

Arcnet Processing

Program overview

Thu, Sep 28, 2000

Overview

This note describes some of the arcnet processing logic used by IRMs to collect data from SRMs in the Linac control system. All arcnet communication is interrupt-driven. The logic is designed following a model used by the token ring chip set, since the original network support software was done to support token ring. All arcnet communications are based upon the Arcnet header. IP protocols are not used for arcnet communications.

In order to keep traffic moving so that task-level processing is not held up awaiting transmit completion, frames ready to be sent to the network are built in a circular transmit buffer area, and a pointer to each frame is placed into a "transmit parameter list" queue that is 8K bytes in length, allowing for 127 entries. When a pointer to a frame is placed in this queue, and arcnet transmissions are not active, then the arcnet transmit interrupt is enabled to "prime the pump." The interrupt routine realizes that no transmit is pending, so it merely starts transmission for the new message. In the case that arcnet transmissions are active when a new pointer is placed into the arcnet TPL, nothing has to be done, since a subsequent transmit interrupt will eventually find the new entry and send it out. Only after the completion interrupt for the last frame is received, and there is no more entry in the TPL, will the transmit interrupt be disabled. This kind of logic is analogous to some RS-232 serial interfaces. The point is that a transmit interrupt occurs whenever both the transmit interrupt enable bit is set and transmitter buffer available status is set. When the transmitter is idle, the transmitter buffer is available, so that the interrupt must be disabled in order to keep the interrupt from occurring.

Reception of arcnet frames requires some care. Most of the arcnet messages received are replies to data requests made to SRMs every 15 Hz cycle. The requests are sent out during Data Access Table processing in the Update Task. It is necessary to receive and process the replies so that such data can be available for subsequent use by data Access Table instructions, including Local Applications. In order to make it seem that the entire data pool is updated instantly every cycle, the Update Task does not relinquish control of the CPU until the data pool is completely updated for that cycle. This means that a DAT instruction following one that has sent a request message must await the arrival of reply data. While it is waiting, the message queue is checked to learn of new messages coming in that may match the node waited upon. For this reason, the dispatch processing for such messages that would normally be handled by the ANet task—which cannot run until the Update Task finishes its work—must be handled by the arcnet receive interrupt logic.

Arcnet interrupts

ARCINT loops as long as the status register indicates an interrupt condition is present by any of several bits of the byte-wide arcnet controller status register being set. The significance of the bits is as follows:

Bit#	Meaning
7	Receive interrupt
0	Transmit interrupt
4	Power-on-reset interrupt (non-maskable)
2	Reconfigure interrupt

Thus, the mask representing all enabled interrupt sources is 0x85. Without a transmit buffer active, it is 0x84.

Receive interrupt

Of the four 512-byte arcnet hardware buffers, two buffers are alternately used for frame reception. As soon as a receive interrupt condition is detected from examining the status register, the other buffer is enabled for reception—by writing to the arcnet controller command register—in order to minimize the dead time during which no buffer is ready to receive a frame. Attention is then placed on the buffer just filled, and ACCEPT is called, followed by RCVINT. Finally, the current buffer is switched to await the next receive interrupt.

ACCEPT first examines the destination node byte in the received frame. It must be either zero, indicating broadcast, or equal to the local node number. (All arcnet controllers in use are set to use node 0x10, with the possible exception of the controller used by NTF in node061B, which may be set to 0xAB. SRMs use node numbers in the range 0xA1–AF.) The frame length is determined, and the contents of the frame are copied into the 64K-byte Arcnet circular receive buffer located at 0x00070000 in "low memory." (This buffer area "wraps" in about 3 seconds for a busy node such as node062E, which collects data from 6 SRMs at 15 Hz.) The frame contents are preceded by a fake token ring-style frame header in order to facilitate subsequent task-level processing. The token ring header includes a DSAP byte in its IEEE 802.2 structure. The SAP used is 0x68, which is used for Acnet-header-based communications. A "receive parameter list" structure, also inspired by token ring, is maintained to keep track of the current frame buffer in use in the circular buffer area. The BUFF1 field holds the pointer to the current buffer. The FSIZE field holds the size of the frame received, and the CSTAT field is set to 0x7000 to indicate a valid frame was received.

RCVINT refers to the BUFF1 field in the receive parameter list structure to find the buffer holding the just-received frame. It checks for suitable size message. (All messages must be more than the size of the 18-byte Acnet header, because that header is included for all valid messages in our implementation of Arcnet communications with SRMs. The CSTAT field is checked to insure it is 0x7000. The two-word header for the entire frame record, a size word and a message count word, is initialized.

A frame reference message (FMREF) structure is prepared. The DSAP byte is checked for a match in a SAP table entry, and if the entry has a message queue id, the FMREF is queued. Assuming no errors, SCANDAQ is called.

The message count word preceding the frame is initialized to 0x7FFF, or to zero in case of errors. Advancing past the received frame, the receive parameter list structure is prepared for reception of the next frame. (This means that BUFF1 is advanced and CSTAT is initialized to 0x8800.

SCANDAQ is called after preparing the FMREF structure, so these fields are used to find the frame in the circular receive buffer area. Every message found in the frame is scanned as a MGREF structure is built for each. If a message is found that is a reply to the task-id used by Data Access Table requests, or if a request is found that targets the SRMD task name, an effort is made to process the message and pass the MGREF structure to the appropriate message queue. If the message does not match either case, it will be passed via the ANet task in the usual way.

The reason that some requests, and not merely replies, must be processed preemptively has to do with server support for SRMD data requests that may be handled via a local application. In the NTF control system, node061C collects data from SRMs in the usual way. But one SRM is actually another node (node061B that is not connected to the normal controls token ring network) that is running a local application that supports the same SRMD protocol that

SRMs do. This arcnet connection serves to isolate the low-level NTF controls from the rest of the network of Linac front ends. This helps to alleviate fears of errors that might affect neutron therapy patient treatment. It also means that node061C can collect correlated data from NTF; data obtained from its arcnet link is measured on the same cycle as that obtained from its other SRM.

Transmit interrupt

XMTINT first checks whether a frame has been transmitted by checking for a bit set in XBFUL. If the bit is not set, the interrupt occurred merely to start the first frame going. If the bit was set, it means that this interrupt marks the completion of a transmitted frame. Completion processing involves checking the XDIS flag to see whether the interrupt was caused by disabling the transmitter after timing out from trying to get the interrupt to occur by setting the transmit enable. (This would imply that TA, or transmit available, status was not set. Disabling the transmitter should force an interrupt in that case.) If this unlikely case occurs, CSTAT is set to 0x7888, signifying address recognized but frame not copied, to put it into token ring parlance.

Assuming that XBFUL was set, and that XDIS was not set, the TMA bit, for Transmit Message Acknowledged, is checked. If it is set, a positive acknowledgment was received from the destination node upon successful reception of the transmitted message. In this case, CSTAT is set to 0x78CC, meaning no errors. (In token ring terms, it means "address recognized" and "frame copied.") If the TMA bit is zero, then CSTAT is set to 0x7800, meaning address not recognized; i.e., the transmission failed.

The time of receiving the transmit interrupt is measured from the time the frame was enabled for transmission by the hardware. This is only a diagnostic.

The CSTAT field is interpreted into a transmit status code that can be returned to a user that specified an xmitStat variable when originally queuing the message. The codes are as follows:

<i>CSTAT status</i>	
0x78CC	0
0x7888	-1
0x7800	-2

The news about the status word is delivered to all xmitStat variables for each message in the frame. A field in the TPL entry identifies which message block pointer entries were used to build up the frame that was sent. Examining each such message block, the pointer to the xmitStat variable is found and the status value stored therein. This feature of returning transmit status to the user is seldom used.

Independent of whether the transmit interrupt occurred as the completion interrupt of a transmitted frame, the TPL is checked for another waiting frame to be transmitted. If there is one, as indicated by the sign bit in the CSTAT field, the frame is copied from the transmit frame buffer into the hardware buffer. Only the third buffer is used for transmitted frames; the fourth hardware buffer is not used at all. The XBFUL bit is set, and the buffer is enabled for transmission by writing to the controller command byte. The time is recorded in the TPL entry.

Pictorial flow

