ISRN SICS-R--92/01--SE

A Host Interface to the DTM Network

Bengt Ahlgren Stephen Pink Per Lindgren
Lars Håkansson Per Gunningberg Lotten Elmstedt
Christer Bohm Mats Björkman

SICS research report
R92:01
ISSN 0283-3638

A Host Interface to the DTM Network*

Bengt Ahlgren[†] Stephen Pink[†] Per Lindgren[‡]
Lars Håkansson[‡] Per Gunningberg[†] Lotten Elmstedt[‡]
Christer Bohm[‡] Mats Björkman[§]

SICS research report R92:01

[†]Swedish Institute of Computer Science,
Box 1263, S-164 28 Kista, Sweden

[‡]Dept. of Telecommunication and Computer Systems,
Royal Institute of Technology, S-164 40 Kista, Sweden

[§]Dept. of Computer Systems, Uppsala University,
Box 520, S-751 20 Uppsala, Sweden

December 5, 1991

Abstract

DTM, dynamic synchronous transfer mode, is a new time division multiplexing technique for fiber networks currently being developed and implemented at the Royal Institute of Technology in Stockholm, Sweden. This paper describes the hardware and software aspects of the design of an SBus host interface to the DTM network for a Sun SPARCstation.

The interface is based on a dual port memory residing on the interface card and accessible over the SBus from the host CPU. The host operating system allocates message buffers directly in this memory. The interface has hardware support for segmenting and reassembling packets to and from the data units of the DTM. The software part of the interface manages the shared memory and the virtual circuits provided by the DTM network.

*Also appearing in the Proceedings of the 3rd MultiG Workshop, KTH, Stockholm, December 17, 1991.

1 Introduction

Dynamic synchronous transfer mode, DTM[1], is a novel access method for high speed fiber networks. It is a TDM technique with similarities to both ATM and STM, but tries to avoid their respective drawbacks. It supports flexible and dynamic bandwidth reallocation, but avoids the overhead associated with the cells and cell headers of ATM.

DTM is being developed and implemented by a group of people at the Department of Telecommunication and Computer Systems at the Royal Institute of Technology, Stockholm, Sweden [5]. The DTM group together with a group at the Swedish Institute of Computer Science, SICS, is designing a host interface for the network. The joint group has decided to design the interface for the Sun SPARCstation and its SBus. This choice was made because of the relative simplicity of the SBus, and because our present equipment mainly consists of Suns.

This work is a part of the MultiG research program [7, 6] in Sweden. MultiG is a collaborative program for research in the area of multimedia applications and high speed networks. The program contains many projects ranging from optical fiber interfaces to distributed virtual world applications and applications using digital audio and video.

2 The DTM medium access method

The DTM is a medium access method offering virtual circuits with guaranteed bandwidth. It is basically a TDM scheme but with the ability to dynamically allocate and reallocate bandwidth between different hosts and connections. Circuit switching makes it possible to do all time consuming work at connection setup. The data transfer phase is simple and implemented in hardware.

The rest of this section gives an overview of the DTM. An article by Pehrson *et al.* [5] describes DTM in more detail.

2.1 Concepts

The optical fiber network is organized as a dual fiber bus. Each node on the network is attached to both of the unidirectional fibers, which in the prototype have data rates of 622.08 Mbit/s.

The TDM scheme (see Figure 1) has *cycles* of 125 μs length. Each cycle is filled with 64-bit *slots*. With the bit rate of the prototype, there is room for about 1200 slots in each cycle.

Bandwidth allocation is done on a slot position basis. If, for example, slots number 7 and 8 are allocated, these slots in all following cycles belong to the allocation (until changed). The slots of an allocation form a virtual

[1] DTM was previously called PTM—programmable synchronous transfer mode.

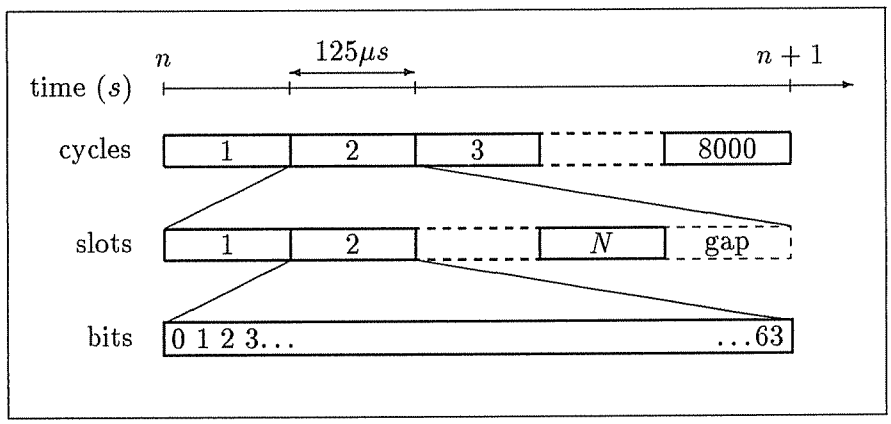


Figure 1: DTM TDM scheme.

circuit, or connection, with 64-bit data units. The number of slots per cycle in the allocation determines the bandwidth of the connection. The least bandwidth that can be allocated is thus a single slot per cycle or 512,000 bits/s.

A small amount of the slots in each cycle are called *static* slots. They are preallocated, one for each node on the fiber bus, and are used for control signalling such as establishing and releasing connections. The rest of the slots are called *dynamic* slots and are used for transfer of data.

2.2 Hardware overview

Figure 2 shows the hardware architecture of the DTM prototype currently being developed. The two fibers are attached to the *fiber access units*. The units first convert from optical to electrical signals and then from serial to 64 bit parallel signals. They are the only parts that need to operate at the full fiber frequency.

The 64 bit parallel stream is fed into the *DTM state machine*. The state machine has a table for selecting slots that are destined for the node. The selected slots are written over the *internal bus* to a unit indicated by the table. The units attached to the internal bus are the host computer interface and the DTM node controller. The design allows more units on the internal bus. For instance, more fiber access units can be attached to facilitate switching between different fiber buses.

The *DTM node controller* is a processor that handles all protocol control signalling for setting up and releasing connections. It also programs the tables of the state machines in order to make the data flow to and from the fiber in the desired manner for the different connections. After a connection has been set up, the node controller is not involved in transferring the data stream. Instead, the state machine directly transfers the data between the host interface and the fiber over the internal bus.

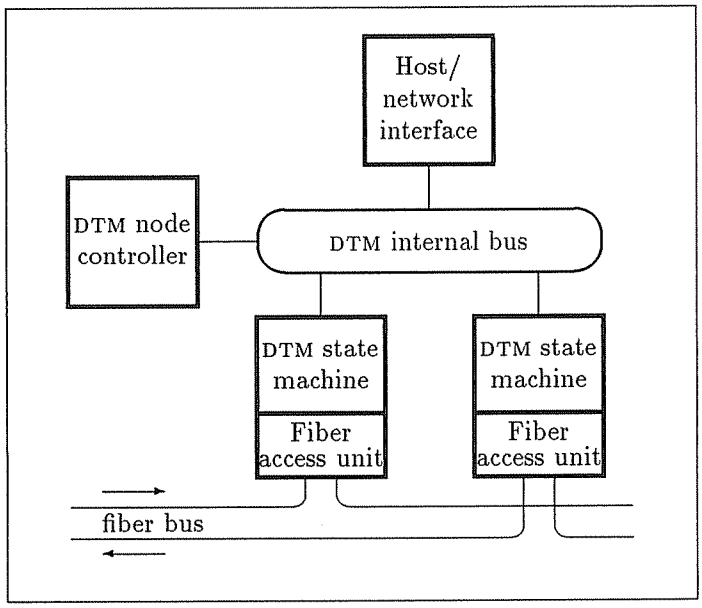


Figure 2: DTM prototype hardware architecture.

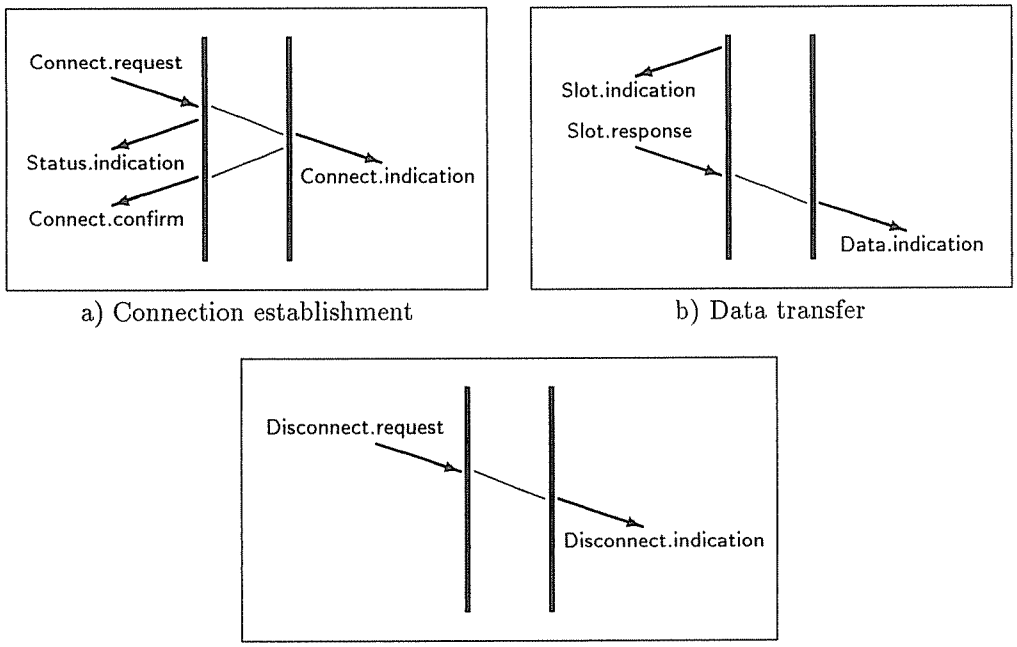
2.3 Service

The DTM service is connection oriented with a simplex or full duplex synchronous data channel. The service provides guaranteed bandwidth with separate and possibly different allocation for each direction. There is also support for changing the bandwidth allocation on an open connection. The service has three distinct phases: connection establishment, data transfer and connection release.

The connection establishment phase (see Figure 3a) has the following primitives:

- DTM-Connect.request: Issued by the service user to request the establishment of a new connection.
- DTM-Connect.indication: Issued by the service provider to indicate a new incoming connection.
- DTM-Status.indication: Issued by the service provider to a service user that previously requested a connection to indicate that connection establishment is in progress.
- DTM-Connect.confirm: Issued by the service provider to indicate that the connection is established.

The DTM-Status.indication is an indication which the service provider returns immediately to a user that requested a connection. The indication says that the requested bandwidth is available from the local side of the connection.



a) Connection establishment

b) Data transfer

c) Connection release

Figure 3: DTM service.

The service user has the option at this stage to begin the transfer of data without waiting for DTM-Connect.confirm. The connection can, however, still be rejected by the remote side.

The data transfer phase (see Figure 3b) has the following primitives:

- DTM-Slot.indication: Issued by the service provider to indicate that the service user should provide data for the next slot.
- DTM-Slot.response: Issued by the service user in response to DTM-Slot.indication supplying data for the next slot.
- DTM-Data.indication: Issued by the service provider when a slot is received.

The connection release phase (see Figure 3c) has the following primitives:

- DTM-Disconnect.request: Issued by the service user to release the connection.
- DTM-Disconnect.indication: Issued by the service provider to indicate that the connection has been released.

The service also has primitives for changing the allocated bandwidth. See the service specification for more details [3].

3 SBus characteristics

The SBus [8] is a chip-level interconnect bus between components such as processors and memory in systems based on microprocessors. It has a maximum clock frequency of 25 MHz, and a 32 or 64 bit data path. Data transfers are supported in sizes of 1, 2, 4, 8, 16, 32 and 64 bytes. The 8 byte and larger transfers are *burst* transfers which use several clock cycles in succession for transferring the data. The peak transfer rate in a burst is 100 MB/s for the 32 bit data path and the double for the 64 bit data path. The maximum sustained transfer rate is 80 MB/s for the 32 bit data path.

The SPARCstation 1 runs its SBus at 20 MHz clock frequency and its largest burst transfer size is only 16 bytes. The maximum sustained transfer rate is about 29 MB/s when the CPU is the bus master, and about 25 MB/s for other masters (which require a virtual address translation cycle before the data transfer).

The SPARCstation 2 also runs its SBus at 20 MHz, but can do full 64 byte burst transfers. According to some memory bandwidth benchmarks posted to the Usenet newsgroup `comp.benchmarks` by John D. McCalpin at University of Delaware, USA, it can do sustained transfers at about 60 MB/s.

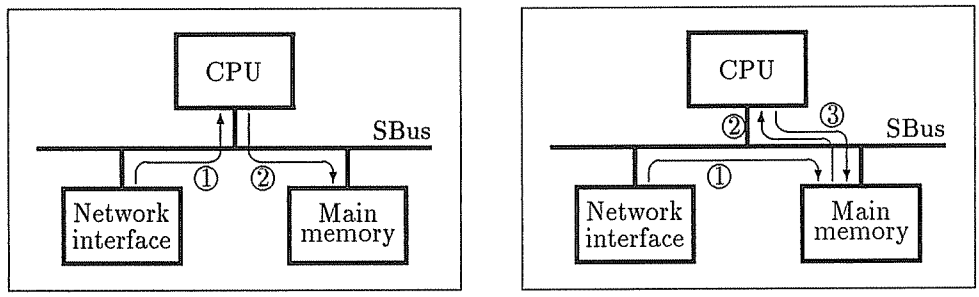
4 Host interface design

The DTM host/network interface is a so called "slave only" SBus device. It looks like ordinary memory to the host system. Slave only means that the device does not itself initiate the transfer of data between the device and other devices connected to the SBus. Another device acting as a bus master, e.g., the CPU, is needed to transfer data. The host interface does, however, have the capability to interrupt the host CPU. We call this design a "non-DMA" design because DMA is not used to transfer data from the interface over the SBus to the main memory.

We choose the non-DMA design since we believe that it has better performance than a design with DMA would have. In the UNIX networking architecture it is hard, if not impossible, to avoid one copy of data from the kernel buffers to the user process address space. If we allocate the kernel buffers in the memory on the interface card, this copy can be made directly (by the CPU) from the interface to the user address space. This makes the data traverse the SBus twice (see Figure 4a).

With this design, the maximum throughput will be limited to half the memory bandwidth of the system. The memory bandwidth of the SPARCstation 2 is about 60 MB/s, which means that the limit on this machine is 30 MB/s, or 240 Mbit/s.

In the alternative design with DMA over the SBus the data would traverse the SBus one additional time in the DMA operation giving a total of three



a) Host interface using dual port memory b) Host interface using DMA

Figure 4: Host interface design.

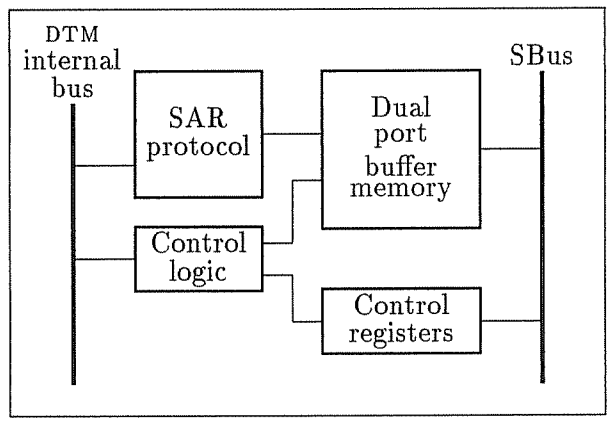


Figure 5: Host interface hardware architecture.

traversals (see Figure 4b). With this design, the maximum throughput would be limited to one third of the system memory bandwidth.

Another reason for not choosing DMA is that the DMA controller available for the SBus, the L64853A DMA+ controller from LSI LOGIC, has a throughput of only 8 MB/s, which is less than half of the transfer rate of the SPARCstation 1 CPU.

4.1 Hardware architecture

Figure 5 shows the main components of the DTM host interface hardware. It is connected to the DTM internal bus (see Figure 2), which allows writing and reading of data to and from the fiber and the DTM node controller. The interface implements two simple protocols in hardware which are described in the following two sections. The protocols are needed for converting the 64-bit synchronous slot stream of the raw DTM to larger asynchronous data units suitable for the host to handle. The parts of the device visible to the host are control registers and a 4 MB dual port memory for I/O buffers.

For more information about the interface hardware design, see the DTM SBus hardware interface specification [2].

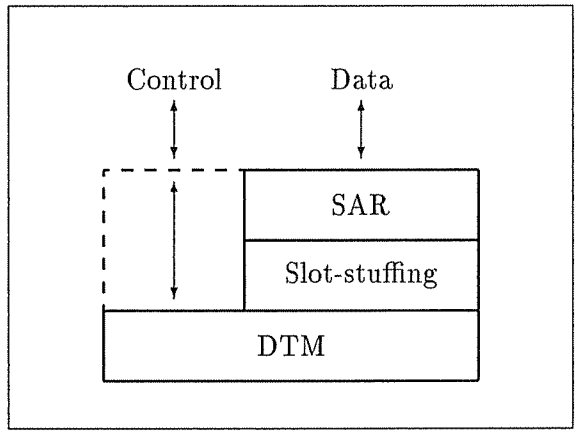


Figure 6: Host interface protocol architecture.

4.2 Segmentation and reassembly

The DTM service provides a synchronous data channel with 64-bit data units. These units are too small for the host computer to process efficiently. The purpose of our segmentation and reassembly, or SAR, protocol is to convert the synchronous 64-bit service to larger asynchronous data units. The SAR service data unit is a multiple of 64 bits with an unlimited maximum size. Neither the DTM nor the SAR, in their present form in the prototype implementation, apply any form of error detection to the data stream. If error detection is desired, it must be taken care of by higher protocol layers.

The SAR protocol is applied to the data transfer part of the DTM service described above in Section 2.3. The DTM service is first enhanced with the slot-stuffing protocol described in the next section, but this does not change the service primitives other than to add the two control flags *end-of-frame* and *idle*. In the protocol architecture the other DTM primitives are remapped, without any processing, to SAR primitives with identical functionality. The SAR service has two primitives for data transfer:

- SAR-Data.request: Issued by the SAR user to request the transfer of a data unit.
- SAR-Data.indication: Issued by the provider to indicate that a data unit has arrived.

Figure 6 illustrates the protocol architecture of the host interface and the relationship between the SAR protocol and the other protocols.

The SAR protocol is straightforward. On the sending side, whenever the synchronous indication that a slot is to be sent is sensed from below, a response is issued with 64 bits of data. If there is no data to send, that is, no SAR-Data.request has been received from the service user, the idle control flag is set in the response. If the slot was the last slot of a data unit, the end-of-frame control flag is set.

On the receiving side, the protocol discards slots with the idle control flag set and puts the data from other slots in a buffer. When a slot with the end-of-frame flag set is received, a SAR-Data.indication primitive is issued.

The details of the SAR protocol can be found in its specification [4].

4.3 Slot-stuffing

To be able to implement the segmentation and reassembly protocol, two control flags are needed for every DTM data slot: one for indicating *end-of-frame*, and one for indicating an empty or *idle* slot. These flags are provided by a simple protocol which we call the *slot-stuffing* protocol. Note that this protocol is only applied to the data transfer part of the DTM service (see also Figure 6).

The end-of-frame and idle control flags are implemented by *control slots* in the data stream. The slot-stuffing protocol uses a technique similar to the bit-stuffing in HDLC to allow user data to contain the same bit patterns as the control slots.

When the user has set one or both of the control flags, a control slot containing the flags is sent before the slot containing the data segment. Should a user data segment have the same bit pattern as a control slot, an *escape* control slot is inserted before the user data segment slot, telling the receiver that the next slot should be interpreted strictly as user data.

The details of the slot-stuffing protocol can be found in its specification [1].

4.4 Software interfacing

The host computer system needs software to be able to use the DTM host interface. This software is incorporated in the host operating system in the form of a device driver. We will develop the driver for Sun's operating system SunOS, and later also adapt it to the coming Berkeley Unix 4.4 BSD.

The device driver for the DTM interface is different from most network drivers in one respect: it handles connections. The connections are identified by using different device units, even though there is really only one device. A device unit is normally used to identify multiple identical devices attached to the same bus or controller and that is serviced by the same device driver.

The SunOS operating system kernel has a standard set of entry points to device drivers, which for the DTM are listed in Table 1. The management of DTM connections is done through the `dtmioctl()` routine.

The `dtmattach()` routine is responsible for setting up the kernel virtual memory maps so that the interface memory can be accessed. The interface buffer areas in the dual port memory are mapped cacheable, but the control registers are mapped non-cacheable. It is important that the accesses from the CPU to the buffer areas are cached, because the cache controller generates

`dtmidentify()`	identifies the device to the system during auto configuration.
`dtmattach()`	makes the device available and known to the system.
`dtminit()`	makes the device ready for operation.
`dtmreset()`	resets the device to a known state.
`dtmintr()`	device interrupt service routine called when new data arrives and when some special event happens.
`dtmioctl()`	performs I/O-controls on the device, that is, device specific control operations.
`dtmoutput()`	initiates output on the device.

Table 1: The DTM device driver entry points.

burst transfers on the SBus when it is filling the cache. This results in considerably higher throughput. Since the buffer memory has dual ports, the driver must be careful to invalidate the cache at the right times. The control registers, on the other hand, should not be cached, because burst transfers cause several adjacent registers to be written together at the same time.

The driver allocates message buffers, called *mbufs*, in the dual port memory on the interface card. When data arrives from the network, it is written by the hardware directly in the already allocated mbuf. The interface interrupts the host system in two cases: when the mbuf is full and when an end-of-frame indication was received. The kernel then calls the `dtmintr()` routine in the driver which provides a new mbuf to the interface and queues the full mbuf for the higher level protocols. All higher layer protocol processing can then be done by the kernel without moving the data. The data is then eventually copied to the address space of a user process in the main memory of the system.

A socket interface will be designed for the raw DTM service for use by application programs. This interface is useful both for debugging and for experimenting with applications needing guaranteed bandwidth, but not complete reliability. A new address family will be introduced for this socket interface, and possibly also a new socket type.

References

[1] Mats Björkman. DTM slot-stuffing specification. Technical report, Swedish Institute of Computer Science (SICS), Box 1263, S-164 28 Kista,

Sweden, November 1991. First draft.

[2] Christer Bohm, Bengt Ahlgren, and Per Lindgren. DTM SBus interface specification. Technical report, Dept. of Telecommunication and Computer Systems, Royal Institute of Technology, S-164 40 Stockholm, Sweden, November 1991. First draft.

[3] Lotten Elmstedt, Per Gunningberg, and Lars Håkansson. DTM service specification. Technical report, Dept. of Telecommunication and Computer Systems, Royal Institute of Technology, S-164 40 Kista, Sweden, November 1991. First draft.

[4] Per Gunningberg, Mats Björkman, and Lotten Elmstedt. DTM SAR service and protocol specification. Technical report, Swedish Institute of Computer Science (SICS), Box 1263, S-164 28 Kista, Sweden, November 1991. First draft.

[5] B. Pehrson, F. Reichert, P. Lindgren, C. Bohm, L. Håkansson, L. Elmstedt, P. von Knorring, and L. Gauffin. The design of the MultiG dual fiber network. In *Proceedings of the 2nd MultiG Workshop*, pages 93–107, Electrum, Stockholm-Kista, Sweden, June 17, 1991.

[6] Björn Pehrson, Per Gunningberg, and Stephen Pink. MultiG—A research program on distributed multimedia applications and gigabit networks. *IEEE Networks Magazine*, 6, January 1992.

[7] Björn Pehrson and Stephen Pink. Multimedia and high speed networking in MultiG. *Computer Networks and ISDN Systems*, 21(4):315–319, June 1991.

[8] Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, CA 94043, USA. *SBus Specification B.0*, Revision A of December 1990. Part number 800-5922-10.