Structured design of an FDDI protocol handler

Simons, P.W.

Award date:
1995

Structured Design of an FDDI Protocol Handler

P.W. Simons

Department of Electrical Engineering
Digital Systems Group
Eindhoven University of Technology

Eindhoven, August 1991

Master's Thesis

Supervisor and Coach: Prof. Ir. M.P.J. Stevens

Abstract

For high speed LANs several standards are defined. One of these is the Fiber Distributed Data Interface (FDDI). This operates at a data rate of 100 Mbps via a fiber.

The standard is defined by the ANSI X3T9.5 group, and is described in four standards: Physical Media Dependent (PMD), PHYsical layer protocol (PHY), Media Access Control (MAC) and Station Management (SMT). This standard only supported packet switching, with a timed token ring protocol and several error detecting and correcting mechanisms.

Later the need for circuit switching was recognized and an additional standard is developed: Hybrid Ring Control (HRC). With this extension both circuit and packet switching will be possible. Further development includes lowering the installing costs (low cost fiber and twisted pair for transmission to the end user), and increasing the speed.

For the basic FDDI protocol a protocol handler has to be designed. This takes a long time, so only a part will be designed, and the rest will be left to other people.

The ANSI standards on FDDI allow for different implementations. The description given is functional, giving services between two entities. The internal behaviour is shortly described, with a state machine, or a block diagram.

To model the protocol the method as described in "Strategies for real-time system specification" by D.J. Hatley and I.A. Pirbhai is used. This is supported via the program PROMOD, and with this program the FDDI protocol is modelled, emphasizing on the Media Access Control entity. After this step, data and control flow diagrams are available, with process specifications. The model is constructed via discussions about the functional division, and the communications between the processes. Because no valid standard on the Station Management was available, this entity is only slightly watched at.

The diagrams and specifications allow for implementation in hardware. For this purpose the program IDaSS (Interactive Design and Simulation System) is used, developed at the Eindhoven University of Technology by Ir. A.C. Verschueren.

The encode and decode functions of the Physical layer protocol are described in IDaSS, as well as the way to obtain data from a user and construct frames of it. Therefore a memory management system and a frame check sequence generator, working with four bits parallel, are developed. While designing, minor changes had to be made to the model, because of the different point of view to the system.

In the report of Mr. P.W.A.J.M. Nas more designs can be found, that have been developed independently from the designs described in this report.

From these facts it appears that the requirements model is satisfying in the total design trajectory.

According to people in the industry, FDDI has many possible applications, but is not useful in many cases, because most users do not need all possibilities offered by packet switched FDDI.

With the development of FDDI, more general services become available (especially circuit switching). Combined with lower prices, this makes FDDI more useful for more and more users. Therefore it may be recommendable to continue the development of the protocol handler, or at least to keep an eye on the subject.

Note: The design process has initially been carried out by two persons (the writer of this report and Mr. P.W.A.J.M. Nas). This ended up in two reports (with the same name) that are written for the greater part in cooperation. This means that a great deal of the chapters in this report can also be found in the other one. Only the parts describing the step to hardware contain several different items.

Acknowledgements

I wish to acknowledge the following companies and persons for the information I and my companion received about their FDDI products or involvements:

AMP Inc., Jack Himes	Harrisburg	USA
CHeSS, Dhr. van Teijlingen	Soest	Netherlands
CMC	Santa Barbara	USA
CMC Ltd., David Griffiths	Hounslow	England
Digital Technology Inc.	Dayton	USA
Digital Technology	Berks	England
Digital Equipment Corp.	Maynard	USA
Digital Equipment B.V.	Utrecht	Netherlands
Fibercom	Roanoke	USA
IBM, Dr. Richard O. LaMaire	Armonk	USA
Millegan, Gene	Oklahoma City	USA
Network Systems Corporation	Minneapolis	USA
Plessey Semiconductors Ltd.	Swindon	England
PTT Telecom, Hans Herlaar	Den Haag	Netherlands
Siemens Nederland N.V.	Den Haag	Netherlands
Sun Microsystems Inc.	Mountain View	USA
Sun Microsystems Nederland B.V.	Amersfoort	Netherlands
Vitel Communicatie B.V., A Sterk	Gouda	Netherlands

I specialy want to acknowledge Mr. Albert Sterk of Vitel Communicatie B.V. for his invitation of me and my companion to discuss FDDI. I thank Advanced Micro Devices and CMC for their generosity of sending useful databooks.

I am pleased to acknowledge Professor Ir. M.P.J. Stevens for his advice and his support.

Also I would like to thank Mr. Peter Nas for being my companion and for his helpful and pleasant cooperation.

Finally I want to thank Miss Rian van Gaalen for her patience with me and my companion, her advice and her willingness to assist us with many different problems.

Contents

List of figures

List of tables

1. Introduction

From history, demands for faster and faster communication channels have come up. This is because of the more intensive use of databases, and distributed processing.

To satisfy these increasing demands, several high speed networks are defined, such as ethernet, and IEEE 802 networks. There are different configurations of networks like star and ring. Also the speeds are various, from 1 Mbps to several 10s of Mbps.

Because of the relative poor accessability of the file servers, a high speed network had to be developed. This resulted in FDDI: The Fiber Distributed Data Interface. It was first developed as a back-end network, to connect large computers with disk and tape controllers. Later FDDI turned out to be satisfying for other kinds of use and the FDDI now also is used for backbone and front-end network.

Since the protocol is rather complicated (frame check sequences must be made, frames must be constructed), and has to be executed real time at high speed (the bit rate on the fiber is 125 Mbps), a satisfying software implementation is very hard to achieve. It can though better (and cheaper ?) be implemented in hardware.

To design this complicated hardware, a structured design method is required, which supports modular design, so that several people can work at the total project, while they have nothing to do with each other but on the boundaries.

Note: The design process has initially been carried out by two persons (the writer of this report and Mr. P.W.A.J.M. Nas). This ended up in two reports (with the same name) that are written for the greater part in cooperation. This means that a great deal of the chapters in this report can also be found in the other one. Only the parts describing the step to hardware contain several different items.

2. What is FDDI?

FDDI stands for Fiber Distributed Data Interface, and it defines the physical layer and the lower part of the data link layer of the ISO/OSI model, and further a station management section which operates in both parts of the ISO/OSI model. The physical layer is divided into two parts namely the Physical Medium Dependent layer (PMD) and the Physical Layer Protocol (PHY). The lower part of the Data Link Layer (DLL) is called Media Access Control (MAC). The Station Management section is referenced as SMT. The data speed of FDDI is 100 Mbps, though because of errors [9] and delays, this speed will not be reached. Further more FDDI uses a timed token ring protocol: The station that captures the token is allowed to transmit during a predefined time and other stations are not allowed.

2.1 Possible use of FDDI

FDDI can be used for different purposes [10, 11, 21]:

- Back-end: This is what FDDI originally was designed for. The request for high speed connections came up with distributed computing and requests for faster data access. Large computers are connected with each other, and with disk and tape controllers. Since FDDI has a large bandwidth and a good fault tolerancy, this kind of use can be easily satisfied, and since there is only a small number of nodes in back-end applications, much of this bandwidth is used for data transport.

- Backbone: Various LANS are connected with each other and to mainframes and file servers, via FDDI. Different methods of connecting are possible like bridges, routers and gateways. Also long distances can be spanned, since the maximum distance between nodes is 2 km and the total ring size can be as long as 100 km. This allows the network to span a large company or university campus. An example is given in [34], where the FDDI technology is scanned for the usability for connection of ethernets.

- Front-end: Work stations and PCs are connected to the ring. Big jobs can be transported to mainframes (if connected), all stations can access common databases or programs on file servers. Since there is a large bandwidth available, many nodes can be connected (which caused trouble with other kinds of LANs, since their performance decreased heavily through the large number of nodes), and large databases can be accessed. It is expected that this kind of use of FDDI will be the most common in the future.

2.2 Definitions

T_Neg: Negotiated Token Rotation Time
: The time that a stations wants to be half of the maximum response time, and is put on claim frames (varies per station).

TTRT: Target Token Rotation Time
: The time that is wanted by some station to be available for synchronous bandwidth in one block (the same for each station).

T_Opr: Operational Token Rotation Time
: The minimum time that is negotiated between all stations to be available for synchronous transmissions

TRT: Token Rotation Timer
: Timer which controls the passed time since the last time the token was captured or passed. When TRT elapses one time before the token has returned, the token is late, and TRT is reloaded with TTRT.

THT: Token Holding Timer
Timer which is loaded with the remainder of TRT when the token is captured, only when the token is not late (during this time asynchronous transmission is allowed).

2.3 Specifications of FDDI

More detailed information can be found in [12, 27, 30].

The ANSI X3T9 group has defined four documents for FDDI, one for each part of the total definition: PMD for the physical medium dependent layer, PHY for the physical layer protocol, MAC for the medium access control and SMT for the station management. These documents are ented on the ISO/OSI model (Open Systems Interconnection from the International Standardization Organization), see figure 2.1. The Logical Link Control (LLC) is not part of FDDI.

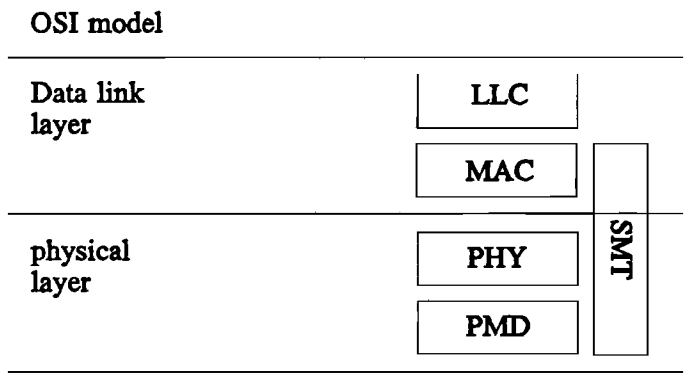


figure 2.1, Relation between the ISO/OSI model and FDDI [35]

Further there are two kinds of stations possible: DAS (Dual Attachment Station) and SAS (Single Attachment Station).

The main connection is made by a dual counter rotating ring. As long as no errors occur, both rings can be used for data transport, though only DASs can attach to the secondary ring. All DASs are connected to both rings. Further a concentrator is defined through which SASs can be connected to the ring.

Physical Medium Dependent layer (PMD)

For the links multi mode fibers are used, and transmission goes with 1300 nm light. The fiber links may not exceed the length of 2 km, and the attenuation of the cable may not be more than 2.5 dB/km. In this way the links can be operated at with relative cheap material (LEDs and receiving diodes), and still have an acceptable rise time at the fiber exit of maximum 5 ns. The transmitted power has to be at least -20 dBm and at most -14 dBm. The input sensitivity of the receiver should be more than -31 dBm, and less than -14 dBm [35].

The Bit Error Rate (BER) of a link at minimum received power should be less than 2.5 10^{-10} and less than 10^{-12} when the received power is 2 dB above the minimum, resulting in an overall BER of 10^{-9}. With the above specifications these requirements can be satisfied.

Also a duplex connector is defined, for use at the station bulkheads. This connector may have as much attenuation that the overall attenuation is not more than 11 dB.

Further an optical bypass switch is defined, by which the rings can be maintained while a station is powered off or misbehaving. This bypass may not exceed an attenuation of 2.5 dB.

Physical layer protocol (PHY)

The PHY defines a full duplex physical connection to the optical hardware. This means that it is possible to transmit and receive at the same time.

The PHY defines further the encoding of the data bits. This is an 4B/5B encoding and it is used for some error detection. From the 32 possibilities, 8 are not allowed, and 8 are used for extra symbols, while the other 16 are codes for the incoming data symbols. Further the NRZI scheme is used (Non-Return-to-Zero-Invert-on-ones). In this encoding no more than three zeros may follow each other, which is necessary for clock recovery.

The clocking is done locally. Each station has its own transmission clock, so slight differences in timing between the different stations will occur. That is why local clock recovery must take place, and an elasticity buffer has to be used to adjust for the differences.

A smoother is defined for maintaining the inter frame gap on about 12 preamble symbols.

Another entity defined for the PHY is that it provides line states to the adjacent stations.

Media Access Controller (MAC)

The MAC controls the data flow, and provides a connection point for devices that want to use the ring.

There are two kinds of traffic defined:
- synchronous traffic: This uses guaranteed bandwidth and response time, the allocation of the bandwidth is done by the SMT at initialization of the ring.
- asynchronous traffic: This is used for less predictable traffic, and only the bandwidth that is not used for synchronous traffic can be used for this. It may cause a long response time. For this traffic a priority level can be used.

The FDDI uses a timed token protocol which means that the token must be back at the same station in a certain amount of time (a function of the Token Rotation Time TRT).

The data is divided in pieces and transmitted in data frames (see figure 2.2).

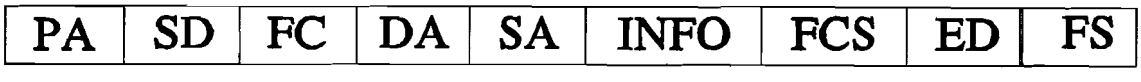


figure 2.2, Format of a data frame [30]

The meaning of the different parts is as follows:
- PA: Preamble, different number of Idle symbols
- SD: Starting delimiter, uniquely decodable sequence of a J and a K symbol (each a sequence of five bits on the ring)
- FC: Frame Control, defines the type of the frame (2 symbols)
- DA: Destination Address, 16 or 48 bits (also addressing a group of stations is possible)
- SA: Source Address, 16 or 48 bits (the frame is stripped from the ring when a station sees that the SA is his own address)
- INFO: Contains the data, length ranges from 256 to 9000 symbols
- FCS: Frame Check Sequence, 32 bit Cyclic Redundancy Check (CRC)
- ED: Ending delimiter (1 symbol)
- FS: Frame Status, these are the EAC symbols (Error, Address recognized, data Copied), each has the value S or R

Other frame types defined by ANSI are:
- Media Access Control (MAC) frames (15):
  - 1 beacon frame
  - 1 claim frame
  - 1 Next Station Addressing (NSA) frame
  - 12 reserved frame
- Station Management (SMT) frames (15)
- Logical Link Control (LLC) frames (32):
  - 1 synchronous frame
  - 8 asynchronous frames
  - 23 reserved frames
- Implementer frames (32):
  - 8 synchronous frames
  - 8 asynchronous frames
  - 16 reserved frames
- 32 additional reserved frames

Station Management (SMT)

The station management is a defined part of FDDI, in contrary with many other network definitions. This has the advantage that all implementations of FDDI can communicate with each other.

The SMT takes care of fault recovery, connection management, frame handling, synchronous bandwidth management, statistics gathering and communication with other SMTs via SMT frames. This is all done to maintain the operability of the ring.

The SMT uses an N-layer protocol which means that it covers more than one layer (layer 1 and 2). This may not interfere with other-layer protocols, which is achieved through the use of different types of frames: SMT and LLC frames.

2.4 Operation of FDDI

More information about the operation can be found in [10, 22, 27, 32].

Before the ring can be used, the characteristics must be determined through an initialization. First a beacon process is performed. This is done to determine the configuration of the ring, and to check the correct operation of the links between two stations. Each station sends it own beacon (with addressing

information) repeatedly until another beacon is received. Then the incoming beacons are echoed (probably read to obtain a ring configuration) until the beaconing station receives its own beacon.

Then a bidding process is performed. This is to determine the Target Token Rotation Time (TTRT) of the total ring, and is done via claim frames containing a T_Neg (negotiation time). On basis of this all station managements can determine how many bandwidth they can use. The lowest T_Neg is used for the TTRT. This is done because the station bidding that time, **needs** this low response time. The response time is in this protocol always shorter than 2*TTRT. After the bidding process, the winning station (with the lowest T_Neg) has to send a token on the ring. When this reaches a station this retransmits the token and it knows that the ring is operating. The next time the token arrives at the station, it can capture the token and start transmitting. When the token comes back at the originating station, then the ring is totally initialized, and operational. All stations have copied the lowest T_Neg in the TTRT register (Target Token Rotation Time).

A station is only allowed to transmit, when it has captured the token. When the token arrives early (TRT<TTRT), both synchronous and asynchronous frames may be transmitted, else (TRT>TTRT) only synchronous frames may be transmitted. Each time the token arrives, THT (Token Holding Timer) is determined by TTRT-TRT when the token is not late (earlier than 1*TTRT). During THT the asynchronous frames may be transmitted.

If synchronous frames are available for transmission, then they have priority above asynchronous frames, and whether or not there are asynchronous frames to be transmitted while the token is early, synchronous frames will then be transmitted. Within the asynchronous frames are 8 priority levels available.

When asynchronous frames have been transmitted and the THT has elapsed then there is still bandwidth available for synchronous transmission. This can be done during a predetermined (at initialization) period of time. This time is a part of the TTRT, and the sum of the synchronous transmission bandwidth of all stations is TTRT. This protocol assures that the maximum response time will be 2*TTRT.

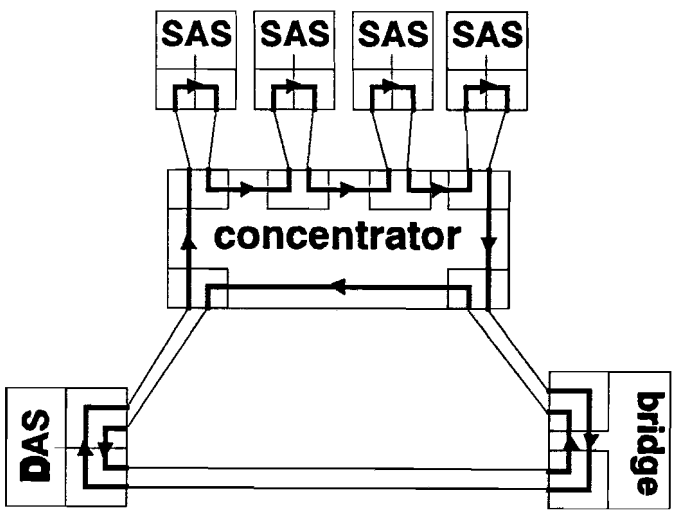


figure 2.3, A possible ring configuration for FDDI

In figure 2.3 an FDDI ring is drawn. An FDDI ring consists of two counter rotating rings, which can be connected to DASs. Via a concentrator, the cheaper SASs can be connected (there are also single attachment concentrators, but these are not drawn in figure 2.3). Connections to another network can be made via a bridge.

Data can be transmitted over both rings, but the secondary ring is meant for backup, and error tolerancy. Each ring can have 100 Mbps, so when no errors occur, the total bit rate can be 200 Mbps.

2.5 Error recovery in FDDI

More information can be found in [9, 10, 12, 16, 23].

When 2*TTRT has elapsed, the station determines it as an error, and starts transmitting claim-token frames to try to regain the token on the ring. When no response comes to this process, the station starts transmitting beacon frames containing his address information. When a station receives a beacon frame, he starts transmitting that frame, in stead of his own. When one station does not receive anything, he knows that his neighbour is bad functioning or not connected anymore (due to cable break), and starts reconfiguration.

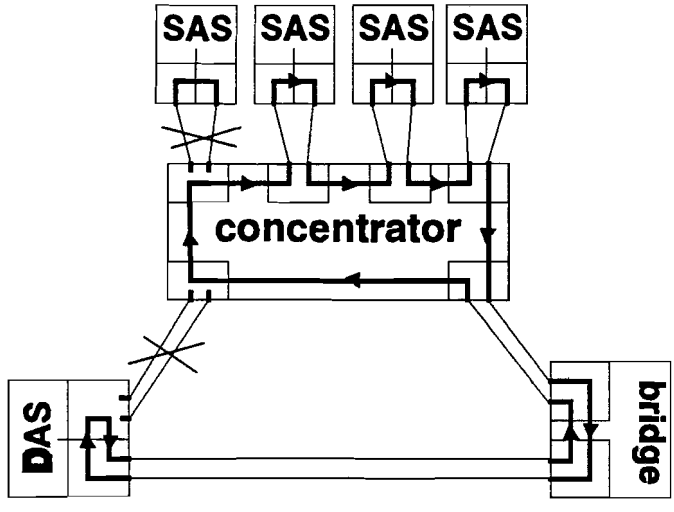


figure 2.4, FDDI ring with some cable breaks

In case of a cable break, the ring has to be reconfigured (see figure 2.4). When a cable in the dual ring breaks, then the secondary ring is put in action (like is shown for the DAS station). When a cable breaks in the connection to a SAS, then the concentrator bypasses this station. When a station shows bad behaviour, then this error can be modelled as a cable break, but if it is a DAS station, then this station can bypass itself, and go of the ring in this way, still leaving a dual ring to tolerate errors in the cables somewhere else. With this bypassing, an internal loopback is made from transmitter to receiver, so the station can check itself on existing errors. If this bypassing is not possible (when the station does not know that it is erroneous), the error will be treated like a cable break.

2.6 Concentrators in FDDI

For more information see [17,18].

A concentrator is a device that provides connection points for other devices in the form of a physical tree topology. The concentrator maps the logical token ring into a hierarchical physical tree. It is an active device, in its simplest form it is a multi-port physical layer repeater. It is called active for the concentrator decodes, re-times and sometimes modifies the repeated data stream.

The concentrator, CON, is an opto-electronic device that requires a power source. It implements the PHY as found in all FDDI stations, including data re-timing, elasticity buffers and error detection. Additionally the CON includes most of the SMT protocols for auto-initialization, auto-configuration and fault isolation within an FDDI LAN.

The concentrator serves three major goals:
- allows more flexible application of FDDI to the structured cable plants.
- an important tool to maintain a large LAN.
- the concentrator isolates the FDDI ring from station failures and end user behaviour, it improves the reliability.

There are three basic behaviours within FDDI:

- The Dual Attachment behaviour forms the physical loop of the Dual Ring. The basic Dual Attachment Station (DAS) requires two attachment points, designated as connector type A en B. A DAS has two PMDs, see figure 2.5, one connected to the primary input (PI) and secondary output (SO), and another connected to the PO and SI. A DAS has at least one MAC and one SMT entity. There is an optional MAC that allows the DAS to support the 200 Mb/s rate. When a network is working properly the A (say PO,SI) of one station is connected with the B (say SO,PI) of its neighbour. The result is a physical loop of Physical Connections that may provide two separate, counter rotating token paths.

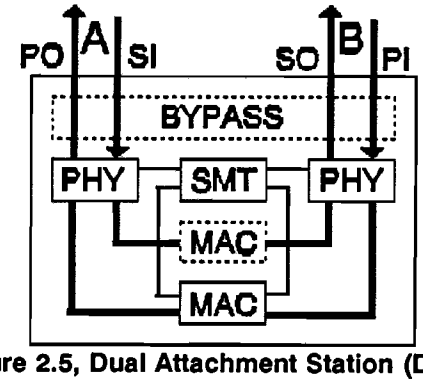


figure 2.5, Dual Attachment Station (DAS)

- The CON behaviour forms a tree by concentrating the inputs of "connector type Ms" from multiple attachments. Stations connected to the concentrator forms a single token path for this station. The basic concentrator is a stand alone device and cannot be connected to other concentrators. There are two types of concentrators, the Dual Attachment (DAC), see figure 2.6, and the Single Attachment (SAC), see figure 2.7. The DAC has two PMDs and PHYs to attach to the dual ring and multiple PMD and PHY for interconnecting other stations. It has at least one MAC and one SMT entity. There is an optional MAC to support two active rings. In this case, a DAC can selectively attach stations to either ring. A SAC has

one PMD, one PHY one MAC and one SMT entity and multiple PMD and PHY for interconnecting other stations.

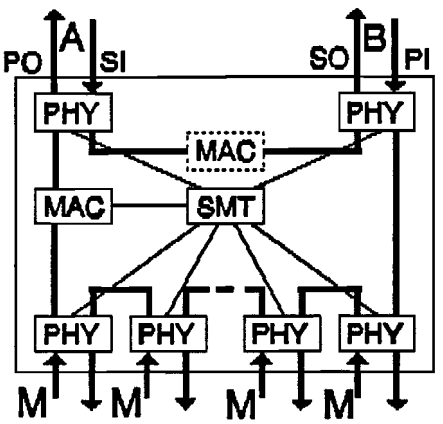


figure 2.6, Dual Attachment Concentrator (DAC)

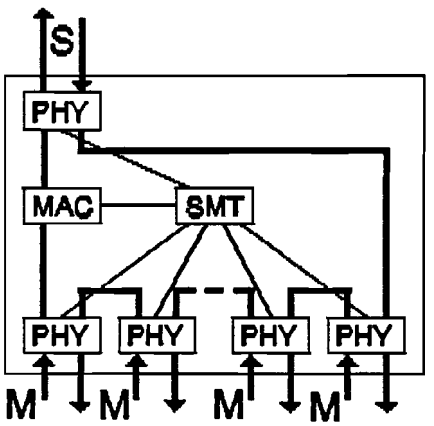


figure 2.7, Single Attachment Concentrator (SAC)

- The Single Attach behaviour, this is the simplest behaviour. The topology impact of the Single Attachment Station (SAS), see figure 2.8, is summed up by "Presence" and "Absence". A SAS has one PMD, one PHY, one MAC and one SMT entity. The SAS occupies the leaf positions in the tree. It is designated by the connector type S and is intended to provide attachment to a single token ring. SAS attaches to the dual ring through concentrators (either SAC or DAC).

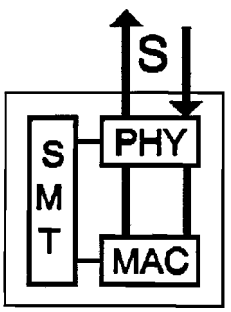


figure 2.8, Single Attachment Station (SAS)

A branching tree is formed using variations of the basic concentrator, that includes A,B or S type connectors. Connections of type S and M are primary used in pairs such that a Physical Connection would have one endpoint of each type.

Dual Rings comprise DASs configured in a physical loop, Trees comprise Concentrators and their attachments. The combination of the Dual Rings and the Trees result from the combination of the Concentrator with the Dual Attachment behaviour. The Dual Attachment Concentrator (DAC) includes an A and B connector and may be a member of the dual ring. The DAC may also extend the Tree as shown in figure 2.9. FDDI also specifies a Single Attachment Concentrator (SAC) which includes an S connector to allow its connection to other CONs in the Tree.

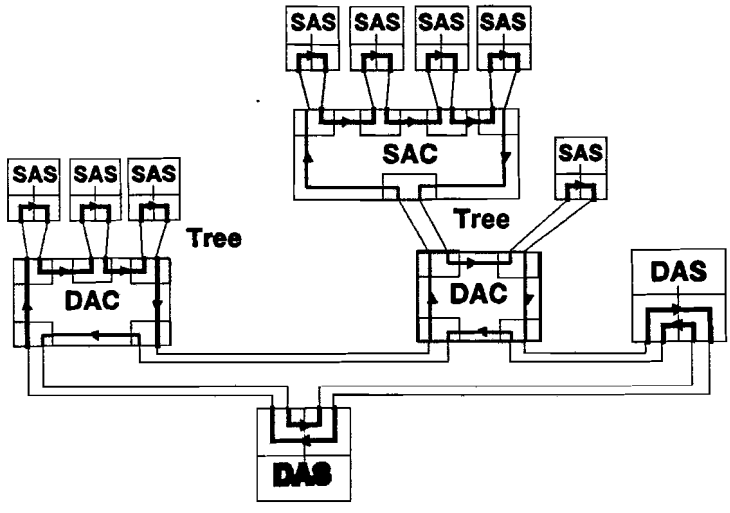


figure 2.9, FDDI dual ring of trees

The actual usage of a DAS and SAS is not limited to the basic behaviour just described. For instance: a DAS may be placed in a Tree and act as a SAS. Otherwise a SAS may be placed directly in a Dual Ring. But note that they are not interchangeable. For instance placing a SAS in the Dual Ring prevents formation of a physical loop, a basic behaviour of the Dual Ring.

Link types and topologies

There are three types of ports on a physical connection defined in FDDI: Peer, master and slave. Peer-to-peer connections are used to form a dual counter-rotating ring between DAS and DAC. Master-slave connections are used to connect DAC and SAC (both master) to SAS (slave). In an FDDI network DAS, SAS, DAC and SAC can be interconnected in peer-to-peer and master-slave modes to form a dual ring of trees. Peer-to-peer connections (DAS and DAC) form a counter-rotating ring at the top of the hierarchy. Master-slave connections (DAC, SAC and SAS) form a tree, as seen in figure 2.9.

The functional model of a concentrator

The concentrator and the DAS employ fundamentally different methods of controlling the network topology. One difference is the method of bypassing a connection and an adjacent station. The DAS involves "wrapping" and optionally the use of "optical bypass relays". When one cable is used to connect two DASs, they form a single token ring, this is a simple connectivity. In this situation, just one logical

ring existing, the Dual Ring is said to be "wrapped". If two logical rings exist, the Dual Ring is said to be "unwrapped".

Every DAS contains **two** PHY entities (PEs). Imagine a ring with for instance four DASs. If a cable were removed, the DASs adjacent to the missing cable internally reconfigure to exclude the related PEs and form a single ring. Internal path switches are used. A DAS also encloses an optical bypass relay, this to allow a station to bypass itself (and maintain the physical loop) in the case of a power failure.

Optical bypass techniques provide that a limited number of stations may be bypassed. This number depends on several issues like distance between adjacent stations and the number of stations. These techniques are useful in small networks (a computer room for instance).

The method of controlling topology in a CON is fundamentally different. The "bypass facilities" are moved from the individual stations into the CON. And the internal path switches are replaced by the CONs path switch. In this situation the optical bypass relays do not enhance the operation of the network and thus are not used. While the CON must be powered up, any number of the stations may be disconnected without affecting the remainder. The latter can be very useful in some situations.

Conceptually the CON is composed of a PHY element taken from each of the DASs. So the DASs become SASs. The CON is the result of rearranging the components such that same number of optical transceivers and cables are required in either topology to support the same four users.

Functionally the CON is composed of many PHY elements and a path switch. The CON reconfigures the token path as a service for attachments. The main advantage of the CON is that the control of the topology is removed from the station and is centrally located in the CON. As a result the CON fits easily in the "star wiring" scheme.

With a CON the function of a station can be separated into two basic categories: "End Stations" for end-users of the LAN and "Network Devices" for providers of LAN services. The CON is a network device and the SAS is the End Station. The CON becomes the major interest of the network manager (administrator) and the SAS eliminates the role of the end users in determining the network topology, this is an important simplification.

FDDI installation with respect to reliability

The CON allows more flexible application of FDDI to cable plants that are designed to meet emerging standards for building wiring such as being developed by EIA TR 48.1 [19]. The CON eliminates the need to restrict cable lengths or cable plant insertion loss for optical bypass operation.

The CON decouples the network managers from the daily activities of end users and repair personnel. The CON helps to organize the wiring and provides ring-map information that closely matches the structure of the building. The usage of a CON in the equipment room increases the resolution of the fault isolation protocols to the granularity of the building equipment room structure. This reduces the Mean Time To Repair.

The CON makes it easier to implement a reliable FDDI Ring. The reliability will always improve if CONs are used to attach End Stations. The reliability of the Dual Ring may be degraded by dual attached end stations, unless special care is taken in the design of those stations.

For some specific configurations equations for the total failure rate [19] can be derived. Some of the issues that must be considered are difficult to control in a multi-vendor environment, especially for user End Stations. In general there is not a standard formula to determine the total failure rate.

2.7 FDDI-II

More information can be found in [15, 20, 26, 32].

Even when the standard for FDDI was not fully completed, ANSI started with the development of another FDDI standard, to satisfy the requests for voice communication. This is called the FDDI-II standard, and is described in the HRC (Hybrid Ring Control) document of ANSI.

The standard for FDDI-II is also ented on the ISO/OSI model as shown in figure 2.10. The I-MAC (Isochronous MAC) is connected to a circuit switch multiplexer, and the MAC is connected to the Logical Link Control (LLC) which is not a part of FDDI.

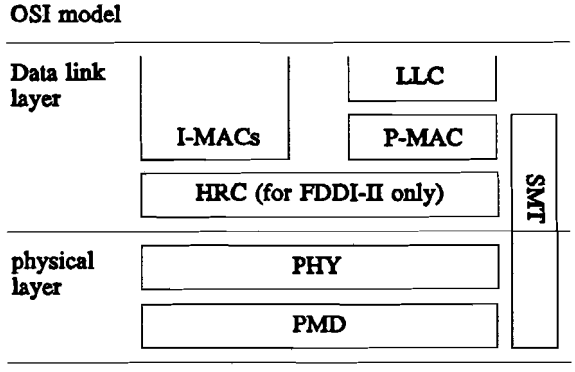


figure 2.10, Relation between the ISO/OSI model and FDDI-II

The PMD is for FDDI-II the same as for FDDI-I. PHY has one extra symbol to decode and has better smoothing requirements. The MAC part is extended with an I-MAC (for isochronous data) and the HRC. The original MAC is now called P-MAC (for packet data).

The station management is extended with some functions special for FDDI-II:
- Hybrid mode with initialization and recovery procedures.
- Management of HRC objects: HMUX, IMAC.
- Resource allocation, for requesting and assigning bandwidth.

FDDI-II is upward compatible with FDDI-I, so all stations supporting FDDI-I, can be connected to an FDDI-II ring. However, the total ring can then only operate in the FDDI-I mode.

FDDI-II supports for a network of up to 500 stations, over geographical distances of up to 100 km. The hybrid (FDDI-II) mode of operation, supports both packet and circuit switching.

There are two kinds of stations: Monitor stations and not-monitor stations. Only a monitor station can become a Cycle Master (CM). This is the station that initiates the hybrid mode of operation, and maintains it.

For voice transmission, a 64 kbps channel is enough, which means that every 125 μs one byte must be transmitted. Therefore the ring is divided in sections (cycles) returning each 125 μs. This is a kind of formatting of the ring, and is done by the cycle master. Only one cycle master is allowed on the ring. This cycle master has a latency buffer to make sure that a cycle starts always at a whole number of 125 μs.

For the data in the cycle is 98.304 Mbps available. This is divided over 96 Cyclic Groups (CGs). Each CG is divided in 16 timeslots, and each timeslot consists of 8 bits. This fully uses this bandwidth: 8 * 16 * 96 * 8000 = 98.304 Mbps

The bytes of one timeslot combined make a Wide Band Channel (WBC) of 6.144 Mbps. When several bytes of one timeslot are combined, it is possible to make sub-WBC connections. Through bit interleaving it is even possible to make sub 64 kbps channels. It is also possible to obtain larger bandwidths than what the WBC supplies, through the combination of several WBCs.

The structure of the cycle is shown in figure 2.11.

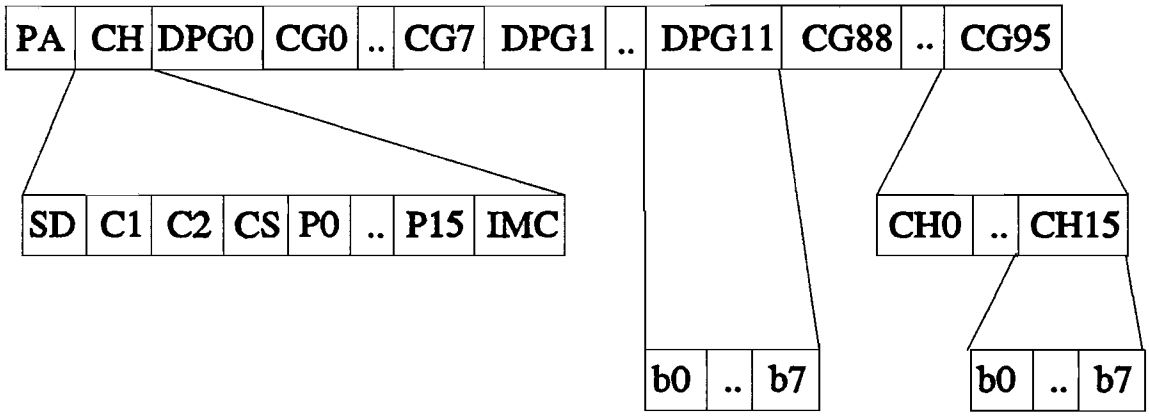


figure 2.11, Structure of the cycle in FDDI-II

Beside the CGs, there are three more parts in the cycle:

- Preamble (PA) to adjust the clocking differences between the different stations. Also in FDDI-II, each station has clock recovery from the incoming bitstream, and an own transmit clock.
- Cycle Header (CH), that carries the programming template. This defines the use (circuit or packet data) of the timeslots in each CG. Timeslot i in every CG has the same function, and when of all Cgs one timeslot is combined, a Wide Band Channel is made of 6.144 Mbps. The CH consists of:
  - Starting Delimiter (SD), built up from two symbols (J and K) which are uniquely determinable, anywhere in a cycle.
  - Two control symbols C1 and C2. Possible values are S or R. C1 is for synchronization control. If C1=S then synchronization is established, and a SD appearing in the middle of a cycle is seen as an error. When C1=R the cycle may be interrupted. When a synchronization error occurs a slave station puts this control to R. A slave station may never put this control to S.
    C2 is for sequence establishment. If the cycle sequence is established the cycle master puts this control to S, else to R. If a slave station determines an error, then he puts this control to R. A slave station may never put this control to S.
  - Cycle Sequence (CS). This is a modulo 192 number ranging from 64 to 255 to indicate whether the cycles are still in order. Lower numbers than 64 indicate a stations monitor rank (then C1=C2=R must be true). The value 0 indicates a monitor with rank NULL. This cannot win a monitor contention process, but still can initiate it.
  - Sixteen place programming template. These indicate wether a timeslot is allocated to circuit (S) or packet (R) data. If a place is not occupied with S or R then that place is filled with T.

- Isochronous Maintenance Channel (IMC). This is dedicated for maintenance purposes, and recommended use is a voice channel (there is 64 kbps available).
- Dedicated Packet data Group (DPG), that assures that there is some packet switching available, even when all timeslots are allocated for circuit data. This DPG can be used to request for circuit bandwidth, releasing bandwidth etc. This group is spread over the cycle. DPG i comes immediate before CG 8*i ($0 \le i \le 11$).

These three groups consume the last remaining 1.696 Mbps bandwidth that was not defined yet:

```
PA:                          20b
CH: SD:                       8b
    C1:                       4b
    C2:                       4b
    CS:                       8b
    P0 to P15: 16*4b =       64b
    IMC:                      8b
DPG: 12*8 =                  96b
                            ====
                            212b
```

8000 * 212 = 1.696 Mbps

FDDI-II also provides for DASs and SASs, and station bypassing (optionally, only for DASs). The timing is as in FDDI-I point to point, with clock recovery for the receiver, and an own transmit clock in each station.

Packet data can be transmitted over all WBCs that are not allocated for circuit data. That means that all WBCs, that are not allocated for isochronous (circuit switched) data, are linked together for packet data (else the response time for packet data would be much to low).

For the circuit switching at least one Channel Allocator (CA) is needed in the system. This processes the requests for isochronous bandwidth, so it tells stations requesting for a circuit which timeslot and CG they can use (for lower bandwidths also which bit of the timeslot). The CM can act as a channel allocator.

Packet data is transmitted in frames (just like in FDDI-I). The isochronous data is transmitted through byte interleaving (and bit interleaving for sub 64 kbps channels).

2.8 Operation of FDDI-II

Most of the information contained in this chapter comes from [20]. Further it is mentioned in [26] and [32].

Ring initialization

The ring initialization starts as with FDDI-I. When the ring is initialized for FDDI-I, this mode is used for a bidding process, to select a Cycle Master (CM). All monitor stations have a rank. Each capable station has an address, and the highest address (detected with the bidding) will be the cycle master (possibly stand-by cycle masters are selected for fast recovery when a cycle master fails).

The station management defines the maximum isochronous bandwidth, and tells that to the CM. This is determined by calculating the minimum packet bandwidth needed to satisfy the demands for response time (TTRT) for all synchronous applications of the system.

To switch from FDDI-I to hybrid (FDDI-II) mode, the CM must capture the token (then all transmissions of other stations have passed). The CM then starts transmitting cycles, with in the cycle header only allocations for packet switched data channels. The token is released in the **first** cycle, and is placed in the DPG. Each station recognizes the cycle header, and switches to hybrid mode. When the first valid cycle has returned to the CM, the ring is fully operating in hybrid mode.

Packet switching

Packet switching goes the same as in FDDI-I mode. The TTRT is as short as possible (this was already determined during FDDI-I setup). For the packet switching all available WBCs are combined to get the highest throughput for the packet switching. The symbols which were transmitted behind one another, now have to be interleaved because of the cycle assignment.

Circuit switching

A CA is given the responsibility or more circuit WBCs. When the CM receives a WBC request from a CA, he **must** capture the token (else packets can be lost since a packet channel is converted into a circuit channel). After that the new programming template is send to all stations. Only when the CM receives the template back correctly, he releases the token again. Then the identity of the WBC is send to the CA, and the CA informs all stations on the ring about the availability of the new bandwidth.

When a station needs a circuit, it requests (via an SMT protocol) bandwidth from a CA (possibly it must check several CAs). The CA confirms the request and gives the identity of the channel to the requesting station. The originating station calls the terminating station via the packet data channel.

After that full duplex communication can be performed, with each station replacing the data received by its own data.

When the communication is done, the channel is released by the **originating** station, via an SMT protocol to the CA.

2.9 Error recovery in FDDI-II

If an error occurs while in hybrid mode, the ring will try to recover that error in hybrid mode. If that is not possible, the ring will switch back to basic mode and start the basic recovery procedure. When that is achieved, an attempt can be made to switch back in hybrid mode. To detect errors, the same coding mechanisms are used as in basic mode.

2.10 Extension to the PMD

At first ANSI had defined a PMD which allowed for a distance between stations of maximum 2 km. This was for some purposes too short, and an extra definition for the PMD layer was developed.

This was called the SMF-PMD (Single Mode Fiber Physical Media Dependent), and may replace the original PMD. This uses laser diodes and single mode fiber for the transmission of the signal.

The SMF-PMD allows for a 60 km distance between two stations in stead of the previous 2 km, but it demands for an adjustment at the transmitter **and** the receiver side, since the received power will be much higher than with multi mode fiber. This will come out a little more expensive than with the normal PMD.

The communication with normal PMD stations can be maintained with the new standard by inserting an attenuator between the SMF-PMD transmitter, and the normal PMD receiver, while communication the other way will be normally possible.

3. Error detection

In a transmission system always occur errors, so error recovery is necessary. This can only be done when the error can be detected. For this purpose several mechanisms are employed.

One mechanism is the 4B/5B encoding. The used encoding is stated in table 3.1.

table 3.1: 4B/5B encoding scheme [9, 3]

Code-bits	symbol	semantics
Data symbols		
11110	0	0000
01001	1	0001
10100	2	0010
10101	3	0011
01010	4	0100
01011	5	0101
01110	6	0110
01111	7	0111
10010	8	1000
10011	9	1001
10110	A	1010
10111	B	1011
11010	C	1100
11011	D	1101
11100	E	1110
11101	F	1111
Line-state symbols		
00000	Q	Quiet
11111	I	Idle
00100	H	Halt
Delimiters and control		
11000	J	first of sequential SD pair
10001	K	second of sequential SD pair
01101	T	termination of data stream
00111	R	Reset
11001	S	Set
Invalid codes		
00001	VH	Patterns marked with V are invalid
00010	VH	Patterns marked with VH are invalid, but
00011	V	are treated like Halt when they are received
00101	V	
00110	V	
01000	VH	
01100	V	
10000	VH	

This encoding provides extra symbols, and is used to check the received data on transmission errors. Some of the symbols are not allowed (V or VH) and others are used for line states and special tasks so they cannot show up in a normal data block, unless an exceptional situation occurs.

3.1 Link Error Monitor (LEM)

To control the link status while that link is on-line, a Link Error Monitor (LEM) can be used [23]. This monitors the error rate of the connection between two stations and controls whether or not it exceeds a certain threshold level. If it does, the LEM will inform the SMT that can reconfigure the ring.

When the ring is reconfigured, the MAC can check the behaviour of the link with a Link Confidence Test (LCT). This tests the link between two stations while there is no other traffic on that link. This is possible

because with the reconfiguration, the excluded link (dual directed), is looped back. For this LCT at least one MAC is necessary.

The LEM works with several fault domains [23]:
- Optical link (PMD-PMD)
- Optical link and transmit and receive PHY function (PHY/PMD-PMD/PHY)
- Optical link, PHY, and transmitting MAC function (MAC/PHY/PMD-PMD/PHY)
- Internal noise on either side of the link.

Errors appearing from the optical link come from attenuation and dispersion, duty cycle distortion, jitter (data dependent, and random), and inter channel crosstalk. Further errors can occur due to internal station (electrical) noise.

The only information the LEM needs is that delivered by the PHY to the SMT (this costs nothing extra). When the LEM detects a degraded link, it is presented to the user who can take action. For this purpose, an administration must be kept of the history of the link and its behaviour. When the link is isolated, it can be repaired, and after that it should be tested by a LCT. When everything is working correctly, the link can be put back in action.

The LEM must be able to execute several functions:
- Detection of changes in the Link Error Rate (LER) link by link.
- Administration of the total occurrence of errors since the link was set up.
- Setting the link error threshold by the administration at which an alarm will be given.
- Detection and removal of links that show bad behaviour.
- Setting the link error threshold by the administration at which the link will be automatically removed.
- Possibility to reintroduction of a link, after a successful LCT.
- Detection of topology oscillations (link removal and insertions).

3.2 Frame Check Sequence (FCS)

The FCS field is used to detect erroneous data bits within the frame as well as erroneous addition or deletion of bits to the frame. It is accounted for each frame, and put behind the data. The fields covered by this FCS are FC, DA, SA, INFO and FCS. This allows the receiver to detect many errors, that could not be detected with the 4B/5B encoding (data to data transfer). This FCS is accounted with a polynomial, that is also used in IEEE 802 LANs and in Autodin II networks.

Definitions and a short explanation on the use of the generated polynomial code.

$F(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + a_{k-3}x^{k-3} + \ldots + a_1x + a_0$
F(x) represents k bits of the frame to be covered by the FCS. The first bit of the frame is represented by $a_{k-1}x^{k-1}$ ($a_{k-1} \neq 0$).

$L(x) = x^{31} + x^{30} + x^{29} + \ldots + x^2 + x + 1$
L(x) represents a 31 degree polynomial with all coefficients equal to one.

$G(x) = x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$
G(x) represents the actual generator polynomial

$C(x) = x^{31}+x^{30}+x^{26}+x^{25}+x^{24}+x^{18}+x^{15}+x^{14}+x^{12}+x^{11}+x^{10}+x^8+x^6+x^5+x^4+x^3+x+1$
C(x) represents a unique polynomial remainder produced by the receiver upon reception of an error-free sequence, i.e. $C(x) = x^{32}*L(x)/G(x)$

R(x) The remainder polynomial that is of degree less than 32

P(x) The remainder polynomial on the receive checking side that is of degree less than 32

Q(x) The greatest multiple of G(x) in $x^{32}*F(x)+x^k*L(x)$

$Q^*(x) = x^{32}*Q(x)$

M(x) The sequence that is transmitted
$M^*(x)$ The sequence that is received

FCS generation

FCS = L(x) + R(x) i.e. the one's complement of R(x)

$[x^{32}*F(x) + x^k*L(x)]/G(x) = Q(x) + R(x)/G(x)$

then $M(x) = x^{32}*F(x) + FCS$

FCS checking

The process of checking involves dividing the received sequence $M^*(x)$ by G(x) and testing the remainder. Because possible leading zeros does not yield a unique remainder, direct division is not possible. Thus a term L(x) is prepended to $M^*(x)$ before it is divided.

$x^{32}*[M^*(x) + x^k*L(x)]/G(x) = Q^*(x) + P(x)/G(x)$

In the absence of errors, the unique remainder is the remainder of the division

$$P(x)/G(x) = x^{32}*L(x)/G(x) = C(x)$$

So if errors occur $P(x)/G(x) \neq C(x)$

An example implementation of the above FCS circuit is shown in figure 3.1.

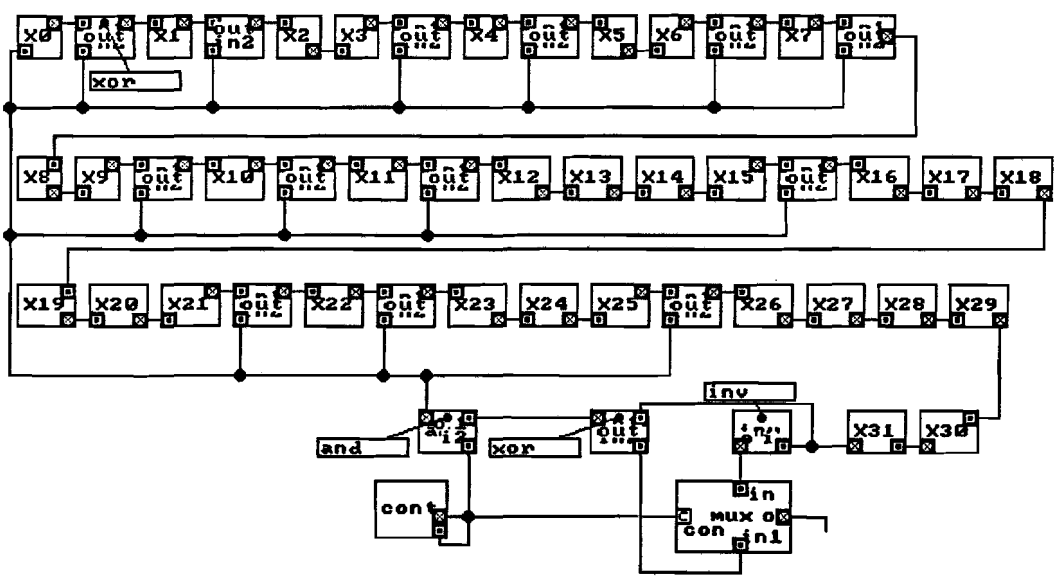


figure 3.1, An example implementation of a FCS circuit

4. FDDI developments

Unlike most new technologies FDDI seems to fulfil an immediate need of many users. Mostly a lag between the introduction and the acceptance of a new technology occurs because most users do not have an immediate need for it. New technologies of any kind are automatically suspected until they have proven their usefulness in the real world. One may discuss wether or not FDDI fulfils an immediate need of (how?) many users. But FDDI has garnered an enormous amount of acceptance at the user level (the most important level), and ANSI has not even finalized the complete standard yet.

One can distinguish three different phases of the FDDI usage territory, see chapter 2.1:
- Back-end: high speed point-to-point network, mainframes, file servers
- Backbone: networks, LANs
- Front-end: desktop, workstations

As the different FDDI usages grow, alternative physical media approaches that facilitate connection of FDDI to the desktop are getting careful consideration. Because lowering the cost of installing FDDI and providing alternate cable plan compatibility is the key to widespread applications. Users will benefit as more and more of the network power offered by FDDI becomes available to the current applications and a host of new applications.

Built-in management protocols and services designed into FDDI's station management (SMT) portion provide the highest level of network manageability of any LAN developed as a standard. With the ability to remotely configure FDDI stations, setting their operating parameters and policies regardless of equipment vendor, FDDI advances true multivendor network management. Users are ensured that all vendors must implement interoperable management for FDDI, as specified in the standard. This increases the users confidence that multi vendor equipment may be rellably integrated into their heterogenous networks, so that investments keep their value.

These two issues, lowering the costs and the growth in the confidence that the investments keep their value, predict a great future for a wide spread FDDI.

Many users first installing FDDI networks as backbones, interconnecting their lower-speed LAN subnetworks with bridges and routers to form global area networks. Most common first-generation networks, whether Ethernet or Token Ring, are increasingly star wired with either twisted pair, coax or fiber-optic cables to concentrators installed in wiring closets for erasure of management and control. Coupling these concentrated networks with FDDI lets users take some advantage of FDDI's speed and potential while preserving the investment they already made in their first-generation network.

More and more of today's local computer networks experience performance bottlenecks that occur much closer to the user, in many cases the problem is right at the desktop or server. Bringing FDDI as close as possible to these systems can provide network performance levels sufficiently higher than that achieved when FDDI is used only as a backbone. The growth of this problem is one of the factors that determine the success of FDDI. Only if one does really need the FDDI potentials one will consider to invest in FDDI.

As with all new technologies, cost remains higher than first-generation networks, but the increased volume, as manufacturers begin shipping more and more units, is already reducing costs and is sure to continue bringing them down.

Despite its inherent reliability, FDDI has some shortcomings that may hamper users and its deployment.

- FDDI's speed poses diagnostic and management challenges. With an FDDI ring running close to its 100 Mbps throughput, FDDI network analyzers have to be able to capture far more packets than with Ethernet or Token Ring, and they have to be able to do it much faster. Only few analyzers are able to capture all data in real time at such speeds.

- FDDI may be too robust for its own good. An FDDI network can recover from a noisy line condition or a lost token with only a few microseconds of downtime. Users may not even notice! Unfortunately if no one notices the ring going temporarily down, then no one knows a problem even existed until it is too late. When the users notice a problem on the FDDI, it is usually catastrophic.

- If any node on the FDDI ring goes bad, its downstream neighbour will automatically remove it from the network without notification, regardless of its function. Users accessing a critical host across an FDDI ring could suddenly find themselves with no link to the mainframe, if the mainframe-to-FDDI connection begins to have problems.

4.1 Managing an FDDI network

FDDI applications are far more complex than with other LAN topologies. Since it was initially deployed as a back-end technology. Managing an FDDI network not only means managing the backbone, but also keeping an eye on the other subnetworks. however, SMT, the network management portion of FDDI, is not compatible with any other industry-standard network management protocols, such as the Simple Network Management Protocol (SNMP) or OSI's Common Management Information Protocol (CMIP). Network managers need to be able to access both SMT and SNMP data to simultaneously manage their FDDI backbone and their Ethernet and Token Ring subnetworks, ideally all from a single screen.

SMT seems to pose as many problems as it solves, if only because the controversy surrounding its implementation is delaying the standard's finalization. The central issue is a portion of the SMT document called Parameter Management Frame (PMF), that is currently optional, but that many FDDI vendors would like to see made mandatory (except the ones that already have a streamlined implementation). While PMF adds performance tuning capabilities and is necessary to access large chunks of the FDDI Management Information Base (MIB).

SMT is not compatible with SNMP, the most popular network management protocol at this moment. While SMT provides detailed information about what is going on the FDDI ring, it does not cross network boundaries, like bridges or routers, it can not provide information about subnets. While SNMP allows users to manage an entire inter-network, it does not provide detailed management information for each FDDI node. With an SMT-to-SNMP proxy agent in the FDDI concentrator, all the stations on the FDDI ring could communicate their management information via SNMP to the network management console.

Dual-homing

One special network architecture is the dual homing architecture [37]. This architecture dictates that only concentrators should be connected to the dual ring and that all other devices should be attached to the concentrators. Essential devices like bridges or other cascade concentrators that need redundant links are connected to two separate concentrators.

Dual homing is preferable mainly from a management standpoint. Having to manage a single type of device, rather than several devices, streamlines network monitoring which is an important simplification with large networks. One of the primary advantages of having only concentrators connected directly to the dual ring is that it allows managers to define local rings. Network managers can section off the nodes attached to a single concentrator into a local ring that does not communicate with the primary ring.

With this local ring, managers can then repair faulty nodes without bringing the entire network down. This local ring also offer some performance benefits (large file transfers on one local ring instead of the

primary ring). And it also could be used to isolate a group of users with stringent security requirements from the rest of the LAN.

4.2 Media connection developments

FDDI is, by nature, fiber optic. Although lower-speed networks like Ethernet can be supported with 820 nm, the high data rates supported by FDDI require 1300 nm optics. Using this 1300 nm optics increases operating distance at FDDI speeds by minimizing signal dispersion caused by the fiber-optic cable's inherent group delay characteristics (which is nearly constant at 1300 nm while 820 nm is excessively variable). The use of multimode fiber (62.5/125 micron) and LEDs limits operating distance compared by singlemode fiber and lasers, but makes FDDI less expensive.
There are two major developments on this territory, one is the extending (altering) of the operating distance and the other is the decreasing of the connection costs.

Extending the operating distance

The 2 km operating distance specified by the original multimode PMD satisfies the connection requirements of many FDDI stations but is not sufficient for widely distributed systems. Another FDDI PMD standard (SMF-PMD), see chapter 2.10 and figure 4.1, extends FDDI's geographic reach, specifying how two FDDI stations can be separated by 60 km of cable without the use of an amplifier (or repeater). This standard works with lasers and singlemode fiber (0.5 micron) to minimize dispersion and increase operating distance.

The multimode and singlemode transceiver operate using the same 1300 nm wavelength light and thus can be mixed in the same network since they maintain optical compatibility.

Prior to the availability of FDDI concentrators the only FDDI connection alternative was to attach systems directly to the dual-ring trunk. Now that concentrators are available and nearly all users want to star their networks, alternative PMD specifications supporting reduced-distance horizontal connections between the wiring closet and end-stations are being investigated.

While the two existing PMD specifications are applicable for concentrator ports, surveys have indicated that more than 95% of the end-stations star wired to concentrators are less than 100 meters from the wiring closet. Because this is only 5% of the 2 km distance specified by the original multimode PMD, and because the media of choice for star-wired LAN installations is either twisted pair (coax) or fiber, opportunities exist for developing other FDDI PMDs.

Coaxial cable was eliminated as a potential media because it did not provide a good fit with the current standard. This is primarily because the connection management relies on a series of bit level signalling handshakes. This was specified with the assumption of the existence of a full duplex link. Coaxial cable is a simplex media and it would be a costly solution to double up the cabling requirement. If a single cable was to be used the electronics required to implement duplex signalling would also be to costly.

Two recently formed X3T9.5 working groups will devote their efforts to developing alternative PMD specifications for horizontal distribution of FDDI LANs. Because a great number of users' buildings and campuses is already wired with twisted-pair cabling, one group is exploring running FDDI over shielded or unshielded twisted-pair cabling for horizontal connections. The other group is determining how the cost of multimode fiber-optic PMD components (the optical transmitters and receivers, cables and connectors) can be reduced for horizontal connections.

Low-cost fiber PMD

The low-cost fiber group is working to develop a low-cost fiber PMD (LCF-PMD) with relaxed specifications for the optical transceivers and connectors to lower their manufacturing costs while maintaining optical compatibility. Maintaining optical compatibility means that the 1300 nm will continue to be used, even though power, receiver sensitivity and duplex connector specifications may change. This approach ensures flexibility, allowing systems with various PMDs to be connected, and the lowest total system cost.

Since the horizontal cabling from a wiring closet to most users' desks is substantially less than FDDI's original 2 km specifications, shortening the distance requirement should make it easier for manufacturers to reduce the costs of optical components. The FDDI PMD standard does not specify operating distance directly in meters of optical cable. Rather a loss budget describing the signal energy in decibels available between the optical transmitter output and the optical receiver input is specified. This loss budget can partially be consumed by all elements intervening in the cable plant connection between two active FDDI station interfaces.

Currently the multimode PMD specifies a link budget of 11 dB while link budgets of about 7 dB are being considered for the LCF-PMD, this should result in reliable operation at distances of at least 500 meters (2 km with the 11 dB link budget).

Twisted-pair PMD

The twisted-pair group is still unclear about which twisted-pair cable will be used. Unfortunately, nearly all installed twisted-pair cable plants are unshielded and there are many versions with many different specifications. Besides, operating unshielded twisted-pair at 125 MHz is substantially more difficult and therefore more costly than operating on shielded twisted-pair. So the unshielded twisted-pair (UTP) alternative is not as obvious as it may seem, looking at the already installed cable length.

Although some are advocating twisted pair as a lower cost alternative to fiber, this cost advantage is most likely achieved only on specific types of shielded twisted pair. However, many user sites require installation of new cable plants before they can support operation on (a specific) shielded twisted-pair. But, when users commit to the expense of a new cable plant, they almost always install a combination of fiber and UTP.

Although it is possible to support both shielded and UTP cable with one PMD interface, this solution costs will most likely be higher than a shielded-only solution. Clear statements of objectives, direction and decisions on the tough issues are required quickly for the twisted-pair/PMD development effort to be successful.

4.3 FDDI enhancements

Widening FDDI

Other alternate media work is helping to extend FDDI's geographic reach. Telephone companies will soon be offering synchronous optical network services. An ANSI group is working on a physical layer mapping, see figure 4.1, to allow Sonet (Synchronous Optical NETwork) to carry full-bandwidth FDDI traffic and make an enterprise "LAN" possible that connects many users across cities or in different cities.

Broadening FDDI

FDDI-II, see chapter 2.7, builds on the basic services of FDDI to provide a mechanism for supporting both asynchronous packet data and isochronous circuit switched data on the same LAN. FDDI-II operates with the same PMD layer as specified for FDDI. Requiring only relatively minor enhancements to the MAC and PHY layers of FDDI. Its additional functions are defined in the hybrid ring control (HRC) document, see figure 4.1.

FDDI follow-on

Another FDDI enhancement is called FDDI follow-on (FFOL), which will include services operating at speeds ranging from 600 megabits per second to more than 1 gigabit per second. FFOL will provide connections to an array of wide area network (WAN) services, including the synchronous optical network.

Requirements for the next generation FDDI, a proposal:

- The ability to provide a backbone for multiple FDDI networks.
- Efficient interconnections to WANs such as Broadband ISDN (BISDN).
- Support for integrated services, including data, graphics, video and audio.
- An initial data rate of less than 1.25 Gbit/sec (easy mapping on existing carriers).
- Data rates matched to the Synchronous Digital Hierarchy.
- Duplex links for reliability.
- The ability to utilize existing FDDI cable plant, where practical.
- The ability to operate over leased-line public network links, such as Sonet.
- Support for both singlemode and multimode fiber.
- Access and recovery protocol relatively insensitive to network size.

FFOL is part of a larger, multistage standards effort to increase functionality of FDDI. The first part will map FDDI onto the synchronous optical network for connecting widely dispersed rings. An interface to Sonet will give FDDI users the ability to route traffic over local exchange and long-distance carrier networks to connect FDDI networks. Another stage, FDDI-II, will add circuit-switched voice, and high-quality audio and video communications, as discussed earlier.

Once FFOL standards are in place, new equipment will be positioned to take advantage of the standard's features. Existing FDDI systems can make the transition to FFOL with the addition of hardware and software module enhancements.

An FFOL standard proposal should support ring and tree physical network topologies. It should provide specific management mechanisms for network components, including connections, stations and protocol entities, while providing fault isolation, reporting and recovery mechanisms. It should also provide access to circuit-switched services, as well as asynchronous and asynchronous-transfer-mode (ATM) like packet services.

The following set of FFOL standards projects have been proposed [36]:

- Physical layer medium dependent standard (PMD) specifies the private optical fiber links and related optical components. It would include single- and multimode fiber with a signalling rate capable of data speeds in the 600 Mbit/sec to more than 1 Gbit/sec range.

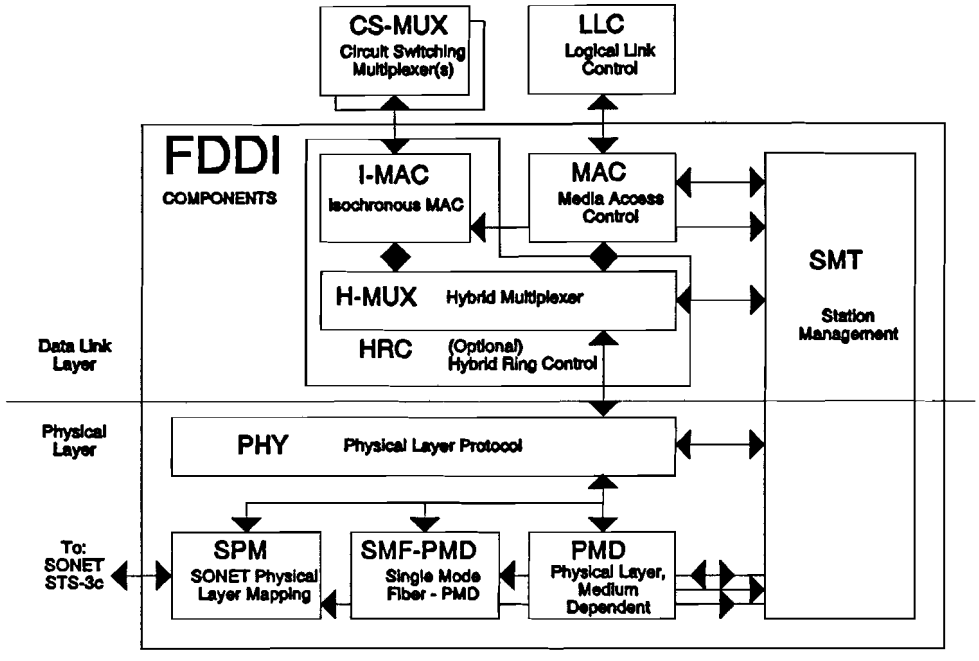


figure 4.1, Structure of FDDI standards

- Physical layer protocol standard specifies the encode/decode clocking and data framing over the PMD and Sonet links. This standard would be scalable from the STS-3 (155.5 Mbit/sec) to STS-48 (2.4Gbit/sec) data rates in the digital signalling hierarchy.

- Service multiplexer standard specifies access to the medium, addressing, data checking and packet generation and reception for circuit-switched service.

- Station management standard provides control for the physical layer, PMD, service multiplexer, asynchronous media access control, and isochronous media access control standards. It also specifies the station configurations, topology configuration and the control required for proper operation of stations as a member of the FFOL network. Station management also provides the interface to the FFOL network management agent.

5. Functional specifications

This chapter contains a combination of outlines of ANSI documents on the PMD [2], the PHY [3], the MAC [4], and the SMT [5].

The PMD document defines the lower half of the Physical layer in the ISO/OSI model.
The PHY document defines the upper half of the Physical layer in the ISO/OSI model.
The MAC document defines the lower half of the Data Link Layer in the ISO/OSI model. The upper half may have a null function.
The SMT document defines a Station Management, which is available to all PMD, PHY and MAC across ISO/OSI boundaries.

5.1 Functions of PMD

PMD has to execute the following functions:
- Transfer electrical signals to light and light to electrical signals.
- Detect wether the incoming light power is above or below the detection threshold.
- Optionally perform a bypass function.

5.2 Functions of PHY

PHY has to execute the following functions:
- Clock synchronization with the upstream code-bit data stream.
- Decoding of the incoming bitstream for use by higher layers
- Encoding and decoding between data and control indicator symbols, and code-bits, medium conditioning and initializing.
- Synchronization of incoming and outgoing code-bit clocks.
- Delineation of octet boundaries for transmission to or from higher layers.
- Information to be transmitted is encoded by the PHY into a grouped transmission code.

5.3 Functions of MAC

MAC has to execute the following functions:
- Construct frames from LLC data, SMT data and MAC data.
- Detect that a token is captured.
- Transmit the frames in correct order.
- Receive frames.
- Reconstruct the data from the frames.
- Direct the data to the right entity.

5.4 Functions of SMT

SMT must monitor the entire station, supply timing values to the MAC, negotiate about bandwidth sharing keep the addressing up to date and execute other management functions.

5.5 Services from PMD to PHY

The services from PMD to PHY allow the PHY to exchange a NRZI code-bit stream with peer PHY entities. The schematic is given in figure 5.1.

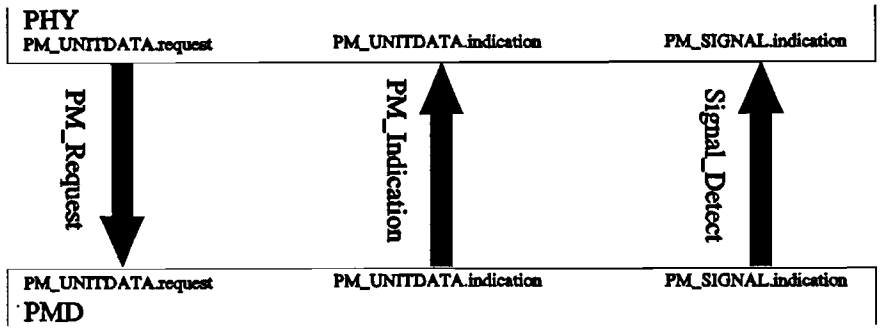


figure 5.1, Schematic of the services between PMD and MAC

PM_UNITDATA.request:
- Defines transfer of NRZI data from PHY to PMD.
- PM_Request is a continuous NRZI code.
- PHY continuously generates the current polarity.
- Receipt causes PMD to convert the electrical NRZI code-bit sequence into the optical equivalent, responding to the level of PM_Request. This may only be done after receipt of a SM_PM_CONTROL.request, with Control_Action set to Transmit_Enable.

PM_UNITDATA.indication:
- Defines transfer of NRZI data from PMD to PHY.
- PM_Indication is a continuous NRZI code.
- PM_Indication is continuously sampled by the clock recovery and receive function of the PHY.

PM_SIGNAL.indication:
- Indicates from PMD to PHY whether the received optical signal is above (Signal_Detect=on) or below (Signal_Detect=off) the detection threshold of PMD.
- The primitive is generated on a change of the status.
- Effect:
  - Off: PHY enters Quiet_Line-state.
  - On: PHY enables detection of other line-states.

5.6 Services from PHY to MAC

When no MAC is inserted in a station (pure repeater), then the interface is PHY-to-PHY.
Services are synchronous: One indication causes exactly one request. The schematic is given in figure 5.2.

PH_UNITDATA.request:
- Defines data transfer from MAC to PHY.
- Each PH_Request stands for one : J, K, T, R, S, n, H and optionally Q or V (n = 0 .. F).

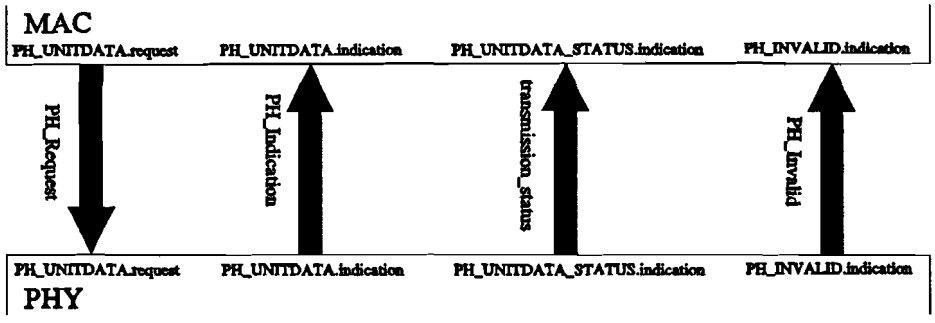


figure 5.2, Schematic of the services between PHY and MAC

- Generated by MAC when PHY has given an indication (according to [4] it is generated whenever MAC has a symbol to transmit; after that MAC has to wait for an indication).
- PHY encodes the received symbol and transmits it.
- When ready for another request, PHY sends a PH_UNITDATA_STATUS.indication.
- Q, H and V are never generated by MAC except when:
  - The repeating function is located in the physical layer: H can be generated by MAC.
  - The repeat filter function is placed after the PH_UNITDATA.request interface.

PH_UNITDATA.indication:
- Defines transfer of data from PHY to MAC.
- Each PH_Indication stands for one : J, K, T, R, S, n, H and optionally Q or V (n = 0 .. F).
- MAC processes the received symbol and generates a PH_UNITDATA.request (with an output).
- Q, V indication is not required when the repeat filter function is located before the PH_UNITDATA.indication interface.

PH_UNITDATA_STATUS.indication:
- Signifies the transmission completion status.
- Responds to every PH_UNITDATA.request, and synchronizes the MAC data with the rate of the medium (when transmission_status is received by MAC, it indicates that the PHY can receive another).
- On receipt MAC is allowed to generate another request.
- PHY provides synchronous service: PH_UNITDATA_STATUS.indication requires an immediate PH_UNITDATA.request.

PH_INVALID.indication:
- PHY tells MAC that the symbol stream is invalid.
- The primitive is generated on:
  - Q, H, Master, Noise_Line-state detection
  - Input error conditions detected by PHY, if not reported as input violation (V) symbols in PH_UNITDATA.indication.
- Receipt causes the MAC receiver to signal FO_Error, or FR_received if ED is already received, to the MAC transmitter and enter the state R0. Depending on the originating state, Frame_Ct, Error_Ct and Lost_Ct are incremented as needed.
  FO_Error or FR_Received causes the MAC transmitter to enter state T0.

5.7 Services from MAC to LLC

The services from MAC to Logical Link Control (LLC) allow the local LLC to exchange LLC SDUs with peer LLC entities. The schematic is given in figure 5.3.

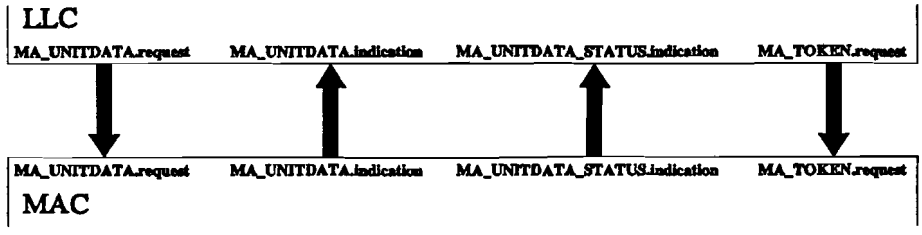


figure 5.3, Schematic of the services from MAC to LLC

MA_UNITDATA.request:
- Defines transfer of one or more SDUs from a local LLC to a single peer LLC, or multiple in case of group addresses.
- Data transferred over the primitive:
  FC_value(1), destination_address(1), M_SDU(1), requested_service_class(1), stream(1)
  ...
  FC_value(n), destination_address(n), M_SDU(n), requested_service_class(n), stream(n)
  token_class
- Each set (1 to n) forms one frame and is referred to as a subrequest:
  * FC_value supplies the frame control field.
  * destination_address may be an individual or group address. From this parameter the DA field should be created. Address length is determined by the L bit in FC_value.
  * M_SDU is the data to be transmitted by the MAC. MAC can determine the length of the SDU from the SDU.
  * requested_service_class is associated with each M_SDU. It can be synchronous or asynchronous. If asynchronous, then the requested_token_class and a priority level can optionally be specified.
  * stream: If Set then at least one other frame is following, if Reset then the last frame of this request has passed.
  * token_class specifies the class of token that MAC has to issue on completion of the transmission (at the end of the request), if no other request is pending that can be honoured.
    On synchronous service request the captured token has to be reissued.
    On asynchronous service request the requested token can be restricted or nonrestricted. When no SDUs were specified, MAC has to issue the requested token_class immediately.

  The frames have to be transmitted in the order presented, regardless of the associated service class. A frame cannot be transmitted when TRT is expired (late_Ct≠0) or because of the requested_service_class and the current value of THT. In this case transmission is terminated and a token is issued according to token_class.
  Subsequently a MA_UNITDATA_STATUS.indication is issued to LLC.
  If transmission_status is successful, then MAC may start transmitting the remaining frames on the next opportunity, or MAC may require reissuance of a new MA_UNITDATA.request.
- Generated by LLC whenever there is data to be transmitted or a token to be issued.
- On receipt MAC adds all MAC specific fields dependent on the media access method, and passes the frame through to lower entities.

- This primitive is the normal way to request the transfer of data. Token capture is implicit to this primitive, and therefore there is no need for a MA_TOKEN.request primitive to be issued in conjunction with it.

MA_UNITDATA.Indication:
- Defines transfer of data from MAC to the local LLC.
- Data transferred over the primitive:
  * FC_value: value of the FC field in the frame.
  * destination_address: Individual or group address specified by the DA field.
  * source_address: Individual address specified by the SA field.
  * M_SDU: MAC SDU received by the local MAC entity.
  * reception_status: Indicates success or failure of the incoming frame. It consists of two elements:
    1. Frame validity: FR_GOOD, FR_BAD. With FR_BAD the reason has to be reported:
    - Invalid FCS. The calculated FCS does not match the received FCS.
    - Length error. Frame has an invalid data length.
    - Internal error. An internal error has occurred preventing MAC from transferring a frame to LLC while A and C are set.
    2. Frame status: Received EAC and optionally other indicator values.
- Generated by MAC to indicate the arrival of a LLC frame addressed to this station.
- Effect of receipt is free implementable.

MA_UNITDATA_STATUS.Indication:
- Provides an appropriate response to MA_UNITDATA.request, indicating success or failure of the request.
- Data transferred over the primitive:
  * number_of_SDUs: Reports the number of M_SDUs transmitted on a given access opportunity as result of this request.
  * transmission_status: Used to pass information back to the requesting LLC. It indicates the success or failure of the previous associated MA_UNITDATA.request. If this specified more than one M_SDU, transmission_status may apply to all of the SDUs transmitted, indicating if all were acknowledged. In this case the resolution of the transmission_status is implementer defined.
  * provided_service_class: Specifies the service class that was provided for the transfer.
- Generated by MAC in response to a MA_UNITDATA.request.
- Effect of receipt is free implementable.
- If there are multiple outstanding requests, additional information may be required for correct association. This may be implied by servicing the requests in a FIFO manner. Possibly MAC maintains multiple queues for different class_of_service, servicing each queue in a FIFO manner.
- It is assumed that if a MAC sets the A and C indicator, the frame is delivered or an internal error is reported to the corresponding LLC.

MA_TOKEN.request:
- LLC requests the capture of the next token.
- requested_toke_class may be restricted or nonrestricted. Optionally a priority level may be specified.
- Generated by LLC when data of a time critical nature is to be transmitted.
- On receipt, MAC captures the next usable token of type requested_token_class. MAC enters state T2 and transmits I symbols until a MA_UNITDATA.request is received, or TRT expires. In the latter case, MAC issues another token of the same token_class as was captured.
- This primitive allows for longer preambles than usual and therefore should not be used for data transfer which is not time critical.

5.8 Services from PMD to SMT

Services from PMD to SMT allow the local SMT to control the operation of the PMD. When PHY services are requested, they are overruled by SMT requests. The schematic is given in figure 5.4.

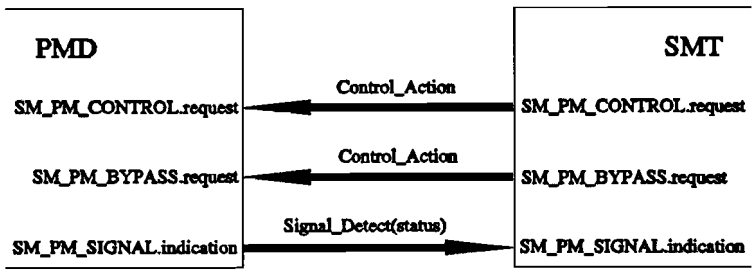


figure 5.4, Schematic of the services from PMD to SMT

SM_PM_CONTROL.request:
- SMT forces the PMD transmit function to place a logic "0" optical signal on the medium.
- Control_Action can be Transmit_Enable or Transmit_Disable.
- Generated by SMT whenever it wants to enable or disable the PMD optical transmitter.
- On receipt of Transmit_Disable, PMD transmits a logic "0" optical signal (no light). This takes precedence over PM_UNITDATA.request.
  On receipt of Transmit_Enable, PMD transmits the optical signal requested by PM_UNITDATA.request.

SM_PM_BYPASS.request:
- Generated by SMT to PMD, indicating that it wants to join or leave the FDDI network.
- Control_Action can be Insert or Deinsert.
- Receipt of Insert causes PMD to activate the optical switch, such that the MIC (Media Interface Connector)inbound signal is directed to the optical receiver, and the output of the optical transmitter is directed to the MIC output.
  Receipt of Deinsert causes PMD to deactivate the optical switch, such that the MIC inbound signal is directed to the MIC output, and the output of the optical transmitter is directed to the input of the optical receiver. This last state is called the bypassed mode.
  Optical bypass switches are optional, so this primitive is not required for stations which do not employ optical bypass switches.

SM_PM_SIGNAL.indication:
- PMD indicates to SMT via Signal_Detect whether the inbound optical signal level is above (status=on) or below (status=off) the detection threshold.
- Generated by PMD to indicate the status of Signal_Detect.
- SMT determines status, and takes the actions specified by the SMT state machines.

5.9 Services from PHY to SMT

Services from PHY to SMT allow the local SMT entity to control the operation of PHY. When MAC services are requested to the PHY, they are executed first (except with SM_PH_LINE-STATE.request commands). The schematic is given in figure 5.5.

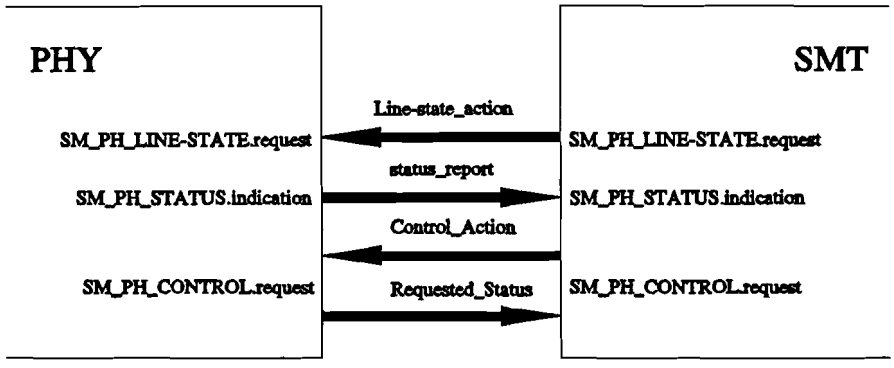


figure 5.5, Schematic of the services between PHY and SMT

SM_PH_LINE-STATE.request:
- SMT requests PHY to send a stream of symbols (to PMD):
  - TRANSMIT_QUIET: PHY must send Q symbols to PMD continuously (no transitions on the medium, this is the initial state of PMD).
    For a proper effect on the optical signal, SMT must Issue an appropriate SM_PM_CONTROL.request to PMD.
  - TRANSMIT_HALT: PHY must send a continuous stream of H symbols to PMD.
  - TRANSMIT_IDLE: PHY must send a continuous stream of I symbols to PMD.
  - TRANSMIT_MASTER: PHY must send an alternating stream of H and Q symbols to PMD.
  - TRANSMIT_PDR: PHY must send the symbol stream presented by MAC on the PH_UNITDATA.request interface to PMD.
- The primitive is generated on station insertion and removal.
- Receipt causes PHY to send a continuous stream of symbols of the type specified by Line-state_action.
- The commands take precedence over MAC-to-PHY commands.

SM_PH_STATUS.indication:
- PHY informs SMT about the line-state activity status changes.
- Status_report can be:
  - QUIET_LINE-STATE_RECEIVED: Quiet_Line-state entered.
  - HALT_LINE-STATE_RECEIVED: Halt_Line-state entered.
  - MASTER_LINE-STATE_RECEIVED: Master_Line-state entered.
  - IDLE_LINE-STATE_RECEIVED: Idle_Line-state entered.
  - ACTIVE_LINE-STATE_RECEIVED: Active_Line-state entered.
  - NOISE_LINE-STATE_RECEIVED: Noise_Line-state entered.
  - LINE-STATE_UNKNOWN: A line-state is exited and the entry conditions to another line-state are not yet satisfied. Indication of the last known line-state must be included.
- The primitives are generated on entrance of the state.
- Receipt causes SMT to take the actions specified by the SMT state machines.

SM_PH_CONTROL.request:
- SMT controls the operation of PHY.

- Generated by SMT for the PHY to take action specified by Control_Action:
    - Reset:
        * PHY resets the transmit mode to TRANSMIT_QUIET.
        * PHY resets the elasticity buffer function.
        * PHY resets the line-state to LINE_STATE_UNKNOWN.
        * PHY resets the line-state counters.
        * PHY resets the smoothing function.
        * PHY resets the repeat filter function.
        * PHY possibly executes extra functions.
    - Present_Status: PHY presents the status to SMT via Requested_Status.
    - Request_Loopback: PHY enters the loopback mode (looping back as close to PMD as possible), to test the local station; symbols on PH_UNITDATA.request are directly passed through to PH_UNITDATA.indication. Symbols may be changed by the repeat filter function.
      PHY also presents continuous NRZI code-bits to PM_UNITDATA.request, such that no light is emitted.
    - Cancel_Loopback: PHY leaves the loopback mode.

5.10 Services from MAC to SMT

The services from MAC to SMT are used by SMT to monitor and control the operation of MAC. The schematic can be found in figure 5.6.

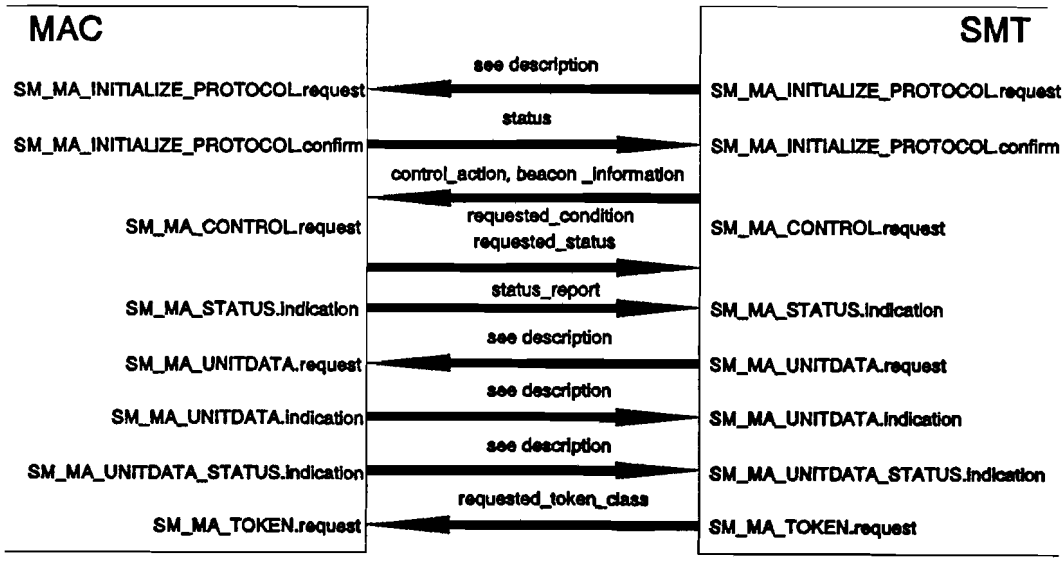


figure 5.6, Schematic from the services from MAC to SMT

SM_MA_INITIALIZE_PROTOCOL.request:
- Used by SMT to reset MAC and optionally to change operational parameters of MAC. The change is not necessary when the MAC is operational on the ring.
- Data transferred over the primitive:
  * individual_MAC_address: (16 bit): 16 bit address for MAC to use as its individual address.
  * individual_MAC_address: (48 bit): 48 bit address for MAC to use as its individual address; both a 16 and a 48 bit address can be supplied.
  * group_MAC_addresses: Addresses that MAC must use as a group address.
  * T_Min_value: Specifies the minimum TTRT supported.
  * T_Max_value: Specifies the maximum TTRT.
  * TVX_value: Specifies the value of TVX to be used.
  * T_Req_value: Specifies the requested TTRT for asynchronous traffic.
  * T_Neg_value: Specifies the negotiated TTRT for asynchronous traffic.
  * T_Pri_value: Specifies a set of priority token rotation time thresholds.
  * indicate_for_own_frame: MAC must also receive and indicate to SMT, frames that it has transmitted itself.
  * indicate_for_rcv_only_good_frame: Value MAC uses to decide between:
    1. Generating MA_UNITDATA.indication and SM_MA_UNITDATA.indication only on good frames.
    2. Generating these primitives on all frames.

  Every parameter is optional. If some parameter is omitted, MAC must use the current value (most recently provided, else the default).
- Generated by SMT when it requires MAC to reconfigure.
- On receipt MAC must reconfigure as denoted by the parameters. After completion, MAC must generate a SM_MA_INITIALIZE_PROTOCOL.confirm.

SM_MA_INITIALIZE_PROTOCOL.confirm:
- Status indicates to SMT the success or failure of the SM_MA_INITIALIZE_PROTOCOL.request.
- Generated by MAC on completion of the request.
- Receipt signifies to SMT that the desired reconfiguration has been accomplished.

SM_MA_CONTROL.request:
- Used by SMT to control operation of MAC.
- Data transferred over the primitive:
  * control_action: Includes one of Reset, Beacon, Present_Status, Reset_Counters, Interrupt_Upon_Conditions or Send_Bad_FCS.
  * beacon_information: Contains the DA field and the information field for the beacon frame, if Beacon is specified by control_action.
  * requested_status: Includes current values of all counters, timers, R_Flag, the current Receiver state and the current Transmitter state.
  * requested_condition: Includes at least one of capture_of_token, receipt_of_frame or passing_of_token.
- Generated by SMT to cause MAC to take the action specified by control_action.
- Receipt causes MAC to take the following actions:
  1. control_action is Reset or Beacon:
     - MAC generates a MAC reset.
     - MAC receiver enters state R0.
     - MAC transmitter enters state T0.
  2. control_action is Beacon (actions are taken after 1. is executed):
     MAC transmitter enters state T5. Beacon frames must be constructed using beacon_information.
  3. control_action is Present_Status:
     MAC presents the status via requested_status.

4. control_action is Reset_Counters:
   MAC resets all counters.
5. control_action is Interrupt_Upon_Condition:
   MAC signals SMT when any of the requested_conditions is detected.
6. control_action is Send_Bad_FCS:
   MAC sends a bad FCS with a specific frame.

SM_MA_STATUS.Indication:
- Informs SMT about errors and significant status changes.
- Status_report specifies appropriate states (p20 [4]).
- Generated by MAC when any of the conditions occur.
- Receipt causes SMT to generate a SM_MA_UNITDATA.request.

SM_MA_UNITDATA.request:
- Defines the transfer of SMT SDUs from SMT to MAC. The same format as for PDUs is used.
- Data transferred over the primitive:
  FC_value(1), destination_address(1), M_SDU(1), requested_service_class(1), stream(1)
  ...
  FC_value(n), destination_address(n), M_SDU(n), requested_service_class(n), stream(n)
  token_class
- Each set (1 to n) forms one frame and is referred to as a subrequest
  * FC_value supplies the frame control field.
  * destination_address may be an individual or group address. From this parameter the DA field should be created. Address length is determined by the L bit in FC_value.
  * M_SDU is the data to be transmitted by the MAC. MAC can determine the length of the SDU from the SDU.
  * requested_service_class is associated with each M_SDU. It can be synchronous, asynchronous or immediate. If asynchronous, then the requested_token_class and a priority level can optionally be specified. If immediate, the frame has to be send immediately, without waiting for a token.
  * stream: If set then at least one other frame is following, if reset then the last frame of this request has passed.
  * token_class specifies the class of token that MAC has to issue on completion of the transmission (at the end of the request), if no other request is pending that can be honoured.
    On synchronous service request the captured token_class has to be reissued.
    On asynchronous service request token_class can be none, restricted or nonrestricted.
    When no SDUs were specified, MAC has to issue the requested_token_class immediately.

  The frames have to be transmitted in the order presented, regardless of the associated service class. A frame cannot be transmitted when TRT is expired (late_Ct≠0) or because of the requested_service_class and the current value of THT. In this case transmission is terminated and a token is issued according to token_class.
  Subsequently a SM_MA_UNITDATA_STATUS.indication is issued to SMT.
  If transmission status is successful, then MAC may start transmitting the remaining frames on the next opportunity, or MAC may require reissuance of a new SM_MA_UNITDATA.request.
- Generated by SMT when there is data to be transferred to peer SMTs or a token to be issued.
- Receipt causes MAC to append all MAC specific fields and fields that are unique to the media access method, and pass the constructed frame through to the lower entities.
- Capture of a token is implicit in this primitive, and therefore there is no need for generation of a SM_MA_TOKEN.request.

SM_MA_UNITDATA.Indication:
- Defines the transfer of data from MAC to SMT (or SMTs in case of group addresses). It is able to report any SMT or MAC frame addressed to this station.
- Data transferred over the primitive:
  * FC_value: value of the FC field in the frame.
  * destination_address: Individual or group address specified by the DA field.
  * source_address: Individual address specified by the SA field.
  * M_SDU: MAC SDU received by the local MAC entity.
  * reception_status: Indicates success or failure of the incoming frame. It consists of two elements:
    1. Frame validity: FR_GOOD, FR_BAD. With FR_BAD the reason has to be reported:
    - Invalid FCS. The calculated FCS does not match the received FCS.
    - Length error. Frame had an invalid datalength.
    - Internal error. An internal error has occurred preventing MAC from transferring a frame to SMT while A and C are set.
    2. Frame status: Received EAC and optionally other indicator values.
- Generated by MAC to indicate the arrival of a MAC or SMT frame addressed to this station. If initialized, this primitive is also generated on receipt of MAC's own frames, allowing examination of elements of reception_status for a frame that has circulated around the ring.
- Receipt causes SMT to generate a SM_MA_UNITDATA_STATUS.indication.

SM_MA_UNITDATA_STATUS.indication:
- Provides an appropriate response to SM_MA_UNITDATA.request, indicating success or failure of the request.
- Data transferred over the primitive:
  * number_of_SDUs: Reports the number of M_SDUs transmitted on a given access opportunity as result of this request.
  * transmission_status: Used to pass information back to the requesting SMT. It indicates the success or failure of the previous associated SM_MA_UNITDATA.request. If this specified more than one M_SDU, transmission_status may apply to all of the SDUs transmitted, indicating if all were acknowledged. In this case the resolution of the transmission_status is implementer defined.
  * provided_service_class: Specifies the service class that was provided for the transfer.
- Generated by MAC in response to a SM_MA_UNITDATA.request.
- Receipt indicates to SMT the success or failure of the request.
- If there are multiple outstanding requests, additional information may be required for correct association. This may be implied by servicing the requests in a FIFO manner. Possibly MAC maintains multiple queues for different class_of_service, servicing each queue in a FIFO manner.
- It is assumed that if a MAC sets the A and C indicator, the frame is delivered or an internal error is reported to the corresponding SMT.

SM_MA_TOKEN.request:
- SMT requests the capture of the next token.
- requested_token_class may be restricted or nonrestricted. The priority level may optionally be specified.
- Generated by SMT when data of a time critical nature is to be transmitted.
- On receipt, MAC captures the next usable token of type requested_token_class. MAC enters T2 and transmits I symbols until a SM_MA_UNITDATA.request is received, or TRT expires. In the latter case, MAC issues another token of the same token_class that has been captured.
- This primitive allows for longer preambles than usual and therefore should not be used for data transfer which is not time critical.

5.11 Symbols

There are several kinds of symbols that can occur on the medium: Line-state symbols, control symbols, data symbols and violation symbols.

Line-state symbols

On detection of a line-state symbol (Q, H, I), the current data transmission process is preempted and abnormally terminated.
- Q: No transitions on the medium.
- H: Indicates control sequences or removal of violation symbols with a minimum DC component.
- I: Indicates a normal condition of the line between transmissions (contains pattern for clock recovery).

Control symbols

- SD: Consists of a JK sequence. SD delineates the starting boundary of a data transmission sequence. It may occur when the line-state is ILS. Further it may succeed or preempt a previous transmission. SD can be detected independent of any previous established symbol boundary.
- ED: Consists of a T . ED terminates all normal data transmissions, and may be followed by one or more control indicator symbols. ED and the control indicators always form a sequence of an even number of symbols. When no control indicators are transmitted, the ED sequence consists of a TT pair. ED cannot be detected Independent of previously established symbol boundaries.
- Control indicators: Consist of Set (S) or Reset (R) symbols, and appear directly behind the ED. They specify logical conditions from a data transmission sequence, and may be altered by repeating stations, without them changing the normal data. ED and control indicators must form an even number of symbols. If that is not so, an extra T symbol must be added behind the last control indicator.

Data symbols

A data symbol is built from a quartet of bits denoted by 0 .. F. A data symbol can be followed by any data symbol. The PHY does not interpret the symbols.

Violation symbol

Indicates a condition on the medium that cannot be modeled by any other line-state. V symbols may not be transmitted onto the medium.

5.12 Line-states

PHY reports the line-state to SMT via SM_PH_STATUS.indication on a SM_PH_CONTROL.request. PHY must report a PH_INVALID.Indication to MAC when QLS, MLS, HLS or NLS is received. The line-state can be determined by SMT at any time. On unknown line-state, LSU (Line-State Unknown) must be reported to SMT along with the most recently known line-state. The line-state is unknown when the current line-state is not valid anymore, and the criteria for entering another line-state have not been met at the time.

Quiet_Line-state (QLS)

PHY sends continuously Q symbols. This state is used as part of the physical connection establishment process, and can indicate the absence of a physical connection.
QLS is entered on loss of Signal_Detect(on) from PMD or after having received 16 or 17 consecutive Q symbols with Signal_Detect(on).
QLS is left on reception of another symbol than Q with Signal_Detect(on).

Master_Line-state (MLS)

PHY sends an alternating stream of H and Q symbols. This state is used as part of the physical connection establishment process.
MLS is entered on reception of 8 or 9 consecutive HQ (or QH) symbol pairs with Signal_Detect(on).
MLS is left on reception of any symbol pair other than HQ (or QH), or loss of Signal_Detect(on).

Halt_Line-state (HLS)

PHY sends continuously H symbols. This state is used as part of the physical connection establishment process.
HLS is entered on the reception of 16 or 17 consecutive H symbols with Signal_Detect(on).
HLS is left on reception of another symbol than H, or loss of Signal_Detect(on).

Idle_Line-state (ILS)

PHY sends continuously I symbols. This state is used to establish and maintain clock synchronization. It is used as part of the physical connection establishment process, and between MAC frames.
ILS is entered on reception of 4 or 5 consecutive I symbols with Signal_Detect(on) (Clock_Detect asserted if implemented).
The number of symbols mentioned may be increased with at most 11 bits, when the Elasticity Buffer is placed before the Line-state Detection function; it removes the maximum number of bits (11).
ILS is left on reception of another than I, or loss of Signal_Detect(on) (Clock_Detect deasserted if implemented).

Active_Line-state (ALS)

When PHY transmits MAC frames to the physical link, it indicates that a physical connection in this station is enabled (MAC frame sequences send or repeated by this station with TRANSMIT_PDR enabled). ALS indicates that symbol stream on the physical link is a MAC sequence, and that the neighbouring PHY has enabled the physical connection.
ALS is entered on reception of a JK sequence independent of previously established code-bit boundaries, with Signal_detect(on) (Clock_Detect asserted if implemented).
ALS is left on reception of any other symbol than I, n, R, S or T (n = 0 .. F), or on loss of Signal_Detect(on) (or Clock-Detect deasserted if implemented), or on entrance of ILS.

Noise_Line-state (NLS)

PHY may not transmit any symbol that can cause the neighbouring PHY to detect a NLS. NLS indicates that the incoming line is noisy. When NLS persists, this line is faulty.

NLS is entered on reception of 16 or 17 potential noise events, without satisfying the criteria for entrance of another Line-state. A noise event can be:
- Decoding a Q, H, J, K, or V symbol (or a symbol pair containing one of these symbols) with Signal_detect(on).
- An elasticity buffer error with Signal_Detect(on).
- Decoding a mixed control and data symbol pair with Signal_Detect(on).
- Decoding a n, R, S or T symbol (or a symbol pair containing one of these symbols) with Signal_Detect(on) (Clock_Detect asserted if implemented) while the last known or current line-state is not ILS or ALS.
- If Clock_Detect is implemented: Decoding an I, n, R, S or T symbol (or a symbol pair containing one of these symbols or JK) with Signal_Detect(on) but Clock_Detect deasserted.

The count of the noise events must be reset to zero when the criteria for entering or continuing another line-state are satisfied.
NLS is left on satisfaction of a criterium to enter another state. When the criteria for NLS and another state occur simultaneously, the other line state takes precedence.

5.13 Coding

Encoding and decoding of the symbols is done in two stages. One is to convert the symbols from the MAC to a NRZ code and vice versa. The other is to convert the NRZ code to a NRZI code and vice versa.

The sequence of the codes transmitted on the medium is determined by the MAC. Via the symbols, the MAC and the PHY exchange information:
- Line-state symbols.
- Control symbols.
- Data symbols, which are exchanged in groups of four bits (data-quartet).

A code-bit symbol consists of five code-bits. The assignment of the codes is already presented in table 3.1.

5.14 PDU types

There are two Protocol Data Units (PDUs), tokens and frames. The token format is shown in figure 5.7, and the frame format in figure 5.8.

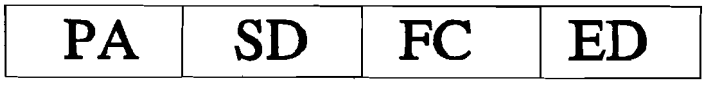


figure 5.7, The token format

The token is used to obtain the right to transmit. Tokens must be recognized when received with a PA of zero or more I symbols, independent of previously established symbol boundaries. When a token is received, and cannot be repeated (due to ring timing or latency constraints), the receiving station must issue a new token.

The frame is used for transmitting MAC and LLC messages to other station(s). The information field may be empty. MAC controls the maximum frame length as required by the physical layer. For counting the frame length, all fields are used, including 4 symbols of the PA. PHY requires a maximum frame length of 9000 symbols.

figure 5.8, The frame format

Fields

PA: Preamble
Transmitted by the originator as a minimum of 16 I symbols. The length may be altered by the subsequent repeating stations. A MAC is not required to be capable to copy frames with a PA shorter than 12 symbols. If it cannot repeat such a frame, then it is not allowed to repeat **any** part of this frame.

SD: Starting Delimiter
Consists of a JK symbol sequence. This sequence is uniquely decodable, independent of previously established symbol boundaries.

FC: Frame Control
Frame control is built up from several parts: CLFF ZZZZ

C: Class bit, 0 indicates an asynchronous frame, 1 indicates a synchronous frame.

L: Address Length bit: 0 indicates a 16-bit address, 1 indicates a 48-bit address. This counts for both the SA and the DA field.

FF ZZZZ: Indicate some kind of frame and extra information understood by the receiving station(s), See table 5.1.

table 5.1: PDU types corresponding with the control bits [4]

CLFF	ZZZZ to	ZZZZ	PDU type
0x00	0000		void frame
1000	0000		non restricted token
1100	0000		restricted token
0L00	0001	1111	SMT frame
1L00	0001	1111	MAC frame
CL01	r000	r111	LLC frame
CL10	r000	r111	reserved for implementer
CL11	rrrr		reserved for future standardization

x = don't care
r = reserved for future implementations, must now be 0

Addresses
Each frame contains these two fields, in the order DA, SA. Addresses may be 16 or 48 bits, and each station **must** be capable to decode 16-bit addresses. When a station can only decode 16-bit addresses, it still must be able to operate in a ring where also 48-bit addresses are used. This station must be capable to:
- Repeat frames with 48-bit addresses.
- Recognize the 48-bit broadcast address (all ones).
- React correctly to claim frames with 48-bit addresses.
- React correctly to beacon frames with 48-bit addresses.

A station using 48-bit addresses must have a minimum 16-bit address capability:
- It has a fully functional 16-bit address.
- It must be able to recognize the 16-bit broadcast address (all ones).

The two address field formats are shown in figure 5.9.

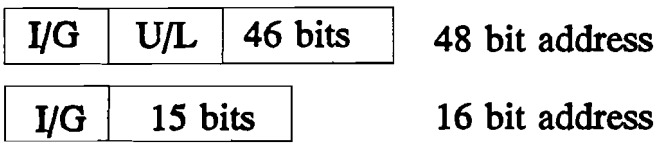


figure 5.9, 16- and 48-bit address formats

Both kinds addresses have an I/G control bit. This stands for Individual or Group.
Only the 48-bit addresses have a U/L control bit. This stands for Universal or Local.

I/G=0 Indicates an Individual address, I/G=1 indicates a group address.
Individual addresses identify one station on the ring (the address is unique in the scope of the administration), while a group address is used to address a frame to multiple stations. The group address may be associated to zero, one or more stations. A group address is associated with a group of logically related stations (by convention).

The broadcast address consists of all ones, and must be recognized by all stations on the ring.

The null address consists of all zeros, and is not allowed to be recognized by any station as its own address.

The U/L bit is used to indicate the kind of address administration. There are two kinds of administration: By a local authority (U/L=1) or by a universal authority (U/L=0). With universal addressing the addresses are distinct from all other addresses on global scope. The procedure for this kind of administration is not specified.

DA: Destination Address
DA identifies the stations to which a frame is sent.

SA: Source Address
SA identifies the originating station of the frame. It has the same format and length as the DA field. I/G must be 0.

INFO: Information field
This contains zero or more data symbol **pairs** whose meaning is determined by the FC field, and interpreted by the destination entity. The length is limited by the requirement of the PHY that the frames may not be longer than 9000 symbols.

The LLC data passed through to MAC is assumed to be an ordered sequence of octets (8 bits). MAC divides these into symbol pairs with the most significant symbol to be transmitted first. This order has to be preserved through all the entities.

FCS: Frame Check Sequence
Used to detect erroneous data bits within the frame and erroneous deletion and addition of bits to the frame. The fields covered by FCS are FC, DA, SA, INFO and FCS. For more information see chapter 3.2.

ED: Ending delimiter
The ED combined with the FS field must always form a balanced symbol sequence (i.e. contain an even number of symbols). It indicates the end of the token or frame.
There are two kinds of EDs: Token EDs and frame EDs. The token ED consists of two T symbols. The frame ED consists of one T symbol.

FS: Frame Status
FS consists of an arbitrary number of control indicator symbols (R and S) following the ending delimiter of the frame. It ends when another symbol than S or R is received. A trailing T symbol, if present is repeated as part of the frame. The three first control indicators stand for the E (error detected) A (address recognized) and C (frame copied) indicators. The meaning of optionally following control indicators is implementer defined, but **must** be repeated by all stations.

E indicator
Transmitted by the originator as R. When some station on the ring detects an error with E=R, than an error is counted, and the E indicator is put to S. When the E indicator was already S, then no error is counted, but E still remains S.

A indicator
Transmitted by the originator as R. When some station recognizes DA as its own individual or group address, it sets A to S, otherwise the repeating station copies the indicator as received.

C indicator
Transmitted by the originator as R. When some station recognizes DA as its own individual or group address **and** it copies the frame (into its receive buffer), then it sets the C indicator to S, otherwise the repeating station copies the indicator as received.

Frames

Void frame (0x00 0000)
This is logically not a frame, and the contents must be ignored. Fields in a void frame have no meaning, except that TVX must be reset, and a void frame must be stripped when SA=MA.

MAC supervisory frames (1L00 0010: beacon, 1L00 0011: claim)
In these frames the length of INFO is at least four octets, of which the first four have the following meaning:
- claim frame: The bid for the TTRT represented as the twos complement of desired TTRT in octets.
- beacon frame: The first octet is the beacon type, and the next three octets are reserved for future assignment. Beacon type 0 means "unsuccessful claim". The other beacon types are not assigned yet.

The claim frame is transmitted during error recovery, to determine the station that must initialize the ring and generate the token. It must be interpreted independent of DA (DA=SA). When bad ring operation is determined by a station, this station enters the claim token state. It sends claim frames and inspects the SAs of the received frames. When SA=MA, it has claimed the token, and generates a new token.

The beacon frame is transmitted to indicate the need for corrective action. It must be interpreted independent of DA (DA=0 or a value supplied by SMT). It is sent as a result of a major ring failure, and is useful to detect the fault.

SMT next station addressing frame (0L00 1111)
The C indicator may only be set by the next addressed station on the ring. This station is distinguished from the others, by a reset A indicator.
The use of this frame is specified in the SMT standard.

Asynchronous LLC frame (0L01 rPPP)
Used for asynchronous transmission. PPP indicates the priority of the frame (111 is the highest, 000 the lowest).

Synchronous LLC frame (1L01 rPPP)
Used for synchronous transmission.

Implementer frame (CL10 rxxx)
Implementer defined frame, xxx is implementer defined.

Note: The r is for future implementations, and must now be 0.

6. Formal description method

To make a proper design, it is important to describe the design in a formal manner, that can be explained in only one way. To be able to do this, certain methods can be used. One of them is described in [1].

The method consists of two parts, a requirements model and an architecture model. The requirements model describes what has to be designed (it contains demands and constraints), the architecture model describes how the requirements can be met. The design is made with many iterations vertically (within one model), and horizontally (between the two models).

6.1 The requirements model

The requirements model consists of the following parts:
- Data Context Diagram (DCD)
- Data Flow Diagrams (DFDs)
- Process Specifications (PSPECs)
- Control Context Diagram (CCD)
- Control Flow Diagrams (CFDs)
- Control Specifications (CSPECs)
- Stores
- Requirements dictionary
- Timing specifications

The DCD shows the flow of data between the system to be designed and the outer world (the context). The data flows are represented by solid arrows. Some of the context parts may not be attached to the system. These parts are not used to obtain data from, but to obtain control information. The parts are shown anyhow to remind that they are present, see figure 6.1.

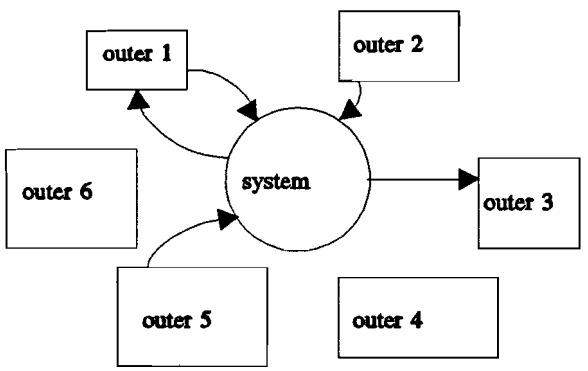


figure 6.1, General example of a Data Context Diagram

The system used in the DCD, is put in the first DFD which is numbered as DFD 0. DFD 0 consists of one process numbered 1. This process consists of several subprocesses, that all have an individual number. They are referred to with the parent process number followed by a dot and the own process number. The subprocesses are placed in a DFD numbered with the parent process number. Each time that an iteration is executed, the design comes to a higher level starting with level number 1 for DFD 0, and level number n+1 when a process named in this level has n dots in its number, see figure 6.2.

The decomposition of a process into smaller processes is done when it makes sense; a primitive process is not decomposed any more. The primitive processes are described by the PSPECs (one for

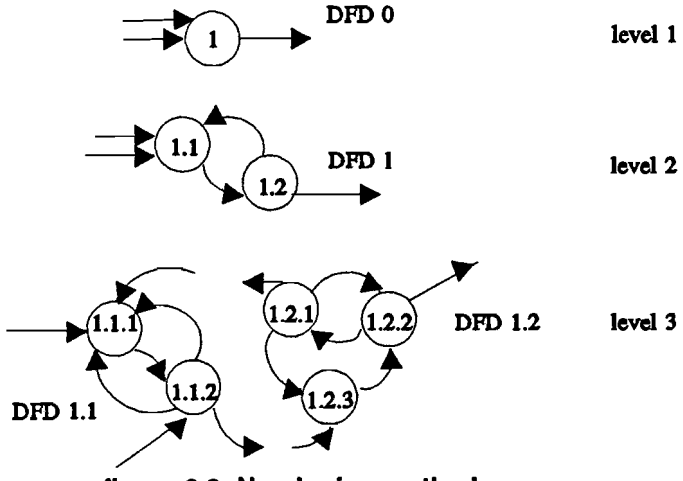


figure 6.2, Numbering methods

each primitive process).

The control part is built up in a similar way as the process part. Starting with the CCD, the context of the control signals is defined. The control flows are represented by a dotted line. The same blocks and processes are used in the CCD as in the DCD, and the context blocks which are not connected, are still drawn to remind their presence, see figure 6.3.

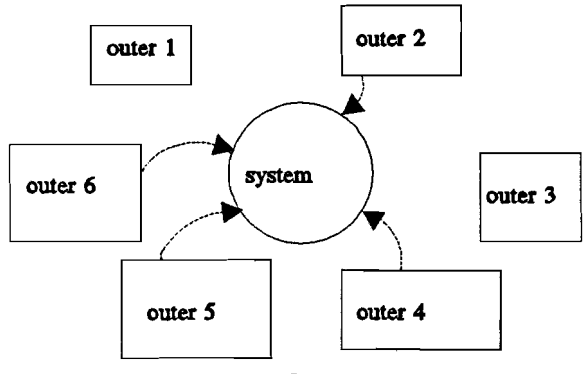


figure 6.3, Example of a Control Context Diagram

The CFD contains the same processes as the DFD. The control flows can flow between processes, and from and to a bar, see figure 6.4.

This bar indicates a connection to the CSPEC that corresponds to this level. In a CFD, only one CSPEC may appear. All bars in a certain diagram refer to the same CSPEC. The CSPECs are optional; they need not to be inserted when no process activators are required.

There are two kinds of stores, data and control stores. These can be used to save data and control flows that may be needed in a later timeslot. Data and control stores may be combined. A store is represented by two lines with the name of the store in between, see figure 6.4.

The requirements dictionary contains all data and control flows along with their definitions. It describes the functions of the data and control flows.

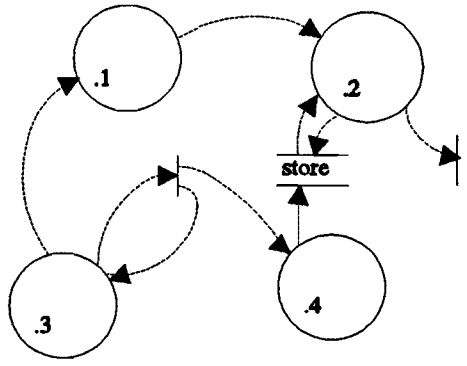


figure 6.4, Use of stores and interface to CSPECs

The last part of the requirements model are the time specifications. Because time is needed in all levels of the design, these are globally available, else there would be to many extra flows in the diagrams, which would make them unreadable. The model itself includes the assumption that it is infinitely fast with respect to the internal data and control flows. Time specifications are only needed for communication with the outer world.

6.2 The architecture Model

The architecture model components are:
- Architecture Context Diagram (ACD)
- Architecture Flow Diagram (AFD)
- Architecture Module Specification (AMS)
- Architecture Interconnect Diagrams (AIDs)
- Architecture Interconnect Specifications (AISs)
- Architecture Dictionary

The relations between these components is shown in figure 6.5.

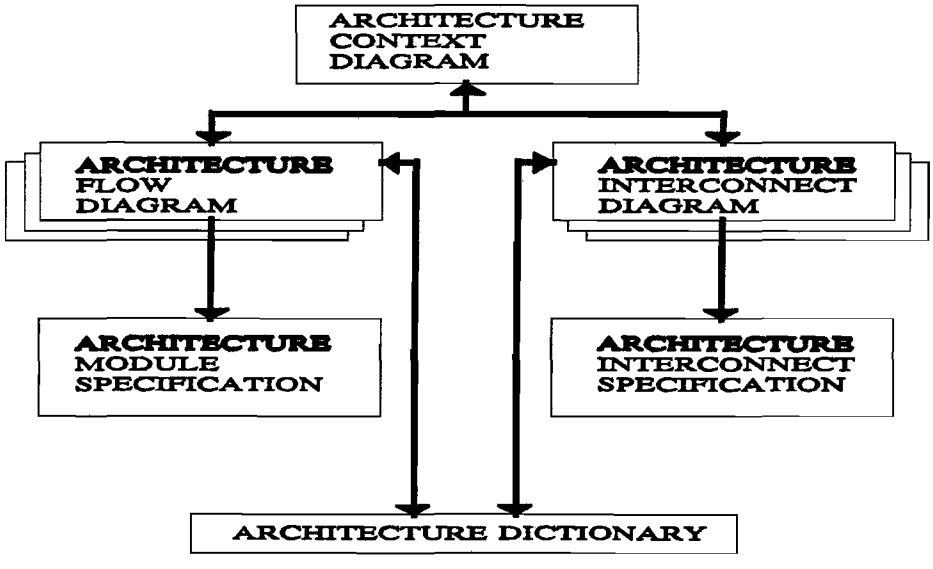


figure 6.5, Architecture model components

The architecture model assigns the processes of the requirements model to physical modules that constitute the system and establishes the relationships between them.
The definition of the physical modules adds four more perspectives to the two addressed by the requirements model (functional and control processing). These are input processing, output processing, user interface processing and maintenance or self-test processing, as shown in figure 6.6.

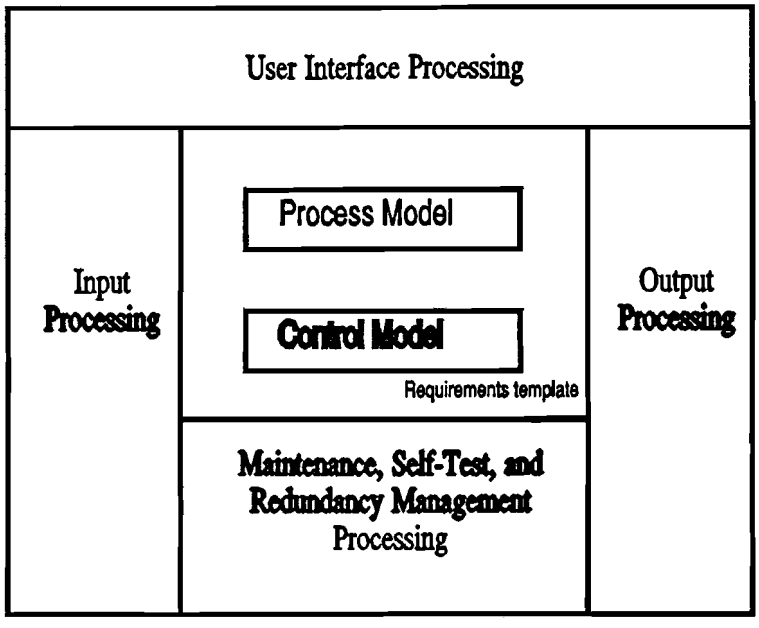


figure 6.6, Architecture template

The input and output processing blocks represent the additional processing needed for each architecture module to communicate with the other modules and to transform the information to and from a usable internal format.

The user interface block is a special case of the input and output processing blocks. This block is separated because there are many special considerations (e.g. human factors) that affect the user interface.

The maintenance and self-test block represents modules required to perform self-monitoring, redundancy management and data collection for maintenance purposes.
The architecture model is a hierarchical layering of modules that are defined by successive application of the architecture template to each of the blocks in the model.

There are three more symbols that will be used in an architecture model, these are:
- Architectural modules or the physical processes, see figure 6.7. The mapping between the architecture modules and the bubbles in the requirements model can be one-to-one or one-to-many in either direction.

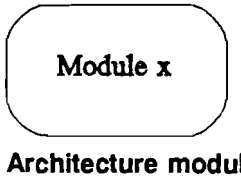


figure 6.7, Architecture module symbol

- Information flow vector, see figure 6.8. This represents the information flow between any two modules, this may be either single elements or grouping of elements, and may contain data flows, control flows, or both.

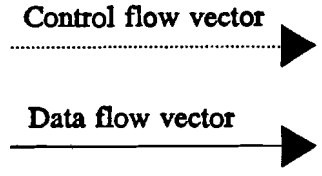


figure 6.8, Information flow vector symbol

- Information flow channel, see figure 6.9. This is the physical means by which information flows from one architecture module to another, it represents the physical communication path. There may be different types of channels, an electrical bus, a mechanical link and an optical link.

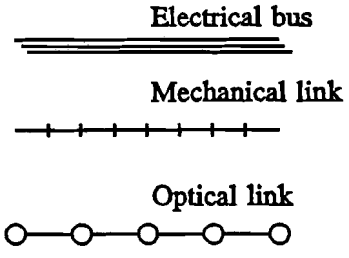


figure 6.9, Information flow channel symbols

The architecture context diagram (ACD) establishes the information boundary between the system being implemented and the environment in which the system has to operate. The ACD is the highest level diagram. The elements that make up the ACD are: one architecture module, representing the system; terminators that represent entities in the environment the system communicates with; and information flow vectors that represent the communications that take place between the system and those entities.

The information flow itself and the information channels can be shown on the same diagram, or on two separate diagrams, depending on the complexity of the system.

The architecture flow diagram (AFD) is a network representation of a system's physical configuration; it documents the information flow between all the architecture modules. The AFD also represents the allocation of processes and flows from the data and control flow diagrams into architecture modules.

The AFD is decomposed to show the next level of system architecture definitions. For instance the Input Processing of a high level AFD will be decomposed to a lower level input and output processing, a user interface and maintenance processing (and a lower level functional and control processing).

The architectural template is used as an allocation guide at each level of the system definition. But recall there is no requirement that every level of the system have modules in every of the four blocks of the template.

The architecture interconnect diagram (AID) is a representation of the communication channels that exist between the architecture modules. An AID always corresponds to an AFD. The AID shows the same architecture modules as its AFD, but it also shows redundant units if they exist.

The components of an AID are architecture modules and communication channels, both may include redundancy.

An architecture module specification (AMS) is written for every module in the architecture model to state the allocation of data flow, control flow, and processing performed by that module. The AMS can be written in different ways, but it always includes at least the requirements model components to show their allocation. The components include DFDs, CFDs, PSPECs and CSPECs.

An AMS consists of three parts:
- A brief narrative specification of what the model is required to do,
- A listing of any architectural requirements for the system,
- A statement of the allocation of the requirements model components to the architecture module.

The purpose of an AMS is to clearly, unambiguously, and completely state the design allocation of the requirements model to architecture modules.

The architecture interconnect specification (AIS) establishes the characteristics of the communication channels on which information travels between architectural modules. It describes the transformation medium as well as the information formats. An AIS is written for every communication channel on an AID, but they may be simplified.

The requirements model timing specification must be allocated to architecture modules, information flow vectors, and information flow channels in the same way as other requirements components are allocated. But the timing specifications play an important role in selecting the implementation technology. They are the determining factor in the tradeoff between different hardware technologies and between hardware and software. Only after these tradeoffs have been made, the requirements model timing specifications are allocated to the resulting architecture components.

Allocation of the timing requirements to the architecture model results in specific timing constraints on the internal architecture modules.

The architecture dictionary consists of a listing of all the data and control elements that flow between the architecture modules as well as between architecture modules and external entities. The architecture dictionary is an enhancement of the requirements dictionary. The additions are the names of the architecture modules between which the information flows, and the names of the these communication channels.

7. Requirements model of FDDI

To model the behaviour and requirements of the FDDI, the method described in chapter 6 is used. This is done with the aid of PROMOD, a program that supports this method. This program checks for inconsistencies, and can generate a report that contains descriptions of the processes and data and control flows. For the diagrams and precise description of processes and flows the reader is referred to appendix F.

Several files are made with PROMOD, these are described in appendix D.

7.1 Formats used in the PROMOD SA report

Each flow is described in the following way:

```
*
 flow_type
 (
 flow_diagram_names, in flow_names
 )
 flow_description
*
 = data_type
```

Flow_type (controlflow or dataflow) indicates to what category the flow belongs to.

The names between braces refer to the flow diagrams (flow_diagram_names), or the **single** flow this flow is a part of.

Behind the closing brace, a short description is given of this flow.

Behind the equal sign, the type of the flow is given, using the following format:

```
type1 + type2        : type1 combined with type2
a { type } b         : combination of at least a times and at most b times type
[ type1 | type2 ]    : type1 or type2
"literal"            : literal
```

In the Minispecs and Controlspecs the question mark (?) is used to indicate the NOT function ("=?" means "not equal to").

The data types and literals used are listed in appendix E.

7.2 Context of FDDI

First the context of the FDDI is determined. The model described here, is one of a single attached station. This can easily be extended to a dual attached station, by extending the SMT part and doubling the FDDI model described here.

On one side there is the fiber ring that transports the data as light, and contains the other connections to the ring. Each connection can be modelled identical to the model described here.

On the other side there is the user, that supplies and receives the data that is transmitted over the ring. The user is a generality for the LLC, that can serve many physical users and applications.

Aside from the FDDI there is an operator part. This can be a part of the LLC. It provides direct access to the FDDI, what can be useful to change parameters etc.

7.3 Subdividing FDDI

To subdivide FDDI into subprocesses, it is logical to choose the division made by the ANSI committee, that defined a PMD, a PHY, a MAC and a SMT. In this way it is easy to divide the work on the total system.

The functions of each part are described in chapter 5. The relation between the primitives and the data and control flows can be easily seen (a primitive can be represented by some data or control flows).

7.4 Interface to the user

As said before, the data flows are related one to one to the primitives defined in chapter 5. When a byte is transferred, it must be in the order Most Significant Nibble, Least Significant Nibble.

MA_UNITDATA.request is built with the following flows:
- userdata_out (data), containing FC_value(i), destination_address(i) and M_SDU(i)
- requested_service_class(i) (control)
- stream(i) (control)
- token_class (control)

MA_UNITDATA.indication is built with the following flows:
- userdata_in (data), containing FC_value, destination_address, source_address and M_SDU
- reception_status (control)

MA_UNITDATA_STATUS.indication is built with the following flows:
- number_of_SDUs (control)
- MA_transmission_status (control)
- provided_service_class (control)

MA_TOKEN.request consists of one flow:
- token_request (control), containing the request itself and requested_token_class

7.5 Interface to the ring

The interface to the ring consists of two data flows: fiber_in, containing the signal that is received by this station, and fiber_out, containing the signal that is transmitted by this station.

The interface is made of fiber and the connectors (MICs) that should be used are described in the standard of the PMD [2].

7.6 Interface to the operator

The interface to the operator is not yet exactly defined. This is because the standard of SMT, that was available, was an old version and the draft version that serves as basis for new SMT standards was not available.

The interface is represented by two data flows (oper_data_in and oper_data_out) and two control flows (oper_control_in and oper_control_out). Each flow may represent a group of other flows.

7.7 Further evolution

The further evolution of the process subdivision is done by discussions (this is possible because the model is built with two persons) and trial and error. This may seem unstructured, but it reflects the growing insight in the operation of the system.

During the evolution, some functions appear to be necessary in several processes, which in turn give rise to some new processes, consisting of a combination of some other processes, or a part of a single process.

7.8 Development of PMD

The PMD mainly consists of a detector and amplifier for the incoming signal and a transmitter (LED) and driver for the outgoing signal. Additionally a bypass can be included, which is an optical switch.

These parts can be directly implemented in hardware and need no further subdivision.

7.9 Development of PHY

The PHY has no more to do than decoding and encoding of data streams, and synchronize the incoming data to the internal clock. These functions can be executed by encode, decode, and synchronize. Synchronize needs the clock the incoming data bits are transmitted with, so an extra process, recover_clock, is added.

All these processes can be directly implemented in hardware, so they need no further subdivision.

7.10 Development of SMT

The SMT has many tasks to fulfil, like ring statistics observation and timing definition. This makes the SMT a very complex and interesting entity. Because the standard that was available was a draft standard that was not fully completed, the SMT part was not further developed.

7.11 Development of MAC

The MAC also has many tasks to fulfil. It must process the userdata to frames, capture tokens, transmit frames as long as allowed, gather ring statistics, receive frames and present frames to the user. It also must receive frames from the SMT, and must be able to generate frames by itself, and transmit them when allowed.

Therefore the MAC is a very interesting entity with many parallellisms and communicating processes. This becomes clear when the ring operation is observed: A frame that is transmitted can return before the token is released. It may also happen that requests occur while transmitting and/or receiving other frames, that possibly must be transferred to the LLC.

To generate frames and process the userdata to frames, process_userdata is defined. This process receives the data from the LLC, SMT and MAC, and adds the control signals like start delimiter, end delimiter, frame check sequence and frame status. It stores the frames until the transmission is allowed and it indicates the number of SDUs transmitted of one request on one token capture. Each process is rather complicated but hard to be subdivided.

Since the outgoing frames may be frames generated by this station, but also frames transmitted by other stations and repeated by this station, the process transmit_frames is needed. This is a kind of multiplexer that switches between repeated and new frames. This is an easy process that needs no further subdivision.

Receive_frames is the process that decodes the frame control and the source and destination addresses, determines the destination of the incoming frames, strips frames, checks the frames on errors and adjusts the frame status.

These functions are executed by the following processes: check_on_function, forward_frames (includes frame stripping), check_on_violations and process_FS. Each process can be subdivided into some other processes, that can be directly implemented in hardware. To improve the understanding of the state machine in check_on_violations, a graphical representation is inserted in appendix G.

Process_frames adjusts the data in such a way that the LLC can read it. This means that the FC, DA, SA and info fields are placed behind one another, and transferred to the LLC.

The monitor takes care of counting events, capturing tokens, issuing tokens, allowing to transmit frames (via timing) and handle SMT and MAC frames. This also is a rather complicated process, that is already subdivided in some other processes. Several of these processes (token_management, timing and frame_handler) should be subdivided themselves.

7.12 Experiences

During the development of the model, a step back had to be made several times. This was to combine some processes, or to place a function in another process.

In this way diagrams have been made that contain as many processes as can be overseen by a human being.

7.13 Further development

As can be seen from the depth of the subdivision, the process MAC has been developed the most. This is done because this is the most important part of the system (it is the connection from the user to the communication system). Apart from the monitor, all processes are subdivided as far as necessary. These processes can be implemented in hardware.

The same counts for the interfaces, of which the hardware definitions must be developed.

More work is to be done to the SMT, that has to be fully subdivided into subprocesses, and the monitor of which several subprocess must be further subdivided.

When a dual attachment station is to be modelled, it can use a great deal of the currently defined processes. Only the MAC has to be extended with a process that can switch the output to the other than the default PHY entity. This is also true for dual attachment concentrators.

For concentrators in general, the SMT will have to be extended from the SMT that should work with the single attachment station. Especially more bypasses will have to be used, since each output must be able to be bypassed.

8. Functional blocks of PHY

In [3] the PHY is subdivided in several functional blocks. This chapter gives an outline to get more insight in the operation of the PHY; it is not used in the design given later in this report.

The proposition made by ANSI about the implementation of the PHY and the interconnection between the several blocks is shown in figure 8.1.

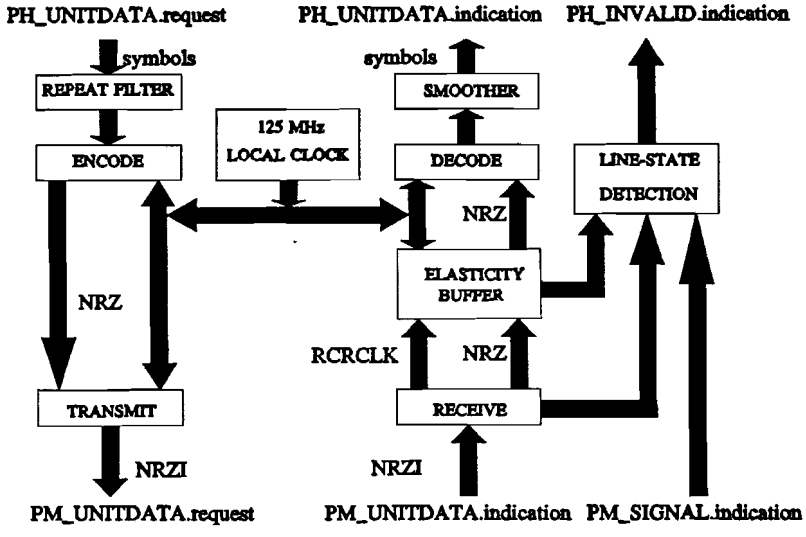


figure 8.1, Possible implementation of the PHY with the interconnection of the blocks

8.1 Encode function

The encode function of the PHY takes care of the proper encoding of the symbols commanded by SMT or presented by PH_UNITDATA.request into NRZ symbols. The code group is presented serially to the transmit function as a continuous stream of NRZ code-bits. This is done by a local fixed frequency oscillator. The encode function is put in the different transmit modes by SMT via SM_PH_LINE-STATE.request. Only in the TRANSMIT_PDR mode, the encode function encodes symbols presented on the PH_UNITDATA.request.

8.2 Transmit function

This function converts the NRZ code-bits into equivalent NRZI code-bits, and presents them to the PMD.

8.3 Receive function

This function converts the NRZI code-bits received from the PMD into equivalent NRZ code-bits, and presents them to other functions in the PHY.

It also must derive a 125 MHz clock from the incoming pulse stream. This is called the Receiver Recovery Clock (RCRCLK) and is used for synchronizing the incoming code-bit cell boundaries during ILS and ALS. It can optionally be used to synchronize detection of the HLS and MLS (this may be done with another synchronization technique).

Optionally a Clock_Detect signal may be provided, to indicate that RCRCLK is locked in frequency and phase. Then this signal will be used by the line-state detection function.

8.4 Elasticity Buffer function

The clock tolerance of the transmit and the receive clock is 0.005%. This insures a maximum frequency difference between the two clocks of 0.01% which equals the length of 4.5 code-bits.

The possible deficiency must be compensated by an Elasticity Buffer (EB). For proper operation of the EB, the MAC that originates a frame, must insert at least 16 I symbols before each frame. The EBs in the subsequent repeating stations may change the length of the I pattern (in this way compensating for differences between the transmit and receive clocks).

The EB works like a FIFO with two clocks: One to clock data into the buffer (RCRCLK) and one to clock data from the buffer to the PMD (the local clock). For correct operation the EB must allow for an elasticity of at least 4.5 bits. The EB will first be filled halfway, before the data can be clocked out.

The implementation of the EB on another place than proposed in figure 8.1, is allowed when several rules (p23 [3]) are followed.

8.5 Decode function

The Decode Function (DF) receives a NRZ bit stream from the EB, synchronized to the local clock. DF establishes symbol boundaries and maintains synchronization to the local symbol clock. It decodes the NRZ pulses to a continuous symbol stream for presentation to the MAC.

Optionally and mostly required by the MAC, the PHY supplies a clock signal to the MAC, for strobing the data from PHY to MAC, and running MAC logic. This clock may have to be continuous, and uninterrupted. Mostly the DF has the ability to insert a 0, 1, 2, 3 or 4 code-bit delay (0 to 9 code-bit delay in byte wide implementations) into the serial pulse stream, to obtain synchronization to the symbol clock (this is especially needed at the start of a new frame). The combined effect of EB and this insertion must satisfy the rules of the EB (p23 [3]).

8.6 Line-State Detection function

The Line-state Detection Function (LDF) determines the state of the incoming physical link. Source for this determination is PM_SIGNAL.Indication, and if implemented signals from the receive function and the EB can be used.

The MAC is informed about invalid signals via PH_INVALID.Indication, and the SMT is informed about changes in the line-state via SM_PH_STATUS.Indication.

The initial line-state is LSU or NLS. The Initial Line-state Detection Interval (ILDI) starts on Signal_Detect(on) and a PM_Indication data stream, which meets the criteria for entering or maintaining HLS, MLS, ILS or ALS. It lasts until the line-state reported to the SMT correctly indicates the state of the received data stream.

The ILDI may not last longer than the maximum PHY acquisition time (AT_Max, default 100 μs). The maximum signal acquisition time (A_Max, time until some signal is detected) used by MAC and SMT equals the sum of AT_Max and AS_Max (maximum PMD acquisition time).The state can be temporarily

lost or changed. Loss results from internal PHY conditions, and changes result from conditions external to the PHY.

Change of line-state or loss of the current line-state may last no longer than the maximum line-state change time (LS_Max, default 15 µs). During this time LSU or NLS must be reported.

Detection of ILS, ALS and optionally HLS and MLS requires RCRCLK synchronization.

8.7 Local Clock

Demands for the local clock:
- Base frequency of 125 MHz ±0.005%
- Phase jitter above 20 kHz < 8° (0.14 rad)
- Harmonic contents above 125.02 MHz < -20 dB
- Nominal code-bit cell time = 8.0 ns
- Nominal symbol time = 40 ns

8.8 Smoothing function

The Smoothing Function (SF) absorbs surplus symbols from longer preambles and redistributes them into shorter preambles. In this way the variance in length of the preambles is reduced.

FDDI design constraints:
- EB is not required to recenter on preambles shorter than 4 symbols.
- MAC is not required to repeat frames with preambles shorter than 2 symbols.
- MAC is not required to copy frames with preambles shorter than 12 symbols.

The SF must be able to insert additional preamble symbols when the preamble is shorter than 14 symbols (gaining these back from preambles longer than 14 symbols). The smoothing capacity centred on 14 symbols (Hi_Max) is at least 2 symbols.

When MAC requires 11 or 12 preamble symbols, SF must be able to insert additional preamble symbols into preambles shorter than 12 symbols (gaining these back from preambles longer than 12 symbols). This extra smoothing capacity centred on 12 symbols (Lo_Max) has to be at least 2 symbols.

The Hi_Max capacity must be located behind the EB in the repeater path. If a MAC is located in the repeater path, Hi_Max and Lo_Max must be located between EB and MAC.

SF must be capable of reclaiming space from stripped or optionally other partial frames by deleting symbols or replacing them by I symbols, provided that a format error is not lost. The format error is not lost when one of the following statements is true:
- The format error is counted in PHY and reported to SMT.
- The format error is correctly propagated to the next MAC or repeat filter.

After a token or partial frame, SF may delete excess symbols from preambles longer than 4 symbols.

SF must be able to maintain its counters and to adjust smoother extension and preamble length in bits, symbols or bytes. The EB quantum may therefore be not larger than the maximum permitted smoother quantum (one byte).

8.9 Repeat filter

The Repeat Filter (RF) is needed for stations where the data must be repeated without the intervenience of a MAC. It must then be placed somewhere after the LDF in the repeat path.

The RF prevents the propagation of code violations and invalid line-states, from the incoming to the outgoing link, while lost frames are propagated to be counted by the next MAC in the line. Also when a MAC is included in the repeat ring, the RF may be included.

9. Design issues of MAC

For the design of the MAC, several items are defined, like timers and state machines. These are formally described in [4], of which this is an outline.

9.1 Timers

Each station has to maintain three timers to regulate the operation on the ring. Each timer is local, and the contents may be different for each station, provided that ring limits are not violated. Reset means that a timer must be reset to its initial value, and restarted. Disable and enable mean stopping and starting of the timer.

Timing values are expressed as the unsigned twos complements of the target or remaining time in octets. This is not needed for implementation, except where these values appear in PDUs on the ring. Timers are assumed to count upward expiring on overflow (X>Y: X elapses sooner than Y). This also is not needed for implementation (depending on the time representation).

Definitions

D_Max = Maximum ring latency. The maximum time that some SD may use to travel around the ring.
Default value is 1.617 ms. This is built upon basis of the default maximum ring length (200 km dual ring, contributing 1.017 ms, 5085 ns/km) and the default number of physical connections (1000, contributing 600 ns, 15 symbols per connection). Other values may be used when other defaults are used for the number of stations or the length of the total ring.

M_Max = Maximum number of MACs. This may be additionally limited by the value of D_Max (two PHYs with one MAC: class A station).
Default value is 1000.

I_Max = Maximum station physical insertion time. Time contributed by the physical layers.
Value is 25.0 ms.

A_Max = Maximum signal acquisition time. Sum of the maximum allowed clock and DC balance signal acquisition time contributed by the attached PHYs.
Value is 1.0 ms.

Token_Time = Token length. Time required to transmit a token (6 symbols) and its preamble (16 symbols). Capture and retransmission is assumed to last a few symbols and not contributing to MAC timing.
Value is 0.88 µs.

L_Max = Maximum transmitter frame setup time. Time permitted for a station to transmit a frame and subsequent frames, after capture of a token.
Value is 0.0035 ms.
L_Max defines the maximum length of the preamble. If L_Max is exceeded, then the station must send a void frame, to allow the other stations to reset TVX.

F_Max = Maximum frame time. Time required to transmit a frame (9,000 symbols) and its preamble (16 symbols).
Value is 0.361 ms.

Claim_FR = Claim frame length. Time required to transmit a claim frame and its preamble (16 symbols). Minimum length is assumed to use 48-bit addresses.
Value is 2.56 µs.

S_Min = Minimum safety timing allowance. Time needed to recover from random noise on the ring. Consists minimally of one frame of maximum length (F_Max), which may be permuted by noise, plus the required L_Max.

Abbreviations used in this chapter can be found in appendix C.

Token Holding Timer (THT)

THT holds the time, during which a station is allowed to transmit asynchronous frames. A new transmission may be initiated when THT has not expired, and is less than T_PRI associated with the frame. THT is initialized with the current value of TRT, when the token is captured.

Valid-Transmission Timer (TVX)

TVX is used to recover from transient ring error situations. The default value for TVX is at least 62,500 symbols (2.50 ms). The timeout value is determined like:
TVX > max(D_Max, F_Max) + Token_Time + F_Max + S_Min

Possibly a time of 2.62144 ms (65,536) can be used, since this time can be implemented with a 16-bit register, clocked by the symbol clock.

Token Rotation Timer (TRT)

TRT is used to control the ring scheduling during normal operation, and to detect and recover from serious ring error situations. It is initialized with different values during different phases of ring operation. When TRT expires, TRT is reinitialized to T_Opr and Late_Ct is incremented.

T_Opr is the operative timeout value of TRT. It is negotiated between the stations to a value between T_Min and T_Max during ring initialization, and is controlled by the transmitter state machine of the MAC. To negotiate for the lowest value of T_Opr, each station uses T_Req between T_Max and T_Min, so that the lowest T_Req becomes T_Opr. The maximum time between two token passings can be 2*T_Opr. If a station requires a guaranteed response time, it should set its T_Req to half of this time.

T_Min is the minimum value for T_Opr for which a station can operate correctly on the ring. If T_Req is smaller, this station cannot operate correctly. The default value of T_Min may not be greater than 100,000 symbol times (4.0 ms) guaranteeing a response time of 8.0 ms.

T_Max is the maximum value for T_Opr, and has to be several times the maximum ring initialization time to permit stable ring recovery.

T_Max > K * T_Init (default K=4)
T_Max > M * T_React + N * T_Resp (default M=2, N=6)

T_Max must be at least 4,125,000 symbols (165 ms). This can be implemented by a 22-bit counter, allowing for 4,194,304 symbols (167,77216 ms), when clocked by the symbol clock.
The time spend in states T0, T4 and T5 depends directly on the choice of the T_Max parameter, so T_Max should not be set to the maximum value mentioned here.

T_Init = T_React + T_Resp < 40.58 ms

T_Init is measured from the time the ring is free of spurious noise. In absence of noise:

T_React < I_Max + D_Max + A_Max + TVX
< 30.24 ms
T_Resp < ((3 * D_Max) + (2 * M_Max * Claim_FR) + S_Min)
< 10.34 ms

(TVX is the actual value used for TVX)

9.2 Counters

Counters are used to count the number of events, creating an error when they exceed a certain threshold.

Late_counter (Late_Ct)

Counts the number of TRT expirations since MAC was reset, or some token was received. Late_Ct is set to one on initialization or reset of the station, and is incremented every time TRT expires. When the ring is operational (Ring_Operational), Late_Ct is cleared to zero every time a valid token has arrived, and Late_Ct is incremented each time TRT expires.

Frame_Ct

This counts the number of all frames received.

Error_Ct

Counts the number of erroneous frames detected by this stations, and not by a previous station.

Lost_Ct

Counts the number of instances in which MAC is receiving a frame or token, and an error occurs so that the credibility of PDU reception is doubted. In this case MAC strips the frame from the ring replacing it with I symbols. Subsequent stations do not increment their Lost_Ct, since they recognize the frame as stripped via the I symbols.

9.3 Operation on the ring

The data is transmitted into frames. The right to transmit a frame on the ring can be obtained by capturing a token.

Frame transmission

Upon a request for SDU transmission, MAC constructs the PDU by placing the SDU in the INFO field of the frame. Upon reception and capture of an appropriate token, MAC starts transmitting the frames according to the token holding rules. During this transmission, the FCS is generated, and appended at the end of the frame.

Token transmission

After transmission is completed, the station immediately transmits a new token. Optionally the station may hold the token until one or more frames have returned. Then the station must check if MA is returned in SA, indicated by M_Flag. Until it has not been returned the station must send I symbols. As soon as FC, DA an SA are received, the station must transmit a token. This last method should not be used with normal data transport, since this decreases the performance dramatically. It may be used for various SMT functions.

Frame stripping

Each station has to strip its own frame from the ring. As soon as a station recognizes SA as MA, it transmits I symbols on the ring. The remnants of the frame (PA, SD, FC, DA, SA) stay on the ring since the recognition is not possible until SA is received (and repeated). This causes not any trouble since the other stations recognize the frame as stripped because of the I symbols after SA and the missing ED. Remnants are removed from the ring as soon as they reach a transmitting station.

Ring scheduling

There are two classes of service, synchronous and asynchronous. The synchronous class is used for traffic that needs a guaranteed bandwidth and response time, the asynchronous class for traffic that does not need that, but can accept dynamic bandwidth sharing.

Synchronous bandwidth is preallocated (via SMT), and asynchronous bandwidth is instantaneously allocated from the remaining bandwidth of the ring.

In the MAC the TRT is used to maintain the ring scheduling. The TTRT is negotiated during a claim token bidding process. The MAC receiver saves the most recently received TTRT (T_Bid_Rc) and passes the final negotiated TTRT (T_Neg) to the MAC transmitter, where it becomes the operational TTRT (T_Opr) upon successful ring initialization.

TRT must be reset each time an early token arrives. When an early token arrives (TRT has not reached TTRT already), the bandwidth may be used for synchronous as well as for asynchronous traffic. When a late token arrives (TRT has exceeded TTRT once), only synchronous transmissions are allowed.

There are different methods to limit the bandwidth used for both kinds of transmissions. However, no station is allowed to hold the token longer than TTRT. This guarantees an average TRT of TTRT and a maximum TRT of 2*TTRT.

Synchronous transmission

The bandwidth that each station may use for synchronous transmission, is expressed as a percentage of TTRT. The allocation is established by the SMT, via SMT PDUs. Initially the allocation is zero for each station, and to change this allocation, every station uses the SMT protocol.

At ring initialization, the station may remember its allocation, provided that there has not been a change in TTRT, or a major ring reconfiguration (no beacon frames have been transmitted or received). The total synchronous bandwidth is expressed as:

TTRT - (D_Max + F_Max + Token_time)

It is not required for interoperability that a station can support synchronous transmission.

Asynchronous transmission

Asynchronous transmission is allocated by two kinds of mechanisms:
- Non restricted token: Bandwidth is usable for all users.
- Restricted token: Bandwidth is usable for specific users.

Ring operation starts in nonrestricted mode. Then the stations may establish the threshold values (T_Pri(n) for each priority level n. Since the lowest priority may only be used when there is much bandwidth left the threshold values must be lower (twos complement!) with decreasing n. When multiple priority levels are not implemented, all T_Pri(n) must be TTRT.

On reception of an early token, the contents of the TRT must be copied to the THT, and TRT must be reset. The difference between the current THT value and TTRT is the time that asynchronous frames may be transmitted. An asynchronous frame of priority n may only be transmitted when THT is lower then T_Pri(n). THT is enabled during asynchronous transmission.

Restricted token mode is entered when some station needs almost all of the asynchronous bandwidth (data burst). The management of the extended communication, is the responsibility of higher level protocols.

Initiation of the restricted token mode is done by the requesting station. It captures a non restricted token, sends initial dialogue frames and finally transmits a restricted token. The destination stations enter restricted token mode upon receipt of the initial dialogue frames, and exchange restricted tokens during the dialogue. The restricted token mode is terminated, when the terminating station transmits its final dialogue frames (after capture of the restricted token), followed by a non restricted token.

In restricted token mode, only the destination stations use the asynchronous bandwidth. Other protocols (even SMT protocols) are not allowed to use this bandwidth. The synchronous transmission can proceed normally.

There is no need for use of THT in restricted token mode, though the stations may not intrude on the synchronous bandwidth. The THT is not needed because the restricted token mode supports fair access, since each station can initiate a dialogue.

To ensure fairness (to the excluded stations) SMT must negotiate and monitor a maximum restricted token mode time. If this time is exceeded, SMT must abort the extended dialogue via the SMT protocol and interface.

Restricted token mode is an option, and not required for interoperability. The ignoring of THT is also optional, and not required.

9.4 Monitoring of the ring

All MACs together fulfil the monitoring function on the ring. This means that each MAC continuously checks the ring on invalid conditions requiring reinitialization. These conditions can be the absence of any activity on the ring, or incorrect activity. Inactivity is usually detected by expiration of TVX, while incorrect activity is detected via the Late_Ct or SMT processes. The relation between the different processes is shown in figure 9.1.

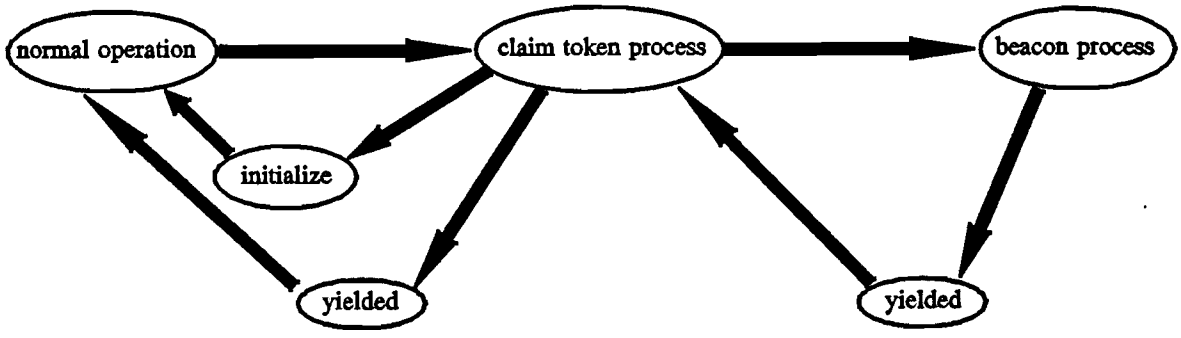


figure 9.1, Relation between the different processes in the MAC

Claim token process

When a station detects a condition that requires the ring to reinitialize, it must start a claim token process. This means that one or more stations bid for the right to initialize the ring, by transmitting claim frames. It also looks for incoming claim frames. When the received bid is higher (shorter time) the station yields, when the received bid is lower (longer time) the station continues bidding its own value. Conflicts are solved as followed:
- The lowest TTRT-bid has precedence (highest bid value).
- In case of equal bid values: The bid with the longest address has precedence (FC.L=1 > FC.L=0)
- In case of equal bid values and equal FC.L: The bid with the highest address has precedence (the highest SA value).

When a station receives its own claim frames back, it has won the bid, and all other stations have yielded. The winning station starts the initialization process.

The claim token process is timed by TRT, set to a large value (T_Max). This ensures stable ring recovery. The TRT will be set to this value, when the station enters the claim token process (not on receipt of individual claim or beacon frames), and may not be reset when the station has left the claim token process until the ring is operational.

If TRT expires while the station is in the claim token process, the process has failed. Possibly external intervention may be required, and the station must start the beacon process.

If TRT expires after the station has left the claim token process, and is waiting for some other station to initialize the ring, it reenters the claim token process.

Initialization process

Each station contains the boolean Ring_Operational, which indicates the current status of the ring. It is cleared when a station detects a claim or beacon process on the ring, and when it receives a MAC reset request from SMT. When Ring_Operational is cleared, all currently serviced (SM_)MA_UNITDATA.requests and (SM_)MA_TOKEN.requests are aborted, no token is issued, and an abnormal confirm is returned to the requester.

Initialization starts on successful completion of the claim token process by a station. First it sets T_Opr to T_Neg, and it resets TRT. At last it issues an initial non restricted token. This is done to align the TTRT values and TRTs over the ring. Since Ring_Operational is clear in each station, none may capture the

Initial token. Upon receipt and repeating of the initial token, each station copies T_Opr to T_Neg, resets TRT, sets Late_Ct to one and sets Ring_Operational.

On the second token rotation, each station accumulates the current synchronous bandwidth utilization, while inhibiting asynchronous transmission, and allowing synchronous transmission. On the third rotation asynchronous transmissions are allowed.

Beacon process

When a fail of the claim token process is detected, or upon request of SMT, the detecting station initiates a beacon process. In this case the ring may have physical damage and may have to be reconfigured (see SMT connection management [5]). To restore the logical ring, some external intervention must be invoked. The purpose of the beacon process is to signal to all remaining stations, that a logical ring break has occurred, and to provide diagnostics or other assistance to the restoration process (via SMT).

When a station enters the beacon process, it starts transmitting beacon frames until a beacon from an upstream station is received; then it yields. When a station receives its own beacon frames, it assumes that the logical ring is restored, and initiates the claim token process.

A station resets TRT on entering the beacon process, and not on receipt of individual claim or beacon frames. When TRT expires after the station has left the beacon process, it enters the claim token process.

9.5 MAC structure

The MAC is built up from two asynchronous cooperating state machines, the transmitter and the receiver, combined with other logic and memories. This is shown in figure 9.2.

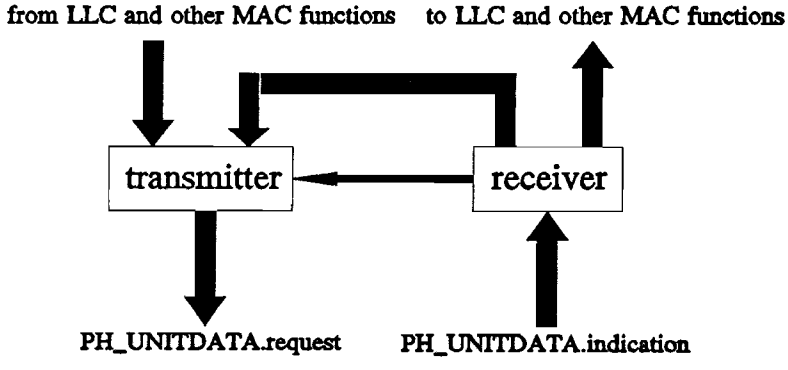


figure 9.2, Basic structure of MAC

The transmitter and receiver are described as state machines. They are separated because of the functions that must be performed parallelly and asynchronously. The state machines are synchronous to the symbol clock which means that if an action takes more than one symbol time, this must be achieved via several states. Actions described as part of a state occur each time the state is entered, actions described as part of a transition, are associated with that transition.

Each state machine must act as follows:
1. Evaluate all conditions within the current state.
2. If the conditions for a state transition are satisfied then:
    - Perform the transition actions in the current state.
    - Enter the new state.
    - Perform the entry actions for the new state.
    - If an immediate transition from the new state is possible, then repeat the sequence starting with 1.
3. If the conditions for in-state actions are satisfied, then execute the specified actions.

PH_UNITDATA.indication is the major trigger event for the receiver. The receiver must perform its processing sequence, using PH_UNITDATA.indication as parameter for condition evaluation. When the sequence is completed, the symbol and associated event signals must be forwarded to the transmitter.

The receipt of a symbol and associated event signals is the major trigger event for the transmitter. For each input symbol, the transmitter must generate a corresponding output symbol. In repeat state the output symbol will be the same as the input symbol, else the input symbol is discarded and a new output symbol is generated.

The contents of specific fields in received and transmitted frames are only known to MAC during the lifetime of the frames. All values that must be remembered, must be saved in MAC variables.

9.6 Receiver state machine

The functions of the receiver are:
- Receive and validate information from the ring.
- Select the portions that are relevant to its station.
- Scan PH_Indication for valid frames (FR) and tokens (TK).
- Pass each frame with a matching DA to the appropriate entity (LLC, MAC, SMT).
- Process the MAC frames after validation.
- Maintain TVX to detect ring failures.
- Generate appropriate signals for the transmitter.
- Save the current values of T_Bid_Rc, T_Neg, A_Flag, C_Flag and E_Flag.
- Maintain Frame_Ct, Error_Ct and Lost_Ct.

Several criteria are defined for frame validity checking, See [4] p40.

For this state machine 6 states are defined numbered R0 to R5. The meaning of the states is as follows:
R0: listen
R1: Await_SD (await starting delimiter)
R2: RC_FR_CTRL (receive frame control field)

R3: RC_FR_BODY (receive frame body)
R4: RC_FR_STATUS (receive frame status)
R5: CHECK_TK (check token)

The description of the states and transitions between them is given in detail in [4] (p41-44).

The state diagram of the state machine is given in appendix A.

9.7 Transmitter state machine

The functions of the transmitter are:
- Repeat information from other stations on the ring.
- Cooperate with other stations to coordinate priorities for use of the ring.
- Operate on the symbol stream from the PHY, and produce a symbol stream to the PHY.
- Repeat the received frames until it needs and receives a usable token.
- Transmit its own data followed by the token, and restart repeating.
- Transmit claim frames when required to recover the ring.
- Maintain the current T_Opr and TRT to ensure correct ring scheduling.

The operation is synchronized by the signals generated by the receiver, and is also using A_Flag, C_Flag and E_Flag saved by the receiver.

For this state machine 6 states are defined T0 to T5. The meaning of the states is as follows:
T0: TX_IDLE (transmitter idle)
T1: REPEAT
T2: TX_DATA (transmit data)
T3: ISSUE_TK (issue token)
T4: CLAIM_TK (claim token)
T5: TX_BEACON (transmit beacon)

The description of the states and transitions between them is given in detail in [4] (p45-50).

The state diagram of the state machine is given in appendix B.

10. Implementation example of PHY

To learn something from other designs, the design of AMD's ENDEC has been watched at. This is not used for the design developed later in this report but only added for extra information and insight.

The Physical Layer Protocol (PHY) simultaneously receives and transmits data. The transmitter accepts symbols from the MAC, converts these five-bit code groups and transmits the encoded serial data stream to the PMD. The receiver receives the encoded serial data stream from the PMD, establishes symbol boundaries and forwards the decoded symbols to the MAC. Additional symbols (Quiet, Idle and Halt) are interpreted by PHY and used to support SMT functions. Since FDDI clocking is point-to-point with each station transmitting on its own fixed frequency clock, an elasticity buffer at each stations PHY is used to compensate for frequency differences between the receive and transmit clocks.

10.1 Propagation delay and station latency

An elasticity buffer and a clock recovery take part of each station. A two stage synchronization both at the bit level as well as at the symbol level, is necessary to regenerate symbols in the receiver. Thus the channel propagation time and the station latency are random variables. Two point-to-point stations can

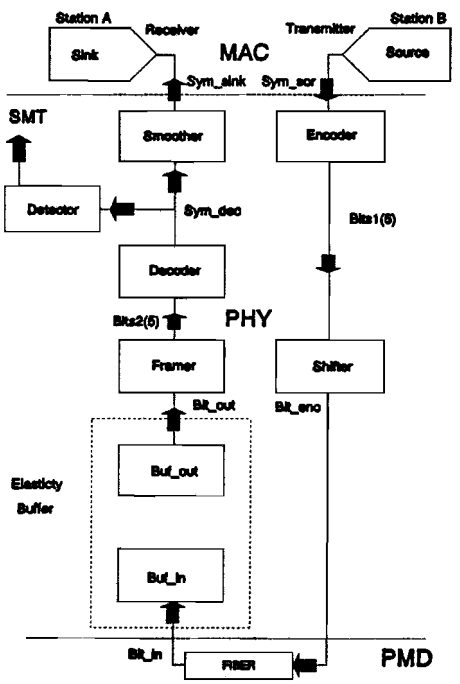


figure 10.1, Transmitter - receiver entities

be modelled as a transmitter-receiver pair, figure 10.1, each pair consisting of nine entities contributing each to the path delay. Theses entities are: Source, Encoder and Shifter in B; Elasticity buffer, Framer, Decoder, Detector and Sink in A. Fiber connects B with A and introduces a Gaussian distributed delay varying from 0 to 8 ns. Station latency is the time that data stays in a station, this is an important parameter, the expected latency average is 19 bits (0.152 μs), [24].

10.2 PHY functional blocks

The main issues of the PHY sublayer are [6]:
- Coding
- Clocking requirements
- Alignment
- Buffering of received data
- Preservation of sufficient preamble
- Filtering spurious symbols
- Station management and diagnostics

4B/5B encoding

For proper data recovery the encoding must ensure that the average DC on the waveform does not deviate from the midpoint of a large extent. The correction of the distortion due to inter symbol interference is accomplished by means of equalization, implemented by a high-pass filter, which eliminates the DC component. Hence as long as the DC fluctuation is minimized by means of proper choice of encoding, it is possible to recover the data with a high degree of accuracy. There are various means of data encoding (Manchester codes, HDB3 in telephone trunk lines, 4B3T in ISDN) with each its own features. For the FDDI required reliability, the 4B/5B coding was chosen. The average DC of the transmitted data varies within 10% of the centre point. Hence distortion introduced by AC coupling in the receiver is minimized to that extent. The FDDI codes can be found in table 3.1.

Encoding must include clock information in the transmitted data to maintain synchronization with the receiver clock recovery circuit. The clock can only be recovered if there is a suitable number of transitions in the transmitted data. With the 4B/5B encoding, this number of transitions is not yet guaranteed. For this purpose another encoding step must take place to guarantee that there are sufficient transitions.

In FDDI Non-Return-to-Zero (NRZ) data is converted to Non-Return-to-Zero-Invert-on-ones (NRZI) data after 4B/5B encoding. In NRZI data, a transition is present for every logic one, these transitions provide the clocking information for the recovery circuit. FDDI uses run-length limited 4B/5B codes, which has at most three consecutive zeros in the serial transmitted stream (run-length limited to three).

The Quiet (five zeros) and Halt (one transition in five bits) symbols violate the run length rule (see table 3.1). If the Quiet and Halt symbols are alternately present, there is only one transition for every 10 bits. These conditions can be handled if the design of the recovery circuit can handle 9 missing transitions for every 10 bits. Of course there is no need for a clock recovery function for a Quiet symbol stream, there is no information present.

Transmit Clock

The data rate on the transmission medium is 125 Megabits per second. FDDI specifications require the clock be accurate in frequency to within 50 parts per million (0.005%). This tight tolerance is imposed to keep the depth of the elasticity buffer within reasonable limits to compensate for frequency differences between the recovered clock and the local clock. If the tolerance on the bit clock is 0.005% the minimum elasticity at the receiver is plus or minus 4.5 bits at the receiver, for a maximum frame length of 4500 bytes.

FDDI also specifies limits on the spectral purity of the clock and the order of suppression of sidebands. these are needed to minimize jitter at the transmitter output: maximum duty cycle distortion of 0.2 ns

and a peak to peak random noise of 0.12 at a bit error rate of $2.5*10^{-10}$. The bit clock can be derived from an internal multiplying Phase-Locked-Loop (PLL).

Receive Clock Recovery

The receive clock is recovered from the received data using a Receive PLL (RPLL). Two kinds of clock recovery mechanisms are used: Phase-Locked-Loop and Surface Acoustic Wave (SAW) filters.

The RPLL must be able to recover the clock information from the received data even when it is jittered up to 3 ns on any bit transition. For a bit cell duration of 8 ns, this works out to 1 ns eye opening, the peak to peak jitter permissible is 3 ns (at a bit error rate of $2.5*10^{-10}$). The maximum acquisition time on the PLL is 100 µs. The worst case, 9 missing transitions for every 10 bits, is the alternating Quiet Halt symbol stream.

The RPLL also needs to handle noisy inputs, such as arise when the input to the optical receiver is either a stream of Quiet symbols or the result of a broken optical link. Under such circumstances, the ring status indicator from the optical receiver can flag active for up to 300 microseconds. At such times, it is vital that the RPLL does not deviate from the expected frequency ranges of the input. In those cases the internal clock can be used for short periods (128 byte periods).

Byte Alignment

Byte alignment (the PHY standard does allow a symbol alignment) is required in the receiver to properly decode received information. When the transceiver is powered up, it sends connection management handshake symbols on the medium. One of the Quiet, Halt, Master or Idle state symbols is selected depending on the node state. For each connection management symbol to be decoded, the alignment of symbols can start with any bit in the symbol. Since the transceiver performs byte alignment on the received data, it can arbitrarily choose the alignment and still decode the information.

The correct receipt of data frames requires proper alignment of the received information. The start of every frame consists of a J and K symbol, the SD. The transceiver recognizes this starting delimiter and realigns itself. It is possible in this realignment process to generate a fragment byte that precedes the JK symbols owing to the new alignment being anywhere from 0 to 9 bits (due to encoding of 8 bits in 10 by 4B/5B) skewed from the previous alignment. If aligned exactly to the previous data, no fragment byte is generated. The FDDI standard allows the fragment byte to contain fragment bits, with the remainder being a few bits of JK. It could therefore potentially be a garbage byte, an Idle byte or it can be completely deleted.

Elasticity Buffer

As FDDI calls for an asynchronous clocking scheme each node must have its own independent clock. Once the data has been recovered by the RPLL, the data must be synchronized to the local clock for the node. An elasticity buffer is needed for the temporary storage of the received data to accommodate to the maximum 0.01% frequency difference. It also permits the deletion or addition of Idle symbols during the preamble. The required elasticity is ±4.5 bits.

When the transceiver is powered up, either Quiet, Halt or Master line state symbols are received. In these cases the buffer will be enabled and get into overflow, or underflow, after 4500 bytes if the clock difference is a maximum of 0.01%. In such case the elasticity buffer flags overflow, or underflow, for one byte and resumes reading the line state symbols. In case of an overflow one byte gets deleted, in case

of underflow one byte of Violation gets inserted. Overflow or underflow is indicated to the rest of the node by means of the Violation symbols, it may be an EB_error.

Smoother

A smoother function is present to ensure that the MAC receives at least 6 bytes of preamble (Idle symbols) before a frame. When the transceiver receives data from a faster station it uses the elasticity buffer to adjust the frequency by deleting a byte as described above. If the next node is also slower, it may also delete a byte of Idle. Over several nodes, preambles that were 8 bytes in number at the originating node may become 5 or less bytes at slower nodes. Since a minimum of 6 bytes of preamble is required to adequately copy addressed frames at the node, the smoother compensates for such preambles by adding additional bytes of preamble.

Any non Idle symbols during the interframe gap are included in the symbol count for the smoother. It is possible that noise could be introduced in the middle of an Idle stream. Since the repeat filter converts non Idle symbols into Idle symbols (in the Force Idle state), it introduces no error for the purpose of counting the symbols.

Line State Overflow/Underflow (OVUF)
Whenever the Elasticity Buffer overflows or underflows, OVUF line state is reported. Whenever OVUF is active, the receive bus (output of the decoder) outputs Violation symbols.

Repeat Filter

The purpose of the repeat filter is to limit the propagation of spurious symbols in the ring. This avoids additional DC imbalance in the ring beyond the 10%. The type of spurious symbol could be a Halt, Quiet, Violation, isolated J or K, or any other unused combination in the encoder truth table (see table 3.1). A byte with parity error could also fall into this class. Whenever one of this set of symbols is sensed in the middle of the frame, four Halt symbols are transmitted before a stream of Idle symbols. This is done to allow the MAC entity in the next node downstream to increment its lost count. The Halt symbols are inserted to distinguish this stream from a stripped frame.

This repeat filter can be modelled by a state machine [6], see figure 10.2. After the station is configured properly this state machine is enabled, if it is not enabled the symbols are transmitted without being filtered.

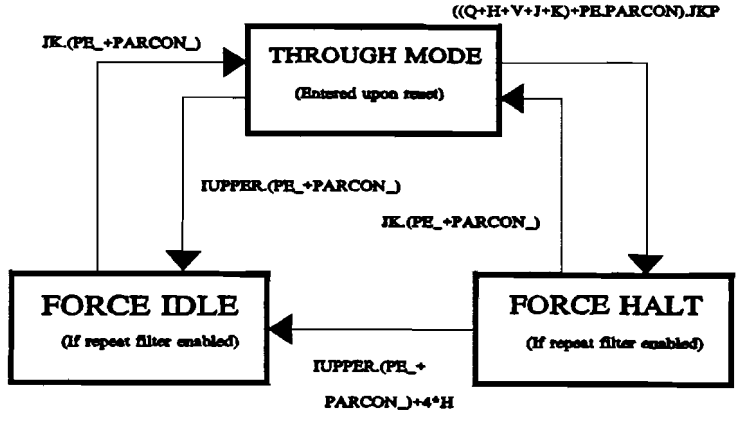


figure 10.2, State machine of the Repeat Filter

There are three states: the Through mode, the Force Halt mode and the Force Idle mode.

Abbreviations:

PE:	Parity Error
PARCON:	Parity error Convert
Q,H,V:	Quiet, Halt, Violation symbol
JK:	Starting delimiter
IUPPER:	Idle as upper nibble
JKP:	JK previously
.,+:	Logical operators

Through mode
This state is entered after reset. If a spurious symbol is seen (in the upper or lower nibble) a transition to state Force Halt takes place. A transition to the other state takes place when an Idle symbol is seen in the upper nibble. A parity error is ignored if the correct bit is set (PARCON).

Force Halt mode
In this state, 4 Halt symbols are sent. The second pair of Halt symbols is sent only if there is not an Idle symbol in the upper nibble, or a JK symbol without parity error is not present at the encoder input. Parity errors can be ignored if the PARCON bit is not set.

Force Idle
Idles are transmitted whenever an Idle symbol is present in the upper nibble. It is also sent after 4 Halt symbols during the presence of a spurious symbol. This conversion does not take place before the first occurrence of JK. The transition to the Through mode takes place after the next JK without parity error. Depending on the PARCON bit, parity errors can be ignored.

10.3 Model of the Encoder/Decoder

The block diagram of the Encoder/Decoder as shown in figure 10.3, is AMD's solution as part of their total solution: the SUPERNET chip-set [6]. This Encoder/Decoder implements the PHY sublayer.

Signals

When a signal name is underlined, this denotes the inverted signal.

R-BUS

$R_{0\text{-}7}$	Receive Bus
RCL	Receive Control Lower
RCU	Receive Control Upper
RP	Receive Bus Parity

TA-BUS

$TA_{0\text{-}8}$	Transmit Bus A
TACL	Transmit Bus A Control Lower
TACU	Transmit Bus A Control Upper
TAP	Transmit Bus A Parity

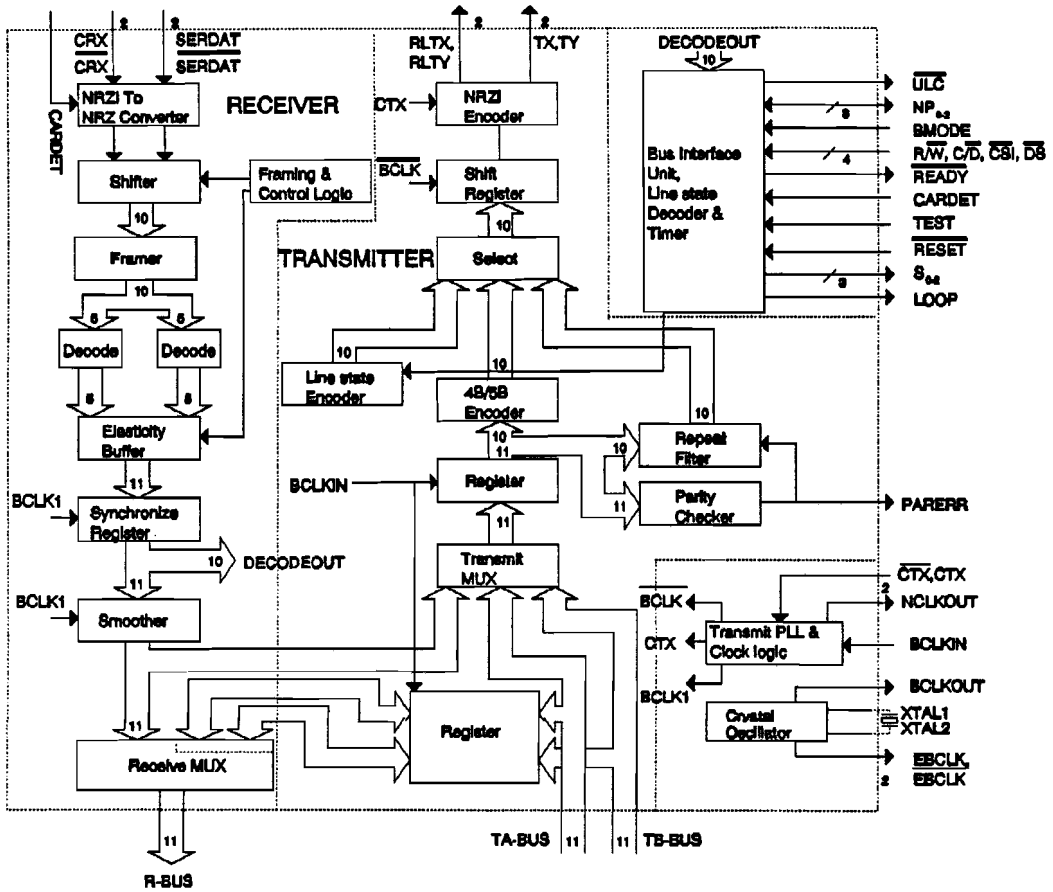


figure 10.3, Model of the Encoder/Decoder

TB-BUS

TB_{0-8}	Transmit Bus B
TBCL	Transmit Bus B Control Lower
TBCU	Transmit Bus B Control Upper
TBP	Transmit Bus B Parity
ULC	Use Local Clock
C/D	Command/Data
CSI	Chip Select
DS	Data Strobe
NP_{0-2}	Node Processor Bus
PARERR	Parity Error
R/W	Read/Write
S_{0-2}	Line Status
BMODE	Bus Mode
BCLKIN	Byte Clock
BCLKOUT	Byte Clock
NCLKOUT	Nibble Clock
CARDET	Carrier Detect
TX,TY	Transmit+, Transmit-
CRX,CRX	Clock Receive+, Clock Receive-
RLTX, RLTY	Receive Loop Transmit+, Receive Loop Transmit-
EBCLK,EBCLK	ECL Byteclock+, Byteclock-

SERDAT,$\overline{\text{SERDAT}}$	Serial Data+, Serial Data-
$XTAL_1$,$XTAL_2$	Crystal 1 and 2
CTX,$\overline{\text{CTX}}$	Clock Test+, Clock Test-

PIN description

Description of the different signals of figure 10.3. Each signal will be described in a short functional way.

R_{0-7}

Receive Bus
The R-bus output sends information being decoded from the serial receive input to either the MAC-Device or another Encoder/Decoder in the station.

RCL

Receive Control Lower
RCL is used to identify the type of information on the lower nibble (R_{0-3}) of the R-bus. If RCL is high then the lower nibble is a control character, if RCL is low then the lower nibble contains data.

RCU

Receive Control Upper
RCU is used to identify the type of information on the upper nibble (R_{4-7}) of the R-bus. If RCU is high then the upper nibble is a control character, if RCU is low then the upper nibble contains data.

RP

Receive Bus Parity
The RP signal ensures odd parity on the R-bus. If the number of "ones" is odd, then the RP will be "zero".

TA_{0-8}

Transmit Bus A
This is one of the two input buses used to accept data and control information from either the MAC-Device or the other Encoder/Decoder (if one exists) at the station.

TACL

Transmit Bus A Control Lower
TACL is used to identify the type of information on the upper nibble (TA_{3-0}) of the TA-bus. If TACL is high then the lower nibble is a control character, if TACL is low then the lower nibble contains data.

TACU

Transmit Bus A Control Upper
TACU is used to identify the type of information on the upper nibble (TA_{4-7}) of the TA-bus. If TACU is high then the upper nibble is a control character, if TACU is low then the upper nibble contains data.

TAP

Transmit Bus A Parity
The TAP signal ensures odd parity on the TA-bus. If the number of "ones" is odd, then the TAP will be "zero". Data with a parity error will be processed by the repeat filter state machine.

TB_{0-8}

Transmit Bus B
This is the other of the two input buses used to accept data and control information from either the MAC-Device or the other Encoder/Decoder (if one exists) at the station.

TBCL

Transmit Bus B Control Lower
TBCL is used to identify the type of information on the (lower) nibble (TB_{3-0}) of the TA-bus. If TBCL is high then the nibble is a control character, if TBCL is low then the nibble contains data.

TBCU

Transmit Bus B Control Upper
TBCU is used to identify the type of information on the (upper) nibble (TB_{4-7}) of the TA-bus. If TBCU is high then the nibble is a control character, if TBCU is low then the nibble contains data.

TBP

Transmit Bus B Parity
The TBP signal ensures odd parity on the TB-bus. If the number of "ones" is odd, then the TAP will be "zero". Data with a parity error will be processed by the repeat filter state machine.

CRX,$\overline{\text{CRX}}$

Clock Receive+, Clock Receive-
The bit rate clock derived from the received serial data (RX, RY or RLTX, RLTY) is sent to the Encoder/Decoder from the receive PLL using CRX and $\overline{\text{CRX}}$.

EBCLK,$\overline{\text{EBCLK}}$

ECL Byteclock+, Byteclock-
The crystal oscillator on the Encoder/Decoder generates the byte rate (12.5 MHz) frequency reference. This signal is synchronous to BCLKOUT.

LOOP

This signal is active when the Encoder/Decoder is in loopback mode. When active, it causes the RLTX, RLTY outputs to be looped back through the RLTX, RLTY inputs of the data separator (see chapter 10.4).

RLTX, RLTY

Receive Loop Transmit+, Receive Loop Transmit-
RLTX, RLTY are an alternate set of serial outputs used to loop back through the data separator. The contain the same data as TX, TY except when in loop back mode.

SERDAT,$\overline{\text{SERDAT}}$

Serial Data+, Serial Data-
SERDAT, $\overline{\text{SERDAT}}$ are NRZI data coming from the data separator.

$\overline{\text{UCL}}$

Use Local Clock
The $\overline{\text{UCL}}$ signal is active whenever CARDET goes inactive, or when in Quiet Line State (QLS). $\overline{\text{UCL}}$ is also controlled by the noise timer inside the Encoder/Decoder. The noise timer forces $\overline{\text{ULC}}$ to be active for a period of 128 BCLK cycles after NLS has been active for at least one byte clock. After 128 BCLK cycles the noise timer releases $\overline{\text{ULC}}$ for the next 128 BCLK cycles. This may be repeated if NLS is still active.

C/D

Command/Data
When high, data on the NP_{0-2} is written into the three bit pointer register. When low, data on the NP_{0-2} is written into one of the control registers or read from one of the control/status registers inside, as pointed to by the pointer register.

CSI

Chip Select
When low, this indicates that the NP has selected the Encoder/Decoder to read or write one of its registers. It should stay low until the READY signal goes low (in asynchronous mode).

DS

Data Strobe
DS is used for defining the presence of data on NP_{0-2}. DS is low when data to be written is valid or when the NP is ready to read. It should stay low until the READY signal goes low.

NP_{0-2}

Node Processor Bus
Data from the node processor.

PARERR

Parity Error
PARERR indicates that there is a parity error on data which is coming in from the selected transmit bus.

READY

The READY signal is a handshake signal for use with an asynchronous NP. When the BMODE is high, as in case of an asynchronous NP, READY is also high and is typically not used in the system. When BMODE is low, the NP runs asynchronous to BCLKIN and the Encoder/Decoder indicates that data being acted upon by bringing READY low. During a write operation, READY goes low after data from the NP is clocked into the Encoder/Decoder. In case of a read operation, READY goes low after the Encoder/Decoder supplies valid data on the NP_{0-2}. Typically the Encoder/Decoder takes between two and three BCLKIN cycles to force READY low as long as DS and CSI stay low.

RESET

This hardware reset initializes the internal logic.

R/W

Read/Write
When R/W is high, the NP reads data. When low, the NP writes data into one of the control or pointer registers.

S_{0-2}

Line Status
These signals indicate the line states of the medium, see chapter 3 for the different codes.

BMODE
Bus Mode
This signal is high when the Encoder/Decoder works in synchronous mode, low when in asynchronous mode.

BCLKIN
Byte Clock
This is the main clock that runs the Encoder/Decoder. It must be driven from its own BCLKOUT or from that of another nearby Encoder/Decoder. The transmit PLL uses BCLKIN as a reference to generate the transmit bit clock (this bit clock is 10 times the frequency of BCLKIN).

BCLKOUT
Byte Clock
BCLKOUT is derived from an internal oscillator which uses an external crystal. This is the main clock running the station. If there are two Encoder/Decoder devices in one station, as in a DAS, only one clock source may be used.

NCLKOUT
Nibble Clock
This signal is derived from the internal PLL and is twice the frequency of BCLKIN.

CARDET
Carrier Detect
This signal is received from the optical receiver. When high, it indicates the presence of data in the fiber. When low, it is used to force a Quiet Line State (QLS). CARDET is ignored in the loopback mode.

TX,TY
Transmit+, Transmit-
These signals provide a differential ECL NRZI output from the Encoder/Decoder. These signals drive the optical transmitter.

$XTAL_1$,$XTAL_2$
Crystal 1 and 2

CTX,$\overline{CTX}$
Clock Test+, Clock Test-

Description of the registers

The Encoder/Decoder has to operate in different modes depending on the state the station or ring is in. For network management reasons these different states must be known and controlled, therefore this Encoder/Decoder needs a set of registers.

Assignments to the four Control Registers and the Pointer Register (all three bits wide) are shown in table 10.1. The Encoder/Decoder can be programmed with these registers.

- CR_0 programs the Encoder/Decoder to transmit data or line states, as described in table 10.2.

- CR_1 selects either the TA-Bus or the TB-Bus inputs. This is accomplished by setting the TMUXSEL bit. CR_1 also forces the Encoder/Decoder into one of the operational modes described in table 10.3.

table 10.1: Control Register and Pointer Bit assignments

Register	Bit 2	Bit 1	Bit 0 (LSB)
CR_0	C2	C1	C0
CR_1	TMUXSEL	LPBK1	LPBK0
CR_2	RESET	EVEN/ODD	PARCON
CR_3	EXTEN2	EXTEN1	SMRESET
$CR_{POINTER}$	REGSEL2	REGSEL1	REGSEL0

table 10.2: CR_0 (Force Line States)

C2	C1	C0	Function	Description
0	0	0	Force Quiet	Q Bytes forced
0	0	1	Force Master	QH Bytes forced
0	1	0	Force Halt	H Bytes forced
0	1	1	Force Idle	I Bytes forced
1	0	0	Enable Repeat Filter	Repeat Filter enabled
1	0	1	Force JKILS	JK followed by Idles
1	1	0	Force IQ	IQ Bytes forced
1	1	1	Unfiltered Encode	Input bits to encoder without filtering.

table 10.3: CR_1 (Loopback Mode)

$LPBK_1$	$LPBK_0$	Function
0	0	Through (figure 10.4)
0	1	Loopback (figure 10.5)
1	0	Short Loopback (figure 10.6)
1	1	Repeat (figure 10.7)

- CR_2 contains the RESET, EVEN/ODD and PARCON bits. When PARCON is set to "one", this enables the conversion of parity by the repeat filter state machine.
- CR_3 is used to reset the smoother (SMRESET), and can be used to read the state of the smoother through EXTEN2 and EXTEN1. When the EXTEN2 bit is set to "one", the smoother is extended by two bits. When EXTEN1 is set to "one", by one byte.

table 10.4: Pointer Register (CR Pointer)

REGSEL2	REGSEL1	REGSEL0	ENABLE
0	0	0	CR_0
0	0	1	CR_1
0	1	0	CR_2
0	1	1	Status
1	0	0	CR_3

- $CR_{POINTER}$ selects one of the control registers as shown in table 10.4. It can also select the line status of the decoded data as indicated by the S_{0-2} lines.

Loading and reading the control registers is done by using the C/D, DS, CSI, R/W and NP_{0-2} signals.

10.4 Operation modes

As shown in figure 10.3, the Encoder/Decoder has a three input receiver and transmitter multiplexer (MUX). The transmit buses originate at either the MAC or the adjacent Encoder/Decoder output. This allows the needed flexibility to the Encoder/Decoder to work under a variety of configurations (four different operation modes).

In a DAS one or two MAC devices are present and two Encoder/Decoders. Each MAC output connects to one of the two inputs in each Encoder/Decoder. This mode is called the Through Mode, as shown in figure 10.4.

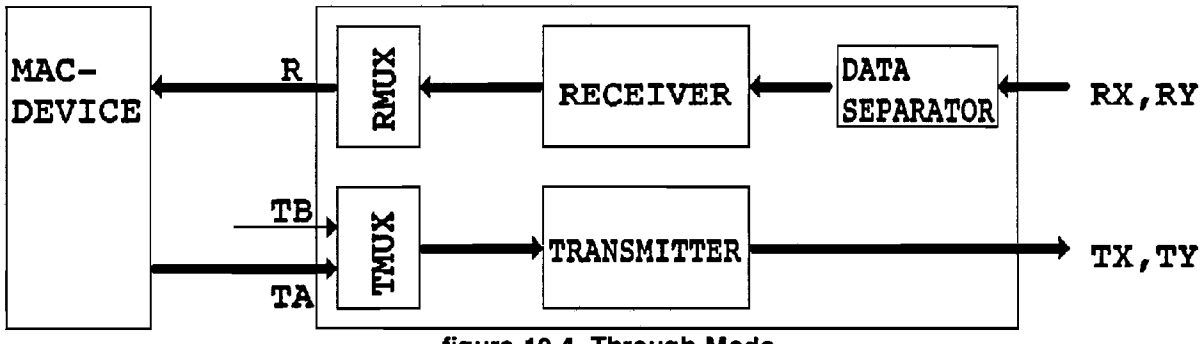


figure 10.4, Through Mode

Through Mode is the normal operating mode with data passed directly between the MAC-Device and the media.

The input at the transmit multiplexer that comes from the Encoder/decoder receiver is useful in situations where the Encoder/Decoder can be used as the repeater. It is also useful during initial connection management sequence when the R-bus output is internally connected back to the transmit inputs.

The loopback mode (figure 10.5) can be used to test the total path until (but excluding) fiber.

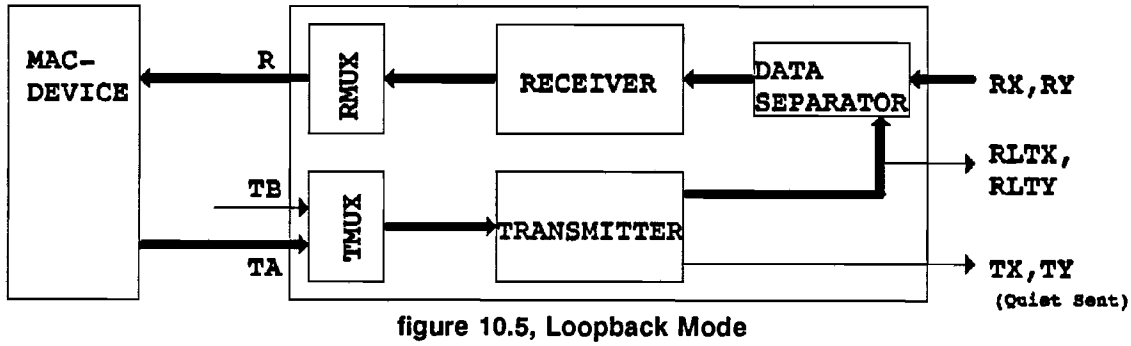


figure 10.5, Loopback Mode

The inputs from the transmit bus are useful for short loopbacks where the MAC is looped back at the input stage of the Encoder/Decoder for diagnostic purposes.

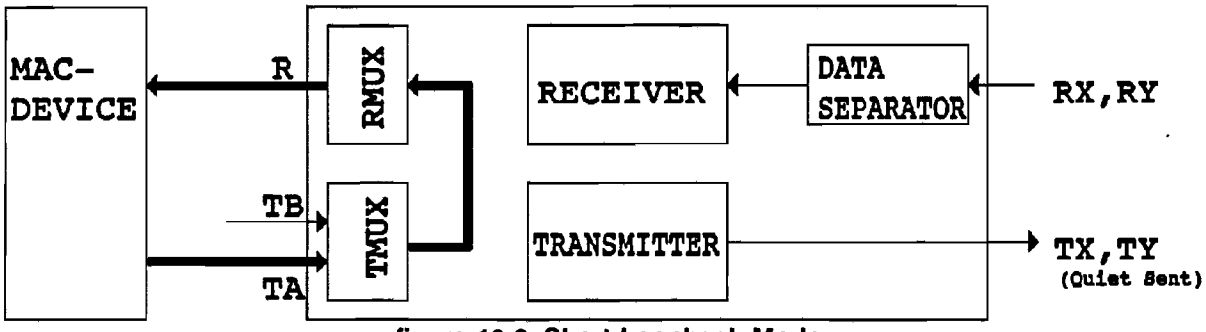


figure 10.6, Short Loopback Mode

The Short Loopback Mode will be used with the MAC external loopback mode (see figure 10.6).

If one of the physical links fails, either MAC has the option of going through the other link during station reconfiguration. In this mode one Encoder/Decoder will be in Short Loopback Mode and one in Repeat Mode, see figure 10.7. More precisely: Encoder/Decoder A will have its control registers to $CR_0 = 100$ and $CR_1 = 11$, i.e. MAC accepts RA-bus. And Encoder/Decoder B will have its registers to $CR_0 = 100$ and $CR_1 = 10$, i.e. MAC accepts RB-bus (see table 10.2 and table 10.3).

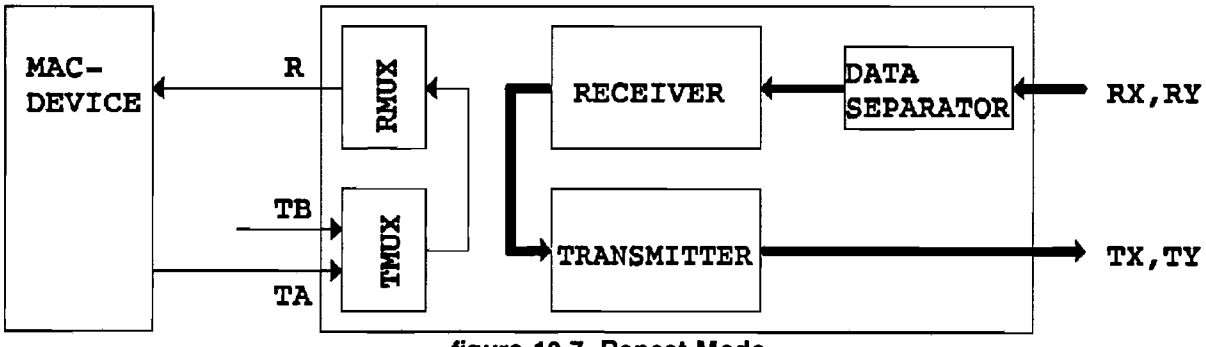


figure 10.7, Repeat Mode

11. Implementation example of MAC

To learn something from other designs, the design of AMD's FORMAC has been watched at. This is not used for the design developed later in this report but only added for extra information and insight.

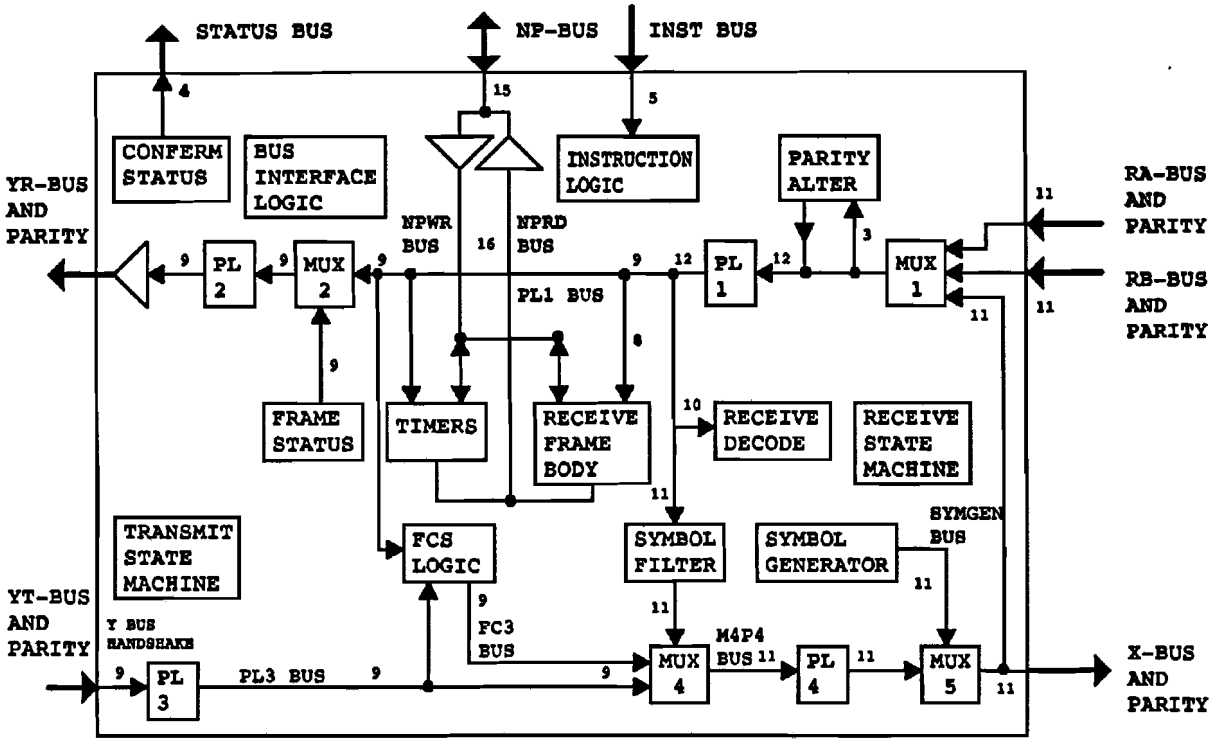


figure 11.1, Functional model of the MAC

In figure 11.1, a functional model of AMD's FORMAC is drawn. This is their MAC implementation and takes part of the SUPERNET chipset [6].

11.1 Block diagram of MAC

The functional blocks as shown in figure 11.1 will be described in a global overview. It may be noticed that this MAC (FORMAC), in practice, will operate in conjunction with a datapath controller, a RAM buffer controller and the PHY as shown in figure 10.3.

External Buses

Seven buses are used to interface the MAC with its surrounding hardware. The NP-bus permits asynchronous and synchronous communication with the NP. The YR-bus is the output for frames received from the media and the YT-bus is the input for data to be transmitted on the media.

The RA-, RB- and X-buses interface with the PHY, the physical layer. Data received from the media is input via the selected R-bus. A MUX at the MAC input permits selection of the active R input bus. Data is sent to the media over the X-bus.

Finally an instruction bus (INST) is used to select between the different read and write operations on the MAC registers.

MUX1
This MUX at the input of the MAC is used to select the input for the receiver logic. Under normal operation the MUX selects one of the R-buses (RA or RB). In case neither R-bus will be used for input, occurring when the MAC is programmed for internal loopback, the MAC X-bus is used as the input (instead of an R-bus). This is useful for "In-circuit" testing of the MAC and associated hardware before the station is placed on the ring. When internal loopback mode is selected, data on the R-buses is ignored.

Pipeline 1 (PL1)
This pipeline captures received data and brings it onto the PL1-bus. This latching allows appropriate setup time for decoding the frame control and address field on incoming frames.

Parity Alter
The parity alter logic changes the parity system used by the MAC. Odd parity is normally used on the selected R-bus (and RCU and RCL inputs, see figure 10.3). This parity system is altered to reflect odd parity on only the R-bus information, by setting input parity based on the state of the RCU and RCL signals. Thus, bad parity received on the RP input will yield bad parity output from this logic.

Receive Frame Logic
This block of logic consists of three basic elements:
- A decoder to interpret the frame control and other special symbols in incoming frames.
- Frame validity logic checks each frame against FDDI frame criteria.
- The receive state machine.

It provides many of the global signals used by other blocks through the station.

Receive Frame Body Logic
This logic sets the flags required for address detection and token reclaiming. The addresses and requested token time associated with this MAC are stored in registers within this MAC. An ALU compares these values with the address and T-bid_Rc fields of received frames. A state machine within the block selects the proper address comparison. The receive frame body logic uses the input from the receive frame logic block to determine the type of address being received (i.e. long, short, individual or group). The flags that are set by this block are used by the transmit state machine and the symbol generator. The flags are also input to the Y-bus handshake block (for data path purposes).

Timers

The timer block contains the timers required for the timed token protocol of FDDI. Three timers are contained within this block.

- Valid Transmission Timer (TVX)
  This 8-bit TVX counts the time between valid frame transmissions on the ring. This is done by checking the EDs of the frames. If the time exceeds the value loaded in TVX, the ring recovery will be initiated through the claim process. The TVX timer is loaded with the default value every time a valid frame of unrestricted token is received by the MAC. The timer is also loaded each time the mode register is programmed to exit the initialization mode.

- Token Rotation Timer (TRT)
  The TRT is used to implement the timed token access protocol. The TRT timer counts the time between receipt of tokens. If TRT exceeds 2 times the operative token rotation

time (2*TOPR) for the network, recovery action is required. When the ring is operational, TOPR will be the time negotiated through the claim process and stored in TNEG. During ring initialization and recovery, TOPR is equal to the default value TMAX.

Negotiated Token Rotation Time (TNEG)
This 21-bit register stores the lowest 21 bits of the bid field of a received Higher Claim or My Claim frame. When the ring becomes operational, this register represents the winning value of the claim process. The upper 16 bits (of TRT) can be read directly with the right instruction, the lower 5 bits can be read with a different instruction. Another instruction is used to load the entire 21-bit TRT register, filling the upper 16 bits with the value stored in TMAX and writing "zeros" in the other five bits.

Timer Rotation Maximum (TMAX)
TMAX (16 bits) specifies the default token rotation time to be used during ring initialization and recovery. It is also used to hold a default TRT (TMAX) during the claim token process, every time the MAC is programmed to exit the initialization mode the TRT is loaded with TMAX.

- Token Holding Timer (THT)
This timer controls the length of time that the node can hold a token for the purpose of transmitting A-frames (Asynchronous). Functionally THT is a down counter. THT is loaded when the MAC captures a token. The value loaded is the difference between the time the MAC expected to capture a token (TOPR) and the actual time it took for the token to circulate the ring. If this difference is greater than the value loaded into TPRI (i.e. the token was received sooner than expected), the MAC can transmit asynchronously. The MAC can continue to transmit A-frames until the THT falls below the TPRI value. Both, THT and TPRI, are 21-bit counters.

Asynchronous Priority Register (TPRI)
This register is used to control the priority of asynchronous messages. The value of TRPI is compared with TRT or THT. If the TRT value is greater than TPRI, a token can be captured for transmission of any pending A-frames. If THT is greater than TPRI, subsequent A-frames are transmitted.

Bus Interface Logic
This block contains the logic necessary to interface the MAC with the Node Processor (NP).

Instruction Decode
This logic decodes the instruction bus. The decoded outputs are used internally as pointers and control signals.

Status/Mode/Interrupts
This block is comprised of the status, mode, and interface logic. The status logic allows for error reporting to the NP. This information can be read directly or by using interrupts. The mode register allows the NP to control the mode of the MAC.

Frame Status
This block is used to map the frame status (FS) of a received frame and other pertinent information into an (8-bit) register. The R and S symbols (see table 3.1) received after the ED of the frame are decoded as a zero and an one, respectively. The contents of the frame status register is appended to the receive frame.

Frame Status Register (FRSTAT)
Frame status is generated and appended to the end of all frames when they are transmitted on the YR-bus.

MUX2
MUX2 is used to append the frame status byte to the frame being sent to the Y-bus. The MUX will output data from the PL1-bus, while received FC and INFO data is contained internally. After the last byte of information has passed through the MUX, it will switch to select the receive frame status register.

Pipeline2
This pipeline latches the received frame information being sent to the Y-bus. This ensures sufficient setup time for a datapath interface.

Confirmation Status
This block is used to confirm the status of the frames originating at the station associated with the MAC.

Transmit State Machine
This block implements the transmit state machine outlined in the FDDI MAC specification.

Pipeline3
This pipeline latches frames before they are transmitted to the MAC internal transmit path. This allows adequate setup time for the frame check sequence logic.

Frame Check Sequence (FCS)
This logic checks the FCS of the received frames, and generates the FCS for the transmitted frames. A MUX steers either the PL1- (internal receive) or the PL3- (internal transmit) bus into the CRC logic. The MUX normally selects the PL1-bus when the MAC is in the repeat or idle mode. When the MAC enters on of the transmit states, PL3 is selected. The CRC is calculated using a 32-bit Autodin II polynomial as specified in the FDDI standard, for more information see chapter 3.2.

MUX4
MUX4 selects on of the different sources to be output on the X-bus. These sources are the output of the symbol filter, the FCS logic and the PL3-bus.

Symbol Filter
The symbol filter logic alters the E_, A_ and C_indicators on the repeated frames as specified by the FDDI standard.

Symbol Generator State Machine
This logic generates the command symbols required for transmission on the media. These include Idle symbols for preamble, as well as the set/reset and Ending delimiter symbols appended to the frames. This logic is also used to generate the tokens when they are issued.

MUX5
MUX5 selects between the Internal transmit and repeat paths and the symbol generator output for transmission onto the X-bus.

Pipeline4
This pipeline ensures PHY setup time.

11.2 Overview other resources

State Machine Register
This register contains the states as described in the FDDI MAC specifications (see figure 11.2).

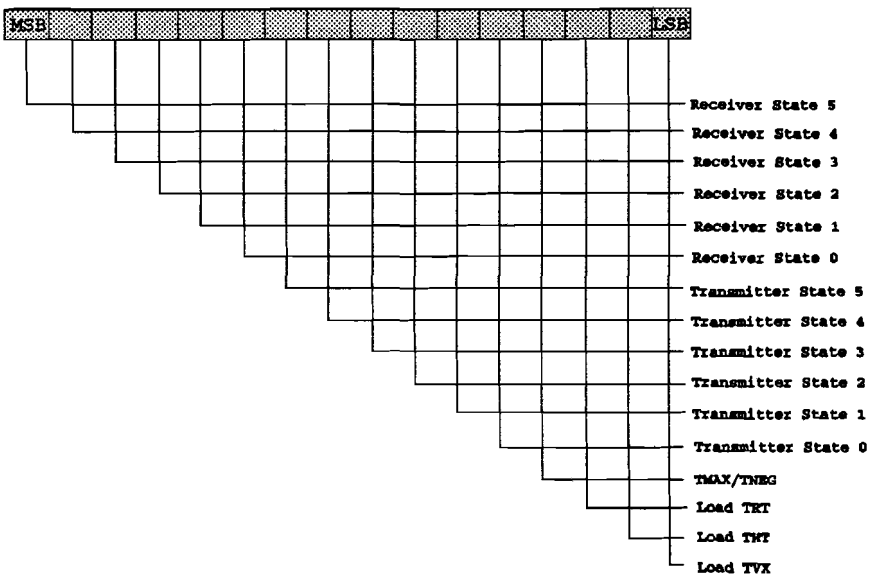


figure 11.2, MAC State Machine Register

Address Register
This address register stores the node's 48-bit address, 16-bit individual and 16-bit group address, and the station T_request time value. These registers are organized in a 8-by-6-bit structure.

MAC Information Register (MIR)
This is a read only register. This register is used to store the first four bytes of the information field of received MAC frames. This register is also used to store the first four information bytes in loopback mode.

Status Register (STAT)
STAT is used to report an error and status to the NP. Individual bits can be used to generate interrupts to the NP.

Interrupt Mask Register (IMSK)
This 13 bit register is used to control maskable interrupts. This register can be accessed using the NP-bus.

Mode Register
On reset, this register is initialized so that all bits are active low (except one). The Node Processor (NP) can access the mode register with the correct instructions. The mode register contains the operational modes of the MAC (see figure 11.3).

$MMODE_{0\text{-}2}$
The three mode bits control the operational mode of the MAC. The assignment is as indicated in table 11.1.

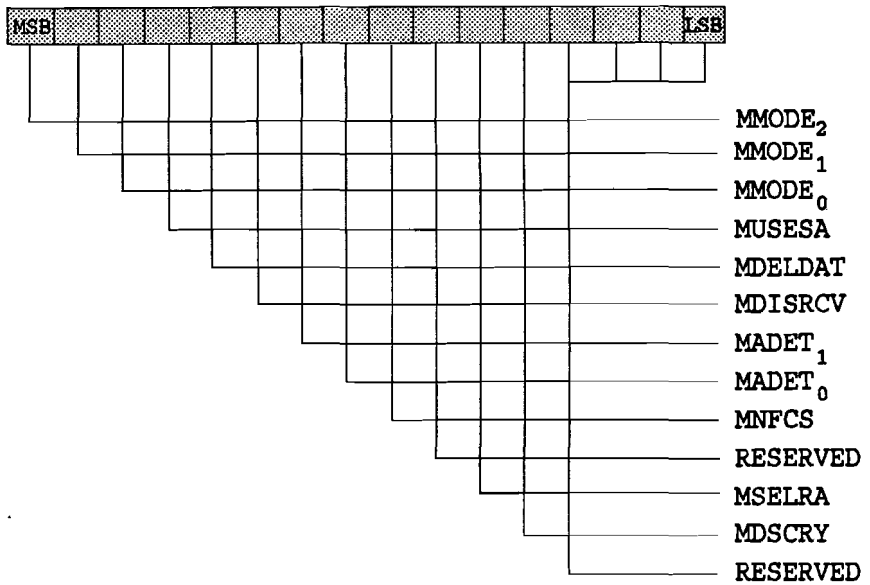


figure 11.3, Mode Register

table 11.1: MAC operational modes

$MMODE_2$	$MMODE_1$	$MMODE_0$	Description
0	0	0	Initialize
0	0	1	Reserved
0	1	0	Reserved
0	1	1	On-Line
1	0	0	Internal Loopback 1
1	0	1	Internal Loopback 2
1	1	0	External Loopback 1
1	1	1	External Loopback 2

Initialize. Initialize is the reset mode of the MAC, this mode is used to read and write many of the internal registers.

On-line. On-line mode places the MAC in the operational FDDI mode.

Internal Loopback 1. Internal Loopback 1 places the MAC in internal loopback mode. The internal transmit bus is connected to the internal receive path. R-bus inputs are ignored.

Internal Loopback 2. Internal Loopback 2 operates exactly like internal loopback 1, except with the receiver operation: Data received in loopback mode will be output on the YR-bus.

External Loopback 1. External Loopback 1 places the MAC in external loopback mode, this mode will be used with the loopback mode of the PHY.

External Loopback 2. External Loopback 2 is analogous to the internal loopback 2.

Loopback

The loopback mode is useful for "in-circuit" testing of the MAC and associated node hardware before insertion in the ring. There are four loopback operations possible:
- If internal loopback is selected, the MAC X-bus is connected internally to the R-bus input. Data on the RA- and RB-buses is ignored.
- If loopback 1 is selected, the data received will not be output on the YR-bus. The shared YT/YR-bus can be used for transmit only. When in loopback, the first four bytes following the SA field are stored in the MIR. This allows limited loopback testing without additional hardware.
- If loopback 2 is selected, the MAC writes the entire received frame to the YR-bus. This data can be read by dedicated hardware. Of course the hardware needed, depends on wether or not a full-duplex system is used.
- If external loopback is selected, the loopback connection is establisched outside. the MAC.

The internal timers are held during loopback mode. This prevents the MAC from entering the recovery state due to a time-out. While in loopback mode, the transmit immediate option should be used to allow the MAC transmitter to enter the transmit data without capturing a token. This is normally used by station management when ring operation is off.

MUSESA (Mode to Use Short Address)
This mode bit is used in the token claiming procedure. This bit is set to "one" if the addresses transmitted in the node's claim frames are 16 bits long. It is important that this bit, the address length bit in the FC fields of the claim and beacon frames stored and the actual address lengths used in the claim and beacon frames, are all consistent. If not, the node will not strip its frames from the ring. This means, a frame could circulate around the ring until another node captures the token and starts transmitting. This could cause the destination node to receive the same frame many times. Further the claim process may never finish, because the node will not recognize its own claim.

MDELADET (Mode Delayed Address Detection)
This mode selects the delayed address detection. If set, external DA detection is accepted until the ED is received. If reset, and DA does not match, the frame is flushed from memory.

MDISRCV (Mode to Disable Receive)
If set the MAC will not receive frames on the YR-bus. The C indicators associated with address recognized valid frames will be repeated as received when the receiver is disabled.

$MADET_{0-1}$ (Mode Address Detect)
The MADET bits reflect the address type of the frame received on the YR-bus, see table 11.2. This mode will be used for flushing purposes.

table 11.2: Address mode detection

$MADET_1$	$MADET_0$	Description
0	0	DA = MA
0	1	DA = MA or SA = MA
1	0	Promiscuous
1	1	Reserved

DA = MA (Destination Address equals My node Address). This is the normal setting. The MAC indicates that the destination address of the received frame does not match the corresponding node address (MA, My node Address) in the MAC. So other functional blocks (data path and ram buffer controller) should not copy the frame.

DA = MA or SA = MA. In this mode the MAC will also inhibit the flush signal for frames it transmitted (SA in the frame matches the node address).

Promiscuous. In this mode the MAC indicates that there is an address mismatch. The frame being received is not addressed to this MAC and is not generated by this MAC.

MNFCS (Mode No Frame Check Sequence)
This bit is used to suppress the generation of CRC on transmitted frames. If set, the MAC generates an ED and FS field immediately after the final byte from the YT-bus is transmitted. This may be useful for loopback testing purposes. Note that the CRC logic will always check the received frame, whether MNFCS is set or not.

MSELRA (Mode to Select RA)
This bit controls the MUX input to the MAC at the interface with the PHY, either the RA- or RB-bus is selected.

MDSCRY (Disable Carry)
This bit is used for internal timer testing. Setting this bit effectively breaks the TRT and THT timers into segments to facilitate testing. These timers are separated into one 5-bit (most significant bits) and four 4-bit segments that clock independently.

11.3 Ring operational

When the hardware has been initialized and the Claim and Beacon frames have been setup in a buffer memory, the station is prepared to begin the Claim Token process with the other stations. To begin this process, the MAC must be placed into the on-line mode (see figure 11.3). This is accomplished by setting the mode bits, $MMODE_{0-2}$.

The Claim Token Process completes when one station receives its own Claim Frame again. Within each station, the boolean variable Ring_Operational indicates the current operating status. This Ring_Operational is cleared whenever the station indicates or detects the Claim or Beacon processes on the ring, and when it receives a MAC_Reset request from SMT.

Ring initialization begins when one station successfully completes the Claim Token process, this station proceeds to initialize the ring. First, it sets the operative TTRT value (T_Opr) to the negotiated TTRT value (T_Neg), what will be the same as its requested TTRT value (T_Req) since this station won the bidding process. Then it resets its Token Rotation Timer. Finally it issues an initial non-restricted Token.

The purpose of this initial rotation of the Token is to align both the TTRT values and the TRT timers in all stations. Since Ring_Operational is clear, no station will capture the initial Token or transmit frames. In each station upon receipt (and repeating) of the initial Token, T_Opr shall be set to T_Neg, TRT shall be reset, Late_Ct shall be set to "one", and Ring_Operational shall be set.

Frame Transmission

Transmit Queue

The transmit queue for either A-frames or S-frames contains three basic components:
- Transmit Frame Descriptor
- Transmit Frame Data field
- Transmit Pointer Field.

Transmit Frame Descriptor

The 32-bit transmit frame descriptor is subdivided into a 16-bit control field and a 16-bit frame length field. The control field is organized as shown in figure 11.4.

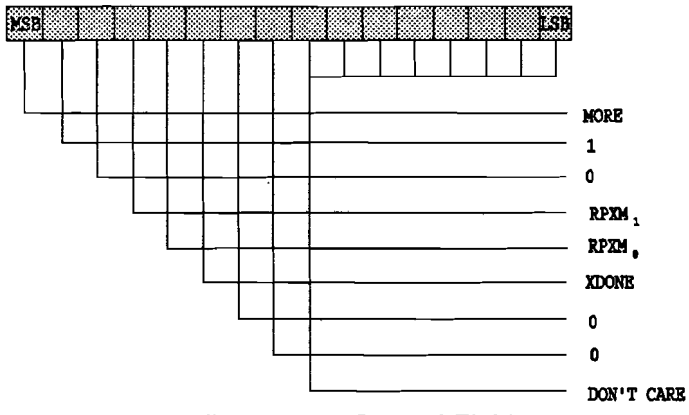


figure 11.4, Control Field

Explanation of the control bits:
- Bit 31, MORE indicates wether or not this is the last frame in the queue.
- Bit 29 and 30 is a fixed pattern.
- Bit 27 and 28, $RPXM_{0-1}$, specify the byte boundary of the first byte of the frame. In some cases, when a header is added to a frame, it becomes necessary to transmit the frame starting on a byte boundary which is not aligned with the 32-bit long word boundary. These bits are used to specify the byte alignment.
- Bit 26, XDONE, this bit is used to limit the number of S-frames transmitted per Token. It is up to the programmer of the node to limit the number of S-frames by setting this bit. A useful limit could be:

  $$\text{Number of frames in chain} = \frac{\text{Allocated Sync bandwidth (ns)}}{\#\text{Bytes in frame} * 80\ \text{ns}}$$

  The last frame in the chain must have the XDONE=1. Chains of frames may be created with XDONE=1 every N frames. Once the transmit sequence starts, all frames up to the frame with the XDONE=1 will be transmitted, followed by a release of the Token. Since the system knows the chain is not complete, transmission automatically continues with the next frame in the sequence when the Token is recaptured. This process is automatically repeated until a frame with the right MORE bit is encountered.
  For A-frames, the XDONE bit is not normally used.
- Bit 24 and 25 is a fixed pattern.
- Bit 16-23 are "don't cares".

Chaining Multiple Frames
Frames in the transmit chain begin with a 32-bit descriptor, followed by the actual frame data, followed by a 32-bit frame pointer to the next frame. The MORE bit must be set in all frames except the last. The XDONE bit should be used to control S-frame transmission. The queued S/A-frame chain will be sent on capture of the next Token.

Length Field
This field contains the number of bytes in the data field of the frame (FC, DA, SA and INFO).

Transmit Frame Data Field
The data field for any transmit frame must include the Frame Control (FC), Destination Address (DA), Source Address (SA) and Information (INFO). The Start of Frame Sequence (SFS), consisting of the Preamble and Start Delimiter, the Frame Check Sequence, and the End of Frame Sequence (EFS), consisting of the End Delimiter and Frame Status are generated by the MAC. When first initializing the transmit queue, a stand-alone transmit frame pointer which contains the 16-bit address of the first frame to be transmitted must be created. The address of the initial transmit frame pointer must be loaded into a ram buffer.

Transmit Frame Pointer Field
The last field immediately following the data field, is the transmit frame pointer. When chaining frames, this field contains the address of the next frame to be transmitted.

Frame Reception

Data is received as frames and placed in prepared locations of a buffer memory. These locations are organized as a logical FIFO.

Operational Modes
There are three basic operation modes of the MAC:
- Initialization mode
- On-line mode
- Loopback mode

Initialization mode
On a software reset, this mode is automatically entered ($MMODE_{0-2}=000$). In this mode, the receive and transmit state machines are locked in the reset condition. The network datapath is locked in a "blind repeat" configuration, see figure 10.7. The MAC address registers can only be written and read proper in this mode.

Although the timer default values can be written in any mode, changing these parameters while on-line may lead to non-deterministic behaviour. The timers can be loaded at any time, and network events can occur asynchronously to the loading of the registers (the user cannot determine whether the timer is operating with the new or old default time).

During initialization mode, no receiver protocol is executed. When the MAC is in this mode, certain ring events such as claim or beacon frames may be missed. Thus a MAC leaving the initialization mode, may not be operating with other than the current ring parameters such as the negotiated token rotation time for the ring. The Claim/Listen and Listen/Beacon instructions do not operate in this mode.

Even though the transmit state machine will not process the information, the internal timers (TRT and THT) continue to operate. This permits rough timer checkout before going on-line. On reset, TVX is loaded with its terminal count value. This timer, during non-initialization operations, will expire in 255

clock cycles. Status is set accordingly. TVX will stop once expired until it is reloaded. TRT and THT are cleared on reset. They expire in 221 BCLK cycles. THT will stop when the terminal count is reached. TRT will reload with the current TMAX value and resume counting.

All counters resume sequencing when reloaded. Reloading can be forced with some specific instructions. These instructions can be used with the timer expiration bits in the status register to verify timer operation against "soft" timers during power on confidence testing of the node hardware. It should also be noticed that the timers are loaded with their default values when the Mode Register MMODE bits are programmed to exit the initialization mode.

On-line mode
When the on-line mode is entered, the MAC performs the operational sequence required for the FDDI. The MAC exits this mode if the MMODE bits are changed or a reset takes place.

Receiver
When on-line the MAC continuously processes the incoming symbol stream of the FDDI receiver state machines. The fundamental actions are frame reception and frame repetition control. The receiver performs also some special functions like outputs for ring statistics, external addressing and providing of general status information.

Frame Reception
Frame reception refers to placing network data on the YR-bus and setting the control signals accordingly. There are four basic actions that can occur when the head of a frame is received from the R-bus:
- Normal frame reception
- Flushed frame. There are two criteria for flushing a frame:
    - At the end of a properly formed frame when the address match criteria have not been met. The address match criteria includes the internal and external address match decisions and the states in the mode register. If no valid address match occurs by the fifth byte of the frame following the SA, this will be indicated. This same indication will occur during repetition of a lost (symbol after the SD is a non-data or non-ED symbol) or stripped (partially repeated, before the ED is repeated idles are transmitted) frame.
    - If the frame ends on an uneven byte boundary.
- Non-reception. In this case the "datavalid" and "receive" indication, and the YR-bus never go active. To activate these signals the receiver state machine must be in SD awaiting state. A SD followed by two data symbols must be received (FC). If FC indicates a claim or token, frame reception will not occur. If all criteria heave been met, the state of the transmitter must be considered. The transmitter must be in the idle or repeat status for frame reception. The only exception is when the FULL strap option is tied high. Finally, reception will not occur if the mode register is programmed to disable receive.
- Frame abort. This occurs when there is an external indication to abort the frame or there is a missframe indication. The abort action also occurs if the MAC enters a non-repeat, non-idle transmit state during frame reception or the disable receive mode register setting is programmed while frame reception is taking place.

Frame Repetition
Frames received on the selected R-bus are repeated on the X-bus. Normally the entire frame is repeated, with the FS modified according to the FDDI specifications. The MAC transmitter can leave the repeat mode for several reasons. The frame received at that time will be stripped. Those reasons could be:
- If the MAC detects a MA=SA, this is the mechanism to remove a frame from the ring.
- Encountering of a lost frame. The byte containing the offensive symbol as well as all subsequent bytes are removed.

- Token capture. The MAC decides to capture a token, this means received tokens will be stripped (only the SD of the token is repeated).

Frame status indicators are repeated, stripped or modified. The MAC will repeat an infinite number of symbols following the frame (provided they are properly formed). The first control symbol encountered must be a "T" in the most significant nibble of the R-bus. This signifies the termination of a frame. If the first control character in the least significant nibble is a "T", the frame has an uneven number of symbol pairs. The octet containing the "T" is repeated, and all subsequent indicators are stripped. In a properly formed frame, the symbol after the "T" is the error indicator. This should be an "S" or "R" symbol, if not an "S" symbol is assumed (the "S" will be recovered), and the MAC will assume that the FS reception is complete.

Aside from the "S" indicator no other indicator will be recovered. If a byte of FS is received with a non "S" or "R" symbol in the most significant or a non "S", "R" or "T" in the least significant nibble, FS processing is completed. That byte and subsequent bytes of status are stripped.

Frames are not repeated when the MAC transmitter is in the transmit data, issue token or beacon states.

Special Function Operation
The MAC processes some additional status information, like the reception of a valid SD, detection of SA=MA, control bits reception and the reception of the first non-status byte. And there is a possibility to allow the user to supplement the MAC's internal address detect logic with additional hardware.

Transmitter
The MAC transmitter controls the timing of node transmission. The transmitter reacts to timer expiration and received MAC frames to queue the claim and beacon frames. The transmitter controls the node's data flow by indicating a token capture. The transmitter makes capture and pass decisions based upon the state of "media" request inputs (i.e. synchronous, asynchronous, etc) and token control register when a token is received.

The MAC allows asynchronous or synchronous transmission, depending on the request and timer values. Only one type of transmission is allowed at a given time. When both asynchronous and synchronous frames may be sent, the synchronous request will take precedence. When synchronous frames are allowed, the THT is held. This prevents S-frames transmission in allocated A-frame bandwidth. The MAC transmitter enters the transmit idle state when reset.

Frame Transmission
If the MAC gets a request to send an A- or S-frame, there will be a delay, due to the required token capture time, between request and acknowledgement. If the media is available the MAC waits for the signal that indicates that the frame on YT is ready. Before this frame is transmitted, the MAC ensures that there are at least 8 bytes of preamble transmitted.

The MAC will encapsulate all bytes input on the YT-bus. An SD is output on the X-bus immediately prior to transmission of the first YT byte. The FCS, if there is no indication in the mode register that indicates otherwise, is appended after the last byte given on YT. After the FCS field, a "T" and "R" symbol are output on the most and least significant nibble of the X-bus, respectively. This is the frame's ED and error status indicator. This is followed by two more "Rs" for the A_- and C_-indicators.

If a transmit abort occurs, the next byte on the X-bus will be an idle. The MAC transmit state machine will be set to idle state.

Recovery Operation
The MAC can go in recovery based upon frames received, instructions from the NP and timer expiration. There are two recovery states specified in FDDI, the claim and beacon state. Claim is used to negotiate the operative time for token rotation and determine which node will issue the token. Beacon is used to guarantee ring integrity by verifying the path of the ring.

Claim and beacon state can be entered at any given time. Once these states are entered, the MAC must respond by placing the corresponding claim or beacon frames on the X-bus. The MAC signals the claim or beacon state so that the proper frame can be queued for transmission.

The MAC can go from claim to beacon state and visa versa. If the MAC exits the claim or beacon state it will go to idle state.

12. Architecture of FDDI

The description of the processes, given in appendix F, will now be mapped into hardware. This is done with the Interactive Design and Simulation System (IDaSS), developed at Eindhoven University of Technology by Ir. A.C. Verschueren. With IDaSS it is possible to design a system in hierarchies, and simulate new components as soon as they are inserted.

The files containing the designs described in this chapter can be found in appendix H.

The implementation examples of PHY and MAC as given in chapters 10 and 11 are given only for extra information; the design described here is started up from the requirements model as found in the PROMOD report (appendix F).

12.1 Constraints of IDaSS

State machine controllers cannot read the contents of a bus or the output of a RAM. This means that in some cases these contents must be read into a register, which takes an extra clock cycle.

The clock cycles are not available, so switching the clock on and off for some blocks is not directly possible. When using state machine controllers for this purpose, a great deal of the flexibility and simplicity of IDaSS is lost.

In stead of using a switched clock (which is a bad habit anyway), the signals that are using a clock frequency are replaced by a register whose contents tell the state machine controller that during the next clock a pulse must be assumed. This does not delay the behaviour of the design more than to place the right contents in the register. Most of the time the loading can be done in a previous clock cycle preserving any delay. This construction is especially used for the strobe signals in the interfaces between MAC and LLC.

12.2 Physical interfaces

The FDDI has to communicate with the outside world (LLC), via an electrical interface on the top side, and a light interface on the bottom side. The data that has to be transmitted, dictate the buses and control signals that have to be present.

Further several interfaces are needed for inside communication.

Control signals are defined as active-low. This is commonly used because of testing purposes. The data signals are defined in normal logic: 1 is high and 0 is low.

Physical interface from MAC to LLC

The physical interface to the LLC consists of two data buses, one from MAC to LLC, and one from LLC to MAC, each combined with control symbols.

The MA_UNITDATA.indication flow transmits its data over a four-bit data bus, combined with a strobe and an active signal (see figure 12.1). Strobe indicates that the data on the data bus is valid. Active indicates that the value on the data bus that is presented, is a symbol of one of the FC, DA, SA or M_SDU fields. As soon as active is deactivated two more strobes are generated, to present the MA_transmission_status (bit 0: frame_good, frame_bad; when frame_bad: bit 1: invalid_FCS; bit 2: length_error; bit 3: internal_error), and the received EAC symbols (bit 0 to bit 2: E, A, C).

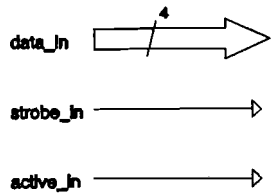


figure 12.1, MA_UNITDATA.indication physical interface

The DA and SA fields are always of equal length, but can be four or twelve symbols each. The LLC must obtain this information from the two FC symbols, that are presented before any address symbol.

The timing diagram is given in figure 12.2.

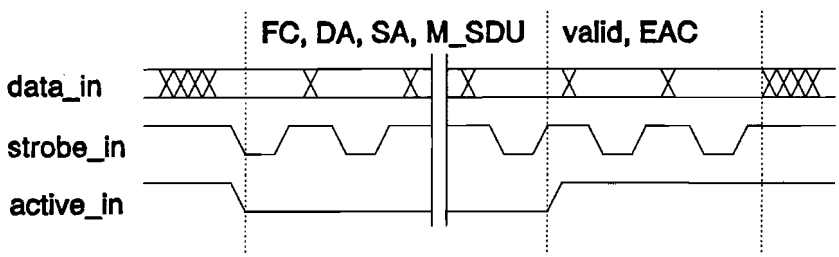


figure 12.2, MA_UNITDATA.indication timing diagram

The MA_UNITDATA_STATUS.indication flow transmits its data over a data bus, and several control signals (see figure 12.3).

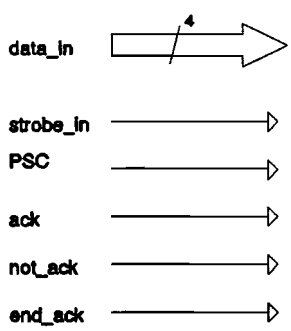


figure 12.3, MA_UNITDATA_STATUS.indication physical interface

Strobe indicates that the data on the data bus and provided_service_class (PSC) is valid, representing respectively the number of SDUs (part of one request) that is transmitted on one token capture and the provided_service_class. These data are connected to the first request that has not yet had a status indication.

Because it is possible that several requests are serviced on one token capture, it may happen that another nr_of_SDUs and PSC is given before all already transmitted frames are acknowledged or not acknowledged.

Ack and not_ack indicate mutual exclusively wether a transmitted frame is acknowledged by the receiver or not (based upon the A and C symbols at the end of the returning frame). Ack and not_ack may not be active at the same time. Acknowledging (or not) is done in the order the frames are received (which is the same as the order they were transmitted and received from the LLC).

When an end_ack comes up, it indicates that no more frames from the last token capture can be expected. This signal is required to make it possible to the LLC to detect lost frames, and can be obtained from token_detect, because no data appears behind a token.

When a request cannot be handled entirely (because of lack of transmission time for a certain frame), nr_of_SDUs is less than the number of SDUs in the request. The rest of the request is discarded and it is left to the LLC to generate another request containing the discarded frames.

The timing diagram is given in figure 12.4.

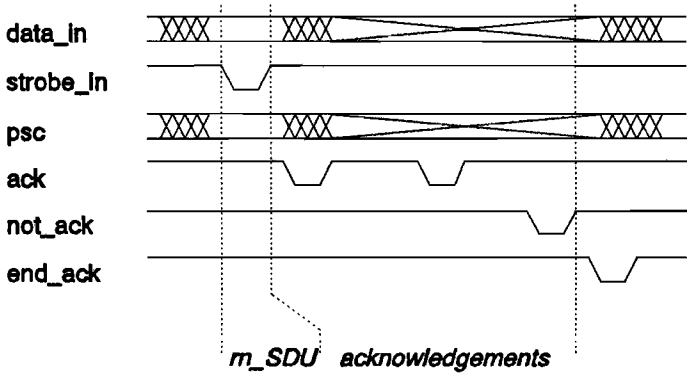


figure 12.4, MA_UNITDATA_STATUS.indication timing diagram

The MA_UNITDATA.indication and MA_UNITDATA_STATUS.indication primitives can both make use of the same data bus and strobe. This is because when these are used for the status indication, the token is captured (which implies that there is no more data to come). When a frame has returned before the token is released, it will not be transferred to the LLC (because it is its own frame).

Both primitives can be distinguished via the active signal indicating that a frame is transferred to the LLC. After it is not active anymore the next two symbols are part of this transfer. When another symbol is transferred while active is not active, it represents a nr_of_SDUs.

This means that at most 15 frames may be part of one request. If nr_of_SDUs = 0, it indicates that a token is captured and immediately another token is issued.

Physical interface from LLC to MAC

MA_UNITDATA.request and MA_TOKEN.request, both use the same four-bit data bus. This is possible because both primitives are generated by the same entity (LLC), guaranteeing mutual exclusiveness. Additional there are some control signals (see figure 12.5).

The MA_TOKEN.request primitive is constructed with data and request. When request is active, this means that the data on the data bus is valid, and that a token must be captured of the requested_service_class with optionally given priority_level (RSC(PL)).

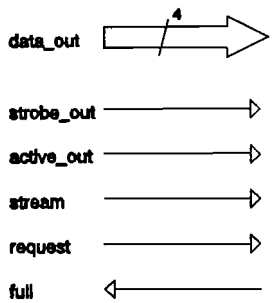


figure 12.5, MA_UNITDATA.request and MA_TOKEN.request physical interface

The MA_UNITDATA.request primitive is constructed with data, strobe, active and stream.

Strobe indicates that the data on the data bus is valid. When active and stream are both active, the symbols transmitted over the data bus represent FC, DA and M_SDU.

The DA field can be four or twelve symbols long. The LLC must provide this information in the two FC symbols, which are presented before any address symbol.

When stream is active and active not, the data represents the requested_service_class combined with a priority level (bit 0 indicates Restricted (0) or Unrestricted (1), in case of Restricted bit 1 to bit 3 indicate the priority level). When both stream and active are not active, each strobe indicates that a token must be captured. Along with this strobe a token_class is transmitted over the data bus. This token_class indicates the type of token that must be issued after all possible frames are transmitted, or immediately, if no frames were specified. It is indicated by bit 0: Restricted 0 and unrestricted 1.

The timing diagram is given in figure 12.6.

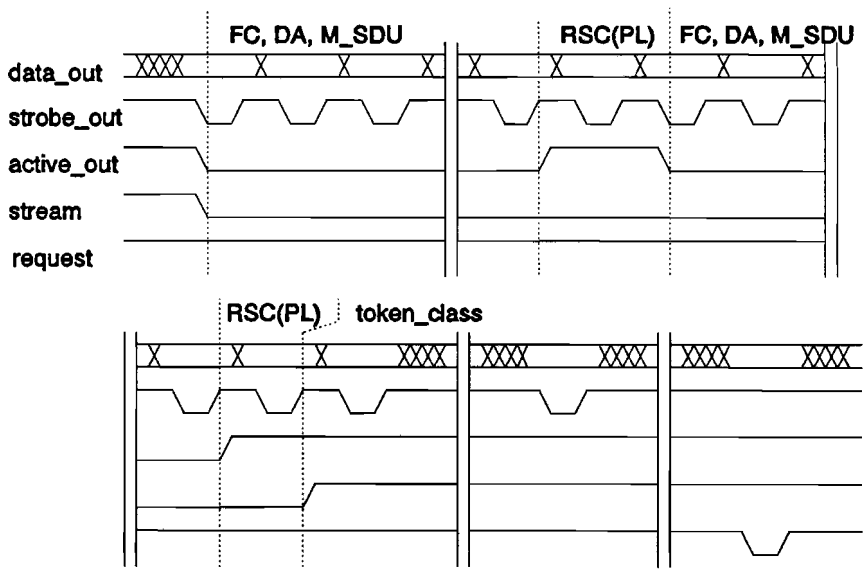


figure 12.6, MA_UNITDATA.request and MA_TOKEN.request timing diagram

Physical interface from PHY to MAC

The interface from PHY to MAC consists of many signals. This is done to ease the detection of symbols at the higher levels, see figure 12.7.

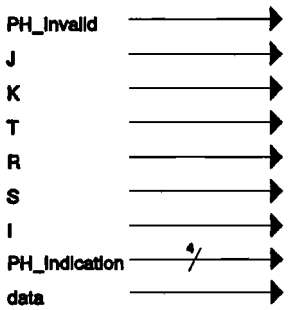


figure 12.7, PH_Indication and PH_Invalid physical interface

The PH_Indication primitive is built up from the following signals: J, K, T, R, S and I, indicating the symbol with the same name, and PH_Indication and data, indicating the normal data symbols 0..F. PH_Indication is only valid when data is active.

The PH_Invalid primitive is mapped directly on one signal. PH_Invalid is activated on a deactivation of Signal_Detect, on overflow or underflow of the elasticity buffer (part of PHY) and on detection of other symbols than can be represented by the interface of the PH_Indication primitive.

Only one of PH_Invalid, J, K, T, R, S, I and data may be active at one moment.

Physical interface from MAC to PHY

This interface consists also of many signals. This is because it eases the generation of these signals at the higher levels, see figure 12.8.

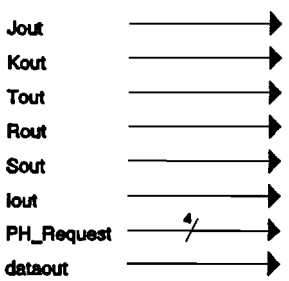


figure 12.8, PH_Request physical interface

The PH_Request primitive is built with Jout, Kout, Tout, Rout, Sout and Iout, indicating the J, K, T, R, S and I symbols to be sent, and PH_Request and dataout, indicating the normal data symbols 0..F. PH_Request is only valid when dataout is active.

Only one of Jout, Kout, Tout, Rout, Sout, Iout and dataout may be active at one moment.

Physical interface between store and add_controls

This interface is needed to transport the data supplied by LLC to add_controls. Add_controls uses only the control signals of the interface, other blocks use the data signals, see figure 12.9.

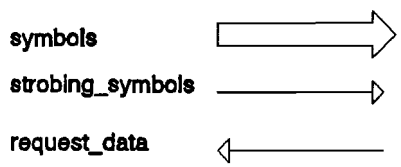


figure 12.9, Physical interface between store and add_controls

On activation of request_data by add_controls, strobing_symbols is activated, and data is put on symbols. This continues until the frame is entirely sent over.

Symbols and strobing_symbols construct the data flow framed_data.

Physical interface between store and monitor

This interface is constructing the control flow frame_class, requesting for an opportunity to transmit, see figure 12.10.

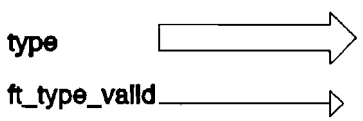


figure 12.10, Physical interface from store to monitor

As soon as a request from LLC has entirely been read, the requested service class of the first frame is put on type and ft_type_valid is activated. When more frames are in this request, ft_type_valid is kept activated. When the last frame in this request is being serviced, ft_type_valid is deactivated.

12.3 Architecture of PMD and SMT

The PMD and SMT are not designed in IDaSS. This is because PMD consists mainly of analogue components, like amplifiers, and SMT is not described in a requirements model because the right standard was not available.

12.4 Partial architecture of PHY

Since the MAC needs some signals of the PHY, it is necessary that the PHY can supply these signals. Because it is rather easy to describe the encode and decode parts of the PHY, these are already described. This simplifies the generation of the signals.

The parallel_to_serial and serial_to_parallel (and synchronize and recover_clock) processes work at a clock rate of 125 MHz, while the encode, decode and MAC processes all work at a clock rate of 25 MHz. When these two clocks are implemented in one IDaSS design, all registers in MAC and encode and decode, must be controlled by a state machine controller, that divides the clock in five and gives commands to the registers at the right moment.

Since the main design is the MAC, it is better to put the processes that need a clock rate of 125 MHz in another design that can be simulated separately, and connected in a later phase of the design. Since the incoming bitstream may arrive at a different clock speed, it may be necessary to take another simulator for this part, while it is very difficult (impossible) to model clock differences of less than 0.01% in IDaSS.

The signals needed by the MAC come directly from the encoder and the decoder. This means that there is no immediate need to describe the other processes of the PHY.

In the design chart, see figure 12.11, no more blocks are drawn than encode and decode with the superconnectors connecting the design to a higher level. This means that there is not yet an interface defined between the PHY and the PMD. However that is not necessary for simulation of the MAC.

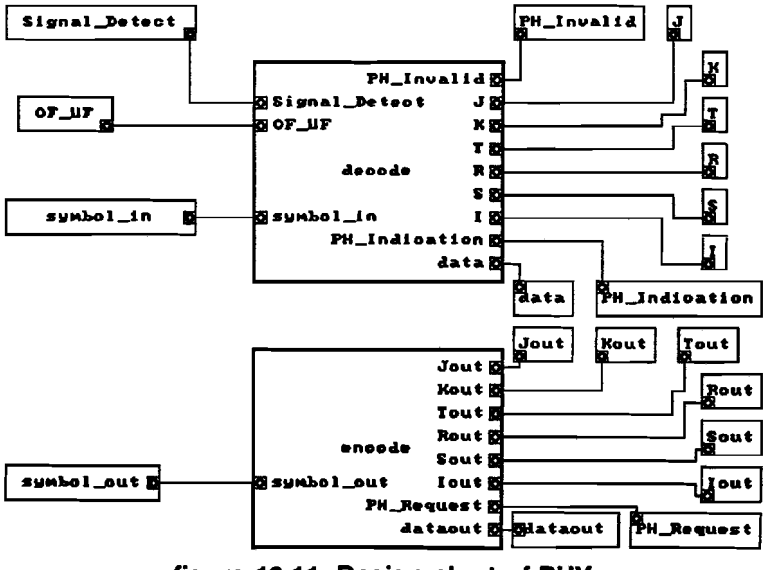


figure 12.11, Design chart of PHY

To simulate the MAC, symbol_out and symbol_in are connected, though this is only useful for checking on the correct operation of MAC on returning frames and not for checking on the behaviour on other frames. Therefore two or more FDDI designs should be linked together via symbol_in and symbol_out. This implies that symbol_in and symbol_out must be taken to a higher level. Because IDaSS is only working on one central clock, it is not useful to check the synchronize and recover_clock processes of the PHY.

The dummy design is needed because of the way IDaSS can model a multiplexer. As soon as the design decoder is designed at a lower level, this dummy design may be omitted.The process decode is mapped to a design decoder combined with a design dummy, see figure 12.12.

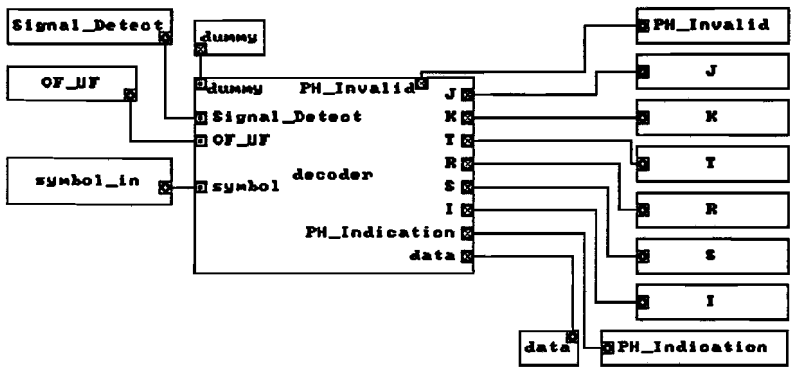


figure 12.12, Design chart of decode

Design decoder contains no more than a functional description of the behaviour (with if..then..else constructions).

The other blocks (with one connector) are superconnectors, connecting this design to a higher level design.

The process encode is mapped to hardware in the same way as the process decode, see figure 12.13. Encoder consists of only a functional description. When encoder is designed at a lower level, the design dummy may be omitted.

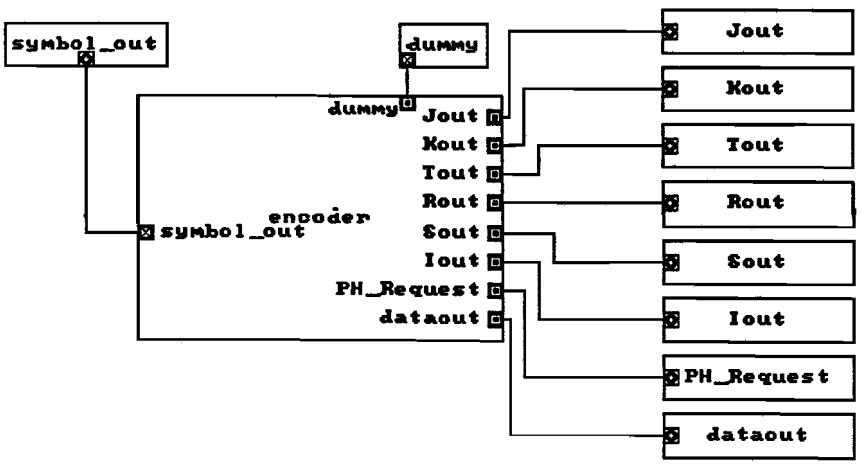


figure 12.13, Design chart of encode

The IDaSS descriptions of operators encoder and decoder can be respectively found in appendices J and I.

12.5 Partial architecture of MAC

The MAC consists of five processes, of which process_userdata is partially designed in IDaSS. This process with reveive_frames and monitor are the most important of MAC. A design of process receive_frames is given in 7.

In chapter 9 two state machines are given. These will be part of the process monitor, and so are not used in process_userdata and receive_frames.

12.6 Partial architecture of process_userdata

As found with the requirements model, the MAC consists of several processes. These can be ented on design blocks accompanied by state machine controllers. The design is shown in figure 12.14.

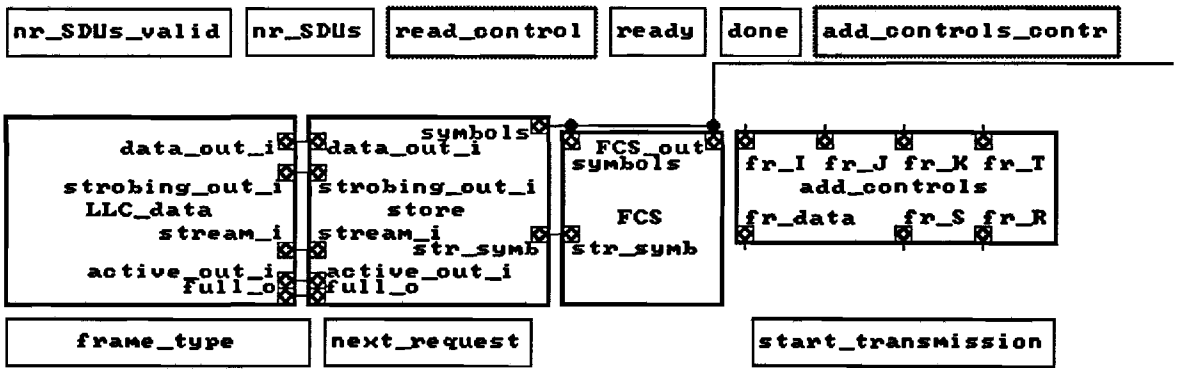


figure 12.14, Design chart of process_userdata

The following registers are used for generating control signals:

done (initialiy 1, inactive): Activated when a frame has entirely been sent.
frame_type (two bits, initially 01): Can be one of:

00	token
10	claim
11	beacon
01	other_type

next_request (initially 0, active): When active, a request may be initiated and nr_SDUs may be sent; when inactive, no frames may be transmitted. This is to make it possible for LLC to detect of which request a frame is lost.
nr_SDUs_valid (initially 1, inactive): Activated when the data of one request are transmitted, so that nr_SDUs can be read. This number may be directed to another entity than LLC, so LLC must always use the last value that appeared on the bus.
ready (initially 1, inactive): When activated, the request has entirely been sent, or the last frame of this opportunity has been sent.
start_transmission (initially 1, inactive): active when a request to transmit can be honoured; it is read at the end of the transmission of a frame.

The process store is made with the blocks store and read_control. Read_control is a state machine, that cannot be placed in the block store, since it makes use of register values that are found in the block add_controls (IDaSS description of read_control can be found in appendix K).

The process add_controls is made with the blocks add_controls, add_controls_contr and FCS. Add_controls_contr is a state machine that cannot be placed in the block add_controls, since it makes use of register values in the block process_userdata and the block store, and controls some parts in the block FCS (IDaSS description of add_controls_contr can be found in appendix L).

Done is an indication that the current frame is entirely sent (including control symbols), and a new frame can be requested for. Ready is an indication that the last frame of this opportunity is sent (end of request, end of time).

Nr_SDUs is a block that contains the number of frames of one request that is sent. When nr_SDUs_valid is active, the value of nr_SDUs can be read by other blocks; it stays active during several cycles.

FCS is the block that generates the frame check sequence.

LLC_data is a block that is only inserted for testing purposes. As soon as the testing is done, it can be placed in a higher level of the total system.

Start_transmission and frame_type are blocks that should be placed in a block monitor, that is not designed yet (monitor generates these control signals). Next_request is a block that should be placed in the block indicate, that is also not designed yet.

12.7 Partial architecture of process store (part of process_userdata)

Store is the process that reads the data supplied by the LLC, and remembers it until an opportunity is offered to send data. By then it must release the data, and erase it from the memory. To be able to do this, a large memory is required, to store the data itself, and several other memories are required to be able to find back the frames and requests.

This asks for a levelled design, and a levelled addressing. The idea for the addressing is shown in figure 12.15.

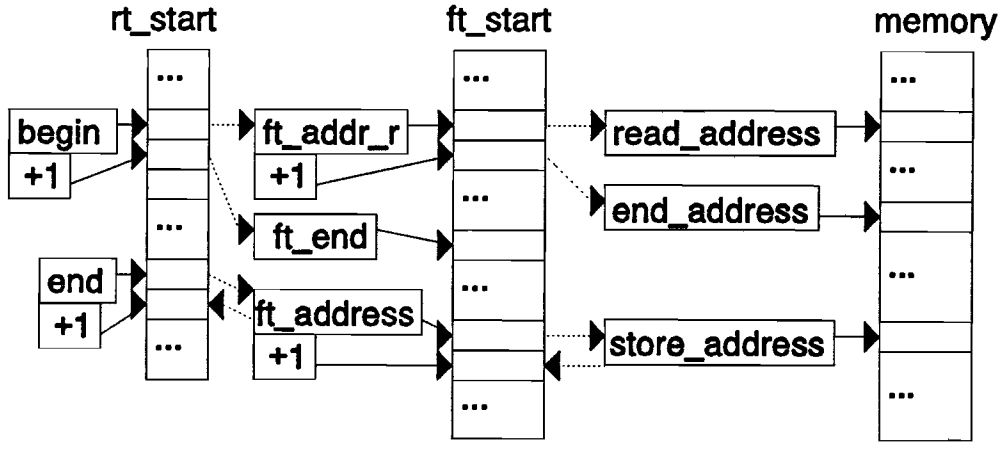


figure 12.15, Structure of the memory manager for LLC data

To store in the memory only store_address and memory are needed, the other blocks are required for finding the right data back.

Rt_start = Request table start address
Ft_start = Frame table start address

Parallel to the two tables, two other tables are used:
Rt_type = Request table token_class (class of token that has to be issued when the request has been serviced)
Ft_type = Frame table requested_service_class (class of service that is requested for this frame: asynchronous with priority or synchronous)

Initial state:
begin = 0
end = 0
rt_start[0] = 0
ft_address = 0
ft_start[0] = 0
store_address = 0

Storing LLC data in memory

To store data in memory, store_address is incremented modulo the length of the memory every time a symbol is written. When an entire frame is written to memory, the contents of store address are written to ft_start[ft_address+1] (the requested_service_class is written to ft_type[ft_address]), and ft_address is incremented modulo the length of ft_start (ft_address+1 is also calculated modulo the length of ft_start).

This is repeated for every frame in one request.

When all frames of a request are written in this way, the contents of ft_address are written to rt_start[end+1] (the token_class is written to rt_type[end]) and end is incremented modulo the length of rt_start (end+1 is also calculated modulo the length of rt_start).

This incrementing is done only when it does not make end equal to begin, because begin = end indicates that there are no requests in memory. By then the incrementing is postponed until a request is entirely read from memory, and begin is incremented.

In this way requests can be written to memory, as long as none of the tables and the memory is filled up.

In case an error on the controls from LLC or a fill-up occurs, the request is erased from the memory by firstly loading ft_address with rt_start[end] (address that ft_address had when the request started), and after this loading store_address with ft_start[ft_address] (address that store_address had when the request started).

Reading the frames from memory

To read the frames from the memory again, ft_addr_r is loaded from rt_start[begin] and ft_end is loaded from rt_start[begin+1]. Thereafter read_address is loaded from ft_start[ft_addr_r] and end_address is loaded from ft_start[ft_addr_r+1], so the begin address and the end address + 1 of the first frame is

known. When the contents of ft_type are not needed anymore, ft_addr_r is incremented and if ft_addr_r ≠ ft_end another request to transmit can be prepared.

By incrementing read_address the frame can be read from the memory, which is done until read_address = end_address. When this happens and another frame can be transmitted (it is available in this request, and timing allows it), end_address is loaded with ft_start[ft_addr_r+1] (read_address is already loaded with the right value via the increments).

When no frames may or can be transmitted anymore, begin is incremented, so the rest of the request is discarded.

Each of the three addressing levels is implemented as a circular buffer, that behaves like a bounded FIFO. In IDaSS the depth of the memories is limited to 2048 addresses, so memory has got that size. For simulation and evaluation purposes this satisfies, but in the eventual implementation this memory must have a size of about:

{depth of rt_start} * {mean number of frames per request} *
{mean number of symbols per frame}

The depth of ft_start must then be about:

{depth of rt_start} * {mean number of frames per request}

In the simulation the following figures are taken:

depth of rt_start	= 8
depth of ft_start	= 32 (mean number of frames per request = 4)
depth of memory	= 2048 (should be larger in the eventual implementation)

The eventually used figures will depend on the use of the system.

Implementation in IDaSS

The first level of design is shown in figure 12.16.

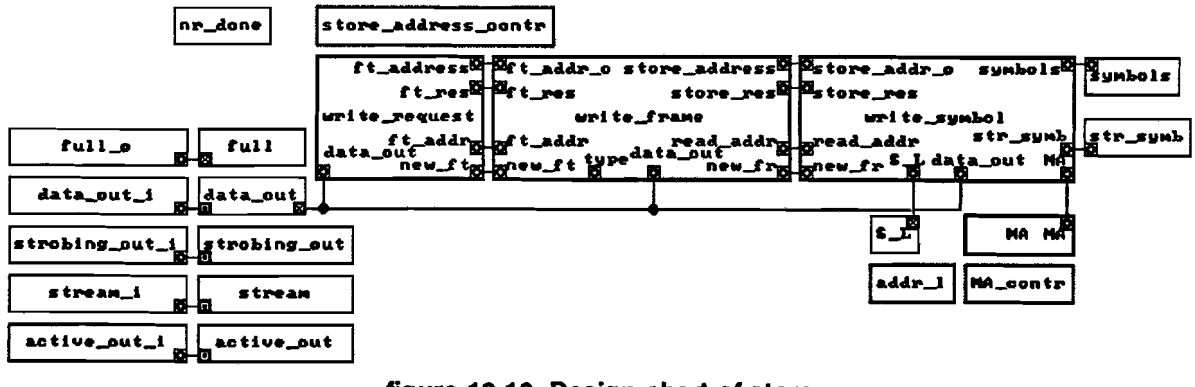


figure 12.16, Design chart of store

The second level of design is the design of several blocks, that will be described later.

In figure 12.16 the three addressing levels are found: Write_request, write_frame and write_symbol. Each block is a design, and will be described later.

Store_address_contr is a state machine that controls (using nr_done) the storing of the frames in the memory (IDaSS description of store_address_contr can be found in appendix M). It controls all three addressing levels and so cannot be placed in a lower level design.

S_L and addr_l scan for the length of the address of the incoming frames (IDaSS description of addr_l can be found in appendix N).

MA and MA_contr supply the address of the station, that **must** be available, according to the required length (IDaSS description of MA_contr can be found in appendix O). It is a simple block as can be seen in figure 12.17, consisting of and addressing register (MA_address), and a memory containing the address (MA_mem).

figure 12.17, Design chart of MA

Full is a block that indicates that there is no more space to store a request, a frame or a symbol. The request that is currently sent, is erased from the memory. When only the request table is full, the last request is not erased. The latter case can be distinguished from the others, because full is activated after both stream and active_out are deactivated by the LLC.

The registers stream, active_out, strobing_out and data_out are fed via superconnectors by LLC_data. These registers are needed for IDaSS, since state machines in IDaSS cannot read the contents of a bus.

12.8 Architecture of write_symbol (part of store)

Write_symbol is the third addressing level, that contains the memory that must remember the SDUs supplied by the LLC, accompanied by the address supplied by MA (in the field of SA), See figure 12.18.

The following register is used for generating a control signal:

strobing_symbols (initially 1, inactive): Activated on request_data (from add_controls), and frame_type = other_type, along with enabling the tri-state output symbols_d.

The memory has two addressable write ports, and one addressable tri-state read port. The write ports are used to get the data from LLC and MA in the memory, and the write port is used to read the data from the memory. It is made tri state to easily add other data (like the FCS) to the data read from the memory. When reading from the memory, strobing symbols indicates that the data is valid.

Store_address is the register that contains the address that data from the LLC must be written to. It has an input, that is fed via a mux. The mux (IDaSS description in appendix P) switches between a restore address (store_res, when something has gone wrong and the request must be erased), and a continue address (MA_address, to continue addressing after the address fields are placed in the memory). The

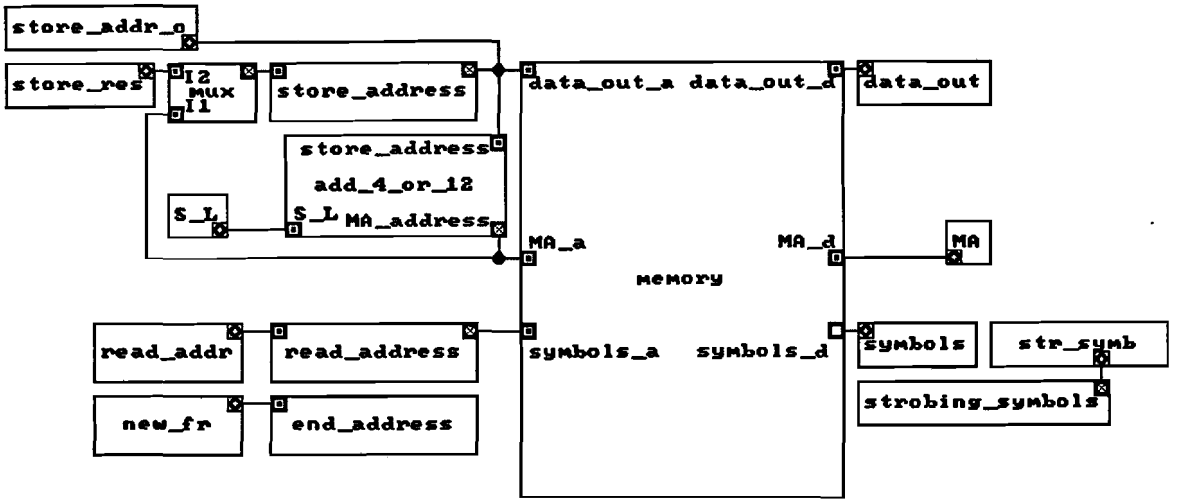


figure 12.18, Design chart of write_symbol

output is also used by the second level addressing, to remember the end of the frames.

Destination Address (DA, from LLC) and source address of this station (MA) can be written simultaneously, making addressing easy, and saving some time. It is done by using two write ports that can be independently addressed. To make the simultaneous addressing possible the operator add_4_or_12 (IDaSS description in appendix Q) is inserted, that outputs the address MA must be written to. When the addresses are entirely written, the output of add_4_or_12 is loaded in store_address to place the rest of the incoming data from the LLC behind the address fields.

Read_address points at the symbol of the frame that is output to symbols_d. End_address points at the place behind the last symbol of that frame. As long as read_address does not reach end_address, this process continues. When at the end of a frame, read address already points at the first symbol of the next frame in the same request, so only end_address has to be updated. When a new request is beginning to be serviced, both registers must be loaded, since it may be possible that some frames of the previous request have not been sent.

12.9 Architecture of write_frame (part of store)

Write_frame is the second level of addressing. It will contain the start address of each frame in memory, and the type of that frame. Therefore two memories are used, see figure 12.19.

The following register is used for generating a control signal:

ft_type_valid (initially 1, inactive): activated to indicate a request to transmit. Only activated when a request from LLC has entirely been loaded.

Ft_type is used to store the type of the frame, and to generate the request with. Monitor bases its decision about sending or not on this type.

Ft_start contains the start addresses of the frames in the memory.

When accepting frames from LLC, the start address of the next frame is stored on address ft_address+1, while the type of the frame is stored at ft_address. For restoring the store_address, the

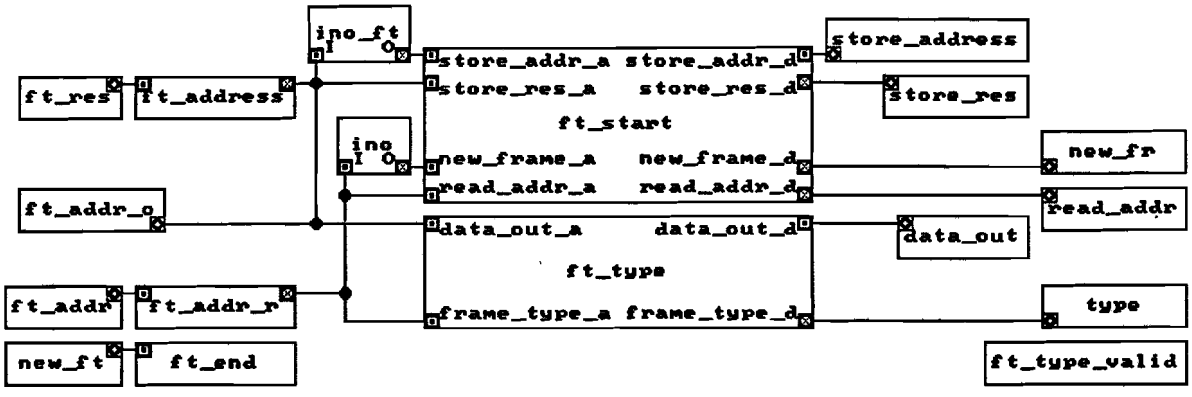


figure 12.19, Design chart of write_frame

address at ft_address is used.

To read the information, the end address of the frame is read from ft_addr_r+1, while the begin address is read from ft_addr_r. The type at place ft_addr_r is used for generating the request for transmitting this frame.

12.10 Architecture of write_request (part of store)

The following states are used by state machine read_control:

begin = end (initial state): Indicates that no requests from LLC are entirely loaded in the memory.
begin ≠ end: Indicates that at least one request is entirely loaded in the memory.

Write_request is the first level of addressing. It will contain the start address of each request in ft_start, and the token_class that must be issued after that request. Therefore two memories are used, see figure 12.20.

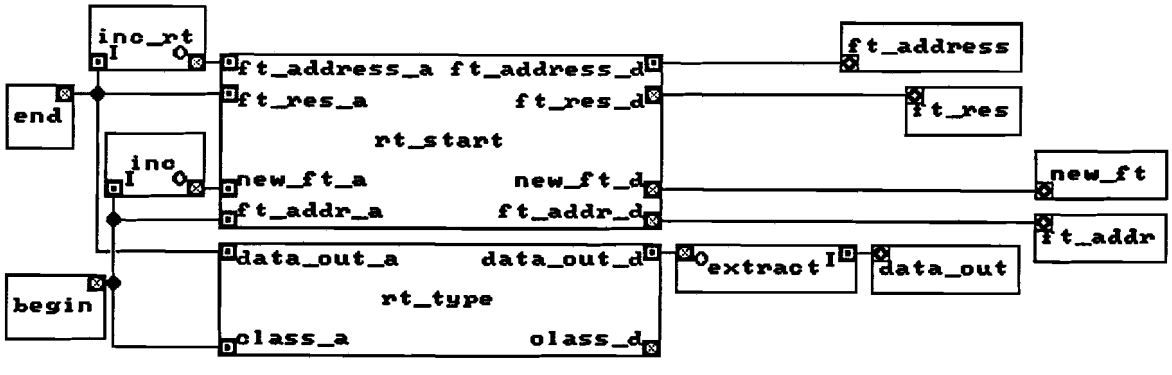


figure 12.20, Design chart of write_request

Rt_type actually is part of the monitor, but since the addressing is the same as that for rt_start, it is placed in this block.

When a request has ended, ft_address is stored at address end+1, while the token_class is stored at address end. Hereafter end is incremented. When something has gone wrong, so that a request must be erased, the restore ft_address is ready at ft_res, at address end.

Extract takes bot 0 for token_class from the data bus.

To start sending a request, the addresses in the second stage have to be updated. They are given via new_ft and ft_addr, at address begin+1 and begin. The token class required by the monitor appears at class_d. When the entire request is serviced, or the time to send is over, begin is incremented, but not sooner than that the current frame has been entirely sent.

12.11 State machines used for process store

State machine read_control (IDaSS description in appendix K)

Used to read data from memory to add_controls.

1. wait_until_request
   Waiting until a request has fully been entered in memory (this is detected when begin ≠ end). When that happens, ft_addr and ft_end can be loaded from rt_start (they are valid now).

2. request_ready
   There is a complete request.
   Since ft_addr_r is read, read_address and store_address can be read from ft_start (they are valid now), and a request to transmit may be given via frame_class (ft_type_valid), because ft_type is valid.

3. request_to_send
   The request to transmit is given, ft_start is loaded into read_address and ft_type is read by the monitor.
   This means that the contents in the memories pointed at by ft_addr_r are not needed anymore, so ft_addr_r can be incremented. When there are no more frames in this request (detected by ft_addr_r = ft_end-1) ft_type_valid **must** be deactivated.

4. wait_for_request_data
   The monitor handles the request, and tries to capture a token. When this happens, it indicates that the transmission may start, and add_controls starts the transmission. When add_controls needs the data in memory, it activates request_data.
   Now begin may be incremented since the rt_type is read by the monitor at token capture, but this may be done only one time per request, so it is done at the end of the transmission of the request.

5. produce_frame
   The tri-state output of memory (symbol) is enabled and the data is sent to the bus until read_address = end_address-1. When this occurs there are two possibilities:

a. There is another frame in this request (ft_addr ≠ ft_end) and transmission is allowed (start_transmission = 0):
The last symbol is written to the bus: After that symbols_d is made tri-state again, the new value for end_address is loaded (read_address is already correct via the reading of the previous frame).
There must be waited until the frame has entirely been sent.

b. There is no other frame to be sent, or no time is left.
Then begin may be incremented, the last symbol is sent after which symbols_d is made tri-state again.
There must be waited until the request (frame) has entirely been sent.

6. hold_until_frame_done
There must be waited until the frame has entirely been sent.
When that is done, the request for another frame is approved by the monitor via start_transmission. This means that ft_type is not needed anymore and another frame can be requested for (by incrementing ft_addr_r) if there is any, or else ft_type_valid can be deactivated. The latter case causes start_transmission to deactivate, but does not interfere with the frame currently sent, since this transmission is already initiated.

7. hold_until_request_done
To make sure that no new request is initiated while the current is still running (causing another request to be sent, while the current request is not fully acknowledged), there must be waited until this request is ready.

State machine store_address_contr (IDaSS description in appendix M)

Used to store data from LLC in memory.

1. wait_and_addressing_till_data
In this state there is space available to store at least one request address, one frame address and one symbol. When data is supplied by LLC, it is stored in memory until the address fields are stored (the length is determined from the frame control).

2. addressing_data
The data after the address fields is stored in memory.
When no more data for this frame comes through, the requested_service_class reception is initiated.

3. next_frame
When a new frame comes up, the reception is initiated, and state wait_and_addressing_till_data is entered, else token_class is read and state wait_and_addressing_till_data is entered. If in either case the available space is entirely used, the state wait_until_free_space is entered and full is indicated to the LLC.

4. wait_until_free_space
This state is kept until in all three addressing stages is at least one place available. When that happens the state loose_request is entered, to erase the latest request in memory.

5. loose_request
Ft_address restore is initiated.

6. restore_frame_address
Store_address restore is initiated.

7. wait_until_end_request
   There is waited until the current request has ended. This is needed because it may happen that LLC does not react to the signal full.

The construction to loose the request takes three states, and so needs three clock cycles. This means that if an error occurs, the recovery time is at least three clock cycles.

12.12 Architecture of block add_controls

Add_controls add the control symbols to the frame to make it complete. It is shown in figure 12.21.

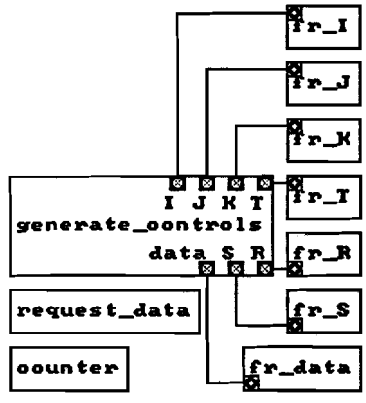


figure 12.21, Design chart of add_controls

Register used for generating a control signal:

request_data (initially 1, inactive): Indicates that the preamble and start delimiter have been sent, and data from store is required.

Generate_controls is an operator that supplies the symbols that are used to complete the frame (IDaSS description in appendix R).

Counter is used to count the length of the preamble and the number of R symbols in the frame status.

12.13 State machine used for process add_controls

State machine add_controls_contr (IDaSS description in appendix L)

1. wait
   Waiting until start_transmission is activated. If so the preamble is initiated.

2. PA
   Makes the entire preamble of 16 I symbols, and thereafter initiates the J symbol of the start delimiter and initiates request_data for store.

3. J
   Initiates the K symbol of the start delimiter.

4. K
   Waits until strobing_symbols is inactive, and dependent on frame_type enters state add_2_T (frame_type = token) initiating a T symbol, or add_FCS (else) initiating the transmission of the frame check sequence.

5. add_2_T
   Initiates the second T symbol and enters state wait.

6. add_FCS
   Finishes adding the frame check sequence, and enters state ED initiating a T symbol and resetting the FCS block.

7. ED
   Adds three R symbols for the frame status, and enters state wait.

12.14 Architecture of FCS (part of process_userdata)

FCS is placed outside block add_controls that takes care of adding the frame check sequence because it is a directly controlled by state machine add_controls_contr.

In process_userdata all data handling is based on the four bit symbol. The circuit to determine the frame check sequence, given in the MAC standard [4] and shown in figure 3.1, is based on a one bit input.

Two options are open to solve this problem. One is to serialize the symbols to bits. This requires a four times faster clock along with the normal clock, and logic to execute the conversion to and from serial bits. This may have the disadvantage of unsynchronized clocks, and trouble to have the frame check sequence in time to be put immediately behind the data.

Another possibility is to design another circuit that accepts four bit input. This has the advantage of having no trouble with synchronisation of two clocks and the two data streams.

Since IDaSS does not support multiple clocks in one design, and because of the disadvantages of the first method, the second option is chosen.

First is determined under what condition it is possible to use this second option.

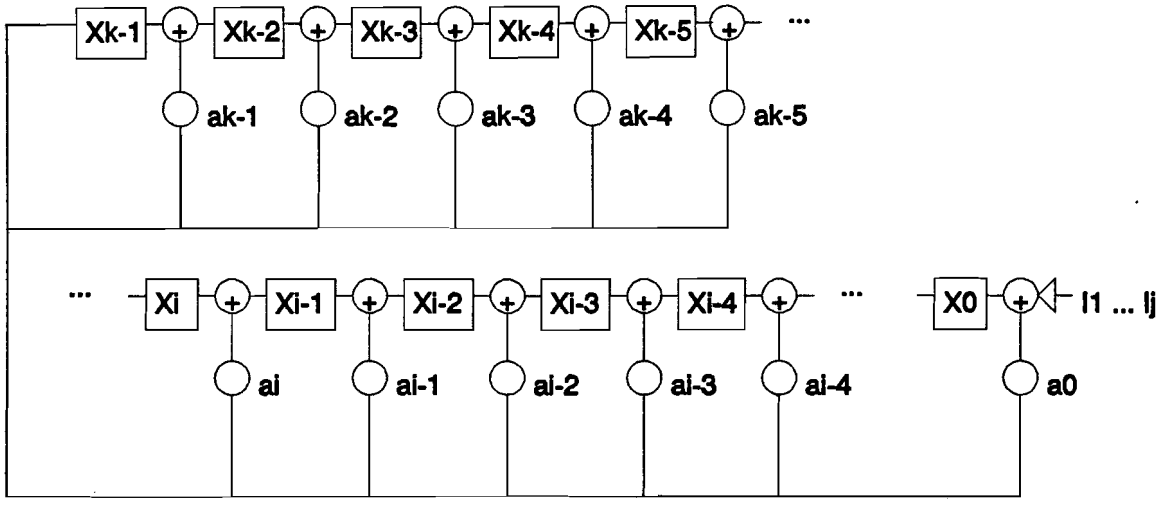


figure 12.22, General circuit of a feed back shift register

Theorem: Given a circuit of a feedback shift register, like in figure 12.22, with I_1 to I_j known, the value of X_i at time t can be expressed as a function of X_{i-j}, X_{k-1} till X_{k-j} and I_1 till I_j at time t-j.

Note: All additions are done modulo 2.

Proof:

$X_i[t] = X_{i-1}[t-1] + a_iX_{k-1}[t-1]$
$X_i[t-1] = X_{i-1}[t-2] + a_iX_{k-1}[t-2]$
$X_i[t-i] = X_{i-1}[t-i-1] + a_iX_{k-1}[t-i-1]$

$X_i[t] = X_{i-2}[t-2] + a_{i-1}X_{k-1}[t-2] + a_iX_{k-1}[t-1]$
$X_i[t] = X_{i-3}[t-3] + a_{i-2}X_{k-1}[t-3] + a_{i-1}X_{k-1}[t-2] + a_iX_{k-1}[t-1]$

In general:
$X_i[t] = X_{i-j}[t-j] + \sum_{m=0}^{j-1} a_{i-m}X_{k-1}[t-1-m]$

$X_{k-1}[t-q] = X_{k-1-j+q}$ given $a_{k-1} \dots a_{k-j+1} = 0$, $X_{-L} = I_L$

Giving:

$X_i[t] = X_{i-j}[t-j] + \sum_{m=0}^{j-1} a_{i-m}X_{k-j+m}[t-j]$

When $a_{k-1} \dots a_{k-j+1} \neq 0$, this formula is not valid, but still the value of a certain cell can be expressed as a function of the values of the other cells and some inputs at some previous time.

$X_{k-p}[t] = X_{k-p-1}[t-1] + a_{k-p}X_{k-1}[t-1]$
$X_{k-p}[t-1] = X_{k-p-1}[t-2] + a_{k-p}X_{k-1}[t-2]$
$X_{k-p}[t-2] = X_{k-p-1}[t-3] + a_{k-p}X_{k-1}[t-3]$

$X_{k-1}[t-2] = X_{k-2}[t-3] + a_{k-1}X_{k-1}[t-3]$
$X_{k-1}[t-1] = X_{k-3}[t-3] + a_{k-2}X_{k-1}[t-3] + a_{k-1}X_{k-2}[t-3] + a_{k-1}X_{k-1}[t-3]$

As can be seen from these formulas:
$X_{k-p}[t] = F(X_{k-1}[t-j], \dots, X_{k-j}[t-j], X_{k-p-1}, I_1, \dots, I_j)$

Consequence:
$X_j[t]$ can be calculated from values stored at time t-j, and so can be calculated in one cycle. This means that the circuit in figure 12.22 can be parallelized to j bits. The polynomial used in FDDI is:

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

Implemented in a feedback register a_{31} till a_{27} are equal to zero, and no special tricks have to be used.

In this case the following feedbacks must me used:
$x^{32} = x^{26} + x^{23} + x^{22} + x^{16} + X^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
$x^{33} = x^{27} + x^{24} + x^{23} + x^{17} + X^{13} + x^{12} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^3 + x^2 + x$
$x^{34} = x^{29} + x^{25} + x^{24} + x^{18} + X^{14} + x^{13} + x^{12} + x^{10} + x^9 + x^7 + x^6 + x^4 + x^3 + x^2$
$x^{35} = x^{29} + x^{26} + x^{25} + x^{19} + X^{15} + x^{14} + x^{13} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^3$

When a_{k-1} till a_{k-j+1} would not be equal to zero, the previous method would result in terms of grade 32 or higher (x^{32}, x^{33}, etc) on the right side of the equal sign. These can be replaced by their equals as accounted with the previous method.

Example with grade 3:
$G(x) = x^3 + x + 1$
This means in the circuit: $x^3 = x + 1$

To parallelize this to three bits, the following feedbacks must be constructed:
$x^3 = x + 1$
$x^4 = x^2 + x$
$x^5 = x^3 + x^2$

X^3 already exists at the left side of the equality signs so an extra conversion is used:
$x^5 = (x + 1) + x^2 = x^2 + x + 1$

Note: The constraint to the parallelization is the number of bits that can be parallelized: That must be a divider of the grade of the generator polynomial.

Implementation in IDaSS

Since the context of FCS is using four bit wide symbols, the circuit is designed in four bits. It is shown in figure 12.23.

It consists of 8 four bit registers (FCS0 to FCS7), that are fed back via operators (fbd0 to fdb7). These operators connect the outshifting values (x^{32} to x^{35}) and the outputs of the previous register in the right way to the inputs of the register it is connected to (IDaSS description of the operators can be found in appendix T).

The registers are controlled via a control connector that is connected to strobing_symbols in the block process_userdata\store\write_symbols and a register reset. The former is used to be able to start soon enough with the frame check sequence generation (all registers are set to load). The latter is used to reset the registers to F again when the frame check sequence has been read out. Reset is controlled by add_controls_contr.

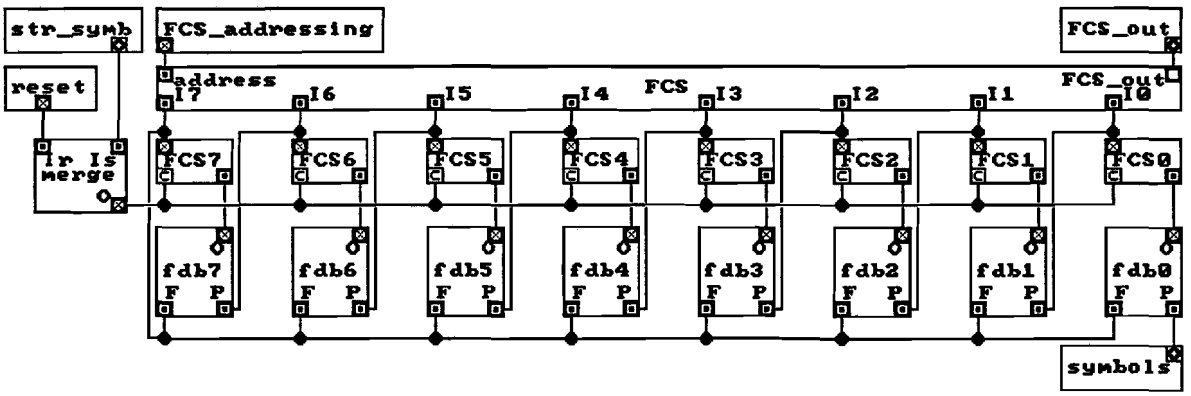


figure 12.23, Design chart of FCS

FCS is an operator that addresses the outputs of the registers when the frame check sequence has been made, to direct it behind the data (IDaSS description in appendix S). It is steered by register FCS_addressing, that is controlled by add_controls_contr.

12.15 Further development of process_userdata

Not all processes of process_userdata have been implemented yet. Compose and indicate are not watched at and store must be extended with the possibility of accepting data from SMT and MAC.

12.16 Note to the design process

As said before, the design process has been started from the requirements model. While designing, some descriptions in the requirements model appeared to be inefficiently implementable. To make the design efficient anyway, the requirements model has been adapted. These were only minor changes, that did not change the essence of the model.

13. Conclusions and recommendations

FDDI can be used for different purposes, like back-end, backbone and front-end networks. For all these purposes enough bandwidth is available, since the defined bandwidth of FDDI is 100 Mbps.

Normal FDDI cannot be used for circuit switched services, so another standard is defined called FDDI-II. This allows for suitable-bandwidth circuit switching, combined with packet switching. Because FDDI-II merges the data and voice (video) services, it has a major power to become a great success.

Fiber offers a high security degree, but the operation of FDDI on it, makes FDDI likely to be used only for private owned networks.

FDDI is a perfect solution for LAN sites, but for long distance data transfer (possibly interconnecting FDDI networks) DQDB networks (B_ISDN) are more likely to be used.

The FDDI follow-on needs efficient connections to future WANs, guaranteeing an effective use of FDDI networks. With the development of FDDI follow-on one has to keep in mind that the offered basic bandwidth has to meet the American, European and Japanese long distance standards (this will probably be about 600 Mbps).

At this moment FDDI networks are mainly used as a backbone to interconnect LANs. However there are a more and more networks that are used to interconnect supercomputers, so called back-end (for instance with NASA, Freedom and USA Defence), for which FDDI initially was developed.

Though many other LANs are reaching their boundaries, the step to FDDI is still too expensive, as well in the economical as in the technical sense. These costs can be decreased via decreasing material costs and "economies of scale".

Many LAN solutions can be found in usage of bridges and routers between the different domains like IBM 3090, VAX and HP. Only 10% of the users wishes a connection with another domain than his own. To them FDDI will possibly offer an economical acceptable solution.

On the international LAN markets there are some saturation developments. There are to many suppliers and distributors on a to slowly growing market. This results in minor investments in new products, that in turn slows down the market growth. As a result many suppliers and distributors cooperate or merge into new companies.

A major problem at this moment is to create services (demands) that really need the FDDI bandwidth and equipment that can handle this bandwidth, this has to be accomplished to make FDDI a great success.

All FDDI standards are defined in agreement with the ISO/OSI definitions, and cover the physical layer and the lower part of the link layer protocol. A station management is defined through both layers. This makes FDDI a useful standard, it allows different suppliers to produce FDDI products that can easily cooperate with other parts of the FDDI network. This eventually creates a competitive market, decreasing the price.

I am convinced that FDDI is already a success, on the other hand I also believe that the need for FDDI is not yet optimal.

In the ANSI documents, a basis is given for implementing the entities. The actual implementation however is still free except for the external behaviour of the entities.

With the method described in [1], a requirements model has been developed using PROMOD (a program that supports the method). The accent has been laid on the MAC entity because that seemed

to be the most interesting part besides the SMT. Discussions about specifications of processes appeared to be very useful to obtain a good model. The SMT entity is only slightly viewed at, because no up to date standard was available.

Going out from the requirements model, some processes have been designed using IDaSS. This resulted in encode and decode IDaSS descriptions, and the possibility to receive frames from the LLC and construct frames from it.

Because of the other point of view that is used to design hardware, the requirements model had to be adapted at some points. These were only minor changes, that did not change the essence of the model, so working via the requirements model is very satisfying.

In this report a major introduction has been given. This is done because the internal operation of FDDI is not very much known about in the Netherlands at this time.

With use of this introduction and more literature (especially a standard on the SMT), parts of the requirements model can be further developed to the level that a design is easily obtained.

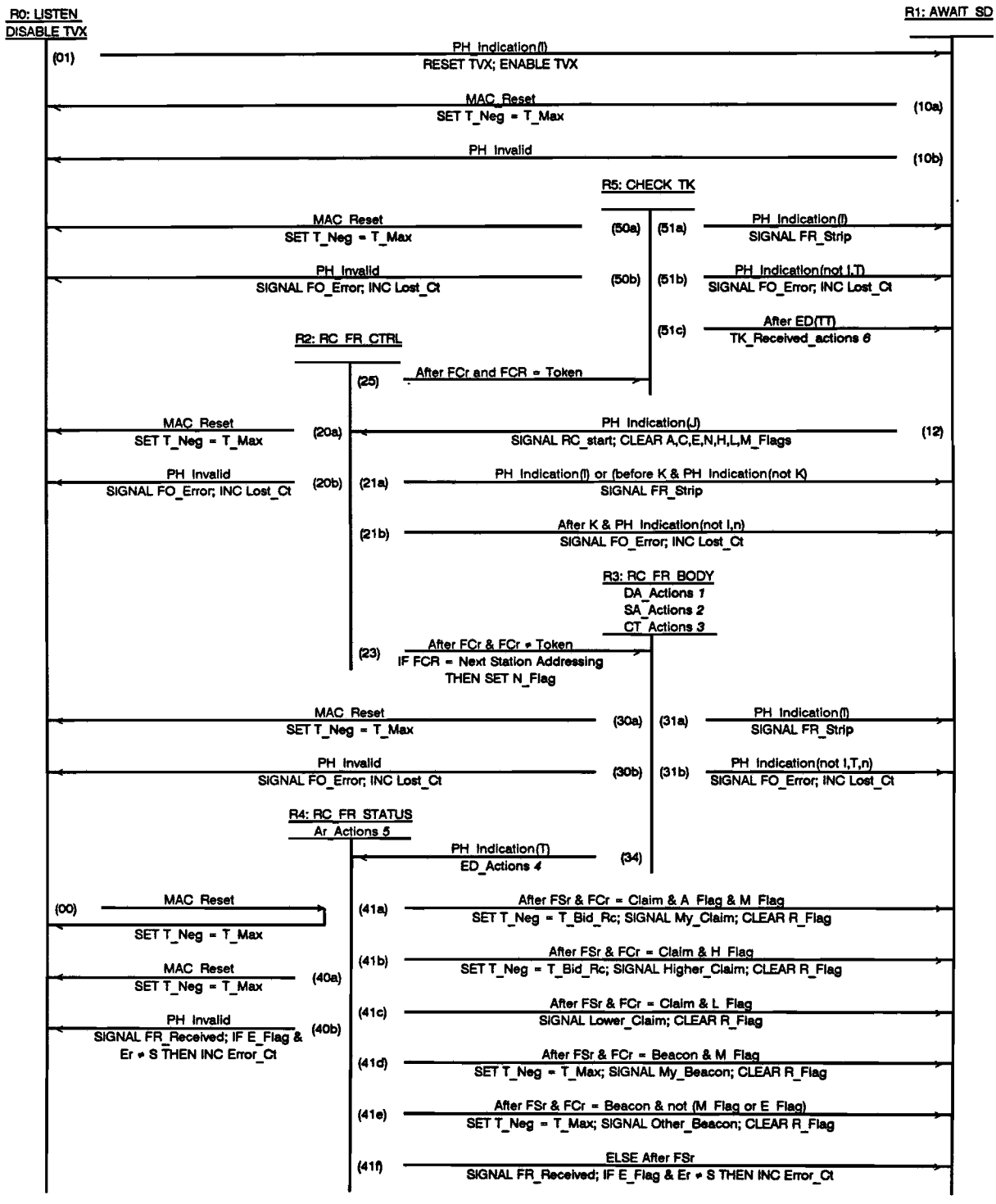
The MAC has been modeled the most extensively, but especially the process monitor will have to be watched at more closely. Further SMT has to be developed from scratch.

The designs that are ready so far, will have to be extended. PHY must be finished by adding the parts that operate on a higher rate clock. Process_userdata has to be extended with indication to LLC about transmission_status, and possibility to compose MAC frames. Also the store must be extended to receive frames also from other sources than LLC.

In [7] the design of process receive_frames (in MAC) is watched at. So for other designs in this area the reader is referred to that report.

Though still a lot of work has to be done, it may be useful to continue with the design of an FDDI protocol handler, or at least keep an eye on the subject, since FDDI is still developing rapidly to more applications, and high speed data transfer is required in the near future.

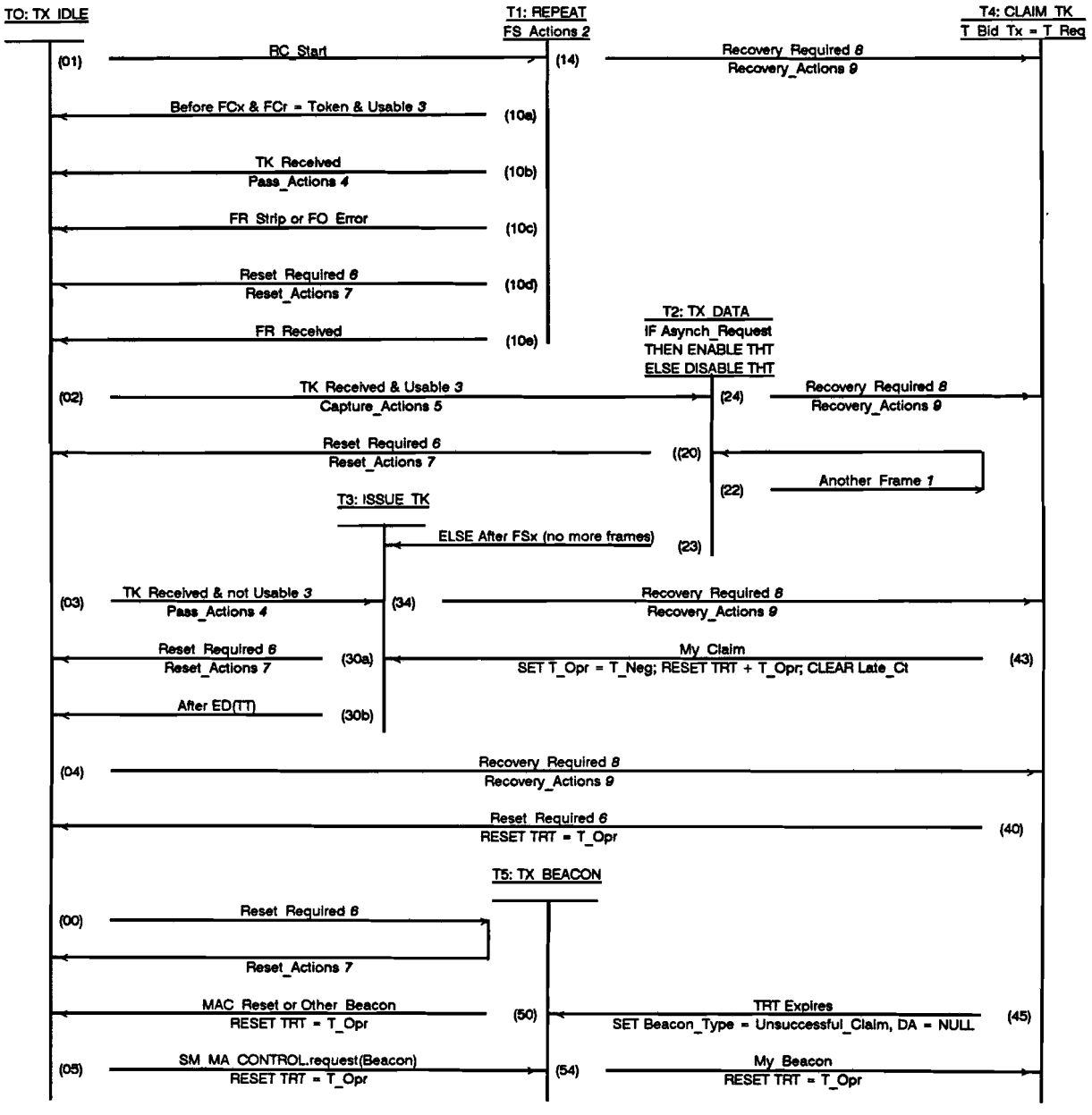
Appendix A: MAC receiver state machine [4]



Vertical staffs indicate states, horizontal shafts indicate transitions. Above the shaft the conditions are given for which the transitions are made, below the shafts the actions are given which have to be performed during the transition.

```
1. DA_Actions:
   After DAr
      IF FCr ≠ Void
         THEN IF Lr = 0 & DAr is contained in the set of Short_Addresses or
                 Lr = 1 & DAr is contained in the set of Long_Addresses
                 THEN SET A_Flag; copy frame

2. SA_Actions:
   After SAr
      IF (Lr = 0 & SAr = MSA & MSA > 0) or
      (Lr = 1 & SAr = MLA & MLA > 0)
      THEN SET M_Flag; SIGNAL FR_Strip
      ELSE IF Lr = 0 & SAr > MSA & MLA = 0 or
              LR = 1 & SAr > MLA
              THEN SET H_Flag
              ELSE IF SAr > 0
                      THEN SET L_Flag

3. CT_Actions:
   After 4_Info_Octets
      IF FCr = Claim
         THEN IF T_Bid_Rc ≠ T_Req
                 THEN CLEAR M_Flag;
                      IF T_Bid_Rc > T_Req
                         THEN IF L_Flag
                                 THEN SET H_Flag; CLEAR L_Flag
                         ELSE IF H_Flag
                                 THEN SET L_Flag; CLEAR H_Flag
              IF L_Flag
                 THEN SIGNAL FR_Strip

4. ED_Actions:
   INC Frame_Ct;
   IF Valid_Data_Length & (Valid_FCS_Rc or (FCr = Void or Implementer))
      THEN RESET TVX;
           IF A_Flag & Valid_Copy
              THEN SET C_Flag
      ELSE SET E_Flag;
           CLEAR A_Flag, H_Flag, M_Flag, L_Flag

5. Ar_Actions:
   After Ar
      IF Ar = R
         THEN CLEAR N_Flag

6. TK_Received_Actions:
   IF Token_Class = Restricted
      THEN IF not R_Flag
              THEN SET R_Flag; Notify SMT
      ELSE RESET TVX; CLEAR R_Flag
   SIGNAL TK_Received
```

Appendix B: MAC transmitter state machine [4]



Vertical staffs indicate states, horizontal shafts indicate transitions. Above the shaft the conditions are given for which the transitions are made, below the shafts the actions are given which have to be performed during the transition.

1. Another_Frame:
   After FSx & Late_Ct = 0 & (Synch_Request or
      (Asynch_Request & Requested_TK_Class = R_Flag &
      (Ignore_THT or THT < T_Pri(Request_Priority))))

2. FS_Actions:
   Before Ex
      IF E_Flag THEN SET Ex = S ELSE SET Ex = Er
   Before Ax
      IF A_Flag THEN SET Ax = S ELSE SET Ax = Ar
   Before Cx
      IF C_Flag & not N_Flag THEN SET Cx = S ELSE SET Cx = Cr

3. Usable_Token:
   Ring_Operational & (Synch_Request or
      (Asynch_Request & Late_Ct = 0 & Requested_TK_Class = R_Flag &
      (Ignore_THT or THT < T_Pri(Request_Priority))))

4. Pass_Actions:
   IF Ring_Operational
      THEN IF Late_Ct = 0
                THEN RESET TRT = T_Opr
                ELSE CLEAR Late_Ct
      ELSE SET T_Opr = T_Neg; RESET TRT = T_Opr;
           SET Late_Ct = 1; SET Ring_Operational

5. Capture_Actions:
   DISABLE THT;
   IF Late_Ct = 0
      THEN SET THT = TRT; RESET TRT = T_Opr
      ELSE SET THT = expired; CLEAR Late_Ct

6. Reset_Required:
   MAC_Reset or Higher_Claim or Other_Beacon

7. Reset_Actions:
   SET T_Opr = T_Max;
   IF Ring_Operational or Late_Ct = 0
      THEN RESET TRT = T_Opr; SET Late_Ct = 1;
           CLEAR Ring_Operational

8. Recovery_Required:
   TVX Expires or (TRT Expires & Late_Ct > 0) or
   (Ring_Operational & T_Opr < T_Req) or
   Lower_Claim or My_Beacon

9. Recovery_Actions:
   T_Opr = T_Max; RESET TRT = T_Opr; CLEAR Ring_Operational

10. TRT_Actions:
    In all states: IF TRT expires
                  THEN INC Late_Ct; RESET TRT = T_Opr

Appendix C: Abbreviations for MAC [4]

Error_Ct	Count of reportable frame errors
Frame_Ct	Count of all frames received
Late_Ct	Count of TRT expirations
Lost_Ct	Count of PDUs detected as lost
A_Flag	Indicates destination address match in last received frame
C_Flag	Indicates successful copying of last received frame
E_Flag	Indicates error detected in last received frame
H_Flag	Indicates Higher Source Address received
L_Flag	Indicates Lower Source Address received
M_Flag	Indicates My Source Address received
N_Flag	Indicates next station addressing
R_Flag	Indicates the token_class of the last valid token received was restricted
A_Max	Maximum signal acquisition time
D_Max	Maximum ring latency time
F_Max	Maximum frame time
I_Max	Maximum station physical insertion time
L_Max	Maximum transmitter frame set-up time
M_Max	Maximum number of MAC entities
S_Min	Minimum safety timing allowance
T_Bid_Rc	Bidding TTRT received by this station in claim frames
T_Bid_Tx	Bidding TTRT transmitted in this station's claim frames
T_Init	Ring initialization time
T_Max	Maximum TTRT to be supported by this station
T_Min	Minimum TTRT to be supported by this station
T_Neg	Negotiated TTRT during claim process (in receiver)
T_Opr	Operative TTRT for this station (in transmitter)
T_Pri	Set of n priority token rotation time thresholds
T_Pri(n)	Element n of set T_Pri
T_React	Worst case time to react to a station insertion or removal
T_Req	Requested TTRT for this station's synchronous traffic
T_Resp	Worst case time to recover a token
THT	Token holding Timer
TRT	Token Rotation Timer
TTRT	Target Token Rotation Time
TVX	Valid-Transmission Timer

Appendix D: Files made with PROMOD

Disk labelled PROMOD 1

\FDDI.PRD : Main file used by PROMOD, containing all descriptions of dataflow diagrams, controlflow diagrams, minispecs, controlspecs and data definitions. This file can be used immediately by PROMOD, without any other files.

Disk labelled PROMOD 2

\FDDI.SA : Analysis report, made by PROMOD, containing descriptions of minispecs, controlspecs and data definitions. This file can be printed via a normal ASCII text editor or via the type command in DOS. **Note: This file contains no flow diagrams.**

\PLOTS*.PCF : Plotfiles in HPGL format, containing controlflow diagrams.

\PLOTS*.PDF : Plotfiles in HPGL format, containing dataflow diagrams.

\LP.EXE : Plotter program, that can be used for plotting the HPGL files produced with PROMOD.

\PETER.LP : Settings for LP.EXE, that make sure that the plot files can be plotted in a correct way.

\PLOT.BAT : Batch file starting up LP.EXE with the settings in PETER.LP

The plotfiles can be produced by PROMOD, by choosing the Data Flow Diagram (DFD) or Control Flow Diagram (CFD) when in the SA-DEFINE menu, followed by the name of the diagram. Choose the ordered options OUTPUT, PLOT, A5 for plotting the diagram to a HPGL-file.

When ready with plotting, it is possible to plot another diagram, by pressing twice the F1 function button and entering the name of a diagram (followed by OUTPUT, PLOT, A5).

When in stead of the dataflow diagram, the control flow diagram (or vice versa) must be plotted, press as much F1 until the SA-DEFINE menu is reached and choose the other of DFD or CFD options.

Appendix E: Data types and literals used with PROMOD

Literals used in the report

The literals are values that certain signals can take. The following are defined; the meaning of them is clear in the context they are used in.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, H, I, J, K, Q, R, S, T, V
acknowledge
Asynchronous
beacon
claim
clock
do_bypass
do_not_bypass
error_in_FCS
error_in_length
False
fr_good
Group
Implementer
Individual
integer
internal_error
LLC
light
Local
long
MAC
me
norequest
not_acknowledge
not_specified
other
other_type
repeat
request
Restricted
send
short
SMT
Synchronous
token
True
Universal
Unrestricted
void

Data types used in the descriptions

With the data descriptions the following data types are used:

```
bit             = ["0"|"1"]
long_address    = 48 { bit } 48
short_address   = 16 { bit } 16
bits_8          = 8 { bit } 8
symbols_4_bit   = ["0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"|"A"|"B"|"C"|"D"|"E"|"F"]
symbol          = [symbols_4_bits|"I"|"J"|"K"|"R"|"S"|"T"|"Q"|"H"|"V"]
S_or_R          = ["S"|"R"]
frame_status    = 3 { S_or_R } 3
priority_level  = "integer"
time            = "integer"
```

Appendix F: PROMOD SA analysis report

```
**********************************
*         P r o M o d            *
*                                *
*        SA  -  REPORT           *
*                                *
*        PROJECT: fddi           *
**********************************
```

SYSTEM_HIERARCHY

```
0 context_of_FDDI
  1 FDDI
    1.1 PMD
      1.1.1 electric_to_optic
      1.1.2 optic_to_electric
      1.1.3 optical_bypass
    1.2 PHY
      1.2.1 decode
      1.2.2 serial_to_parallel
      1.2.3 synchronize
      1.2.4 recover_clock
      1.2.5 encode
      1.2.6 parallel_to_serial
    1.3 MAC
      1.3.1 process_userdata
        1.3.1.1 store
        1.3.1.2 indicate
        1.3.1.3 add_controls
        1.3.1.4 compose
      1.3.2 transmit_frames
      1.3.3 receive_frames
        1.3.3.1 forward_frames
          1.3.3.1.1 switch
          1.3.3.1.2 buffer
        1.3.3.2 process_FS
          1.3.3.2.1 get_EAC
          1.3.3.2.2 make_EAC
        1.3.3.3 check_on_function
          1.3.3.3.1 check_on_FC
          1.3.3.3.2 check_on_DA
          1.3.3.3.3 check_on_SA
        1.3.3.4 check_on_violations
          1.3.3.4.1 convert_to_control
          1.3.3.4.2 check_on_FCS
          1.3.3.4.3 check_on_length
      1.3.4 process_frames
      1.3.5 monitor
        1.3.5.1 token_management
        1.3.5.2 timing
        1.3.5.3 lost_ct
        1.3.5.4 frame_ct
        1.3.5.5 error_ct
        1.3.5.6 frame_handler
    1.4 SMT
```

PROBLEM_DESCRIPTION context_of_FDDI

- FDDI must transport data with 100 Mbps.

- FDDI must operate on a fiber optic ring, with a timed token protocol. This means that the token must pass a certain station within a predefined amount of time.

- To prevent cable breaks to interrupt the proper working of the system, the ring is made dual and counter rotating. This opens the possibility of reconfiguration on a cable break.

- FDDI must accept data on >userdata_out (during >stream) and divide it into frames

- When there is a frame to be sent, a token must be captured, that satifies >token_class

- To be able to discover the beginning and ending of a frame, and for some other (error) controls, the data presented to fiber_out is encoded by a 4B/5B encoding scheme.

- When a token is captured, frames may be transmitted to <fiber_out as long as token timing allows

- When a request is serviced number_of_SDUs and provided_service_class is generated, and on return of the frames MA_transmission_status is generated

- When >token_request is received, a token must be captured while there is not yet any data to be sent

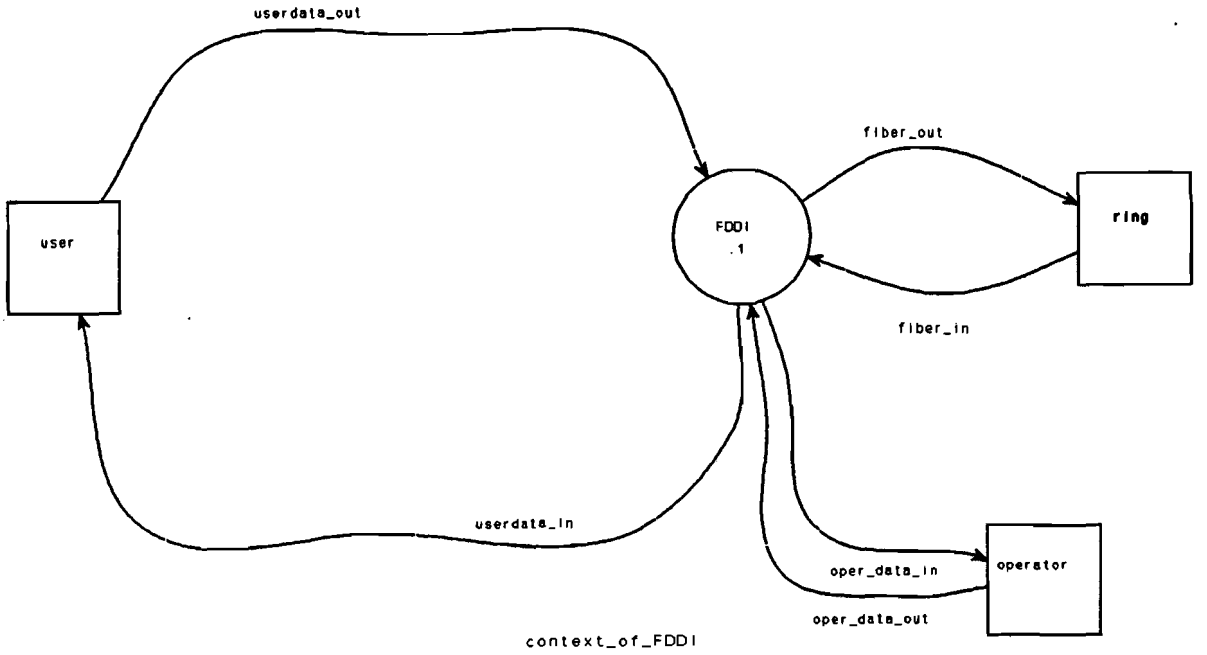
- FDDI accepts frames coming in on >fiber_in

- When a frame is received that comes from another station, and is directed to this station FDDI checks it on errors and presents it to <userdata_in allong with <reception_status

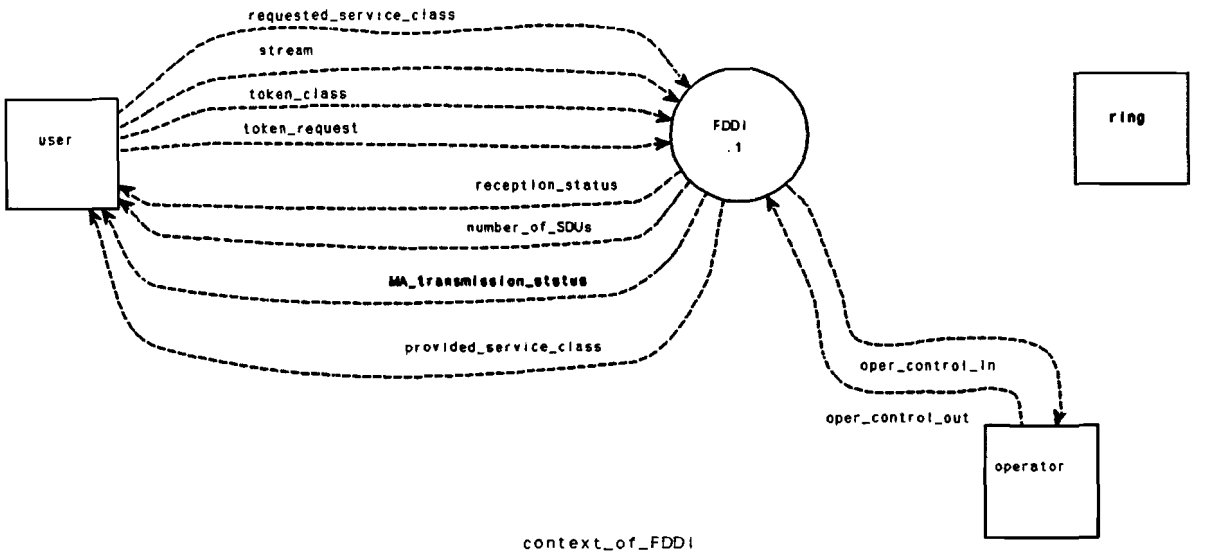
- Because the FDDI must serve several types of users, the ring operation must be adaptive in timing.

- FDDI must accept commands presented on >oper_control_out and >oper_data_out, and react accordingly on <oper_control_in and <oper_data_in

FD 0 context_of_FDDI



context_of_FDDI

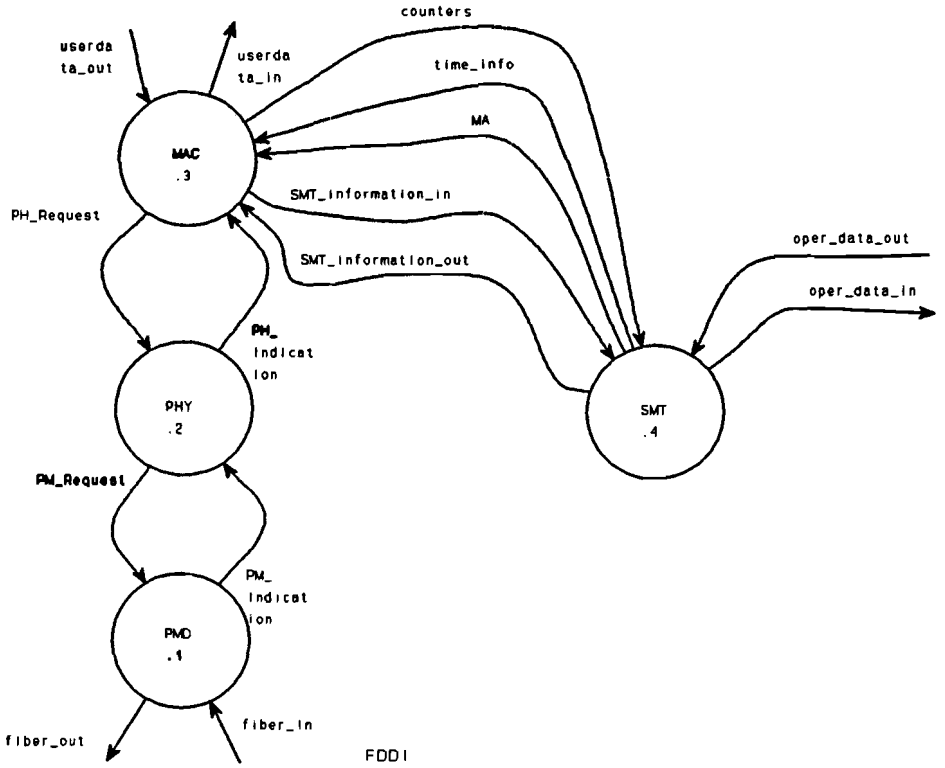


context_of_FDDI

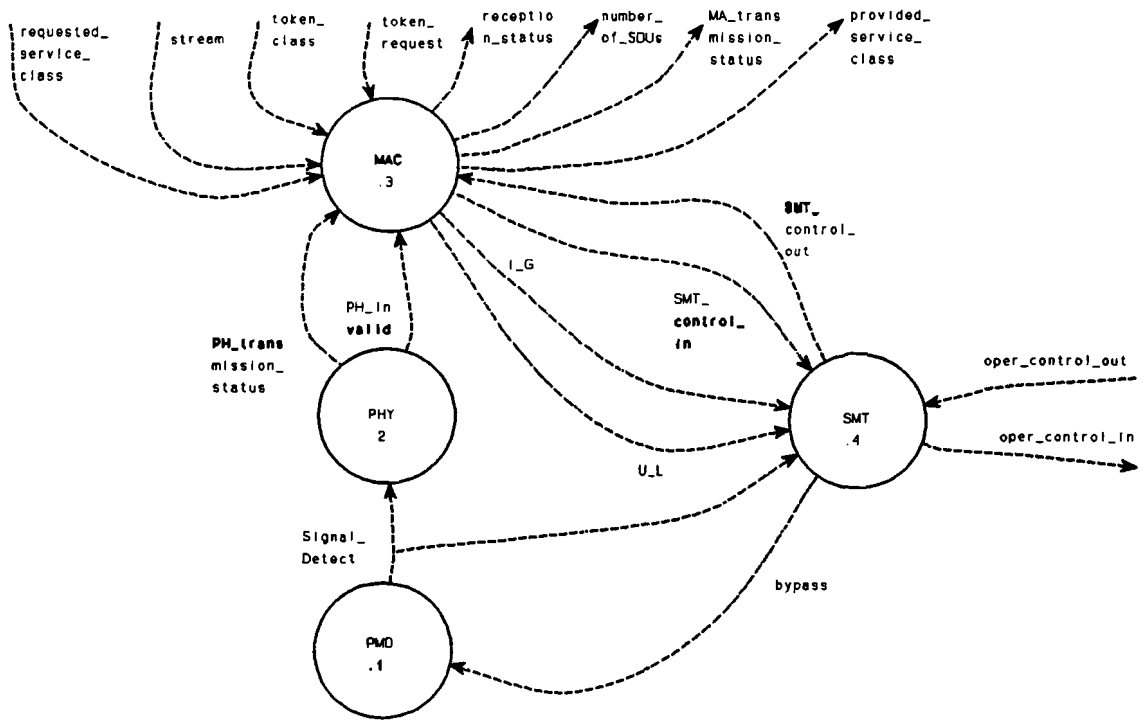
MSP 1 FDDI

- take >userdata_out, divide it in pieces and send it to a fiber ring (<fiber_out).
- the protocol used wil be a timed token protocol.
- take >fiber_in, make user data and send it to <userdata_in.
- the system will be synchronous to a system clock

FD 1 FDDI



FDDI



FDDI

MSP 1.1 PMD

- transfer from light to bits (>fiber_in to <PM_Indication) and vice versa (>PM_Request to <fiber_out)
- detect whether incoming light on >fiber_in is under/above threshold and indicate by <Signal_Detect
- on >bypass: bypass from >fiber_in to <fiber_out and from >PM_Request to <PM_Indication

MSP 1.2 PHY

- recover clock from >PM_Indication
- synchronize the incoming bits (>PM_Indication) with the system clock
- decode >PM_Indication to <PH_Indication
- encode >PH_Request to <PM_Request
- proceed only on >Signal_Detect

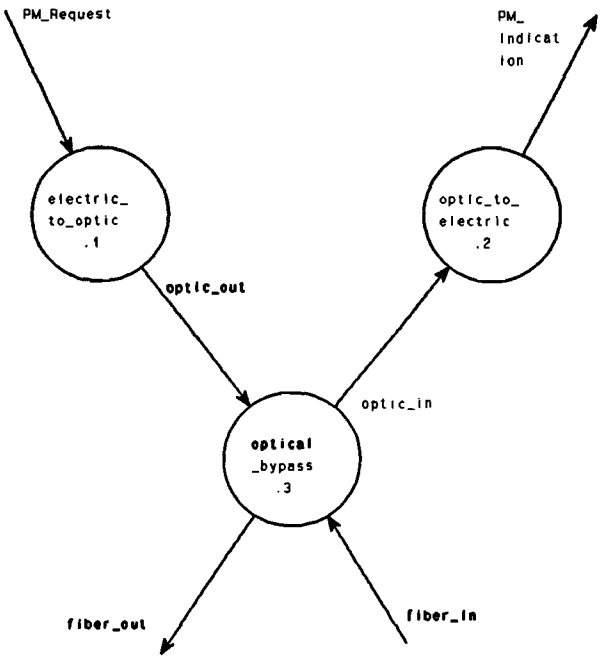
MSP 1.3 MAC

- on <userdata_out: - store the incoming data and process to frames until ? >stream
  - wait for an appropriate token based on >requested_service_class
  - on token capture: send frames as long as is allowed to <PH_Request
  - on end of transmission, generate <number_of_SDUs indicating the number of frames of each request (>stream) that is transmitted, and generate <provided_service_class indicating the type of token that was captured
  - when frames are returned on PH_Indication: indicate by <MA_transmission_status based on internal checks and >PH_transmission_status
- on >token_request: capture a token of the requested type and hold as long as allowed
- on >PH_Indication: - check for the destination and source
  - depending on destination and source transfer the frames to >PH_Request and/or >userdata_in or strip the frame;
    if transfered to >userdata_in : generate <reception_status, indicating the frame was received intact

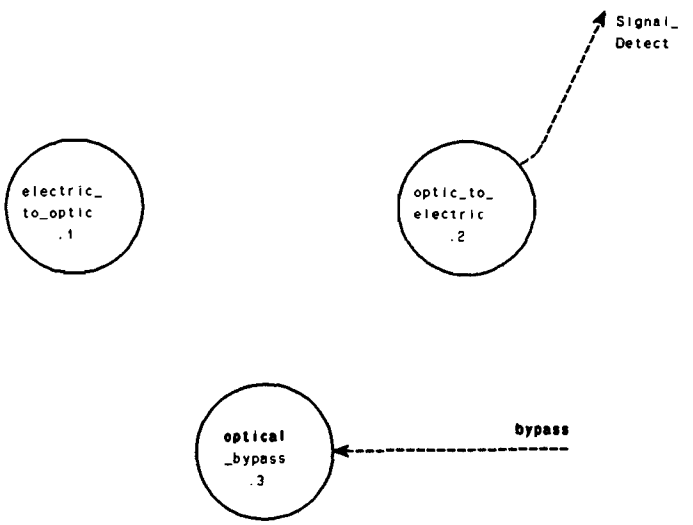
MSP 1.4 SMT

- manages the entire station

FD 1.1 PMD



PMD



PMD

MSP 1.1.1 electric_to_optic

- convert the electrical signal on >PM_Request to an optical signal on <optic_out

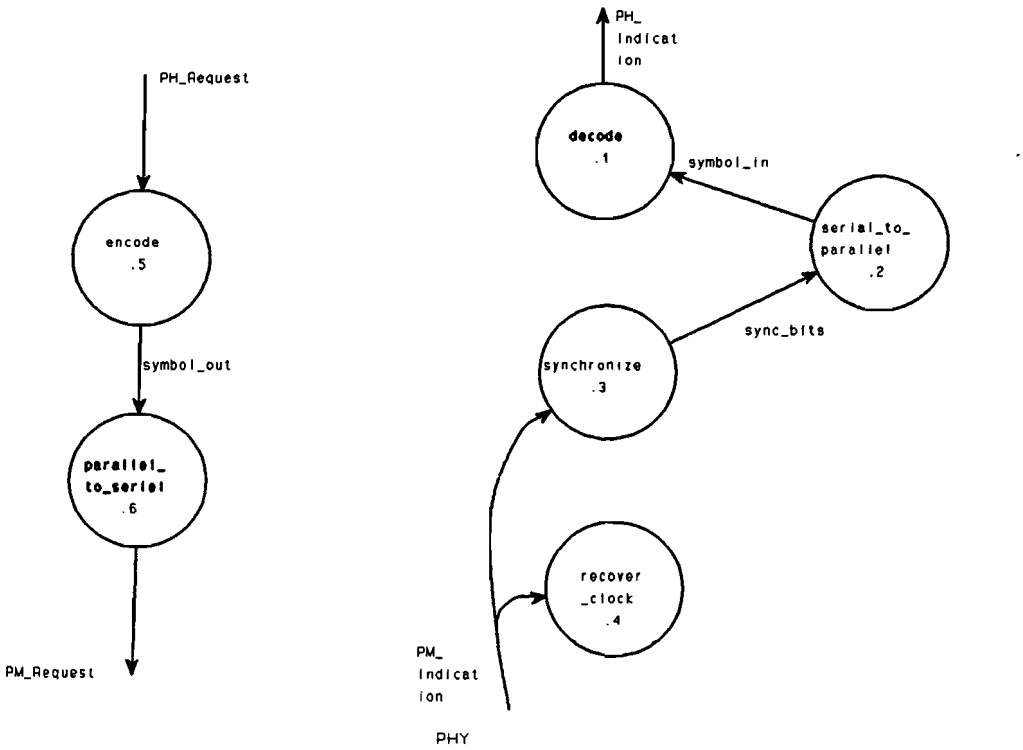
MSP 1.1.2 optic_to_electric

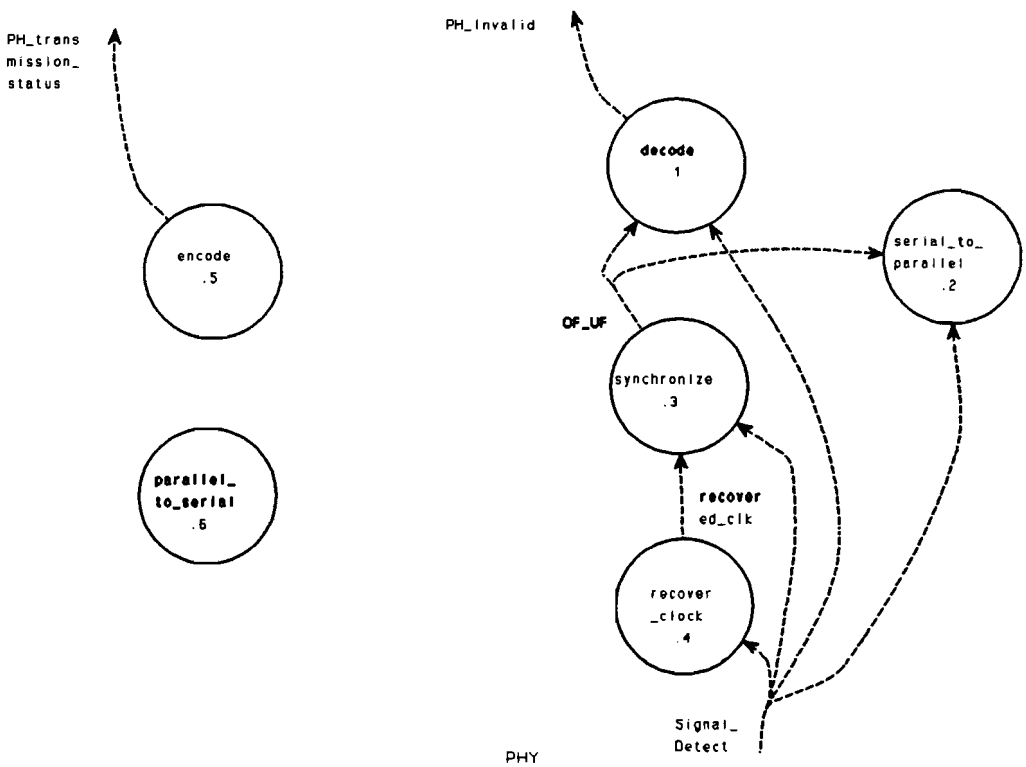
- convert the optical signal on >optic_in to an electrical signal on <PM_Indication
- when the optical power on >optic_in is above the detection threshold: generate <Signal_Detect

MSP 1.1.3 optical_bypass

- initially: link >fiber_in to <optic_in, link >optic_out to <fiber_out
- on >bypass: link >fiber_in to <fiber_out, link >optic_out to <optic_in

FD 1.2 PHY


PH_Request
encode
.5
symbol_out
parallel_
to_serial
.6
PM_Request
PH_
Indicat
ion
decode
.1
symbol_in
serial_to_
parallel
.2
sync_bits
synchronize
.3
recover
_clock
.4
PM_
Indicat
ion
PHY


PH_trans
mission_
status
encode
.5
parallel_
to_serial
.6
PH_Invalid
decode
1
serial_to_
parallel
.2
OF_UF
synchronize
.3
recover
ed_clk
recover
_clock
.4
Signal_
Detect
PHY

MSP 1.2.1 decode

- decodes >symbol_in synchronous to the system clock and presents the symbols to <PH_Indication
- on >OF_UF or absence of >Signal_Detect no function is executed

MSP 1.2.2 serial_to_parallel

- as long as no symbol boundaries are found: read >sync_bits and generate <symbol_in every bit time
- when symbol booundaries are found: read >sync_bits and generate <symbol_in every five bit times
- obtain the symbol boundaries by searching for a "J""K" sequence every bit time, each time that "J""K" is found, it introduces a new symbol boundary
- on >OF_UF or absence of >Signal_Detect: perform no functions

MSP 1.2.3 synchronize

- the incoming bit stream from >PM_Indication synchronized to >recovered_clk is to be adjusted so that it is synchronized to system clock. It is sent to <sync_bits
- in case of an error: generate <OF_UF

MSP 1.2.4 recover_clock

- recover the clock with which the incoming bits on >PM_Indication were sent, and send it to <recovered_clk

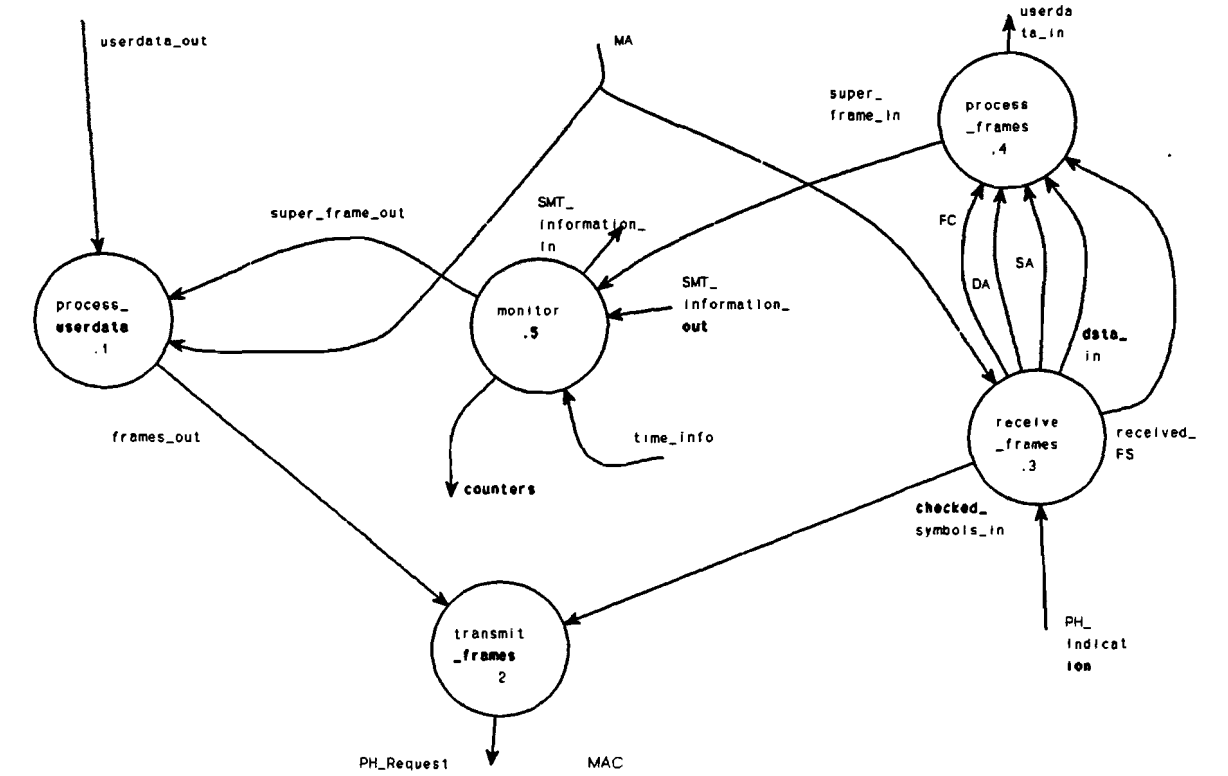
MSP 1.2.5 encode

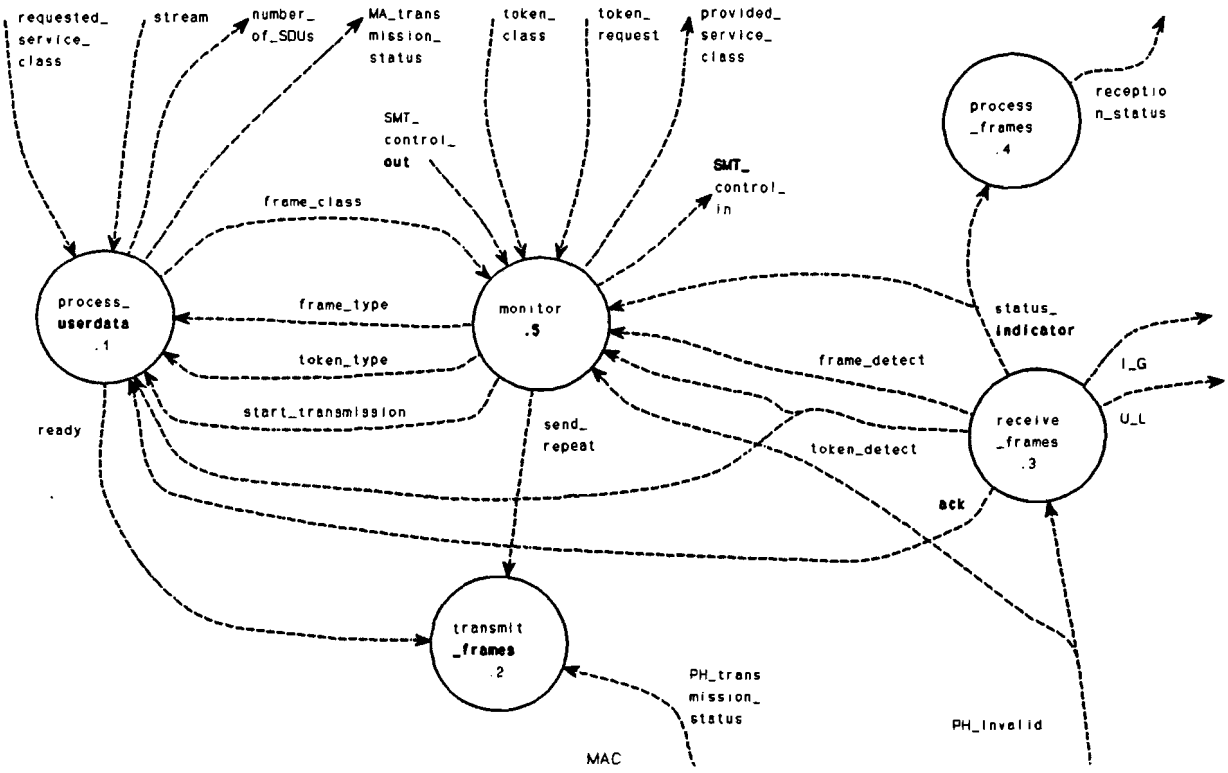
- encodes the incoming symbols from >PH_Request synchronous to system clock and presents the coded symbols to <symbol_out

MSP 1.2.6 parallel_to_serial

- Read >symbol_out and generate the seperate bits to <PM_Request

FD 1.3 MAC


userdata_out
MA
userda
ta_in
super_
frame_in
process
_frames
.4
super_frame_out
SMT_
information_
in
FC
SA
DA
process_
userdata
.1
monitor
.5
SMT_
information_
out
data_
in
receive
_frames
.3
received_
FS
frames_out
time_info
counters
checked_
symbols_in
transmit
_frames
.2
PH_
indicat
ion
PH_Request
MAC


requested_
service_
class
stream
number_
of_SDUs
MA_trans
mission_
status
token_
class
token_
request
provided_
service_
class
process
_frames
.4
receptio
n_status
SMT_
control_
out
SMT_
control_
in
frame_class
process_
userdata
.1
frame_type
monitor
.5
status_
indicator
token_type
frame_detect
I_G
start_transmission
ready
send_
repeat
token_detect
U_L
receive
_frames
.3
ack
transmit
_frames
.2
PH_trans
mission_
status
PH_invalid
MAC

MSP 1.3.1 process_userdata

- read >userdata_out, insert >MA and store
- read >requested_service_class and store, count number of SDUs received
- if >frame_type =? "other_type": make the requested type of frame
- on >start_transmission: copy the correct frame to <frames_out adding PA, SD, FCS, ED and EAC depending on >frame_type
- on end of >start_transmission: finish sending, generate <ready
- when no more frames left to be sent: generate <ready
- when stopped with sending while >frame_type = "other_type": generate <number_of_SDUs and <MA_transmission_status based on >ack

MSP 1.3.2 transmit_frames

- >send_repeat =
  "send": copy >frames_out to <PH_Request
  "repeat": copy >checked_symbols_in to <PH_Request

MSP 1.3.3 receive_frames

- initially: >PH_Indication is connected to <checked_symbols_in

- check on volations from >PH_Indication, generating <SD_Detect, <ED_Detect, <status_indicator
- check on FC, check for token from >PH_Indication
  in case of token: generate <token_detect
  else : generate <frame_detect, <frame_type
- check on (from >PH_Indication)
  DA = MA --> connect >PH_Indication to <data_in
  (<checked_symbols_in is already connected)
  SA = MA --> disconnect >PH_Indication from outputs
  else --> (<checked_symbols_in is already connected to >PH_Indication)
- check on frame status from >PH_Indication

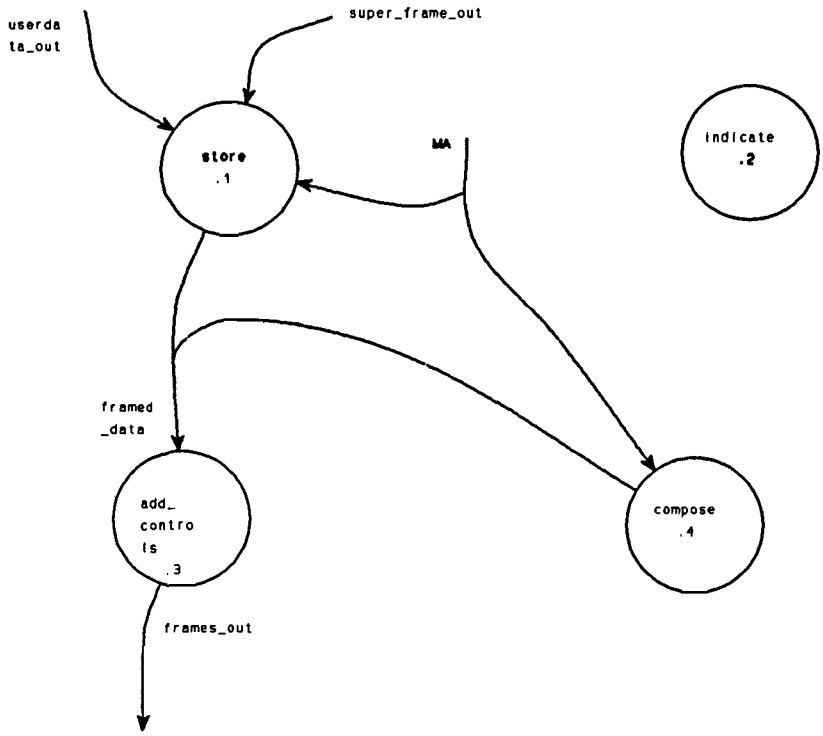
MSP 1.3.4 process_frames

- combine data with FC, DA and SA and send it to <userdata_in
- generate <reception_status from >status_indicator and internal events

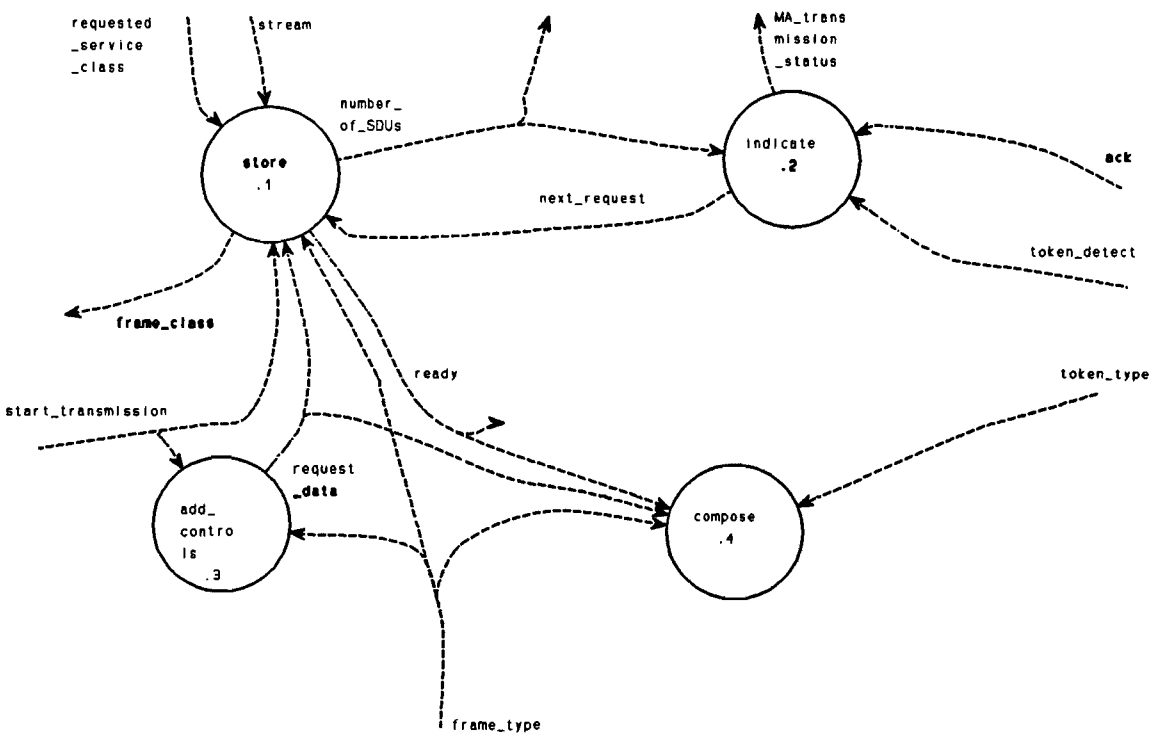
MSP 1.3.5 monitor

- manage the token
- manage counters
- control timing
- handle MAC and SMT frames

FD 1.3.1 process_userdata



process_userdata



process_userdata

MSP 1.3.1.1 store

```
- on >userdata_out:
   #SDUr := 0
   if >stream :
      repeat
         take >userdata_out, insert >MA, store in memory
         take >requested_service_class and store
         (when not enough space: do not store)
         #SDUr := #SDUr + 1
      until ? >stream
      store #SDUr
- if frames in memory: generate <frame_class based on
   stored requested_service_class, but only when the
   corresponding request has fully been loaded
- on >request_data and (>frame_type = "other_type") :
   #SDUs := 0
   get #SDUr
   repeat
      send frame to >framed_data
      #SDUr := #SDUr - 1
      #SDUs := #SDUs + 1
   until #SDUr = 0 or ? >start_transmission
   store #SDUs
   if #SDUr = 0: generate <ready
   if ? >start_transmission : delete rest of frames of same
      request
- on >next_request : if some request is sent:
   generate <number_of_SDUs = #SDUs
```

MSP 1.3.1.2 indicate

```
- initially: next_request outstanding

- on >number_of_SDUs : store >number_of_SDUs in #SDUa
- on >ack : #SDUa := #SDUa - 1
            if (>AR = S) and (>CR = S):
       generate <MA_transmission_status =
                   "acknowledge"
            else :
               generate <MA_transmission_status =
                   "not_acknowledge"
- on >token_detect :
   if #SDUa =? 0: generate
                   MA_transmission_status = end_acknowledge;
                   #SDUa := 0
- on #SDUa = 0 generate <next_request
```

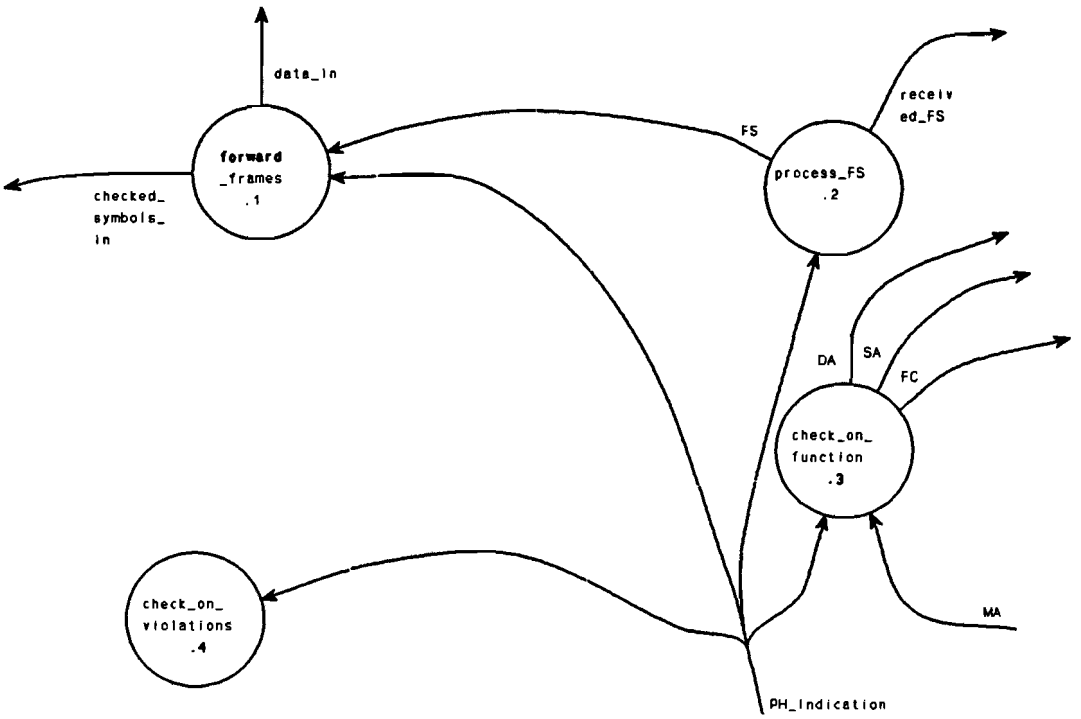
MSP 1.3.1.3 add_controls

```
- on >start_transmission :
   - put PA * 16 I symbols *
   - put SD; generate <request_data
   - put >framed_data
   - if >frame_type =? token:
      - add FCS
      - add "T" symbol
      - add three "R" symbols
     else:
      - add two "T" symbols
   - send to <frames_out
```

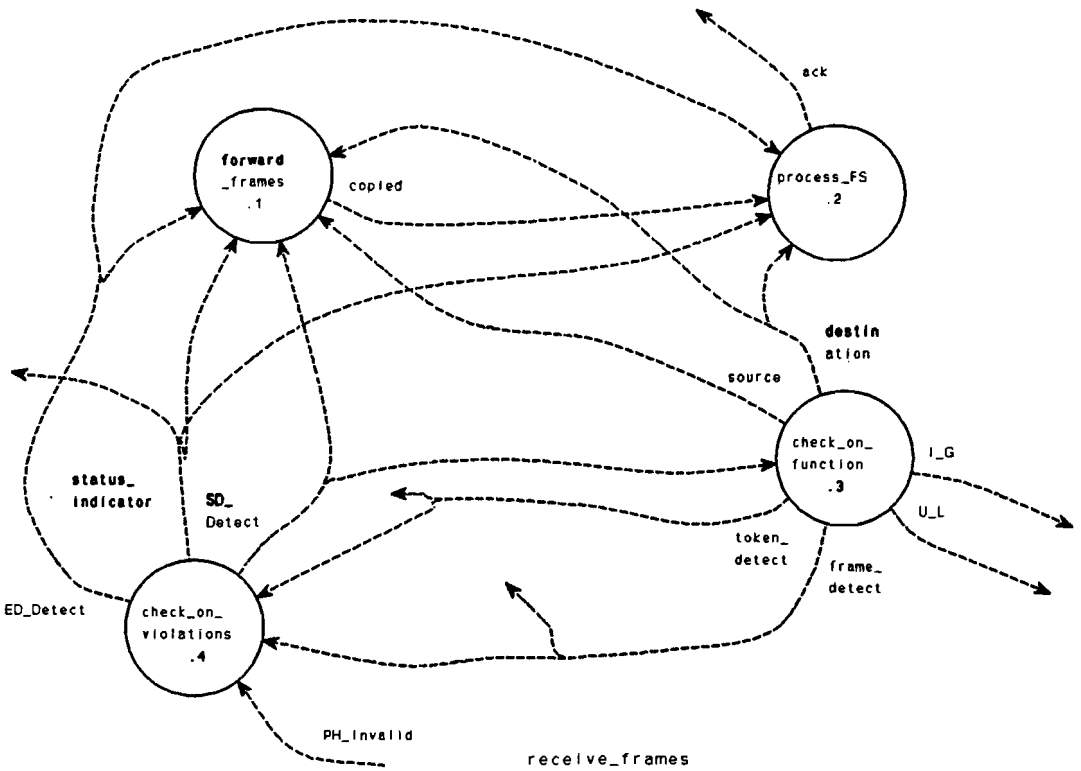
MSP 1.3.1.4 compose

```
- >frame_type =
   "beacon" --> make [beacon FC] + [DA = 0] + [>MA]
   "claim" --> make [claim FC] + [DA = >MA] + [>MA] + [time]
   "token" --> make [token FC = >token_type]
- on >ready and >request_data: send frame to
   <framed_data
```

FD 1.3.3 receive_frames



receive_frames



receive_frames

MSP 1.3.3.1 forward_frames

- on >SD_Detect : start copying >PH_Indication to
  <checked_symbols_in
- on >source = "me" : stop copying
- on >status_indicator : start sending "I" symbols to
  <checked_symbols_in with delay of one symbol
- on >ED_Detect while copying, copy >FS to
  <checked_symbols_in
- after >FS is received: start copying >PH_Indication to
  <checked_symbols_in
- on >status_indicator : send "T" symbols to
  <checked_symbols_in

- on >destination = "me": copy >PH_Indication to <data_in
  (from FC to but not including FCS)
- on >ED_Detect, while copying: generate <copied

MSP 1.3.3.2 process_FS

- on >ED_Detect: take frame status from >PH_Indication and
  remember
- on >length_error or >FCS_error or >invalid_symbols:
  generate <E = "S"
- on >destination = "me": generate <A = "S"
- on >copied: generate <C = "S"
- when frame status is completely received: generate <FS

MSP 1.3.3.3 check_on_function

all parts are taken from >PH_Indication

- on SD_Detect :
  check for FC, if token: generate <token_detect
                else   :   generate <frame_detect

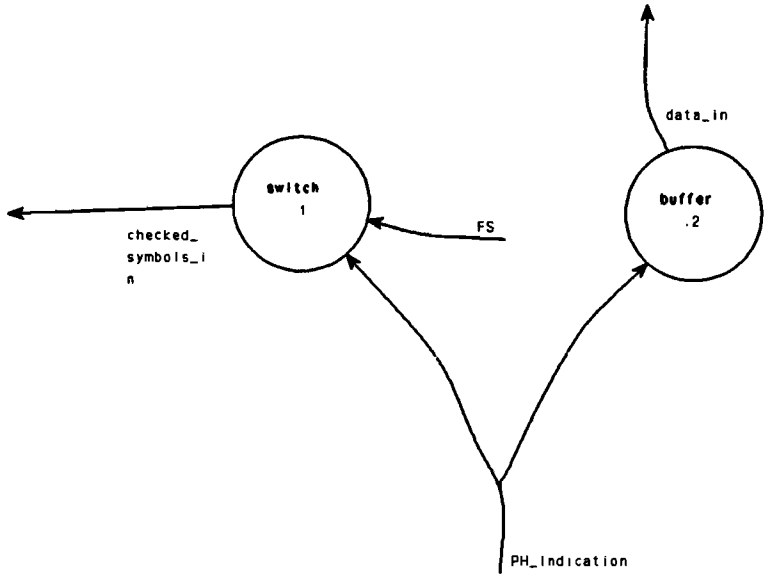
- when no token is detected
  take DA,   if DA = >MA: generate <destination = "me",
                             generate <DA
                else: generate <destination = "other"

- when DA is taken
  take SA,   if SA = >MA: generate <source = "me"
                else: generate <source = "other"
  if >destination = "me" : generate <SA

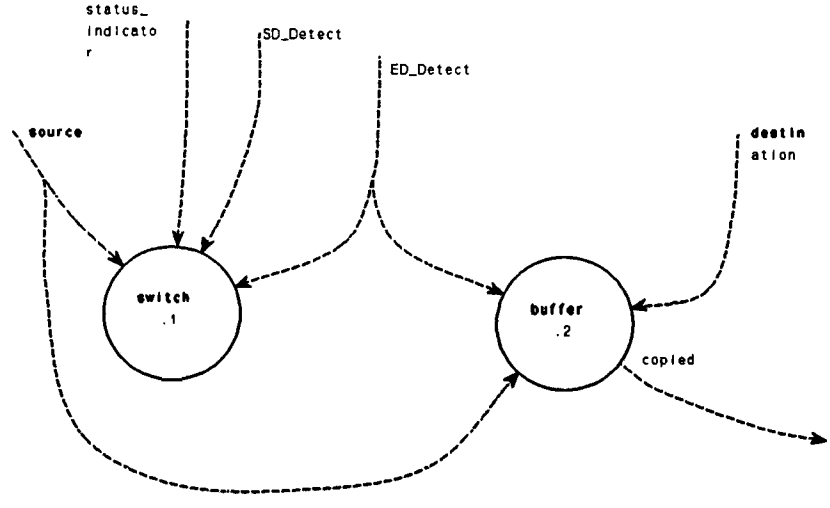
MSP 1.3.3.4 check_on_violations

- search for a start delimiter on >PH_Indication, and generate <SD_Detect
- search for an end delimiter on >PH_Indication, and generate <ED_Detect
- during a frame (>frame_detect): check on the frame check sequence until >ED_Detect and generate <FCS_error as needed
- on >ED_Detect: check on the length of the frame (>frame_detect) or token (>token_detect) and generate <length_error as needed
- on detection of invalid symbols in a frame or token on >PH_Invalid, generate <invalid_symbols

FD 1.3.3.1 forward_frames



forward_frames



forward_frames

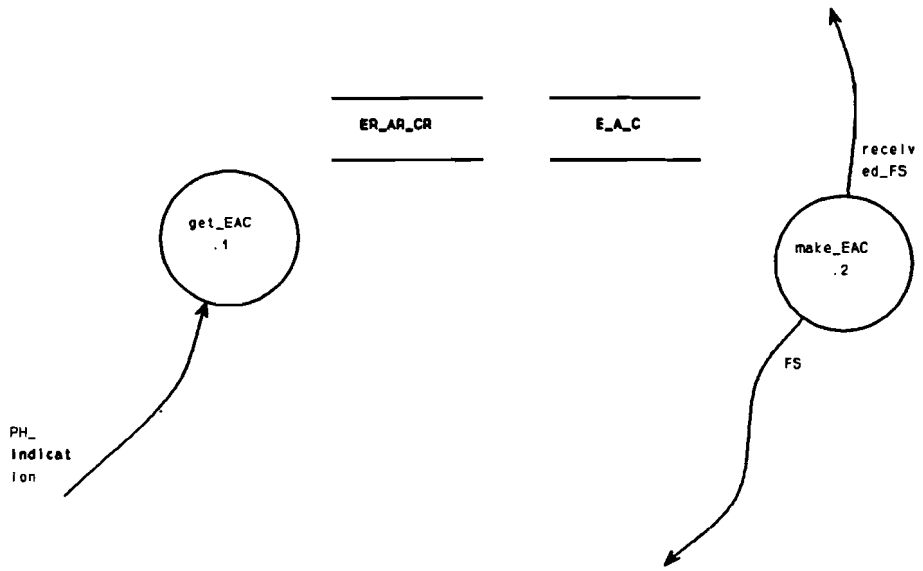
MSP 1.3.3.1.1 switch

```
- initially: copy >PH_Indication to <checked_symbols_in

- on >source = me: start sending "I" symbols to
   <checked_symbols_in
- on >SD_Detect    : start copying >PH_Indication to
   <checked_symbols_in
- on >ED_Detect    : if copying: copy >FS to
                         <checked_symbols_in;
                         start copying >PH_Indication to
                         <checked_symbols_in
- on >status_indicator : start sending "I" symbols to
                              <checked_symbols_in with delay
                              of one symbol
```

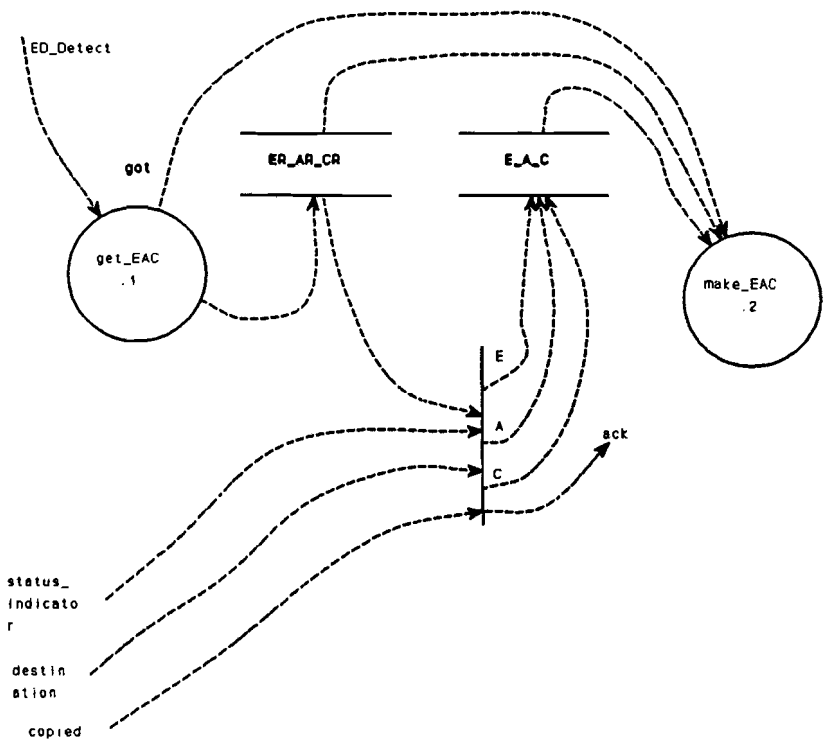
MSP 1.3.3.1.2 buffer

```
- initially: no actions to <data_in

- on >destination = "me": after >source: start copying
   >PH_Indication to <data_in
- on >ED_Detect        : if copying: generate <copied,
                                     stop copying
```

FD 1.3.3.2 process_FS



process_FS



process_FS

MSP 1.3.3.2.1 get_EAC

- on >ED_Detect: take next three symbols from >PH_Indication
  and generate <ER, <AR, <CR corresponding to these symbols
- generate <got

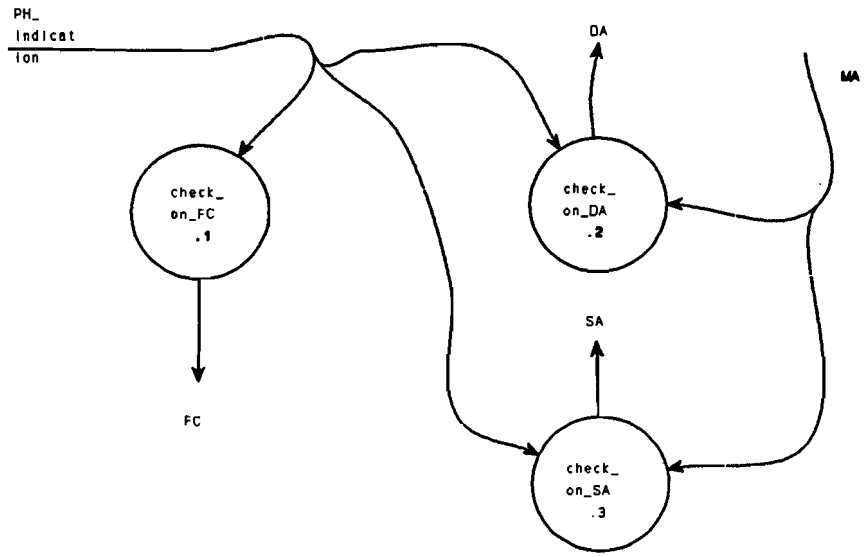
MSP 1.3.3.2.2 make_EAC

- on >got: - take >E, >A, >C and send the corresponding
  symbols to <FS

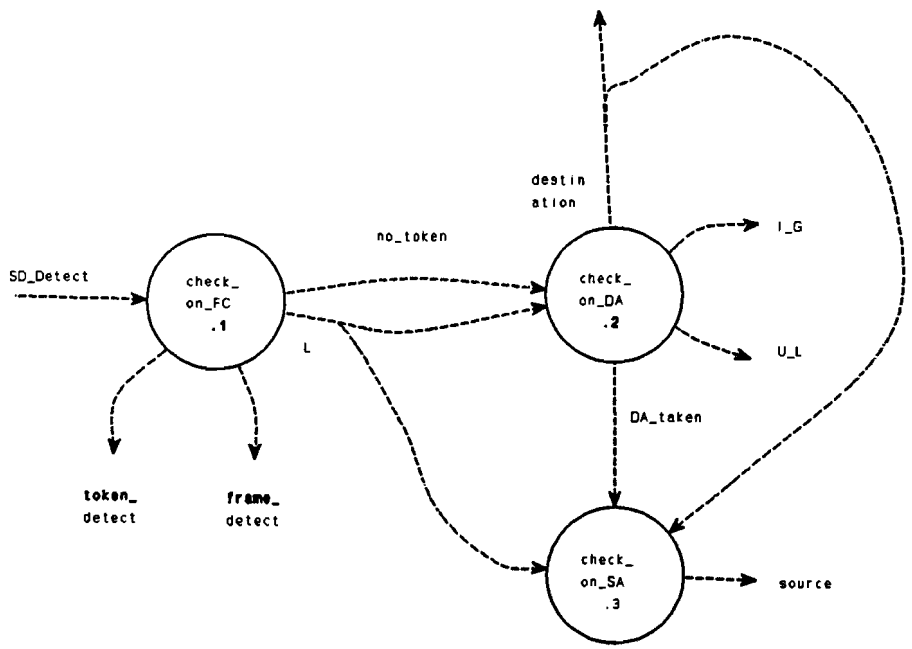
CS 1.3.3.2 process_FS (Sheet 1)

```
- on >ER_AR_CR:
   - generate <E = (if >ER = "S" : "S"
                    if >status_indicator = "FCS_error" or
                       "length_error" or
                       "invalid_symbols" : "S"
                    else "R")
               <A = (if >AR = "S" : "S"
                    if >destination = "me" : "S"
                     else "R")
               <C = (if >CR = "S" : "S"
                    if >copied : "S"
                     else "R")
   - if and >AR and >CR : generate <ack = "acknowledge"
     else               : generate <ack = "not_acknowledge"
```

FD 1.3.3.3 check_on_function



check_on_function



check_on_function

MSP 1.3.3.3.1 check_on_FC

```
on >SD_Detect:
 take FC from >PH_Indication
 generate <FC
 if FC=1000 0000 b: generate <token_detect="Unrestricted"
 else
  if FC=1100 0000 b: generate <token_detect="Restricted"
  else
  generate <no_token
  if FC=0x00 0000 b: generate <frame_detect="void"
  else
   if FC=0x00 xxxx b: generate <frame_detect="SMT"
   else
    if FC=1x00 0010 b: generate <frame_detect="beacon"
    else
     if FC=1x00 0011 b: generate <frame_detect="claim"
     else
      if FC=xx01 0xxx b: generate <frame_detect="LLC"
      else
       if FC=xx10 0xxx b: generate
                                <frame_detect="Implementer"
       else
       skip.
```

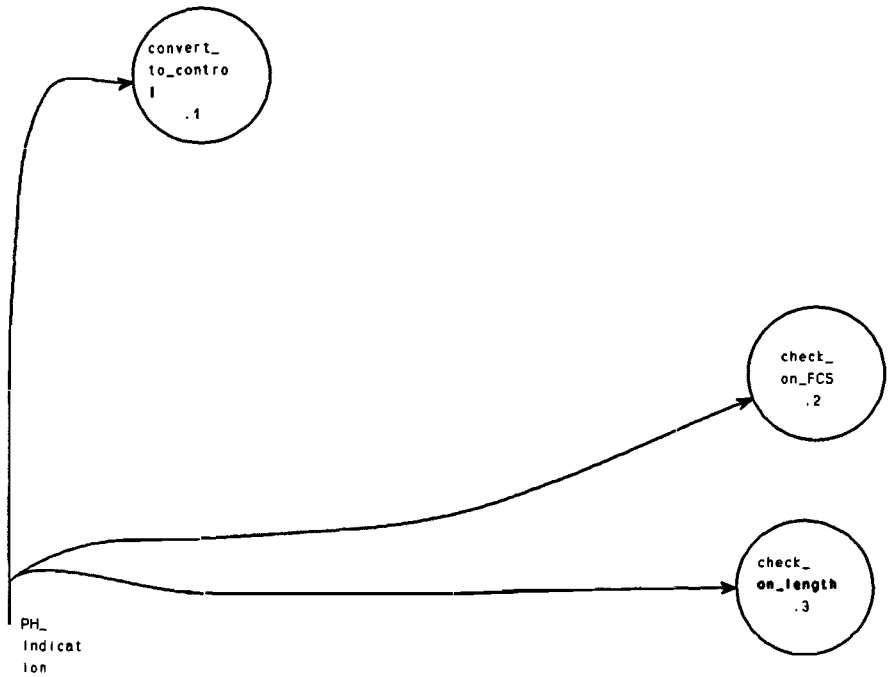
MSP 1.3.3.3.2 check_on_DA

```
on >no_token: if >L="long" take long DA from >PH_Indication
               else take short DA from >PH_Indication
               generate <DA_taken
       if DA=<MA: generate <DA
                  generate <destination="me"
       else generate <destination="other"
```

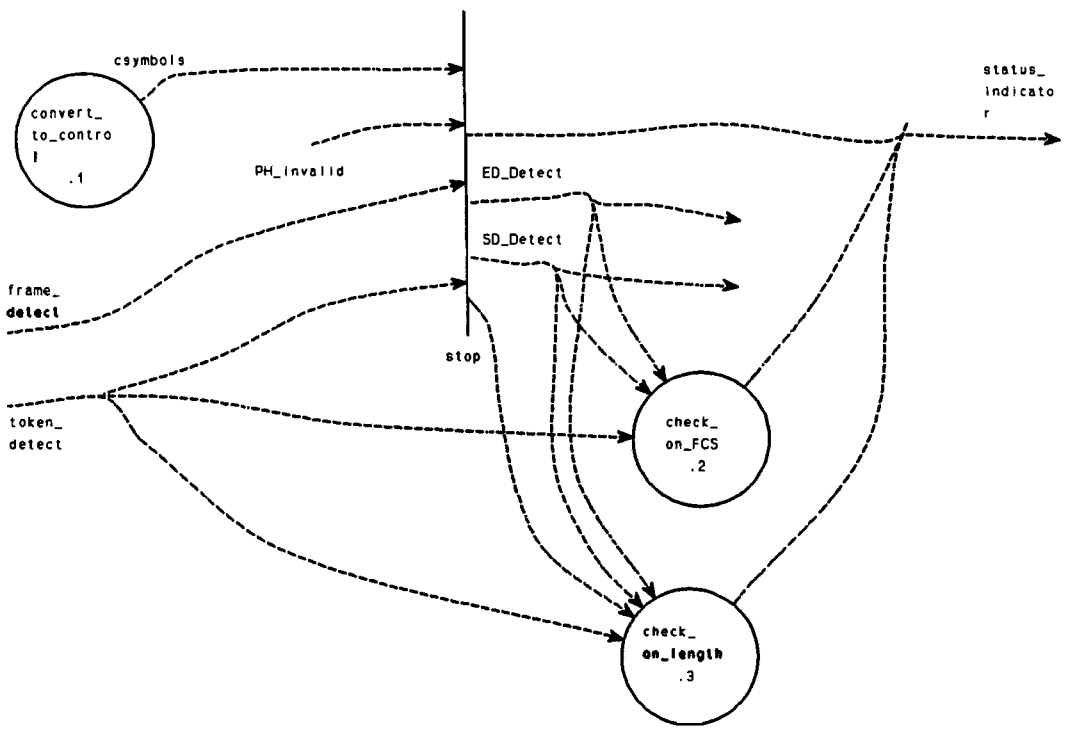
MSP 1.3.3.3.3 check_on_SA

```
on >DA_taken: if >L = "long" take long SA from
                 >PH_Indication
               else take short SA from >PH_Indication
       if SA = >MA: generate <SA
                    generate <source = "me"
       else generate <source = "other"
```

FD 1.3.3.4 check_on_violations



check_on_violations



check_on_violations

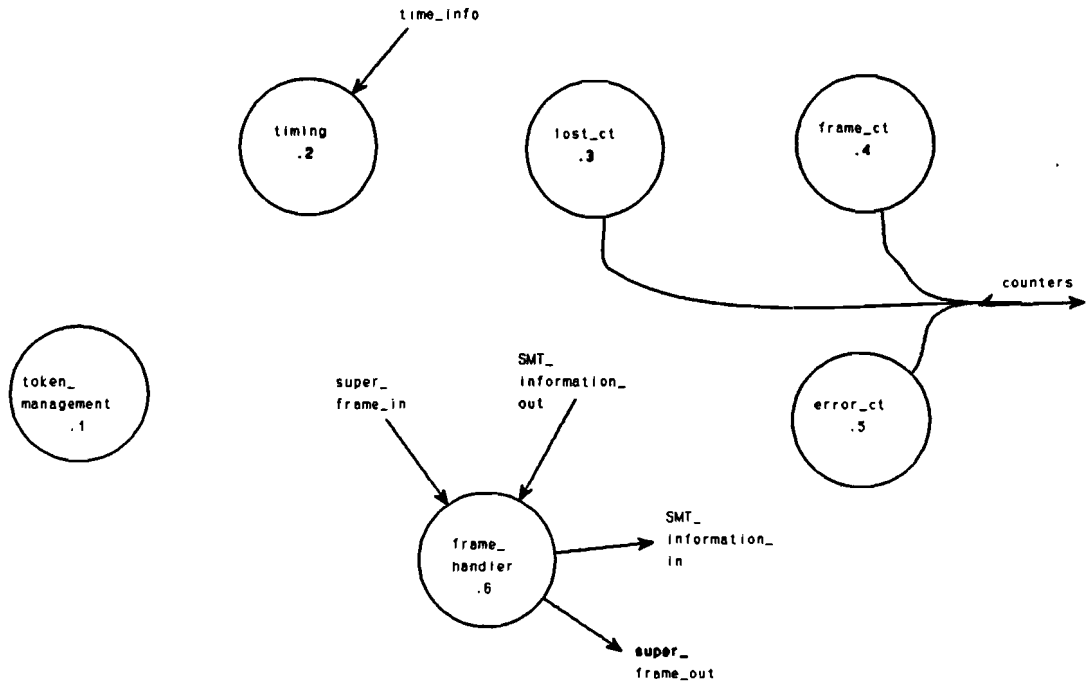
```
MSP 1.3.3.4.1 convert_to_control

- generate <csymbols corresponding to >PH_Indication


MSP 1.3.3.4.2 check_on_FCS

- on >SD_Detect : start making FCS from >PH_Indication, with
   delay of one symbol
- on >ED_Detect : - stop making FCS, and compare the result
                    with a reference value
                  - on error: generate <FCS_error
- on >token_detect : stop making FCS
- on >stop         : stop making FCS


MSP 1.3.3.4.3 check_on_length

* checks whether the length of the frame is even *
- on >SD_Detect : start counting symbols from >PH_Indication
                  (starting with zero)
- on >stop      : stop counting
- on >ED_Detect :
   if >token_detect : if count =? 4 :
                         generate <length_error
                      else          : skip
   else             : if count = even :
                         generate <length_error
```

CS 1.3.3.4 check_on_violations (Sheet 1)

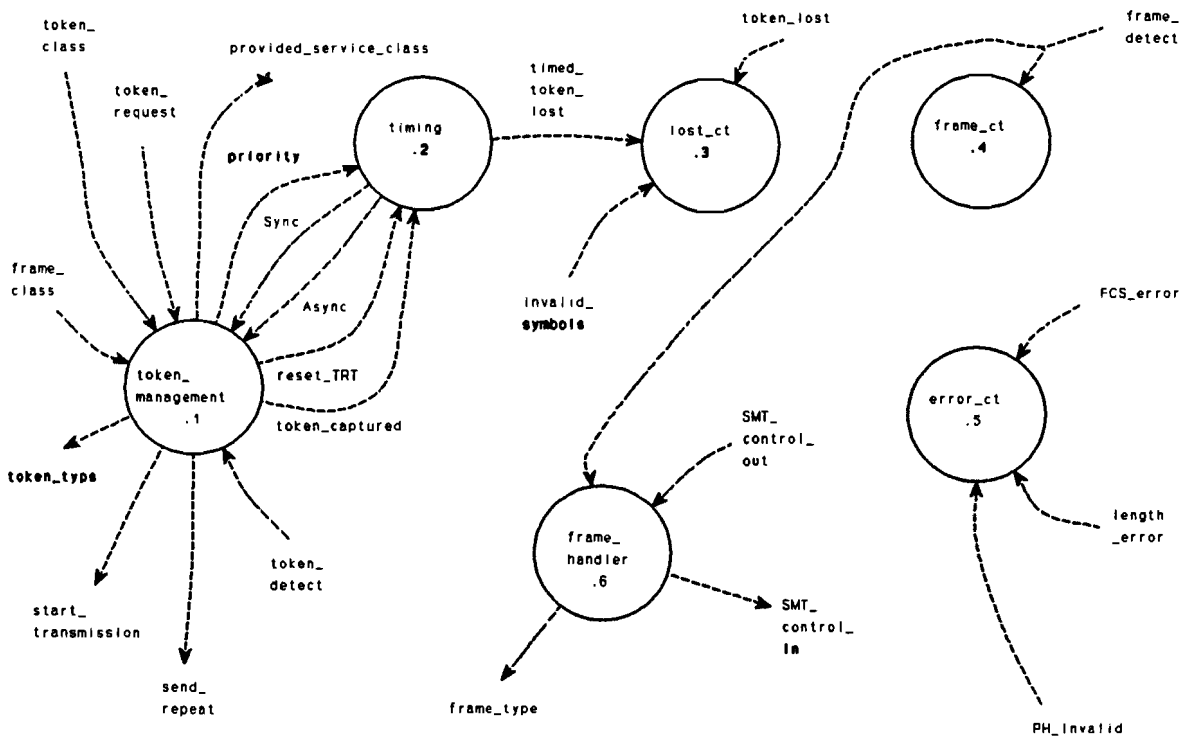
```
Initial state: generate_I

generate_I:
- on >csymbols = "J"
   : goto state J_received

J_received:
- on >csymbols = "K"
   : goto state in_FC
   ; generate <SD_Detect
- on >csymbols =? "K"
   : goto state generate_I
   ; generate <invalid_symbols, <stop
- on >PH_Invalid
   : goto state generate_I
   ; generate <invalid_symbols, <stop

in_FC:
- on >csymbols = "J"
   : goto state J_received
- on >token_detect and >csymbols = "0".."F"
   : goto state expect_2_T
   ; generate <stop
- on >frame_detect and >csymbols = "0".."F"
   : goto state in_frame
- on >csymbols = "0".."F"
   : goto state in_FC
- on >csymbols =? "0".."F", "J"
   : goto state generate_I
   ; generate <invalid_symbols, <stop
- on >PH_Invalid
   : goto state generate_I
   ; generate <invalid_symbols, <stop

expect_2_T:
- on >csymbols = "T"
   : goto state expect_1_T
- on >csymbols =? "T"
   : goto state generate_I
   ; generate <token_lost
- on >PH_Invalid
   : goto state generate_I
   ; generate <invalid_symbols

expect_1_T:
- on >csymbols = "T"
   : goto state generate_I
   ; generate <ED_Detect
- on >csymbols =? "T"
```

```
   : goto state generate_I
   ; generate <length_error
- on >PH_Invalid
   : goto state generate_I
   ; generate <invalid_symbols

in_frame:
- on >csymbols = "0".."F"
   : goto state in_frame
- on >csymbols = "J"
   : goto state J_received
- on >csymbols = "I"
   : goto state generate_I
   ; generate <stop, <strip
- on >csymbols = "T"
   : goto state expect_E
   ; generate <ED_Detect
- on >csymbols =? "0".."F", "I", "J", "T"
   : goto state generate_I
   ; generate <invalid_symbols, <stop
- on >PH_Invalid
   : goto state generate_I
   ; generate <invalid_symbols, <stop

expect_E
- on >csymbols = S or >csymbols = "R"
   : goto state generate_I
   ; generate <strip
- on csymbols =? "R", "S"
   : goto state generate_I
   ; generate <invalid_symbols
- on >PH_Invalid
   : goto state generate_I
   ; generate <invalid_symbols, <stop
```

FD 1.3.5 monitor



monitor



monitor

MSP 1.3.5.1 token_management

```
- on >token_class :
   - wait for >token_detect
   - if >token_class = "Not_specified" :
        capt_token := >token_detect.type
     else :
        capt_token := >token_class
   - generate <provided_service_class = >token_detect.type
              <token_captured
              <token_type = >token_detect.type
   - repeat
      - on frame_class = "Synchronous":
         if >Sync : generate <start_transmission
                             <send_repeat = "send"
      - on >frame_class = "Asynchronous":
         - generate <priority
 - if >Async : generate <start_transmission
                                 <send_repeat = "send"
     until ? >Sync
- on >token_request:
   - wait for >token_detect
- on >token_detect: generate <reset_TRT
```

MSP 1.3.5.2 timing

```
*
 SHT = Synchronous Holding Time
*
- on >token_captured :
   - if late_Ct =? 0 : copy TRT to THT
   - reset TRT to >T_opr, late_Ct to 1, SHT to >T_sync
- repeat
   - if THT =? 0:
      - generate <Sync
      - if THT <T_Pri(>priority): generate <Async
     else :
      - if SHT =? 0 : generate <Sync
  until SHT = 0 or late_Ct = 0
- on >reset_TRT : reset TRT to T_opr, late_Ct to 1
- on TRT expiring: late_Ct := late_Ct - 1
- on TRT = 0 and late_Ct = 0 : generate <timed_token_lost
- on late_Ct = 0 : set THT = 0, SHT = 0
```

MSP 1.3.5.3 lost_ct

```
- on >timed_token_lost or >token_lost or >invalid_symbols :
   increment counter
- generate continuously <nr_lost = counter
```

MSP 1.3.5.4 frame_ct

- on >frame_detect : increment counter
- generate continuously <nr_frames = counter

MSP 1.3.5.5 error_ct

- on >FCS_error or >length_error : increment counter
- generate continuously <nr_errors = counter

MSP 1.3.5.6 frame_handler

- construct MAC and SMT frames from >SMT_information_out and >SMT_control_out and direct to <super_frame_out
- receive frames from >super_frame_in and direct to SMT_information_in and SMT_control_out
- generate <frame_type based on SMT_control_out

DATA_DICTIONARY
****

A

*
 controlflow
 (
 process_FS, in E_A_C
)
 indicates the received address_detect status bit
*

= ["S"
   | "R"]

ack

*
 controlflow
 (
 MAC, process_userdata, receive_frames, process_FS
)
 indicates that a frame is acknowledged by another station
*

= AR
 + CR

AR

*
 controlflow
 (
 in ER_AR_CR
)
 indicates the received address_detect status bit
*

= ["S"
   | "R"]

```
Async

*
 controlflow
 (
 monitor
 )
 indicates that an asynchronous frame may be sent
*

=    [ "True"
     | "False" ]


bypass

*
 controlflow
 (
 FDDI, PMD
 )
 tells PMD to bypass
*

=    [ "do_bypass"
     | "do_not_bypass" ]


C

*
 controlflow
 (
 process_FS, in E_A_C
 )
 indicates the new made frame_copied status bit
*

=    [ "S"
     | "R" ]
```

```
checked_symbols_in

*
 dataflow
 (
 MAC, receive_frames, forward_frames
 )
 symbols copied from symbols in and checked
*

=   symbol


copied

*
 controlflow
 (
 receive_frames, forward_frames, process_FS
 )
 indicates that the frame is entirely copied to frames_in
*

=   [ "True"
    | "False" ]


counters

*
 dataflow
 (
 FDDI, MAC, monitor
 )
 consists of several counter values
*

=   nr_frames
  + nr_lost
  + nr_errors
```

```
CR

*
 controlflow
 (
 in ER_AR_CR
 )
 indicates the received frame_copied status bit
*

=    [ "S"
     | "R" ]


csymbols

*
 controlflow
 (
 check_on_violations
 )
 contains the symbols which are to be checked. This is
 only created to be able to use the symbols coming on
 >symbols_in in the state machine
*

=    symbol


DA

*
 dataflow
 (
 MAC, receive_frames, check-on_function
 )
 destination address supplied by user
*

=    [ long_address
     | short_address ]
```

```
data_in

*
 dataflow
 (
 MAC, receive_frames, forward_frames
 )
 framed pieces for construction of userdata_in, followed by
 received EAC symbols
*

=   symbols_4_bit


DA_taken

*
 controlflow
 (
 check_on_function
 )
 indicates that the destination address is taken from
 >symbols_in
*

=   [ "True"
    | "False" ]


destination

*
 controlflow
 (
 receive_frames, forward_frames, process_FS,
 check_on_function
 )
 indicates that a frame is addressed to this station (me)
 or another (other)
*

=   [ "me"
    | "other" ]
```

```
E

*
 controlflow
 (
 process_FS, in E_A_C
 )
 indicates the new made error status bit
*

=    [ "S"
     | "R" ]


ED_Detect

*
 controlflow
 (
 receive_frames, forward_frames, process_FS,
 check_on_violations
 )
 generated each time an ED is detected after a SD_Detect
*

=    [ "True"
     | "False" ]


ER

*
 controlflow
 (
 in ER_AR_CR
 )
 indicates the received error status bit
*

=    [ "S"
     | "R" ]
```

```
ER_AR_CR

*
 controlflow, store
 (
 process_FS
 )
 contains the received error, address_detect, frame_copied
 status bits
*

=   ER
  + AR
  + CR


E_A_C

*
 controlflow, store
 (
 process_FS
 )
 contains the new made error, address_detect, frame_copied
 status bits
*

=   E
  + A
  + C


FC

*
 dataflow
 (
 MAC, receive_frames, check-on_function
 )
 frame control derived from user_FC
*

=   bits_8
```

```
FCS_error

*
 controlflow
 (
 monitor, in status_indicator
 )
 indicates that an error has occurred while checking the
 frame check sequence. This can be an invalid symbol or a
 violence of the FCS
*

=    [ "True"
     | "False" ]


fiber_in

*
 dataflow
 (
 context_of_FDDI, FDDI, PMD
 )
 light, representing bits, to PMD
*

=    "light"


fiber_out

*
 data_flow
 (
 context_of_FDDI, FDDI, PMD
 )
 ligth, representing bits, from PMD
*

=    "light"
```

```
framed_data

*
 dataflow
 (
 process_userdata
 )
 data assembled with basic controls (FC, DA, SA)
*

=    symbols_4_bit


frames_out

*
 dataflow
 (
 MAC, process_userdata
 )
 framed userdata_out
*

=    symbol


frame_class

*
 controlflow
 (
 MAC, process_userdata, monitor
 )
 indicates the class of frame that is next to be sent
*

=    [ "Synchronous"
     | "Asynchronous" ]
   + ( priority_level )
```

```
frame_detect

*
 controlflow
 (
 MAC, receive_frames, monitor, check_on_function,
 check_on_violations
 )
 indicates the type of frame that is received at the time
*

=    [ "void"
     | "LLC"
     | "mac"
     | "SMT"
     | "implementer" ]


frame_type

*
 controlflow
 (
 MAC, process_userdata, monitor
 )
 indicates one of the following frame types to be sent:
 claim (time), beacon, token, other
*

=    [ "claim"
     | "beacon"
     | "token"
     | "other_type" ]
  + ( time )


FS

*
 dataflow
 (
 receive_frames, forward_frames, process_FS
 )
 group of received_FS and adjusted_FS
*

=    3 { S_or_R } 3
```

got

```
*
 controlflow
 (
 process_FS
 )
 indicates that the received EAC symbols are taken
*

=    [ "True"
     | "False" ]
```

invalid_symbols

```
*
 controlflow
 (
 monitor, in status_indicator
 )
 indicates that invalid symbols are found
*

=    [ "True"
     | "False" ]
```

I_G

```
*
 controlflow
 (
 FDDI, MAC, receive_frames, check_on_function
 )
 indicates an individual or group address of the frame
*

=    [ "Individual"
     | "Group" ]
```

```
L

*
 controlflow
 (
 check_on_function
 )
 indicates the address length used in the frame
*

=    [ "short"
     | "long" ]


length_error

*
 controlflow
 (
 monitor, in status_indicator
 )
 indicates that the received frame is to long (no end
 delimiter received at the boundary), or ends at an ineven
 symbol boundary
*

=    [ "True"
     | "False" ]


MA

*
 dataflow
 (
 FDDI, MAC, process_userdata, receive_frames,
 check_on_function
 )
 some source address of MAC (many are possible)
*

=    [ long_address
     | short_address ]
```

```
MA_transmission_status

*
 controlflow
 (
 context_of_FDDI, FDDI, MAC, process_userdata
 )
 copies the acknowledges of each returned frame, indicating
 that a frame is received or not. The order of the
 acknowledges is the same as the order in which the frames
 were presented
*

=    [ "acknowledge"
     | "not_acknowledge" ]


next_request

*
 controlflow
 (
 process_userdata
 )
 indicates that the current MA_UNITDATA_STATUS.indication
 is ready, and a new one can be generated
*

=    [ "True"
     | "False" ]


no_token

*
 controlflow
 (
 check_on_function
 )
 indicates that the frame that is received now is no token
*

=    [ "True"
     | "False" ]
```

```
nr_errors

*
 dataflow
 (
 in counters
 )
 indicates the number of errors counted
*

=   "integer"


nr_frames

*
 dataflow
 (
 in counters
 )
 indicates the number of frames received
*

=   "integer"


nr_lost

*
 dataflow
 (
 in counters
 )
 indicates the number of token and frames lost during
 receiption
*

=   "integer"
```

```
number_of_SDUs

*
 controlflow
 (
 context_of_FDDI, FDDI, MAC, process_userdata
 )
 indicates the number of SDUs of one request that were
 transmitted
*

=   "integer"


OF_UF

*
 controlflow
 (
 PHY
 )
 indicates the elasticity buffer overflow or underflow
*

=   [ "True"
    | "False" ]


oper_control_in

*
 controlflow
 (
 context_of_FDDI, FDDI
 )
 flow by which the operator can observe the operation of
 the FDDI
*

=   unspecified_group
```

```
oper_control_out

*
 controlflow
 (
 context_of_FDDI, FDDI
 )
 flow by which the operator can observe the operation of
 the FDDI
*

=   unspecified_group


oper_data_in

*
 dataflow
 (
 context_of_FDDI, FDDI
 )
 flow by which the operator can observe the current
 settings of the FDDI
*

=   unspecified_group


oper_data_out

*
 dataflow
 (
 context_of_FDDI, FDDI
 )
 flow by which the operator can supply settings to the
 FDDI
*

=   unspecified_group
```

```
optic_in

*
 dataflow
 (
 PMD
 )
 light received by the PMD, coming from the bypass
*
=   "light"


optic_out

*
 dataflow
 (
 PMD
 )
 light generated in PMD, directed to the bypass
*
=   "light"


PH_Indication

*
 dataflow
 (
 FDDI, PHY, MAC, receive_frames, froward_frames,
 process_FS, check_on_function, check-on_violations
 )
 data, constructing frames, from PHY to MAC
*
=   symbol
```

```
PH_Invalid

*
 controlflow
 (
 FDDI, PHY, MAC, receive_frames, monitor,
 check_on_violations
 )
 indicates that the current frame received is faulty
*

=    [ "True"
     | "False" ]


PH_Request

*
 dataflow
 (
 FDDI, PHY, MAC
 )
 data, constructing frames, from MAC to PHY
*

=    symbol


PH_transmission_status

*
 controlflow
 (
 FDDI, PHY, MAC
 )
 indicates that the previous symbol is processed, and the
 next can be transmitted
*

=    [ "True"
     | "False" ]
```

```
PM_Indication

*
 dataflow
 (
 FDDI, PHY, PMD
 )
 data, constructing symbols, from PMD to PHY
*

=   bit


PM_Request

*
 dataflow
 (
 FDDI, PMD, PHY
 )
 data, constructing symbols, from PHY to PMD
*

=   bit


priority

*
 controlflow
 (
 monitor
 )
 indicates the priority of the next frame to be send,
 extracted from frame_class
*

=   priority_level
```

provided_service_class

*
controlflow
(
context_of_FDDI, FDDI, MAC, monitor
)
indicates the service class that was used for the
currently indicated request (restricted or unrestricted)
*

= ["Restricted"
| "Unrestricted"]

ready

*
controlflow
(
MAC, process_userdata
)
indicates that there are no more frames to be sent at the
time
*

= ["True"
| "False"]

received_FS

*
dataflow
(
MAC, receive_frames, process_FS
)
gives the frame status as received with the frame by this
station
*

= 3 { S_or_R } 3

```
reception_status

*
 controlflow
 (
 context_of_FDDI, FDDI, MAC
 )
 indicates to LLC the status of reception of the frame
 just send
*

=    [ "fr_good"
     | "error_in_FCS"
     | "error_in_length"
     | "internal_error" ]
   + frame_status


recovered_clk

*
 controlflow
 (
 PHY
 )
 clock obtained from incoming data
*

=    "clock"


requested_service_class

*
 controlflow
 (
 context_of_FDDI, FDDI, MAC, process_userdata
 )
 indicates the service class that is required for this
 frame (restricted or unrestricted)
*

=    [ "Restricted"
     | "Unrestricted" ]
```

```
request_data

*
 control flow
 (
 process_userdata
 )
*

=    [ request
     | norequest ]


reset_TRT

*
 controlflow
 (
 monitor
 )
 indicates that a token has passed, but not captured, and
 tells that TRT has to be reset
*

=    "pulse"


SA

*
 dataflow
 (
 MAC, receive_frames, check_on_function
 )
 source address of the frame
*

=    [ long_address
     | short_address ]
```

```
SD_Detect

*
 controlflow
 (
 receive_frames, forward_frames, check_on_function,
 check_on_violations
 )
 generated each time a SD is detected
*

=   "pulse"


send_repeat

*
 controlflow
 (
 MAC, monitor
 )
 tells wether frames_out or checked_symbols_in are to be
 transmitted
*

=   [ "send"
    | "repeat" ]


Signal_Detect

*
 controlflow
 (
 FDDI, PMD, PHY
 )
 indicates that the ligth power level is above threshold
*

=   [ "T"
    | "F" ]
```

SMT_control_in

*
 controlflow
 (
 FDDI, MAC, monitor
)
 contains the control flows of MAC to SMT
*

= unspecified_group

SMT_control_out

*
 controlflow
 (
 FDDI, MAC, monitor
)
 contains the control flows of SMT to MAC
*

= unspecified_group

SMT_information_in

*
 dataflow
 (
 FDDI, MAC, monitor
)
 contains the data flows of MAC to SMT
*

= unspecified_group

```
SMT_information_out

*
 dataflow
 (
 FDDI, MAC, monitor
 )
 contains the data flows of SMT to MAC
*

=   unspecified_group


source

*
 controlflow
 (
 receive_frames, forward_frames, check_on_function
 )
 indicates that a frame comes from this station (me) or
 another (other)
*

=   [ "me"
    | "other" ]


start_transmission

*
 controlflow
 (
 MAC, process_userdata, monitor
 )
 indicates that the transmission can start
*

=   [ "True"
    | "False" ]
```

status_indicator

*
 controlflow
 (
 MAC, receive_frames, forward_frames, process_FS,
 check_on_violations
)
 indicates that a frame is bad or that idles have to be
 sent. It consists of several components.
*

= FCS_error
  + length_error
  + token_lost
  + invalid_symbols
  + strip

stop

*
 controlflow
 (
 check_on_violations
)
 indicates the FCS check and the length check to stop
*

= ["True"
   | "False"]

stream

*
 controlflow
 (
 context_of_FDDI, FDDI, MAC, process_userdata
)
 indicates that another frame is following this frame
*

= ["True"
   | "False"]

```
strip

*
 controlflow
 (
 in status_indicator
 )
 indicates that such a situation is found that the rest of
 the frame may be stripped
*

=    [ "True"
     | "False" ]


super_frame_in

*
 dataflow
 (
 MAC, monitor
 )
 contains frames to MAC and SMT (SMT information is extracted)
*

=    symbols_4_bit


super_frame_out

*
 dataflow
 (
 MAC, process_userdata, monitor
 )
 contains frames from MAC and SMT (SMT information is
 supplied by SMT)
*

=    symbols_4_bit
```

```
symbol_in

*
 dataflow
 (
 PHY
 )
 symbols constructed from the incoming bits. In case the
 boundaries have not yet been determined, symbol_in changes
 every bit time, else every five bit times
*

=   symbol


symbol_out

*
 dataflow
 (
 PHY
 )
 symbols to be serialized. Changes every five bit times
*

=   symbol


Sync

*
 controlflow
 (
 monitor
 )
 indicates that a synchronous frame may be sent
*

=   [ "True"
    | "False" ]
```

```
sync_bits

*
 dataflow
 (
 PHY
 )
 data bits synchronized to the local clock
*

=   bit


timed_token_lost

*
 controlflow
 (
 monitor
 )
 indicates that the token has not arrived within 2 T_opr
*

=   "pulse"


time_info

*
 dataflow
 (
 FDDI, MAC, monitor
 )
 contains information about the timing parameters, the
 station is using; T_sync = x% of T_opr
*

=   T_sync
  + T_opr
  + T_Pri
```

token_captured

*
 controlflow
 (
 monitor
)
 indicates that a tokenis detected, and captured. TRT must
 be copied into THT; TRT must be reset to T_opr, SHT must
 be reset to T_sync (Synchronous Holding Time)
*

= "pulse"

token_class

*
 controlflow
 (
 context_of_FDDI, FDDI, MAC, monitor
)
 indicates the type of token that has to be issued after
 the corresponding request has been served; in case of N
 the captured token must be reissued
*

= ["Restricted"
   | "Unrestricted"
   | "Not_specified"]

token_detect

*
 controlflow
 (
 MAC, receive_frames, monitor, check_on_function,
 check_on_violations
)
 indicates that some token is detected
*

= ["Restricted"
   | "Unrestricted"]

```
token_lost

*
 controlflow
 (
 monitor, in status_indicator
 )
 indicates that a token is lost
*

=   "pulse"


token_request

*
 controlflow
 (
 context_of_FDDI, FDDI, MAC, monitor
 )
 indicates that a token must be captured and hold and the
 type requested
*

=   [ "Restricted"
    | "Unrestricted" ]
  + ( priority_level )


token_type

*
 controlflow
 (
 MAC, process_userdata, monitor
 )
 indicates the type of token that is to be sent
*

=   [ "Restricted"
    | "Unrestricted" ]
```

```
T_opr

*
 dataflow
 (
 in time_info
 )
 gives the operational target token rotation time to be
 used by MAC (determined via claim process)
*

=   time


T_Pri

*
 dataflow
 (
 in time_info
 )
 contains the thresshold value for each priority
*

=   T_Pri0
  + T_Pri1
  + T_Pri2
  + T_Pri3
  + T_Pri4
  + T_Pri5
  + T_Pri6
  + T_Pri7


T_Pri0

*
 dataflow
 (
 in T_Pri
 )
 contains the threshold value for level 0
*

=   time
```

```
T_Pri1

*
 dataflow
 (
 in T_Pri
 )
 contains the threshold value for level 1
*
=   time


T_Pri2

*
 dataflow
 (
 in T_Pri
 )
 contains the threshold value for level 1
*
=   time


T_Pri3

*
 dataflow
 (
 in T_Pri
 )
 contains the threshold value for level 3
*
=   time
```

```
T_Pri4

*
 dataflow
 (
 in T_Pri
 )
 contains the threshold value for level 4
*
=   time


T_Pri5

*
 dataflow
 (
 in T_Pri
 )
 contains the threshold value for level 5
*
=   time


T_Pri6

*
 dataflow
 (
 in T_Pri
 )
 contains the threshold value for level 6
*
=   time
```

```
T_Pri7

*
 dataflow
 (
 in T_Pri
 )
 contains the threshold value for level 7
*

=   time


T_sync

*
 dataflow
 (
 in time_info
 )
 gives the allowed synchronous transmission time determined
 by SMT
*

=   time


userdata_in

*
 dataflow
 (
 context_of_FDDI, FDDI, MAC
 )
 data from MAC to LLC
*

=   symbols_4_bit
```

userdata_out

*
 dataflow
 (
 context_of_FDDI, FDDI, MAC, process_userdata
)
 data from LLC to MAC, contains FC, DA and SDU
*

= symbols_4_bit

U_L

*
 controlflow
 (
 FDDI, MAC, receive_frames, check_on_function
)
 indicates a universal or local administration of the
 address
*

= ["Universal"
   | "Local"]

ERROR/WARNING STATISTICS
========================

 severe errors : 0
 errors detected : 0

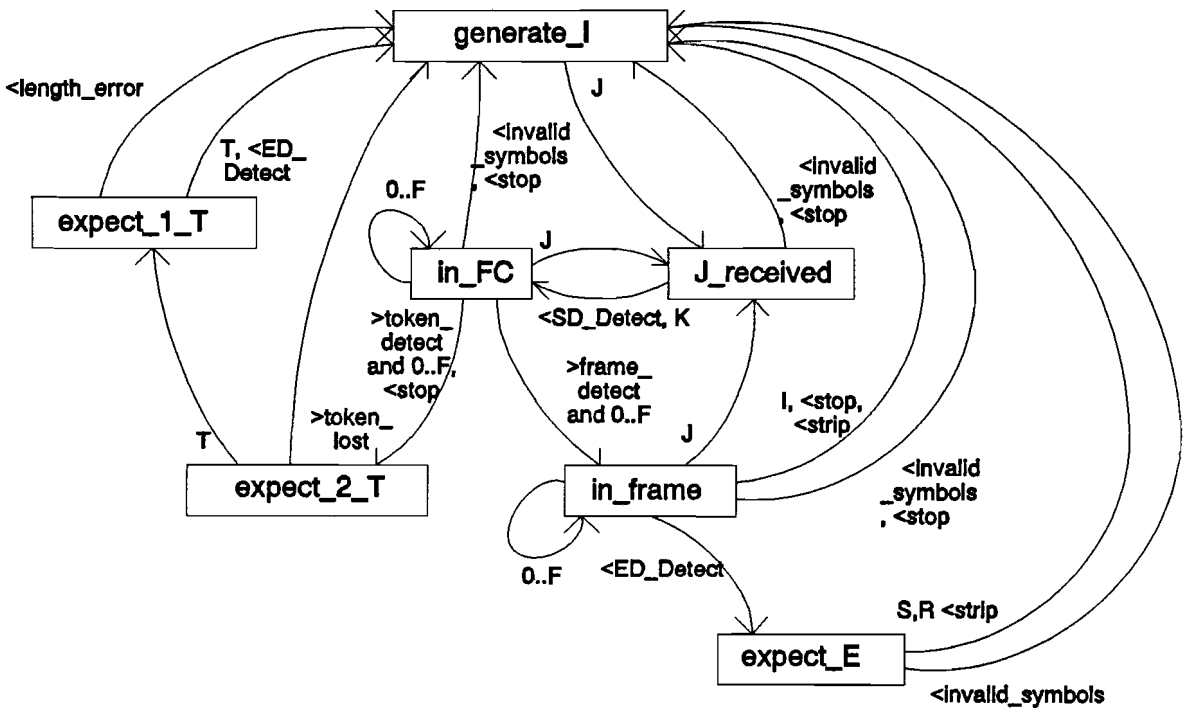
 system model may be transformed

Appendix G: Graphic representation of control spec check_on_violations

This graphic representation of the control spec is included to improve the insight in the operation of the control spec. The textual specification can be found on p29 of the PROMOD SA analysis report in appendix F.

Most arrows have an expression that must be satisfied to take the arrow to another state. If not it means that the arrow must be taken when none of the expressions on the other arrows are satisfied. The > sign indicates an incoming flow that is tested. Symbols names on with the arrows mean that these symbols occur on the flow >csymbols.

With most arrows an action is given via a generation of an outgoing flow. This is indicated by the < sign.



Appendix H: Files made with IDaSS

The diskette labelled with IDaSS contains the following files made with IDaSS:

\IDASS\PHY.DES	Design of process PHY
\IDASS\PROCUSER.DES	Design of process process_userdata
\HEX\MA.HEX	File that process_userdata\store\MA writes data to and reads data from (required for start-up)
\HEX\TESTING.HEX	File that process_userdata\LLC_data\testing writes data to and reads data from (required for start-up)
\HEX\	MA_BASE: Base contents of MA (needed for restoring) ONE_SHRT: Test pattern for one frame per request with short address (for process_userdata\LLC_data) TWO_SHRT: Test pattern for two frames per request with short address (for process_userdata\LLC_data)

When a file is loaded in a memory, it must always be saved to the file <memory_name>.hex before saving the total design after a simulation cycle. This is because the memories are saved to the latest access file, destroying the contents in the latest used file.

Appendix I: IDaSS description of decoder

This text is taken from a system document generated by IDaSS.

```
Text for function 'decode' of 'PHY\decode\decoder':
----------------------v----------------------
PH_Indication := (
(symbol = 11110b)
 if1: 0000b
 if0:
((symbol = 01001b)
 if1: 0001b
 if0:
((symbol = 10100b)
 if1: 0010b
 if0:
((symbol = 10101b)
 if1: 0011b
 if0:
((symbol = 01010b)
 if1: 0100b
 if0:
((symbol = 01011b)
 if1: 0101b
 if0:
((symbol = 01110b
 if1: 0110b
 if0:
((symbol = 01111b)
 if1: 0111b
 if0:
((symbol = 10010b)
 if1: 1000b
 if0:
((symbol = 10011b)
 if1: 1001b
 if0:
((symbol = 10110b)
 if1: 1010b
 if0:
((symbol = 10111b)
 if1: 1011b
 if0:
((symbol = 11010b)
 if1: 1100b
 if0:
((symbol = 11011b)
 if1: 1101b
 if0:
((symbol = 11100b)
 if1: 1110b
 if0:
```

```
((symbol = 11101b)
 if1: 1111b
 if0: dummy

))))))))))))))))
).

data := (
(symbol = 11110b)
 if1: 0b
 if0:
((symbol = 01001b)
 if1: 0b
 if0:
((symbol = 10100b)
 if1: 0b
 if0:
((symbol = 10101b)
 if1: 0b
 if0:
((symbol = 01010b)
 if1: 0b
 if0:
((symbol = 01011b)
 if1: 0b
 if0:
((symbol = 01110b
 if1: 0b
 if0:
((symbol = 01111b)
 if1: 0b
 if0:
((symbol = 10010b)
 if1: 0b
 if0:
((symbol = 10011b)
 if1: 0b
 if0:
((symbol = 10110b)
 if1: 0b
 if0:
((symbol = 10111b)
 if1: 0b
 if0:
((symbol = 11010b)
 if1: 0b
 if0:
((symbol = 11011b)
 if1: 0b
 if0:
((symbol = 11100b)
 if1: 0b
 if0:
((symbol = 11101b)
```

```
 if1: 0b
 if0: ((dummy) from: 0 to: 0)

))))))))))))))))
).

J := (
(symbol = 11000b)
 if1: 0b
 if0: ((dummy) from: 0 to: 0)
).

K := (
(symbol = 10001b)
 if1: 0b
 if0: ((dummy) from: 0 to: 0)
).

T := (
(symbol = 01101b)
 if1: 0b
 if0: ((dummy) from: 0 to: 0)
).

R := (
(symbol = 00111b)
 if1: 0b
 if0: ((dummy) from: 0 to: 0)
).

S := (
(symbol = 11001b)
 if1: 0b
 if0: ((dummy) from: 0 to: 0)
).

I := (
(symbol = 11111b)
 if1: 0b
 if0: ((dummy) from: 0 to: 0)
).

PH_Invalid := (
(Signal_Detect \/ ((OF_UF) not))
 if1: 0b
 if0:
((symbol = 00000b)
 if1: 0b
 if0:
((symbol = 00100b)
 if1: 0b
 if0:
((symbol = 00001b)
 if1: 0b
```

```
 if0:
((symbol = 00010b)
 if1: 0b
 if0:
((symbol = 00011b)
 if1: 0b
 if0:
((symbol = 00101b)
 if1: 0b
 if0:
((symbol = 00110b)
 if1: 0b
 if0:
((symbol = 01000b)
 if1: 0b
 if0:
((symbol = 01100b)
 if1: 0b
 if0:
((symbol = 10000b)
 if1: 0b
 if0: ((dummy) from: 0 to: 0)
))))))))))
)
----------------------^----------------------

End of function descriptions.
```

Appendix J: IDaSS description of encoder

This text is taken from a system document generated by IDaSS.

```
Text for function 'encode' of 'PHY\encode\encoder':
----------------------v----------------------
symbolout := (
(Jout = 0b)
 if1: 11000b
 if0:
((Kout = 0b)
 if1: 10001b
 if0:
((Tout = 0b)
 if1: 01101b
 if0:
((Rout = 0b)
 if1: 00111b
 if0:
((Sout = 0b)
 if1: 11001b
 if0:
((dataout = 0b)
 if0: dummy
 if1:
 ((PH_Request = 0000b)
  if1: 11110b
  if0:
 ((PH_Request = 0001b)
  if1: 01001b
  if0:
 ((PH_Request = 0010b)
  if1: 10100b
  if0:
 ((PH_Request = 0011b)
  if1: 10101b
  if0:
 ((PH_Request = 0100b)
  if1: 01010b
  if0:
 ((PH_Request = 0101b)
  if1: 01011b
  if0:
 ((PH_Request = 0110b)
  if1: 01110b
  if0:
 ((PH_Request = 0111b)
  if1: 01111b
  if0:
 ((PH_Request = 1000b)
  if1: 10010b
  if0:
 ((PH_Request = 1001b)
  if1: 10011b
  if0:
```

```
 ((PH_Request = 1010b)
  if1: 10110b
  if0:
 ((PH_Request = 1011b)
  if1: 10111b
  if0:
 ((PH_Request = 1100b)
  if1: 11010b
  if0:
 ((PH_Request = 1101b)
  if1: 11011b
  if0:
 ((PH_Request = 1110b)
  if1: 11100b
  if0:
 ((PH_Request = 1111b)
  if1: 11101b
  if0: dummy "this option never exists, only added because
              it is required for IDaSS"
))))))))))))))
)))))
)
----------------------^----------------------

End of function descriptions.
```

Appendix K: IDaSS description of read_control

This text is taken from a system document generated by IDaSS.

```
Text for state number 1 (reset state) of
'process_userdata\read_control':
----------------------v----------------------
wait_until_request:
[store\write_request\begin = store\write_request\end
|1 <<
|0 store\write_frame\ft_addr_r load;
   store\write_frame\ft_end load;
   ready setto: 1;
   -> request_ready
]
----------------------^----------------------

Text for state number 2 of 'process_userdata\read_control':
----------------------v----------------------
request_ready:
store\write_frame\ft_type_valid setto: 0;
store\write_symbol\read_address load;
store\write_symbol\end_address load;
->request_to_send
----------------------^----------------------

Text for state number 3 of 'process_userdata\read_control':
----------------------v----------------------
request_to_send:
[_temp_end := (store\write_frame\ft_end -1 from: 0 to: 4).
 store\write_frame\ft_addr_r = _temp_end
|1 store\write_frame\ft_type_valid setto: 1;
];
nr_SDUs setto: 0;
store\write_frame\ft_addr_r inc;
-> wait_for_request_data
----------------------^----------------------

Text for state number 4 of 'process_userdata\read_control':
----------------------v----------------------
wait_for_request_data:
[add_controls\request_data
|1 <<
|0 nr_SDUs inc;
   store\write_symbol\strobing_symbols setto: 0;
   -> produce_frame
]
----------------------^----------------------
```

```
Text for state number 5 of 'process_userdata\read_control':
----------------------v----------------------
produce_frame:
store\write_symbol\read_address inc;
store\write_symbol\memory enable: symbols_d;
[_temp_end := (store\write_symbol\end_address - 1 from: 0 to:
10).
 store\write_symbol\read_address = _temp_end
|0 "already strobing via strobing_symbols"
   <<
|1 store\write_symbol\strobing_symbols setto: 1;
   [(start_transmission=0) /\
    (store\write_frame\ft_addr_r ~=
     store\write_frame\ft_end)
   |1 "another frame"
      store\write_symbol\read_address inc;
      store\write_symbol\end_address load;
      -> hold_until_frame_done
   |0 "end request or end of time"
      store\write_frame\ft_type_valid setto: 1;
      store\write_request\begin inc;
      [frame_type = 01b
      |1 nr_SDUs_valid setto: 0
      ];
      -> hold_until_request_done
   ]
]
----------------------^----------------------

Text for state number 6 of 'process_userdata\read_control':
----------------------v----------------------
hold_until_frame_done:
[done
|1 <<
|0 [_temp_end := (store\write_frame\ft_end -1 from: 0 to: 4).
    store\write_frame\ft_addr_r = _temp_end
   |1 store\write_frame\ft_type_valid setto: 1;
   ];
   store\write_frame\ft_addr_r inc;
   done setto: 1;
   -> wait_for_request_data
]
----------------------^----------------------
```

```
Text for state number 7 of 'process_userdata\read_control':
---------------------v----------------------
hold_until_request_done:
[done
|1 <<
|0 done setto: 1;
   ready setto: 0;
   nr_SDUs_valid setto: 1;
   -> wait_until_request
]
---------------------^----------------------

End of state descriptions.
```

Appendix L: IDaSS description of add_controls_contr

This text is taken from a system document generated by IDaSS.

```
Text for state number 1 (reset state) of
'process_userdata\add_controls_contr':
----------------------v---------------------
wait:
[start_transmission
|0 add_controls\generate_controls I;
   add_controls\counter dec;
   -> PA
|1 <<
]
----------------------^---------------------

Text for state number 2 of 'process_userdata\add_controls_contr':
----------------------v---------------------
PA:
[add_controls\counter = 0
|1 add_controls\generate_controls J;
   add_controls\request_data setto: 0;
   -> J
|0 add_controls\counter dec;
   add_controls\generate_controls I;
   <<
]
----------------------^---------------------

Text for state number 3 of 'process_userdata\add_controls_contr':
----------------------v---------------------
J:
add_controls\generate_controls K;
add_controls\request_data setto:1;
-> K
----------------------^---------------------

Text for state number 4 of 'process_userdata\add_controls_contr':
----------------------v---------------------
K:
[store\write_symbol\strobing_symbols
|0 add_controls\generate_controls data;
   "make FCS"
   <<
|1 [frame_type ~= 00b
   |1 add_controls\generate_controls data;
      FCS\FCS_addressing inc;
      FCS\FCS_enable;
      -> add_FCS
   |0 add_controls\generate_controls T;
      -> add_2_T
   ]
]
----------------------^---------------------
```

```
Text for state number 5 of 'process_userdata\add_controls_contr':
----------------------v----------------------
add_2_T:
add_controls\generate_controls T;
done setto: 0;
-> wait
----------------------^----------------------

Text for state number 6 of 'process_userdata\add_controls_contr':
----------------------v----------------------
add_FCS:
[FCS\FCS_addressing = 0
|0 FCS\FCS_addressing inc;
   add_controls\generate_controls data;
   FCS\FCS enable;
   <<
|1 add_controls\generate_controls T;
   add_controls\counter setto: 2;
   FCS\reset setto:0;
  -> ED
]
----------------------^----------------------

Text for state number 7 of 'process_userdata\add_controls_contr':
----------------------v----------------------
ED:
add_controls\generate_controls R;
[add_controls\counter = 0
|1 done setto: 0;
   -> wait
|0 add_controls\counter dec;
   <<
]
----------------------^----------------------

End of state descriptions.
```

Appendix M: IDaSS description of store_address_contr

This text is taken from a system document generated by IDaSS.

```
Text for state number 1 (reset state) of
'process_userdata\store\store_address_contr':
----------------------v---------------------
wait_and_addressing_till_data:
"space is available"
[(active_out\/stream\/strobing_out)
|1 [(nr_done=0)
   |1 write_symbol\mux continue;
      <<
   |0 "FC or DA lost"
      -> loose_request
   ]
|0  [((S_L /\ (nr_done=5)) \/ ((S_L not) /\ (nr_done=13)))
   |1 write_symbol\store_address loadinc;
      write_symbol\memory write: data_out_d;
      write_symbol\memory write: MA_d;
      [_temp_store_addr := (write_symbol\store_address + 1 +
       ((S_L)
        if1: (4 width: 11)
        if0: 12) from: 0 to: 10).
       write_symbol\read_address = _temp_store_addr
      |0 -> addressing_data
      |1 full setto: 0;
         -> wait_until_free_space
      ]
   |0 write_symbol\store_address inc;
      nr_done inc;
      write_symbol\memory write: data_out_d;
      [(nr_done>=2)
      |1 write_symbol\memory write: MA_d;
         MA\MA_address inc
      |0
      ];
      [_temp_store:=(write_symbol\store_address + 1 +
                    ((S_L)
                     if1: (4 width: 11)
                     if0: 12
                    ) from: 0 to: 10).
       write_symbol\read_address = _temp_store
      |0 <<
      |1 full setto: 0;
         -> wait_until_free_space
      ]
   ]
]
----------------------^---------------------
```

```
T e x t   f o r   s t a t e   n u m b e r   2   o f
'process_userdata\store\store_address_contr':
----------------------v----------------------
addressing_data:
[(active_out\/stream\/strobing_out)
|0 write_symbol\store_address inc;
   write_symbol\memory write: data_out_d;
   [_temp_store_addr := (write_symbol\store_address + 1 from: 0
to: 10).
    write_symbol\read_address = _temp_store_addr
   |0 <<
   |1 full setto: 0;
      -> wait_until_free_space
   ]
|1 [(stream\/strobing_out) "active_out not"
   |0 "end of data, following RSC(PL)"
      write_frame\ft_type write: data_out_d;
      write_frame\ft_start write: store_addr_d;
      write_frame\ft_address inc;
      nr_done reset; "for next frame, this one is ok"
      [_temp_ft_addr := (write_frame\ft_address + 2) from: 0 to:
4.
       write_frame\ft_addr_r = _temp_ft_addr
      |0 -> next_frame
      |1 full setto: 0;
         -> wait_until_free_space
      ]
   |1 "RSC(PL) not found"
      -> loose_request
   ]
]
----------------------^----------------------
```

```
T e x t   f o r   s t a t e   n u m b e r   3   o f
'process_userdata\store\store_address_contr':
----------------------v----------------------
next_frame:
[(active_out\/stream\/strobing_out)
|0 "new frame"
   write_symbol\store_address inc;
   nr_done inc;
   write_symbol\memory write: data_out_d;
   [_temp_store_addr := (write_symbol\store_address + 1 from: 0
to: 10).
    write_symbol\read_address = _temp_store_addr
   |0 -> wait_and_addressing_till_data
   |1 full setto: 0;
      -> wait_until_free_space
   ]
|1 [((active_out not)\/(stream not)\/strobing_out)
   |0 "end request, following token_class"
      write_request\rt_type write: data_out_d; "token class"
      write_request\rt_start write: ft_address_d;
      write_request\end inc;
      [_temp_end := (write_request\end + 2) from: 0 to: 2.
       write_request\begin = _temp_end
      |0 -> wait_and_addressing_till_data
      |1 full setto: 0;
         -> wait_until_free_space
      ]
   |1 "token class not found"
      -> loose_request
   ]
]
----------------------^----------------------

T e x t   f o r   s t a t e   n u m b e r   4   o f
'process_userdata\store\store_address_contr':
----------------------v----------------------
wait_until_free_space:
[_temp_end := (write_request\end+1) from: 0 to: 2.
 _temp_ft_addr := (write_frame\ft_address+1) from: 0 to:4.
 (write_request\begin = _temp_end)
 \/(write_frame\ft_addr_r = _temp_ft_addr)
 \/(write_symbol\read_address = write_symbol\store_address)
|1 <<  "no free space"
|0 full setto: 1;
   -> loose_request
]
----------------------^----------------------
```

```
T e x t    f o r    s t a t e    n u m b e r    5    o f
'process_userdata\store\store_address_contr':
----------------------v----------------------
loose_request:
write_frame\ft_address load;
-> restore_frame_address
----------------------^----------------------

T e x t    f o r    s t a t e    n u m b e r    6    o f
'process_userdata\store\store_address_contr':
----------------------v----------------------
restore_frame_address:
write_symbol\mux new;
write_symbol\store_address load;
nr_done reset;
-> wait_until_end_request
----------------------^----------------------

T e x t    f o r    s t a t e    n u m b e r    7    o f
'process_userdata\store\store_address_contr':
----------------------v----------------------
wait_until_end_request:
[(active_out/\stream)
|1 -> wait_and_addressing_till_data
|0 <<
]
----------------------^----------------------

End of state descriptions.
```

Appendix N: IDaSS description of addr_l

This text is taken from a system document generated by IDaSS.

```
Text for state number 1 (reset state) of
'process_userdata\store\addr_l':
----------------------v----------------------
waiting:

[(active_out\/stream\/strobing_out)
|1 << "no symbol found"
|0 [(data_out from:2 to: 2)
    |1 S_L setto: 1 "short address"
    |0 S_L setto: 0 "long address"
   ];
   ->listening_frame
]
----------------------^----------------------

Text for state number 2 of 'process_userdata\store\addr_l':
----------------------v----------------------
listening_frame:

[(active_out\/stream\/strobing_out)
|0 <<
|1 -> waiting
]
----------------------^----------------------

End of state descriptions.
```

Appendix O: IDaSS description of MA_contr

This text is taken from a system document generated by IDaSS.

```
Text for state number 1 (reset state) of
'process_userdata\store\MA_contr':
----------------------v----------------------
wait_for_frame:
[S_L
|1 "short"
   MA\MA_address setto: 12
|0 "long"
];
-> wait_for_end_of_frame
----------------------^----------------------

Text for state number 2 of 'process_userdata\store\MA_contr':
----------------------v----------------------
wait_for_end_of_frame:
[(nr_done=0)
|1 MA\MA_address setto: 0;
   -> wait_for_frame
|0 <<
]
----------------------^----------------------

End of state descriptions.
```

Appendix P: IDaSS description of mux

This text is taken from a system document generated by IDaSS.

```
Text for function 'continue' of
'process_userdata\store\write_symbol\mux':
----------------------v----------------------
mux := I1
----------------------^----------------------

Text for function 'new' of
'process_userdata\store\write_symbol\mux':
----------------------v----------------------
mux := I2
----------------------^----------------------

End of function descriptions.
```

Appendix Q: IDaSS description of add_4_or_12

This text is taken from a system document generated by IDaSS.

```
Text for function 'convert' of
'process_userdata\store\write_symbol\add_4_or_12':
----------------------v----------------------
MA_address :=
((S_L)
 if1: store_address + 4
 if0: store_address + 12
)
----------------------^----------------------

End of function descriptions.
```

Appendix R: IDaSS description of generate_controls

This text is taken from a system document generated by IDaSS.

```
T e x t   f o r   f u n c t i o n   ' d a t a '   o f
'process_userdata\add_controls\generate_controls':
---------------------v---------------------
I:=1. J:=1. K:=1. T:=1. R:=1. S:=1. data:=0
---------------------^---------------------

T e x t   f o r   f u n c t i o n   ' I '   o f
'process_userdata\add_controls\generate_controls':
---------------------v---------------------
I:=0. J:=1. K:=1. T:=1. R:=1. S:=1. data:=1
---------------------^---------------------

T e x t   f o r   f u n c t i o n   ' J '   o f
'process_userdata\add_controls\generate_controls':
---------------------v---------------------
I:=1. J:=0. K:=1. T:=1. R:=1. S:=1. data:=1
---------------------^---------------------

T e x t   f o r   f u n c t i o n   ' K '   o f
'process_userdata\add_controls\generate_controls':
---------------------v---------------------
I:=1. J:=1. K:=0. T:=1. R:=1. S:=1. data:=1
---------------------^---------------------

T e x t   f o r   f u n c t i o n   ' R '   o f
'process_userdata\add_controls\generate_controls':
---------------------v---------------------
I:=1. J:=1. K:=1. T:=1. R:=0. S:=1. data:=1
---------------------^---------------------

T e x t   f o r   f u n c t i o n   ' S '   o f
'process_userdata\add_controls\generate_controls':
---------------------v---------------------
I:=1. J:=1. K:=1. T:=1. R:=1. S:=0. data:=1
---------------------^---------------------

T e x t   f o r   f u n c t i o n   ' T '   o f
'process_userdata\add_controls\generate_controls':
---------------------v---------------------
I:=1. J:=1. K:=1. T:=0. R:=1. S:=1. data:=1
---------------------^---------------------

End of function descriptions.
```

Appendix S: IDaSS description of FCS

This text is taken from a system document generated by IDaSS.

```
Text for function 'connect' of 'process_userdata\FCS\FCS':
----------------------v----------------------
FCS_out :=
((address = 0)
 if1: I7
 if0:
((address = 1)
 if1: I6
 If0:
((address = 2)
 if1: I5
 if0:
((address = 3)
 if1: I4
 if0:
((address = 4)
 if1: I3
 if0:
((address = 5)
 if1: I2
 if0:
((address = 6)
 if1: I1
 if0:
((address = 7)
 if1: I0
 if0: 0
)))))))
----------------------^----------------------

End of function descriptions.
```

Appendix T: IDaSS description of fdb operators

This text is taken from a system document generated by IDaSS.

```
Text for function 'extract' of 'process_userdata\FCS\fdb0':
----------------------v----------------------
_X32 := F at: 0.
_X33 := F at: 1.
_X34 := F at: 2.
_X35 := F at: 3.

_P0 := P at: 0.
_P1 := P at: 1.
_P2 := P at: 2.
_P3 := P at: 3.

_O0 := _P0 >< _X32.
_O1 := _P1 >< _X32 >< _X33.
_O2 := _P2 >< _X32 >< _X33 >< _X34.
_O3 := _P3 >< _X33 >< _X34 >< _X35.

O := _O3, _O2, _O1, _O0
----------------------^----------------------

End of function descriptions.

================================================
Text for function 'extract' of 'process_userdata\FCS\fdb1':
----------------------v----------------------
_X32 := F at: 0.
_X33 := F at: 1.
_X34 := F at: 2.
_X35 := F at: 3.

_P0 := P at: 0.
_P1 := P at: 1.
_P2 := P at: 2.
_P3 := P at: 3.

_O4 := _P0 >< _X32 >< _X34 >< _X35.
_O5 := _P1 >< _X32 >< _X33 >< _X35.
_O6 := _P2 >< _X33 >< _X34.
_O7 := _P3 >< _X32 >< _X34 >< _X35.

O := _O7, _O6, _O5, _O4
----------------------^----------------------

End of function descriptions.

================================================
```

```
Text for function 'extract' of 'process_userdata\FCS\fdb2':
----------------------v----------------------
_X32 := F at: 0.
_X33 := F at: 1.
_X34 := F at: 2.
_X35 := F at: 3.

_P0 := P at: 0.
_P1 := P at: 1.
_P2 := P at: 2.
_P3 := P at: 3.

_O8  := _P0 >< _X32 >< _X33 >< _X35.
_O9  := _P1 >< _X33 >< _X34.
_O10 := _P2 >< _X32 >< _X34 >< _X35.
_O11 := _P3 >< _X32 >< _X33 >< _X35.

O := _O11, _O10, _O9, _O8
----------------------^----------------------

End of function descriptions.

================================================
Text for function 'extract' of 'process_userdata\FCS\fdb3':
----------------------v----------------------
_X32 := F at: 0.
_X33 := F at: 1.
_X34 := F at: 2.
_X35 := F at: 3.

_P0 := P at: 0.
_P1 := P at: 1.
_P2 := P at: 2.
_P3 := P at: 3.

_O12 := _P0 >< _X32 >< _X33 >< _X34.
_O13 := _P1 >< _X33 >< _X34 >< _X35.
_O14 := _P2 >< _X34 >< _X35.
_O15 := _P3 >< _X35.

O := _O15, _O14, _O13, _O12
----------------------^----------------------

End of function descriptions.

================================================
```

```
Text for function 'extract' of 'process_userdata\FCS\fdb4':
---------------------v---------------------
_X32 := F at: 0.
_X33 := F at: 1.
_X34 := F at: 2.
_X35 := F at: 3.

_P0 := P at: 0.
_P1 := P at: 1.
_P2 := P at: 2.
_P3 := P at: 3.

_O16 := _P0 >< _X32.
_O17 := _P1 >< _X33.
_O18 := _P2 >< _X34.
_O19 := _P3 >< _X35.

O := _O19, _O18, _O17, _O16
---------------------^---------------------

End of function descriptions.

=============================================
Text for function 'extract' of 'process_userdata\FCS\fdb5':
---------------------v---------------------
_X32 := F at: 0.
_X33 := F at: 1.

_P0 := P at: 0.
_P1 := P at: 1.
_P2 := P at: 2.
_P3 := P at: 3.

_O20 := _P0.
_O21 := _P1.
_O22 := _P2 >< _X32.
_O23 := _P3 >< _X32 >< _X33.

O := _O23, _O22, _O21, _O20
---------------------^---------------------

End of function descriptions.

=============================================
```

```
Text for function 'extract' of 'process_userdata\FCS\fdb6':
----------------------v----------------------
_X32 := F at: 0.
_X33 := F at: 1.
_X34 := F at: 2.
_X35 := F at: 3.

_P0 := P at: 0.
_P1 := P at: 1.
_P2 := P at: 2.
_P3 := P at: 3.

_O24 := _P0 >< _X33 >< _X34.
_O25 := _P1 >< _X34 >< _X35.
_O26 := _P2 >< _X32 >< _X35.
_O27 := _P3 >< _X33.

O := _O27, _O26, _O25, _O24
----------------------^----------------------

End of function descriptions.

================================================
Text for function 'extract' of 'process_userdata\FCS\fdb7':
----------------------v----------------------
_X34 := F at: 2.
_X35 := F at: 3.

_P0 := P at: 0.
_P1 := P at: 1.
_P2 := P at: 2.
_P3 := P at: 3.

_O28 := _P0 >< _X34.
_O29 := _P1 >< _X35.
_O30 := _P2.
_O31 := _P3.

O := _O31, _O30, _O29, _O28
----------------------^----------------------

End of function descriptions.
```

Literature

1. "Strategies for real-time system specification", D.J. Hatley and I.A. Pirbhai
   1988, Dorset House, New York, USA

2. American National Standard for information systems
   Fiber distributed data interface (FDDI) - token ring physical layer medium dependent (PMD)
   ANSI X3.166-1990

3. American National Standard for information systems
   Fiber distributed data interface (FDDI) - token ring physical layer protocol (PHY)
   ANSI X3.148-1988

4. American National Standard for information systems
   Fiber distributed data interface (FDDI) - token ring media access control (MAC)
   ANSI X3.139-1987

5. American National Standard for information systems
   Fiber distributed data interface (FDDI) - token ring station management (SMT)
   ANSI X3, still under development

6. "The SUPERNET family for FDDI", Technical manual
   Advanced Micro Devices, Inc. 901 Thompson Place P.O. Box 3453 Sunnyvale, California 94088-3453

7. "Structured Design of an FDDI Protocol Handler", P.W.A.J.M. Nas
   Digital Systems Group, Eindhoven University of Technology, the Netherlands, Sep, Oct 1991

8. "IDaSS for ULSI", Manual for IDaSS V0.08d, Mini-IDaSS V1.00, Ir. A.C. Verschueren
   Digital Systems Group, Eindhoven University of Technology, July 1990

9. "Error characteristics of FDDI", R. Jain
   IEEE trans commun, USA, V38, N8, P1244-1252, Aug 90

10. "Communications via FDDI", T. van Gelder (in dutch)
    Elektronica, V38, N9, P20-31, 4 May 90

11. "Working with FDDI: A designer's guide", K. Parker
    Electron des (USA), V38, N8, P95-98,100,102,105, 26 April 1990

12. "Fiber optic links hold the keys to the fast LANs of the future", K. Parker, S. Kaufman
    Electron des (USA), V37, N25, P46-53 (14 Dec 1989)

13. "Engineering building and campus networks for FDDI", TF. McIntosh
    Proceedings of the SPIE - Int soc opt eng (USA), V1179, P240-251 (1990)
    (Fiber networking and telecommunications)

14. "FDDI interoperability between vendors", AR. Khan
    Proceedings of the SPIE - Int soc opt eng (USA), V1179, P266-275 (1990)
    (Fiber networking and telecommunications)

15. "FDDI - an overview", FE. Ross
Standardization: ASC X3T9; IS ISO/IEC JTC1/SC25
Proceedings of the 14th conference on Local Computer Networks, 10-12 Oct 1989 Minneapolis, Minnesota (IEEE cath no 89CH2746-6), P5-8

16. "FDDI ring management", KB. Ocheltree
Proceedings of the 14th conference on Local Computer Networks, 10-12 Oct 1989 Minneapolis, Minnesota (IEEE cath no 89CH2746-6), P18-23

17. "The role of concentrators in FDDI rings", J. Hutchison
Proceedings of the 14th conference on Local Computer Networks, 10-12 Oct 1989 Minneapolis, Minnesota (IEEE cath no 89CH2746-6), P24-40

18. "A high performance FDDI network and its applications", RM. Montalvo and K. Ocheltree. Proceedings of the International Conference on Public Networks (Networks 90), Birmingham, UK , June 1990, P135-147.

19. "FDDI MAC services design considerations", JF. Torgerson
Proceedings of the 14th conference on Local Computer Networks, 10-12 Oct 1989 Minneapolis, Minnesota (IEEE cath no 89CH2746-6), P41-48

20. "FDDI-II: Operation and architectures", M. Teener
Proceedings of the 14th conference on Local Computer Networks, 10-12 Oct 1989 Minneapolis, Minnesota (IEEE cath no 89CH2746-6), P49-61

21. "Standards for robust second generation LAN management", RB. McClure
Proceedings of the 14th conference on Local Computer Networks, 10-12 Oct 1989 Minneapolis, Minnesota (IEEE cath no 89CH2746-6),P133-142

22. "Performance analysis of FDDI", R. Sankar
Proceedings of the 14th conference on Local Computer Networks, 10-12 Oct 1989 Minneapolis, Minnesota (IEEE cath no 89CH2746-6), P328-332

23. "The FDDI link error rate monitor", GW. Kajos, DH. Hunt
Proceedings of the 14th conference on Local Computer Networks, 10-12 Oct 1989 Minneapolis, Minnesota (IEEE cath no 89CH2746-6), P347-357

24. "Modelling and performance evaluation of FDDI ring", M. Cerqueiro, G. Makhoul, Ruey-Lin Ju, and K. Jabbour. Department of Electrical and Computer Engineering Syracuse University, 111 Link Hall. 89CH2785-4

25. "FDDI ICs and components: Reduced costs key FDDI's acceptance", M. Wright
EDN (USA), V34, N19, P81-84,86-91 (14 Sept 89)

26. "Emerging standards, hardware and software light the way to FDDI", K. Marrin
Comput des (USA), V28, N7, P51-54,56-57 (1 April 1989)

27. "FDDI - A new high speed LAN with optical fibers", C. Mittasch (in German)
Nachr tech elektron (East Germany), V39, N3, P96-98, (1989)

28. "FDDI and BWN backbone networks, a performance comparison based on simulation", T. Welzel
The IEEE 8th annual international Phoenix conference on computers communications, proceedings (IEEE cath no 89CH2713-6)

29. "Optical high speed (100 Mbps) token ring system", K. Ohno, et al
Globecom 88, IEEE global telecommunications conference and exhibition - communications for the information age
Conference record (IEEE cath no 88CH2535-3) hollywood (FL) (USA) 28 Nov-1 Dec 88
New York USA IEEE 88, V2, P645-652

30. "FDDI - A sign of things to come in datacommunication", D. van Mierop
Computer communication technologies for the 90s, proceedings of the 9th international conference, Tel Aviv, Israel 30 Oct-3 Nov 88, p155-161

31. "Fiber optic transceiver for FDDI application", Y. Uda, et al
FOC/LAN '88, proceedings 12th international fiber optics communications and Local Area Networks expositions, Atlanta, GA, USA, 12-16 Sept 88, P126-130

32. "The FDDI - A new generation standard for LANs", TJ. King
Eurocon 88: 8th European conference on electronics
[proceedings on:area communications (cath no 88CH2607-0) Stockholm Sweden 13-17 June 88]
New York USA, IEEE 1988, P239-242

33. "FDDI-II: Integrated services on high speed LANs", P. Food
ISDN 88, Proceedings of the international conference, London UK, June 88, P303-312

34. "Extended ethernet connectivity with FDDI technology", D. van Mierop, A. Eldad
Networking technology and architectures, Proceedings of the international conference London UK, June 88, P127-136

35. "Fibre optic components for the FDDI 100 Mbps LAN", T. King
Proceedings of the SPIE, int soc opt eng (USA) vol 949, P2-13 (1988)

36. "Distributing FDDI's power", R. Bruce McClure
Special Report: FDDI, Lightwave February 1991, P32,34,36

37. "Unsnarling FDDI's Kinks", Shyamala Reddy
LAN Magazine, June 1991, P36-39

Index