Design of a Token Ring Controller in IDaSS

Gerritsen, B.H.H.

Award date:
1993

Eindhoven Technical University of Eindhoven

Department of Electrical Engineering
Section of Digital Information Systems (EB)

Design of a Token Ring Controller in IDaSS

By : Ing. B.H.H. Gerritsen

Master Thesis Report

Supervisors : Prof. Ir. M.P.J. Stevens
Dr. Ir. A.C. Verschueren

Abstract

Since the beginning of the eighties the Local Area Networks have become increasingly popular. This was due to the fact that small computers became more powerful but resources where still expensive. The Local Area Network brought the possibility to connect computers which where placed in a relatively small (Local) area. With these high speed interconnections, computers became able to share the more expensive resources.
In those years the 802 group of the Institute of Electrical and Electronic Engineers (IEEE) worked on the standardization of the LAN. This led to the release of the 802 standard family in the mid-eighties. Not much later this standard was adopted by the American National Standards Institute (ANSI).

Several companies, like Texas Instruments and IBM, have developed Integrated Circuits that implement the functioning of the IEEE 802.5 standard.

The architecture for the implementation of this standard is specific designed for the Object-Oriented (Hardware) System Design project library. Therefore the architecture is bounded by several restrictions.
First of all, the architecture is designed in IDaSS. Because of the use of IDaSS it was not possible to implement the Phase Lock Loop with the acquired specifications. Instead the controller is equipped with a set of ring interface signals that are customary for controlling a Token Ring Interface Circuit. This external Ring Interface circuit is supposed to perform the phase lock looping, and the delivering of the received bits to, and the asking for the bits to transmit from the bit-processor, synchronous to the system clock of the controller.
Secondly the controller is provided with a system interface as specified in [VERS93]. All elements in the project library have this interface.

The controller consists of a bit- and a byte-processor. Roughly speaking, the bit-processor is charged with the reception and transmission of bits. The received bits are put together in bytes and forwarded to the byte-processor, where they can be processed. The byte-processor transfers bytes for transmission to the bit-processor, where they are broken down in bits, for transmission.

The byte-processor is something like a very minimum micro-processor using 32 addressable registers. In the 20 bits wide ROM 1K words of instructions can be loaded and executed. Most key operations take only one control word and one clock-cycle.
A DMA unit is provided for the transfer of frames with an external memory. Several registers in the byte-processor are duplicated to allow the receive and transmit process to have their own resources.
The byte-processor is able to access the bit-processor through six of its 32 registers. By writing to and reading from these registers, the byte-processor controls the bit-processor and receives/transmits frames.

On the other side the bit-processor signals the byte-processor when a particular receive event has taken place through a 4 bits event bus. Normally the byte-processor is in the transmit mode where it's handling the transmit process. When the byte-processor recognizes a receive event, as indicated on the event bus, it switches to the receive process. After the receive event has been processed it switches back to the transmit process.

The bit-processor performs the functioning of a receive and transmit process simultaneously, as both processes are implemented in separate finite-state machines.
Inside the bit-processor almost the complete functioning of the Operational Finite-State Machine, as specified in [IEEE802.5], is implemented. The byte-processor only has to perform the address recognition, and to control the Token Holding Timer.
When the station as Active Monitor or Standby Active monitor, detects an improper functioning of the ring, it must start a process of re-initialization of the ring. During this process the station transmits special MAC frames continuous, without regarding the access and priority mechanism of the Operational FSM. Therefore the bit-processor gives the possibility for the transmission of frames on direct command from the byte-processor.
The byte-processor only has to load the bytes of the FC, DA, SA and INFO fields of a frame into the transmit buffer of the bit-processor, for transmission. The SD, AC, FCS and ED are generated by the bit-processor. The bit-processor on his turn, only queues the bytes of the AC, FC, DA, SA, INFO and FS fields of a received frame, or the AC field of a token, in the receive buffer. The frame check sequence checking is performed in the bit-processor, as all possible error checking on frames and tokens.

Finally speaking, the monitor program for the byte-processor, that has to be designed, has to perform the following functions :
Initialization, which is setting the configuration of the bit-processor. Address recognition and controlling the THT timer, which are the only actions to perform in order to implement the Operational FSM completely. Implement the Standby and Active Monitor. Transfer frames to and from an external memory.

Contents

Preface

In front of you lays the result of my graduation project which took place from april to december 1993. This report is the finishing touch of my study Information-technology at the faculty of Electrical Engineering at the Eindhoven University of Technology. In this final piece of work I want to thank a few persons for supporting me in my past three and a half years of study.
As the graduation project is performed at the Section of Digital Information Systems at the faculty, I'm very grateful to the Section for giving me the change to finish my study. My special appreciation goes out to my supervisors prof.ir. M.P.J. Stevens and dr.ir. Verschueren for their advise and coaching along the side.
Of course I want to thank my friends who made my college life as good as it was. And last but not least my parents and Elise and Paul, for supporting me, and for whom all I have great respect.
Thanks to you all!!

Bas Gerritsen

Eindhoven 13 december 1993

Introduction

In this thesis report an architecture for the implementation of a Token Ring controller for the IEEE/ANSI standard 802.5 is presented. The controller is designed with the Interactive Design and Simulation System (IDaSS).

At the faculty of Electronics of the Eindhoven University of Technology the section of Digital Information Systems is working on the project "Object-Oriented (Hardware) System Design". An important part of this ongoing project is formed by the creation of a library of re-usable processor cores and input/output controllers designed in IDaSS. With this library, building a custom processor to perform specific tasks is eased to the point that only a few library components must be picked from the library, interconnected with IDaSS and converted to silicon with ASA.
The Token Ring Controller presented in this paper is designed for the project library. Therefore the interface of the controller follows the strict rules of the project library to be able to interconnect the controller to other elements of this library.

The first two chapters of this paper give an introduction into the world of Local Area Networks and the Token Ring in particular. These two chapters are added for the reader who is not well known with the Token Ring Local Area Network. For a thorough description of the Token Ring standard is referred to [IEEE802.5].
In the past years several Token Ring systems where designed, which were compatible with the standard ANSI/IEEE 802.5. The third chapter contains a summary of these systems to give an indication of the possibilities and the architecture of those systems.
In the fourth chapter the restrictions that bounded the design of the TRC, are specified. These restrictions are mainly caused by the fact that the TRC is designed for the project library.

The designed Token Ring controller can be divided in two parts. A byte and a bit-processor. The byte-processor is something like a minimum micro-processor executing a monitor program. This byte-processor was designed by dr. ir. Verschueren and was originally developed for a HDLC/SDLC controller. The architecture of the designed bit-processor and used byte-processor are discussed in chapter five. The monitor program which should be executed by the byte-processor, and control the bit-processor, has not been developed yet. Therefore chapter five also contains an extensive specification of the functioning an controlling of the bit-processor.

Finally the results of this graduation report are concluded.

The Appendices contain several back ground information. One Appendix contains the instruction set of the byte-processor. The other Appendices contain the document files of the schematics used in the bit-processor.
When this document was delivered for publishing, the elucidation of the used operators and registers in the several schematics was not finished. Readers interested in this information can contact the section of Digital Information Systems.

Chapter 1
Local Area Networks

Since the beginning of the eighties the Local Area Networks have become increasingly popular. This was due to the fact that small computers became more powerful but resources where still expensive. The Local Area Network brought the possibility to connect computers which where placed in a relatively small (Local) area. With these high speed interconnections, computers became able to share the more expensive resources.
In those years the 802 group of the Institute of Electrical and Electronic Engineers (IEEE) worked on the standardization of the LAN. This led to the release of the 802 standard family in the mid-eighties. Not much later this standard was adopted by the American National Standards Institute (ANSI).

1.1 The ANSI/IEEE 802 Standard Family

The 802 standards specify the Datalink and Physical layer as defined by the Open Systems Interconnection (OSI) reference model of the International Standard Organization (ISO). The 802 standards although, divides the Datalink layer into a Medium Access Control and Logical Link Control layer.
Figure 1.1 shows the relation between the OSI reference model and the IEEE/ANSI layer partitioning.

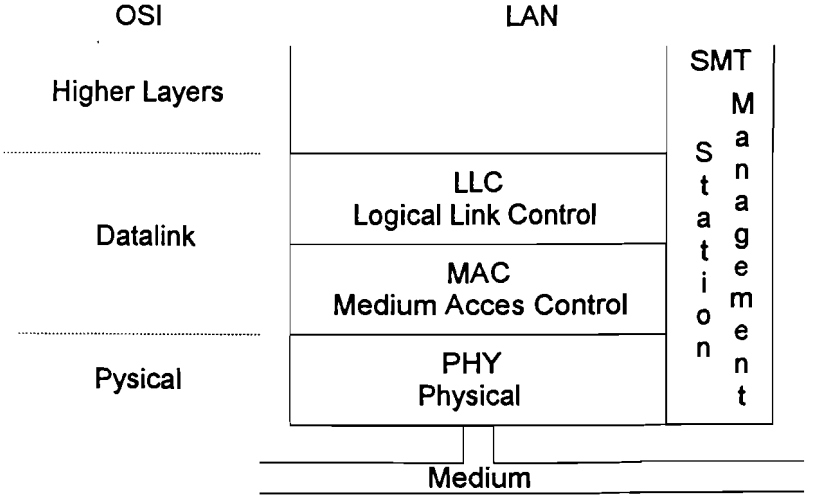


Figure 1.1 Relation between OSI Reference model and IEEE 802

Generally speaking, the MAC layer as defined by the IEEE/MAC standards specifies the functions for controlling the functioning and configuration of the physical medium and its attached stations.

Several LAN architectures are proposed in the 802 standards. One of them is the Token Ring architecture. This LAN is described by the following 802 standards :

- 802.1, which describes the relation between the LAN standards and the OSI reference model.
- 802.2, which describes the Logical Link Control (ref.[IEEE802.2])
- 802.5, which describes the Medium Access Control and the Physical layer.(ref.[IEEE802.5])

In this paper an implementation of the 802.5 standard is presented. The design of the implementation was made in IDaSS.

1.2 Token Ring : An Introduction

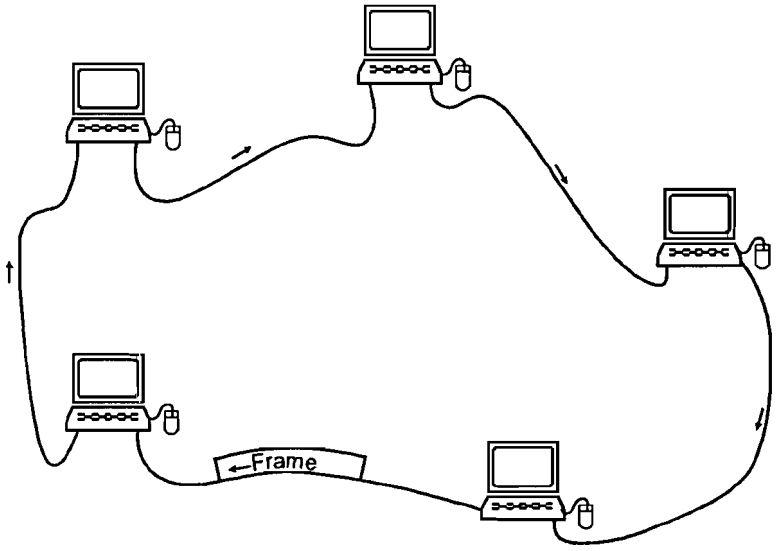


Figure 1.2 A Token Ring System

A Token Ring system consists of a set of stations connected to a transmission medium; A coax or twisted-pair, ring. Information between active stations is transferred sequentially, bit by bit. The basic function of a station is to regenerate and repeat each bit. If a station is the destination station it also copies the information while it is repeated. The station sending the information is responsible for removing the information from the ring, when it returns to the station. Therefore a station has two basic functions; it is either repeating the incoming data, or it is transmitting, and removing, its own information.

Information of a station is transmitted in frames. The Token Ring standard defines two types of frames; The Medium Access Control (MAC) frame and the Logical Link Control (LLC) frame. When a station receives a MAC frame it should respond as specified in the MAC layer. A LLC frame destinated to the station is simply passed by the MAC layer to the LLC layer.

Besides the frames a second type of information package called a token can circulate on the ring. A token is a special type of frame through which stations can gain the right to transmit frames. The method for accessing the ring will be outlined in the next paragraph. The sending station transmits fill, a random sequence of 0's and 1's, between the frames and tokens. All information, frames, tokens and fill, is transmitted in differential Manchester code. This leads to a transmission medium which is never silent, which is advantageous for timing characteristics.

1.2.1 Token Ring Access Method

Sending by a station point of view, is divided into three steps, which are:

1. capturing the token
2. sending the frames
3. releasing the token.

A station gains the right to transmit its information when it detects a token passing on the medium. When a station detects an appropriate token it changes it into a Start of Frame Sequence (SFS), this is called capturing the token, and appends the necessary control, status, address and information fields.
A station is allowed to transmit information until it has no more information to transmit or a predefined time limit, called the Token Hold Time (THT), will be violated.
If one of these situations occurs the station initiates a new token.

1.2.2 Priorities

The term "appropriate token" is not just for any reason. Gaining the right to transmit is based on priorities. The token, as well as the frames, contain an Access Control (AC) field. This field contains three priority, and three reservation bits.
A station may capture a token if its priority, represented by the three priority bits, is equal to or lower than the priority of the information it wants to transmit. The priority of the transmitted frames are always equal to the priority of the token the sending station captured.
If a station wants to transmit information and detects a token with a higher priority, or a frame, it sets the reservation bits. A station is only allowed to do this if the priority in the reservation bits is lower as the priority of the information it wants to send.
The priority of the token released by a station, after transmitting, is based upon a mechanism which depends on the priority of the captured token and the reservation bits in the last transmitted and returned frame. Further on this mechanism will be outlined. Worth mentioning at this point is the responsibility of a station which upgraded the priority of the network is to degrade the priority of the network as well.

1.2.3 Network Management

In the 802.5 standard some Network Management functions are specified. These Network Management functions are specified to guarantee the correct functioning of each ring in a Token Ring Network.
The standard defines that each ring has a set of server stations (servers) which are the means through which the System Manager manages the stations in the network.
Servers are data collection and distribution points on the ring where reports from the data stations are gathered. The System Manager communicates the necessary information to and from the servers for the purpose of managing the ring. A station being a server should also have the functionality of a normal data station.
The data stations communicate with the servers by reporting detected errors, requesting operating parameters when inserting into the ring, reporting changes in the configuration, responding to requests for various status information or removal from the ring when requested.
Furthermore the 802.5 standard specifies the format and protocol of information interchange between servers and stations. However, for the communication between the servers and the System Manager only the abstract entities are specified. The System Manager is referred to as System Management (SMT).

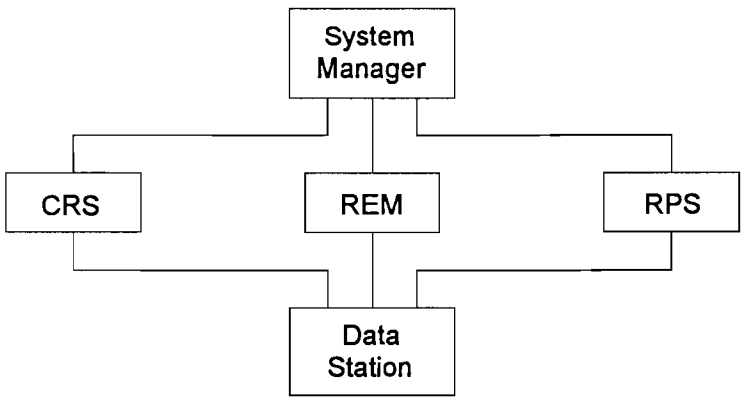


Figure 1.3 Relationship of Data Stations, Servers, and System Manager

As pointed out, there is a frame or a token circulating on the ring at all time. To maintain this continue presence of data, one of the stations in the ring system is the Active Monitor. The other stations are Standby Monitors. The function of the Active Monitor is to recover the ring from various error situations such as absence of validly formed frames or tokens.
After a station has been inserted in the ring it starts as a Standby Monitor. In this state the station continuously observes the tokens and frames on the ring as they are repeated. If tokens or (special) AMP frames are not detected periodically the Standby Monitor tries to claim the function of Active Monitor by transmitting special Claim Token frames.

Chapter 2
Token Ring

Before starting with the Token Ring design an introduction of the Token Ring standard is given. For those who are interested in all ins and outs of the design, reading [IEEE802.5] is strongly advised.

2.1 Formats and Facilities

The Token Ring standard defines the formats in which the information is transferred.
As pointed out in Chap. 1.2 the data is transmitted in Differential Manchester Code. Besides binary one's and zero's, a station can also transmit the non-data-symbols J and K. These symbols, which are code violation of the Differential Manchester Code, are used to indicate the start and end of the frame or token. Chapter 2.2 explains the differential Manchester Code.
For the functioning of the Token Ring protocols several facilities are defined. These facilities include MAC frames for controlling the ring system, timers, flags, and counters for detecting events in the ring system.

2.1.1 Formats

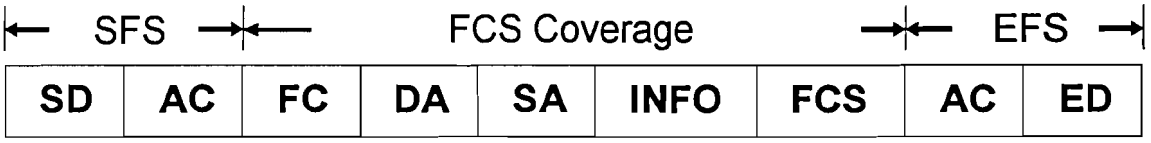
The figures in this chapter depict the formats of the fields in the sequence they are transmitted on the medium, with the left-most bit or symbol transmitted first. For the purpose of comparison in the processes the left-most bit or symbol is considered first, hence the left-most bit or symbol is most significant.

2.1.1.1 Token Format

SD = Starting Delimiter (1 octet)
AC = Access Control (1 octet)
ED = Ending Delimiter (1 octet)

The token is the means by which the right to transmit (as opposed to the normal process of repeating) is passed from one station to another.
The description of the fields will be given in the next paragraph.

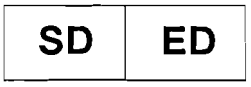
2.1.1.2 Frame Format



SFS = Start-of-Frame Sequence
SD = Starting Delimiter (1 octet)
AC = Access Control (1 octet)
FC = Frame Control (1 octet)
DA = Destination Address (2 or 6 octets)
SA = Source Address (2 or 6 octets)
INFO = Information (0 or more octets)
FCS = Frame-Check Sequence (4 octets)
EFS = End-of-Frame Sequence
ED = Ending Delimiter (1 octet)
FS = Frame Status (1 octet)

The frame format is used for transmitting both MAC and LLC messages to the destination(s). It may or may not have an information field. The MAC and LLC messages are called Protocol Data Units (PDU).

2.1.1.3 Abort Sequence



The Abort Sequence is used for the purpose of terminating the transmission of a frame prematurely. This sequence may occur anywhere in the frame, even on a non-octet boundary.

2.1.1.4 Fill

When a station is transmitting (as opposed to repeating), it transmits fill preceding or following frames, tokens, or abort sequences. This is to avoid an inactive or indeterminate transmitter state. Fill can be either 0 or 1 bits or any combination thereof, and can be any number of bits in length, within the constraints of Token Hold Time (THT).

2.1.1.5 Field Descriptions

STARTING DELIMITER (SD)

J = non-data-J 0 = binary zero
K = non-data-K

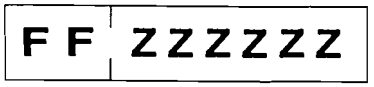
A frame or token starts with this octet. If otherwise, it will not be considered valid.
J and K are the non-data-symbols, which are a violation of the Differential Manchester Code (See Chap. 2.2).

ACCESS CONTROL(AC)

PPP = priority bits T = token bit
RRR = reservation bits M = monitor bit

The priority bits indicate the priority of the token (or frame). These bits indicate which stations are allowed to use the token.
The token bit is a 0 in a token and a 1 in a frame. When a station with PDU's to transmit detects a token that has a priority equal to or less than the PDU, it will change the token to a SFS and transmit the data.
The monitor bit is used to prevent a token with a priority greater than 0 or any frame from continuously circulating on the ring. The active monitor removes a frame or high priority token, with the monitor bit set to 1 from the ring. The active monitor inspects and modifies this bit. All the other stations repeat this bit as received.
The reservation bits are used by the stations, which have high priority PDU's queued for transmission, to request the next token to be issued at the requested priority.

FRAME CONTROL (FC)



FF = frame-type bits
ZZZZZZ = control bits

In the FC field the type of the frame and certain MAC and information frame functions are defined. The FF bits define the type of the frame as follows :

00 = MAC frame (contains a MAC PDU)
01 = LLC frame (contains a LLC PDU)
1X = undefined format, reserved for future use

If the frame-type bits indicate a MAC frame, all stations on the ring will interpret and act on the control bits. If the frame-type bits indicate a LLC frame, the control bits are designated as rrrYYY. The rrr bits are reserved and transmitted as 0's, and are ignored. The YYY bits can be used to carry the priority (Pm) of the PDU. Note : The priority of the PDU is always equal to or lower than the priority in the AC field.

DESTINATION AND SOURCE ADDRESS (DA AND SA) FIELDS

All frames contain two address fields: the destination (station) address and the source (station) address. The addresses can be either 2 or 6 octets in length. However, all stations of a specific LAN have addresses of equal length. The DA field identifies the station(s) for which the frame is intended. There are different types of destination addresses. A destination address can be :

Individual address
Group address
Broadcast address
Null address
Functional address

The source address identifies the station originating the frame and will have the same format and length as the DA in the given frame. The SA though can not be a Group or Broadcast address. For the complete specification of the address fields is referred to [IEEE802.5].

INFORMATION FIELD

The information field contains zero, one or more octets which are intended for MAC, SMT, or LLC. There is no maximum length specified for the information field. The transmission of a frame though may not exceed the token holding period the station has left. The frame type bits in the FC field indicate the format of the information field.

MAC frame format. The MAC frame format contains information for the MAC or SMT. For a complete description of the format is referred to [IEEE802.5].

LLC frame format. The format of the information field of the LLC frames are not specified in the IEEE standard 802.5. A ring station should however be capable of receiving LLC frames whose information field is up to and including 133 octets in length.

FRAME CHECK SEQUENCE (FCS)

The FCS is a 32-bit sequence based on the following standard generator polynomial of degree 32.

$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

The FCS is generated with the FC, DA, SA and INFO fields. The generated FCS is transmitted most-significant bit first.
The receiver generates a FCS is over the FC, DA, SA, INFO and FCS field of the frame. This leaves a unique remainder value.
For the exact specification of the generation of the FCS is referred to [IEEE802.5].

ENDING DELIMITER (ED)

J = non-data-J I = intermediate frame bit
K = non-data-K E = error-detected bit
1 = binary one

The delimiter shown above is transmitted by the station to indicate the end of a token or near end of a frame. The ED is considered valid if the first six symbols (JK1JK1) are received correctly.

The I flag indicates the end of a stations transmission. The I flag is transmitted as 1 in intermediate, or first, frames in a multiple frame transmission. The I bit is transmitted as 0 in the last or only transmitted frame.
The E bit is transmitted as 0 by the originating station of the token, frame or abort sequence. The E bit of tokens and frames is set to 1 by the station that detects a frame with error; otherwise it is repeated as received. All stations check the tokens and frames on errors like FCS error, non-data-symbols between start- and end-delimiter etc.

FRAME STATUS (FS)

A = address recognized bits
C = frame copied bits
r = reserved bits

The FS field is the last field of a frame. The r-bits are for future use, and are transmitted as 0's, and ignored by the receivers.
The A and C bits are transmitted as 0's by the station originating the frame. A station sets the A bits to 1 if it recognizes the DA as its own address or relevant group address. If this station is also able to copy the frame, it also sets the C bits to 1. The A and C bit are set without regard to the value of the E bit and only if the frame is good (as defined later).

2.1.2 Facilities

The facilities as defined by [IEEE802.5] will be described very brief. A complete description of these facilities would otherwise have led to a complete copy of [IEEE-802.5].

2.1.2.1 MAC Frames

In the Token Ring standard 23 MAC frames are specified, each containing one or more sub-frames. All these MAC frames will not be discussed in this paper. For those readers interested [IEEE802.5] is referred. At this point only the way frames should be handled will be mentioned. The way frames are handled depends on the value of the FF bits in the FC field in the following way:

(1) If the value of FF equals X'00' and it is addressed to the station, it will be copied only if there is sufficient free buffer available for copying.
(2) If the value of FF equals X'01' and it is addressed to the station, every effort will be made to copy the frame including overwriting previously received information.
(3) If the value of FF is greater than X'01' it will be addressed to all stations on the ring. It will be copied only if there is sufficient free buffer available. If the frame is not copied, action will be based on the value of the FC field.

The digits enclosed by X' and ' (like X'01') are the hexadecimal value of the assigned code point.

If a MAC frame is received which does not follow the description of the MAC frames as given in [IEEE802.5] it will be ignored by the receiving station(s).

2.1.2.2 Counters

Counters are defined to keep track of the number of times a particular error occurred. The Frame Count counter though is used to indicate the number of frames originated by the station which are still on the ring. Table I summarizes the defined counters.

2.1.2.3 Timers

For the functioning of the MAC protocols several timers are used. If a timer has been reset it resets to its initial value and is re-started. A time-out of a timer, signals an event upon which an action is started according to defined protocols. Table II contains the names of the timers, their default time-out and a short description of their function and the finite-state-machine (FSM) that uses it. (See chap 2.3.11).

Table I Counters

Line Error	Increments when token or frame with error and E=0 is received.
Internal Error	Increments when a recoverable error occurred in the station.
Burst Error	Increments on the absence of transitions for five half bit times.
AC Error	Increments on the incorrect sequence of AMP and SMP receiption.
Abort Delimiter Transmitted	Increments on the transmission of an abort sequence.
Lost Frame Error	Increments when the TRR expires
Receive Congestion Error	Increments when the station is not able to copy a frame addressed to the station.
Frame Copied Error	Increments on receipt of a frame addressed to its specific address with the A bits set.
Frequency Error	Increments when the frequency of the incomming bit stream differs more than specified.
Token Error	Increments when a station as active monitor has to issue a new token.
Frame Count	Counts the number of frames originated by the station that are still on the ring.

Table II Timers

Timer	Default Time Out (ms)	Description
Return to Repeat (TRR)	2.5	Each station has a timer TRR, which is used by the Operational FSM. The timer is used to ensure that the station shall return to the Repeat State. TRR shall have a value greater then the maximum ring latency.
Holding Token (THT)	10	The timer THT controls the maximum period of time the station may transmit frames after capturing the token. A station may transmit a frame if such transmission can be completed before timer THT expires. (Operational FSM).
Queue PDU (TQP)	10	TQP is used for the enqueuing of a SMP PDU after the receiption of a AMP PDU. Chap 2.3.6 explains the SMP and AMP PDU. (Standby Monitor FSM).
Valid Transmission (TVX)	THT + TRR	Each station shall have a timer TVX that is used to detect the absence of valid transmissions. (Standby and Active Monitor FSM).
No Token (TNT)	100	The timer TNT is used to recover from various token-related error situations. (Standby and Active Monitor FSM).
Active Monitor (TAM)	300	The TAM stimulates the enqueuing of a AMP PDU for transmission (See chap. 2.3.6). (Active Monitor FSM).
Standby Monitor (TSM)	700	The TSM is used by the standby monitors to assure that there is an active monitor on the ring, and to detect a continuous stream of tokens.
Error Report (TER)	200	When the TER timer expires, the station queues a Report Error PDU and resets this timer. (Standby and Active Monitor FSM).
BCN Transmit (TBT)	2600	The TBT timer specifies the length of time a station shall remain in BCN transmit before entering Bypass. (Standby Monitor FSM).
BCN Receive (TBR)	160	The TBR timer controls the time a station receives BCN frames from its downstream neighbour before entering Bypass state.

2.1.2.4 Flags

Flags are used to remember the occurrence of a particular event. They will be set when the event occurs. In Table III the names of the flags, and the events on which they are set are given.

Table III Flags

I	set upon receiving an ED with the I bit equal to 0.
SFS	set upon receiving an SFS sequence.
MA	set upon receiving an SA that is equal to the station's address.
SMP	set upon receiving an SMP or AMP with the A and C bits set to 0. only used by the standby monitors.
NN	set upon receiving an AMP or SMP with the A and C bits set to 0. only used by the standby monitor.
BR	set upon receiving a BCN frame othet than type 1, from its active downstream neighbour. reset upon receiving any other frame.
ETR	indication of wether the ETR oprion is selected or not.
NOT_MA	indication if the SA of the last transmitted frame is not equal to MA.

2.2 Physical Layer

The function of the PHY layer include data symbol encoding and decoding, symbol timing and reliability. The Token Ring system demands very strict requirements for symbol timing, data signalling rates, jitter and phase lock. These requirements on the PHY layer will be discussed in the next chapter.

Symbol Encoding

In the MAC four different symbols are used. The PHY encodes and transmits these symbols as they are presented at its MAC interface by the MAC.
The four symbols are :

0 = binary zero
1 = binary one
J = non-data-J
K = non-data-K

The symbols are transmitted to the medium by the PHY in the form of Differential Manchester Encoding that is characterized by the transmission of two signal elements per symbol.
In the case of the binary zero and one, the polarity of the signal in the second half of the symbol time is the opposite of the polarity in the first half. The polarities in both halfs are equal for the transmission of the non-data symbols J and K.

The polarity of the first signal element is completely dependent on the polarity of the trailing signal element of the previous transmitted data or non-data symbol. If a binary zero or a non-data-K is to be transmitted, the polarity of the leading signal element is opposite to that of the trailing signal element of the previous symbol. If a binary one or a non-data-J is to be transmitted the polarity of the leading signal element is equal to the polarity of the trailing element of the previous data or non-data symbol.
The non-data symbols are in fact code violations of the differential Manchester code.

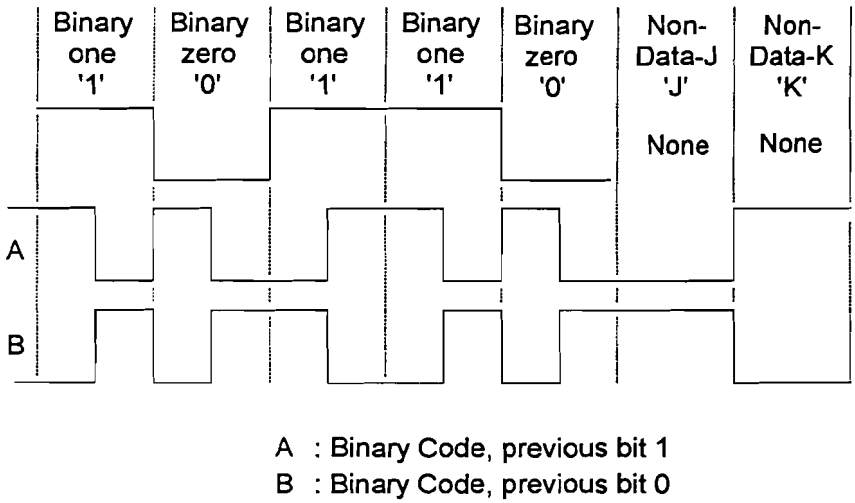


Figure 2.1 Example of Symbol Encoding

Symbol Decoding

The algorithm to decode the received symbols is the inverse of the one described for symbol encoding. The decoded symbols shall be presented to the MAC at the MAC interface.
If the PHY receives more than four signal elements of the same polarity in succession, it shall introduce a change of polarity at the end of the fourth signal element. The PHY shall continue to introduce a transition each signal element time until a transition is received from the ring. The resulting bit stream is then decoded and the symbols presented to the MAC interface. The same shall be done during periods of loss-of-clock synchronisation or under/overrun of the latency buffer.

Latency Buffer

A latency buffer shall be provided by the active monitor for two distinct functions.

Assured Minimum Latency. If all stations are in repeat mode a token must be able to circulate on the ring. Therefore the active monitor must introduce a delay of 24 symbols (48 signal elements).

Phase Jitter Compensation. Although the mean data signalling rate around the ring is controlled by the active monitor station, segments of the ring can operate instantaneously

at speeds slightly higher or lower than the frequency of the master oscillator. The cumulative effect of these variations in speed are sufficient to cause variations in the latency of the ring. Based on the jitter specifications as defined in [IEEE802.5], the total latency variations shall not exceed B bits. The parameter values are :

Data rate	= 4	16	Mbits/s
B	= 3	15	bits

An elastic buffer with the length of n bits is added to the fixed 24 bit buffer. This maintains constant ring latency. The parameter values are :

Data rate	= 4	16	Mbit/s
n	≥ 6	32	bits

The resulting latency buffer is initialized to 24 + n/2 bits. Constant total latency is a requirement to avoid adding or dropping bits from the data stream.

2.3 Medium Access Control Layer

In the Medium Access Control (MAC) layer the protocols for the reception and transmission of the different types of frames is defined. In this chapter an overview of the protocols is given.

2.3.1 Frame Transmission

Access to the physical medium (the ring) is gained by capturing a token. Upon request for transmission of a LLC PDU, MAC PDU or SMT PDU, the MAC enqueues the PDU with the appropriate FC, DA, and SA fields and waits for the reception of a token that may be used for transmission.
Such a token has a priority equal to or less than the priority of the enqueued PDU. When such a token is received, it is changed to a SFS by setting the token bit, repeating is stopped and the PDU is transmitted. During transmission of the PDU the FCS for the frame is accumulated and appended to the end of the information field.
If a non-usable token is received, a usable token is requested by setting the RRR bits in the token.

2.3.2 Token Transmission

After completion of the transmission of frames, the MAC checks to see if the station's address has returned in the SA field. This is indicated by the MA flag. If it has not been seen, the station transmits fill until the MA flag is set, at which time the token will be transmitted.

2.3.3 Stripping

The station remains in the transmit state after the transmission of the frames until all of the frames originated by the station are removed from the ring.

2.3.4 Frame Reception

While repeating the incoming bit stream the station checks it for frames it should copy or act upon. If a MAC frame is received the control bits are interpreted by all stations on the ring. In addition, if the frames DA field matches the stations's address the FC, DA, SA, INFO and FS fields are copied into the receive buffer and forwarded to the appropriate sublayer.
A received frame can be indentified as one of three varieties: *good frame*, *validly formed frame*, and *frame with error*. These frame varieties are indicated by combinations of the following properties:

Properties of a frame

(A) Is bounded by a valid SD and ED
(B) Has the E (error) bit equal to 0
(C) Is an integral number of octets in length
(D) Is composed of only 0 and 1 bits between the SD and ED
(E) Has the FF bits of the FC field equal to 00 or 01
(F) Has a valid FCS
(G) Has a minimum of 10 octets for 2-octet addressing or 18 octets for 6-octet addressing between SD and ED
(H) Does not contain a valid SD or ED between the bounding SD or ED

The three varieties are :

Good Frame (FR_GOOD). A bit sequence that satisfies the following conditions :

A & C & D & E & F & G

Validly Formed Frame. A bit sequence that satisfies the following conditions :

A & C & E & G & H

Frame with Error (FR_WITH_ERROR). A bit sequence that satisfies the following conditions :

A & (not C | not D | (E & not F) | (E & not G))

2.3.5 Priority Operation

An attempt is made to match the service priority of the ring system to the highest priority PDU that is enqueued for transmission with the help of the priority (PPP) and reservation (RRR) bits in the AC field. The current ring service priority is indicated by the priority bits in the circulating frame or token.
As previously noted, the priority- and reservation bits of the last received frame or token are stored in the Pr and Rr register. Also used for the priority operation are the stacks Sx and Sr. These stacks are only used if the station has become a stacking station, after the station has raised the service priority of the ring. The station who raised the service priority, is also responsible for degrading it to the original service priority level.

The top of the stack Sx contains the service priority to which the station has raised the ring system the last time.
The top of the stack Sr contains the service priority the last time before the station raised the service priority.
Pm is the priority of the enqueued PDU. P is the priority of the token.

With the above information the priority operation is explained as follows.
There are three cases in which a station originates a new token. These cases are :

1. After the station has captured a token with $P \leq Pm$, and has transmitted frames. The station stops transmitting frames if the THT expires or it has no more frames queued with $Pm \geq P$.
2. If a token is received with $P = Sx$ and $Pm \leq Sx$.
3. If a token is received with $P = Sx$ and no PDU's queued.

If one of these three cases occur, the following stacking actions are performed, and token is released:

If Pr > Sx	and Pr ≥ Px	then TK(P = Pr and R = Px)	
If Pr > Sx	and Pr < Px	then TK(P = Px and R = 0)	and STACK(Sx = P and Sr = Pr)
If Pr = Sx	and Sr ≥ Px	then TK(P = Sr and R = Px)	and POP(Sx and Sr)
If Pr = Sx	and Rs < Px	then TK(P = Px and R = 0)	and RESTACK(Sx = P)

(Px = MAX(Pm,Rr), Sx(Sr) is top of stack Sx(Sr))

If the station has a PDU queued for transmission and non of the above cases are valid, the station will request an appropriate token by setting the reservation bits in the passing frames or tokens. The station will only set the reservations bits if $Rr < Pm$.
The stacks Sx and Sr will be cleared if a token or frame is received with $P < Sx$.

2.3.6 Beaconing and Neighbour Notification

When a hard failure is detected in the ring system, its cause must be isolated to the proper failure domain so that recovery actions can take place. A failure domain consists of the following properties:

- the station reporting the failure (the beaconing station)
- the station upstream of the beaconing station
- the ring medium between them

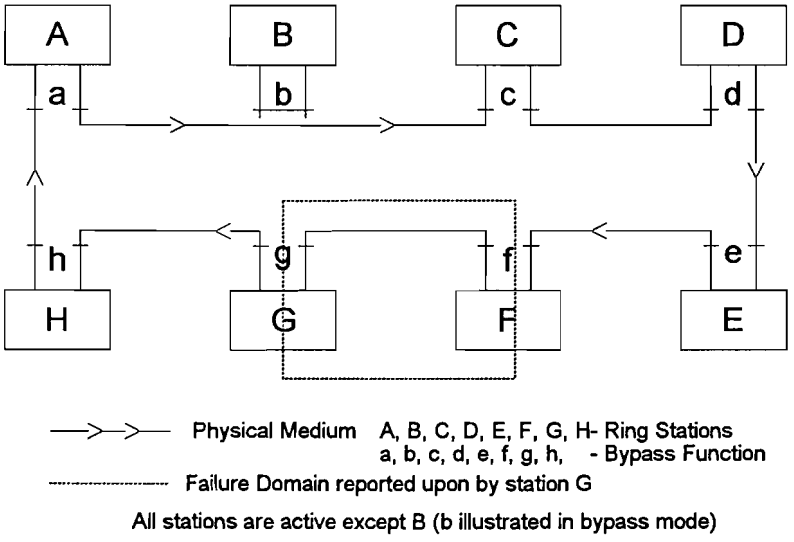


Figure 2.2 An Example of a Failure Domain

Figure 2.2 supports the next example.
If a failure occurs within the shown "failure domain", station G reports this failure by transmitting Beacon (BCN) MAC frames. A failure that causes bit disruption within the transmitter side of station F, or within the receiver side of station G, hence in the medium between stations F and G, is detected and reported upon by station G using a BCN MAC frame. This alerts all other stations that the normal (token) protocol is suspended until such at the time that the disruption terminates or is removed.

For accurate problem termination, all stations should know the identity of its upstream neighbour station. The process for obtaining this identity, known as neighbour notification is as follows:
The neighbour notification process is based on the A and C bits in the FS field, which are transmitted as zero's.
When a frame is broadcast to all stations, they recognize the DA field, and will try to set the A bits, and C bits if the frame is also copied. But it's the first downstream station that sees the A and C bits as zero, and is able to set the A (and C) bits. Hence, if a station receives a broadcast frame with the A bits zero, the SA field identifies its upstream neighbour.

The monitor starts the neighbour notification by broadcasting the Active Monitor Present (AMP) MAC frame. The first downstream neighbour takes the following actions:

- Resets the Timer Standby Monitor (TSM), based on seeing the AMP value in the FC field,
- If possible, copies the broadcast AMP MAC frame and stores the upstream station identity in the UNA memory location,
- Sets the A bits (and C bits if the frame is copied) of the repeated frame to one's.
- At a suitable transmit opportunity, broadcasts a similar Standby Monitor Present (SMP) MAC frame.

One by one, each station receives a SMP frame with the A and C bits set to zero's, stores its SA in the UNA memory, and continues the process by broadcasting a SMP frame itself.

The AMP frame is also used for detecting the continuous transmission of tokens, since the AMP frame must pass each station on a regular basis. If the timer TSM in an standby monitor expires, that station begins transmitting Claim Tokens frames.

2.3.7 Error Reporting

When a station detects an error condition on the ring it increments the appropriate error counter. These errors are reported on a periodic basis to the REM. Persistent errors can be detected, isolated, and necessary action taken.

2.3.8 Administration

To assure compatible operation among stations in the ring the RPS keeps track of the various settable ring operation parameters. A station requests the values of these parameters upon insertion in the ring.

2.3.9 Configuration Control

The CRS is notified by a station when they detect a change in the address of their upstream neighbour. Stations can also get requests from the CRS for reporting status information, or remove themselves from the ring. In this way the CRS maintains the configuration of the ring system.

2.3.10 Early Token Release

With the Early Token Release (ETR) option set, a station is allowed to release a token as soon as it completes frame transmission, before the header of the last frame has returned to the station. The ETR option increases available bandwidth and improves the data transmission efficiency of the token ring protocols.
Stations implementing the ETR option are compatible and interoperable with stations that do not.

2.3.11 Finite State Machines

The Token Ring standard defines the protocols in three finite-state machines.

The Operational Finite-State Machine defines the actions for repeating, copying, and transmitting any type of frame or token. The Operational Finite-State machine does not consider the type of frame to handle. The priority operation is also captured in this finite-state machine. Controlling the finite-state machine is done by the Standby or Active Monitor Finite-State Machine.

Upon coming on-line or after a reset, the Standby Monitor Finite-State Machine is active. First (re)initialization is performed to assure that no other station on the ring has the same address as this station and that its (re)entry into the ring is notified to its downstream neighbour.
After initialization the station as standby monitor, monitors the ring to assure that there is a properly functioning active monitor in the ring system. It does so by observing the tokens and AMP frames as the are repeated on the ring. If these tokens or AMP frames are not detected periodically, the standby monitor will time-out and initiate claiming token.
In the Claim Token state the standby monitor tries to become the active monitor. After becoming the active monitor the Standby Monitor Finite-State Machine is abandoned.
While being in the Claim Token or BCN states the station uses its own (master) oscillator for transmission and timing.

In the Active Monitor Finite-State Machine the functioning of the active monitor is defined. The function of the active monitor is to recover the ring-system from various error situations such as absence of validly formed frames or tokens on the ring, a persistently circulating priority token, or a persistently circulating frame. In normal operation there is only one active monitor in the ring-system. The active monitor uses its own oscillator to provide timing for all symbols repeated or transmitted on the ring. The active monitor also supplies the latency buffer for the ring-system.

Chapter 3
Token Ring Systems

Since the release of the Token Ring standard 802.5 several implementations where introduced on the market. This chapter gives an overview of three of these implementations. This overview is included to give the reader a possibility to compare the design, presented further, on with these existing implementations.

3.1 Toshiba

The Token Ring LAN Controller developed by Toshiba was first announced at the IEEE Custom Integrated Circuits Conference in may 1989 in San Diego. (See [TANA89])
The TRC is a CMOS 510K-Transistor single-chip compatible with the IEEE 802.5 MAC protocol and is fabricated with 1.2-μm p-well CMOS and double-level AL technology. The 510K mosfets are integrated in a 14.49 X 14.62-mm chip area.

The TRC consists of five functional blocks that form a dedicated 16-bit microprocessor. These blocks are:

- 11K-word X 20-bit protocol firmware ROM (COMFRM)
- finite state machines for real-time handling of frames (RING I/F)
- 896-word X 16-bit dual-port RAM for frame buffer FIFO's and working memory (FIFO/RAM)
- host processor bus interface (BUS I/F)
- three-channel DMA controller (DMAC)

It is possible to construct a compact adapter card with the TRC and a MDR, which is a medium driver and receiver LSI for the IEEE 802.5 physical layer.

COMFRM
COMFRM is a dedicated 16-bit microprocessor. It is equipped with an 11K-word X 20-bit firmware ROM, a 452-word X 16-bit MAC frame subvector ROM, and a 7-word X 16-bit RAM. It interprets and executes the TRC commands and processes the receive- and transmit frames according to the protocol.

FIFO/RAM
This includes a 896-word X 16-bit dual-port RAM, which is composed of the FIFO area (128-byte X 7, 40-byte X 1) and the 856-byte RAM area used by the COMFRM for holding various parameters.

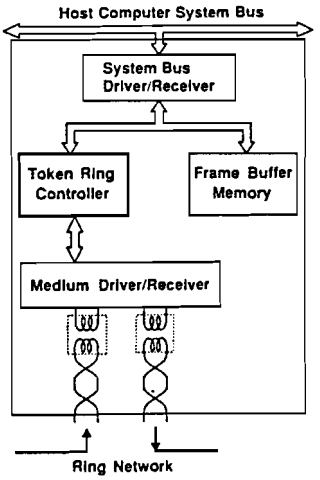


Figure 3.1 Adapter Card

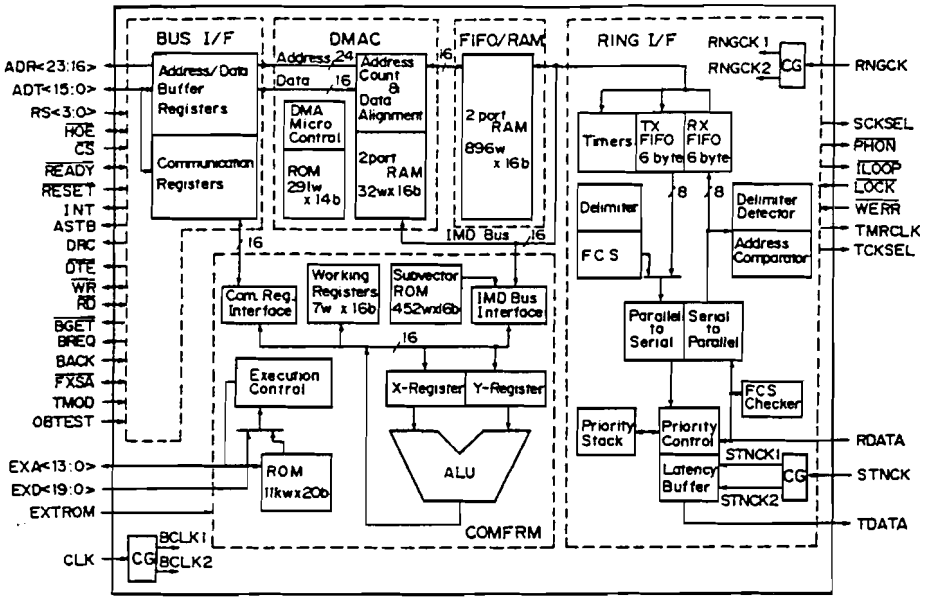


Figure 3.2 Functional Block Diagram of the Toshiba TRC

RING I/F

The ring interface processes the incoming and outgoing frames in real-time. It consists of the priority control circuits, latency buffer, serial/parallel conversion, the 32-bit frame check sequence (FCS) generator and checker, address comparators and timers.

The TRC uses two externally generated clock signals for transmitting bits to, and receiving bits from the ring. One clock is the master clock which is used when the station is the active monitor. The other clock is the clock extracted from the received frames.

BUS I/F

Communication between the TRC and a host CPU is established through seven 16-bit registers, and an interrupt unit. One of those registers is the COMMAND register in which the CPU can write one of the 16 commands recognized by the TRC. These commands are used for starting, stopping, resetting and setting the TRC.

DMAC

The DMAC is a three-channel DMA controller, which controls the bidirectional data transfer between the BUSI/F and the FIFO/RAM. The DMAC is controlled by 291-word X 14-bit firmware on ROM, and has a 32-word X 16-bit dual-port RAM. Two out of three channels can send or receive frame data with a specific list structure.

Conclusions

Toshiba has been able to integrate a Token Ring Controller on a CMOS 510K Transistor Single-Chip compatible with the IEEE 802.5 MAC protocol. The TRC does not contain the functions for the physical layer of the standard. Therefore a Medium Driver Receiver is necessary. Inside the TRC clock buffers are used to eliminate the clock-skew problem.
One disadvantage is the fact that the TRC has a multiplexed data/address bus. This degrades the data transfer rate between the TRC and an external memory.
Controlling the TRC by an attached system is very straight forward with the use of 16 commands of 6-bits.

3.2 IBM

The TRC of IBM is one of the commercially available implementations of the IEEE 802.5 Token Ring standard. IBM has been able to develop a 9.02 X 9.02 mm chip built in 1μm CMOS. The chip contains both analog and digital circuits and has provisions for self-test. The function of the TRC includes the transmitter, receiver, protocol handler, microprocessor as well as several interfaces for attachment to different bus systems.

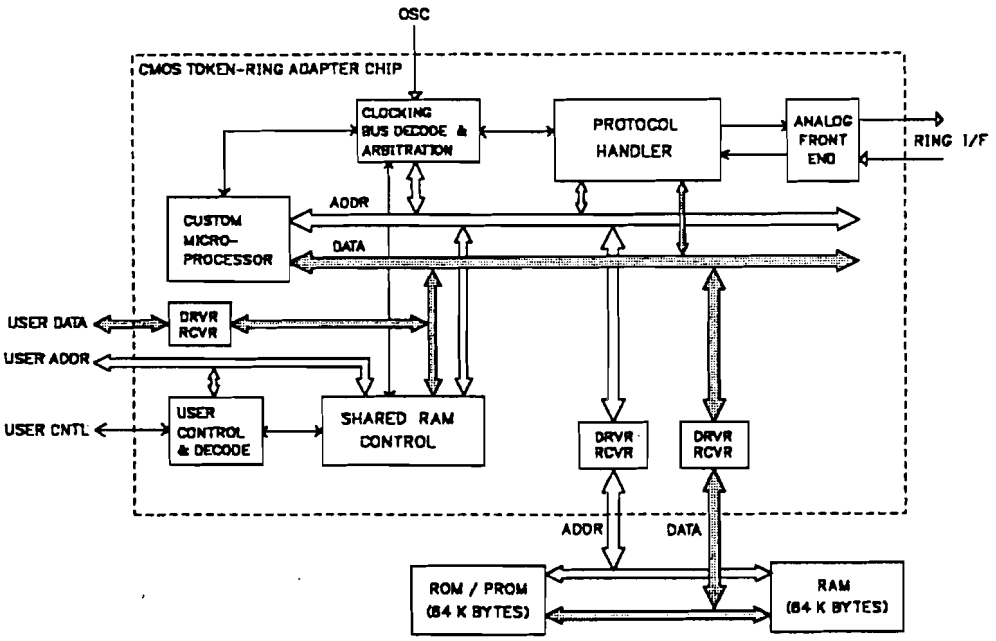


Figure 3.3 Functional block diagram of the IBM TRC

A complete token ring adapter can be formed with this TRC when combined with external PROM and RAM modules, bus drivers and discrete line interface components.

Analog Front End
This on-chip analog macro performs signal conversion and clock recovery functions as well as detecting and compensating for line impairments.

Protocol Handler
Most bit- and byte level functions required to implement the IEEE 802.5 standard are performed by the protocol handler. Receive functions include decoding, deserialization, address recognition, frame disassembling, linked buffer list processing, and interrupt control. Transmit functions include linked buffer list processing, frame assembly, token capturing, serialization, and encoding.
About 50% of the protocol handler logic is operated at a 32 Mhz clock rate, when running at the 16 Mbit/s data rate, while the remaining 50% operating at 5.34 Mhz.

Microprocessor
The integrated 16-bit microprocessor controls all receive- and transmit functions. Off-chip PROM and RAM modules are necessary to provide program storage, workspace, and data buffers. The processor yields a performance of about 3 MIPS.

User control & decode
This block provides a multiple-mode bus interface. There are interfaces for the IBM PC™ bus, IBM PS/2™ bus, IBM 3174™ bus, and Motorola 68000™ bus.
The bus interface function contains nine 16-bit registers that provide bidirectional interrupt capability and the ability to create two read-only regions in the shared RAM window.

Shared RAM
The adapter RAM appears in the memory map of the attached system, while remaining accessible to the integrated microprocessor and protocol handler. Through this shared RAM and memory mapped registers implemented on the chip, the transmit, receive and status commands between the chip and user are executed.

Conclusions

On the embedded microprocessor of the TRC up to 64K bytes of microcode can be executed, providing a logical link control (LLC) interface that supports up to 255 links.
The IBM TRC therefore not only provides the functions for the implementation of the IEEE 802.5 standard, but also provides the functions for the IEEE 802.2 standard. There is though external memory necessary for the functioning of the TRC.
On the other hand the TRC is provided with analog circuits performing the physical layer of the IEEE 802.5 standard.

3.3 Texas Instruments

The TMS380 LAN Adapter Chipset of Texas Instruments consists out of five IC's. With these five IC's and a memory expansion unit a Token Ring Controller compatible with the IEEE 802.5 standard can be built. The adapter chipset was introduced in the mid eighties. As technology was not as integrated as today's technology, Texas Instruments was forced to divide the standard in 5 IC's.
Figure 3.4 shows a functional block diagram of the adapter chipset.

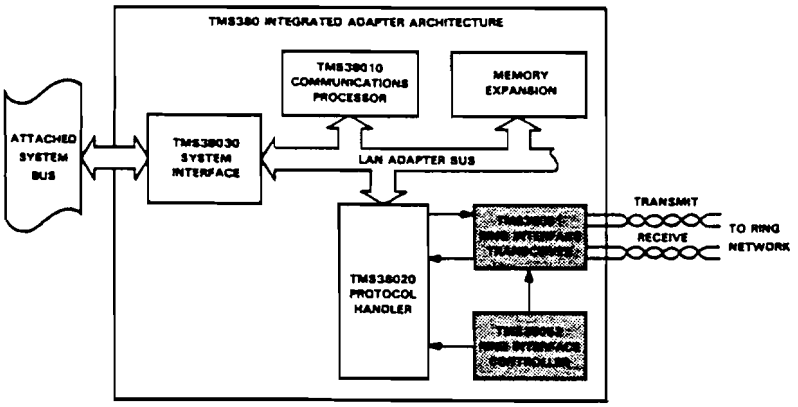


Figure 3.4 Functional block diagram of the TMS380

As the diagram shows the TMS38030 System Interface provides the communication between the LAN Adapter and the attached system. The System Interfaces can be configured for an Intel- or Motorola-like system bus.

The LAN Adapter chipset is not up to date any more. Not only because of the fact that 5 IC's are needed, but also because the system interface uses a multiplexed address/bus and has only a 10 Mhz maximum clock rate.
One of the other minors is the absence of the wanted real-time characteristics based on the priorities. (See [MARC89]).

Chapter 4
Design Definitions

The design of the Token Ring Controller (TRC) is bounded by several definitions and restrictions. These include the definition of the system- and ring interface, restrictions on the composition of the architecture of the TRC, and restrictions due to the use of IDaSS as the design tool.

4.1 IDaSS

As pointed out in the introduction the TRC must be designed for the Object-Oriented (Hardware) System Design project library. Therefore IDaSS will be used as the design system. The use of IDaSS brings a few restrictions along.

First, it is possible to design synchronous systems only. In this case that can hardly be seen as a real restriction. It although gives some problems to check the synchronization of the incoming bit stream (See chap. 7 Testing).

Secondly, all memory elements like registers and queues use the same system clock provided by IDaSS. Therefore it is not possible to control memory elements directly by for instance the ring clock.

Finally only digital systems can be designed in IDaSS. This brings a problem with respect to the very high demands the token ring standard specifies for the timing of repeating and transmitting data. Building a Phase Lock Loop (PLL) according to the token ring specifications is not possible with IDaSS. Therefore the TRC will not contain a PLL, but will instead have an interface that is compatible with existing integrated circuits containing a PLL.

4.2 System Interface

All elements of this library must follow the strict rules for the interfaces. A complete specification of system interface can found in [VERS93].
Here follows an abstract of the interfaces.

4.2.1 Input/Output Interface

An input/output device, like the TRC, is connected to a processor core using an interface based upon the 'external register space' model found in the 8051 processor. The input/output interface uses the following three buses:

1 Data output bus
This is an eight bits wide bus which is always driven by the processor core during a write or modify action. During other clock cycles, the value may be UNKnown or three-state.

2 Data input bus
This is an eight bits wide bus which is driven by the addressed input/output device during a read or modify action. During other cycles, this bus must be three-stated by all connected devices.

3 Control/address bus
This is a ten (10) bits wide bus which is always driven by the processor core. Its value may never be UNKnown or three-state.

Bits 0..7 constitute the address of the register which must be read and/or written.
Bit 8 is the 'write' bit, which is set to %1 for a write action.
Bit 9 is the 'read' bit which is set to %1 for a read action.

Bits 8 and 9 may be %1 at the same time indicating a modify action (the processor reads the addressed register and immediately writes new data into it, all within a single clock cycle). The address bits will be ignored by all devices when bits 8 and 9 are both %0.

The timing of the input/output interface is very simple, as each access only lasts as single clock cycle, and no handshaking is involved. The input/output interface is always byte oriented.

4.2.2 Direct Memory Access

The DMA controller of the TRC should be fitted with the standard memory interface for processor core. Connecting the DMA controller of the TRC and a processor with the memory can be done through the standard memory interface multiplexer/arbiter already in the project library. The memory interface consists out of the following buses:

1 Data bus

The number of bits of this bidirectional bus is not fixed, but will generally be 8, 16 or 32 bits. Subdivision of this bus is possible by using separate write strobes for each of these subvisions. Partial updating of words may be indicated. Reading is always done for whole words.

2 Address bus

This is an output bus by the device in the direction of the memory, and has no fixed number of bits. The address bus indicates which data word in memory must be read or written.

3 Control bus

The number of bits of this bus, driven by the device to the memory, equals the number of logical buses on the data bus plus 1.
The least significant bit (bit 0) of the control bus indicates a read action to be performed. The higher significant bits of the control bus indicate write actions to be performed on the separate logical buses. The order of the write bits follows the same order as the logical buses on the data bus; bit 1 on the control bus writes the least significant part of the bus.

4 Handshake bus

This is a bus driven by the memory in the direction of the device. The minimum width is one (1) bit, needed to indicate the 'ready' condition. A memory access ends when the memory places a non-zero value on the handshake bus during a clock cycle.
The following figure shows a timing table for memory access.

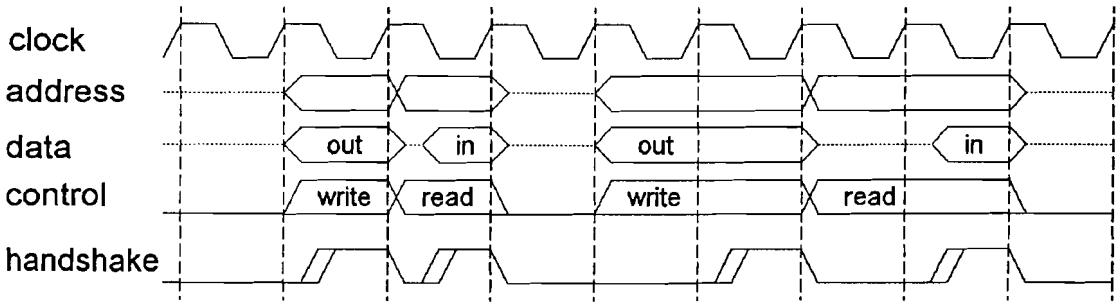


Figure 4.1 Fast and Slow Memory Access Cycles

4.2.3 Interrupt Interface

The input/output device must be capable of interrupting the processor core. The basic interrupt interface consists out of a single bit bus, driven by the device, carrying %1 to indicate the presence of an interrupt. This bit, set by the device when it sees fit, must be reset during the interrupt handling of the processor core.
An external interrupt controller may be used but will not be outlined here because its irrelevant to the TRC.

4.3 Ring Interface

Paragraph 4.1 mentioned that the TRC will not contain a PLL. At this time several integrated circuits are available, which are able to establish physical connection to the token ring transmission medium. These integrated circuits contain a PLL that can meet with the timing specifications of the token ring standards.
The TRC must be designed with a ring interface which is capable of interfacing with these integrated circuits.

4.4 Architectural Decomposition

The decomposition of the architecture of the TRC in a bit- and a byte-processor is the last definition restricting the design of the TRC. This decomposition will be made because a very basic processor core is available for byte-wise operations. This byte-processor was designed by dr. ir. Verschueren for a HDLC controller but can be used in the TRC without much change.
Both processors and their interface will be described in the next chapter.

Chapter 5
The Token Ring Controller

This chapter contains the specifications of the interfaces, timing and basic architecture of the TRC.
The TRC can be divided in two main parts; a bit- and a byte-processor. The bit-processor controls the receipt and transmission of frames and tokens on bit-level. The byte-processor consists out of a very trimmed-down 8051 like processor. The monitor program of the TRC that should be executed by this processor, and control the bit-processor, has not been developed yet. Therefore this chapter contains an extensive specification of the interface between bit- and byte-processor and the functions of the bit-processor.

5.1 The TRC

The TRC provides the necessary functionality for implementing the IEEE/ANSI 802.5 Medium Access Control standard. Not present in the TRC is a PLL for extracting timing information out of the incoming data. Instead the TRC delivers a set of ring interface signals that are customary for controlling several Token Ring Interface circuits.

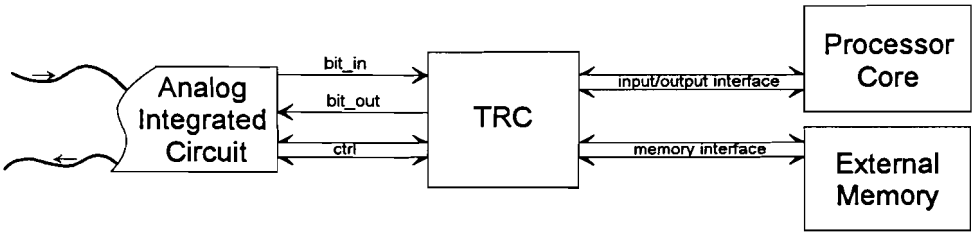


Figure 5.1 The TRC System

5.1.1 Connector Description

The connectors of the TRC can be divided in two groups; The ring interface and the system bus interface.
The ring interface provides the signals for the controlling of and interfacing with, an external ring interface circuit, with a PLL. The system interface consists of the buses and signals for interfacing with an attached processor core. This interface is according to the specifications of the standard system and I/O buses for the Object-Oriented (Hardware) System Design project library (See chap. 4.2). For a complete description of the specification is referred to [VERS93].

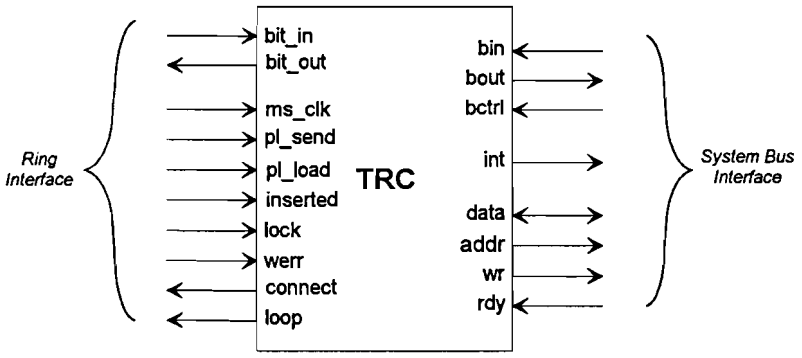


Figure 5.2 The TRC Connectors

Ring interface

The ring interface consists of 9 connectors of 1 bit wide each. These connectors are:

connect Output to ring interface circuit. *Connect* is asserted high by the TRC if it wants to be inserted in the ring.

loop Output to the ring interface circuit. *Loop* is asserted low by the TRC if it wants to perform an internal loop test.

bit_out The output bit stream to the ring interface circuit.

bit_in The input bit stream from the ring interface circuit.

inserted Input from the ring interface circuit. The TRC assumes it's inserted in the ring, or the loop circuit has been established (when *loop* is asserted active low), when *inserted* is asserted high.

lock Input from the ring interface circuit. The TRC assumes the PLL has locked on the incoming bit stream, within the allowed frequency range, when *lock* is asserted high.

werr Input from the ring interface circuit. The TRC assumes a wiring error in the interface cable to the ring, when *werr* is asserted low.

ms_clk The TRC uses this signal for transmission timing when it's the Active Monitor. Chap 5.1.4 give an extensive specification of the receive and transmit timing.

pl_send The TRC uses this signal for transmission timing when it's a Standby Monitor. See chap. 5.1.4.

pl_load The TRC fetches a new bit from *bit_in* when *pl_load* is asserted high. See chap. 5.1.4.

System bus interface

The following connectors for the communication between the TRC and the attached processor core are according to the Input/Output interface specifications in [VERS93] (See chap. 4.2).

bin Data output bus. The output bus from the processor core is the input bus for the TRC.

bout Data input bus. The input bus for the attached system is the output bus from the TRC.

bctrl Control/address bus driven by the attached system for addressing the TRC.

int The one bit interrupt signal from the TRC to the attached processor core.

The following connectors for the communication between the TRC and an external memory are according to the memory interface specification in [VERS93] (See chap. 4.2).

data Data bus. This bidirectional data bus is 8 bits wide. Through this bus the TRC reads/writes words from/to the external memory.

addr Address bus. This output bus driven by the TRC is 16 bits wide. The value on this bus indicates the location in the external memory the TRC wants to read/write.

wr Control bus. This 2 bits wide output bus indicates if the TRC wants to read or write the word at the location indicated by the addr bus.

wr = %00 no action
wr = %01 read
wr = %10 write
wr = %11 not allowed

rdy Handshake bus. This one bit input bus driven by the external memory indicates when it has completed the read/write cycle.

5.1.2 Architecture

The TRC consists out of the bit-processor and the byte-processor. The main parts of the bit-processor are the Ring Interface, a Differential Manchester Encoder and Decoder, a Receive Filter, a Priority Machine and Bit-to-Byte-Processor Interface. The byte-processor contains a DMA unit, an ALU, a Control Unit and a CPU Interface.
The byte-processor is able to access the bit-processor through its 32 X 8-bits register space. The bit-processor appears in the registers 0, 1, 2, 10, 11 and 12. By writing and reading from these registers, the byte-processor controls the bit-processor and receives/transmits frames. Chap 5.2 specifies the use of these six registers.
On the other side the bit-processor signals the byte-processor when a particular receive event has taken place through a 4 bits *event* bus.
The TRC is always running, and switching, between two processes; A Transmit and a Recieve process. Inside the bit-processor these two processes are running simultaneous and independent. The byte-processor though switches between the two processes. When a receive event occurs, as indicated on the *event* bus, the byte-processor switches immediately to the Receive process. The byte-processor remains in the Receive process until the received event has been handled, after which it switches back to the Transmit process. Several registers in the byte-processor are duplicated for the two processes. In this way switching between the two processes takes no time at all.

Bit-processor

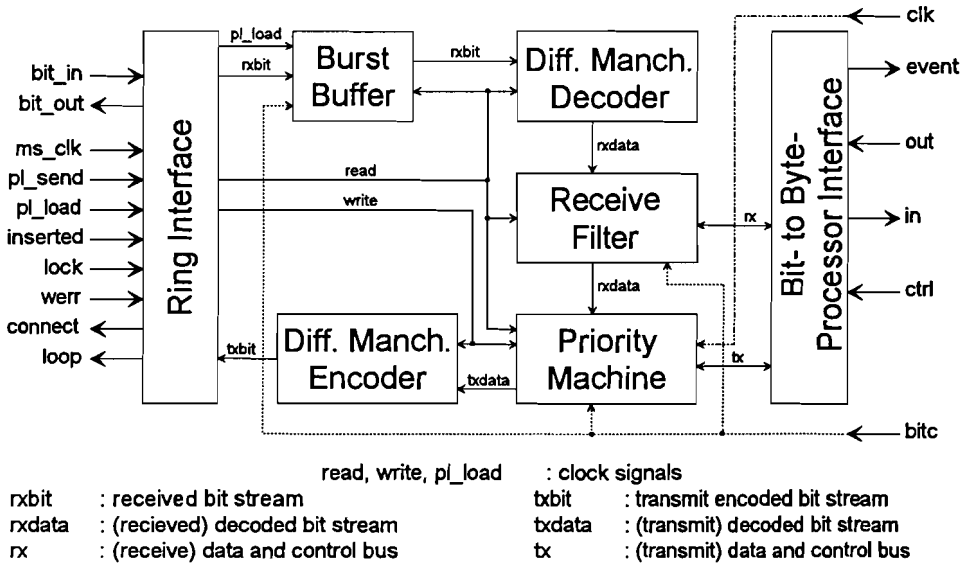


Figure 5.3 Functional Block Diagram of the Bit-processor

The Ring Interface of the bit-processor provides the control and signalling for an external ring interface circuit. Controlling includes inserting in the ring and performing an internal loop test. Possible signalling by the external circuit includes indication of being connected to the ring, the PLL being locked, presence of a wiring fault, the master clock and the PLL-clock extracted from the incoming bit-stream (See 5.1.1).
The Burst/Buffer contains a burst-five error detector, and the latency buffer. The burst detector not only detects a burst-five error, it also changes it into a burst-four. After the detector has detected the absence of a transition on the input bit stream (*rxbit*) during 5 bit times, it starts to introduce a transition each bit time until a transition from the input is detected.
Machester decoding and start/end delimiter detection is performed in Diff. Manch. Decoder. Diff Manch. Encoder encodes the data for transmission.
These three functional blocks are the part of the physical layer of the Token Ring, that could be implemented in the TRC.

While repeating the incoming data, the Receive Filter copies the passing bits in order to perform error checking, and transports the necessary fields of a frame byte-wise to the Bit-to-Byte-Processor Interface.
In the Priority Machine the complete priority mechanism of the Token Ring is implemented. The Priority Machine keeps track of the last received values of the priority and reservation bits, and performs stacking operation when it changes the priority of the ring while releasing a new token.

When an event occurs during the receipt of a frame or token the Bit-to-Byte-Processor Interface signals this event to the Byte-processor, through the *event* bus. The Byte-processor recognizes such an event and will perform the necessary actions. A receive

event can be the receipt of a token, the presence of a byte of a frame that has to be fetched by the Byte-processor, or the occurrence of an error in the received data or physical connection. Chap 5.2 specifies the functioning of the Bit- to Byte-Processor Interface and the used interface buses (*clk*, *out*, *in*, *ctrl*, and *bitc*).

Byte-processor

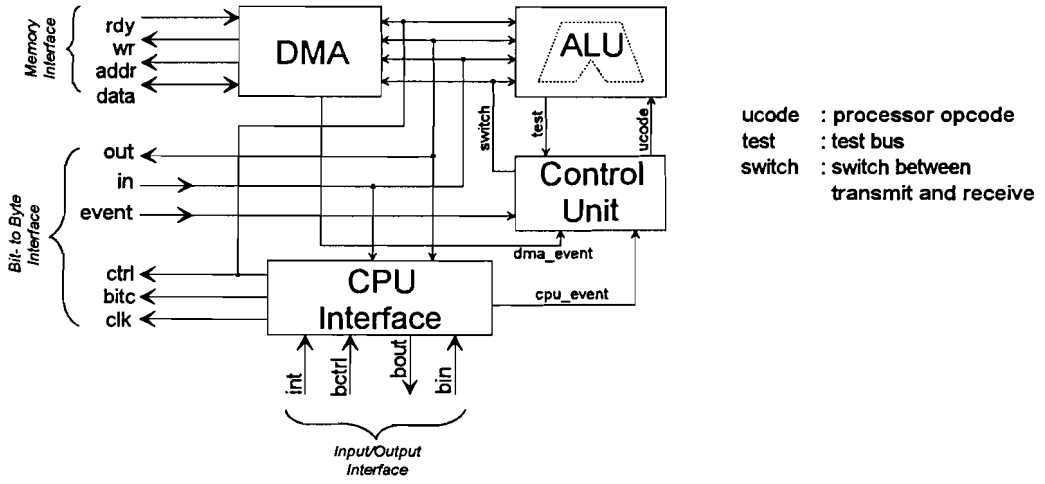


Figure 5.4 Functional Block Diagram of the Byte-processor

The ALU and Control unit together are the Processing Unit of the TRC. This processing unit is a very trimmed 8051 like processor using 32 bit addressable registers from which the first 16 are bit testable. The control vector size is reduced to 20 bits, while most key operations take only one control word and one clock cycle.
In the Control unit a 20 X 1K ROM is placed. The monitor program for controlling the TRC is placed is this ROM. The monitor program is responsible for controlling the DMA unit, communicating with the attached processor core, and controlling and communicating with the Bit-processor. This monitor program has not been developed yet.

The DMA unit is only equipped with the most basic functions for performing Direct Memory Access. Only two DMA channels are provided for sequential accessing an external memory. One channel for the Receive process, and one channel for the Transmit process. No features like paging or segmentation are provided.
The DMA unit is equipped with one address, data, counter and flag register, but all duplicated for the transmit and receive process.

The CPU Interface contains 11 registers which are accessible by the Processing Unit. These additional registers are located at the physical addresses 8 to 11 and 21 to 27. The registers located at the addresses 10 and 11 are directly tapped around the byte-processor to the bit-processor (*bitc* bus). This 16 bits bus is used for setting the bit-processor mode (5.2.4), and is thus a part of the Bit- to Byte-Processor Interface.

The bit-processor itself is also located in the register space, at the physical addresses 0, 1, 2 and 12. These three registers are used to control the bit-processor, and to transfer bytes. (See chap. 5.2.1 and 5.2.2)

Three buses are used for the operations on a register. The two 8-bits data buses *in* and *out*, and the 7-bits *ctrl* bus. Data is read from a register through the *in* bus, and written to through the *out* bus.
The *ctrl* bus is a seven bits address and control bus. This bus defines which register is accessed, and if its for reading, writing or modifying. The contents of the *ctrl* bus are explained as follows :

bit0..4 The physical address of the register which is accessed, with bit0 being the least significant bit.

bit5..6 Indication wether the located register must be read, written or modified.

00	: nop	10	: read
01	: write	11	: modify

As the bit-processor appears in the register space of the byte-processor these three buses are part of the Bit- to Byte-Processor Interface signals.

5.1.3 Synchronization

The TRC uses the external clock signals *ms_clk*, *pl_send* and *pl_load* for receiving and transmitting bits. These clock signals must be synchronous to the system clock of the TRC. Figure 5.5 shows the relation between the incoming and outgoing bit stream and when the clock signals must be asserted.

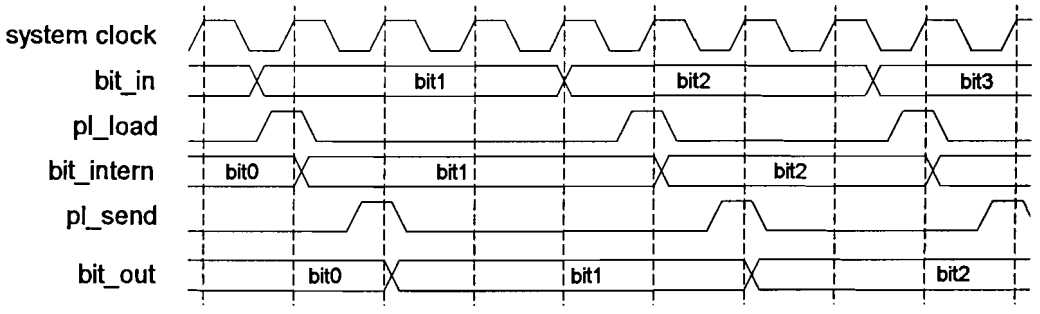


Figure 5.5 Best Case Synchronization

Pl_load is always used to load received bits into the TRC. The external Ring Interface Circuit must extract this signal from the clock recovered by the PLL out of the incoming bit stream, and the system clock.
Pl_send or *ms_clk* is used for transmission timing depending on the mode of the bit-processor, as indicated on *bitc*. (See 5.2.3).
Pl_send may have a maximum delay of 1 bit-time with respect to pl_load, when *pl_send* is used for transmission timing. Otherwise bits will be lost.

Ms_clk must be a clock signal extracted from the master oscillator. *Ms_clk* has no direct relation with *pl_load*. When the frequency of *ms_clk* and *pl_load* differ to much the latency buffer, when inserted, will of course over/under-run.

The TRC introduces a delay in the bit stream even when the TRC is a Standby Monitor. The amount of delay is dependent on the relationship between *pl_load* and the incoming bit stream, and the relationship between *pl_load* and *pl_send*.
Figure 5.5 shows the best case, and figure 5.6 the worst case situation of the delay. As these figures shows the delay varies between 1 system clocks and 2 bit-times.

The frequency of the incoming and outgoing, when Standby Monitor, bit stream may fluctuate in time. In order to guarantee the proper functioning of the TRC, a system clock rate of at least twice the bit rate is advised.

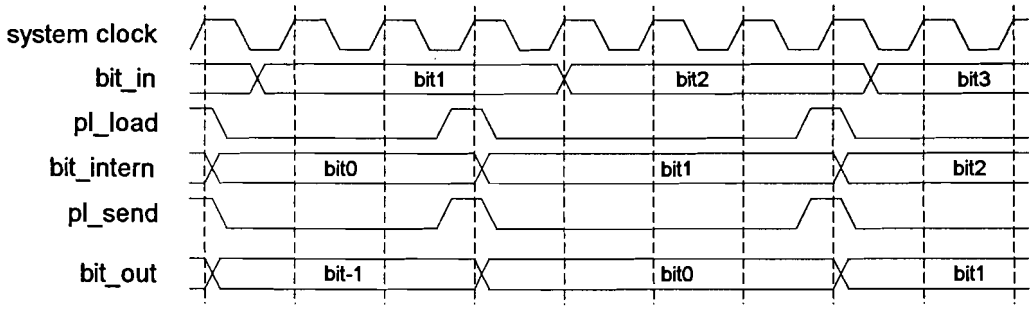


Figure 5.6 Worst Case Synchronization

5.1.4 Characteristics

At this time the monitor program for the TRC has not been developed. The architecture of the TRC possesses the following characteristics and application possibilities:

- 16 or 4 Mbit/s ring interface.
- Universal or Local Administration
- Early Token Release
- Internal loop test
- Wiring fault indication
- PLL Lock indication
- Delay between 1.5 system clocks or 2 bit times
- Minimum system clock of twice the bit rate.

The first three 'parameters' have to be set during the initialization of the TRC. The first two of these parameters have to be set according to the type of ring system the TRC is connected to. Stations using the Early Token Release option are compatible and interoperable with stations that do not.

The monitor program can be featured with a routine for performing an internal loop test. The loop test can be performed with or without an external delay. When no external delay is used, the latency buffer must be inserted by making the TRC the Active Monitor. (See chap. 5.2.3)
The internal loop test, wiring and phase lock error indication can only be used when the external Ring Interface circuit provides these options.
As explained in the previous paragraph, the TRC introduces a delay between 1 system clocks and 2 bit times. So special care should be taken with the generation of the clock signals. For the proper functioning of the TRC a system clock rate of 32 Mhz is advised.

5.2 Bit- to Byte-Processor Interface

The communication between the bit- and byte-processor is based on four communication channels. A Control channel is provided to read status information from the bit-processor and to initiate actions in the bit-processor. The byte-processor fetches received bytes out of, and places bytes for transmission into the bit-processor, through the Data channel. The configuration of the bit-processor is set through the Setting channel. This channel is provided to indicate status of the station, wether it's the Active monitor, operating on 4 or 16 Mbit/s, it has recognized the stations address in the received frame etc.
These first three channels appear in the register space of the byte-processor. Therefore using these channels is the same as performing an operation on every other register, with one exception. The function of some of these communication registers during a read operation is different from when performing a write operation. A modify operation on these registers causes a write and a dummy read operation. The dummy read operation puts zero on the *in* bus. Register 1 is read only!!
Through the fourth channel the bit-processors signals when a particular event has taken place. This channel is called the Event channel.

5.2.1 Control

The Control channel is located at the physical addresses 0,1 and 2 in the register space. Through this channel the byte-processor can initiate actions, read status information, and initialize a timer. The functions of the three registers are as follows:

Register 0 , write; Function : COMMAND register
Writing to register 0 initiates an action in the bit-processor dependent on the command written to it. Table IV indicates for which command what code must be written into the Command register, and what action is initiated. Thsi table refers to several Transmit states of the bit-processor. These states are discussed in chapter 5.3.1.

Table IV Command Register Commands

Command	Code	Action
NOP	00000000	No Action
LAST_BYTE	XXXX0001	Marks the last byte in the transmit buffer as the last byte of the frame in transmission.
LAST_FRAME	XXXX0010	Marks the frame in transmission as the last frame to be transmitted (I=0).
LAST_B&FR	XXXX0011	Marks the last byte in the transmit buffer as the last byte of a frame, and the frame in transmission as the last frame to be transmitted (I=0).
TX_FR	XXXX0100	Start transmission of a new frame. This command may only be given when the bit-processor is in the Transmit Halted states TX Zeros or TX Frame, or in the Transmit Operational states TX Frame or Wait. (See chap. 5.3.1)
TX_TK	XXXX1000	Starts the transmission of a token. This action may only be initiated when the bit-processor is in the Transmit Halted states TX Zeros or TX Frame. (See chap. 5.3.1)
TX_AB	XXXX1100	Aborts the transmission of a frame with an abort sequence immediately. This command has no effect when the bit-processor is not in the Transmit Halted/-Operational state TX Frame.
CLEAR_TX	XXXX1111	Clears the Transmit buffer.
READ_LEC	0001XXXX	The value of the Line Error Counter will be given on the following read actions on register 2, after this command is given.
READ_BEC	0010XXXX	The value of the Burst Error Counter will be given on the following read actions on register 2, after this command is given.
READ_LFEC	0011XXXX	The value of the Lost Frame Error Counter will be given on the following read actions on register 2, after this command is given.
READ_TXAB	0100XXXX	The value of the Abort Delimiter Transmitted Counter will be given on the following read actions on register 2, after this command is given.
READ_CNTS	0111XXXX	The indication of which counters are overflown will be given on the following read actions on register 2, after this command is given.
RESET_TRR	1000XXXX	Resets the Timer Return to Repeat.
RESET_ALL	11111111	Resets the complete bit-processor.

Register 0, read; Function: TRANSMIT STATUS register
Reading from register 0 gives the status information on the transmit process. This byte indicates the following status:

Bit0..2 : P
The first three bits of the transmit register indicate the priority at which the ring system is operating.
Bit3..4 : TX State
Bit 3 and 4 indicate the status of the transmission of data as follows :

00 No transmission in progress
01 Wait. When the transmission is in this state, the bit-processor waits for the transmit frame command (TX_FR). When this status is shown the transmit process is in the Operational state Wait (See 5.3.1).
10 The bit-processor is transmitting a frame.
11 The bit-processor is transmitting a token.

Bit5 : Abort
If bit5 is set, the bit-processor is aborting the transmission with an abort sequence.

Bit6 : Inserted
Inserted indication. This bit is set as long as the Ring Interface connector *inserted* is asserted high, indicating that the TRC is inserted in the ring system, or the loop circuit is established.
Bit7 : TX Buf Full
This bit is set during the time the transmit buffer is full.

Register 1, read; Function : RECEIVE STATUS register (read)
The bit-processor passes status information regarding to the receive process when register 1 is read. The following receive status is passed when reading this register:

Bit0 : Data Error bit
This bit is set when a token is received with a code violation, or when a Frame with Error is received. When a frame was received, this bit resets when this register is read after the frame has passed, or a new frame/token arrives. If a token was received this bit is reset on reading the register, or when a new frame/token arrives.
Bit1 : Not Good bit
When a token is received that did not end properly, or a Frame Not Good is received this bit is set. When a frame was received, this bit resets when this register is read after the frame has passed, or a new frame/token arrives. If a token was received this bit is reset on reading the register, or when a new frame/token arrives.
Bit2 : End Error
Set when a frame did not end properly, hence the frame is aborted or an SD is received before the closing ED. This bit resets when this register is read, or a new frame/token arrives.
Bit3 : TX Buf Overflow
When the receive buffer overflows this bit is set. This bit resets after the complete frame has passed, and this register is read.
Bit4 : CNT Overflow
This bit is set when one of the Error Counters overflows. It resets when the overflown counters are read.(Reading also resets a counter).
Bit5 : TRR Expired
This bit is set when the Timer Repeat to Return has expired. This bit resets when the RESET_TRR command is written in the Command register, or the bit-processor resets the TRR itself.
Bit6 : Wiring Error
The wiring error bit is set as long as the Ring Interface connector *werr* is asserted low, indicating the existence of a wiring fault.
Bit7 : PLL Error
The pll error bit is set as long as the Ring Interface connector *lock* is asserted low, indicating that the PLL has not been able to lock within the allowed frequency range.

Register 2, write; Function: INITIAL TRR register
When a value X is written into this register, then the time-out value of the Timer Return to Repeat is set to X**clk*. *Clk* is the clock signal from the timers of the byte-processor. This signal and the timers have not been implemented yet.

Register 2, read; Function: COUNTER register
Reading from register 2 will give counter information. Which counter information is passed is depend on the last READ command written in the Command register (0). When one of the counters is read, the value of the counter is passed, and the counter resets. If the READ_CNTS command was the last READ command given, the bit-processor passes the following information:

bit0 1 if the Line Error Counter has overflown. Otherwise 0.
bit2 1 if the Burst Error Counter has overflown. Otherwise 0.
bit3 1 if the Lost FRame Error Counter has overflown. Otherwise 0.
bit4 1 if the Abort Delimiter Transmitted Counter has overflown. Otherwise 0.
bit5..7 Not used.

5.2.2 Data

The bit-processor contains two data buffers. A three bytes transmit buffer, and a two bytes receive buffer.
The transmit buffer is used to queue the bytes of a frame for transmission. The byte-processor only has to queue the bytes for the FS, DA, SA and INFO fields of a frame, as the other fields (SD, AC, FCS, ED and FS) are generated by the bit-processor (See chap. 5.3.1). Tokens are also generated by the bit-processor.
The bit-processor queues the received bytes in the receive buffer, in order to being processed by the bit-processor. The bit-processor queues the bytes of the AC, FS, DA, SA, INFO and FS fields of a frame and the AC field of a token. The presence of a byte in the Receive buffer is signalled through the Event channel.
The two data buffers can be accessed through the Data channel which is located at the physical address 12 in the register space.

Register 12, write; Function: TRANSMIT BUFFER register
A write operation on register 12 queues a new byte at the back of the transmit buffer. Writing to a full transmit buffer will cause loss of data!!! Bit 7 of the Transmit Status indicates wether the buffer is full or not.

Register 12, read; Function: RECEIVE BUFFER register
A read operation on register 12 pops the first received byte from the receive buffer. Reading from an empty receive buffer will cause insertion of a ghost byte which is never received. The event bus indicates when a byte is present in the buffer (bit0, bit1 or bit2 set). A read operation resets bit0, bit1 and bit2 of the *event* bus.

5.2.3 Setting

The registers located at the physical addresses 10 and 11 are directly tapped to the bit-processor (*bitc* bus). Through this channel the byte-processor sets the configuration of the bit-processor and passes some status information of the frame being received and to be transmitted. The *bitc* bus contains the following flags and information (register10 = lower byte) :

Bit0 : 16/4
The bit-processors configuration is set for connection to a 16Mhz Token Ring system when this bit is set. Otherwise the bit-processor is set for a 4Mhz ring.
Bit1 : ETR
The bit-processor uses the Early Token Release option when this flag is set.
Bit2 : FR_LENGTH
The bit-processor expects 6 byte addresses when this flag is set. Otherwise 2 byte addresses are expected.
Bit3 : CONNECT
During the time this flag is set the bit-processor tries to insert into the ring system, by asserting the Ring Interface connector *connect* active high. Bit 6 of the Transmit Status register indicates wether the TRC is indeed inserted.
Bit4 : LOOP
Through this bit the byte-processor indicates it wanst to perform an internal loop test. The bit-processor asserts the Ring Interface connector *loop* active low during the time LOOP is set[1]. Bit 6 of the Transmit Status register indicates wether the loop circuit has been established.
Bit5 : ACTIVE
The bit-processor assumes its the Active Monitor during the time this flag is set[2]. The bit-processor inserts the latency buffer when this bit is set[1], and performs the extra Active Monitor functions (See chap. 5.3.1.3).
Bit6 : FORCE_TX
When this bit is set, the bit-processor transmits a frame or token directly on the TX_FR or TX_TK command[3]. (See 5.3.1.2)
Bit7 : RESUME
The functions of the Operational FSM, which are build in the bit-processor, are suspended during the time this bit is reset.
Bit8..10 : Pm
Bit0..2 indicate the priority of the frame queued in the transmit buffer for transmission.
Bit11 : PDUQ
When this bit is set the bit-processor tries to capture a token for the transmission of a frame with priority Pm.

note 1 : An external delay must be provided when performing a loop test (LOOP=1), or the latency buffer must be inserted by setting the ACTIVE bit.
note 2 : Transmission timing is based on the master-oscillator (*ms_clk*) when bit5 or bit6 are set.
note 3 : Bit6 and bit7 should never be set at the same time. This causes an undefined bit-processor mode.

Bit12 : MA Flag
The bit-processor assumes that the byte-processor has received a frame with a Source Address equal to its station address, when this bit is set.
Bit13 : NOT_MA Flag
The bit-processor assumes that the byte-processor has received a frame with a Source Address not equal to its station address, when this bit is set.
Bit14 : ADR
If this bit is set the bit-processor sets the address recognized bits in the FS field of the repeated frame.
Bit15 : COPY
If this bit is set the bit-processor sets the frame copied bits in the FS field of the repeated frame.

5.2.4 Event

The bit-processor signals particular receive events through the *event* bus to the byte-processor. The byte-processor switches from the transmit to the receive process when one or more bits of this bus are set[1].

Bit0 : Byte received
This bit is set when a new byte of a frame is ready to be fetched from the receive buffer. Reading from the receive buffer resets this bit.
Bit1 : FS received
This bit is set when the last byte (FS field) of a frame is ready to be fetched from the receive buffer. Reading from the receive buffer resets this bit.
Bit2 : Token Received
This bit is set when the AC field of a token is ready to be fetched. Reading from the recieve buffer resets this bit.
Bit3 : Error
This bit is set when an error occurs in a frame or token, an error counter overflows, the timer TRR expires, the Reciev buffer overflows, or a fault in the physical connection exists. Reading the Receive Status register (register 2) is necessary to identify the type of error. Reading this register will also resets this bit. Chap 5.3.2 gives an extensive specification of the functioning of this bit.

5.2.5 Clock

One clock signal for the bit-processor must be provided by the byte-processor. This clock is used to control the timer Return to Repeat. The timer TRR increments, synchronous with the system clock, when *clk* is asserted high. *Clk* is asserted high at the rate of ones a 0.1 ms. The time-out value of the timer is written into the INITIAL TRR register (register 2).

note 1 : The value of the *event* bus must be loaded somewhere in the free register space of the byte-processor (Counter register ??). This is necessary for the byte-processor in order to process an event. The byte-processor does not recognize the type of event on the *event* bus.

5.3 Bit-Processor

The bit-processor of the TRC implements almost the complete functionality of the Operational Finite-State Machine and Receive Actions as defined in [IEEE802.5]. Also the bit-processor gives the possibility to transmit a frame, token or abort sequence immediately on command from the byte-processor. This feature can be used to transmit when the monitor is in the Transmit Claim Token, Transmit BCN, Transmit Fill, or Transmit Purge states, hence when the activity of the Operational FSM is suspended.
The operation of the bit-processor is divided in two processes. A transmit and a receive process. The operation of these two processes is completely independent as they are controlled by separate finite state machines, registers and control commands.
The process of inserting in the ring will not be outlined in here, as this is controlled by the byte-processor.

5.3.1 Transmit Process

The transmit process of the bit-processor has two 'main'-states ; Operational and Halted.
In the Operational state, the bit-processor transmits its frames according to the specifications of the Operational FSM of IEEE 802.5. If the process is Halted, frames and tokens are transmitted when the transmit command is written into the Command register. The Halted state is used for the transmission of frames and tokens when the station is in monitor states Transmit Claim Token, Transit BCN, Transmit Fill, or Transmit Purge. By setting and resetting the Resume bit on *bitc* the transmit process switches between these two states.
For the transmission of frames the byte-processor has only to put the bytes of the FC, DA, SA and INFO fields in the transmit buffer. The SD, AC, SFS, ED and FS fields are generated by the bit-processor. Tokens are completely generated by the bit-processor.
Switching from Operational to Halted resets the priority stacks Sx and Sr.

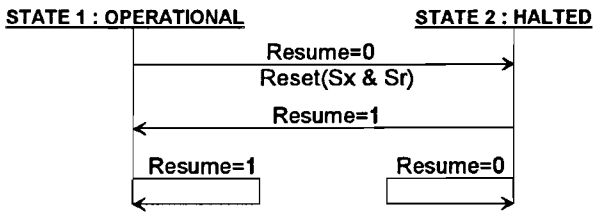


Figure 5.7 Transmit Finite State Machine

5.3.1.1 Operational

In the Operational state the transmit process can be presented as a FSM which is quite similar to the Operational FSM of the IEEE 802.5 standard.
In this state the byte-processor can queue a PDU for transmission. For queueing a frame the following three actions have to be performed by the byte-processor :

1. place bytes in the transmit buffer, by writing to the Transmit Buffer register.
2. set the priority bits in *bitc*, Pm (bit8..10), to the priority of the queued frame.
3. set the PDUQ bit (bit 11) on *bitc* indicating that a frame is queued for transmission.

In the Operational state the processor uses the master oscillator (*ms_clk*) for transmission timing when the station is the Active Monitor (Active bit on *bitc* equals 1). Otherwise the processor uses the PLL clock (*pl_clk*) for transmission timing, when in the Operational state.

State TO0 : REPEAT
In this state the incoming bit stream is repeated, and certain bits or fields may be modified without changing state. A transition to another state is made when a token is captured.

TO01 : Usable Token Received
If a PDU is queued for transmission (PDUQ = 1) and a token is received whose priority (P) is equal to or less than the PDU priority (Pm), the bit-processor changes the token into a SFS (by changing the token bit from 0 to 1), transmits M and R as 0, initiates the transmission of the byte(s) in the buffer, and sets the Frame Counter to 1. Transition is made to state TO1A.

TO02 : Resume
Every time when the Transmit process is set Operational by changing the Resume bit from '0' to '1', the Transmit Operational FSM is resumed in the Repeat state.

TO03 : Re-Stack Operation
If there is no frame queued with priority (Pm) equal to or greater than the Sx and a token is received with priority (P) equal to the Sx, the following actions are taken: The token is changed into a SFS, timer TRR and the SFS flag reset and transition is made to state TO4.

Table V Transmit Action Table

REF	Input	Action
TO0A	PDUQ & (FR(R<Pm) ¦ TK(P>Pm>R & P≠Sx))	SET R=Pm
TO0B	FR_WITH_ERROR	SET E=1
TO0C	DA=MA(ADDRESS RECOGNIZED)	SET A=1
TO0D	FR_COPIED	SET C=1

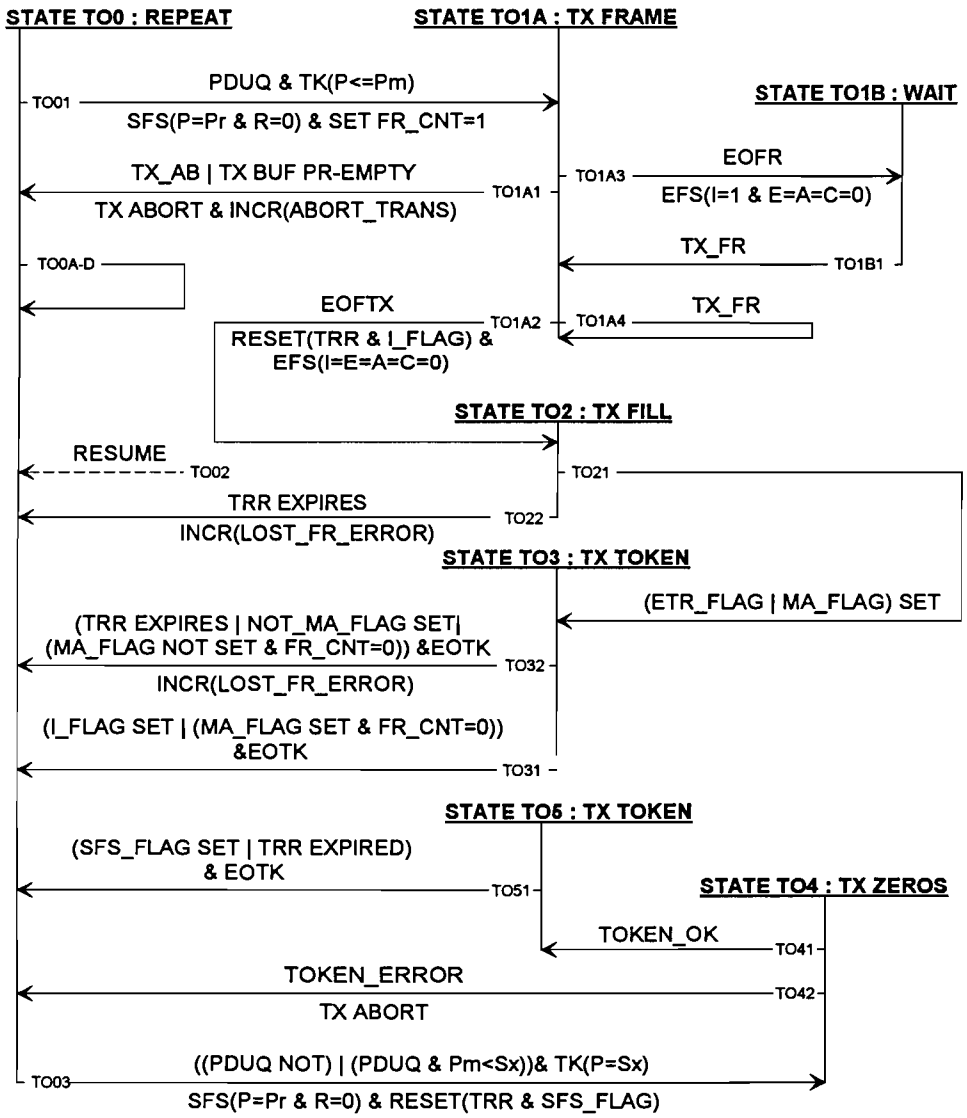


Figure 5.8 Transmit Operational Finite State Machine

A number of actions is made without changing state. These actions are shown in table V and are explained as follows :

TO0A : Request Usable Token
If there is a PDU queued for transmission with priority Pm, the reservation bits are set to Pm on frames with reservation less than Pm, and on tokens with priority greater than Pm and the reservation less than Pm and priority is equal to the Sx.

TO0B : Frame With Error
The E (error) bit is transmitted as 1 if a frame with error is detected. (See chap. 2.3.4) Otherwise it's repeated unchanged.

TO0C : Own Address Detected
The A bits in the FS field of the repeated frame are transmitted as 1 when the byte-processor has set the Address Recognized bit on *bitc*. Otherwise the A bits are repeated unchanged.

TO0D : Frame Copied
The C bits in the FS field of the repeated frame are transmitted as 1 when the byte-processor has set the Frame Copied bit on *bitc*. Otherwise the C bits are repeated unchanged.

State TO1A : TX FRAME
While in this state the station transmits a frame. When this state is entered on transition TO01 the bytes in the buffer are appended to the captured token that is changed into an SFS. If this state is entered on transition TO1B1, an SFS istransmitted first with priority Pr, and M and R equal to 0.

TO1A1 : Abort Transmission
The transmission of the frame is aborted with an abort sequence when the byte-processor gives the abort command (TX_AB), or when the transmit buffer empties and the last byte was not marked as the last byte of the frame by the byte-processor.
The Abort Delimiter Transmitted counter increments on this transition.

TO1A2 : EOFrame Transmission
This transition will be made after the frame is transmitted completely, if the byte-processor marks the frame in transmission as the last frame to transmit. The I bit in this last frame is transmitted equal to 0. On this transition the bit-processor resets its timer TRR and I flag, and makes transition to state TO21.

TO1A3 : EOFrame
When a byte in the transmit buffer is marked as the last byte of the INFO field of the PDU an FCS, ED and FS are appended to it. After the FS is transmitted this transition to state TO1B is made.

TO1A4 : Transmit new frame
The transmission of a new frame starts immediately after the frame is transmitted, when the TX_FR command is given during the transmission of this frame. The Frame Counter increments on this transition.

State TO1B : WAIT
In this state the bit-processor transmits zero's and waits for a command of the byte-processor to restart the transmission. In this state the byte-processor should only restart transmission of a frame by giving the TX_FR command.

TO1B1 : Restart Frame Transmission
This transition is made when the byte-processor gives the TX_FR command, to state TO1A. The Frame Counter increments on this transition.

State TO2 : TX Fill
In this state the bit-processor transmits zero's until the ETR or MA flag is set, or the TRR expires.

TO21 : Token Transmission
If the header of the frame is received (MA Flag set) or the ETR option is selected, the bit-processor starts with the transmission of a token.

TO22 : TRR Expires
If, while waiting for the MA Flag to be set, timer TRR expires, transition is directly made to the Repeat state (TO0), and the Lost Frame Error Counter is incremented.

State TO3 : TX TOKEN & Fill
In this state the bit-processor releases the token with new priority and reservation. The processor performs the necessary stacking operations on Sx and Sr, and starts transmitting zero's, after the token is released. The following token is released, and stacking operations are performed :

If Pr > Sx	and Pr ≥ Px	then TK(P = Pr and R = Px)	
If Pr > Sx	and Pr < Px	then TK(P = Px and R = 0)	and STACK(Sx = P and Sr = Pr)
If Pr = Sx	and Sr ≥ Px	then TK(P = Sr and R = Px)	and POP(Sx and Sr)
If Pr = Sx	and Rs < Px	then TK(P = Px and R = 0)	and RESTACK(Sx = P)

(Px = MAX(Pm,Rr), Sx(Sr) is top of stack Sx(Sr))

TO31 : Strip Complete
After the token is transmitted this transition is made if the I flag is set, or the MA flag is set and the ED of the last frame (Frame Counter = 0) is received.

TO32 : Lost Frame
After the token is transmitted and the TRR expires, or if the SA of the last frame transmitted is received that does not equal MA (NOT_MA flag set), or if while waiting for the MA flag to be set, the ED of the last frame transmitted is received (FR_CNT = 0 and MA flag not set), the Lost Frame Error Counter is incremented and transition is made to the Repeat state (TO0).

State TO4 : TX ZEROS
The bit-processor transmits 0's following the SFS until one of the following transitions can be made.

TO41 : Token Transmission
When the bit-processor recognizes that the captured frame ended properly, transition is made to state TO5.

TO42 : Token Error
If after changing the token to an SFS, the bit-processor detects that the token did not end properly, the transmission is aborted with an abort sequence. Transition is made to state TO0.

State TO5 : TX TOKEN & FILL
In this state the bit-processor releases a new token with a new priority and reservation. When the token is released the processor performs the necessary stacking operations. The token released and the stacking operations performed are the same as in the state TO3.
After the token is transmitted the bit-processor transmits zero's until the conditions for a state transition is valid.

TO51 : Strip Complete
Upon reception of the SFS or TRR expiring, transition is made to state TO0.

5.3.1.2 Halted

In the Halted state, the transmission of information is directly controlled by the byte-processor. For the transmission of frames the byte-processor has to perform the following two actions :

1. place bytes in the transmit buffer, by writing to the Transmit Buffer register.
2. start the transmission with the command TX_FR.

Transmission of a token is initiated when the TX_TK command is given. The bit-processor generates the complete token. The processor uses the PLL clock (*pl_clk*) for transmission timing in the Halted state Idle. Otherwise the master-clock (*ms_clk*) is used, when in the Halted state.
The following finite state machine defines the transmit process in the Halted state.

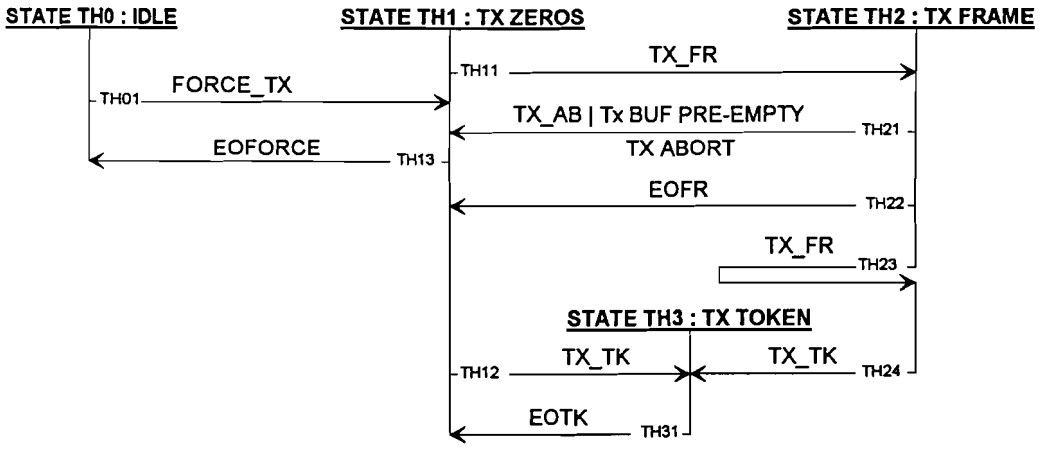


Figure 5.9 Transmit Halted Finite State Machine

State TH0 : IDLE
In the IDLE state the bit-processor repeats the incoming bit stream without change. A state transition from this state is always initiated by the byte-processor.

TH01 : Force Transmission
This transition is made when the FORCE_TX bit on the *bitc* bus is set.

State TH1 : TX ZEROS
In the TX ZEROS state, the processor transmits zeros on the timing of the master-oscillator (*ms_clk*).

TH11 : Start Frame Transmission
The transmission of a frame starts when the TX_FR command is given.

TH12 : Start Token Release
The release of a new token starts when the TX_TK command is given.

TH13 : End of Forced Transmission
By resetting the FORCE_TX bit, the bit-processor returns to the IDLE state. The FORCE_TX bit may only be reset when the bit-processor is in the TX ZEROS state. Otherwise the transmission of the frame or token can be corrupted!!!

State TH2 : TX FRAME
In this state the bit-processor transmits data frames with the P, M and R bits 0. The SD, AC, FCS, ED, and FS are generated by the bit-processor, hence the byte-processor only has to place the bytes for the FC, DA, SA and INFO field in the transmit buffer.

TH21 : Abort Transmission
The transmission of the frame is aborted with an abort sequence when the byte-processor gives the abort command (TX_AB), or when the transmit buffer empties and the last byte was not marked as the last byte of the INFO field by the byte-processor.

TH22 : EOFrame
When a byte in the transmit buffer is marked as the last byte of the INFO field of the PDU an FCS, ED and FS are appended to it. After the FS is transmitted this transition to state TH1 is made. If the frame was marked as the last frame to be transmitted, the I bit is transmitted equal 0. Otherwise the I bit is transmitted equal 1.

TH23 : Transmit new frame
The transmission of a new frame starts immediately after the frame is transmitted, when TX_FR is the last transmit command given during the transmission of this frame.

TH24 : Transmit Token
The transmission of a new token starts immediately after the frame is transmitted, when TX_TK is the last transmit command given during the transmission of this frame.

State T3 : TX TOKEN
In this state the bit-processor transmits a new token with the priority set to Rr and the reservation set to 0. At the end of the transmission the priority and reservation of the transmitted token are stacked on Sx and Sr. The TX TOKEN state should only be used when the station is the active monitor and it must release a new token.

TH31 : EOToken
When the token is transmitted transition is made to state TH1.

5.3.1.3 Active

When the Transmit process is in the Operational state and the Active bit on *bitc* is set, the bit-processor performs two more tasks. As the following figure shows, both actions are initiated from the Repeat state.

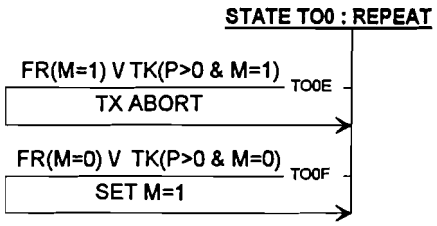


Figure 5.10 Transmit Active Finite State Machine

TO0E : M-bit error
When the TRC is the Active Monitor, and it repeats a token with priority (P) greater than zero and M equal to 1, or any frame with M equal to 1, it aborts the token or frame with an abort sequence.

TO0F : Set M-bit
When the TRC is the active monitor, and it repeats a token with priority (P) greater than zero or any frame, it sets the M-bit equal to 1.

5.3.2 Receive Process

The receive process of the bit-processor makes no distinction wether its Operational or Halted. Several actions are performed with no direct relation to the state of the receive process. Table V summarizes these actions. All the actions shown in that table are performed by the bit-processor without intervention of the byte-processor.

RX1 : Line Error
If the frame received is a Frame With Error with the E bit equal to 0, then the Line Error Counter is incremented.

RX2 : Priority Level Error. If there is a Sx stored and a token is received with a priority (P) less than the value Sx, then the stacks are cleared.

RX3 : AC Field Received
The priority (P) and the reservation (R) of a received token or frame are stored as Pr and Rr. The old values of Pr and Rr are discarded.

RX4 : I Bit Equal Zero received
If an EFS with I bit equal to 0 is received the I Flag is set.

RX5 : SFS Received
The SFS Flag is set when an SFS is received.

RX6 : The Frame Counter is decremented when an ED is received.

Table VI Receive Action Table

REF	RECEIVE	ACTION
RX1	FR_WITH_ERROR & E=0	INCR(LINE ERROR)
RX2	TK(P<Sx)	CLEAR STACKS
RX3	TOKEN | FRAME	STORE(Pr & Rr)
RX4	I=0	SET I_FLAG
RX5	SFS	SET SFS_FLAG
RX6	RCV_ED	DECR(FR_CNT)

As mentioned before the bit-processor signals particular receive events through the *event* bus (Chap. 5.2.4). The error bit (bit3) on this bus is set under various conditions.
The error bit is set immediately when:

- The received frame did not end properly, at which time the End Error bit (bit2) of the Receive Status register is set also.
- One of the error counters overflows, at which time the CNT Overflow bit (bit4) of the Receive Status register is set also.
- The Timer Repeat to Return expires, at which time the TRR Expired bit (bit5) of the Receive Status register is set also.
- A wiring error occurs, at which time the Wiring Error bit (bit6) of the Receive Status register is set also.
- A PLL lock error occurs, at which time the PLL Error bit (bit7) of the Receive Status register is set also.

The error bit is also set when the first new byte is ready to be fetched from the Receive buffer, after an error in the data has occurred. So the error bit and one of the three receive bits (bit0 to bit2) on the *event* bus are set.
At this time the Data Error, Not Good, or TX Buf Overflow bit of the Receive Status register is (are) set also, dependent on the type of error that has occurred. The Data Error bit, when set, remains set until the Receive Status register is read after the last byte of the frame , the FS field, or the first and last byte of the token is fetched from the Receive buffer. The same goes for the Not Good and TX Buf Overflow bit.
The error bit in the *event* bus is reset on reading the Receive Status register. But from the time an error has occurred in the received frame, the presence of a new byte from that frame will be signalled with the error bit set. This is performed to keep reminding the byte-processor of the occurrence of an error in that frame.

note 1 : The error bit on the *event* bus always, and only, resets on reading the Receive Status register. The error bits in the Receive Status register reset when the particular error is over. So when the error bit on *event* is reset, that doesn't mean the wiring error is solved, for instance.

note 2 : When a new frame or token arrives before the Receive Status register is read to identify the type of error that has occurred in the previous frame or token, the Data Error, Not Good and End Error bits are reset.

5.4 Byte-Processor

The byte-processor used in the TRC is a slightly modified version of the processor designed by dr.ir. Verschueren for an HDLC/SDLC controller. The modifications are made with respect to *bitc* bus and an additional set of programmable timers.

The architecture looks somewhat like a very trimmed-down 8051, using 32 addressable registers. All instructions are 20 bits wide, while most key operations take only one control word and one clock-cycle (as opposed to three in an earlier architecture).

The byte-processor must perform the following tasks :

0. Initialization.
1. Control bit-processor, which is implementing the Operational Finite-State Machine.
2. Implement the Standby Monitor Finite-State Machine.
3. Implement the Active Monitor Finite-State Machine.
4. Transfer frames to and from an external memory (DMA)

These tasks are performed by the monitor program, which is placed in the ROM of the byte-processor. This monitor program has not been developed at this time.

A special instruction set was designed by dr.ir. Verschueren. This instruction set is specifically targeted at the operations that can be found in handling SDLC/HDLC-like protocols, like the token ring protocol. Some operations have different behaviour depending on flag settings ('normal' or 'modified' behaviour which generally amounts to manipulating bit fields). The instruction set can be found in appendix 1.

The processor is build to allow full duplex operation which is a necessity for implementing the token ring protocols. Full duplex operation is established through a mechanism that switches rapidly between transmitting and receiving. In general, the receive process is event driven, it performs some actions until it must wait for something, at which time the transmit process continues. When the event occurs, the receive process is continued automatically. The interrupt based like reception can be introduced with a special 'WAIT' operation and event recogniser, to switch from sending back to receiving.
The events on which the switching is done, are a combination of the events from the bit-processor (the *event* bus), DMA and CPU process. Switching between the two processes takes no time at all.
Several registers in the byte-processor are duplicated to allow the receive and transmit processes to have their own resources.

The architecture allows 32 registers to be accessed, of which the lower 16 are bit addressable. Register 10 and 11 are directly tapped around the byte-processor for controlling the bit-processor (*bitc* bus). The registers 0, 1, and 2 are physicly located in the bit-procesor. These registers are used to transfer bytes and initiate actions. The contents of these registers is defined in chap 5.2 Bit- to Byte-Processor Interface.

The registers are addressed with a local 5 bits address, which need not map directly onto addresses used by the main processor (address translation is done in hardware - using two separate addressing circuits).
An ACCU register (not bit addressable, at register 31) is available as source and/or destination for several two-operand instructions and for register-register moves. One TEMP register is available for temporary storage and mask holding. One FLAGS register is available which holds carry and zero flags. It holds also flags that can change the behavior of compare and arithmetic instructions. The ACCU, TEMP and FLAGS registers are duplicated for receive and transmit processes (This also holds for the program counter and subroutine return registers). The bit addressable register FLAGS is located in register 15, and the (not bit addressable) register TEMP is located in register 30.

The CPU Interface of the byte-processor also contains a set of timers, as mentioned before. These timers are not implemented at this moment. The impementation to make is bounded by two restrictions.
First, 8 timers must be provided wich are accessible through a minimum number of registers. Only the free register space may be used. Table VII shows which registers are used.
Secondly, this set of timers must generate the *clk* signal. This signal must give a synchronous puls at the rate of once a 0.1 ms.

Table VII Overview of used Registers

Reg.nr		Reg.nr	
0	write : COMMAND read : TRANSMIT STATUS	16	DMAADDRLO *
1	write : Dummy function read : RECEIVE STATUS	17	DMAADDRHI *
2	write : INITIAL TRR read : COUNTER	18	DMACNTLO *
3	not used	19	DMACNTHI *
4	not used	20	CPUREGISTER
5	not used	21	CPUREGISTER
6	not used	22	CPUREGISTER
7	not used	23	CPUREGISTER
8	CPUREGISTER	24	CPUREGISTER
9	CPUREGISTER	25	CPUREGISTER
10	CPUREGISTER	26	CPUREGISTER
11	CPUREGISTER	27	CPUREGISTER
12	write : TRANSMIT BUFFER read : RECEIVE BUFFER	28	not used
13	DMADATA *	29	not used
14	DMAFLAGS *	30	TEMP *
15	FLAGS *	31	ACCU *
^ Bit testable		^ Not bit testable	

'*' : These registers are seperate for the Transmit and Recieve processes.

Conclusions

The past five chapters (hopefully) gave the reader an insight in the world of Token Ring and the architecture of the designed controller. As known by now, the right to transmit frames and the re-transmission of a token is based on quite a complicated process. This process is specified in [IEEE802.5] in the Operational FSM.

The bit-processor of the controller implements almost the complete functioning of the Operational FSM. Address recognition and the controlling of the THT timer are the only tasks to be performed by the byte-processor, in order to implement this FSM completely. When the ring is not functioning properly, a process must be started to re-initialize the ring. During this process the station transmits frames without regarding the access mechanism of the Operational FSM. Therefore the bit-processor gives the possibility to transmit frames directly on command from the byte-processor.

The byte-processor used in the controller, was originally developed for a HDLC/SDLC controller. At this time the byte-processor is slightly redesigned in order to synthesize it. Therefore the byte-processor has not been modified for this controller. A set of timers, which have to be controlled to the free register space of the byte-processor, must still be designed. Also the monitor program has to be developed.
Because of the ongoing modification, the appendices do not contain the document files of the byte-processor.

The byte-processor recognizes an event on the event bus, but cannot be identified. Therefore, the value of the event bus must be stored in register space of the byte-processor. In this way the byte-processor is able to identify the type of event.
At this moment the bit-processor does not store the value of the event bus in the register space. Advised is to expand the functioning of the Counter register in a way that reading from the Counter register also passes the value of the event bus. An extra READ command, READ_EVENT = %0101XXXX for example, must be provided.

The reader interested in the latest developments of the controller can contact the Digital Information Systems section of the faculty of Electrical Engineering at the Eindhoven University of Eindhoven.

Literature

[Blai89]
Blair J.D. et al.
A 16-Mbit/s Adapter Chip for the IBM Token-Ring Local Area Network.
IEEE Journal of Sloid-State Circuits., Vol.24(1989), No.6, p.1647-1654.

[Carl86]
Carlo J.T. and G.R. Samsen
VLSI Token-Ring Network Communications Adapter Implementation Options.
In: Conference on Local Computer Networks. Proc. 11th Conf., Minneapolis, 6-8 October 1986.
Washington: IEEE Computer Society Press, 1986.
P. 100-104.

[Carl87]
Carlo J.T. and J.M. Hughes
TMS380 Token Ring Adapter with Built-in Network Management.
In: IEEE International Conference on Computer Design. Proc. ??th Annual Conf., 5-8 october 1987.
New York: IEEE, 1987.
P. 96-99.

[Coch85]
Cochran R.G. and G.R. Samsen
A VLSI Chipset for Token-Ring LAN's
In: IEEE Phoenix Conference on Computers and Communications. Conf Proc., Scotsdale (Ar.), 20-22 March 1985.
Silver Spring: IEEE Computer Society Press, 1985.
P. 428-431.

[IEEE802.2]
IEEE 802.2
IEEE Standards for Local Area Networks: Logical Link Control.
New York: IEEE, 1985

[IEEE802.5]
IEEE 802.5
IEEE Standards for Local Area Networks: Token Ring Access Methods and Physical Layer Specifications.
New York: IEEE, 1985.

[Kanu90]
Kanuma A. et al.
A CMOS 510K-Transistor Single-Chip Token-Ring LAN Controller (TRC) Compatible woth IEEE802.5 MAC Protocol.
IEEE Journal of Solid-State Circuits, Vol.25(1990), NO.1, p.132-141.

[Kris89]
Krishnakumar, A.S. and K. Sabnani
VLSI Implementations of Communication Protocols - A Survey
IEEE journal on selected areas in communications, Vol.7(1989), No.7,
P. 1082-1089.

[Lang89]
Lang K.W. et al.
A 16 MBPS Adapter Chip for the IBM Token-Ring Local Area Network.
In: IEEE Custom Integrated Circuits Conference. Proc. 11th Annual Conf.,
San Diego (Cal.), 15-18 May 1989.
New York: IEEE, 1989.
P.11.3.1-11.3.5.

[Laff83]
Laffitte D.S. and M.W. Patrick
An Architecture for VLSI Support of Token Ring Local Area Networks.
In: COMPCON Spring'83, Proc. 26th Annual Conf., San Fransisco,
28 February - 3 March 1983.
San Fransisco: IEEE Computer Society Press 1983.
P. 102-103.

[Marc89]
Marchok T.E. and J.K. Strosnider, H. Tokuda
Token-Ring Adapter-Chipset Architectural Considerations for Real-Time Systems.
In: Real-Time Systems. Proc. 10th Annual Symp., Santa Monica (Cal.), 5-7 December 1989.
Alamitos: IEEE Computer Society Press, 1989.
P.79-89.

[Mart89]
Martin J.
Local Area Networks, Architectures and Implementations.
Englewood Cliffs (NJ): Prentice Hall, 1989.

[Saca84]
Sacarisen S.P. et al.
A VLSI Communication Processor Designed for Testability.
In: IEEE International Solid-State Circuits Conference. Proc. 31th Annual Conf., San Francisco, 22-24 february 1984. Ed. by M. Winner
New York: IEEE, 1984
Vol.XXXVII, first edition, p.172-173.

[Sayd87]
Saydam T. and A.S. Sethi
Token Bus/Ring Local Area Network Management Concepts and Architecture.
In: INFOCOM'87, IEEE Conference on Computer Communications. Proc. 6th Annual Conf., San Fransisco, 31 March - 2 April 1987.
Washington: IEEE Computer Society Press, 1987.
P. 988-993.

[Stal87]
Stallings W.
Handbook of Computer-Communications Standards.
Vol.2, Local Area Networks.
New York: MacMillan, 1987.

[Szcz85]
Szczepanek A. et al.
An IEEE802.5 Compatible Controller with On-chip ROM.
In: IEEE International Solid-State Circuits Conference. Proc. 32th Annual Conf., San Francisco, 13-15 february 1985. Ed. by M. Winner
New York: IEEE, 1985
Vol.XXXVIII, first edition, p.188-189.

[Tana89]
Tanaka K. et al.
VLSI Architecture for IEEE802.5 Token-Ring LAN Controller.
In: IEEE Custom Integrated Circuits Conference. Proc. 11th Annual Conf.,
San Diego (Cal.), 15-18 May 1989.
New York: IEEE, 1989.
P.15.2-15.2.5.

[Tms380]
TMS380 Adapter Chipset User's Guide.
Texas Instruments Inc., DB064, September 1986.

[VERS93]
Verschueren A.C.
Specifications of standard system and I/O buses for the Object-Oriented (Hardware) System Design project library.
Eindhoven University of Technology, Section of Digital Information Systems
Eindhoven: April 21, 1993

Appendices

Appendix 1 : Instruction Set

This document was generated by dr.ir. A.C. Verschueren during the development of the byte-processor for a HDLC/SDLC controller.

File: SDLCCTRL.DOC
Cont: Some ideas for improving the byte processor of the SDLC controller
Frst: January 7, 1993
Date: January 18, 1993
Auth: dr.ir. A.C. Verschueren

1) Introduction

The byte handler of the SDLC controller designed by Pepijn Lavrijssen uses a state machine with more than 400 states just to implement the 8044 subpart of the complete protocol. Preliminary calculations show that the size of the control vector exceeds 50 bits. This state machine cannot be converted into a microprogrammed controller because conditional commands are used in some states.

This text describes a new architecture and instruction set which should suffice to provide the same functionality while reducing both the number of states and the number of bits in the control vector. The architecture looks somewhat like a very trimmed-down 8051 - using 32 addressable registers from which a part is bit-testable. The control vector size is reduced to 20 bits, while most key operations take only one control word and one clock cycle (as opposed to three in the original architecture).

The instruction set is specifically targetted at the operations which can be found in handling SDLC/HDLC-like protocols. Some operations have different behaviour depending on flag settings ('normal' behaviour or modified behaviour which generally amounts to manipulating bit fields).

Several registers are duplicated in the machine to allow receive and transmit processes (see section 4 in this text) to have their own resources. In general, the receive process is event driven - it performs some actions until it must wait for something, at which time the transmit process continues. When the event occurs, the receive process is continued automatically. Switching between processes takes no time at all.

2) Input/Output and storage

The architecture allows 32 'registers' to be accessed, the lower 16 of these are bit addressable. Input and output (both to the processor side and the bit handler) must be mapped in this register space. Mode bits in processor-side registers which are used by the bit handler can be 'tapped' directly and fed around the controller. The registers are addressed with a

local 5 bits address, which need not map directly onto addresses used by the processor (address translation is done in hardware - using two separate addressing circuits).

An ACCU register (not bit addressable, at register address 31) is available as source and/or destination for several two-operand instructions and for register-register moves. One TEMP register is available for temporary storage and mask holding. One FLAGS register is available which holds carry and zero flags. It also holds flags which change the behaviour of compare and arithmetic instructions. The ACCU, TEMP and FLAGS registers are duplicated for receive and transmit processes (this also holds for the program counter and subroutine return registers).

Note: Register and bit locations are defined in the file SDLCREGS.ASM (to be included in each assembly run).

3) Instructions and control vector contents

The control vector contains several fields, including a jump address field which is used for conditional jumps. The lower 8 bits of this field are also used for holding constants. We currently assume the jump address field contains 10 bits (allowing 1024 words of 'program'). This can be reduced to 9 bits (reducing the control vector to 19 bits).

The following is the list of operations which will be supported (question marks indicate don't care bits). The symbols between brackets indicate which of the ZERO and/or CARRY flags are updated as side effect. The CARRY flag indicates a carry out of the full 8 bits adder (can also be used in shift instructions), the zero flag indicates if the result or (result AND TEMP) equals 0 (the latter if a mask is used):

%JJJJJJJJJ000BBBRRRR JB bit,label
Test bit BBB of register 0RRRR, jump to JJJJJJJJJ if it is %1.

%JJJJJJJJJ001BBBRRRR JNB bit,label
Test bit BBB of register 0RRRR, jump to JJJJJJJJJ if it is %0.

Note: For both JB and JNB, the bit may be specified as reg.bit (reg in range 0..15, bit in range 0..7) or as direct bit address (calculated as reg * 8 + bit, in range 0..127).

%JJJJJJJJJ01CXXRRRRR
Conditional operations in register RRRRR, jumping to JJJJJJJJJ if the specified condition XOR 'C' holds ('C' complements the test indicated by 'XX'). The actual operation of these instructions is modified by the 'COMPMASK' bit in the FLAGS register:

XX=00: JEQ/JNEQ reg,label
COMPMASK=0:
reg = ACCU
COMPMASK=1:
(reg /\ TEMP) = (ACCU /\ TEMP)

XX=01: JGT/JLTEQ reg,label
COMPMASK=0:
reg > ACCU
COMPMASK=1:
(reg /\ TEMP) > (ACCU /\ TEMP)

XX=10: JZ/JNZ reg,label
COMPMASK=0:
reg = 0
COMPMASK=1:
(reg /\ TEMP) = 0

XX=11: JIEQ/JINEQ reg,label
COMPMASK=0:
(reg + 1) = ACCU
COMPMASK=1: "Same operation, but now on bitfield:"
(((reg /\ TEMP) + (TEMP /\ (TEMP shl: 1) not)) /\
TEMP) = (ACCU /\ TEMP)

%JJJJJJJJJ0100011111
Unconditional jump to JJJJJJJJJ (actually a compare and jump
if ACCU = ACCU, which is always true):

JMP label

%JJJJJJJJJ0110011111
Unconditional call to JJJJJJJJJ (special case, this code
would normally perform a conditional jump on ACCU ~= ACCU,
which is always false):

CALL label

%?YDDDDDDDD100XXRRRRR
Constant operations on register RRRRR, using DDDDDDDD as
constant. Some of these instructions are modified by the
ADDMASK bit in the FLAGS register:

YXX=000: MOV reg,#data
reg := #data

YXX=001: OR reg,#data / SET bit
reg := reg \/ #data [z]

YXX=010: AND reg,#data / RES bit
reg := reg /\ #data [z]

YXX=011: XOR reg,#data / NOT reg / CPL bit
reg := reg >< #data [z]

YXX=100: MOVM reg,#data
reg := reg merge: #data mask: TEMP

YXX=110: ADDC reg,#data
ADDMASK=0:
reg := reg + #data + cy [cz]
ADDMASK=1:
reg := reg merge: reg + #data + cy mask: TEMP [cz]

YXX=111: ADD reg,#data
ADDMASK=0:
reg := reg + #data [cz]
ADDMASK=1:
reg := reg merge: reg + #data mask: TEMP [cz]

Note: the above group is also used to set (OR), reset (AND) and complement (XOR) individual bits. Bits may be specified here by reg.bit (reg in range 0..31 or 'ACCU', bit in range 0..7) or by direct bit addresses (bit address is register number * 8 + bit number, in range 0..255).

%?YDDDDDDDD101XXRRRRR
Constant operations from register RRRRR to ACCU, using DDDDDDDD as constant (MOV and MOVM not present in this group). Some of these instructions are modified by the ADDMASK bit in the FLAGS register:

YXX=001: OR ACCU,reg,#data
ACCU := reg \/ #data [z]

YXX=010: AND ACCU,reg,#data
ACCU := reg /\ #data [z]

YXX=011: XOR ACCU,reg,#data
ACCU := reg >< #data [z]

YXX=110: ADDC ACCU,reg,#data
ADDMASK=0:
ACCU := reg+#data+cy [cz]
ADDMASK=1:
ACCU := reg merge: reg+#data+cy mask: TEMP [cz]

YXX=111: ADD ACCU,reg,#data
ADDMASK=0:
ACCU := reg+#data [cz]
ADDMASK=1:
ACCU := reg merge: reg+#data mask: TEMP [cz]

%?Y??????0P110XXRRRRR
ACCU-to-register operations on register RRRRR. The 'P' bit (if %1) performs a subroutine return in addition to the indicated operation (this is indicated by either including 'RET,' before the parameters or the 'RET' mnemonic itself, which generates a NOP with the 'P' bit set). Some of these instructions are modified by the ADDMASK bit in the FLAGS register:

YXX=000: MOV (RET,)reg,ACCU
reg := ACCU

YXX=001: OR (RET,)reg,ACCU
reg := reg \/ ACCU [z]

YXX=010: AND (RET,)reg,ACCU
reg := reg /\ ACCU [z]

YXX=011: XOR (RET,)reg,ACCU
reg := reg >< ACCU [z]

YXX=100: MOVM (RET,)reg,ACCU
reg := reg merge: ACCU mask: TEMP

YXX=110: ADDC (RET,)reg,ACCU
ADDMASK=0:
reg := reg+ACCU+cy [cz]
ADDMASK=1:
reg := reg merge: reg+ACCU+cy mask: TEMP [cz]

YXX=111: ADD (RET,)reg,ACCU
ADDMASK=0:
reg := reg+ACCU [cz]
ADDMASK=1:
reg := reg merge: reg+ACCU mask: TEMP [cz]

Note: The 'MOV ACCU,ACCU' opcode is used as 'NOP' in this group. The 'RET' is coded as 'MOV RET,ACCU,ACCU'.

%?Y??????0P111XXRRRRR
Register-to-ACCU operations on register RRRRR. The 'P' bit adds a concurrent subroutine return as indicated above. Some of these instructions are modified by the ADDMASK bit in the FLAGS register:

YXX=000: MOV (RET,)ACCU,reg
ACCU := REG

YXX=001: OR (RET,)ACCU,reg
ACCU := ACCU \/ reg [z]

YXX=010: AND (RET,)ACCU,reg
ACCU := ACCU /\ reg [z]

YXX=011: XOR (RET,)ACCU,reg
ACCU := ACCU >< reg [z]

YXX=100: MOVM (RET,)ACCU,reg
ACCU := ACCU merge: reg mask: TEMP

YXX=110: ADDC (RET,)ACCU,reg
ADDMASK=0:
ACCU := ACCU+reg+cy [cz]
ADDMASK=1:
ACCU := ACCU merge: ACCU+reg+cy mask: TEMP [cz]

```
YXX=111: ADD (RET,)ACCU,reg
ADDMASK=0:
ACCU := ACCU+reg [cz]
ADDMASK=1:
ACCU := ACCU merge: ACCU+reg mask: TEMP [cz]
```

%?Y??????1P110XXRRRRR
Unary operations on register RRRRR. The 'P' bit adds a concurrent subroutine return as indicated above. Some of these instructions are modified by the ADDMASK bit in the FLAGS register:

```
YXX=000: SHL (RET,)reg
reg := reg shl: 1 [cz]

YXX=001: SWAP (RET,)reg
reg := reg rol: 4 [z]

YXX=010: SHR (RET,)reg
reg := reg shr: 1 [cz]

YXX=011: SHRC (RET,)reg
reg := cy, (reg from: 1 to: 7) [cz]

YXX=100: INC (RET,)reg
ADDMASK=0:
reg := reg + 1 [cz]
ADDMASK=1: "Same but now on specific bitfield:"
reg := reg merge:
  (reg /\ TEMP) + (TEMP /\ (TEMP shl: 1) not)
    mask: TEMP [cz]

YXX=101: DEC (RET,)reg
ADDMASK=0:
reg := reg + 255 [cz]
ADDMASK=1: "Same but now on specific bitfield:"
reg := reg merge:
  (reg /\ TEMP) + TEMP
    mask: TEMP [cz]

YXX=110: NEGC (RET,)reg
ADDMASK=0:
reg := reg not + cy [cz]
ADDMASK=1: "Modify specific bitfield only:"
reg := reg merge: reg not + cy mask: TEMP [cz]

YXX=111: NEG (RET,)reg
ADDMASK=0:
reg := reg not + 1 [cz]
ADDMASK=1: "Modify specific bitfield only:"
reg := reg merge: reg not + 1 mask: TEMP [cz]
```

%?Y??????1P111XXRRRRR
Unary operations from register RRRRR to ACCU. The 'P' bit adds a concurrent subroutine return as indicated above. Some of these instructions are modified by the ADDMASK bit in the FLAGS register:

```
YXX=000: SHL (RET,)ACCU,reg
ACCU := reg shl: 1 [cz]

YXX=001: SWAP (RET,)ACCU,reg
ACCU := reg rol: 4 [z]

YXX=010: SHR (RET,)ACCU,reg
ACCU := reg shr: 1 [cz]

YXX=011: SHRC (RET,)ACCU,reg
ACCU := cy, (reg from: 1 to: 7) [cz]

YXX=100: INC (RET,)ACCU,reg
ADDMASK=0:
ACCU := reg + 1 [cz]
ADDMASK=1: "Same but now on specific bitfield:"
ACCU := reg merge:
  (reg /\ TEMP) + (TEMP /\ (TEMP shl: 1) not)
    mask: TEMP [cz]

YXX=101: DEC (RET,)ACCU,reg
ADDMASK=0:
ACCU := reg + 255 [cz]
ADDMASK=1: "Same but now on specific bitfield:"
ACCU := reg merge:
  (reg /\ TEMP) + TEMP
    mask: TEMP [cz]

YXX=110: NEGC (RET,)ACCU,reg
ADDMASK=0:
ACCU := reg not + cy [cz]
ADDMASK=1: "Modify specific bitfield only:"
ACCU := reg merge: reg not + cy mask: TEMP [cz]

YXX=111: NEG (RET,)ACCU,reg
ADDMASK=0:
ACCU := reg not + 1 [cz]
ADDMASK=1: "Modify specific bitfield only:"
ACCU := reg merge: reg not + 1 mask: TEMP [cz]
```

4) Full duplex operation

The original implementation only allows simplex or half duplex operation. To allow full duplex operation, the control machine must either be split into separate receiving and transmitting machines (with interfaces between them), or some means must be provided to switch rapidly between sending and receiving functions. The latter solution is used in this implementation. Reception should be done on a kind of interrupt basis, which can be introduced with a special 'WAIT' operation and event recogniser (to switch from sending back to receiving):

%?YDDDDDDDD100XX?????
Wait instruction as a special form of the constant operations on registers (with register number a don't care), using DDDDD-DDD as constant:

YXX=101: WAIT #data / SWTCH
While in RX (receive) process:
resume TX process until (event /\ #data) ~= 0
event bits specified in file SDLCREGS.ASM
While in TX (transmit) process:
resume RX process immediately ('SWTCH')
DDDDDDDD is don't care here

Notes:

- The bitmask in DDDDDDDD allows the receiver to wait for several events at the same time (polling must then be used to determine which event actually occurred).

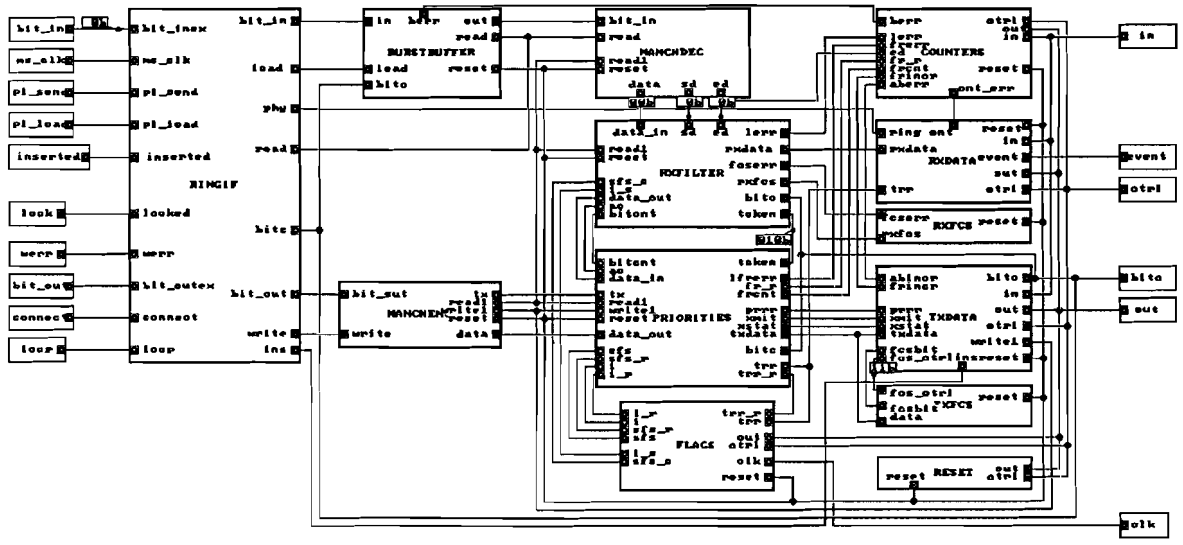
- The transmitting and receiving machine program counters should point to different locations in the program ROM following system reset (actually, to the instruction before the first instruction to execute). Which machine should be running after startup is determined by the asynchronous reset value in SWREG.

5) Other considerations

The bit patterns have been devised such that it is possible to reduce the microcode width from 20 bits to 19 bits. This reduces the JJJJJJJJJ (jump address) field to 9 bits, allowing only 512 words of microprogram. Extending the microcode width increases the possible amount of microprogram words without any further consequences.

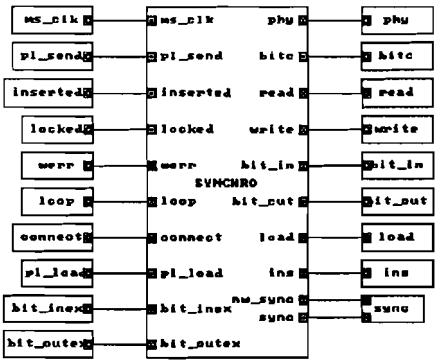
Appendix 2 : Document Files

This appendix contains the document files of the schematics used in the bit-processor. The information regarding to buses and control connectors has been deleted.



The bit-processor

RINGIF



IDaSS V0.08m document generated Wednesday December 15, 1993, 10:49:28.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========

'TopLevel\BITPROC\RINGIF' is a schematic.

Designer comments:
----------v----------
This schematic communicates with the attached Ring Interface Circuit. The schematic loads the received bits into the bit-processor, and puts the bits for transmission on the ring. The loading and transmission of bits is controlled by the external clock signals received from the ring interface circuit. Besides this the schematic processes the indication signals from the ring interface circuit, and generates the communication signals to the ring interface signals.

Inputs :

bitc	:	16 bits setting of the bit-processor.
bit_inex	:	The input bit stream from the ring.
bit_out	:	The bit stream to be send to the ring, from MANCHENC.
inserted	:	1 bits bus. Indicates if the station is inserted in the ring. 0 : not inserted 1 : inserted
locked	:	1 bits bus. Indicates if the PLL has locked. 0 : not locked 1 : locked
ms_clk	:	1 bits bus. The master oscillator.
pl_send	:	1 bits bus. The phase lock loop clock, for transmission timing.
pl_load	:	1 bits bus. the phase lock loop clock, for receive timing.
werr	:	1 bits bus. Indicates if there is a wiring error between the station and the ring. 0 : wiring error 1 : no wiring error.

Outputs :

bit_in	:	1 bits input bit stream, which is synchronized with the system clk.
connect	:	1 bits signal, indicating if the station must be inserted in the ring. 0 : do not insert 1 : insert
ins	:	Asserted when inserted.
loop	:	1 bits signal indicating if a loop test must be performed. 0 : normal operation 1 : perform loop test.
phy	:	2 bits bus indicating the status of the physical connection. bit0 = Asserted when wiring error. bit1 = Asserted when not locked.
load	:	Equal to pl_load.
read	:	Equal to pl_load when : Active not Equal to ms_clk when : Active
write	:	Equal to pl_load when : Active not /\ Force_TX not Equal to ms_clk when : Active \/ Force_TX

----------^----------
==========
'TopLevel\BITPROC\RINGIF\sync' is a register.

This register is 2 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register is used for the synchronization of the incomming bit stream, and putting the bits for transmission on the ring. Bit0 is the received bit, and bit1 is the bit send to the ring.

----------^----------

==========
'TopLevel\BITPROC\RINGIF\SYNCHRO' is an operator.

This operator has 1 function.
The default function is 'SYNC'.

Text for function 'SYNC' of 'TopLevel\BITPROC\RINGIF\SYNCHRO':
----------v----------
"This function load the received bits into the bit-processor, and sends the bits"
"for transmission to the ring. It also generates the read and write signals. "

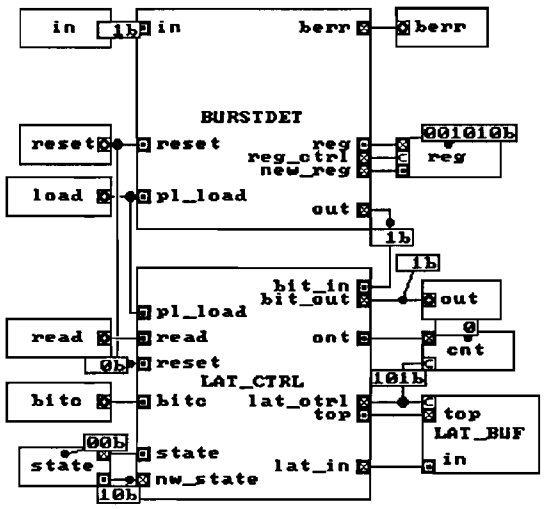
```
_active := (bitc at: 5).
_force_tx := (bitc at: 6).

_read := (_active
        if1: ms_clk
        if0: pl_load
       ).
_write := (_active \/ _force_tx
         if1: ms_clk
         if0: pl_load
        ).
nw_sync := (_write
          if1: bit_out
          if0: (sync at: 1)
         ),
         (_read
          if1: bit_inex
          if0: (sync at: 0)
         ).
bit_outex := (sync at: 1).
bit_in := (sync at: 0).

load := pl_load.
read := _read.
write := _write.

ins := (inserted not).
phy := (locked not),(werr not).

connect := (bitc at: 3).
loop := (bitc at: 4).
```

----------^----------

End of function descriptions.

==========
END OF DOCUMENT.

BURSTBUFFER



IDaSS V0.08m document generated Wednesday December 15, 1993, 10:55:43.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========

'TopLevel\BITPROC\BURSTBUFFER' is a schematic.

Designer comments:

----------v----------

In this schematic the Burst Error Detector and the Latency Buffer are implemented. The Latency Buffer is inserted when the station is the Active Monitor. Burst five errors are changed into burst four errors.

Inputs :
bitc : 16 bits setting of the bit-processor.
in : The received bit stream, from RINGIF
load : Clock signal from RINGIF.
read : Clock signal from RINGIF.
reset : The reset signal.

Outputs :
berr : Asserted during load, on the detection of a Burst Five error.
out : The input bit stream, without Burst Five errors, but with a possible delay because of the Latency buffer.

----------^----------

==========

'TopLevel\BITPROC\BURSTBUFFER\BURSTDET' is an operator.

This operator has 1 function.
The default function is 'BURSTERROR'.

Designer comments:
----------v----------
The operator BURSTDET detects Burst Five Errors and changes those in Burst Four Errors. The operator will keep on introducing transitions each bit-time until a transition is detected on 'in'.

The register REG is used for detection, introducing transitions, and generating the BERR signal.
reg : bit0..bit3 : The last four received bits.
bit4 : The previous bit put on 'out'.
bit5 : Is set when a Burst Five Error has occured, and is resetted when a transition is detected on 'in'.
----------^----------

Text for function 'BURSTERROR' of 'TopLevel\BITPROC\BURSTBUFFER\BURSTDET':
----------v----------
```
_zeros := (reg from: 0 to: 3) zerocnt.
_ones := (reg from: 0 to: 3) onecnt.
_idem := ((_zeros = 4) /\ (in not)) \/ ((_ones = 4) /\ in).      "burst 5 error"
_out := (_idem
        if0: in
        if1: ((reg at: 4) not)
       ).
new_reg := _idem,_out,(reg from: 0 to: 2),in.
reg_ctrl := reset, pl_load.
out := _out.
berr := _idem /\ ((reg at: 5) not) /\ pl_load.
```
----------^----------

End of function descriptions.

==========
'TopLevel\BITPROC\BURSTBUFFER\cnt' is a register.

This register is 7 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This counter is used to keep track of the number of bits in the buffer :LAT_BUF
----------^----------

Control specification:
----------v----------
%1XX reset. '
%010 dec.
%001 inc.
%000 hold.
%011 hold.
----------^----------

```
==========================================================
'TopLevel\BITPROC\BURSTBUFFER\LAT_BUF' is a FIFO.

This FIFO contains 112 words of 1 bit each and is controlled by an unnamed control input.
There is no contents file attached.

Designer comments:
----------v----------
This FIFO queue is used for the Latency buffer.
----------^----------

Control specification:
----------v----------
%1XX reset.
%011 read;write.
%010 read.
%001 write.
"%000 hold"
----------^----------

==========================================================
'TopLevel\BITPROC\BURSTBUFFER\LAT_CTRL' is an operator.

This operator has 1 function.
The default function is 'LAT_CTRL'.

Designer comments:
----------v----------
This operator controls the contents of the Latency Buffer. When the Latency
Buffer is not inserted, BIT_IN is directly connected with BIT_OUT.
When the buffer is inserted, the bits are loaded into the buffer with the
clock signal LOAD. Reading from the buffer is done with the clock signal
READ.
The READ signal is equal to the LOAD signal when the buffer is not in-
serted. In this way the data on BIT_OUT is always synchronous with READ.

The register CNT keeps track of the number of bits in the buffer.
The register STATE keeps track of state of the buffer, and of the previous
bit put on BIT_OUT.
bit0 : 0, if the buffer has not been completely inserted.
       1, after the buffer has been inserted, and has been filled to its
         initial setting.
bit1 : contains the previous outputted bit. This bit is used to introduce
       state transitions each bit time during initialisation, or underrun
       of the buffer.
----------^----------

Text for function 'LAT_CTRL' of 'TopLevel\BITPROC\BURSTBUFFER\LAT_CTRL':
----------v----------
"--------- functions for determination of the length of the buffer ----------"
_buffer := (bitc at: 5).                                  "insert buffer"
_freq := (bitc at: 0).                                    "16 or 4 Mhz"
_full := (_freq                                           "maximum length"
          if1: cnt = (56 width: 7)
          if0: cnt = (30 width: 7)
         ).
```

```
_init := (_freq                                    "initial length"
        if1: cnt = (39 width: 7)
        if0: cnt = (27 width: 7)
       ).
_empty := (cnt = 0).

"---------- calculate the output bit ----------------------------------------"
_bit_out := (_buffer
          if0: bit_in                                  "no buffer"
          if1: (((state at: 0) not) \/ _empty
               if1: ((state at: 1) not)          "buffer full or not init"
               if0: top                        "buffer not full and init"
          )    ).
"--------- calculate new state of the buffer ---------------------------------"
nw_state := (read
          if1: _bit_out
          if0: (state at: 1)
          ),
          (_buffer /\ (reset not)
          if1: (_init /\ read
               if1: (1 width: 1)
               if0: (state at: 0)
              )
          if0: (0 width: 1)
          ).
"-------- _rl, indicates if a bit must be loaded and/or read ----------------"
_rl := (((state at: 0) not) \/ (cnt = 0)
       if1: (0 width: 1),pl_load          "during init or underrun, only load"
       if0: (_full /\ (read not)
            if1: (0 width: 2)                   "during overrun only read"
            if0: read,pl_load            "during normal operation read/load"
     )    ).

lat_ctrl := (reset \/ (_buffer not)),_rl.
lat_in := bit_in.
bit_out := _bit_out.
----------------------^----------------------
```

End of function descriptions.

==========

'TopLevel\BITPROC\BURSTBUFFER\reg' is a register.

This register is 6 bits wide and is controlled by an unnamed control input.
The default function is 'load'.
This register is loaded with value 1 following system reset.

The value loaded for the 'reset' command is 1.

Designer comments:

```
----------------------v----------------------
```

This register is used for detecting Burst Five Errors, changing them into Burst Four Errors, introducing transitions until a transition on the input is detected, and generating the BERR signal.

```
----------------------^----------------------
```

Control specification:
----------v----------
%1X reset.
%00 hold.
%01 load.
----------^----------

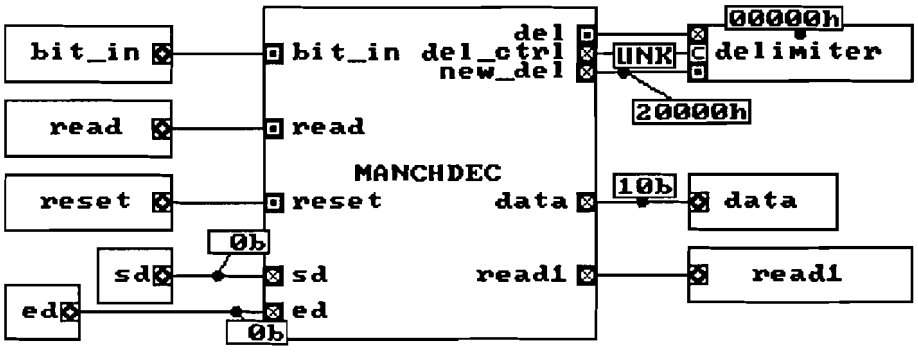
==========
'TopLevel\BITPROC\BURSTBUFFER\state' is a register.

This register is 2 bits wide.
The default function is 'load'.
This register is loaded with value 2 following system reset.

The value loaded for the 'reset' command is 0.

==========
END OF DOCUMENT.

MANCHDEC



IDaSS V0.08m document generated Wednesday December 15, 1993, 11:06:51.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========
'TopLevel\BITPROC\MANCHDEC' is a schematic.

Designer comments:
----------v----------
In this schematic the incoming bit stream is decoded with the Differential Machester Decoding algorithm, and presented on the 2 bits output data. Also detection of the SD and ED is performed. Finally the schematic generates the read1 clock signal.

Inputs :
bit_in : Input bit stream, from BURSTBUFFER
read : Clock signal from RINGIF.
reset : The reset signal.

Outputs :
data : The decoded input bit stream, only changes value after a transition
1/0 of read.
00 = binary-zero 10 = non-data symbol K
01 = binary-one 11 = non-data symbol J
Indication of non-data symbols only during trailing bit of symbol.
ed : Asserted during read in the trailing bit of the sixth symbol of an ED.
read1 : Asserted during read in the trailing bit of a symbol.
sd : Asserted during read in the trailing bit of the eigth symbol of an SD.
----------^----------

==========
'TopLevel\BITPROC\MANCHDEC\delimiter' is a register.

This register is 18 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register is used for the detection of the SD and ED, the generation of
the clock signal read1, and the decoding of the incoming it stream.
----------^----------

Control specification:
----------v----------
%1X reset.
%00 hold.
%01 load.
----------^----------

==========
'TopLevel\BITPROC\MANCHDEC\MANCHDEC' is an operator.

This operator has 1 function.
The default function is 'MANCHDEC'.

Text for function 'MANCHDEC' of 'TopLevel\BITPROC\MANCHDEC\MANCHDEC':
----------v----------
"Delimiter is a 18 bits register"
"Bit 0 to 15 are used for SD and ED detection"
"Bit 16 contains the previous decoded value of data at: 0"
"Bit 17 contains the previous value of output even"

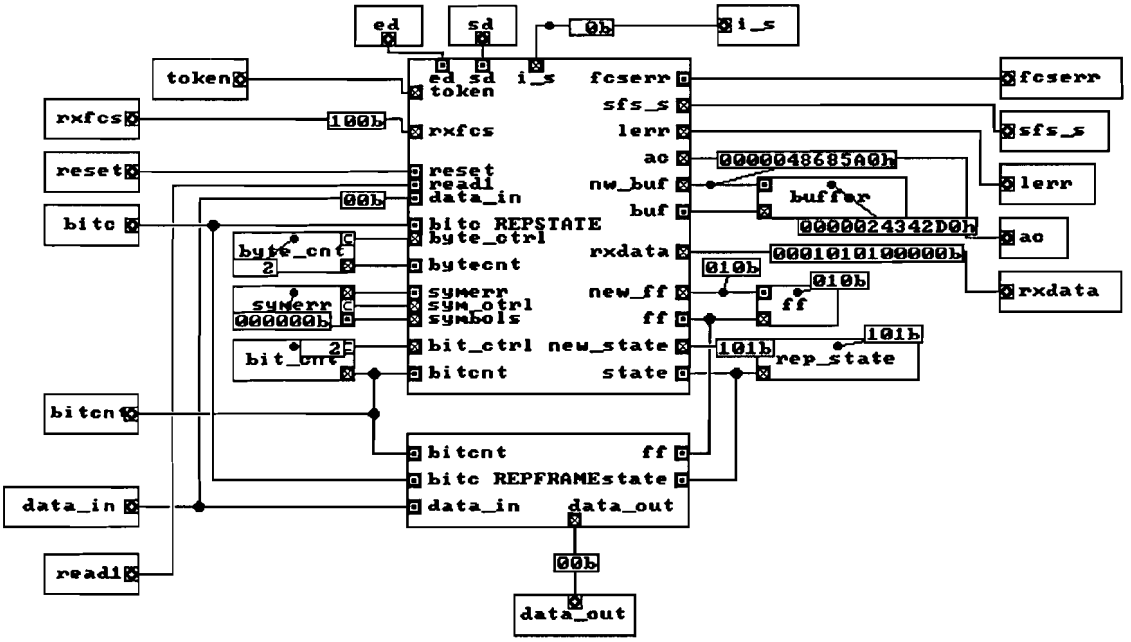
```
_bit0 := (del at: 0).
_databit := (bit_in = _bit0).
_sd :=    (((del from: 0 to: 15) = %0001101110010101) /\ (bit_in = 0))
     \/ (((del from: 0 to: 15) = %1110010001101010) /\ (bit_in = 1)).
_ed :=    (((del from: 0 to: 11) = %000111000111) /\ (bit_in = 0))
     \/ (((del from: 0 to: 11) = %111000111000) /\ (bit_in = 1)).
```

```
_neven := (del at: 17).
_even := ((_sd \/ _ed)
          if0: (_neven not)
          if1: (1 width: 1)
         ).

_data :=(_neven
         if0: _databit,(del at: 16)
         if1: (0 width: 1),_databit
        ).

read1 := (read /\ _even).
data := _data.
sd := _sd /\ read.
ed := _ed /\ read.
new_del := _even,(_data at: 0),(del from: 0 to: 14),bit_in.
del_ctrl := reset,read.
----------------------^----------------------
```

End of function descriptions.

==========
END OF DOCUMENT.

RXFILTER



IDaSS V0.08m document generated Wednesday December 15, 1993, 11:11:12.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========

'TopLevel\BITPROC\RXFILTER' is a schematic.

Designer comments:

----------v----------

In the RXFILTER schematic, incoming frames and tokens are repeated, and the E,A and C bits will be set if necessary. The received frames and tokens are passed, byte-wise, through RXDATA including error indication.

Inputs :

bitc : 16 bits setting of the bit-processor.
data_in : Decoded input bit stream from MANCHDEC.
ed : ED received signal from MANCHDEC.
fcserr : FCS error indication from RXFCS.
read1 : Clock signal from MANCHDEC.
reset : The reset signal.
sd : SD received signal from MANCHDEC.

Outputs :

ac : 2 bits signal that indicates when and what type of AC field is received.
bit0: Asserted during receipt of AC field, and no code violation.
bit1: Asserted when Token bit is received equal zero.
bitcnt : Indicates the bit number of the received bit of a byte.
data_out : Data_in with possible setting of the E, A and C bits.
i_s : Asserted during the I bit, when I equals 0.
lerr : Asserted during read1 and the E symbol, when the received E equals zero and the frame is a Frame With Error.
rxdata : 13 bits bus. Contains received byte and status information.
bit0..7 : data byte
bit8 : set when byte valid during one system clock.
bit9 : set when recieved frame is not Frame Good, or token did not end properly.
bit10 : set when received frame is Frame with Error, or token with code violation.
bit11..12: type of data received, set during receipt of data.
00 nop 01 FS field
10 token 11 Frame
note : When a frame ends properly bit11/12 change immediately from 11 to 01. Otherwise bit11/12 become 00 first (abort).
rxfcs : Three bits control bus for the controlling of RXFCS.
bit0 : Data bit.
bit1 : Asserted when data bit valid.
bit2 : Asserted when FCS calculation must be reset.
sfs_s : Asserted during last symbol of the SFS.
token : 3 bits bus. Indicates the receipt of a token.
bit0 : set when token received during one clock system.
bit1 : set when token did not end properly.
bit2 : set when token had code violation.
note : bit2 and 3 are only valid when bit0 is set.

----------^----------

==========
'TopLevel\BITPROC\RXFILTER\bit_cnt' is a register.

This register is 3 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register is used to indicate the bit-number of the received bit, in a byte.
----------^----------

Control specification:
----------v----------
%1X reset.
%01 inc.
%00 hold.
----------^----------

==========
'TopLevel\BITPROC\RXFILTER\buffer' is a register.

This register is 46 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This buffer is used to extract the FCS field and the ED.
----------^----------

==========
'TopLevel\BITPROC\RXFILTER\byte_cnt' is a register.

This register is 5 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register is used to count the number of received bytes of a frame. The
counter must not be incremented above 32 (overflow). This counter is used to
check the minimum length of a frame.
----------^----------

Control specification:
----------v----------
%01 inc.
%1X reset.
----------^----------

```
==========================================================
'TopLevel\BITPROC\RXFILTER\ff' is a register.

This register is 3 bits wide.
The default function is 'load'.
This register is loaded with value 3 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------------------v----------------------
This register contains the status of the received frame/token regarding to
errors. bit0: set when FF bits in FC field of frame are 10 or 11, or when
            a token is received.
       bit1: set when a frame not(Frame Good) is received, or a token that
            did not end properly.
       bit2: set when a Frame with Error is received, or a token with a code
            violation in the AC field.
----------------------^----------------------

==========================================================
'TopLevel\BITPROC\RXFILTER\rep_state' is a register.

This register is 3 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------------------v----------------------
This registers contains the state of the finite-state machine, implemented in
REPSTATE.
----------------------^----------------------

==========================================================
'TopLevel\BITPROC\RXFILTER\REPFRAME' is an operator.

This operator has 1 function.
The default function is 'SETFRAME'.

Designer comments:
----------------------v----------------------
This operator sets the E, A and C bits in the passing frames, when it's supposed
to do so as indicated by the bitc bus and contents of register FF.
----------------------^----------------------

Text for function 'SETFRAME' of 'TopLevel\BITPROC\RXFILTER\REPFRAME':
----------------------v----------------------
"This operator sets the E, A and C bits when required as indicated by FF and BITC"

_resume := (bitc at: 7).
_a := (bitc at: 14).
_c := (bitc at: 15).
_frerr := (ff at: 2).
_frgood := (ff at: 1) not.
```

```
data_out := (_resume
          if1: (((state from: 1 to: 2) = %11) /\ (bitcnt = 7) /\ _frerr
               if1: (1 width: 2)                                  "TO0B"
               if0: ((state = %101) /\ _frgood
                    if1: (((bitcnt = 0) \/ (bitcnt = 4)) /\ _a
                         if1: (1 width: 2)                        "TO0C"
                         if0: (((bitcnt = 1) \/ (bitcnt = 5)) /\ _c
                              if1: (1 width: 2)                   "TO0D"
                              if0: data_in
                             )
                        )
                    if0: data_in
                   )
              )
           if0: data_in
          ).
----------------------^----------------------
```

End of function descriptions.

==========

'TopLevel\BITPROC\RXFILTER\REPSTATE' is an operator.

This operator has 1 function.
The default function is 'REPSTAT'.

Designer comments:

```
----------------------v----------------------
```

This operator keeps track of the position in which the received frame or token is. Therefore the operator implements a sort of finite-state machine with the help of register STATE. The operator also performs the error checking, and the passing of received bytes towards RXDATA on the bus rxdata.

```
----------------------^----------------------
```

Text for function 'REPSTAT' of 'TopLevel\BITPROC\RXFILTER\REPSTATE':

```
----------------------v----------------------
```

"This function keeps track of the position of the frame/token that is repeated."

```
"---------- Local function for controlling the finite-state machine ----------"
_frleng := (bitc at: 2).
_abort := ed /\ (bytecnt = 0) /\ (bitcnt = 5).
_ac0 := ((state = %100) \/ (state = %001)) /\ (bytecnt < 1).
_sfs := (state = %001) /\ (bitcnt = 7) /\ (sd not) /\ read1.
_token := (state = %001) /\ (bitcnt = 3) /\ ((data_in at: 0) not) /\ read1.
_tkleng := (bitcnt = 5) /\ (bytecnt = 1) /\ read1.

"---------- The finite-state machine for frame/token receipt ----------"
_nw_state := ((state = %000)                                      "Idle"
          if1: (sd
               if1: (1 width: 3)
               if0: state
              )
```

```
if0: ((state = %001)                                    "AC field"
     if1:(_token                                      "rec token"
          if1: (4 width: 3)
          if0: (_abort                              "rec abort del."
               if1: (0 width: 3)
               if0: (ed                             "rec end del."
                    if1: (7 width: 3)
                    if0: (_sfs                        "rec frame"
                         if1: (3 width: 3)
                         if0: state
                        )
                   )
              )
         )
if0: ((state = %100)                                    "Token"
     if1: (ed
          if1: (6 width: 3)
          if0: (sd
               if1: (1 width: 3)
               if0: state
              )
         )
if0: ((state = %011)                                    "FC....ED"
     if1: (sd
          if1: (1 width: 3)
          if0: (ed
               if1: (7 width: 3)
               if0: state
              )
         )
if0: ((state = %111)                                    "I,E bit na frame"
     if1: ((bitcnt = %111) /\ read1
          if1: (5 width: 3)
          if0: state
         )
if0: ((state = %110)                                    "I,E bit na token"
     if1: ((bitcnt = %111) /\ read1
          if1: (0 width: 3)
          if0: state
         )
if0: ((state = %101)                                    "FS"
     if1: ((bitcnt = %111) /\ read1
          if1: (0 width: 3)
          if0: state
         )
if0: (0 width: 3)
)))))).

new_state := (reset
           if1: (0 width: 3)
           if0: _nw_state
          ).
"---------- Counters keeping track on the number of received data ----------"
bit_ctrl := (reset \/ sd),read1.
byte_ctrl := (reset \/ sd),(read1 /\ (bitcnt = %111) /\ (bytecnt < %11111)).
```

```
"---------- Functions for error detection ----------------------------------"
_biterr := ((_ac0 \/ (state = %101)) /\ (data_in at: 1)) \/ ((state = %011) /\
          (symerr at: 5)).
_tkerr := ((state = %100) /\ ((_tkleng /\ (ed not)) \/ ((_tkleng not) /\ ed))).
_frlerr := ((ed /\ (state = %011))
          if1: (_frleng
               if1: (bytecnt < 17)
               if0: (bytecnt < 9)
              )
          if0: (0 width: 1)
         ).
_octerr := ed /\ (state = %011) /\ (bitcnt ~= %101).
_ff := (ff at: 0).
_fcserr := fcserr /\ (state = %111).
_frerr := _octerr \/ _biterr \/ (_fcserr /\ _ff) \/ (_frlerr /\ _ff) \/ (ff at: 2).
_frng := (_octerr \/ _biterr \/ _fcserr \/ _frlerr \/ (_ff not) \/ _tkerr)
         \/ (ff at: 1).

"---------- Control signal for PRIORITY, set during AC field and no _biterr --"
ac := (_ac0 /\ (state = %100)),(_ac0 /\ (_frerr not)).

"---------- Buffer to eliminate ED for code violation (J/K) detection --------"
symbols := (_biterr \/ (symerr at: 4)),(symerr from: 0 to: 3),(data_in at: 1).
sym_ctrl := (reset \/ sd \/ ed),(read1 /\ (state = %011)).

"---------- Control signals to COUNTERS, FLAGS and TXDATA --------------------"
lerr := ((state from:1 to: 2) = %11) /\ (bitcnt = 7) /\ read1 /\ (data_in = %00) /\ _frerr.
i_s := _abort \/ ((state = %111) /\ (bitcnt = 6) /\ ((data_in at: 0) not)).
sfs_s := _sfs.
token := _frerr,_frng,((state = %100) /\ (_nw_state ~= %100)).

"---------- The FF register keeps track of errors, and frame type ------------"
"ff0 : set if frame has FF equal to 00 or 01                                  "
"ff1 : set when Frame with Error or Token with code violation                 "
"ff2 : set when frame was not Frame Good, or Token did not end properly       "
new_ff := (sd \/ (state = %000)
          if1: (1 width: 3)
          if0: (read1
               if1: ((state = %011) /\ (bitcnt = 0) /\ (bytecnt = 1)
                    if1: (_frerr,_frng,((data_in at: 0) not))
                    if0: (_frerr,_frng,_ff)
                   )
               if0: ff
              )
         ).

"---------------------- functions for creating rxdata -------------------------"
_valid := ( ((bitcnt = %110) /\ (bytecnt > 4) /\ (state = %011)) \/
          ((bitcnt = %110) /\ (state = %111)) \/
          ((bitcnt = %111) /\ (state = %101)) \/
          (_tkleng /\ (state = %100)) ) /\ read1.

_type := ((((state from: 0 to: 1) = %11) /\ (bytecnt > 4)) \/ (state = %100)),
        ((state at: 0) /\ (bytecnt > 4)),
        _frerr, _frng.
```

```
rxdata := (((state from: 0 to: 1) = %11)
        if1: _type,_valid,(buf from: 38 to: 45)
        if0: ((state = %101)
             if1: _type,_valid,(buf from: 0 to: 6),(data_in at: 0)
             if0: ((state = %100)
                  if1: _type,_valid,(buf from: 5 to: 12)
                  if0: _type,_valid,(0 width: 8)
       )   )   ).
"---------- Buffer for extracting the FCS field and ED ----------"
nw_buf := (read1
        if1: (buf from: 0 to: 44),(data_in at: 0)
        if0: buf
       ).
"---------- Control bus to RXFCS ----------------------------------------"
rxfcs := ((state from: 0 to: 1) ~= %11),
    ((state = %011) /\ ((bytecnt > 1) \/ ((bitcnt > 6) /\ (bytecnt = 1))) /\ read1),(buf at: 5).
---------------------^---------------------
```

End of function descriptions.

==========

'TopLevel\BITPROC\RXFILTER\symerr' is a register.

This register is 6 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:

```
---------------------v---------------------
```

This register is used to extract the ED in the code violation checking of
frames and tokens.

```
---------------------^---------------------
```

Control specification:

```
---------------------v---------------------
```

%1X reset.
%01 load.
%00 hold.

```
---------------------^---------------------
```

==========

END OF DOCUMENT.

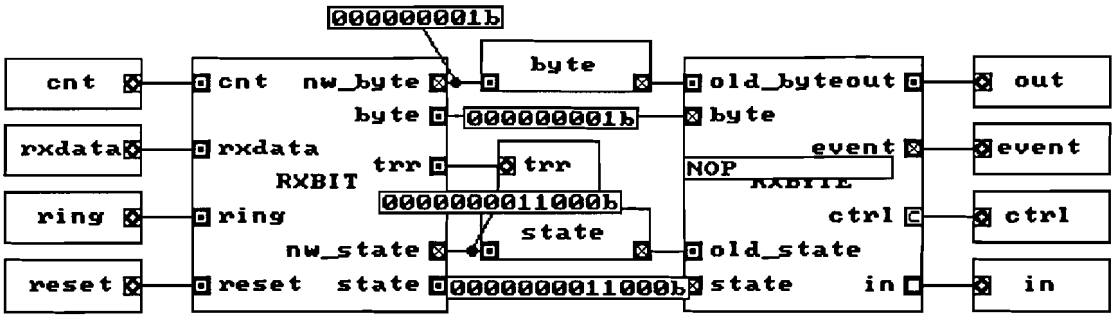
RXDATA

IDaSS V0.08m document generated Wednesday December 15, 1993, 11:14:01.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========

'TopLevel\BITPROC\RXDATA' is a schematic.



Designer comments:
----------v----------
RXDATA processes the information of the receive process towards the byte-processor. This operator implements the functioning of the Event channel, the Receive buffer and Receive Status register.

Inputs :
cnt : Indicates if one of the error counters has overflown, from COUNTERS.
ctrl : 7 bits address and control bus from the byte-processor.
out : 8 bits data output bus from the byte-processor.
ring : Indication on the physical connection from RINGIF.
reset : The reset signal.
trr : Indicates if the TRR is expired.
rxdata : 13 bits data and status bus from RXFILTER.

Outputs :
Event : Four bits event bus.
bit0 : byte from frame in Receive buffer.
bit1 : FS field in Recieve buffer.
bit2 : AC field of token in Receive buffer.
bit3 : error in received data, or physical connection.
in : 8 bits tri-state bus towards the byte-processor.
----------^----------

==========
'TopLevel\BITPROC\RXDATA\byte' is a register.

This register is 9 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
The Receive buffer. Bit0 indicates if a byte is present. Bit1..8 is the byte.
----------^----------

==========
'TopLevel\BITPROC\RXDATA\RXBIT' is an operator.

This operator has 1 function.
The default function is 'RXBIT'.

Designer comments:
----------v----------
This operator stores the information on the Receive process in the registers
BYTE and STATE.
----------^----------

Text for function 'RXBIT' of 'TopLevel\BITPROC\RXDATA\RXBIT':
----------v----------

```
"This function processes the information of the receive process. The received "
"bytes are placed in the register 'byte', the status information in the regis-"
"ter 'state'. The first three bits of 'state' are the 'event' flags. Bit4..11 "
"are the status bits. Bit12 is used to identify wether the received frame is  "
"ended properly with a FS field (not aborted).                                "
"Bit12; set when frame received (_code = %11), reset when _code ~= %11.       "
"End error occurs when; Bit12 = 1 /\ _code = %00                              "

"---------- Local functions extracting info from rxdata ----------"
_valid := (rxdata at: 8).
_code := (rxdata from: 11 to: 12).
"---------- '_nonw' is set when; (receive new frame \/ receive new token)not -"
_nonw := ((_code = %10) \/ ((_code = %11) /\ ((state at: 12) not))) not.

_bit12 := (_code = %11) \/ ((state at: 12) /\ (_code ~= %01)).
_bit0 := (_valid /\ (_code = %11)) \/ (state at: 0).              "rx byte"
_bit1 := (_valid /\ (_code = %01)) \/ (state at: 1).               "rx FS"
_bit2 := (_valid /\ (_code = %10)) \/ (state at: 2).             "rx token"
_bit4 := ((rxdata at: 9) /\ _valid) \/ ((state at: 4) /\ _nonw).  "data error"
_bit5 := ((rxdata at: 10) /\ _valid) \/ ((state at: 5) /\ _nonw).  "not good"
_bit6 := ((_code = %00) /\ _bit12) \/ ((state at: 6) /\ _nonw).   "end error"
_bit7 := (_valid /\ (byte at: 0)) \/ ((state at: 7) /\ _nonw).
                                                      "RXbuf overfl"
_bit8 := cnt.                                           "cnt overfl"
_bit9 := trr.                                            "trr exp"
_bit10 := (ring at: 0).                                      "werr"
_bit11 := (ring at: 1).                                   "pll error"

_bit3 := ((_bit4 \/ _bit5 \/ _bit7) /\ _valid) \/ _bit6 \/          "error"
       (_bit8 /\ ((state at: 8) not)) \/ (_bit9 /\ ((state at: 9) not)) \/
       (_bit10 /\ ((state at: 10) not)) \/ (_bit11 /\ ((state at:11) not)) \/
       ((state at: 3) /\ _nonw).

nw_state := (_bit12 /\ (_code ~= %00)),
         _bit11,_bit10,_bit9,_bit8,_bit7,_bit6,_bit5,_bit4,
         _bit3,_bit2,_bit1,_bit0.

"---------- Control function for the RX buffer ----------"
nw_byte := (reset
         if1: (0 width: 9)
         if0: (_valid
              if1: (rxdata from: 0 to: 7),(1 width: 1)
              if0: byte
        )    ).
```

----------^----------

End of function descriptions.

```
==========================================================
'TopLevel\BITPROC\RXDATA\RXBYTE' is an operator.

This operator has 3 functions and is controlled by control input 'ctrl'.
The default function is 'NOP'.

Designer comments:
----------v----------
This operator passes the information on the Receive process from the registers
BYTE and STATE, on command from the byte-processor.
----------^----------

Control specification:
----------v----------
%1001100 RD_RXBUFFER.
%1000001 RD_RXSTATUS.
%1001100 enable: in.
%1X00001 enable: in.
----------^----------

Text for function 'NOP' of 'TopLevel\BITPROC\RXDATA\RXBYTE':
----------v----------
"In this function no operations are performed except signalling the events.    "

event := (old_state from: 0 to: 3).
state := old_state.
byte := old_byte.
----------^----------

Text for function 'RD_RXBUFFER' of 'TopLevel\BITPROC\RXDATA\RXBYTE':
----------v----------
"This function reads and pops the byte from the receive buffer.              "
"It's the implementation of the Receive buffer register 12.                  "

byte := (0 width: 9).
_state := (old_state from: 3 to: 12),(0 width: 3).
state := _state.
event := (_state from: 0 to: 3).
in := (old_byte from: 1 to: 8).
----------^----------

Text for function 'RD_RXSTATUS' of 'TopLevel\BITPROC\RXDATA\RXBYTE':
----------v----------
"This function passes the status information from the receive process.        "

byte := old_byte.
event := (old_state from: 0 to: 3).
in := (old_state from: 4 to: 11).
state := (old_state from: 4 to: 12),(0 width: 1),(old_state from: 0 to: 2).
----------^----------

End of function descriptions.
```

==========
'TopLevel\BITPROC\RXDATA\state' is a register.

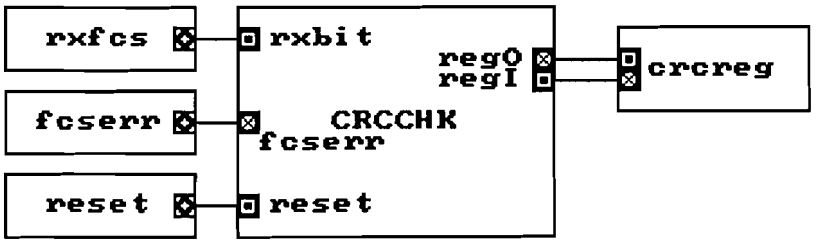
This register is 13 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register contains the 4 bits of the event channel, the 8 bits of the
Receive status register, and one bit wich is used for the detection of the
improper ending of a frame.
----------^----------

==========
END OF DOCUMENT.

RXFCS



IDaSS V0.08m document generated Wednesday December 15, 1993, 11:16:37.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========
'TopLevel\BITPROC\RXFCS' is a schematic.

Designer comments:
----------v----------
This schematic is used to perform the FCS checking on the received frames. The
operator in the schematic is controlled by RXFILTER.

Inputs :
reset : The reset signal.
rxfcs : Three bits control bus from RXFILTER, that controls the FCS checker.

Outputs :
fcserr : Indicates when the FCS checker detects an FCS error. This signal is
continuously generated.
----------^----------

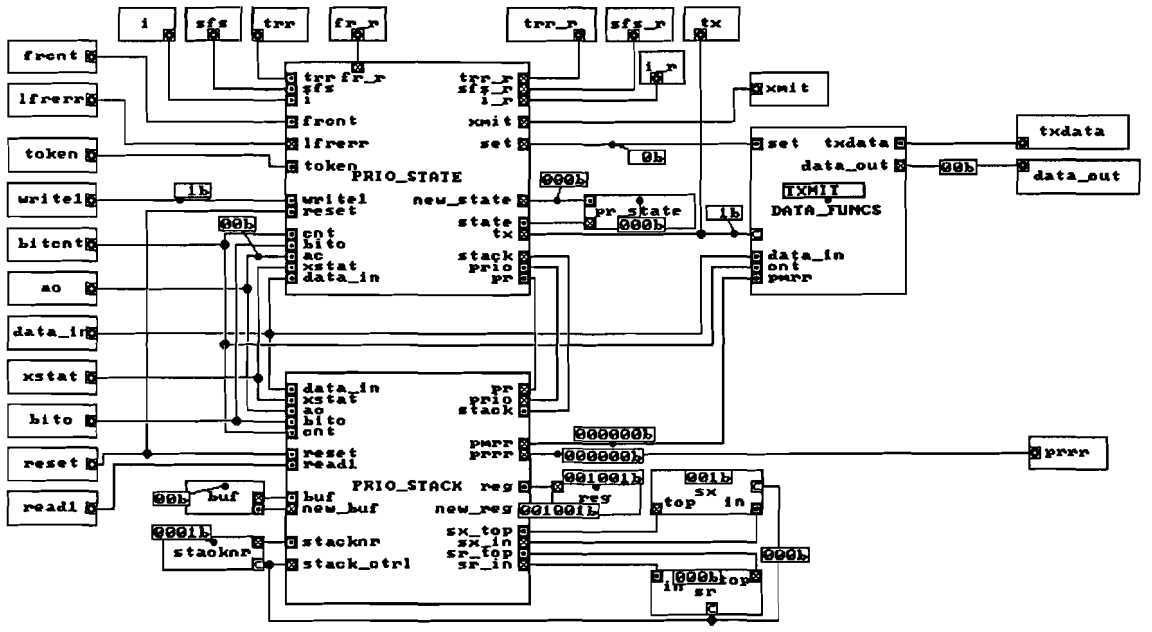
```
==========================================================
'TopLevel\BITPROC\RXFCS\CRCCHK' is an operator.

This operator has 1 function.
The default function is 'CRC32'.

Text for function 'CRC32' of 'TopLevel\BITPROC\RXFCS\CRCCHK':
----------------------v----------------------
"This function performs the FCS checking on the received frames.          "
_reset := (rxbit at: 2).
_load := ((rxbit from: 1 to: 2) = %01).

_Bit := (rxbit at: 0) >< (regI at:0).

regO :=(_load
        if0: RegI
        if1: (_reset
              if0: (_load
                    if1: _Bit,
                         ((regI at: 31) >< _Bit),"32-1 = 31"
                         ((regI at: 30) >< _Bit),"32-2 = 30"
                         (regI at: 29),
                         ((regI at: 28) >< _Bit),"32-4 = 28"
                         ((regI at: 27) >< _Bit),"32-5 = 27"
                         (regI at: 26),
                         ((regI at: 25) >< _Bit),"32-7 = 25"
                         ((regI at: 24) >< _Bit),"32-8 = 24"
                         (regI at: 23),
                         ((regI at: 22) >< _Bit),"32-10 = 22"
                         ((regI at: 21) >< _Bit),"32-11 = 21"
                         ((regI at: 20) >< _Bit),"32-12 = 20"
                         (regI from: 17 to: 19),
                         ((regI at: 16) >< _Bit),"32-16 = 16"
                         (regI from: 11 to: 15),
                         ((regI at: 10) >< _Bit),"32-22 = 10"
                         ((regI at: 9) >< _Bit),"32-23 = 9"
                         (regI from: 7 to: 8),
                         ((regI at: 6) >< _Bit),"32-26 = 6"
                         (regI from: 1 to: 5)
                    if0: regI
                   )
              if1: $FFFFFFFF
             )
         ).
fcserr := (regI = $C7B1AFF3) not.
----------------------^----------------------

End of function descriptions.

==========================================================
'TopLevel\BITPROC\RXFCS\crcreg' is a register.

This register is 32 bits wide.
The default function is 'hold'.
This register is loaded with unknown values after a system reset.
```

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register is used for the FCS checking.
----------^----------

==========
END OF DOCUMENT.

PRIORITIES



IDaSS V0.08m document generated Wednesday December 15, 1993, 11:27:45.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========
'TopLevel\BITPROC\PRIORITIES' is a schematic.

Designer comments:
----------v----------
In this schematic almost the complete functioning of the Operational FSM is implemented. The schematic also stores the received priority and reservation of the last received frame or token, and the stack Sx and Sr.

Inputs :
ac : Indication when and what type AC field is received from RXFILTER.
bitc : Setting of the bit-processor.
bitcnt : Bit number of the received bit from RXFILTER.
data_in : Received data stream from RXFILTER.

```
frcnt   : Indicates if the Frame Counter equals zero, from COUNTERS.
i       : Indicates if the I flag is set, from FLAGS.
read1   : Clock signal for reception timing from MANCHDEC.
reset   : The reset signal.
sfs     : Indicates if the SFS flag is set, from FLAGS.
token   : Signalling of the receipt of a token from RXFILTER.
trr     : Indicates if the TRR is expired, from FLAGS.
txdata  : Data for transmission from TXDATA.
write1  : Clock signal for transmission timing from MANCHENC.
xstat   : Indicates the status of the transmission by TXDATA.

Outputs :
data_out    : The data for transmission.
i_r         : Signals when the I flag must be reset.
fr_r        : Signals when the Frame Counter must be reset.
lfrerr      : Signals when the Lost Frame Error counter must be incremented.
prrr        : Indicates the priority and reservation of the AC field to be
              transmitted.  bit0..2 : reservation.
                            bit3..5 : priority.
sfs_r       : Signals when the SFS flag must be reset.
trr_r       : Signals when the TRR must be restarted.
tx          : Signals wether data_in or txdata is transmitted.
                  %0  data_in     %1  txdata
xmit        : Signals what kind of transmission must be started.
                  %00 Nop         %10 Token
                  %01 Frame       %11 Abort Delimiter
----------------------^----------------------

==========================================================
'TopLevel\BITPROC\PRIORITIES\buf' is a register.

This register is 2 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

==========================================================
'TopLevel\BITPROC\PRIORITIES\DATA_FUNCS' is an operator.

This operator has 2 functions and is controlled by an unnamed control input.
The default function is 'TXMIT'.

This operator has the following connectors:

Control specification:
----------------------v----------------------
%0 REPEAT.
%1 TXMIT.
----------------------^----------------------
```

Text for function 'REPEAT' of 'TopLevel\BITPROC\PRIORITIES\DATA_FUNCS':

```
---------------------v---------------------
"This function is used for setting the reservation bits in the Ac field."
"This is only done when Pm > Rr."

_data_bit := (set
          if1: ((cnt = 3) \/ (cnt = 4)
               if1: (1 width: 1)
               if0: ((cnt = 5)
                    if1: ((data_in at: 0) \/ (pmrr at: 5))
                    if0: ((cnt = 6)
                         if1: (((pmrr from: 0 to: 1) =< (pmrr from: 4 to: 5))
                              if1: (pmrr at: 4)
                              if0: (data_in at: 0)
                             )
                         if0: (((pmrr from: 0 to: 2) =< (pmrr from: 3 to: 5))
                              if1: (pmrr at: 3)
                              if0: (data_in at: 0)
                             )
                        )
                   )
              )
          if0: (data_in at: 0)
         ).

data_out := (data_in at: 1),_data_bit.
---------------------^---------------------
```

Text for function 'TXMIT' of 'TopLevel\BITPROC\PRIORITIES\DATA_FUNCS':

```
---------------------v---------------------
"This function is used for transmitting data."
"This data can be a frame, token, zero's or fill."

data_out := txdata.
---------------------^---------------------
```

End of function descriptions.

==========

'TopLevel\BITPROC\PRIORITIES\pr_state' is a register.

This register is 3 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

==========

'TopLevel\BITPROC\PRIORITIES\PRIO_STACK' is an operator.

This operator has 1 function.
The default function is 'STACK_CTRL'.

Text for function 'STACK_CTRL' of 'TopLevel\BITPROC\PRIORITIES\PRIO_STACK':

```
----------v----------
"These fuctions control the stacks sx and sr and the register Pr and Rr"

"---------- Local functions for demultiplexing input ----------"
_halt := ((bitc at: 7) not).
_pduq := (bitc at: 11).
_pm := (bitc from: 8 to: 10).
_pr := (reg from: 3 to: 5).
_rr := (reg from: 0 to: 2).
_txtk := ((xstat from: 0 to: 1) = %10).

_px := ((_pm > _rr) /\ _pduq
       if1: _pm
       if0: _rr
      ).
_stacking := (stacknr ~= %00).

"---------- Detect the situation of priority and reservation ----------"
_prio := (_halt
        if0: (_stacking
             if1: ((_pr > sx_top)
                  if1: ((_pr >= _px)
                       if1: (0 width: 2)
                       if0: (2 width: 2)
                      )
                  if0: (_stacking
                       if1: ((sr_top >= _px)
                            if1: (1 width: 2)
                            if0: (3 width: 2)
                           )
                       if0: (3 width: 2)
                      )
                 )
             if0: ((_pr >= _px)
                  if1: (0 width: 2)
                  if0: (2 width: 2)
                 )   )
        if1: (2 width: 2)
        ).

prio := _prio.

"---------- Stack control and stack input functions ----------"
_sx_in := (_halt
         if1: _rr
         if0: _px
        ).
sx_in := _sx_in.

_sr_in := (_halt
         if1: (0 width: 3)
         if0: _pr
        ).
sr_in := _sr_in.
```

```
_resetlifo := (_stacking /\ (_txtk not)
              if1: ((sx_top > _pr)
                    if1: (1 width: 1)
                    if0: (0 width: 1)
                   )
              if0: (0 width: 1)
             ) \/ stack.

_stack_ctrl := ((xstat = %110)
              if1: _prio
              if0: (0 width: 2)
             ).

stack_ctrl := (reset \/ _resetlifo),(_stack_ctrl).

"---------- Extra buffer function for loading piority/reservation ----------"
"---------- value in register in one clock cycle                  ----------"
new_buf := (read1
           if1: (buf at: 0),(data_in at: 0)
           if0: buf
          ).
_3bit := buf,(data_in at: 0).

"---------- Output fuction for setting reservation bits by DATA_FUNCS --------"
pmrr := _pm,_3bit.

"---------- Function for outputting priority and reservation ----------------"
"---------- for frame or token in transmission by TXDATA     ----------------"
prrr := (_halt
        if1: (_txtk
              if1: _rr
              if0: (0 width: 3)
             )
        if0: (_txtk
              if1: ((_prio at: 1)
                    if1: _px
                    if0: ((_prio at: 0)
                          if1: sr_top
                          if0: _pr
                         )
                   )
              if0: _pr
       )    ),
       (_txtk /\ ((_prio at: 1) not)
        if1: _px
        if0: (0 width: 3)
       ).

"---------- Current priority of the ring for PRIO_STATE ----------------------"
pr := _pr.
```

```
"---------- Function for loading last received priority and ----------"
"---------- reservation in register 'reg'                  ----------"
_ac := (ac at: 0).
new_reg := (_ac /\ (cnt = 2) /\ (_txtk not)
          if1: _3bit
          if0: ((_stack_ctrl at: 1)
               if1: _sx_in
               if0: _pr
          )    ),
          (_ac /\ (cnt = 7) /\ (_txtk not)
           if1: _3bit
           if0: ((_stack_ctrl = %010)
                if1: _sx_in
                if0: _rr
          )    ).
---------------------^----------------------
```

End of function descriptions.

==========

'TopLevel\BITPROC\PRIORITIES\PRIO_STATE' is an operator.

This operator has 1 function.
The default function is 'STATE_CTRL'.

This operator has the following connectors:

Text for function 'STATE_CTRL' of 'TopLevel\BITPROC\PRIORITIES\PRIO_STATE':

```
---------------------v----------------------
"These functions implement the most of the Operational Finite State Machine"

"---------- Local functions for demultiplexing the input ----------"
_active := (bitc at: 5).
_ma := (bitc at: 12).
_not_ma := (bitc at: 13).
_pm := (bitc from: 8 to: 10).
_pduq := (bitc at: 11).
_etr := (bitc at: 1).
_force := (bitc at: 6).
_halt := ((bitc at: 7) not).
_abort := (xstat = %011).
_pduqe := (xstat = %000) /\ write1.

"---------- Local functions for determination of state transitions ----------"
_tkbit := (cnt = 3) /\ (ac at: 0) /\ ((data_in at: 0) not).
_TO01 := _pduq /\ _tkbit /\ (pr <= _pm).
_TO03 := (prio at: 0) /\ ((_pduq not) \/ (_pduq /\ (_pm < pr))) /\ _tkbit.
_TO31 := (state = %101) /\ (i \/ (_ma /\ (frcnt = 0))) /\ write1.
_TO32 := (state = %101) /\ (trr \/ _not_ma \/ ((_ma not) /\ (frcnt = 0))) /\ write1.
_TO0F := (_active /\ (cnt = 4) /\ (ac at: 0) /\
        (((pr ~= %000) /\ (ac at: 1)) \/ ((ac at: 1) not))).
_TO0E := (state = %001) /\ _TO0F /\ write1 /\ (data_in at: 0).
```

```
"---------- Function implementing the finite state machine ----------"
_nw_state := ((state = %000)                                    "HALTED"
            if1: ((_halt not)                                "TO02 RESUME"
                 if1: (1 width: 3)
                 if0: state
                )
if0: ((state = %001)                                          "TO0  REPEAT"
     if1: (_TO01 /\ write1
          if1: (7 width: 3)                                  "->TO1 (TO01)"
          if0: (_TO03 /\ write1
               if1: (2 width: 3)                             "->TO4 (TO03)"
               if0: (_TO0E
                    if1: (6 width: 3)
                    if0: state
          ) ) )
if0: ((state = %010)                                          "TO4 TX ZEROS"
     if1: (_abort
          if1: (1 width: 3)                                  "->T0 (TO42)"
          if0: ((token = %001)
               if1: (4 width: 3)                             "->T4 (TO41)"
               if0: state
          ) )
if0: ((state = %111)                        "T01 setzero, for M and R bits"
     if1: ((cnt = 7) /\ write1
          if0: state
          if1: (6 width: 3)                                  "->T1 (TO01)"
         )
if0: ((state = %110)                                          "TO1 TX FRAME"
     if1: (_abort
          if1: (1 width: 3)                                 "->TO0 (TO1A1)"
          if0: (_pduqe
               if1: (3 width: 3)
               if0: state                                  "->TO2 (TO1A2)"
          ) )
if0: ((state = %011)                                          "TO2 TX FILL"
     if1: ((_etr \/ _ma) /\ write1
          if1: (5 width: 3)                                  "->TO3 (TO21)"
          if0: (trr /\ write1
               if1: (1 width: 3)                             "->TO0 (TO22)"
               if0: state
          ) )
if0: ((state = %101)                                          "TO3 TX TOKEN"
     if1: ((_TO31 \/ _TO32) /\ _pduqe
          if1: (1 width: 3)                          "->TO0 (TO31 or TO32"
          if0: state
         )
if0: ((state = %100)                                          "T5 TX TOKEN"
     if1: ((sfs \/ trr) /\ write1 /\ _pduqe
          if1: (1 width: 3)                                  "->TO0 (TO51)"
          if0: state
         )

if0: (0 width: 3)
)))))))).
```

```
new_state := (reset \/ _halt
              if1: (0 width: 3)
              if0: _nw_state
             ).

"---------- Functions for indicating DATA_FUNCS that bits have to be set -----"
"---------- The bits to be set are the frame and reservation bits in AC -----"
_TO01TO03 := (_TO01 \/ _TO03) /\ (state = %001).
_TO12 := (state = %110) /\ (_nw_state = %001).
_TO0A := _pduq /\ (cnt > 4) /\ (ac at: 0).
set := _TO01TO03 \/_TO0A \/ _TO0F.

"---------- Functions for resetting some flags ------------------------------"
fr_r := _TO01.
i_r := _TO12.
lfrerr := _TO32 \/ ((state = %011) /\ (_nw_state = %001)).      "_TO32 \/ TO22"
sfs_r := _TO01TO03.
trr_r := _TO01TO03 \/ _TO12.

"---------- Function for signalling TXDATA to start transmission ------------"
"---------- of a token or frame                                  ------------"
xmit := ((state = %111) /\ (_nw_state = %110)
        if1: (1 width: 2)
        if0: (((state = %011) /\ (_nw_state = %101)) \/
              ((state = %010) /\ (_nw_state = %100))
              if1: (2 width: 2)
              if0: ((state = %010) /\ (token at: 1) \/ _TO0E
                    if1: (3 width: 2)
                    if0: (0 width: 2)
       )    )    ).

"---------- Function to signal PRIO_STACK that the stacks must be reset ------"
stack := ((state ~= %000) /\ _halt).

"---------- Function for putting DATA_FUNCS in REPEAT or TXMIT ---------------"
tx := ((_halt /\ (_force not)) \/ (state = %001)
      if1: (0 width: 1)
      if0: (1 width: 1)
     ).
----------------------^----------------------
```

End of function descriptions.

==========

'TopLevel\BITPROC\PRIORITIES\reg' is a register.

This register is 6 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

==========
'TopLevel\BITPROC\PRIORITIES\sr' is a LIFO.

This LIFO contains 10 words of 3 bits each and is controlled by an unnamed control input.
There is no contents file attached.

Control specification:
```
----------v----------
%1XX reset.
"%000 hold."
%001 pop.
%010 push.
"%011 hold"
----------^----------
```

==========
'TopLevel\BITPROC\PRIORITIES\stacknr' is a register.

This register is 4 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Control specification:
```
----------v----------
%1XX reset.
%010 inc.
%001 dec.
"%000 hold.
 %011 hold."
----------^----------
```

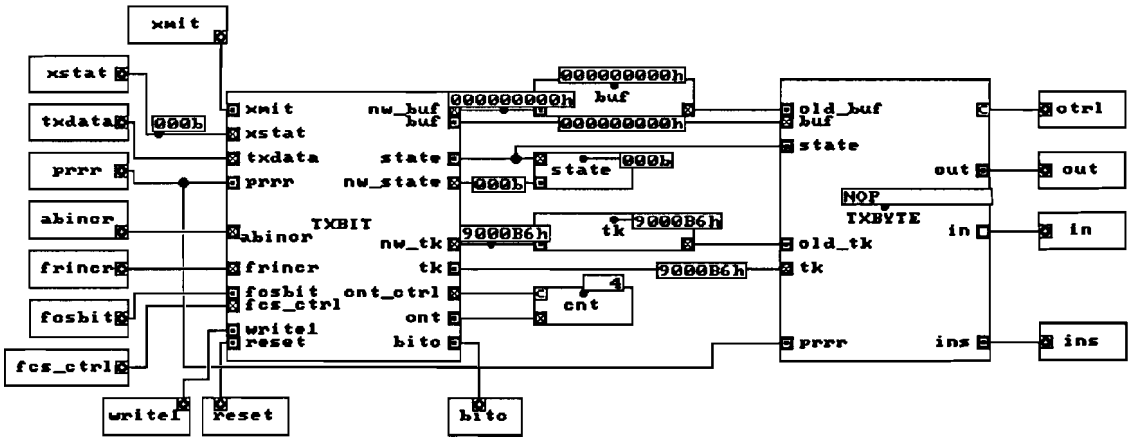
==========
'TopLevel\BITPROC\PRIORITIES\sx' is a LIFO.

This LIFO contains 10 words of 3 bits each and is controlled by an unnamed control input.
There is no contents file attached.

Control specification:
```
----------v----------
%1XX reset.
"%000 hold."
%001 pop.
%010 push.
%011 replace.
----------^----------
```

==========
END OF DOCUMENT.

TXDATA



IDaSS V0.08m document generated Wednesday December 15, 1993, 11:35:01.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========

'TopLevel\BITPROC\TXDATA' is a schematic.

Designer comments:

----------v----------

In this schematic the functions for the transmission of frames and tokens are implemeted. This schematic implements the Transmit Status register and the Transmit buffer, and a part of the Command register.

Inputs :

bitc : 16 bits setting of the bit-processor.
ctrl : 7 bits address and control from the byte-processor.
fcsbit : 1 bits bus contains the bit for the FCS field which has to be transmitted.
From : TXFCS
out : 8 bits data output bus from the byte-processor.
prrr : Contains the value's for the priority and reservation for the transmission of the AC field, from PRIORITIES.
reset : The reset signal.
write1 : Clock signal from MANCHENC.
xmit : Signals when a token or frame must be transmitted, from PRIORITIES.

Outputs :

abincr : Asserted during write1 in the last symbol of an Abort Delimiter.
fcs_ctrl : 2 bits bus which controls the fcs generator.
00 : hold 10 : transmit new bit
01 : load new bit 11 : reset fcs generator
frincr : Signals the transmission of a new frame.
in : 8 bits tri-state data input bus towards the byte-processor.
txdata : 2 bits bus containing the symbol for transmission.

xstat : 3 bits bus which indicates and signals the status of the transmission.
bit0..2 : %00 no transmission.
%01 frame(s) in transmission.
%10 token in transmission.
%11 transmission aborted.
bit3 : Signals the end of a token transmission can be used for stacking operation.
----------^----------

==========
'TopLevel\BITPROC\TXDATA\buf' is a register.

This register is 33 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register contains a three byte FIFO queue for transmission and some status information.

Bit32 : The transmission in progress, is a frame.
Bit31 : The transmission is aborted.
Bit29..30 : Last given Transmit Command : %01 TX_FR, %10 TX_TK.
Bit27..28 : Number of bytes in the Transmit Buffer.
Bit19..26 : First byte to be transmitted.
Bit18 : First byte is last byte of a frame.
Bit10..17 : Second byte to be transmitted.
Bit9 : Second byte is last byte of a frame.
Bit1..8 : Third byte to be transmitted.
Bi0 : Third byte is last byte of a frame.
----------^----------

==========
'TopLevel\BITPROC\TXDATA\cnt' is a register.

This register is 5 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register keeps track of the number of bits transmitted, and is used for controlling the finite-state machine.
----------^----------

Control specification:
----------v----------
%1X reset.
%01 inc.
%00 hold.
----------^----------

==========
'TopLevel\BITPROC\TXDATA\state' is a register.

This register is 3 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register contains the state of the finite-state machine which is implemented is TXBIT.
----------^----------

==========
'TopLevel\BITPROC\TXDATA\tk' is a register.

This register is 24 bits wide.
The default function is 'load'.
This register is loaded with value 9441462 (9010B6h) following system reset.

The value loaded for the 'reset' command is 9441462 (9010B6h).

Designer comments:
----------v----------
This register contains the token which would be send. It's reset value is :

```
 J K 0 J K 0 0 0 P P P T M R R R J K 1 J K 1 I E
% 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 1 1 0
```

Generating the non-data symbol J and K is done by the operator in TXBIT.
----------^----------

==========
'TopLevel\BITPROC\TXDATA\TXBIT' is an operator.

This operator has 1 function.
The default function is 'TXBIT'.

Designer comments:
----------v----------
The operator TXBIT contains one function. This fuctions controls the transmission of frames, tokens and abort sequences. In TXBIT a finite-state machine is implemented for controlling the transmission. The transitions in the finite-state machine are controlled by the inputs XMIT, BUF, STATE, TK and CNT.

The finite-state machine has the following states:
%000 : IDLE, no transmission.
%001 : WAIT SIGNAL, state between transmission of multiple frames. The bit-processor waits for a signal from the byte-processor to restart transmission.
%010 : FC..INFO, in this state the bytes in the transmit buffer are send. These bytes are the fields FS, DA, SA, and INFO.
%011 : ED, the ending delimiter is transmitted in this state.
%101 : SD, the starting delimiter is transmitted in this state.

```
%100 : FS, the frame status field is transmitted in this state.
       This field is transmitted as zero's.
%111 : AC, the access control field is transmitted in this state.
----------^----------

Text for function 'TXBIT' of 'TopLevel\BITPROC\TXDATA\TXBIT':
----------v----------
"These functions control the transmission of tokens and frames"
"This is done with the help of a build-in finite state machine"

"---------- Local functions for controlling state, tk and buf ----------"
_nhalt := (bitc at: 7).
_cnt7 := ((cnt width: 3) = 7) /\ write1.
_com := (buf from: 29 to: 30).
_tel := (buf from: 27 to: 28).
_tx := (_com ~= %00) /\ write1.

_abort := ((state = %010) /\ (_tel = 0)) \/ ((state = %000) /\ (xmit = %11)) \/
          ((buf at: 31) /\ (state ~= %000)).
_eofr := (state = %010) /\ _cnt7 /\ (buf at: 18).
_fr := (((state from: 1 to: 2) = %00)
        if1: ((xmit = %01) \/ (_com = %01))
        if0: (buf at: 32)
       ).
_i := (_abort \/ (_fr not) \/ ((tk at: 1) not)) not.

_pr := (prrr from: 3 to: 5).
_rr := (_fr
        if1: (0 width: 3)
        if0: (prrr from: 0 to: 2)
       ).

"---------- Function implementing a finite state machine ----------"
_nw_state := ((state = %000)                                    "IDLE"
              if1: ((xmit = %01)
                    if1: (2 width: 3)
                    if0: ((xmit at: 1) \/ _tx
                          if1: (5 width: 3)
                          if0: ((xmit = %11)
                                if1: (1 width: 3)
                                if0: state
                               )
                         )
                   )
             )

if0: ((state = %001)                                      "WAIT SIGNAL"
      if1: (_tx                                          "TO1B WAIT"
            if1: (5 width: 3)
            if0: state
           )

if0: ((state = %010)                                       "FC...INFO"
      if1: (_eofr
            if1: (6 width: 3)
            if0: (_abort /\ write1
                  if1: (5 width: 3)
                  if0: state
                 )
           )
```

```
if0: ((state = %110)                                    "FCS"
     if1: (write1
          if1: (_abort
               if1: (5 width: 3)
               if0: ((cnt = 7)
                    if1: (3 width: 3)
                    if0: state
               ) )
          if0: state
          )

if0: ((state = %011)                                    "ED"
     if1: (_cnt7
          if1: (_abort \/ (_fr not)
               if1: (0 width: 3)
               if0: (4 width: 3)
               )
          if0: state
    )

if0: ((state = %101)                                    "SD"
     if1: (_cnt7
          if1: (_abort
               if1: (3 width: 3)
               if0: (7 width: 3)
               )
          if0: state
          )

if0: ((state = %100)                                    "FS"
     if1: (_cnt7
          if1: (_i /\ _nhalt
               if1: (1 width: 3)
               if0: (0 width: 3)
               )
          if0: state
          )

if0: ((state = %111)                                    "AC"
     if1: (_cnt7
          if1: (_fr
               if1: (2 width: 3)
               if0: (3 width: 3)
               )
          if0: state
          )

if0: (0 width: 3)
)))))))).

nw_state := (reset
          if1: (0 width: 3)
          if0: _nw_state
          ).
```

```
"---------- Function for controlling a bit counter ----------"
cnt_ctrl := (reset \/ (state ~= _nw_state)), write1.

"---------- Functions for controlling the FCS generator ----------"
_resetfcs := (state ~= %010) /\ (state ~= %110).
fcs_ctrl := (_resetfcs \/ ((state = %110) /\ write1)),
          (_resetfcs \/ ((state = %010) /\ write1)).

"---------- Function to increase counters in COUNTERS ----------"
frincr := (state ~= %010) /\ (_nw_state = %010).
abincr := (state = %011) /\ (_nw_state = %000) /\ _abort /\ _nhalt.

"---------- Functions for controlling the contents of register 'buf' ---------"
"---------- Register 'buf' contains a queue of three bytes and some ---------"
"---------- status bits.                                            ---------"
_bytes := (((state = %010) /\ write1)
         if1: (_cnt7
              if1: (buf from: 0 to: 17),(0 width: 9)
              if0: (buf from: 19 to: 25),(0 width: 1),(buf from: 0 to: 18)
             )
         if0: (buf from: 0 to: 26)
        ).
_nw_tel := _tel - (((state = %010) /\ _cnt7) width: 2).
_state2 := (state from: 1 to: 2).
_nw_state2 := (_nw_state from: 1 to: 2).
nw_buf := ((_state2 = %00) /\ (_nw_state2 ~= %00)
          if1: _fr,_abort,(0 width: 2),_nw_tel,_bytes
          if0: ((_state2 ~= %00) /\ (_nw_state2 = %00)
               if1: _fr, (0 width: 1), _com,_nw_tel,_bytes
               if0: _fr,_abort,_com,_nw_tel,_bytes
         )    ).

"---------- Function for controlling contents of register 'tk' ----------"
"---------- Register 'tk' contains the token                  ----------"
_nw_i := _i \/ (_nw_state = %000).
nw_tk := (write1
         if1: ((state = %101)
              if1: (tk from: 16 to: 22),(tk at: 23),_pr,_fr,(tk at: 11),_rr,
                   (tk from: 2 to: 7),_nw_i,(tk at: 0)
              if0: ((state = %111)
                   if1: (tk from: 16 to: 23),(tk from: 8 to: 14),(tk at: 15),
                        (tk from: 2 to: 7),_nw_i,(tk at: 0)
                   if0: ((state = %011)
                        if1: (tk from: 16 to: 23),_pr,_fr,(tk at: 11),_rr,
                             (tk from: 0 to: 6),(tk at: 7)
                        if0: (tk from: 16 to: 23),_pr,_fr,(tk at: 11),_rr,
                             (tk from: 2 to: 7),_nw_i,(tk at: 0)
              )   )   )
         if0: tk
        ).
```

```
"---------- Function for outputting the correct data for transmission --------"
txdata := (((state = %011) \/ (state = %101)) /\ ((cnt = 0) \/ (cnt = 1) \/ (cnt = 3) \/ (cnt = 4))),
        ((state = %010)
         if1: (buf at: 26)
         if0: ((state = %110)
              if1: fcsbit
              if0: ((state = %011)
                   if1: (tk at: 7)
                   if0: ((state = %101)
                        if1: (tk at: 23)
                        if0: ((state = %111)
                             if1: (tk at: 15)
                             if0: (0 width: 1)
        )   )   )   )   ).

"---------- Signal to PRIO to indicate the state of transmission and ---------"
"---------- stack control                                          ---------"
_ed2idle := (state = %011) /\ (_nw_state = %000).
xstat := (_ed2idle /\ (_abort not)),
        (((state ~= %000) /\ (_fr not)) \/ _ed2idle),
        (((state ~= %000) /\ _fr) \/ (_ed2idle /\ _abort)).
----------------------^----------------------
```

End of function descriptions.

==========

'TopLevel\BITPROC\TXDATA\TXBYTE' is an operator.

This operator has 4 functions and is controlled by an unnamed control input.
The default function is 'NOP'.

Designer comments:

```
----------------------v----------------------
```

Through this operator the byte-processor initiates actions in the transmit process, and places bytes for transmission in the Transmit buffer.
The transmit command given is placed in register BUFFER, the byte to be transmitted also.

```
----------------------^----------------------
```

Control specification:

```
----------------------v----------------------
%X100000 COMMAND.
%1000000 RD_TXSTATE.
%X101100 WR_TXBUFFER.
%1101100 enable: in.  "Dummy read for modify on Transmit/Receive buffer"
%1000000 enable: in.
"Otherwise NOP"
----------------------^----------------------
```

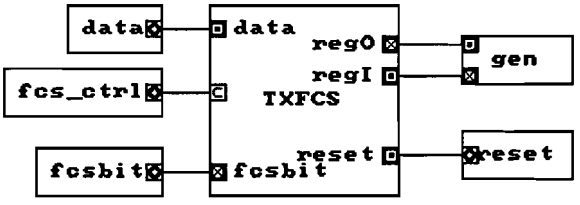
Text for function 'COMMAND' of 'TopLevel\BITPROC\TXDATA\TXBYTE':

```
----------------------v----------------------
"This function implements a part of the Command register. The following com-  "
"mands are processed : LAST_BYTE, LAST_FRAME, LAST_B&FR, TX_FR, TX_TK, TX_AB  "
"and CLEAR_TX. This operator puts zero on the in bus for the dummy read in    "
"case of a modify operation."
```

```
_command := (out from:0 to: 3).
_tel := (old_buf from: 27 to: 28).

_abort := (_command = %1100) \/ (old_buf at: 31).                    "TX_AB"
_com := ((_command = %0100) \/ (_command = %1000)                    "TX_FR \/ TX_TK"
       if1: (_command from: 2 to: 3)
       if0: (old_buf from: 29 to: 30)
      ).
_bytes := ((_command = %0001) \/ (_command = %0011)    "LAST_BYTE \/ LASTB&FR"
        if1: ((_tel = %01)
             if1: (old_buf from: 19 to: 26),(1 width: 1),
                  (old_buf from: 0 to: 17)
             if0: ((_tel = %10)
                  if1: (old_buf from: 10 to: 26),(1 width: 1),
                       (old_buf from: 0 to: 8)
                  if0: ((_tel = %11)
                       if1: (old_buf from: 1 to: 26),(1 width: 1)
                       if0: (old_buf from: 0 to: 26)
            )   )   )
        if0: (old_buf from: 0 to: 26)
       ).

buf := ((_command = %1111)                                    "CLEAR_TX"
       if1: (old_buf from: 31 to: 32),(0 width: 31)
       if0: (old_buf at: 32),_abort,_com,_tel,_bytes
      ).

tk := ((((_command = %0010) \/ (_command = %0011)) /\ (state ~= %011))
      if1: (old_tk from: 2 to: 23),(0 width: 2)     "LAST_FRAME \/ LAST_B&FR"
      if0: old_tk
     ).

in := (0 width: 8).                    "Dummy read for modify operation"
----------^----------

Text for function 'NOP' of 'TopLevel\BITPROC\TXDATA\TXBYTE':
----------v----------
"This function changes nothing."
buf := old_buf.
tk := old_tk.
----------^----------

Text for function 'RD_TXSTATE' of 'TopLevel\BITPROC\TXDATA\TXBYTE':
----------v----------
"This function implements the Transmit Status register, register 0. This func-"
"tion is only activated when a read operation on register 0 is performed.     "
buf := old_buf.
tk := old_tk.
_xstate := ((state ~= %000)
         if1: ((state = %001)
              if1: (1 width: 2)
              if0: ((old_buf at: 32)
                   if1: (2 width: 2)
                   if0: (3 width: 2)
             )   )
         if0: (0 width: 2)
        ).
```

```
in := ((old_buf from: 27 to: 28) = %11),ins,(old_buf at: 31),_xstate,
     (prrr from: 3 to: 5).
----------------------^----------------------

Text for function 'WR_TXBUFFER' of 'TopLevel\BITPROC\TXDATA\TXBYTE':
----------------------v----------------------
"This function implements the Transmit Buffer. The function also generated the"
"dummy read for the modify operation."

_tel := (old_buf from: 27 to: 28).
_byte := out,(0 width: 1).
_bytes := ((_tel = %00)
          if1: _byte,(old_buf from: 0 to: 17)
          if0: ((_tel = %01)
               if1: (old_buf from: 18 to: 26),_byte,(old_buf from: 0 to: 8)
               if0: ((_tel = %10)
                    if1: (old_buf from: 9 to: 26),_byte
                    if0: (old_buf from: 0 to: 26)
        )    )    ).
_nw_tel := ((_tel ~= %11)
          if1: (_tel + (1 width: 2))
          if0: _tel
         ).
buf := (old_buf from: 29 to: 32),_nw_tel,_bytes.

tk := old_tk.

in := (0 width: 8).                                    "Dummy read"
----------------------^----------------------

End of function descriptions.

==========================================================
END OF DOCUMENT.
```

TXFCS



IDaSS V0.08m document generated Wednesday December 15, 1993, 11:37:11.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========
'TopLevel\BITPROC\TXFCS' is a schematic.

Designer comments:
----------v----------
This schematic is used to generate the FCS for the transmitted frames. It's
controlled by the schematic TXDATA.

Inputs :
data : Data bus from TXDATA.
fcs_ctrl : Two bits control bus from TXDATA.
reset : The reset signal.

Outputs :
fcsbit : The next bit of the FCS field to be transmitted.
----------^----------

==========
'TopLevel\BITPROC\TXFCS\gen' is a register.

This register is 32 bits wide.
The default function is 'hold'.
This register is loaded with value 4294967295 (FFFFFFFFh) following system reset.

The value loaded for the 'reset' command is 4294967295 (FFFFFFFFh).

Designer comments:
----------v----------
This register is used for the generation, and transmission of the CRC.
----------^----------

==========
'TopLevel\BITPROC\TXFCS\TXFCS' is an operator.

This operator has 4 functions and is controlled by an unnamed control input.
The default function is 'HOLD'.

Control specification:
----------v----------
%11 RESET.
%01 LOAD.
%10 SEND.
%00 HOLD.
----------^----------

Text for function 'HOLD' of 'TopLevel\BITPROC\TXFCS\TXFCS':
----------v----------
"Hold the CRC generator"

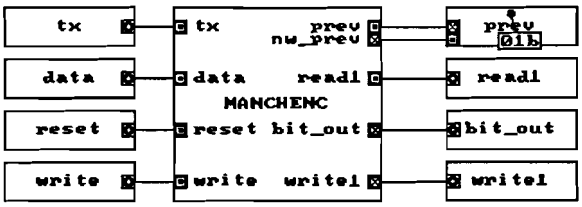
regO := regI.
fcsbit := (regI at: 0)not.
----------^----------

Text for function 'LOAD' of 'TopLevel\BITPROC\TXFCS\TXFCS':
----------v----------
"Calculate a 32 bits CRC:"

_Bit0 := regI at: 0.
_bit := (data at: 0) >< _Bit0.

```
regO := (reset
        if1: $FFFFFFFF
        if0: (
           _Bit,
           ((regI at: 31) >< _Bit), "32-1=31"
           ((regI at: 30) >< _Bit), "32-2=30"
           (regI at: 29),
           ((regI at: 28) >< _Bit), "32-4=28"
           ((regI at: 27) >< _Bit), "32-5=27"
           (regI at: 26),
           ((regI at: 25) >< _Bit), "32-7=25"
           ((regI at: 24) >< _Bit), "32-8=24"
           (regI at: 23),
           ((regI at: 22) >< _Bit), "32-10=22"
           ((regI at: 21) >< _Bit), "32-11=21"
           ((regI at: 20) >< _Bit), "32-12=20"
           (regI from: 17 to: 19),
           ((regI at: 16) >< _Bit), "32-16=16"
           (regI from: 11 to: 15),
           ((regI at: 10) >< _Bit), "32-22=10"
           ((regI at: 9) >< _Bit),  "32-23=9"
           (regI from: 7 to: 8),
           ((regI at: 6) >< _BIT),  "32-26=6"
           (regI from: 1 to: 5) )
        ).

fcsbit := _Bit0 not.
----------------------^----------------------

Text for function 'RESET' of 'TopLevel\BITPROC\TXFCS\TXFCS':
----------------------v----------------------
"Initialise the CRC generator"

regO := $FFFFFFFF.
fcsbit := (regI at: 0) not.
----------------------^----------------------

Text for function 'SEND' of 'TopLevel\BITPROC\TXFCS\TXFCS':
----------------------v----------------------
"Transmit the CRC,LSB first(inverted)"

regO := 1 ones,(regI from: 1 to: 31).
fcsbit := (regI at: 0) not.
----------------------^----------------------

End of function descriptions.

==========================================================
END OF DOCUMENT.
```

MANCHENC



IDaSS V0.08m document generated Wednesday December 15, 1993, 11:43:09.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========

'TopLevel\BITPROC\MANCHENC' is a schematic.

Designer comments:

----------v----------

Differential Manchester Encoder. The schematic also generates the clock signal writel.

Inputs :

data : Data stream to be transmitted.
readl : Clock signal from MANCHDEC.
reset : The reset signal.
tx : Indicates if the station is transmitting its own data, or repeating the incoming data stream, from PRIORITIES.
write : Clock signal from RINGIF.

Outputs :

bit_out : The encoded data stream.
writel : Clock signal that is asserted during write in the trailing bit of a symbol. Writel equals readl when input tx equals zero. Otherwise this schematic generates writel.

----------^----------

==========

'TopLevel\BITPROC\MANCHENC\MANCHENC' is an operator.

This operator has 1 function.
The default function is 'MANCHENC'.

Text for function 'MANCHENC' of 'TopLevel\BITPROC\MANCHENC\MANCHENC':

----------v----------

```
"This function performs the Diff. Manch. Coding.                    "
"prev0 contains the previous bit send to the ring.                  "
"prev1 indicates wether the leading or trailing bit is coded.       "

_data := ((prev at: 1)
        if1: (data at: 0)
        if0: (data at: 1)
       ).
```

```
_bit_out := ((prev at: 0) = _data).
nw_prev := (write
          if1: (tx
               if1: ((prev at: 1) not)       "(*)"
               if0: read1
              ),_bit_out
          if0: prev
         ).
bit_out := _bit_out.
write1 := write /\ ((prev at: 1)not).
```

"(*) When the station is transmitting its own data, the operator takes over the generation of write1. Otherwise the loss of a bit in the reception will possibly corrupt the transmission."

End of function descriptions.

==========

'TopLevel\BITPROC\MANCHENC\prev' is a register.

This register is 2 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

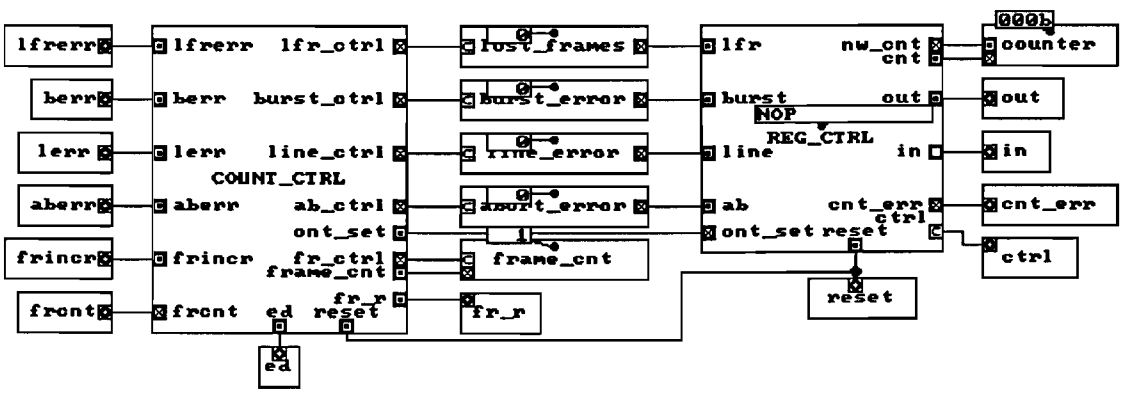
Designer comments:

This register is used to generate the output bit stream, and write1.
Bit0 contains the previous bit send to the ring.
Bit1 indicates if the bit, which is currently send, is the trailing or leading bit of a symbol.

==========

END OF DOCUMENT.

COUNTERS



IDaSS V0.08m document generated Wednesday December 15, 1993, 11:21:18.

Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========
'TopLevel\BITPROC\COUNTERS' is a schematic.

Designer comments:
----------v----------
This schematic contains the Burst Error, Line Error, Lost Frame Error, Abort Delimiter Transmitted and Frame Count counters. The error counters are 8 bits wide. An overflow indication is given when bit 8 becomes set.

Inputs :
aberr : Abort Delimiter Transmitted signal from PRIORITIES.
berr : Burst error indication from BURSTBUFFER.
ed : ED received signal from MANCHDEC.
fr_r : Frame Counter reset signal.
frincr : Frame Counter increment signal.
lfrerr : Lost Frame error signal from PRIORITIES.
lerr : Line error signal from RXFILTER.
out : 8 bits output bus from byte-processor.
reset : Reset signal.
ctrl : 7 bits address and control bus from byte processor.

outputs :
cnt_err : Indicates wether one of the counters is overflowed.
1 : overflow 0 : no overflow.
frcnt : Asserted when the Frame Counter does not equal zero.
in : 8 bits tri-state input bus towards the byte-processor.

The following actions are initiated through ctrl and out.
Write COMMAND register (register 0).
ctrl := %X100000 /\ out := %0001XXXX : READ_LEC
Makes the Line Error Counter the counter to be read and reset.
ctrl := %X100000 /\ out := %0010XXXX : READ_BEC
Makes the Burst Error Counter the counter to be read and reset.
ctrl := %X100000 /\ out := %0011XXXX : READ_LFEC
Makes the Lost Frame Error Counter the counter to be read and reset.
ctrl := %X100000 /\ out := %0100XXXX : READ_TXAB
Makes the Abort Delimiter Transmitted Counter the counter to be read and reset.
ctrl := %X10000 /\ out := %0111XXXX : READ_CNTS
Passes the information on overflown counters on the following read.

Read COUNTER register
ctrl := %1000010 /\ out := %XXXXXXXX
Puts the value of the counter for which the last READ command was given on out.

A dummy read is produced when ctrl := %1100010.
----------^----------

==========
'TopLevel\BITPROC\COUNTERS\abort_error' is a register.

This register is 8 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
Abort Delimiter Transmitted Counter
----------^----------

Control specification:
----------v----------
%1X reset.
%X1 inc.
----------^----------

==========
'TopLevel\BITPROC\COUNTERS\burst_error' is a register.

This register is 8 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
Burst Error Counter
----------^----------

Control specification:
----------v----------
%1X reset.
%X1 inc.
----------^----------

==========
'TopLevel\BITPROC\COUNTERS\COUNT_CTRL' is an operator.

This operator has 1 function.
The default function is 'CONTENTS'.

Designer comments:
----------v----------
This operator control the contents of the counters. The cnt_set input indicates
which counter must be reset.
----------^----------

Text for function 'CONTENTS' of 'TopLevel\BITPROC\COUNTERS\COUNT_CTRL':
----------v----------
"This function controls the contents of the error counters."

line_ctrl := (cnt_set at: 0) \/ reset,lerr.
burst_ctrl := (cnt_set at: 1) \/ reset,berr.
lfr_ctrl := (cnt_set at: 2) \/ reset,lfrerr.
ab_ctrl := (cnt_set at: 3) \/ reset,lfrerr.
fr_ctrl := fr_r \/ reset,ed,frincr.
frcnt := (frame_cnt ~= %00000000).
----------^----------

End of function descriptions.

==========
'TopLevel\BITPROC\COUNTERS\counter' is a register.

This register is 3 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
Countains the number of the counte to be read and reset.
----------^----------

==========
'TopLevel\BITPROC\COUNTERS\frame_cnt' is a register.

This register is 8 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 2 following system reset.

The value loaded for the 'reset' command is 2.

Designer comments:
----------v----------
The Frame Counter.
----------^----------

Control specification:
----------v----------
%1XX reset.
%001 inc.
%010 dec.
%011 hold.
----------^----------

==========
'TopLevel\BITPROC\COUNTERS\line_error' is a register.

This register is 8 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
Line Error Counter
----------^----------

Control specification:
----------v----------
%1X reset.
%X1 inc.
----------^----------

==========
'TopLevel\BITPROC\COUNTERS\lost_frames' is a register.

This register is 8 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
Lost Frame Error Counter
----------^----------

Control specification:
----------v----------
%1X reset.
%X1 inc.
----------^----------

==========
'TopLevel\BITPROC\COUNTERS\REG_CTRL' is an operator.

This operator has 3 functions and is controlled by control input 'ctrl'.
The default function is 'NOP'.

Designer comments:
----------v----------
This operator implements the COUNTER register, from which counter values can be
read, and a part of the COMMAND register. The counter register indentifies for
which the last READ command was.
----------^----------

Control specification:
----------v----------
%X100000 COMMAND.
%1000010 READ_COUNTER.
%1X00010 enable: in. "incl. dummy read for modify on register 2"
%1100000 enable: in. "Dummy read for modify on register 0"
----------^----------

Text for function 'COMMAND' of 'TopLevel\BITPROC\COUNTERS\REG_CTRL':

```
----------------------v----------------------
"This function processes the READ commands."
"---------- Functions for holding the counter number to be read ----------"
_out3 := (out from: 4 to: 6).
nw_cnt := (reset \/ (_out3 = %111)
          if1: (0 width: 3)
          if0: _out3
         ).
"--------- Standard function for creating the counter overflow signal ---------"
_full := %10000000.
cnt_err := (lfr >= _full) \/ (burst >= _full) \/ (line >= _full) \/ (ab >= _full).

"---------- Non of the counters is read and reset ----------"
cnt_set :=  (0 width: 4).

"---------- Dummy read for modify on register 0 ----------"
in := (0 width: 8).
----------------------^----------------------
```

Text for function 'NOP' of 'TopLevel\BITPROC\COUNTERS\REG_CTRL':

```
----------------------v----------------------
"This function is performed when there is no READ or COMMAND operation"
"---------- Standard function for creating the counter overflow signal --------"
_full := %10000000.
cnt_err := (lfr >= _full) \/ (burst >= _full) \/ (line >= _full) \/ (ab >= _full).

"---------- Counter setting does not change ----------"
cnt_set := (0 width: 4).
nw_cnt := cnt.

"---------- Dummy read for modify on register 2 ----------"
in := (0 width: 8).
----------------------^----------------------
```

Text for function 'READ_COUNTER' of 'TopLevel\BITPROC\COUNTERS\REG_CTRL':

```
----------------------v----------------------
"This function processes the read function performed on the error counters"
_cnt0 := cnt = %001.
_cnt1 := cnt = %010.
_cnt2 := cnt = %011.
_cnt3 := cnt = %100.
_flags := cnt =%000.
_full := %10000000.
_overfl := (0 width: 4),(ab >= _full),(lfr >= _full),(burst >= _full),(line >= _full).

"---------- Put value of addressed counter on in ----------"
in := (_flags
      if1: _overfl
      if0: (_cnt0
           if1: line
           if0: (_cnt1
                if1: burst
                if0: (_cnt2
                     if1: lfr
                     if0: ab
    )    )    )    ).
```

"---------- Reset the addressed counter ----------"
cnt_set := _cnt3,_cnt2,_cnt1,_cnt0.

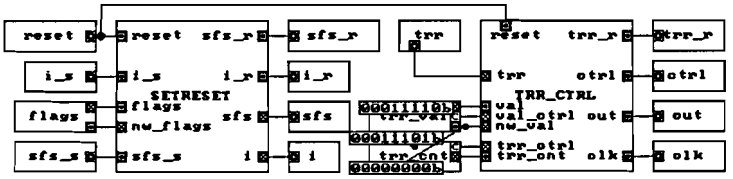
"---------- Addressed counter stays addressed ----------"
nw_cnt := cnt.

"--------- Standard function for creating the counter overflow signal ---------"
cnt_err := (_overfl > 0).
----------^----------

End of function descriptions.

==========
END OF DOCUMENT.

FLAGS



IDaSS V0.08m document generated Wednesday December 15, 1993, 11:41:29.
(c) August 1993 Eindhoven University of Technology, Netherlands,
Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========
'TopLevel\BITPROC\FLAGS' is a schematic.

Designer comments:
----------v----------
In this schematic the flags SFS and I and the counter TRR are implemented.

Inputs :
clk : Clock signal that increments the TRR.
ctrl : 7 bits address and control bus from the byte-processor.
i_r : Signal that resets the I flag, from PRIORITIES.
i_s : Signal that sets the I flag, from RXFILTER.
out : 8 bits data output bus from the byte-processor.
reset : The reset signal.
sfs_r : Signal that resets the SFS flag, from PRIORITIES.
sfs_s : Signal that sets the I flag, from RXFILTER.
trr_r : Signal that restarts the TRR.

Outputs :
i : Indicates if the I flag is set. (%1 is set)
sfs : Indicates if the SFS flag is set. (%1 is set)
trr : Indicates if the TRR is expired. (%1 is expired)
----------^----------

==========
'TopLevel\BITPROC\FLAGS\flags' is a register.

This register is 2 bits wide.
The default function is 'load'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register contains the I (bit0) and SFS flag (bit1).
----------^----------

==========
'TopLevel\BITPROC\FLAGS\SETRESET' is an operator.

This operator has 1 function.
The default function is 'SET_RESET'.

Text for function 'SET_RESET' of 'TopLevel\BITPROC\FLAGS\SETRESET':
----------v----------
"This function controls the setting and resetting of the sfs and i flag. "
_sfs := (((flags at: 1) /\ (sfs_r not)) \/ sfs_s) /\ (reset not).
_i := (((flags at: 0) /\ (i_r not)) \/ i_s) /\ (reset not).
sfs := _sfs.
i := _i.
nw_flags := _sfs,_i.
----------^----------

End of function descriptions.

==========
'TopLevel\BITPROC\FLAGS\trr_cnt' is a register.

This register is 8 bits wide and is controlled by an unnamed control input.
The default function is 'increment'.
This register is loaded with value 0 following system reset.

The value loaded for the 'reset' command is 0.

Designer comments:
----------v----------
This register contains the current value of the TRR.
----------^----------

Control specification:
----------v----------
%1X reset.
%01 inc.
%00 hold.
----------^----------

```
==========================================================
'TopLevel\BITPROC\FLAGS\TRR_CTRL' is an operator.

This operator has 1 function.
The default function is 'TRR'.

Text for function 'TRR' of 'TopLevel\BITPROC\FLAGS\TRR_CTRL':
----------v----------
"This function controls the Timer Return to Repeat.                    "
_reset := (reset \/ trr_r \/ ( ((ctrl = %0100000) \/ (ctrl = %1100000))
                          /\ ((out from: 4 to: 7) = %1000))).

_trr_exp := (trr_cnt = val).
trr_ctrl := _reset,(clk /\ (_trr_exp not)).
trr := _trr_exp.
nw_val := out.
val_ctrl := reset,((ctrl = %0100010) \/ (ctrl = %1100010)).
----------^----------

End of function descriptions.

==========================================================
'TopLevel\BITPROC\FLAGS\trr_val' is a register.

This register is 8 bits wide and is controlled by an unnamed control input.
The default function is 'hold'.
This register is loaded with value 30 (1Eh) following system reset.

The value loaded for the 'reset' command is 30 (1Eh).

Designer comments:
----------v----------
This register contains the time-out value of the TRR.
----------^----------

Control specification:
----------v----------
%1X reset.
%01 load.
----------^----------

==========================================================
END OF DOCUMENT.
```

RESET

Reset is just an operator. Therfore there is no printing like at the other schematics.

IDaSS V0.08m document generated Wednesday December 15, 1993, 11:44:03.

Digital Systems Group (EB). Email: verschue@eb.ele.tue.nl .

This is a partial document.

==========
'TopLevel\BITPROC\RESET' is an operator.

This operator has 1 function.
The default function is 'RESET'.

Designer comments:
----------v----------
This operator generates the reset signal to reset the complete bit-processor.
It implements the RESET_ALL command of the Command register.
----------^----------

Text for function 'RESET' of 'TopLevel\BITPROC\RESET':
----------v----------
reset := ((ctrl = %0100000) \/ (ctrl = %1100000)) /\ (out = %11111111).
----------^----------

End of function descriptions.

==========
END OF DOCUMENT.

Appendix 3 : The Design

This appendix describes the design of the TRC in a top-down fashion. This partitioning does not reflect the actual design approach. Indeed the first stage of the design was the partitioning of the TRC in a bit- and byte-processor, and of the bit-processor in several logic blocks each containing a distinct function. But this partitioning was pure specified in natural language.
In the second stage, each of the logic blocks of the bit-processor was implemented in IDaSS. Those blocks who where strongly dependent were designed concurrent.
As the used byte-processor was an existing architecture, only the design of the bit-processor is described. First the clock circuit used is described, followed by the description of the 13 logic blocks of the bit-processor.
The layout of the bit-processor, and the schematics, can be found in appendix 2.

3.1 Timing

The timing of the reception and transmission of bits is based on the three external clock signals *pl_load*, *pl_send* and *ms_clk*. Chapter 5.1.3 already explained the timing of the reception and transmission of the bit stream.
Inside the bit-processor five clock signals are used to process the bits and data. (We speak of data when decoded information, symbols, are meant). These clock signals are *load*, *read*, *read1*, *write*, *write1*.
The *load* signal is always equal to *pl_load*. *read* and *write* are equal to *pl_load* or equal to *ms_clk* dependent on the state of the bit-processor as indicated on the *bitc* bus. These three signals are generated in the schematic RINGIF.
read1 and *write1* operate on half the frequency of *read* and *write*, as they are only asserted during the trailing bit of a symbol. *read1* is generated by MANCHDEC. MANCHDEC continuously adjusts *read1* on the detection of a start/end delimiter to correct the loss or insertion of a bit. *Write1* is generated by MANCHENC. MANCHENC equals *write1* to *read1* or it generates *write1* on its own, again dependent on the mode of the bit-processor.
The signals *load*, *read* and *read1* are used to process the received bits/data. *write* and *write1* are used to process the bits/data for transmission.
Figure A3.1 shows which schematics uses which clock signals.

Dependent on the mode of the bit-processor the internal clock signals are equal to the external signals *pl_load*, *ms_clk*. Four different cases can be distinguished :

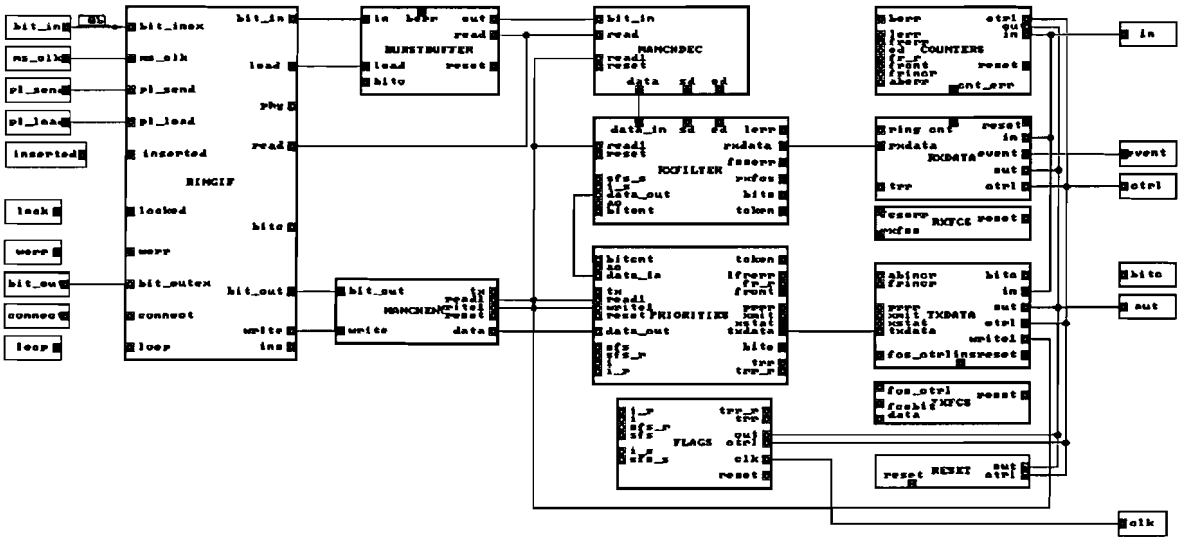


Figure A3.1 Bit-processor clock signals and data lines

Case 1 : Active = 0, Force_TX = 0
In this case the TRC is a normal functioning Standby Monitor. Received bits are repeated, or its own data is transmitted, according to the Operational FSM. Reception and transmission is based on *pl_load* and *pl_send*. The internal clock signals *load*, *read* and *write* are all equal to *pl_load*. *write1* is equal to *read1* when the bit-processor is repeating data. When the bit-processor starts to transmit its own data, the MANCHENC takes over the generation of *write1*. In this way the loss of a bit in the reception of data will have no impact on the transmission.

Case 2 : Active = 1, Force_TX = 0
In this case the TRC is a normal functioning Active Monitor. Here also the received bits are repeated, or its own data is transmitted, according to the Operational FSM. But now the reception of bits is based on *pl_load* and the transmission on *ms_clk*.
Loading the received bits into the latency buffer, placed in the schematic BURSTBUF, is controlled by *(pl_)load*. The clock signal *read* indicates when a new bit must be read from the latency buffer, and processed to MANCHDEC. As bits are possibly repeated, *write* is equal to read, and as the transmission timing is based on *ms_clk*, *write* and *read* are equal to *ms_clk*. The generation of *write1* is performed in the same way as in case 1.

Case 3 : Active = 0, Force_TX = 1
In this case the TRC is a Standby Monitor but an error has occurred because of which the station entered the Transmit Claim Token or Transmit Beacon state. The received bits are no longer repeated, and the TRC should continuously be transmitting particular frames.
The latency buffer is not inserted, so *read* is equal to *(pl_)load*.
Transmission though is based on the master oscillator. Therefore *write* is equal to *ms_clk*. In this case *write1* is always generated by MANCHENC, as *write* and *read* are not equal.

Case 4 : Active = 1, Force_TX = 1
In this case the TRC is an Active Monitor but an error has occurred which caused the station to enter the monitor state Transmit Purge or Transmit Fill state. The received bits are no longer repeated, and the TRC should continuously be transmitting Purge frames, or fill.
Loading the received bits into the latency buffer is still controlled by *(pl_)load*. Reading bits from the latency buffer is controlled by *read*, which is in this case also equal to *ms_clk*. Transmission is based on the master oscillator, so *write* is also equal to *ms_clk*. *write1* is always generated by MANCHENC.

The generation of the clock signals *read1* and *write1* will be outlined in chapter A3.4 and A3.11.

3.2 RINGIF

The RINGIF schematic controls the reception and transmission of bits, and generates several clock signals. These clock signals are used by the other schematics to process the bits and data. (The decoded bits, the actual symbols, are referred to as data)

INPUT

bitc : Setting of the bit-processor.
bit_inex : The input bit stream from the ring.
bit_out : The bit stream to be send to the ring, from MANCHENC.
inserted : When asserted the TRC assumes it's inserted into the ring, or the loop circuit is established.
locked : When asserted the TRC assumes the PLL has phase locked within the timing limits.
ms_clk : The clock signal based on the master oscillator, for transmission timing. See chap. 5.1.3.
pl_load : The clock signal based on the clock recovered from the incoming bit stream by a PLL, used for reception timing. See chap. 5.1.3.
pl_send : The clock signal based on the clock recovered from the incoming bit stream by a PLL, used for transmission timing. See chap. 5.1.3.
werr : When asserted low the TRC assumes there is a wiring fault.

PROCESS

Operator : SYNCHRO
This operator implements the function of loading received bits into the TRC, and sending bits to the ring with the help of one register. The operator also determines the clock signals *load*, *read* and *write*. Chapter A3.1 explained the determination of these clock signals. The operator also controls the signalling to and from the external Ring Interface.

Register : SYNC
SYNC is a two bits register used for the synchronization of the incoming and outgoing bit stream. Bit0 stores a received bit. Bit1 contains the bit that is put on the ring.

OUTPUT

bit_in : The input bit stream from the ring, that is loaded into the TRC.
bit_outex : The output bit stream to the ring.
connect : Asserted active high when the Connect bit on *bitc* is set.
load : Equal to the *pl_load* input.
loop : Asserted active low when the Loop bit on *bitc* is set.
phy : Two bits bus that indicates if the PLL has locked and if there is a wiring error.
bit0 : Asserted when the PLL has not locked.
bit1 : Asserted when there is a wiring error.
read : Clock signal that indicates when a new received bit must be processed. See chap. A3.1.
write : Clock signal that indicates when a new bit must be processed for transmission. See chap. A3.1.

3.3 BURSTBUFFER

The schematic BURSTBUFFER contains the burst five error detector and the latency buffer. Two operators, two registers and one FIFO queue implement these two functions. The registers and FIFO queue are reset when reset is asserted high.

INPUT

bitc : Setting of the bit-processor.
in : Input bit stream, from RINGIF.
load : Clock signal from RINGIF.
read : Clock signal from RINGIF.
reset : The reset signal.

PROCESS

Operator : BURSTDET
When the burst five error detector receives more than four bits of the same polarity in succession, it introduces a change of polarity, a transition, at the end of the fourth bit. It continues to introduce such a transition each bit time until a transition is received from the ring.
The error signal *berr* is asserted during one system clock when a burst five error is detected. A burst six (or more) error will not cause multiple assertion of *berr*.

Register : REG
Register REG contains the following information:

bit0..3 : The previous three received bits.
bit4 : The previous output bit.
bit5 : Is set on the detection of a burst five error, and reset when a transition is received at the input. This bit is used to avoid multiple assertion of *berr* in case of a burst six (or more) error.

The timing of the operator BURSTDET, and the contents of the register, are controlled by the *load* signal.

Operator : LAT_CTRL
The latency buffer is inserted when the Active bit (bit6) on *bitc* is set. During initialization and underrun of the buffer, the operator introduces a transition each bit time. The buffer is first filled to its initial length when it is inserted. During this initialization bits are loaded into the buffer but not read out of it.
The operator uses the FIFO queue LAT, register STATE and register CNT to implement the latency buffer.

Register : CNT
This 7 bits wide counter is used to keep track of the number of bits in the Latency buffer, which is the FIFI LAT.

FIFO : LAT_BUF
The FIFO LAT contains the latency buffer. Bits are loaded into the buffer using the *load* signal. Bits are read out of the buffer using the *read* signal, which is equal to *ms_clk* when the Active bit is set and the latency buffer is inserted.

Register : STATE
The register STATE contains the following information:

bit0 : Is set when the buffer is initialized.
bit1 : The previous output bit. This bit is used to introduce transitions during initialization and underrun of the buffer.

OUTPUT

out : The output bit stream. This bit stream is always timed to read.

3.4 MANCHDEC

The schematic MANCHDEC performs the differential Manchester decoding and the start/end delimiter detection. A clock signal (*read1*) is generated to indicate the presence of the second bit of a symbol. One operator and one register are used to perform these functions.

INPUT

bit_in : The input bit stream from BURSTBUFFER.
read : Clock signal from RINGIF.
reset : The reset signal.

PROCESS

Operator : MANCHDEC
The input bit stream (*bit_in*) is decoded on the two bits data bus (data). During the leading bit of a symbol, data0 indicates if the symbol is a 1/J (data0=1) or a 0/K (data0=0), and data1 is don't care. During the trailing bit of a symbol, data0 doesn't change, and data1 indicates if the symbol is a binairy data symbol (data1=0), or a non-data symbol (data1=1).
The timing signal *read1* indicates when the trailing bit of a symbol is present. This signal is continuous adjusted on the detection of a start/end delimiter. In this way *read1* can be asserted in two successive bits!!

Register : DELIMITER
The 18 bits register DELIMITER is used for decoding and delimiter detection. The register contains the following information:

Bit0..15 : The previous 16 received bits.
Bit16 : The previous decoded value put on data0.
Bit17 : Indicates wether the leading or trailing bit of a symbol is received.

OUTPUT

data : Decoded input bit stream.
ed : Asserted when *read* is asserted during the trailing bit of the sixth symbol of the ending delimiter.
read1 : Asserted when *read* is asserted during the trailing bit of a symbol.
sd : Asserted when *read* is asserted during the trailing bit of the eight symbol of the starting delimiter.

3.5 RXFILTER

The RXFILTER extracts the required information from the incoming bit stream, and also sets the E, A and C bits when necessary. The schematic uses two operators and six registers.

INPUT

bitc : The setting of the bit-processor.
data_in : The received data stream from MANCHDEC.
ed : ED received signal from MANCHDEC.
fcserr : Asserted when RXFCS detected a FCS error.
read1 : Clock signal from RINGIF.
sd : SD received signal from MANCHDEC.

PROCESS

Operator : REPSTATE
This operator performs several functions simultaneously which are all related to the global function of receiving data.
The primary function of this function is to keep track which field of a frame or token is received. Therefore the operator contains a function that implements a finite-state machine. This finite-state machine is controlled by the inputs *sd*, *ed* and the values of the registers BIT_CNT and BYTE_CNT. Register STATE holds the current state of the finite-state machine.

Error detection is the second function of this operator. Every frame is checked on the presence of a code violation, octet error, and frame length error. The FF bits of the FC field are also checked if they are equal to %00 or %01. The operator generates the three bits wide bus *rxbit*. Through this bus the operator controls the frame check sequence checker in RXFCS. RXFCS indicates on *fcserr* if a FCS error has occurred. The operator holds the error status of the received frame in register FF. This register indicates wether the FF bits where correct, if the frame is a Frame Not Good, and if the frame is a Frame With Error.
A token is checked on code violations in the AC field and on the proper ending. The operator indicates on the three bits bus *token* wether a token is received with or without a code violation, and if it ended properly or not. This bus is used by the schematic PRIORITIES.

The third function of the operator is passing the received frames and tokens byte-wise, with error indication, to the schematic RXDATA. The operator passes the AC, FC, DA, SA, INFO and FS fields of a frame and the AC field of a token. In order to extract the FCS field and the ending delimiter the 46 bits wide register BUFFER is used. In this way the first byte of a frame, the AC field, is passed with a delay of 38 bits.

The last function of the operator is the generation of four control signals. The first signal, is the control signal to increment the Line Error Counter. This function implements the receive function RX1.
Two control signals are generated to set the I and SFS flag when required. These signals implement the Receive functions RX4 and RX5.
The fourth control signal is asserted during the reception of the AC field of a frame or token. It is deasserted after the AC field has passed or a code violation has occurred. Schematic PRIORITIES uses this signal to claim tokens, store the priority and reservation of the received frame or token, and to set the reservation in the repeated frame or token.

Register : BIT_CNT
This three bits register indicates the bit-number of the received bit.

Register : BYTE_CNT
This 5 bits wide counter is used to count the number of received bytes of a frame. The counter is mainly used to check the minimum frame length. Therefore the register is no wider than 5 bits, and will not be incremented when it reaches its maximum value.

Register : BUFFER
The operator REPSTATE uses this 46 bits wide buffer to delete the FCS and ED fields of received frames, and the ED of received tokens.

Register : FF
This three bits register indicates the error states of the received frame or token. Operator REPSTATE resets this register when the received frame or token ends, or an SD is received.

bit0 : Reset when a frame with FF bits not equal to %00 or %01 is received.
bit1 : Set when a Frame Not Good is received.
bit2 : Set when a Frame With Error is received.

Register : REP_STATE
This three bits register contains the state of the FSM implemented in the operator REPSTATE.

Register : SYMERR
This register is used for the extraction of the ending delimiter for the checking on code violations. The register is reset on the detection of an SD or ED.

bit0..3 : The last four received values of data1, which is the non-symbol indication (code violation).
bit4 : Set when a code violation has occurred.

Bit4 is set when a code violation has occurred in the AC or FS field, or bit3 is set when the reception of a frame is somewhere in the FC, DA, SA, INFO or ED field. The first three bits are used to extract the ED field.

Operator : REPFRAME
This operator sets the E, A and C bits when required. The C and or A bits are set when the COPY and or ADR bits on *bitc* are set and the frame was a Frame Good as indicated in register FF. The E bit is set when the contents of FF indicate that the frame is a Frame With Error. The operator implements the Transmit Operational functions TO0B, TO0C and TO0D.

OUTPUT

ac : This two bits bus indicates when, and what kind of AC field is received.
- bit0 : Asserted during the receipt of an AC field, until a code violation has occurred.
- bit1 : Asserted from the Token bit, when it equalled 0, until bit0 is deasserted.

bitcnt : The value of register BIT_CNT.

data_out : The output, received, data stream.

i_s : Asserted when *readl* is asserted during the I bit of the ED of a frame when this I bit equals 0, or when an abort sequence is detected.

lerr : Asserted when *readl* is asserted during the trailing bit of the E symbol, when the E bit equals 0, and the frame was a Frame With Error or the token had a code violation.

rxfcs : A three bits wide data bus carrying the received bits and control signals for the FCS calculation.

- bit0 : The received bit.
- bit1 : When asserted, the value on bit0 is considered valid and added to the calculation.
- bit2 : When asserted the calculation resets.

sfs_s : Asserted when *readl* is asserted during the last bit of the AC field of a frame.

token : This three bits bus indicates the reception of a token used by the schematic PRIORITIES.

- bit0 : Asserted when *readl* is asserted during the trailing bit of the symbol which should be the sixth symbol of the ED of the token, and the ED is detected.
- bit1 : Asserted when *readl* is asserted during the trailing bit of the symbol which should be the sixth symbol of the ED of a token, but no ED is detected. Hence, the token did not end properly.
- bit2 : Asserted when a code violation is detected in the AC field of the token.

rxdata : *Rxdata* is a 13 bits bus carrying the received bytes including identifications of the byte, and error information.

- bit0..7 : The received byte.
- bit8 : Asserted when the byte is valid. This bit is asserted during the time *readl* is asserted.
- bit9 : This bit is asserted when the received frame is identified as a Frame Not Good, or when a token is detected that did not end properly.
- bit11..12 : These two bits identify the type of information that is received. These bits are set during the time the specific type of information is received, and not only when bit8 is asserted.
  - %10 A token is received.
  - %11 The FC, DA, SA, INFO, FCS and ED fields are received. (The FCS and ED fields are not passed!!)
  - %01 The FS field is received.
  - %00 Nothing is received.

  A not proper ending of a frame, with an abort sequence for example, is identified when these bits change from %11 to %00.

3.6 RXDATA

The schematic RXDATA processes the information received on *rxdata*, towards the byte-processor. This schematic implements the Receive buffer, the Receive Status register and the Event channel (*event* bus). Two operators and two registers are used.

INPUT

cnt : Asserted when one of the error counters in COUNTERS has overflown.
ctrl : The seven bits address and control bus from the byte-processor.
out : The eight bits output bus from the byte-processor.
ring : Two bits bus that indicates wether the PLL has locked, and a wiring error exists, from RINGIF (*phy*).
trr : FLAGS asserts this signal when the TRR has expired.
reset : The reset signal.
rxdata : Rxdata is the 13 bits wide bus through which the schematic RXFILTER passes the information on the received frames and tokens.

PROCESS

Operator : RXBIT
The RXBIT operator processes the information it receives on the *rxdata* bus. The processed information is stored in the registers STATE and BYTE. The received byte is stored in register BYTE. This register also contains an extra bit to indicate the presence of a byte in the register. Register STATE contains the four bits for the *event* bus, the eight bits of the Receive register, and one extra status bit. The operator uses this extra bit to identify wether the received frame ended properly or not.

Register : BYTE
BYTE contains one byte of the two bytes Receive buffer. The other byte is placed in the schematic RXFILTER, where is's bit-wise filled. One extra bit is provided to indicate the presence of a byte in the register (bit0).

Register : STATE
This register is used to implement the Event channel and the Receive Status register. Bits 0 to 3 contain the value put on the *event* bus. Bits 4 to 11 is the Receive status register. Chapter 5.2.1 specifies the functioning of these bits. Bit 12 is used to identify wether the received frame ended properly or not.

Operator : RXBYTE
The RXBYTE operator contains three functions of which one is selected through a control connector connected to the *ctrl* bus.
Through this operator, the byte-processor accesses the Receive Buffer and the Transmit Status register. All three functions also generate the values for the *event* bus.

Function 1 : RD_RXSTATUS ctrl := %1000001
The function implements the reading of the Receive Status register. Bits 4 to 11 are put on the *in* bus, which is now enabled. *Event* equals the first three bits of register STATE. Bit 3 in STATE, which is the error bit on *event*, is reset.

Function 2 : RD_RXBUFFER ctrl := %1001100
This function implements the reading of the Receive Buffer. The byte in register BYTE is put on in, which is now enabled. *event* equals the first three bits of register STATE. Bits 0 to 2 in STATE, which are bit 0 to 2 on *event*, are reset.
The byte indication bit in BYTE is also reset.

Function 3 : NOP ctrl := otherwise
The operator performs the NOP function when non of the above functions is selected by *ctrl*. This functions puts the value of bit 0 to 2 of register STATE on *event*.
The register STATE and BYTE are left unchanged.

OUTPUT

event : The four bits Event channel.
in : Tri-state bus input bus of the byte-processor.. This bus is driven by the REG_CTRL operator when *ctrl* equals %1X00001 or %1001100

3.7 RXFCS

This schematic performs the frame check sequence checking on the received frames. The loading of bits, and resetting of the checker is controlled from out of the schematic RECFILTER. This schematic is a slightly modified version of the FCS checker of the HDLC/SDLC bit-processor.

INPUT

reset : Resets the calculation when asserted.
rxfcs : The three bits bit and control bus, generated by RXFILTER.

PROCESS

Operator : CRCCHK
This straightforward operator is used to calculate the FCS. The calculation is performed with the help of one 32 bits register. The operator outputs the result of the FCS calculation continuously on *fcserr*. The result is a simple correct (fcserr=0), or incorrect (fcserr=1).

Register : CRCREG
A 32 bits register used to perform the FCS calculation.

OUTPUT

fcserr : Asserted when the calculation in progress indicates a FCS error.

3.8 PRIORITIES

The schematic PRIORITIES implements almost the complete Operational FSM of the Token Ring standard. It also stores the last received priority and reservation. The schematic uses three operators, four registers and two FIFO queues.

INPUT

ac : This two bits bus indicates when, and what kind of AC field is received, and is generated by the schematic PRIORITIES.
bitc : The setting of the bit-processor.
bitcnt : The value of register BIT_CNT in schematic RXFILTER.
data_in : The received data, after the schematic RXFILTER.
frcnt : Asserted when the Frame Counter does not equal zero. Generated by COUNTERS.
i : Asserted when the I flag is set in FLAGS.
read1 : Clock signal generated by MANCHDEC.
reset : The reset signal.
sfs : Asserted when the SFS flag is set in FLAGS.
trr : Asserted when the TRR has expired in FLAGS.
token : This three bits bus indicates the reception of a token, and is generated by the schematic PRIORITIES.
txdata : The stations data for transmission from TXDATA.
write1 : Clock signal generated by MANCHENC.
xstat : The three bits signalling of the status of the transmission, as indicated by TXDATA.

PROCESS

Operator : PRIO_STATE
This operator performs the functioning of the Operational FSM, with the help of the register PR_STATE. This register stores the state of the FSM. The operator also generates the necessary signalling for the setting of the I and SFS flags (Receive actions RX4 and RX5), the TRR timer and Frame Counter. When the FSM indicates the start of the transmission of the station's data, or the release of a new token, it's signalled to the schematic TXDATA. The FSM remains in the initial state "HALTED", when the Resume bit on *bitc* is not set. The clock signal *read1* controls the state transitions of the FSM.

This schematic has only one transmit frame state TO1, that is the combination of the TX Frame and Wait states TO01A and TO1B. The WAIT state (TO01B) is generated by the schematic TXDATA.
The signal *tx* is generated to indicate wether the station is repeating the incoming data stream, or it's transmitting its own data. Hence, when the bit-processor is not in the Transmit Operational state Repeat, or Transmit Halted state Idle.
The *set* signal indicates when DATA_FUNCS has to set the Token bit, Monitor bit or reservation in a passing frame or token.

Register : PR_STATE
This three bits register contains the current state of the FSM.

Operator : PRIO_STACK
This operator stores the priority (Pr) and reservation (Rr) of the last received frame or token (Receive action RX3). It also controls the stacks SX and SR (Receive action RX2 and the stacking operations in the Transmit Operational states TO3 and TO5 and Transmit Halted state TH3). The *xstat* bus indicates when a stacking operation has to be performed. The storing of the priority and reservation is always controlled by the *write1* clock signal. When a stacking operation is performed the values of Pr and Rr are updated. PRIO_STACK indicates, on the *prrr* bus, the values of the priority and reservation for the token to be released. The bus pmrr is generated to indicate to DATA_FUNCS wether the reservation can be set to Pm.

Register : BUF
This two bits register is used for the storing of the priority and reservation of the received frame or token in register REG. The operator PRIO_STACK uses this register to avoid the storage of these values when the AC field has a code violation.

Register : REG
Register REG contains the priority and reservation of the last received frame or token.

Register : STACKNR
The value in this register indicates the number of values on the stacks SX and SR.

FIFO : SX
The Sx stack.

FIFO : SR
The Sr stack.

Operator : DATA_FUNCS
The operator DATA_FUNCS has two functions controlled by the control connector, connected to the *tx* signals from PRIO_STATE.

Function 1 : REPEAT tx = %0
In this state the incoming data stream is repeated. The functions sets the Token bit equal to one or the reservation bits equal to Pm when the *set* signal is asserted.

Function 2 : TX0MIT tx = %1
In this state the data from the schematic TXDATA, bus *txdata*, is transmitted. The incoming data is no longer repeated.

OUTPUT

data_out : The output data stream for transmission. This is either *data_in* or *txdata*.
fr_r : Asserted when the Frame Counter must be reset.
i_s : Asserted when the I flag must be reset.
lfrerr : Asserted when a Lost Frame error is detected.
prrr : Six bits bus that contains the priority and reservation of AC field to be transmitted.
sfs_s : Asserted when the SFS flag must be reset.
trr_r : Asserted when the TRR must be restarted.
tx : Asserted when the station's own data is transmitted.
xmit : Two bits signal that indicates the start of a transmission. This signal is only generated when the Resume bit on *bitc* is set. Otherwise the start of a transmission is controlled by the byte-processor.
%00 : Nop
%01 : Start frame transmission.
%10 : Start token transmission.
%11 : Start transmission of a Abort Sequence.

3.9 TXDATA

The schematic TXDATA generates the frames and tokens that are to be transmitted by the station. It contains the Transmit buffer, the Receive status register and a part of the Command register. Two operators and three registers are used.

INPUT

bitc : The setting of the bit-processor.
ctrl : 7 bits address and control bus from the byte-processor.
fcsbit : The FCS bit from TXFCS, for transmission.
out : 8 bits data output bus from the byte-processor.
prrr : The priority and reservation for the transmission of the AC field.
reset : The reset signal.
write1 : Clock signal from RINGIF.
xmit : Signals when a token, frame or Abort Delimiter must be transmitted.

PROCESS

Operator : TXBIT
The operator TXBIT passes the data that has to be transmitted to the schematic PRIORITIES. The operator gets the data for transmission from the registers BUF and TK, which contain the Transmit buffer and the token. A transmit command, TX_FR TX_TK or TX_AB, is also passed through the register BUF, by TXBYTE.
In order to keep track of the status of the transmission, the operator contains a function that implements a finite-state machine. The register STATE contains the status of this finite-state machine. This finite-state machine also causes the Transmit Operational state TO1B WAIT.
As mentioned, this operator communicates with the operator TXBYTE through the registers BUF and TK.

Register : BUF
This 33 bits wide register contains the three bytes Transmit buffer, the last transmit command given and some status bits.

bit32	:	Set when a frame is transmitted.
bit31	:	Set when the transmission is/must be aborted.
bit29..30	:	The last transmit command given. %00 Nop %10 TX_TK %01 TX_FR %11 Not defined.
bit27..28	:	The number of bytes in the Transmit buffer.
bit26..19	:	The first byte in the Transmit buffer.
bit18	:	Set when the first byte in the buffer is the last byte of a frame.
bit10..17	:	The second byte in the Transmit buffer.
bit9	:	Set when the second byte in the buffer is the last byte of a frame.
bit8..1	:	The third byte of the Transmit buffer.
bit0	:	Set when the third byte in the buffer is the last byte of a frame.

Register : State
This register contains the current state of the FSM, that is implemented in the operator TXBIT.

Register : TK
This 24 bits wide buffer contains the SD, AC and ED fields for the frame or token to be transmitted. The contents of this register depends on the value on prrr, if a token or frame is transmitted, and if when a frame is transmitted if it's the last frame.

bit0..bit7	:	The ED to be transmitted.
bit8..bit15	:	The AC field to be transmitted.
bit16..bit23	:	The SD to be transmitted.

The operator TXBIT knows when to indicate on data1 when the transmitted symbol is a non-data symbol.

Operator : TXBYTE
The operator TXBYTE contains the function of filling the Transit buffer by the byte-processor, a part of the Command register, and the Transmit status register. The operator recognizes and processes the TX and LAST commands.

When a byte is written into the Transmit buffer, the byte is placed in BUF. If the Transmit buffer is full this byte will be lost.
The TX_FR and TX_TK are written into the Command register by the byte-processor, these commands are indicated in bit29..30 of register BUF. The TX_AB command is indicated by setting bit31 of register BUF.
If the LAST_BYTE or LAST_B&FR command is given, the appropriate bit, bit0, bit9 or bit18, in the register BUF is set depending on the number of bytes in the Transmit buffer. The LAST_FRAME and the LAST_B&FR commands reset bit1, the I flag, in register TK. In this way the operator TXBIT knows that the transmitted frame is the last frame the station wants to transmit. The I symbol will only be reset when the operator TXBIT is not transmitting the ED field. Also will the I symbol not be reset when there is no frame transmitted.

OUTPUT

abincr : Asserted during *write1* in the last symbol of an Abort Delimiter.
fcs_ctrl : 3 bits bus which controls the FCS generator (TXFCS)
%00 hold %10 transmit new bit
%01 load new bit %11 reset FCS generator
frincr : Asserted during *write1* in the last symbol of the AC field of a transmitted frame.
in : 8 bits tri-state input bus towards the byte-processor.
txdata : 2 bits data bus containing the symbol for transmission.
xstat : 3 bits bus that indicates and signals the status of the transmission.
bit0..2 : %00 no transmission (zero's)
: %01 frames in transmission
: %10 token in transmission
: %11 transmission aborted
bit3 : Asserted during *write1* in the last symbol of a token.

3.10 TXFCS

This schematic is used for the generation of the FCS field for the transmitted frames. The schematic uses one operator and one register. The functioning of the schematic is controlled by TXDATA. This schematic is a slightly modified version of the FCS generator of the HDLC/SDLC bit-processor.

INPUT

data : The transmitted data from TXDATA.
fcs_ctrl : 2 bits data and control bus from TXDATA.
reset : The reset signal.

PROCESS

Operator : CRCGEN
This straightforward operator is used to calculate the FCS. The calculation is performed with the help of one 32 bits register. When the control bus indicates to transmit the result, the contents of register CRCREG are transmitted on fcsbit.

Register : GEN
A 32 bits register used to perform the FCS calculation.

OUTPUT

fcsbit : The FCS bit to be transmitted.

3.11 MANCHENC

The schematic MANCHENC transforms the two bits input data into a differential Manchester coded bit stream. MANCHENC determines also the *write1* signal. One operator and one register are used.

INPUT

data : The data to be coded and transmitted, from TXDATA.
read1 : Clcoks signal from MANCHDEC.
tx : Asserted when the station is transmitting its own data, from PRIORITIES.
write : Clock signal from RINGIF.

PROCESS

Operator : MANCHENC
A very short and straight forward operator is used to encode the data and to generate the clock signal *write1*.
During the time the TRC is repeating data, the FORCE_TX bit on *bitc* and input *tx* equal zero, *write1* equals *read1*. In this situation *write1* can also be asserted in two successive bits. This is caused by the loss or insertion of one bit, in the receive process.
When the station starts to transmit its own data, the assertion of *write1* in two successive bits will corrupt the transmitted frame or token. Therefore the schematic takes over the generation of *write1* when the TRC transmits it's own data, with the use of bit2 of the register PREV.

Register : PREV
A two bits register is used for encoding the data, and to generate *write1*. This register contains the following information.

bit0 : The previous bit put on *bit_out*.
bit1 : Is set when the first bit of a symbol is encoded. The value of this bit is controlled by *read1* and/or *write*.

OUTPUT

bit_out : The encoded output bit stream.

3.12 COUNTERS

This schematic contains the Line Error, Burst Error, Lost Frame Error, Abort Delimiter Transmitted and Frame Count counters. These counters, with exception of the Frame Counter, are accessible through the register space of the byte-processor. The Counter register and a part of the Command register appear in this schematic. Chapter 5.2 specifies the controlling of the counters.
All counters are 8 bits wide. Overflow indication is given when the most significant bit becomes set. The dummy read for the modify operation on the Command register is implemented in this schematic.

INPUT

ab : The Abort Delimiter Transmitted signal from TXDATA.
berr : The Burst Error signal from BURSTBUFFER.
ctrl : 7 bits address and control bus from byte-processor.
ed : ED received signal from MANCHDEC.
fr_r : Frame Counter reset signal from PRIORITIES.
frincr : Frame Counter increment signal from PRIORITIES.
lerr : Line Error signal from RXFILTER.
lfrerr : Lost Frame Error signal from PRIORITIS.
out : 8 bits output bus from the byte-processor.
reset : The reset signal.

PROCESS

Operator : REG_CTRL
The operator REG_CTRL implements the functioning of a part of the Command register and the Counter register. The operator contains three functions, of which one is selected through the control connector connected to *ctrl*.

Function 1 : COMMAND ctrl := %X100000
This function recognizes if one of the five READ commands is given. If one of READ commands is give, bit4..6 are stored in register CNT, with one exception. Zero is loaded into the register when the READ_CNTS command is given.

Function 2 : READ ctrl := %1000010
This function is activated when the byte-processor wants to read the Counter register. What information is put on in, is dependent on the value in register CNT, hence dependent on the last READ command given. The function signals on the four bits wide *cnt_set* bus, which counter is read, and thus reset.

Function 3 : NOP
If non of the above functions is selected, the function NOP performs nothing special. Important is the fact that the function puts zero on the tri-state bus. This is used for the dummy read when a modify operation is performed on the Command register.

The operator also drives the *in* bus when *ctrl* = %1X00010.

Operator : COUNT_CTRL
This function controls the five error counters. The appropriate counter is incremented when *berr*, *lerr*, *lfrerr* or *ab* is asserted. On the other hand, the function resets the appropriate counter when the accessory bit on *cnt_set* is asserted. *Cnt_err* is asserted when the most significant bit of one of the counters is set.
Besides the five error counters, the operator also controls the Frame Count counter.

Register : ABORT_ERROR
Abort Delimiter Transmitted counter.

Register : BURST_ERROR
Burst error counter.

Register : COUNTER
This three bits register identifies which READ command was last given.

Register : FRAME_CNT
Frame Count counter.

Register : LINE_ERROR
Line Error counter.

Register : LOST_FRAMES
Lost Frame Error counter.

OUTPUT

cnt_err : Asserted when the eight bit of one of the counters is set.
in : 8 bits tri-state input bus towards the byte-processor. This bus is driven by the REG_CTRL operator when *ctrl* equals %1X00010.

3.13 FLAGS

The schematic FLAGS implements the I and SFS flag, and the Timer Return to Repeat. The Initial TRR register and a part of the Command register appear in this schematic. The functioning of the flags is implemented in one operator and one register. The TRR uses one operator and two register.

INPUT

clk : The Timer Return to Repeat increments when this input is asserted.
i_r : The I flag is reset when this input is asserted.
i_s : The I flag is set when this input is asserted.
sfs_r : The SFS flag is reset when this input is asserted.
sfs_s : The SFS flag is set when this input is asserted.

PROCESS

Operator : SETRESET
One simple operator with one function controls the two flags. When the set and reset signal of a flag are asserted at the same time, the flag is set.

Register : FLAGS
Flags is the two bits register that contains the two flags.
bit0 : I flag.
bit1 : SFS flag.

Operator : TRR_CTRL
The operator recognizes when the byte-processor writes the RESET_TRR function into the Command register. When the operator does so, by simply looking to the *ctrl* and *out* bus, it resets the TRR.
Also the operator controls the register val containing the time-out value of the TRR. The operator puts this register in the load mode when the byte-processor writes to the Initial TRR register, ctrl = %X100010.
When the TRR, register TRR, reaches the value in VAL, the operator stops increasing the TRR, and asserts trr until the timer is reset.

Register : TRR_VAL
This register contains the time-out value of the TRR. VAL is the implementation of the Initial TRR register.

Register : TRR_CNT
This is the actual timer TRR.

OUTPUT

i : Asserted when the I flag is set.
sfs : Asserted when the SFS flag is set.
trr : Asserted when the TRR has expired.

3.14 RESET

This simple operator asserts the *reset* signal when the RESET_ALL command is written to the command register.