+ masters pass the token to TS and then process one station in the gap during each token hold cycle. A token hold cycle consists of passing the token to TS, processing one request/response and a gap update. The current masters do not initialize all masters on the first cycle.

B2 Actual Examples of S7-200 PPI Network Traffic

Following are examples of the S7-200 communication sequences with two masters and two slave devices present on the network.

TD 200 at address 50 (hex 32)
S7-214 master at address 5
S7-214 slave at address 2
S7-212 slave at address 7

The leftmost column describes the type of communication frame along with the "from" and "to" addresses (in hex). The second column is the time between the last character of the previous frame and the first character of the current frame. The third column shows the actual communication frame. Note that all values are in hex notation.

The S7-214 master (address 5) is initially the only master online. It will pass the token to itself, check for other devices on the network and make requests of the S7-212 slave (address 7).

The S7-214 makes a write request (service ID 5) to address 7. The request is to write two bytes (values 01 02) starting at Q0.

```
Token Pass   26.554 mSec   DC 05 05
05 -> 05

SD2 Req      5.563 mSec   68 21 21 68 07 05 7C 32 01 00 00 D1 D1 00 0E 00
05 -> 07                  06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                          04 00 10 01 02 40 16

SC ACK       2.398 mSec   E5

Poll         5.250 mSec   10 07 05 5C 68 16
05 -> 07

SD2 Resp     2.444 mSec   68 12 12 68 05 07 08 32 03 00 00 D1 D1 00 02 00
07 -> 05                  01 00 00 05 01 FF F3 16
```

The S7-214 master issues a status request to one of the addresses in the gap. The gap is the range of addresses between any two masters. The S7-214 waits one slot time (greater than 25 mSec to allow for the PG702) for the device to respond. Since there is no response, the S7-214 passes the token to itself so that other masters which are not yet on the network will know that there is a token holding master already on the network.

```
FDL Req      5.290 mSec   10 30 05 49 7E 16
05 -> 30

Token Pass   26.560 mSec   DC 05 05
05 -> 05
```

Here is another write request to address 7, Q0 with data of 00 00. There is also a gap update and token pass.

```
SD2 Req      5.540 mSec   68 21 21 68 07 05 7C 32 01 00 00 D2 D2 00 0E 00
05 -> 07                  06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                          04 00 10 00 00 3F 16

SC ACK       2.406 mSec   E5

Poll         5.242 mSec   10 07 05 5C 68 16
05 -> 07
```

```
SD2 Resp     2.340 mSec   68 12 12 68 05 07 08 32 03 00 00 D2 D2 00 02 00
07 -> 05                  01 00 00 05 01 FF F5 16

FDL Req      5.273 mSec   10 31 05 49 7F 16
05 -> 31

Token Pass   26.550 mSec  DC 05 05
05 -> 05
```

Here is another write request to address 7.

```
SD2 Req      5.540 mSec   68 21 21 68 07 05 7C 32 01 00 00 D3 D3 00 0E 00
05 -> 07                  06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                          04 00 10 00 00 41 16

SC ACK       2.435 mSec   E5

Poll         5.306 mSec   10 07 05 5C 68 16
05 -> 07

SD2 Resp     2.375 mSec   68 12 12 68 05 07 08 32 03 00 00 D3 D3 00 02 00
07 -> 05                  01 00 00 05 01 FF F7 16
```

The S7-214 master polls the TD 200 with a status request as part of the gap update and finds that there is a master there which is ready to enter the network. The S7-214 logs address 50 (32 hex) into its List of Active Stations (LAS) and passes it the token.

```
FDL Req      5.256 mSec   10 32 05 49 80 16
05 -> 32

FDL Resp     2.531 mSec   10 05 32 20 57 16
32 -> 05

Token Pass   5.284 mSec   DC 32 05
05 -> 32
```

The TD 200 is now able to make a request to the slave at address 2 since it now holds the token. This is a read request for one byte to read the stop/run status of the S7-214 slave.

```
SD2 Req      5.473 mSec   68 1B 1B 68 02 32 6C 32 01 00 00 02 02 00 0E 00
32 -> 02                  00 04 01 12 0A 10 02 00 01 00 00 03 00 0F 18 43
                          16

SC ACK       2.336 mSec   E5
```

```
Poll          5.319 mSec   10 02 32 5C 90 16
32 -> 02

SD2 Resp      2.269 mSec   68 16 16 68 32 02 08 32 03 00 00 02 02 00 02 00
02 -> 32                   05 00 00 04 01 FF 04 00 08 01 8D 16
```

The TD 200 now initiates a gap update starting just beyond its own address. It then passes the token back to the S7-214 master.

```
FDL Req       5.319 mSec   10 33 32 49 AE 16
32 -> 33

Token Pass    30.565 mSec  DC 05 32
32 -> 05
```

The S7-214 master makes another write request to addres 7.

```
SD2 Req       5.352 mSec   68 21 21 68 07 05 7C 32 01 00 00 D4 D4 00 0E 00
05 -> 07                   06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                           04 00 10 00 00 43 16

SC ACK        2.412 mSec   E5

Poll          5.226 mSec   10 07 05 5C 68 16
05 -> 07

SD2 Resp      2.360 mSec   68 12 12 68 05 07 08 32 03 00 00 D4 D4 00 02 00
07 -> 05                   01 00 00 05 01 FF F9 16
```

The S7-214 at address 5 can now shorten its gap since it found another master. The gap for the S7-214 is now the address in the range 6 to 49.

```
FDL Req       5.273 mSec   10 06 05 49 54 16
05 -> 06

Token Pass    26.559 mSec  DC 32 05
05 -> 32
```

The TD 200 has no requests to make so it just does a gap update and passes the token back the the S7-214 master.

```
FDL Req       5.496 mSec   10 34 32 49 AF 16
32 -> 34
```

```
Token Pass    30.562 mSec   DC 05 32
32 -> 05
```

The S7-214 master makes yet another write request to address 7.

```
SD2 Req      5.320 mSec   68 21 21 68 07 05 7C 32 01 00 00 D5 D5 00 0E 00
05 -> 07                  06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                          04 00 10 00 00 45 16

SC ACK       2.377 mSec   E5

Poll         5.284 mSec   10 07 05 5C 68 16
05 -> 07

SD2 Resp     2.395 mSec   68 12 12 68 05 07 08 32 03 00 00 D5 D5 00 02 00
07 -> 05                  01 00 00 05 01 FF FB 16

FDL Req      5.320 mSec   10 07 05 49 55 16
05 -> 07

FDL Resp     2.390 mSec   10 05 07 00 0C 16
07 -> 05

Token Pass   5.294 mSec   DC 32 05
05 -> 32
```

The TD 200 still does not have a request so it passes the token back.

```
FDL Req      5.522 mSec   10 35 32 49 B0 16
32 -> 35

Token Pass   30.565 mSec  DC 05 32
32 -> 05

SD2 Req      5.281 mSec   68 21 21 68 07 05 7C 32 01 00 00 D6 D6 00 0E 00
05 -> 07                  06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                          04 00 10 00 00 47 16

SC ACK       2.369 mSec   E5

Poll         5.274 mSec   10 07 05 5C 68 16
05 -> 07

SD2 Resp     2.413 mSec   68 12 12 68 05 07 08 32 03 00 00 D6 D6 00 02 00
07 -> 05                  01 00 00 05 01 FF FD 16

FDL Req      5.320 mSec   10 08 05 49 56 16
05 -> 08
```

```
Token Pass   26.559 mSec   DC 32 05
05 -> 32
```

The TD 200 issues a read request for ten bytes starting at V0. The response data is 54 44 10....6E.

```
SD2 Req      5.530 mSec   68 1B 1B 68 02 32 7C 32 01 00 00 03 03 00 0E 00
32 -> 02                  00 04 01 12 0A 10 02 00 0A 00 01 84 00 00 00 B9
                        16

SC ACK       2.284 mSec   E5

Poll       5.268 mSec   10 02 32 5C 90 16
32 -> 02

SD2 Resp     2.324 mSec   68 1F 1F 68 32 02 08 32 03 00 00 03 03 00 02 00
02 -> 32                  0E 00 00 04 01 FF 04 00 50 54 44 10 30 04 00 00
                        0A 00 6E 33 16

FDL Req      5.397 mSec   10 37 32 49 B2 16
32 -> 37

Token Pass   30.557 mSec   DC 05 32
32 -> 05

SD2 Req      5.379 mSec   68 21 21 68 07 05 7C 32 01 00 00 D8 D8 00 0E 00
05 -> 07                  06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                        04 00 10 00 00 4B 16

SC ACK       2.382 mSec   E5

Poll       5.253 mSec   10 07 05 5C 68 16
05 -> 07

SD2 Resp     2.437 mSec   68 12 12 68 05 07 08 32 03 00 00 D8 D8 00 02 00
07 -> 05                  01 00 00 05 01 FF 01 16

FDL Req      5.300 mSec   10 0A 05 49 58 16
05 -> 0A

Token Pass   26.550 mSec   DC 32 05
05 -> 32
```

The TD 200 issues a read request for one byte at V110. The response data is 8F.

```
SD2 Req      5.462 mSec   68 1B 1B 68 02 32 7C 32 01 00 00 04 04 00 0E 00
32 -> 02                  00 04 01 12 0A 10 02 00 01 00 01 84 00 03 70 25
                        16
```

```
SC ACK      2.351 mSec  E5

Poll        5.306 mSec  10 02 32 5C 90 16
32 -> 02

SD2 Resp    2.286 mSec  68 16 16 68 32 02 08 32 03 00 00 04 04 00 02 00
02 -> 32                05 00 00 04 01 FF 04 00 08 8F 1F 16

FDL Req     5.299 mSec  10 38 32 49 B3 16
32 -> 38

Token Pass  30.559 mSec DC 05 32
32 -> 05
```

Another write from the S7-214 master.

```
SD2 Req     5.346 mSec  68 21 21 68 07 05 7C 32 01 00 00 D9 D9 00 0E 00
05 -> 07                06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                        04 00 10 00 00 4D 16

SC ACK      2.430 mSec  E5

Poll        5.213 mSec  10 07 05 5C 68 16
05 -> 07

SD2 Resp    2.367 mSec  68 12 12 68 05 07 08 32 03 00 00 D9 D9 00 02 00
07 -> 05                01 00 00 05 01 FF 03 16

FDL Req     5.262 mSec  10 0B 05 49 59 16
05 -> 0B

Token Pass  26.559 mSec DC 32 05
05 -> 32
```

The TD 200 issues a read request to read one byte at M0.

```
SD2 Req     5.502 mSec  68 1B 1B 68 02 32 7C 32 01 00 00 05 05 00 0E 00
32 -> 02                00 04 01 12 0A 10 02 00 01 00 00 83 00 00 00 B2
                        16

SC ACK      2.331 mSec  E5

Poll        5.352 mSec  10 02 32 5C 90 16
32 -> 02

SD2 Resp    2.263 mSec  68 16 16 68 32 02 08 32 03 00 00 05 05 00 02 00
02 -> 32                05 00 00 04 01 FF 04 00 08 00 92 16
```

```
FDL Req     5.326 mSec   10 39 32 49 B4 16
32 -> 39

Token Pass  30.555 mSec  DC 05 32
32 -> 05
```

Another write from the S7-214 master.

```
SD2 Req     5.320 mSec   68 21 21 68 07 05 7C 32 01 00 00 DA DA 00 0E 00
05 -> 07                 06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                      04 00 10 00 00 4F 16

SC ACK      2.371 mSec   E5

Poll        5.274 mSec   10 07 05 5C 68 16
05 -> 07

SD2 Resp    2.406 mSec   68 12 12 68 05 07 08 32 03 00 00 DA DA 00 02 00
07 -> 05                 01 00 00 05 01 FF 05 16

FDL Req     5.231 mSec   10 0C 05 49 5A 16
05 -> 0C

Token Pass  26.558 mSec  DC 32 05
05 -> 32
```

The TD 200 reads one byte at V110 and again receives 8F as response data.

```
SD2 Req     5.527 mSec   68 1B 1B 68 02 32 7C 32 01 00 00 06 06 00 0E 00
32 -> 02                 00 04 01 12 0A 10 02 00 01 00 01 84 00 03 70 29
                      16

SC ACK      2.298 mSec   E5

Poll        5.255 mSec   10 02 32 5C 90 16
32 -> 02

SD2 Resp    2.338 mSec   68 16 16 68 32 02 08 32 03 00 00 06 06 00 02 00
02 -> 32                 05 00 00 04 01 FF 04 00 08 8F 23 16

FDL Req     5.351 mSec   10 3A 32 49 B5 16
32 -> 3A

Token Pass  30.596 mSec  DC 05 32
32 -> 05
```

Another write from the S7-214 master.

```
SD2 Req      5.384 mSec   68 21 21 68 07 05 7C 32 01 00 00 DB DB 00 0E 00
05 -> 07                  06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                       04 00 10 00 00 51 16

SC ACK       2.397 mSec   E5

Poll       5.237 mSec   10 07 05 5C 68 16
05 -> 07

SD2 Resp     2.442 mSec   68 12 12 68 05 07 08 32 03 00 00 DB DB 00 02 00
07 -> 05                  01 00 00 05 01 FF 07 16

FDL Req      5.293 mSec   10 0D 05 49 5B 16
05 -> 0D

Token Pass   26.557 mSec   DC 32 05
05 -> 32
```

Another write from the S7-214 master.

```
SD2 Req      5.351 mSec   68 21 21 68 07 05 7C 32 01 00 00 DC DC 00 0E 00
05 -> 07                  06 05 01 12 0A 10 02 00 02 00 00 82 00 00 00 00
                       04 00 10 00 00 53 16

SC ACK       2.426 mSec   E5

Poll       5.211 mSec   10 07 05 5C 68 16
05 -> 07

SD2 Resp     2.379 mSec   68 12 12 68 05 07 08 32 03 00 00 DC DC 00 02 00
07 -> 05                  01 00 00 05 01 FF 09 16

FDL Req      5.254 mSec   10 0E 05 49 5C 16
05 -> 0E

Token Pass   26.549 mSec   DC 32 05
05 -> 32
```

The TD 200 reads 20 bytes starting at V10. The response data is 50 52...85.

```
SD2 Req      5.482 mSec   68 1B 1B 68 02 32 7C 32 01 00 00 08 08 00 0E 00
32 -> 02                  00 04 01 12 0A 10 02 00 14 00 01 84 00 00 50 1D
                       16
```

```
SC ACK       2.358 mSec  E5

Poll         5.299 mSec  10 02 32 5C 90 16
32 -> 02

SD2 Resp     2.284 mSec  68 29 29 68 32 02 08 32 03 00 00 08 08 00 02 00
02 -> 32                 18 00 00 04 01 FF 04 00 A0 50 52 45 53 53 55 52
                         45 3D 20 20 20 20 20 10 53 46 40 EB 85 F2 16

FDL Req      5.320 mSec  10 3C 32 49 B7 16
32 -> 3C

Token Pass   30.751 mSec DC 05 32
32 -> 05
```

Appendix C Medium Access Methods and Transmission Protocol

C1 Profibus Overview

This appendix is essentially a partial rewrite from DIN 19 245, Section 4, Medium Access Methods and Transmission Protocol(Data Link Layer, FDL).

Profibus uses a decentralized token passing method in conjunction with the Master-Slave principle. Medium access control is practiced by Master(active) stations while Slave(passive) stations are neutral with regard to medium access control, i.e., they transmit only on request from a Master. Communication is always initiated by the master station which holds the token. The token is passed from master station to master station in a logical ring and thus determines the instant in which any master may exercise control of the medium. Token passing is controlled by the master stations in a fixed manner requiring each master to know its ring predecessor(Previous Station - PS), its ring successor(Next Station - NS) and its own address(This Station - TS). Each master station determines its PS and NS on first-time initialization of the logical ring and further by an algorithm for dynamic addition and removal of ring stations. If the logical ring consists of only one master station and several slave stations, it is considered to be a pure master - slave ring.

Master stations must deal with the following error conditions, exceptions and operational states of the logical ring:

1. multiple tokens
2. lost token
3. error(s) in token passing
4. duplicate stations addresses
5. stations having faulty transmitters and/or receivers
6. all combinations of master and slave stations

C2 Fieldbus Data Link(FDL) Controller and Transmission Procedures

Exchanging of messages on the ring occurs in cycles. One message cycle consists of a request or send/request action frame from a master and the acknowledgment or response frame from another master or slave. User data may be transmitted in both the send frame and the response frame while an acknowledgment frame contains no user data. The complete message cycle is interrupted only for token passing or for broadcast messages, i.e., transmitting data with no acknowledgment. A master initiating a broadcast message uses the address of the highest station address consisting of all address bits being set to 1(address 127).

All stations with the exception of the token-holding master shall generally monitor all requests. Stations send an acknowledgment or response only when they are addressed and the acknowledgment or response must arrive within the defined ***slot time(Tslot)***. If this is not a "first request" to a station(see discussion of FCBs in section A9.3), and the acknowledgment/response is not received within the slot time, the master repeats the request. A retry or new request shall not be initiated by the master before the expiration of an ***idle time(Tidle)***. If the addressed station does not acknowledgment/respond within a specified number of retries, it shall be marked as "non-operational". Any unsuccessful request to a station which is marked as "non-operational" shall not be repeated.

Four types of transmission operations determine the time sequence of message cycles:

1. token handling
2. acyclic request or send/request operation
3. cyclic send/request operation, polling
4. station registration

C2.1 Procedures for Token Handling

C2.1.1 Token Passing

The token is passed from master to master in ascending address order except in the case of the station with the highest address in which case the token is passed to the station with the lowest address to close the ring. The token frame is used to pass the token.

Reception of the Token:

When a master station receives a token frame with its address(TS) in the destination address and the source address is from a master registered as the PS in its LAS(List of Active Stations - LAS), this station(TS) holds the token and may execute message cycles. The LAS is constructed by the master in the ***Listen_Token state***(see state diagram in Appendix A and description in A2.2) after power-up. The LAS is updated and corrected as necessary after receiving the token during operation. In the case that a token is received from a master station which is not the PS according to the LAS, the receiving master shall treat this as an error and not accept the token. On a subsequent retry to pass the token by the same PS, the token is accepted by the addressed master(TS) and the assumption is made that the logical ring has changed resulting in the old PS in the LAS being replaced by this new PS.

Transmission of the Token:

When the master station has finished its message cycles including maintenance of the GAP Station List (GAPL), the token is passed to its NS as shown in its LAS. During transmission of the token frame, transceiver operation is monitored while in this ***Pass_Token state***(see state in Appendix A and description in A2.2).

After transmission of the token frame by TS, if TS receives a valid frame, i.e. a plausible frame header with no errors, after expiration of the ***Sync Time(Tsync)*** but within the ***Slot Time(Tslot)***, TS assumes that its NS to which the token was passed now owns the token and may execute message cycles. If TS receives an invalid frame, it assumes that another master has control of the bus. In either case, TS ceases monitoring the token-passing and enters the ***Active_Idle state***(see state diagram in Appendix A and description in A2.2). If TS detects no bus activity within ***Tslot*** time, it retries the token pass and waits another ***Tslot*** time. If bus activity is detected within this 2nd ***Tslot*** time, TS ceases monitoring the token-passing and enters the ***Active_Idle state***. If no bus activity is detected during this 2nd ***Tslot*** time, TS retries passing the token to its NS one last time. If TS recognizes bus activity during this second retry ***Tslot*** time, TS ceases monitoring the token-passing and enters the ***Active_Idle state***. If no bus activity is detected during this second retry ***Tslot*** time, TS attempts to pass the token to the next master station in its LAS. The procedure is repeated until TS is able to pass the token to a master station in its LAS. If TS is not able to pass the token to a master station in its LAS, it assumes that it is the only master station on the logical ring and either keeps the token and executes message cycles or if no message cycles are necessary, TS transmits the token to itself.

C2.1.2 Addition and Removal of Slave and Master Stations

Stations, either slave or master, may be connected to or disconnected from the transmission medium at any time. Each master station in the logical ring is responsible for addition of new stations and removal of existing stations within the address range of TS to NS. This address range is know as the GAP and is represented in the GAP List(GAPL). The GAP does not include the address range between the Highest Station Address(HSA) and address 127. Each master station in the ring periodically examines its GAP addresses for addition/removal of master/slave stations. The GAP address range interrogation occurs during the ***GAP Update Time(Tgud)*** interval and is accomplished by interrogating one address per receipt of the token using the "Request FDL Status" action frame.

When the token is received, GAP maintenance begins immediately following the completion of all queued message cycles given that there is still transmission time available(see A2.1.4). If there is no transmission time available, GAP maintenance begins at the next or consecutive token receipts immediately following high priority message cycles. The master implementation should ensure that low priority message cycles and GAP maintenance do not block one another.

GAP addresses are interrogated in ascending order except for the GAP surpassing the HSA. The HSA and address 0 are not used by a master station. Address 0 is reserved for a programming device. The HSA should only be a slave device. The GAP update continues at address 0 after checking the HSA. If the "Request FDL Status" is acknowledged positively with either the state "not ready" or "slave station", it is marked in the GAPL and the next address is interrogated on the next token hold. If a station acknowledges with the state "ready to enter logical token ring", TS modifies its GAP or GAPL and passes he token to this new NS. This new NS has already built its LAS while in the ***Listen_Token state***(see state diagram in Appendix A and description in A2.2). Passive stations registered in the GAPL and which repeatedly fail to respond to "Request FDL Status" are removed from the GAPL and noted as unused station addresses.

C2.1.3 Initializing/Re-Initializing the Logical Token Ring

Initializing is a special case of updating the LAS and the GAPL. After power on of a master station into the ***Listen_Token state***(see state diagram in Appendix A and description in A2.2), if no bus traffic is detected within ***Timeout Time(Ttout)***, TS shall claim the token by transitioning into the ***Claim_Token state***(see state diagram in Appendix A and description in A2.2) and start initializing by passing the token to itself. When the entire Profibus is initialized, the master station with the lowest address starts initialization. The station transmits one token frame addressed to itself to notify any other master stations that it is the only master station in the logical token ring. The other stations are then brought into the logical token ring through subsequent normal GAP maintenance.

When the token is lost, it becomes necessary to re-initialize the logical token ring. In the case of a lost token, the LAS and GAPL already exist in each of the master stations and an entire Profibus initialization cycle is not necessary. A timeout occurs first on the master with the lowest station address and this station then takes the token and begins executing message cycles or passes the token to its NS.

C2.1.4 Token Rotation Time

After a master station receives the token, measurement of token rotation time begins. Measurement of the rotation time ends with the next reception of the token. This measured time gives ***Real Rotation Time(Trealr)***. At the end of one measurement, measurement of the token rotation time for this next cycle begins. ***Trealr*** is of importance in carrying out low priority message cycles.

To keep within the required system reaction time, the ***Target Rotation Time(Ttargr)*** of the token in the logical token ring shall be defined. The time ***Ttargr*** is the time it should take for the token to complete one cycle around the network.

Independent of ***Trealr***, each master station may always execute one high priority message cycle per token receipt. To be able to perform low priority message cycles during a token receipt, the condition of ***Trealr < Ttargr*** must exist at the time to execute low priority message cycles. If this is not the case, the master station must retain low priority message cycles and complete them at the next or subsequent token receipts.

The ***Ttargr*** of a system is dependent upon the number of master stations and therefore upon the number of token cycles and the ***Token Cycle Time(Ttcycle)*** as well as the time duration of ***High Priority Message Cycles(high Tmcycle)***. The predefined ***Ttargr*** must also include sufficient time for execution of low priority message cycles as well as for possible retries.

To achieve a minimum ***Ttargr***, it is recommended that only very important and rarely-occurring events be considered as high priority message cycles. Further, it is recommended that the length of such high priority message cycles be kept to ≤ 20 frame characters in the data-unit of a frame. Using the cycle times for ***Token Cycle Time(Ttcycle)*** and ***Message Cycle Time(Tmcycle)*** defined in A.3 and allowing for possible retries, the ***Ttargr*** which is necessary for initialization is:

Minimum *Ttargr* =
na x (*Ttcycle* + ***high Tmcycle***) + k x ***low Tmcycle*** + mt x ***retry Tmcycle***, where

na	number of master stations
k	est. number of low priority msg. cycles/token rotation
Ttcycle	token cycle time
Tmcycle	message cycle time(depending upon frame length; see A3.2)
mt	number of msg. retry cycles/token rotation
retry Tmcycle	msg. retry cycle time

The first term accounts for one high priority message cycle per master station and token rotation. The second term accounts for the estimated number of low priority message cycles per token rotation. The last term accounts for message retries.

C2.1.5 Priorities of Messages

The user of the FDL interface, i.e., the application layer, may choose two priorities in the service classes of message cycles: low priority and high priority. The priority is passed to the FDL with the service request.

Upon receipt of the token, a master station performs all high priority message cycles first followed by the low priority ones. Upon receipt of the token, if ***Trealr*** $\geq$ ***Ttargr***, only one high priority message cycle including retries may be performed and the token shall then be passed to the NS.

After receipt of the token or after completion of the first high priority message cycle, the following procedure should be considered. High priority message cycles may be executed only if at the beginning of the execution ***Trealr*** < ***Ttargr*** meaning that the ***Token Hold Time(Tthold)*** = ***Ttargr*** - ***Trealr*** is still available. Once started, a high priority or low priority message cycle is always completed including any retries necessary even if ***Trealr*** equals or exceeds ***Ttargr*** during execution of the message cycle. If ***Tthold*** becomes prolonged because of this, the transmission time available for message cycles upon the next token receipt is automatically shortened.

C2.2 Fieldbus Data Link(FDL) Controller States

The FDL controller of a master station behaves as described by 10-FDL states and associated transitions. A slave station behaves as described by 2 FDL states and associated transitions. Appendix A gives the combined(master and slave) state diagram. A master station may be in the following states: ***Offline, Listen_Token, Active_Idle, Claim_Token, Use_Token, Await_Data_Response, Check_Acccess_Time, Await_Status_Response, Pass_Token, Check_Token_Pass.*** A slave station may be in the following states: ***Offline, Passive_Idle.***

C2.2.1 Offline

The ***Offline*** state is entered immediately after power up or after certain errors have been detected. Each station should perform internal diagnostics and remain in the ***Offline*** state until the operating parameters(see Appendix B) have been initialized. The FDL may then connect to the transmission medium in receive mode.

C2.2.2 Passive_Idle

After a slave station's parameters are initialized, the FDL enters the ***Passive_Idle*** state and listens to the line. If a plausible action(send/request) frame addressed to this station is received, the FDL shall acknowledge or respond as required except for broadcast messages or for token frames addressed to this station. Token frames addressed to this station are discarded.

C2.2.3 Listen_Token

After the operating parameters have been initialized by a master station, the FDL transitions into the ***Listen_Token*** state if it is ready to enter the logical token ring. In this state, the FDL shall monitor the line to identify other master stations already in the ring. Token frames are analyzed and the station addressed are used to construct this station's LAS. The master station shall listen to two complete, identical token rotations during LAS generation and then remain in the ***Listen_Token*** state until it receives a "Request FDL Status" frame from its PS. The master shall respond with "ready to enter logical token ring" and then transition into the ***Active_Idle*** state. During LAS generation, the master station shall respond to any "Request FDL Status" frame with a "not ready" response. Any other frame received in the ***Listen_Token*** state is neither acknowledged nor answered.

If the FDL recognizes its own address as the source address in two token frames while registering other master stations, it shall assume that another master station with the same address is already in the ring and shall transition back into the ***Offline*** state.

If no bus activity is observed by the FDL for the timeout period(***Ttout***), it shall assume that an initialization/re-initialization of the logical token ring is necessary. It shall attempt to claim the token and carry out the initialization/re-initialization of the logical token ring. ***Ttout*** is defined as:

> ***Ttout*** = (6 * ***Tslot***) + (2 * n * ***Tslot***), where
> n = station address of TS
> ***Tslot*** = the time a master waits for a response to start(~ 100 bit times)

C2.2.4 Active_Idle

The master station's FDL shall transition from the ***Listen_Token*** state into the ***Active_Idle*** state and monitor the bus without becoming active. If the FDL receives a plausible action frame addressed to it, it acknowledges or responds as required. Upon receiving a token addressed to itself, the FDL transitions into the ***Use_Token*** state if it wants to remain in the logical token ring; otherwise, it transitions back into the ***Listen_Token*** state. The ***Listen_Token*** state shall also be entered in case two token frames in immediate succession with the source address = TS are received.

If no bus activity is observed for the timeout period(***Ttout***), the FDL shall assume that an initialization/re-initialization of the logical token ring is necessary. It shall attempt to claim the token and carry out the initialization/re-initialization of the logical token ring.

C2.2.5 Claim_Token

The FDL enters the ***Claim_Token*** state following a timeout expiration while in the ***Active_Idle*** state or the ***Listen_Token*** state. In this state, it shall attempt to carry out the initialization/re-initialization of the logical token ring. When re-initializing, the LAS and GAPL are still intact and it shall immediately transition into the ***Use_Token*** state. When initializing, the FDL shall first send the token twice to itself, i.e., NS = TS, in the ***Pass_Token*** state. This is done to allow other master stations to make an LAS entry. Following the token transmissions to itself, the FDL shall create its own LAS and GAPL by addressing "Request FDL Status" requests to the following station addresses.

C2.2.6 Use_Token

The FDL shall enter the ***Use_Token*** state upon receipt of the token or after initialization/re-initialization. In this state, high priority and low priority message cycles may be executed.

Upon entering this state, the FDL shall read the ***Trealr*** timer and restart the timer. One high priority message cycle is always allowed. An additional high priority message cycle or a low priority message cycle or generally a low priority message cycle may only be executed if ***Trealr*** < ***Ttargr*** at time for execution.

Upon transmission of each action frame, the FDL shall enter the ***Await_Data_Response*** state and start the ***Tslot*** slot timer. The FDL may transition back into the ***Use_Token*** state after the transmitted frame has been acknowledged by the receiver. If no high priority message cycle is to be performed upon entry into the ***Use_Token*** state or after completion of a high priority or low priority message cycle in that state, the FDL shall transition into the ***Check_Access_Time*** state.

C2.2.7 Await_Data_Response

This state is entered following the transmission of an action frame. The FDL controller waits up to ***Tslot*** time for receipt of the response or acknowledgment. No acknowledgment is awaited in the case of an SDN(Send Date with No Acknowledgment) service and the FDL transitions back into the ***Use_Token*** state. If a response or acknowledgment is expected, the FDL awaits one of the following events:

1) a response or acknowledgment frame addressed to TS
2) another frame(token or action frame)
3) an invalid frame or expiration of ***Tslot*** time

Following the receipt/processing of an acknowledgment or response frame, the FDL controller shall transition back into the ***Use_Token*** state to process any further service requests.

Receipt of another frame indicates an error condition and the FDL shall transition into the ***Active_Idle*** state and discard the received frame.

Receipt of an invalid frame or expiration of to ***Tslot*** time shall cause the FDL to retry the action frame. If no valid response or acknowledgment frame is received after the allowed number of retries to the station, the FDL shall declare an error and transition back into the ***Use_Token*** state. Further requests to this station are not retried in the case of error until a correct message cycle(Send/Request Data) has been completed to the station.

C2.2.8 Check_Access_Time

This state is used for computing the available ***Token Hold Time(Tthold) = Ttargr - Trealr***. After computing ***Tthold***, the FDL may transition back into the ***Use_Token*** state only if there is still token holding time available. If there is no token holding time available, the FDL shall transition into the ***Pass_Token*** state.

C2.2.9 Pass_Token

In this state, the FDL controller shall attempt to pass the token to its NS in the ring. The transmission of the token shall be monitored to ensure that the transmitter and receiver are working correctly. If TS does not receive its own transmitted token frame, either the receiver or the transmitter must not be functioning. The FDL shall remove itself from ring activity, declare an error and transition into the ***Offline*** state. If TS receives its own token frame corrupted, it may be because of the receiver, transmitter or the bus itself. In this case, the FDL shall enter the ***Check_Token_Pass*** state just as after receiving its own token frame correctly. If the token frame is re-transmitted because of no activity from its NS in the ring and the token transmission is monitored as being incorrect, the FDL shall remove itself from ring activity, declare an error and transition into the ***Listen_Token*** state.

If there is still token holding time*(Tthold)* available when the ***GAP Update Time(Tgud)*** has expired, the FDL shall, if necessary, attempt adding one new station into its LAS prior to passing the token. The FDL transmits a "Request FDL Status" request to a station in its GAP and transitions into the ***Await_Status_Response*** state. If a new master station acknowledges with a "ready to enter logical token ring", it is added to the LAS and the token is passed to this new NS. After successful passing of the token, the GAP of TS is shortened to the new station.

If the status request is acknowledged by a new slave station or by a new master with a "not ready" status, the station address is entered into the GAPL. If an existing station does not respond after a retry, it is deleted from the GAPL, i.e., it is marked as an unused address, so that retries are no longer sent to that address.

If no new master station answers during this GAP maintenance, the FDL passes the token to its original NS and transitions into the ***Check_Token_Pass*** state. If no other successor is known, i.e., TS is the only known active station in the ring, the FDL shall pass the token to itself and transition back into the ***Use_Token*** state.

C2.2.10 Check_Token_Pass

In this state, the FDL waits ***Tslot*** time for the station to which it has passed the token to react. The wait time allows for the delay between receipt of the token frame and any following transmission reaction of the addressed station.

If a valid frame header is detected within ***Tslot*** time, the FDL assumes that the token was passed successfully. This frame is to be processed as though it were received in the ***Active_Idle*** state and the FDL is to enter this state.

If an invalid frame is detected within ***Tslot*** time, the FDL shall assume that another master is active and shall also transition into the ***Active_Idle*** state.

If no frame is received within ***Tslot*** time, the FDL shall transition into the ***Pass_Token*** state. Processing shall follow the description in A2.1.1(Token Passing).

C2.2.11 Await_Status_Response

This state is transitioned into from the ***Pass_Token*** state during initialization or GAP maintenance after a status request has been addressed to a successor not in the LAS. The FDL shall wait for ***Tslot*** time for an acknowledgment frame. If either no frame or a corrupted frame is received, the FDL shall transition into the ***Pass_Token*** state where the request can be repeated or the token can be passed to itself or to its known NS.

If any other frame than an acknowledgment is received(indicating the possible existence of multiple tokens), the FDL shall transition into the ***Active_Idle*** state.

C2.3 FDL Initialization

After power up actions, the FDL controller of both master and slave stations shall enter the ***Offline*** state. In this state, the FDL controller neither receives nor transmits frames from/to the bus. After initializing the operating parameters, the FDL controller may transition into the ***Passive_Idle*** state(slave) or the ***Listen_Token*** state(master).

Appendix D Recommended 3rd Party Device Testing

This section outlines system testing recommended to be performed with the 3rd party device(s). Install the MPI card into the PC.

Connect the PPI/PC cable from your PC communication port to the S7-214 CPU.

Bring up Micro/DOS and enter the Micro/DOS program given at the end of this appendix.

Set the CPU network address of the S7-214 to 10 using the Station Address utility.

Download the Program Block (OB1) to the S7-214 using the utilities.

Network the CPU 212 and CPU 214 together with the network connectors provided as indicated by the documentation enclosed in the network connector box.

Connect the CPU 212 to power and place the CPU 212 into RUN mode by positioning the switch under the top flap.

Connect the CPU 214 to power and place the CPU 214 into RUN mode by positioning the switch under the top flap.

With the CPU 212(preset at address 2) and CPU 214(at address 10) networked together and with the CPU 214 in RUN mode executing the downloaded program and with the 3rd party device(s) connected into the network, perform the following tests. Any network errors detected by the CPU 214 will be indicated by incrementing data byte Q0(visible in the output points Q0.0 - Q0.7) or by timeouts on the 3rd party device

- Perform valid reads/writes of all CPU 212 and CPU 214 data types identified in the PPI+ specification.
- Perform attempted out-of-range access to verify error-handling to at least one data type on both the CPU 212 and CPU 214.
- Perform a valid time-of-day clock read on the CPU 214.
- Perform a valid time-of-day clock write to the CPU 214.
- Perform attempted time-of-day clock read on CPU 212 to verify error-handling.
- Perform attempted time-of-day clock write to CPU 212 to verify error-handling.
- Perform attempted time-of-day clock write to CPU 214 with non-BCD data to verify error-handling.
- Perform reads(via a Micro/DOS chart) of the 8-word data area VW100 - VW114 on both the CPU 214 and the CPU 212 to verify that the locations on both CPUs are continuously incrementing. The CPU 214 is incrementing its own 8-word data area VW100 - VW114 and then writing/reading back the corresponding data area on the CPU 212.

- Randomly power-cycle the 3rd party device(s) and the CPU 214(both network masters) while performing data reads/writes with the 3rd party device to verify that the network resumes normal operation, i.e., recovers from lost token, etc.

- Perform program downloads and uploads from the programming package(using the MPI card interface) to either S7-200 CPU while the network is being exercised and while the 3rd party device is performing data reads/writes. Note the occurrence of errors on the S7-214 master or the 3rd party device or the programming package. *Note that to download to a CPU the mode switch must be in the TERM position.*

Note that the 3rd party device must have the capability of setting its network address.

```
NETWORK1
    Networking program between S7-214 and S7-212 @ address 2
    Area of incrementing 16-bit counters is VW100 - VW114
    NETW control table is at VB300
    NETR control table is at VB200
    First scan initialization
LD          SM0.1
CALL        K0
JMP         K0
NETWORK2
    Initiate first NET write operation to station 2
LD          M0.0
BMW         VW100  VW307  K8
NETW        VB300  K0
R           M0.0    K1
JMP         K0
NETWORK3
    Check for NETW operation completed or errors
    If completed with no errors, read back data from station
    no. 2 & compare with data written
LD          V300.7
EU
NOT
JMP         K2
NETWORK4
     Check for an error
     On error, increment error counter & output display
      Increment data area counters & initiate next NETW
LD          V300.5
CALL        K4
CALL
JMP         K0
NOT
CALL        K3
JMP         K0
LBL         K2
NETWORK5
    Check for NETR operation just completed
LD          V200.7
EU
NOT
JMP         K0
NETWORK6
     Check for an error
     On error, increment error counter
     If no error, compare data read with that writeen
LD          V200.5
CALL        K4
JMP         K1
NOT
CALL        K2
NETWORK7
     Check results of compare
    On error, increment error counter & display
    Increment data area counters & initiaate next NETW
LD          M0.1
NOT
CALL        K4
LBL         K1
NETWORK8
    Initiate NETW
LD          SM0.0
CALL        K1
LBL         K0
MEND
NETWORK9
    1st scan init. routine
SBR         K0
NETWORK10
    Clear counter words & error counter
    Set NETW & NETR status to "done"
LD          SM0.0
FILL        KH0     VW100 8
MOVW        KH0     QW0
MOVB        KH80    VB200
MOVB        KH80    VB300
MOVB        KH2     VB201
MOVB        KH2     VB301
MOVD        &VB100          VD202
MOVD        &VB100          VD302
MOVB        KH10    VB206
MOVB        KH10    VB306
S           SM30.1 K1
S           M0.0    K1
RET
NETWORK11
SBR         K1
NETWORK12
    Incement counter words & initiate write
INCW        VW100
INCW        VW102
INCW        VW104
INCW        VW106
INCW        VW108
INCW        VW110
INCW        VW112
INCW        VW114
BMW         VW100 VW307 K8
NETW        VB300  K0
RET
NETWORK13
SBR         K2
NETWORK14
    Compare data read with that xmitted
LDW=        VW100, VW207
AW=         VW102, VW209
AW=         VW104, VW211
```

```
AW=         VW106, VW213
AW=         VW108, VW215
AW=         VW110, VW217
AW=         VW112, VW219
AW=         VW114, VW221
=           M0.1
RET
NETWORK15
SBR         K3
NETWORK16
   Clear receive data area & initiate read
LD          SM0.0
FILL        KH0    VW207 K8
NETR        VB200  K0
RET
NETWORK17
SBR         K4
NETWORK18
   Increment error counter & display in outputs
LD          SM0.0
INCW        VW400                  // Increment error counter
MOVB        VB401, QB0             // & display in outputs
RET
```

DWG NO. 2805090	SH 1

APPLICATION	
	2267

REVISION

REV	DESCRIPTION	DATE	APPROVED
1.0	Formal Release		

UNLESS OTHERWISE SPECIFIED • DIMENSIONS ARE IN INCHES • TOLERANCES: ANGLE +/- 1 DEGREE 3 PLACE DECIMALS +/- .010 2 PLACE DECIMALS +/- .02 1 PLACE DECIMAL +/- .03	DWN R. Mitchell DATE 14-Aug-96	SIEMENS Siemens Industrial Automation Inc. *Johnson City, Tennessee*
	ENGR Steve Hausman	
	APVD J. Karczmarczyk, S7-200 Dev. & Mktng. Mgr.	TITLE Point-to-Point Interface (PPI), S7-200 Communication Protocol Specification for 3rd Party Products
	APVD Ned Cox, Prog. Mgr.	
	QA	
	CONF	SIZE A | DRAWING NUMBER 2805090
	RLSE	SCALE : NONE | SHEET 1 OF 74

DIST.