APRIL 1991

WRL Research Report 91/4

TurboChannel T1 Adapter

David Boggs

digital **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

The Western Research Laboratory (WRL) is a computer systems research group that was founded by Digital Equipment Corporation in 1982. Our focus is computer science research relevant to the design and application of high performance scientific computers. We test our ideas by designing, building, and using real systems. The systems we build are research prototypes; they are not intended to become products.

There is a second research laboratory located in Palo Alto, the Systems Research Center (SRC). Other Digital research groups are located in Paris (PRL) and in Cambridge, Massachusetts (CRL).

Our research is directed towards mainstream high-performance computer systems. Our prototypes are intended to foreshadow the future computing environments used by many Digital customers. The long-term goal of WRL is to aid and accelerate the development of high-performance uni- and multi-processors. The research projects within WRL will address various aspects of high-performance computing.

We believe that significant advances in computer systems do not come from any single technological advance. Technologies, both hardware and software, do not all advance at the same pace. System design is the art of composing systems which use each level of technology in an appropriate balance. A major advance in overall system performance will require reexamination of all aspects of the system.

We do work in the design, fabrication and packaging of hardware; language processing and scaling issues in system software design; and the exploration of new applications areas that are opening up with the advent of higher performance systems. Researchers at WRL cooperate closely and move freely among the various levels of system design. This allows us to explore a wide range of tradeoffs to meet system goals.

We publish the results of our work in a variety of journals, conferences, research reports, and technical notes. This document is a research report. Research reports are normally accounts of completed research and may include material from earlier technical notes. We use technical notes for rapid distribution of technical material; usually this represents research in progress.

Research reports and technical notes may be ordered from us. You may mail your order to:

Technical Report Distribution
DEC Western Research Laboratory, WRL-2
250 University Avenue
Palo Alto, California 94301 USA

Reports and notes may also be ordered by electronic mail. Use one of the following addresses:

Digital E-net:	`DECWRL::WRL-TECHREPORTS`
Internet:	`WRL-Techreports@decwrl.dec.com`
UUCP:	`decwrl!wrl-techreports`

To obtain more details on ordering by electronic mail, send a message to one of these addresses with the word ‘‘`help`’’ in the Subject line; you will receive detailed instructions.

TurboChannel T1 Adapter

David Boggs

Abstract

This report is the technical reference for the T1D4PKT adapter, a communication option module for TurboChannel I/O systems. It uses basic HDLC (flags, bit-stuffing, CRC) to send and receive packets over a T1 circuit with D4 framing (1.544 Mb/s digital telephone circuit). The adapter was designed for use in packet switching routers linking local area networks into metropolitan-area and wide-area networks.

This adapter is a research prototype; it is not a product.

Copyright © 1991 Digital Equipment Corporation

digital **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

1. Overview

The T1D4PKT adapter is a communication option module for TurboChannel I/O systems. It uses basic HDLC (flags, bit-stuffing, CRC) to send and receive packets over a T1 circuit with D4 framing (1.544 Mb/s digital telephone circuit). The adapter was designed for use in packet switching routers linking local area networks into metropolitan-area and wide-area networks.

The adapter's electrical interface is DSX-1 and its physical interface is an RJ48 modular telephone jack. The *Data Service Unit* (DSU) function is included in the adapter, so it can directly connect to a *Channel Service Unit* (CSU), a multiplexer, etc.

Figure 1 shows some possible configurations for connecting two packet routers with a T1 circuit. The top connection shows a T1 circuit using wire pairs provided by a telephone company; the two central offices could be in the same city or on opposite coasts. The middle connection shows a T1 circuit multiplexed with three other T1 circuits, perhaps containing voice channels, and carried over a microwave radio T2 link; maximum distance is about 15 kilometers between radios. The bottom connection shows a T1 circuit multiplexed with twenty-seven other T1 circuits and carried over a private fiber optic T3 link. Digital Equipment Corporation owns and operates a fiber optic network connecting most company sites in New England.

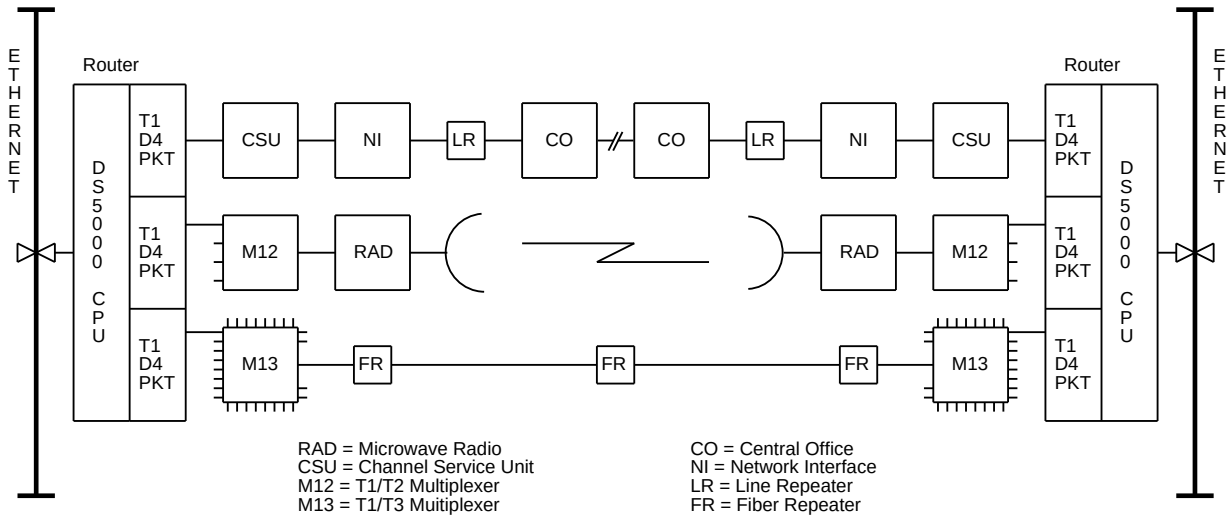

Figure 1: Some Possible Configurations

2. Installation Information

Figure 2 is a full-scale drawing of the adapter. It is a single-slot TurboChannel option module. It uses programmed I/O, not DMA, and occupies 4 MBytes of address space. It is designed to operate with a 25 Mhz bus clock; a PAL change is required for systems operating slower than 16 Mhz. Power consumption is 1.5 Amps at 5 Volts, or 7.5 watts; no 12 Volt power is used. There are no jumpers or switches to configure; just plug it in.

Two LEDs are visible through a hole next to the modular connector. The *Carrier* LED lights when the adapter is seeing a legal T1 signal. The *FrameSync* LED lights when the adapter has synchronized with the D4 framing pattern imbedded in the T1 bit stream. The LEDs correspond to bits by the same name in the control/status register. Both LEDS must be lit for the adapter to work.

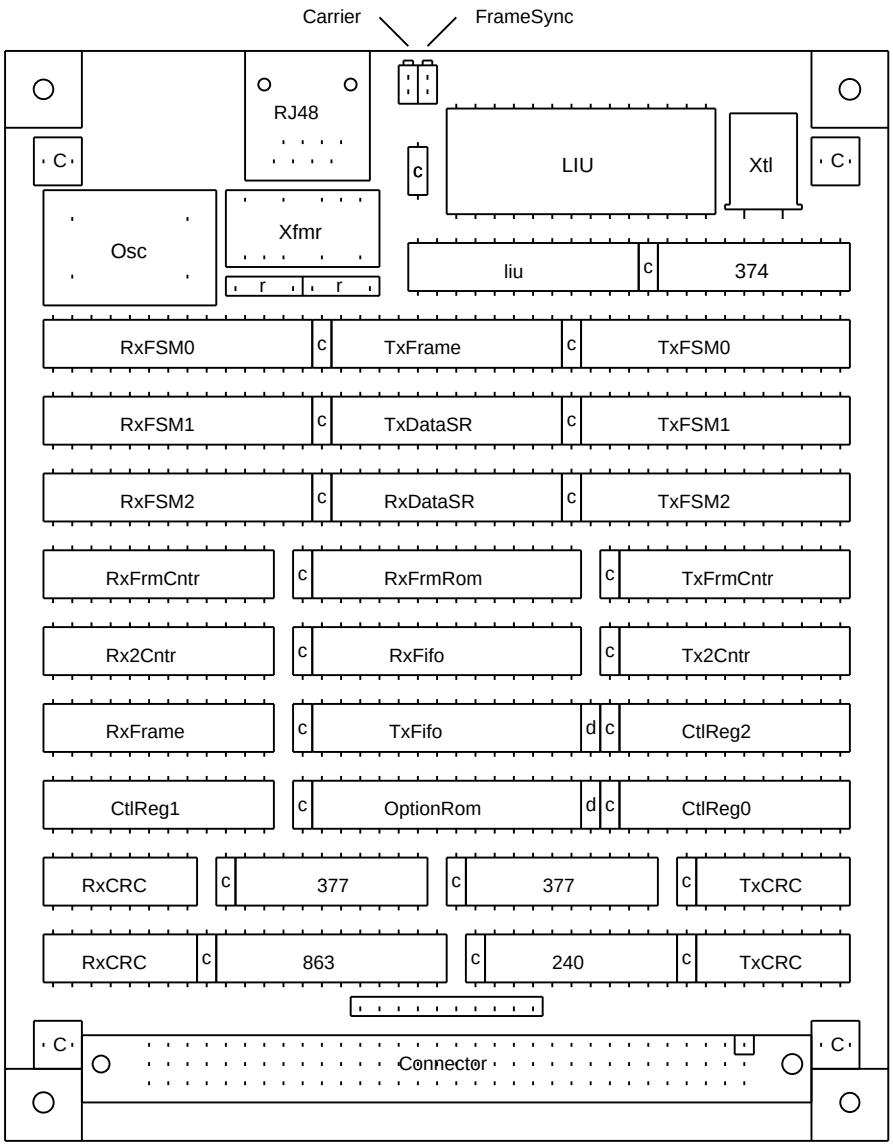

Figure 2: Option Module Layout

The gross bit rate of a T1 circuit is 1.544 Mb/s, but one out of every 193 bits is overhead, so the net bit rate is 1.536 Mb/s. HDLC overhead consumes a few more Kb/s, but the adapter can operate with a sustained user data rate in excess of 1.5 Mb/s (the exact rate depends on the data pattern; worst case is 1.28 Mb/s for all ones data). A T1 circuit is *full duplex*: a packet can be moving in each direction simultaneously, for an agregate link bit rate of 3 Mb/s (in contrast, an Ethernet is *half duplex*: only one packet can be moving on the Ether at any time, but the bit rate is 10 Mb/s). Three T1D4PKT adapters can be plugged into a DS5000/200 system.

The application driving the design of this adapter was a router that forwards packets between Ethernet local networks. The maximum size of an Ethernet packet is 1518 bytes; The maximum packet size of this adapter is 2048 bytes. The limiting factor is the size of FIFO ICs, currently 4K bytes. Since the capacity of a FIFO chip can grow without changing its pinout, maximum packet size can be increased as bigger FIFOs become available simply by loading different ICs.

Two types of connectors are used on T1 equipment: an 8-pin modular telephone connector designated RJ48 by the FCC, and a 15-pin D-series subminiature connector. This adapter uses the new RJ48 connector, but it may connect to equipment using the older DA15 connector. Figure 3 shows the corresponding pins in the two connectors. If the other equipment has a modular jack, then the cable is wired straight through. A *null modem cable* connecting two nearby adapters with wire and no electronics, *turns over* the transmit and receive pairs between the two connectors. Looking into the cable entry hole on a modular plug with the locking tab oriented up, pin one is on the right; DA15 connectors have pin numbers molded into the plastic shell.

Signal	RJ48	DA15	Direction
Xmt Tip	5	1	Adapter to Net
Xmt Ring	4	9	Adapter to Net
ground	7	2	shield ground
Rcv Tip 1	2	3	Net to Adapter
Rcv Ring 1	1	11	Net to Adapter
ground	8	10	shield ground

Figure 3: Connector Pins

T1 is designed to use twisted wire pairs; a *Channel Service Unit*, or CSU, can drive one mile of 19-gauge wire before a *line repeater* is required; up to fifty line repeaters can be used, although three is typical. At the end of a T1 circuit, a telephone company installs a *Network Interface*, or NI, which can be looped back on command from a telco test board. The electrical interface of this adapter is called *Digital Signal Cross-connect level 1* or DSX-1, a standard signal level used between T1 equipment in a telephone central office where distances are limited to a few hundred feet [2] [10] [16]. Runs of less than 1 meter can use flat modular cable; length is limited because flat modular cable is not shielded and not twisted. Runs from 1 to 10 meters can use unshielded twisted pairs such as that used for twisted pair Ethernet. Runs longer than about 10 meters should use shielded twisted pairs such as that used for token rings. Maximum cable length is about 100 meters. Both ends are transformer coupled and the signalling is insensitive to polarity, so wires within a pair can be swapped.

3. What is T1?

T1 was developed at Bell Laboratories in the late 1950s, and the first commercial system was installed in 1962 [20] [21] [22] [23]. Urban cable conduits and manholes were filling up and it was becoming necessary to carry more than one *voice channel* on a wire pair. Silicon transistors were available, making possible low-power high-reliablility line repeaters. Digital technology was maturing; all previous multiplexing schemes used analog technology. 24 analog voice channels were converted to 8-bit digital samples 8000 times per second, for a combined data rate of 1.536 Mb/s. Between each *frame* of 24 samples (192 bits), an overhead bit was inserted for synchronizing terminal equipment at the ends of a circuit, making the total bit rate 1.544 Mb/s. Bell System multiplexing schemes traditionally were assigned letter designations, and this first digital system was named *T-carrier*; the electrical format was named *DS1*, for Digital Signalling level 1. A 64 Kb/s voice or data channel is level 0 of the hierarchy; *multiplexers* combine several bit streams from a lower level in the hierarchy to form a higher level bit stream. Figure 4 summarizes the hierarchy of digital transmission systems that has evolved over the last three decades [2] [10] [15].

Level	Bit Rate	Components
DS0	0.064 Mb/s	Voice Channel
DS1	1.544 Mb/s	24 DS0s
DS2	6.312 Mb/s	4 DS1s
DS3	44.736 Mb/s	7 DS2s
DS4	274.176 Mb/s	6 DS3s

Figure 4: Digital Hierarchy

A T1 signal is a sequence of pulses 3 volts high and 324 nanoseconds wide. The presense of a pulse in a bit cell represents one data value and the absense of a pulse represents the other binary value. Pulses alternate in polarity, making it a 3-level code: +3v, 0 and -3v. This *Alternate Mark Inversion*, or AMI code, has no DC signal component and the highest fundamental frequency component is half the bit rate. A bit error, changing a pulse into a no-pulse or vise-versa, results in two pulses of the same polarity, called a *bipolar violation*. Figure 5 shows a T1 signal with a bipolar violation near the end of the bit string.

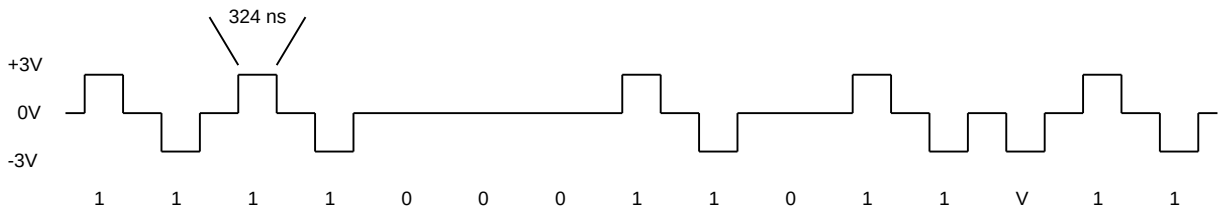

Figure 5: T1 Waveform

Timing information, defining when a receiver should sample the value of a bit, is recovered from the pulse stream, so a T1 signal must contain a minimum *pulse density*. The average pulse density must be at least 12.5%, or one pulse in eight bit cells, and there must be no more than 15 consecutive bit cells in which no pulse is present. Channel Service Units are required to insert pulses into a T1 stream if the pulse density is too low. As will be explained below, HDLC bit stuffing guarantees that the bit stream produced by this adapter meets the pulse density requirement.

When equipment can't guarantee meeting the pulse density requirements, a scheme called *Bipolar with 8 Zero Substitution* or B8ZS is used. B8ZS replaces eight consecutive bit cells that contain no-pulses with a sequence of pulses containing a distinctive bipolar violation. Normally, intermediate equipment in a circuit removes bipolar violations, but when a circuit is ordered with the B8ZS feature, these particular bipolar violations are not eliminated and are not counted as errors. This adapter never generates long strings of no-pulses so it does not use B8ZS.

The clock used by a T1 transmitter may come from a local oscillator (*local* or *master timimg*), or from the received pulse stream (*loop* or *slave timing*) [1] [8]. Local oscillators can be as much as 75 bits per second fast or slow, so multiplexers must be prepared to handle unsynchronized pulse streams. Currently, most point-to-point circuits offered by common carriers go through such *asynchronous multiplexers* and the end-user equipment must provide timing [5] [9] [11]. As analog transmission equipment is retired from service and digital VLSI continues to spread, carriers are installing a new generation of *synchronous multiplexers* which require all transmitters to be frequency-locked to a single timing source provided by the network and recovered from the received pulse stream [12] [14].

Timing *jitter*, or short-term variations of the *significant instants* (when to sample the values of bits) from their ideal positions in time, is a big problem for T1 (and token rings and FDDI for similar reasons). Line repeaters introduce *pattern-dependent jitter* [25] and multiplexers introduce *waiting time jitter* [24]. Long-term variations, called *wander*, are caused by such things as daily temperature variation in pole-mounted cable which can change the electrical length of a circuit by many bits. This adapter includes a wander and jitter attenuator which meets all requirements for operation while frequency-locked to the received pulse stream [9] [18].

The *framing bit pattern* is the sequence of overhead bits inserted between groups of 24 8-bit samples. While twisted pairs and line repeaters do not distinguish frame bits from data bits, multiplexers usually require that T1 signals contain a framing pattern. *D4*, the most common framing format and the one implemented by this adapter, consists of the 12-bit repeating sequence 100011011100 [6] [17]. A new framing format called *Extended Super Frame* or ESF, uses some of the frame bits as a checksum over preceding data bits, and other frame bits as a low-speed data link for monitoring and controlling equipment in the circuit. There are two competing standards for how to use the facility data link in ESF: AT&T [7] and BellCore [13] do it one way, and ANSI [3] does it a different way; manufacturers offer both versions.

A continuous stream of pulses without an imbedded framing pattern is generated by transmission equipment when it is taken out of service for maintenance (such as a loopback test) and when a failure is detected that prevents delivery of data. This signal, called *blue alarm* or *all ones*, is part of a larger failure reporting scheme that has evolved over the years. When a piece of transmission equipment loses its upstream signal, it sends blue alarm downstream (since it has no real data to send) and it sends *yellow alarm* upstream.

Europe adopted the good ideas from the American digital telephone system, and fixed many of its faults. Consequently European T1, called *E1*, is similar but not compatible. The bit rate is 2.048 Mb/s, or 32 8-bit samples 8000 times per second. Coax cable is often used rather than twisted wire pairs. *High Density Bipolar order 3*, or HDB3 coding, replaces four consecutive no-pulses with a 4-bit sequence containing a bipolar violation, similar to B8ZS. Frame bits are not sprinkled in between groups of bytes as in T1, but rather two *time slots*, bytes 0 and 16, are reserved for synchronization patterns, checksums, and facility data link, similar to ESF. This format is called PCM30, because it carries 30 *Pulse Code Modulation* voice channels.

There is much more to T1 than has been explained here, but most of it only applies to circuits carrying voice channels. The T1 signal generated by this adapter resembles 24 PCM voice channels just enough to easily pass through telephone networks.

4. What is HDLC?

Synchronous Data Link Control, or SDLC [27] [28], was developed by IBM in the late 1960s as an improvement over BiSync [26], an older method for sending packets over a serial link. During the 1970s various standards bodies adopted modified versions: the American National Standards Institute called their version *Advanced Data Communication Control Procedure* or ADCCP [4]; The International Standards Organization called their version *High-Level Data Link Control* or HDLC [19].

The good idea in SDLC is *bit stuffing* (as opposed to *byte stuffing* used in BiSync) to distinguish control information from user data. Between packets, an idle transmitter sends continuous *Flags*: (01111110). If the eight bits after a flag are not also a flag, then they are the first eight bits of a packet. Within a packet, a transmitter *stuffs*, or inserts, a 0-bit after five consecutive 1-bits; a receiver *unstuffs*, or deletes, a 0-bit after five consecutive 1-bits. Bit stuffing prevents data patterns from looking like flags. Six or more consecutive 1-bits terminates a packet. If there are exactly six consecutive 1-bits (another flag) then the packet ends normally. If there are seven or more 1-bits (an *Abort*) then the packet is damaged and should be discarded.

The bytes just before a packet's ending flag are a *Cyclic Redundancy Checksum* or CRC. The CRC is calculated by the transmitter, appended to the end of the data before the flag, and checked by the receiver. If the CRC is not correct, the packet is damaged and should be discarded. The CRC is the remainder after dividing the packet bits by a polynomial. When the receiver does the same division on the incoming packet bits including the CRC, the result should be zero.

Bit stuffing can be used to satisfy the pulse density requirements of a T1 circuit by encoding a 0-bit as a pulse and a 1-bit as a no-pulse. This is called *Inverted HDLC* because voice channels traditionally encode 1-bits as pulses (*all ones* is more correctly called *all pulses*). The worst case happens when a no-pulse frame bit falls in an HDLC flag, resulting in seven consecutive no-pulses, which is 12.5% pulse density.

There is much more to HDLC than has been explained here, but most of it is not required for sending unreliable datagrams. If desired, HDLC's address and control fields can be implemented by host software.

5. What is a TurboChannel?

A TurboChannel is a synchronous asymmetric input/output bus designed by Digital Equipment Corporation for use in workstations [29] [30] [31]. It is *synchronous* because there is a global clock whose rising edge is the reference for all bus signals. It is *asymmetric* because there is one *system module* and some number of *option modules* connected to it. The system module can read and write the option modules, and option modules can read and write the system module, but option modules can't talk to each other.

When a program running in the system module executes a load or store instruction whose effective address falls within the address space of an option module, an *I/O read* or *I/O write* bus transaction is performed. The system module asserts a signal selecting an option module, and places an address on the 32-bit address/data bus, which the option module saves in a register. In the next cycle the system module reads or writes a word; if the option module is not ready, it can assert a signal to stall the transaction.

For DS5000 systems, I/O reads and writes to the adapter should use addresses in the uncached and unmapped KSEG1 memory region (the base address of I/O slot 0 is 0xBE000000 and that of slot 1 is 0xBE400000). A DS5000/200 can do 5 I/O writes or 2 I/O reads per microsecond. The T1 transmitter requires an I/O write and the receiver requires an I/O read about every 5 microseconds when going full-tilt boogey.

An option module can read and write main memory via the system module by performing a *Direct Memory Access* or DMA bus transaction. An option module asserts a read or write signal to start a DMA operation, and then waits for an acknowledgement from the system module. Upon receipt of an ack, the option module drives an address onto the bus and then writes, or after a pause, reads 32-bit words on each succeeding cycle, for a peak bandwidth of 100 MBytes/sec (one 32-bit word every 40 ns = 800 Mbits/sec = 100 MBytes/sec). At 192 KBytes/sec each way, this adapter does not use DMA.

The TurboChannel clock can be any fixed frequency between 12.5 and 25 MHz. For slow TurboChannels, the IOCTL2 PAL at position E14 must be changed. The fast PAL works from 27 MHz to 15.6 MHz (DS5000/200); the slow PAL works from 16.6 MHz to 7.8 MHz (DS5000/1xx). This will be fixed somehow in the next artwork revision.

There is nothing more to the TurboChannel; it is refreshingly simple. Many of DEC's I/O buses are asynchronous and symmetric, making them slow and complex, and raising the cost to implement simple devices.

6. Programming Interface

The T1D4PKT adapter is a programed I/O device; it does not use DMA. A device driver communicates with the adapter by executing load and store instructions whose effective addresses are within the adapter's 4 MByte address space.

There are three regions in the adapter address space: control/status, data, and option rom.

- The ***Control/Status Register*** (CSR) is a single address. Some of the bits in this register are read-only, such as FIFO status, and some are read/write, such as interrupt enable and loopback.

- The ***Data Register*** is a single address to which transmitter bytes are written and from which receiver bytes are read. Even though the transmitter and receiver are completely independent of each other, their data registers share the same I/O address.
- The ***Option ROM*** occupies 128K addresses in the adapter's space. It contains information about the adapter (type, manufacturer, revision level, etc) and diagnostic programs that can be loaded into memory and executed under the ROM Executive.

6.1. Control/Status Register

The control/status register, figure 6, is addressed when the adapter is selected and address bits 17 and 4 are one. Other address bits are ignored, so there are many other I/O addresses which will also reference the CSR.

```
  15   14   13   12   11   10    9    8    7    6    5    4    3    2    1    0
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| RX | RX | RX | xx | TX | TX | TX | TX | RX | RX | TX |    | TX | TX | RX |    |
|Fifo|Fifo|Fifo| xx |Fifo|Fifo|Fifo|Strt|Carr|Frm |Undr|Loop|Lcl |Int |Int |Int |
|Full|Half|Empt| xx |Full|Half|Empt|    |ier |Sync|Flow|Back|Clk |Req |Req |Enbl|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+

  Bits 31..16: no connection
  Bit 15 - Rx Fifo Full; read-only; cleared by hardware reset.
   One if the Receiver FIFO is full (4096 bytes).
  Bit 14 - Rx Fifo Half; read-only; cleared by hardware reset.
   One if the Receiver FIFO is more than half full (2049-4096 bytes).
  Bit 13 - Rx Fifo Empty; read-only; set by hardware reset.
   One if the Receiver FIFO is empty.
  Bit 12 - no connection.
  Bit 11 - Tx Fifo Full; read-only; cleared by hardware reset.
   One if the Transmitter FIFO is full (4096 bytes).
  Bit 10 - Tx Fifo Half; read-only; cleared by hardware reset.
   One if the Transmitter FIFO is more than half full (2049-4096 bytes).
  Bit  9 - Tx Fifo Empty; read-only; set by hardware reset.
   One if the Transmitter FIFO is empty.
  Bit  8 - Tx Start; read-write; cleared by hardware reset.
   Set by a write to the Data register or CSR with bit 8 on.
   Cleared by the Transmitter when it starts sending a packet.
  Bit  7 - Rx Carrier; read-only; unaffected by hardware reset.
   One if the receiver sees a legal T1 signal.
   If this bit is off then no packets will be received.
  Bit  6 - Rx Frame Sync; read-only; unaffected by hardware reset.
   One if the receiver is locked onto the D4 framing pattern.
   If this bit is off then no packets will be received.
  Bit  5 - Tx UnderFlow; read-write; cleared by hardware reset.
   Set by hardware if the transmitter needs a byte but the FIFO is empty.
   Cleared by software writing the CSR with a one bit in this position.
  Bit  4 - Loopback; read-write; cleared by hardware reset.
   One if the transmitter output should be looped into the receiver.
  Bit  3 - Tx Local Clock; read-write; set by hardware reset.
   One if the transmitter should use the on-board oscillator.
   Zero if the transmitter should use the receiver clock.
  Bit  2 - Tx Interrupt Request; read-write; cleared by hardware reset.
   Set by hardware after the last bit of a packet has been transmitted.
   Cleared by software writing the CSR with a one bit in this position.
  Bit  1 - Rx Interrupt Request; read-write; cleared by hardware reset.
   Set by hardware after writing receiver status into the FIFO.
   Cleared by software writing the CSR with a one bit in this position.
  Bit  0 - Interrupt Enable; read-write; cleared by hardware reset.
   Set by software to permit the adapter to cause interrupts.
```

Figure 6: Control/Status Register (CSR)

`csr.TxIntReq`, `csr.RxIntReq`, and `csr.TxUnderflow` are set by hardware and cleared by software. Since any of these bits can be set at any time, clearing them must be done with care. To clear one of these bits, write the CSR with a one in the bit position to be cleared; be careful not to change any other bits such as `csr.IntEnbl`, `csr.LoopBack`, and `csr.TxLclClk`.

The adapter can be operated without interrupts by clearing `csr.IntEnbl` and polling `csr.RxIntReq` and `csr.TxIntReq`. In addition to the Interrupt Enable bit in the adapter CSR, a bit in a CPU register must be set for interrupts to occur. `csr.TxIntReq` and `csr.RxIntReq` must both be cleared before returning from an interrupt or else another interrupt will immediately happen.

`csr.TxUnderflow` is the only transmitter status bit. Transmitter FIFO underflow means the hardware and software are seriously out of sync. The packet being transmitted is terminated by an HDLC abort with no CRC, insuring that the receiver will notice that it is damaged.

Nearly all of the adapter can be tested by setting `csr.LoopBack` and verifying that every packet transmitted is also received. The receiver ignores data from the T1 circuit when looped back, so any packets sent by the far end will be lost. If packets were transmitted on the T1 circuit as well as being looped into the near-end receiver, the far-end receiver might think that the loopback test packets were real data that it should act on. Therefore, the transmitter sends unframed all-ones onto the T1 line when in loopback. All-ones keeps the equipment in the circuit happy but because it doesn't contain the framing pattern, the receiver at the far end will consider the circuit down (`csr.FrameSync` will be false).

The proper value for `csr.TxLclClk` depends on properties of the particular T1 circuit to which the adapter is connected. Some circuits require that the transmitter use the clock recovered by the receiver from the incoming data (`TxLclClk=0`). Other circuits require the transmitter to supply its own locally generated clock (`TxLclClk=1`). When the circuit provides the clock, the adapters at both ends must set `TxLclClk=0`; when the circuit does not provide the clock, at least one of the adapters must set `TxLclClk=1` (preferably both). Fine point: the transmitter uses the on-board oscillator regardless of the state of `csr.TxLclClk` if `csr.LoopBack` is true or `csr.Carrier` is false.

6.2. Receiver Data Register

The receiver data register, figure 7, is the read-port of a 4K by 9-bit FIFO. On the T1 line side of the FIFO, incoming data bits are shifted into an 8-bit register and then written in parallel into the write-port of the FIFO (see figure 11). One byte of *receiver status*, figure 8, is written into the FIFO after the last data byte of a packet. This receiver status byte is distinguished from the data bytes by the ninth bit of the FIFO, which appears as `RxData.PktEnd`. If `PktEnd` is zero, then bits 7..0 are a data byte, otherwise they are the receiver status applying to the previously read data bytes. The last four bytes before the receiver status are the packet's CRC, and they should be discarded by software.

Software must protect itself from very long packets. There is no inherent limit on packet length imposed by the hardware, and even if a transmitter never sends very long packets, they can occur at the receiver during initialization, or because of slow service by the device driver, or because of data errors on the T1 line.

```
  15   14   13   12   11   10    9    8    7                                  0
+----+----+----+----+----+----+----+----+-------------------------------------+
| RX | RX | RX | xx | TX | TX | TX |End |                                     |
|Fifo|Fifo|Fifo| xx |Fifo|Fifo|Fifo| of |            Data or CRC Byte          |
|Full|Half|Empt| xx |Full|Half|Empt|Pkt |                                     |
+----+----+----+----+----+----+----+----+-------------------------------------+

  Bits 31..16: no connection
  Bits 15..9: Rx and Tx FIFO status
   See the description in Control/Status Register.
  Bit 8 - End of Packet
   One if bits 7..0 is Ending Status
   Zero if bits 7..0 is a Data byte.
  Bits 7..0: Data or CRC Byte
   A data byte; the last four bytes before End Of Pkt are CRC bytes.
```

Figure 7: Receiver Data Register (RxData)

During operating system initialization, the device driver must synchronize with the adapter. The receiver runs all the time, and attempts to write packet bytes into the FIFO even if software isn't servicing the adapter. When a device driver first starts up, it is likely to find that the receiver FIFO has overflowed and is full of stale packet fragments. These bogus fragments must be flushed until the FIFO is empty and a good receiver status has been read.

Efficient transfer of data from the receiver to main memory depends on minimizing I/O read references to the adapter, which take 10 cycles. Ideally the ratio of I/O reads to data bytes moved is slightly greater than one but much less than two. The FIFO status bits in the receiver data register reflect the state of the FIFO including the effect of the current read operation. If `RxData.FifoEmpty`, is true then bits 7..0 are the last data byte available at the instant the read was happening, and the FIFO status bits in the CSR should be checked before reading the data register again. Do not read the receiver data register when `csr.RxFifoEmpty` is true.

```
  15   14   13   12   11   10    9    8    7    6    5    4    3    2    1    0
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| RX | RX | RX | xx | TX | TX | TX |End |Loss|Loss|Loss|    |    |    |Fifo|    |
|Fifo|Fifo|Fifo| xx |Fifo|Fifo|Fifo| of | of | of | of |  0 |  0 |Abrt|Over|Bad |
|Full|Half|Empt| xx |Full|Half|Empt|Pkt |Sgnl|Fram|Data|    |    |    |Flow|CRC |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+

  Bits 31..16: no connection
  Bits 15..9: Rx and Tx FIFO status
   See the description in Control/Status Register.
  Bit 8 - End of Packet
   One if bits 7..0 is Ending Status.
   Zero if bits 7..0 is a Data byte.
  Bit 7 - Loss of Signal
   Zero if the receiver sees a legal T1 signal.
  Bit 6 - Loss of Frame
   Zero if the receiver is locked to D4 framing pattern.
  Bit 5 - Loss of Data
   Zero if the packet is a multiple of eight bits; one if bits lost/added.
  Bit 4 - zero
  Bit 3 - zero
  Bit 2 - Abort
   One if the packet ended in an HDLC abort.
  Bit 1 - FIFO Overflow
   One if the receiver wanted to write a byte but the FIFO was full.
  Bit 0 - Bad CRC
   One if the Cyclic Redundancy Checksum is not correct.
```

Figure 8: Receiver Status (RxStatus)

The correct receiver status for a packet is bits 7..0 all zeros:

- **`LossOfSignal`** is one if the T1 signal did not contain sufficient pulses to keep the clock recovery circuit functioning. If no pulse is received in 32 consecutive bit cells then `LossOfSignal` goes to one, terminating any packet in progress.
- **`LossOfFrame`** is one if the T1 signal did not contain the D4 framing pattern. The frame detector will tolerate up to two incorrect bits out of 12 consecutive frame bits. If the receiver loses the framing pattern then `LossOfFrame` goes to one, terminating any packet in progress.
- **`LossOfData`** is one if the number of bits between HDLC flags, after deleting stuffed bits, was not evenly divisible by eight. If the remainder is non-zero then bits have been added or dropped from the data stream.
- **`Abort`** is one if a packet ended with an HDLC abort. Packets normally begin and end with HDLC flags, (01111110). If the T1 circuit dies (no more pulses), then the receiver will see all ones, which is an HDLC abort.
- **`FifoOverflow`** is one if the receiver wanted to write the FIFO, but it was full. The data is lost, but the occurance of the error is remembered as a receiver status bit. `FifoOverflow` is cleared after successfully writing a receiver status (including the `FifoOverflow` error bit) into the FIFO.
- **`BadCRC`** is one if the Cyclic Redundancy Checksum on the packet was incorrect. The last four bytes of a packet are a 32-bit CRC appended by the transmitter. If a bit is inverted, added or dropped between transmitter and receiver then the CRC will be bad at the receiver with very high probability.

6.3. Transmitter Data Register

The transmitter data register, figure 9, is the write-port of a 4K by 9-bit FIFO. On the T1 line side of the FIFO, data bytes are read from the read-port of the FIFO and loaded into a shift register (see figure 13). The last byte of a transmitter packet is indicated by setting `TxData.EndOfPkt`, which becomes the ninth bit in the FIFO. When this byte comes out the other end of the FIFO, the transmitter appends the checksum and generates the ending HDLC flag.

Note that a receiver packet ends with a status byte with the ninth bit set, but a transmitter packet ends with a data byte with the ninth bit set. The transmitter hardware generates and appends the CRC without software help, and the receiver hardware checks but does not delete the CRC, which the software must do.

If `TxStart` is one, the transmitter stops sending flags and starts reading and sending bytes from the FIFO. `TxStart` is asserted by setting `TxData.EndOfPkt` or `csr.TxStart`. The transmitter hardware clears `TxStart` within 8 bit times, or about 5 microseconds. If `TxStart` is one but the FIFO is empty, then the transmitter simply clears `TxStart` and continues to send flags.

To transmit a packet, software writes a sequence of bytes into the FIFO through the transmitter data register. Writing the last byte of the packet (with bit 8 one) sets `TxStart` which starts the transmitter. When the transmitter reads the last packet byte from the FIFO, (the byte

```
  31                              9    8   7                                    0
+----------------------------------+----+--------------------------------------+
|                                  |End |                                      |
|           unused                 | of |              Data  Byte              |
|                                  |Pkt |                                      |
+----------------------------------+----+--------------------------------------+

  Bits 31..9: no connection
  Bit 8 - End of Packet
   One if bits 7..0 are the last byte of a transmitter packet.
   Zero if bits 7..0 are a data byte except the last.
  Bits 7..0: Data Byte
   A data byte.
```

Figure 9: Transmitter Data Register (TxData)

with bit 8 set) it appends 4 bytes of CRC, sets `csr.TxIntReq` and finally goes idle sending HDLC flags.

It is not possible to send back-to-back packets using the simple strategy of writing one packet into the FIFO and then waiting for the transmitter to interrupt. The transmitter will be idle for the time it takes to service the interrupt and load the next packet into the FIFO. To send back-to-back packets, software must write the next packet into the FIFO while the transmitter reads the current packet out of the FIFO.

Using the FIFO to double-buffer packets means that a pair of packets must fit into the 4 Kbyte FIFO. If no packet is longer than 2 Kbytes then this strategy works. The software writes the first packet into the FIFO, then waits for `csr.TxStart` to go false, then writes the next packet into the FIFO, and so on. Software may only write one extra packet into the FIFO, even if more packets would fit, because `TxStart` is a single bit and not a counter.

Efficient transfer of data from main memory to the transmitter depends on minimizing I/O references to the adapter. Ideally the ratio of I/O writes to data bytes moved is slightly greater than one but much less than two. Do not write the transmitter data register when `csr.TxFifoFull` is true. If the double-buffered strategy is used then the FIFO status bits don't have to be checked in the data transfer loop.

`csr.TxUnderflow` is set if the transmitter attempts to read a byte from the FIFO but the FIFO is empty. The transmitter sends an HDLC abort without sending a CRC and then goes idle sending flags. This should never happen because the transmitter is never started until an entire packet is in the FIFO, but sometimes cosmic rays do funny things.

If the transmitter gets "stuck" in idle with bytes in the FIFO, software can unstick it by setting `csr.txStart`. This wakes up the transmitter without writing any more bytes into the FIFO and causes it to read from the FIFO. This must be repeated until the FIFO runs dry. In any case, some packet fragments will be transmitted.

6.4. Option ROM

The 32 KByte Option ROM occupies the lowest 128 KBytes of adapter address space. The Option ROM is addressed when the adapter is selected and address bit 17 is zero. Address bits 21..18 are ignored so there are many other I/O address ranges which also reference the ROM. Address bits 16..2 select a location in the ROM. Address bits 1..0 are ignored because I/O

references are to 32-bit words. Consecutive ROM bytes are the least significant bytes of consecutive 32-bit words.

The Option ROM has two parts: fixed-format configuration information and variable-format objects such as scripts, and executable code. System module firmware called the *ROM Executive*, or REX, can interpret the script objects and execute the code objects. Rather than reprogramming a ROM for each new version during development, REX can be directed to read an Option ROM image from an Ethernet boot server rather than from the ROM on the Option module. A Unix program written in Modula-2 generates the PROM programmer file and the net-bootable file.

REX loads a code object at a fixed address in memory and transfers control to it, passing a *callback vector*, an array of utility procedures that the code object can use to print messages on the console, etc. There is only one code object in this adapter's Option ROM, a keyboard-driven diagnostic program written in Modula-2. The program is loaded in the cached kernel memory region because it runs about four times faster than uncached.

To use an Option ROM image booted from the net rather than the ROM on the adapter, register the file with a boot server, and use the undocumented REX command `boot -D n 6/mop`, where `n` is the slot number of the adapter whose ROM is to be replaced (`6/tftp` works too). REX commands invoking code objects will read this image rather than the adapter ROM until REX is reinitialized.

7. Diagnostic Program

The diagnostic program is started by typing `t n` to the REX prompt of `>>`, where `n` is the TurboChannel slot number of the adapter to be tested; i.e. `t 0` starts the diagnostic program of an adapter in slot zero.

7.1. Program Variables

There are five *program variables* which can be set by command and influence the behavior of other commands:

- **`Length`** is a cardinal in the range 2..2048. If the `Random` program variable (see below) is true, then transmitted packets have random lengths between two and `Length` bytes, otherwise transmitted packets have a constant length of `Length` bytes.
- **`Random`** is a boolean variable. If true, then transmitted packets have random lengths and are filled with pseudo random numbers, otherwise packets have constant lengths equal to `txLength` and are filled with the constant pattern (0 1...255).
- **`DataCheck`** is a boolean variable. If it is true, then received packets will be checked for the proper data pattern and discrepancies printed. If the first four bytes of a packet are (0 1 2 3), then the data pattern in the packet is assumed to be the constant pattern (0 1...255), otherwise the first four bytes are assumed to be the initial value of a pseudo random number generator which is used to check the remaining data bytes.

- **`LoopBack`** is a boolean variable. This value is loaded into `csr.LoopBack` whenever the CSR is written. If true, the transmitter output is looped into the receiver input, all ones is transmitted to the T1 line, and packets arriving from the T1 line are ignored.
- **`Clock`** is a boolean variable. This value is loaded into `csr.txLclClk` whenever the CSR is written. If true, the transmitter uses a local crystal oscillator, otherwise it uses the clock recovered from the received signal.

7.2. Diagnostic Commands

The diagnostic is a keyboard-driven program which is used to debug adapters and test T1 circuits without the complication of going through an operating system. Typing "?" lists all of the commands; typing "help" prints a list of the commands along with one-line descriptions. To invoke a command, type an unambiguous initial substring terminated by a carriage return or space. To stop a running command, type any character.

- The ***Quit*** and ***Exit*** commands leave the adapter diagnostic program and return to the ROM Executive.
- The ***Test*** command transmits and receives packets continuously. Test is terminated by typing any character. For every 10 good packets received, a "!" is printed. Bad transmit and receive status values are printed and counted. At the end of the test, the number of good and bad transmit and receive packets is printed. This command is simply TxTest and RxTest (see below) called in a loop.
- The ***TxTest*** command transmits packets continuously. TxTest is teminated by typing any character. For every 10 good packets transmitted, a "!" is printed. If `csr.txUnderFlow` is true, the CSR is printed and the packet is counted as bad. At the end of the test, the number of good and bad packets is printed. Packet length and contents are controlled by the `Length` and `Random` program variables.
- The ***RxTest*** command receives packets continuously. RxTest is terminated by typing any character. For every 10 good packets received, a "!" is printed. If any error bits are set, the receiver status is printed and the packet is counted as bad. At the end of the test, the number of good and bad packets is printed. Packet contents are checked if the `DataCheck` program variable is true.
- The ***Echo*** command transmits any packets received. Echo is terminated by typing any character. For every 10 good packets echoed, a "!" is printed. Bad transmit and receive status values are printed. At the end of the test, the number of packets echoed is printed.
- The ***Watch*** command prints all packets received. Watch is terminated by typing any character. Each packet begins on a new line preceded by "data=". Packet data bytes are printed as two character hex values separated by spaces. Each packet ends by printing the receiver status on a new line.
- The ***Reset*** command attempts to put the adapter in an idle state. It clears `csr.rxIntReq`, `csr.txIntReq`, and `csr.txUnderflow`. It sets `csr.LoopBack` and `csr.txLclClk` to the current value of the program variables of the same name. It flushes the transmitter and receiver FIFOs. Remember that the receiver runs continuously and will put any packet that arrives into the

RxFIFO, so if `csr.LoopBack` is false and packets are arriving, the status after a reset may still show `csr.rxIntReq` true and `csr.rxFifoEmpty` false.

- The ***Status*** command prints the contents of the Control/Status Register (CSR) as a hex number and as named fields.
- The ***Length*** command sets the `Length` program variable. Acceptable values are between 2 and 2048.
- The ***Random*** command toggles the `Random` program variable. Transmitted packets are random length and filled with random data if `Random` is true.
- The ***DataCheck*** command toggles the `DataCheck` program variable. Data in received packets are checked if `DataCheck` is true.
- The ***Loop*** command toggles the `LoopBack` program variable. Transmitter output is looped to receiver input if `LoopBack` is true. `FrameSync` and `Carrier` LEDs should light when `LoopBack` is set.
- The ***Clock*** command toggles the `Clock` program variable. The transmitter uses the local crystal oscillator if `Clock` is true.
- The ***Speed*** command transmits back-to-back packets and measures bits per second and packets per second. Speed is terminated by typing any character. The length of the transmitted packets is controlled by the `Length` program variable. The data in the transmitted packets is the constant pattern (0 1...255). This data pattern will cause HDLC bit-stuffing to occur, so the bit/sec reported by the command will be somewhat less than the maximum rate of 1.536 Mb/s. Five overhead bytes are added to each packet when calculating the bit rate (one flag and four CRC bytes). The clock used for this test has a resolution of 1 second, so the accuracy increases with increasing test time. The speed test is a tight loop running on the bare hardware; your mileage may vary when running the adaptor through an operating system.
- The ***AbortTest*** command tests the HDLC abort generation and detection logic in the transmitter and receiver. AbortTest is terminated by typing any character. At the end of the test the number of good and bad results is printed. The transmitter is forced to generate an HDLC abort by deliberately causing a transmitter FIFO underflow: two bytes are written into the FIFO and then the transmitter is started by setting `csr.txStart` instead of setting the ninth bit of the last data byte. If `RxStatus.abort` is true, then everything is working.
- The ***ScopeLoop*** command continuously reads the control and status register. ScopeLoop is terminated by typing any character. This command generates steady oscilloscope pictures for debugging the TurboChannel control logic.

7.3. Using the Commands

By looping a T1 circuit at various points and using the diagnostic commands, a number of questions about the link and adapters can be answered.

- ***Does-the-Adapter-Work Test.*** A quick and thorough way to test the adapter is to set `LoopBack` and run the *Test* command. This tests the entire adapter except for the modular jack, transformers, and driver/receiver circuits. If none of the program

variables have been changed from their initial values, *Test* will transmit packets of randomly varying length filled with random numbers and check the data in the returning packets. At the beginning of the test, a few bad statuses may be reported as the software synchronizes with the hardware. At the end of the test, the number of good packets received may not exactly match the number of good packets sent.

- ***Does-the-T1-Circuit-Work Test.*** Rather than looping the adapter internally, there are usually several ways to loop the T1 circuit itself. Once the circuit has been looped externally, *Test* can be run without setting `csr.LoopBack`. Setting the loopback switch on the CSU will test all of the adapter, the cable from the adapter to the CSU, and most of the CSU. CSUs also usually have a switch which will send a special signal down the T1 circuit and loop the far-end CSU. Doing this will test all of the adapter, near-end CSU, T1 circuit, and most of the far-end CSU.
- ***Two-Way Test.*** To test both adapters and the entire T1 circuit, run the *Echo* command on one adapter and run *Test* on the other adapter. In this case `csr.LoopBack` should be false at both adapters.
- ***One-Way Test.*** If *Test* fails, it isn't always obvious whether the receiver or the transmitter is having problems. These can be tested separately by using a known-good adapter. Connect the two adapters with a null-modem cable, run *RxTest* on the good adapter and run *TxTest* on bad adapter. If the test fails, the problem is a bad transmitter. Similarly, run *TxTest* on the good adapter and *RxTest* on the bad adapter. If the test fails, the problem is a bad receiver. This technique can also be used to isolate a failure in one direction of a T1 circuit.
- ***Is-Anybody-Out-There Test.*** Often when installing a new T1 circuit, it is useful to know if *any* packet is being received. Also, when debugging new operating system software, it is sometimes useful to get an indendent opinion on whether packets are being transmitted and if they are well-formed. The *Watch* command will print out anything that is not an HDLC flag, including line noise, test patterns generated by the telephone company, and malformed packets.

8. Technical Description

The *Line Interface Unit*, or LIU, contains analog black magic: delay-line clock recovery, a jitter attenuator, and an output pulse shaper. The LIU recovers the clock from the incoming pulse stream with a digital phase locked loop containing a 13-tap *variable* delay line. Jitter in the recovered clock is attenuated by means of a 32-bit FIFO and a *variable* crystal oscillator. Output pulses are formed by a slew-rate controlled fast digital-to-analog converter generating a piecewise linear approximation of a wire-length dependent pulse shape.

The *Transmitter* and *Receiver Finite State Machines* (TxFSM and RxFSM), use 8Kx8 registered PROMs. The RxFSM has eight states and 21 outputs, and the TxFSM has seven states and 23 outputs; each machine uses three PROMs. The FSMs clock about eight times per T1 bit cell; a T1 bit is 648 ns wide and the FSMs clock every 80 ns. This gives plenty of cycles between each bit during which to shift shift registers, count counters, test for underflow, etc. 1.5 Mb/s is loafing: things can be done by brute force in TTL without pushing any speed limits.

The programmed I/O interface to the device driver requires only seven ICs. The 4KB FIFOs greatly ease the interrupt latency requirements on the operating system and make it possible to

handle several packets during one interrupt context switch when the adapter is busy. They also make it easy to handle back-to-back packets (separated by only one HDLC flag) without heoric efforts by the device driver. Careful thought was given to minimizing the number of instruction cycles per byte transferred. By returning the receiver FIFO status when a data byte is read, an extra I/O read of the CSR is avoided inside the copy loop; similarly, an extra I/O read of the CSR is not necessary in the transmitter copy loop.

8.1. Data Path

Figure 10 is a block diagram of the data paths in the adapter. Starting at the T1 line, pulses are coupled through a transformer into the LIU. The LIU derives a clock from the incoming pulses and supplies data bits to the RxFSM. The RxFSM deletes the T1 frame bits and the HDLC stuff bits and accumulates the data bits in a *Shift Register*, SR. When the SR is full, the RxFSM writes the data byte into the FIFO; it can also parallel load the SR with receiver status and write that into the FIFO. Data bytes flow out of the FIFO, through the I/O Interface and onto the TurboChannel when a program executes a load instruction whose effective address falls within the adapter's 4 MByte space.

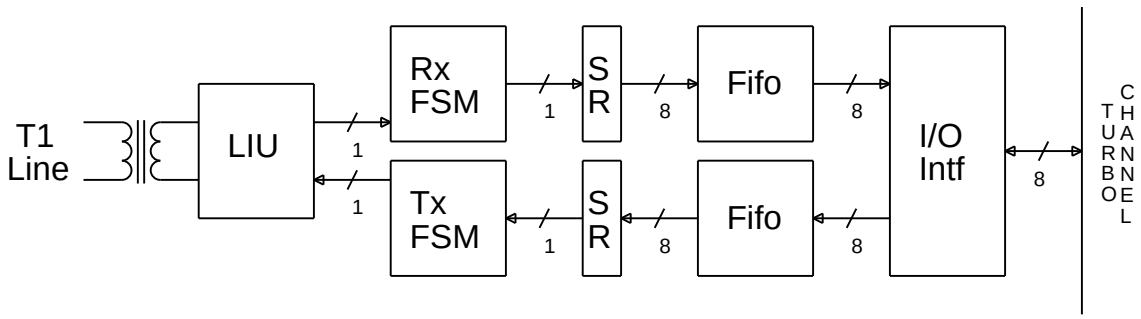

Figure 10: Data Paths in the Adapter

The data path for the transmitter is symmetric with the receiver. Data bytes flow over the TurboChannel, through the I/O interface and into the FIFO when a program executes a store instruction whose effective address falls within the adapater's address space. The TxFSM reads bytes from the FIFO loads them into the SR, inserts T1 frame bits, HDLC stuff bits and CRC bits, and feeds the resulting bit stream to the LIU. The LIU generates the positive and negative pulses and drives the DSX-1 interface through a transformer.

8.2. Receiver

Figure 11 is a block diagram of the receiver; arrows show the path of the data bits. The LIU supplies three signals to the RxFSM through synchronizers: clock, data and loss-of-signal. The RxFSM shifts data bits into *8SR*, counting the bits using *8Cnt*. When 8Cnt underflows, the byte in 8SR is written into the FIFO; status is also loaded into 8SR and written into the FIFO. As bits shift *out* of 8SR, they shift *in* to the CRC; the ending HDLC flag pushes the last data byte into the CRC. HDLC stuff bits are detected by counting consecutive no-pulses using *5Cnt*; HDLC flags are recognized by a 3-state FSM attached to it.

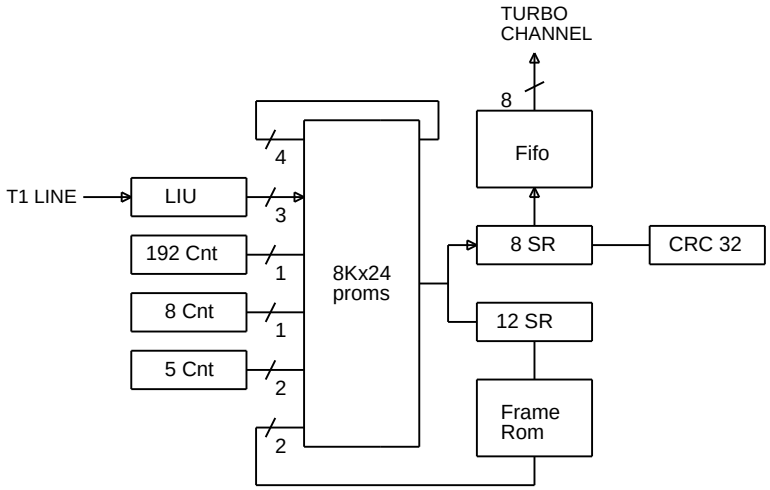

Figure 11: Block Diagram of the Receiver

T1 frame synchronization uses *12SR*, *192Cnt*, and *FrameRom*; the object is to make 192Cnt underflow when a frame bit is expected. 8Cnt and 5Cnt are also used since data bytes are not accumulated and stuff bits are not deleted while trying to locate the framing pattern. The first step is to zero 12SR and preset 192Cnt, 8Cnt and 5Cnt; the next bit to emerge from the LIU is tentatively assumed to be a frame bit and it is shifted into 12SR. The FrameRom looks at 12SR and renders a judgement as to whether it contains the framing pattern (100011011100) in one of its 12 rotations:

- No. 12SR does not contain anything remotely like a framing pattern.
- Maybe. 12SR contains a framing pattern up to the last 1-bit (recall that 12SR is initialized to 0s and less than 12 frame bits may be in the register).
- Error. 12SR contains a framing pattern with one or two bits in error.
- Yes. 12SR contains a framing pattern exactly.

If after shifting a potential frame bit into 12SR, FrameRom says No or Error then move over one bit in the stream and assume that *it* is a frame bit by going back to step one. As long as FrameRom says Maybe or Yes, continue shifting every 193rd bit into 12SR, and counting 8Cnt and 5Cnt; when they both underflow, 40 good frame bits have been seen so `csr.FrameSync` is set. In the worst case, frame synchronization takes (192*193*40) bits, or about one second, typically it takes about 100 ms. D4 framing requires tolerance of up to three bits out of 12 in error without losing frame sync, so once `csr.FrameSync` is set, only No from the FrameRom initiates reframing; but while frame sync is being attempted, no errors are tolerated. Figure 12 shows the tradeoff between Error and No judgements from the FrameRom as the number of bad frame bits is varied. Requiring 40 good frame bits to aquire sync and losing sync if more than two bits are bad is much more stringent than D4 requires, but voice is quite tolerant of errors and reframing disconnects calls.

Bad	Yes	No	Maybe	Error
4	12	443	50	3591
3	12	1682	50	2352
2	12	3190	50	844
1	12	3902	50	132
0	12	4034	50	0

Figure 12: Loss-of-frame lock threshold

8.3. Transmitter

Figure 13 is a block diagram of the transmitter; arrows show the path of the data bits. Note how similar it is to the receiver; the major difference is that less machinery is needed to transmit the frame pattern than to receive it. The *Mux* generates the framing pattern from a modulo-12 counter, as well as steering data, CRC and stuff bits to the LIU. Mux feeds back the value of the outgoing bit to TxFSM so that it can count bits for HDLC stuffing. The signal from LIU to TxFSM is the transmitter clock, informing it that a bit has been accepted and the next bit should be set up. The signal from the FIFO to the RxFSM is fifoEmpty; the signal from 8SR is EndOfPkt (ninth fifo bit).

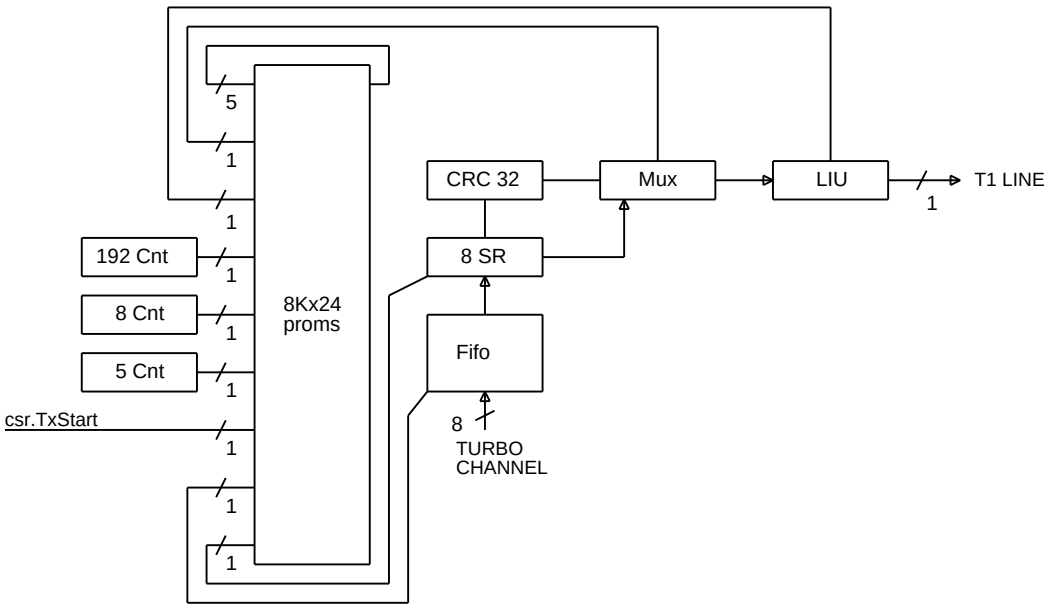

Figure 13: Block Diagram of the Transmitter

9. Design Choices

Why use Crystal Semi's LIU? At least thirteen companies make Line Interface Unit ICs; some just second-source others, some make separate receiver and transmitter chips, and some partition functions in odd ways. Figure 14 compares the nine LIU chips that looked promising. Tech = chip technology; RClk = Receiver clock recovery technique; Comp = external components; LOS = reports loss of signal; Loop = loopback capability; BPV = reports bipolar violations; Puls = transmitter pulse shaping; Jitt = attenuates jitter.

Company	Chip	Tech	RClk	Comp	LOS	BPV	Loop	Puls	Jitt
AT&T	LC1046	CMos	APLL	None	Yes	Yes	Yes	Yes	No
Crystal	CS61574	CMos	DDly	Xtal	Yes	No	Yes	Yes	Yes
Dallas	DS2185	CMos	APLL	C	Yes	Yes	xxx	xxx	Yes
Exar	T5683	Bip	Tank	RLC	No	No	No	No	No
Thomson	EFB7332	NMos	DPLL	Xtal	No	No	No	No	No
Rockwell	R8069	CMos	APLL	Xtal	Yes	Yes	Yes	Yes	Yes
Siemens	PEB2235	CMos	DPLL	Xtal	No	No	Yes	Yes	No
SilSys	78P233	Bip	APLL	RC	Yes	Yes	Yes	Yes	No
TI	TCM2203	Bip	Tank	RLC	Yes	No	No	No	No

Figure 14: Line Interface Unit ICs

Jitter attenuation was the deciding factor in choosing an LIU. When the transmitter clock is derived from the received pulse stream, jitter must be eliminated or it may build up to unacceptable levels by the time it gets back to the far end. The Rockwell chip was new and only available for sampling, and its input jitter tolerance was half that of the Crystal chip; the Dallas chip was a receiver only (but nicely done). The TI and Exar chips were rather old designs requiring many external components; the Silicon Systems chip required fewer components. The Mostek and Siemans chips were European designs using oversampling digital phase locked loops and no external components except crystals; both of them looked very good and would have been contenders if they had contained jitter attenuators. The AT&T chip was designed by the people who invented T1, but they didn't include a jitter attenuator. (In defense of LIUs without jitter attenuation, it is often done elsewhere in a design.) The Crystal Semiconductor LIU did everything required and only a few things not required (output driver monitoring and serial control interface).

Why use through-hole components rather than surface mount? The adapter uses UV-erasable and reprogrammable ROMs and PALs in socketed dual-inline packages, permitting custom design and easy modification of shift registers, counters, FSMs and random logic. Sockets for surface mount and dual-inline packages occupy about the same board area, but dual-inline sockets are more reliable and through-hole printed circuit boards are easier to make and modify.

Rather than doing extensive simulation before making the printed circuit board, the adapter was designed to be easy to modify by reprogramming the ROMs and PALs; every reprogrammable part was changed at least once during debugging and two wires were added to unused inputs. The added wires were the result of misunderstanding the operation of two ICs, so the behavioral models would have been wrong and the errors would not have been caught during simulation. The receiver frame sync algorithm *was* simulated, and it helped, but several changes were necessary when a real T1D4 signal was seen for the first time. Experience using the adapter suggested several changes to the software interface, which were easy to make by reprogramming. A programmable gate array, such as those made by Xilinx, is an attractive alternative to partitioning the design among a number of smaller chips.

Why include the DSU function rather than using an external DSU/CSU? Integrating the DSU with the HDLC logic resulted in a very compact design. An external DSU implies an electrical interface, such as EIA422/423/449 or V.35, between the HDLC logic and the DSU, and the ability to run at fractional T1 speeds. While designing a general-purpose synchronous serial adapter is a worthy goal, the goal here was to build a single-purpose T1 adapter. Multiprotocol routers exist and the cost of T1 circuits is plummeting, so the need to split a T1 circuit between two routers implementing different protocols, or to save money with a fractional T1 circuit will fade. Integrating the CSU function into the adapter would have been going too far; a CSU's interface is not as flexible as DSX-1.

Why implement framing rather than using a framer chip? Designing the frame recognizer was an intellectual challenge. Several companies make VLSI chips that do D4 and ESF framing, plus many features for treating a T1 circuit as 24 voice channels. Other than framing, these features are not needed, are sometimes hard to disable, and are always complicated to configure. Getting bits past these features without mangling is like walking through a mine field.

Why D4 framing rather than ESF? ESF is very complicated; D4 is *much* simpler to implement. Most T1 circuits use D4 framing; a circuit with ESF (and usually B8ZS also) is a special-order item not available everywhere. If ESF is absolutely required for some reason, then there are CSUs which can speak D4 on the customer side and ESF on the telco side; an EIA232 terminal port on the CSU gives access to the ESF features.

Why implement HDLC rather than using a serial chip? Many companies make VLSI chips that implement HDLC, from simple serial chips to complicated protocol chips that interpret the address and control fields. Like the T1 framer chips, HDLC chips are usually quite complex to configure, sometimes even including scatter/gather DMA. Transporting unreliable datagrams between routers requires only the most basic aspects of HDLC; the rest gets in the way.

Why programmed I/O rather than DMA? There was no room on the printed circuit board. A DMA, interface to the TurboChannel I/O bus uses lots of ICs to avoid a few extra kernel cycles copying data between CPU and adapter, but still requires an interrupt per packet. To get the full benefit, DMA must be coupled with main memory buffer management so that the CPU does not take an interrupt for each packet. This requires even more ICs and there simply wasn't room. Tightly packing surface mount ICs on two stacked PC boards might have done it, but it would have taken much longer to design and been much more complicated to use.

Why 32-bit CRC rather than 16-bit CRC? The error detecting ability of a 16-bit CRC decreases beyond a packet size of about 4K bits; the maximum packet size of this adapter is 16K bits. Ethernet, which has a maximum packet size of about 12K bits, uses a 32-bit CRC recommended in HDLC [19]. This adapter uses the same CRC because packets moving through the adapter probably come from or are destined for an Ethernet, so they should get the same level of protection.

Instead of hand-crafting D4 framing and HDLC using a few simple FSMs, a T1 adapter could be designed using a T1 framer chip connected to an HDLC chip which DMAed into an on-board buffer ram, similar to the design of DEC's PMAD-AA Ethernet adapter. Such a design would not require understanding the details of T1 or HDLC. It would certainly be more complex to program and there would be more ways to lose performance, but it is a workable approach.

10. Bugs

If the last five bits of the CRC were ones, then the TxFSM failed to insert a stuff bit. The receiver took the first bit of the ending flag as a stuff bit, and things went downhill from there. This turns out to be a common error in HDLC implementations; several standards documents mention that a stuff bit may be required after the last CRC bit and before the closing flag. One extra state was added to the TxFSM to fix this bug. IP protocols managed to work despite the fact that about 3 percent of the packets were being discarded due to bad receiver status.

The receiver's frame synchronizer would falsely lock onto the transmitter's HDLC flag pattern 25 percent of the time. If a zero bit in an HDLC flag was picked as a possible frame bit, then on an idle link with all flags, every 193rd bit after that would also be a zero, the FrameRom would say Maybe 40 times, and frame synchronization would be falsely declared. Simulations of the frame sync algorithm used random data and did not find this problem. A two-line change to the RxFSM fixed this bug.

The TurboChannel clock is used to clock the FSMs in the adapter. For a maximum-speed 25 MHz TurboChannel, the clock is divided by two giving an 80 ns period, or about eight cycles per T1 bit. For a minimum-speed 12.5 Mhz TurboChannel, dividing by two results in four cycles per bit, which is not enough. The clock division is done in a PAL, so the quick fix is to use a different PAL for fast and slow systems; the long-term fix is to make this a CSR bit.

Several companies make routers that use T1 lines and it would be good if they could interoperate with this adapter. They use HDLC chips (such as the 8530) through a V.35 interface to a DSU/CSU which handles framing and clocking. Most HDLC chips only implement 16-bit CRCs. The CRC chips used in this design were advertised to implement both 16- and 32-bit polynomials, but 16-bit polynomial checking only works if the CRC bits are inverted before being shifted into the chip (it does the right thing for 32 bits). By reprogramming PROMs to invoke hidden features, the adapter can be made to generate packets with correct 16-bit CRCs, but it can't check the 16-bit CRC of an incoming packet (software can do it fairly inexpensively).

HDLC shifts data least significant bit first, but the adapter shifted out the most significant bit first. The order was reversed by reprogramming the transmitter and receiver shift register PALs, but the receiver CRC input was hard-wired to the wrong end of the SR. A printed circuit trace

had to be cut and a wire added; the adapter talked to itself or others of its kind just fine the old way.

The RxFSM was generating FIFO write pulses when the FIFO was full. The data sheet implied that this was harmless, but by pounding on the adapter for about six hours in a particular way, the FIFO could be driven into a bad state. FifoWrite was already passing through a PAL for other reasons, so it was simple to quash it if FifoFull was true; this required adding a wire from FifoFull to an unused input on the PAL. The FifoOverflow logic had to be reprogrammed as a side effect.

11. Acknowledgements

Paul Vixie wrote an Ultrix device driver for IP packets and helped to work out the operation of TxStart. Paul, Joel Bartlett and Bill Hamburgen suggested ways to improve this report.

12. References

[1] American National Standards Institute.
Synchronization Interface Standards for Digital Networks.
American National Standard T1.101-1987, ANSI, December, 1986.

[2] American National Standards Institute.
Digital Hierarchy - Electrical Interfaces.
American National Standard T1.102-1987, ANSI, August, 1987.

[3] American National Standards Institute.
Carrier-to-Customer Installation--DS1 Metallic Interface.
American National Standard T1.403-1989, ANSI, 1989.

[4] American National Standards Institute.
Advanced Data Communication Control Procedures (ADCCP).
American National Standard X3.66-1979, ANSI, January, 1979.

[5] American Telephone and Telegraph Company.
High Capacity Terrestrial Digital Service.
Technical Reference 41451, AT&T, January, 1983.

[6] American Telephone and Telegraph Company.
Digital Channel Bank Requirements and Objectives.
Technical Reference 43801, AT&T, November, 1982.

[7] American Telephone and Telegraph Company.
Extended Superframe Format.
Technical Reference 54016, AT&T, May, 1986.

[8] American Telephone and Telegraph Company.
Digital Synchronization Network Plan.
Technical Reference 60110, AT&T, December, 1983.

[9] American Telephone and Telegraph Company.
ACCUNET T1.5 Service Description and Interface Specifications.
Technical Reference 62411, AT&T, October, 1985.

[10] American Telephone and Telegraph Company.
Interconnection Specification for Digital Cross-Connects.
Compatibility Bulletin 119, AT&T, October, 1979.

[11] Bell Communications Research.
Asynchronous Digital Multiplexes Requirements and Objectives.
Technical Reference TSY-000009, BellCore, May, 1986.

[12] Bell Communications Research.
Synchronous DS3 Add-Drop Multiplex (ADM 3/X) Requirements and Objectives.
Technical Reference TSY-000010, BellCore, February, 1988.

[13] Bell Communications Research.
Extended Superframe Format (ESF) Interface Specification.
Technical Reference TSY-000194, BellCore, December, 1987.

[14] Bell Communications Research.
Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria.
Technical Reference TSY-000253, BellCore, September, 1989.

[15] International Telegraph and Telephone Consultative Committee.
Digital Hierarchy of Bit Rates.
Recommendation G.702, CCITT, November, 1988.

[16] International Telegraph and Telephone Consultative Committee.
Physical/Electrical Characteristics of Hierarchical Digital Interfaces.
Recommendation G.703, CCITT, November, 1988.

[17] International Telegraph and Telephone Consultative Committee.
Functional Characteristics of Interfaces Associated with Network Nodes.
Recommendation G.704, CCITT, November, 1988.

[18] International Telegraph and Telephone Consultative Committee.
The control of jitter and wander within digital networks which are based on the 1544 kbit/s hierarchy.
Recommendation G.824, CCITT, November, 1988.

[19] International Standards Organization.
HDLC Frame structure.
International Standard 3309, ISO, 197?

[20] C. Davis.
An Experimental Pulse Code Modulation system for Short-Haul Trunks.
Bell System Technical Journal 41(1):1-24, January, 1962.

[21] J. Mayo.
A Bipolar Repeater for Pulse Code Signals.
Bell System Technical Journal 41(1):25-97, January, 1962.

[22] M. Aaron.
PCM Transmission in the Exchange Plant.
Bell System Technical Journal 41(1):99-141, January, 1962.

[23] K. Fultz and D. Penick.
The T1 Carrier System.
Bell System Technical Journal 44(5):1405-1451, September, 1965.

[24] A. Duttweiler.
Waiting Time Jitter.
Bell System Technical Journal 51(1):165-207, January, 1972.

[25] C. Byrne, R. Karafin and D. Robinson.
Systematic Jitter in a Chain of Digital Regenerators.
Bell System Technical Journal 42(6):2679-2714, November, 1963.

[26] International Business Machines Corporation.
Binary Synchronous Communications.
General Information GA-27-3004-2, IBM, 1970.

[27] International Business Machines Corporation.
IBM Synchronous Data Link Control.
General Information GA-27-3093, IBM, 197?

[28] R. Donnan and J. Kersey.
Sunchronous Data Link Control: A Perspective.
IBM Systems Journal 13:140-162, May, 1974.

[29] Digital Equipment Corporation.
TurboChannel Hardware Specification.
Document EK-369AA-OD, DEC, 1991.

[30] Digital Equipment Corporation.
TurboChannel Firmware Specification.
Document EK-TCAAD-FS, DEC, 1991.

[31] Digital Equipment Corporation.
TurboChannel Mechanical Drawings.
Document EK-TCAAD-OM, DEC, 1991.

WRL Research Reports

‘‘Titan System Manual.’’
Michael J. K. Nielsen.
WRL Research Report 86/1, September 1986.

‘‘Global Register Allocation at Link Time.’’
David W. Wall.
WRL Research Report 86/3, October 1986.

‘‘Optimal Finned Heat Sinks.’’
William R. Hamburgen.
WRL Research Report 86/4, October 1986.

‘‘The Mahler Experience: Using an Intermediate Language as the Machine Description.’’
David W. Wall and Michael L. Powell.
WRL Research Report 87/1, August 1987.

‘‘The Packet Filter: An Efficient Mechanism for User-level Network Code.’’
Jeffrey C. Mogul, Richard F. Rashid, Michael J. Accetta.
WRL Research Report 87/2, November 1987.

‘‘Fragmentation Considered Harmful.’’
Christopher A. Kent, Jeffrey C. Mogul.
WRL Research Report 87/3, December 1987.

‘‘Cache Coherence in Distributed Systems.’’
Christopher A. Kent.
WRL Research Report 87/4, December 1987.

‘‘Register Windows vs. Register Allocation.’’
David W. Wall.
WRL Research Report 87/5, December 1987.

‘‘Editing Graphical Objects Using Procedural Representations.’’
Paul J. Asente.
WRL Research Report 87/6, November 1987.

‘‘The USENET Cookbook: an Experiment in Electronic Publication.’’
Brian K. Reid.
WRL Research Report 87/7, December 1987.

‘‘MultiTitan: Four Architecture Papers.’’
Norman P. Jouppi, Jeremy Dion, David Boggs, Michael J. K. Nielsen.
WRL Research Report 87/8, April 1988.

‘‘Fast Printed Circuit Board Routing.’’
Jeremy Dion.
WRL Research Report 88/1, March 1988.

‘‘Compacting Garbage Collection with Ambiguous Roots.’’
Joel F. Bartlett.
WRL Research Report 88/2, February 1988.

‘‘The Experimental Literature of The Internet: An Annotated Bibliography.’’
Jeffrey C. Mogul.
WRL Research Report 88/3, August 1988.

‘‘Measured Capacity of an Ethernet: Myths and Reality.’’
David R. Boggs, Jeffrey C. Mogul, Christopher A. Kent.
WRL Research Report 88/4, September 1988.

‘‘Visa Protocols for Controlling Inter-Organizational Datagram Flow: Extended Description.’’
Deborah Estrin, Jeffrey C. Mogul, Gene Tsudik, Kamaljit Anand.
WRL Research Report 88/5, December 1988.

‘‘SCHEME->C A Portable Scheme-to-C Compiler.’’
Joel F. Bartlett.
WRL Research Report 89/1, January 1989.

‘‘Optimal Group Distribution in Carry-Skip Adders.’’
Silvio Turrini.
WRL Research Report 89/2, February 1989.

‘‘Precise Robotic Paste Dot Dispensing.’’
William R. Hamburgen.
WRL Research Report 89/3, February 1989.

‘‘Simple and Flexible Datagram Access Controls for Unix-based Gateways.’’
Jeffrey C. Mogul.
WRL Research Report 89/4, March 1989.

‘‘Spritely NFS: Implementation and Performance of Cache-Consistency Protocols.’’
V. Srinivasan and Jeffrey C. Mogul.
WRL Research Report 89/5, May 1989.

‘‘Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines.’’
Norman P. Jouppi and David W. Wall.
WRL Research Report 89/7, July 1989.

‘‘A Unified Vector/Scalar Floating-Point Architecture.’’
Norman P. Jouppi, Jonathan Bertoni, and David W. Wall.
WRL Research Report 89/8, July 1989.

‘‘Architectural and Organizational Tradeoffs in the Design of the MultiTitan CPU.’’
Norman P. Jouppi.
WRL Research Report 89/9, July 1989.

‘‘Integration and Packaging Plateaus of Processor Performance.’’
Norman P. Jouppi.
WRL Research Report 89/10, July 1989.

‘‘A 20-MIPS Sustained 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance.’’
Norman P. Jouppi and Jeffrey Y. F. Tang.
WRL Research Report 89/11, July 1989.

‘‘The Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance.’’
Norman P. Jouppi.
WRL Research Report 89/13, July 1989.

‘‘Long Address Traces from RISC Machines: Generation and Analysis.’’
Anita Borg, R.E.Kessler, Georgia Lazana, and David W. Wall.
WRL Research Report 89/14, September 1989.

‘‘Link-Time Code Modification.’’
David W. Wall.
WRL Research Report 89/17, September 1989.

‘‘Noise Issues in the ECL Circuit Family.’’
Jeffrey Y.F. Tang and J. Leon Yang.
WRL Research Report 90/1, January 1990.

‘‘Efficient Generation of Test Patterns Using Boolean Satisfiablilty.’’
Tracy Larrabee.
WRL Research Report 90/2, February 1990.

‘‘Two Papers on Test Pattern Generation.’’
Tracy Larrabee.
WRL Research Report 90/3, March 1990.

‘‘Virtual Memory vs. The File System.’’
Michael N. Nelson.
WRL Research Report 90/4, March 1990.

‘‘Efficient Use of Workstations for Passive Monitoring of Local Area Networks.’’
Jeffrey C. Mogul.
WRL Research Report 90/5, July 1990.

‘‘A One-Dimensional Thermal Model for the VAX 9000 Multi Chip Units.’’
John S. Fitch.
WRL Research Report 90/6, July 1990.

‘‘1990 DECWRL/Livermore Magic Release.’’
Robert N. Mayo, Michael H. Arnold, Walter S. Scott, Don Stark, Gordon T. Hamachi.
WRL Research Report 90/7, September 1990.

‘‘Pool Boiling Enhancement Techniques for Water at Low Pressure.’’
Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.
WRL Research Report 90/9, December 1990.

‘‘Writing Fast X Servers for Dumb Color Frame Buffers.’’
Joel McCormack.
WRL Research Report 91/1, February 1991.

‘‘Analysis of Power Supply Networks in VLSI Circuits.’’
Don Stark.
WRL Research Report 91/3, April 1991.

‘‘TurboChannel T1 Adapter.’’
David Boggs.
WRL Research Report 91/4, April 1991.

‘‘Procedure Merging with Instruction Caches.’’
Scott McFarling.
WRL Research Report 91/5, March 1991.

‘‘Don’t Fidget with Widgets, Draw!.’’
Joel Bartlett.
WRL Research Report 91/6, May 1991.

‘‘Pool Boiling on Small Heat Dissipating Elements in Water at Subatmospheric Pressure.’’
Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.
WRL Research Report 91/7, June 1991.

‘‘Incremental, Generational Mostly-Copying Garbage Collection in Uncooperative Environments.’’
G. May Yip.
WRL Research Report 91/8, June 1991.

WRL Technical Notes

‘‘TCP/IP PrintServer: Print Server Protocol.’’
Brian K. Reid and Christopher A. Kent.
WRL Technical Note TN-4, September 1988.

‘‘TCP/IP PrintServer: Server Architecture and Implementation.’’
Christopher A. Kent.
WRL Technical Note TN-7, November 1988.

‘‘Smart Code, Stupid Memory: A Fast X Server for a Dumb Color Frame Buffer.’’
Joel McCormack.
WRL Technical Note TN-9, September 1989.

‘‘Why Aren’t Operating Systems Getting Faster As Fast As Hardware?’’
John Ousterhout.
WRL Technical Note TN-11, October 1989.

‘‘Mostly-Copying Garbage Collection Picks Up Generations and C++.’’
Joel F. Bartlett.
WRL Technical Note TN-12, October 1989.

‘‘Limits of Instruction-Level Parallelism.’’
David W. Wall.
WRL Technical Note TN-15, December 1990.

‘‘The Effect of Context Switches on Cache Performance.’’
Jeffrey C. Mogul and Anita Borg.
WRL Technical Note TN-16, December 1990.

‘‘MTOOL: A Method For Detecting Memory Bottlenecks.’’
Aaron Goldberg and John Hennessy.
WRL Technical Note TN-17, December 1990.

‘‘Predicting Program Behavior Using Real or Estimated Profiles.’’
David W. Wall.
WRL Technical Note TN-18, December 1990.

‘‘Systems for Late Code Modification.’’
David W. Wall.
WRL Technical Note TN-19, June 1991.

TurboChannel T1 Adapter

Table of Contents

1. Overview **1**
2. Installation Information **1**
3. What is T1? **4**
4. What is HDLC? **6**
5. What is a TurboChannel? **7**
6. Programming Interface **7**
6.1. Control/Status Register **8**
6.2. Receiver Data Register **9**
6.3. Transmitter Data Register **11**
6.4. Option ROM **12**
7. Diagnostic Program **13**
7.1. Program Variables **13**
7.2. Diagnostic Commands **14**
7.3. Using the Commands **15**
8. Technical Description **16**
8.1. Data Path **17**
8.2. Receiver **17**
8.3. Transmitter **19**
9. Design Choices **20**
10. Bugs **22**
11. Acknowledgements **23**
12. References **24**

List of Figures

Figure 1: Some Possible Configurations **1**
Figure 2: Option Module Layout **2**
Figure 3: Connector Pins **3**
Figure 4: Digital Hierarchy **4**
Figure 5: T1 Waveform **4**
Figure 6: Control/Status Register (CSR) **8**
Figure 7: Receiver Data Register (RxData) **10**
Figure 8: Receiver Status (RxStatus) **10**
Figure 9: Transmitter Data Register (TxData) **12**
Figure 10: Data Paths in the Adapter **17**
Figure 11: Block Diagram of the Receiver **18**
Figure 12: Loss-of-frame lock threshold **19**
Figure 13: Block Diagram of the Transmitter **19**
Figure 14: Line Interface Unit ICs **20**