and *A_Collector*. The manager maps a request for connection establishment onto a free connection processor, which from now on works exclusively for this connection until the connection will be closed. The pair (connection_number, processor_id) will be sent to the units *A_Collector*, *N_Collector* and *Receiver* to assign incoming data to a connection processor and outgoing data or events to a connection number.

The four assignment units *Management*, *A_Collector*, *N_Collector* and *Receiver* are connected with the processors by busses, on the two busses between the *CP*s and the *Collector*s can be contention which is solved by a bus reservation scheme. The connections between the *CP*s and the *MMU*s are used for segmentation and reassembly only, therefore a pointer to the user data and the length of one segment is given to the *MMU,* which returns a list of pointers to the segments or a list of pointers to segments is given to the *MMU,* which returns a pointer to the reassembled user data, respectively. A checksum should be calculated on the fly in the network adapter or the *MMU*s have to be extended to this task.

The main components inside a connection processor are the *FSM* (Finite State Machine) processing units that perform all the functions of a protocol. The FSMs of a formal protocol specification can be mapped onto these components. All *FSM* processing units work independently and communicate with each other and with the other components via signals. Therefore, not only the processing for the connections, but also all the processing within one connection is performed concurrently. The *FSM*s use *Timer*s which have either a global (multiple *FSM*s can start the timer, the time-out event is sent to multiple *FSM*s, which need not to be the same) or a local character (only one *FSM* uses this timer). In addition, there is a global *ALU* to manipulate global registers that store all the variables used by more than one *FSM* (e.g. XTP). The *FSM*s send a command to the *ALU* that performs the required operation on the data and returns a result if necessary (e.g. additions, assignments, comparisons). To guarantee consistency the *FSM*s can access the registers only via the *ALU*. Multiplexers (*MUX*) and demultiplexers (*DMX*) are the interfaces of a processor to its environment. The multiplexers collect all data that come from the *FSM*s and the registers and send it after an acknowledge from the *Collector*s over the connected bus. The demultiplexers receive all data sent to a connection processor and forward it to the receiver(s) (*FSM* or *Register*). The mapping of the data to the receiver is supported by a content addressable memory.

The *FSM* processing unit is the basis for implementation of parallel extended FSMs described in a protocol specification. Each of these units consists of local registers, a local ALU and a control unit. Each of the local ALUs is customised for the requirements of the *FSM*. An ALU can be as simple as an adder, but also as complex as a unit to manage a dynamic list. To decouple the *FSM*s from each other and from other units inside a connection processor, each *FSM* has got an input queue. Every signal to a *FSM* will be inserted into this queue according to the implicit mechanisms of the protocol specification language SDL (asynchronous communication).

One of the most complex local ALUs is a unit to manage dynamic lists, such as a dynamic retransmission queue that stores gaps in transmitted data. The queue is kept in a local memory, several operations on the data can be performed (e.g. insert_gap, delete_gap, get_gap). Using such a list more complex retransmission strategies, such as selective repeat instead of a simple and inefficient go-back-n, can be implemented.

The complete system with all its components and interconnections is described with VHDL to allow simulations and synthesis. Based on the description of the whole architecture in VHDL the system has been simulated with a VHDL simulator for validation purposes. The protocol used for simulation includes connection management, acknowledgement, retransmission and other typical functions of transport oriented layers. To get better estimations on space and time requirements parts of the design were synthesised with a high level synthesis tool. Due to the high integration and the application specific design a larger number of connection processors can be implemented, one possible extension is to implement context switching.

[1] A.S. Krishnakumar, J.G. Kneuer, A.J. Shaw: *HIPOD: An Architecture for High-Speed Protocol Implementations*, 4th IFIP conference on high performance networking, Liège, Belgium, 1992

[2] T.S. Balraj, Y. Yemini: *Putting the Transport Layer on VLSI - the PROMPT protocol chip*, 3rd IFIP Workshop on Protocols for High Speed Networks, Stockholm, May 1992