Alarms “tries needed” Upgrade

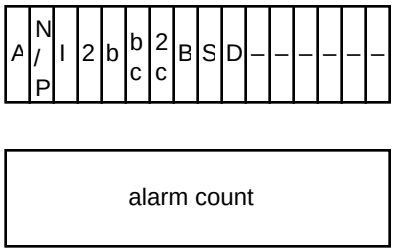
For a more patient alarm system

Thu, Jan 7, 1993

It has been requested that more than a “two-times” option be allowed in the alarm scanning logic of the Local Control Station systems. Until now, one could optionally require an analog or digital signal to be in the alarm state for one or two times before it was declared “bad” and emited an alarm message and optionally asserted a beam inhibit signal. This note describes a new feature that supports the specification of up to 16 times before an alarm is declared. A mapping is made with the accelerator RETDAT/SETDAT protocol so that the ***tries_now*** and the ***tries_needed*** byte fields of the Acnet alarm blocks are supported.

Implementation

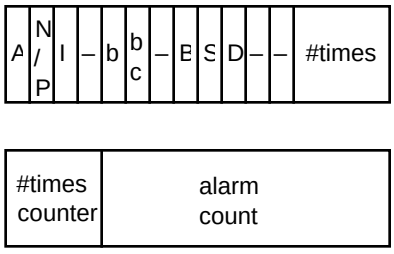
The previous structure of the alarms flags and alarm count words was:



A= Active
N/P= Nominal bit state/Pattern
I= Inhibit beam
2= Two times option
b= Beam pulses option
bc= Bypass control
2c= Two times counter
B= Good/Bad state
S= Silent (no message)
D= Bad data—ignore

A single bit was used for the two-times option and another for the count to 2.

The new structure for the alarm flags and alarm count words is now:



A= Active
N/P= Nominal bit state/Pattern
I= Inhibit beam
b= Beam pulses option
bc= Bypass control
B= Good/Bad state
S= Silent (no message)
D= Bad data—ignore
#times= Tries needed - 1

Here a 4-bit field is used to support ***tries_needed*** values of 1–16. The 2 and 2c bits are no longer used. The alarm count is now 12 bits to provide room for the counter.

Note that using this scheme for large values of ***tries_needed*** necessarily delays the response of the alarm system for generating an alarm message and for inhibiting beam for up to 16 cycles—about one second at 15 Hz.

Analog Descriptor Page

Local Station Application

Feb 25, 1992

Introduction

An analog descriptor is the Local Station's static database for an analog channel device. The database is organized as an array of Pascal records. The fields of this 64-byte record are used to support the full functionality of an analog channel. This page allows access to any node's analog database.

Display layout

```
A ANALOG DESCRIPT 02/25/92 1107
CHAN<0611:0502>  -SE  FAM<0000>        Node:Chan
NAME<GR2MID> "                  "      Name
TITL<RF2 GRDIENT MANAUT>02/13/92 ö     Date-of-last-change
CONV<00>  Z=4,C=2,G=8,P=9         ÷
A/D FS< 8.34  > +< 0.166 ><NRM > ÷
D/A FS< 18000 > +< 0.199 >        ÷     ADESC fields
ANLG CTRL <02> <A3> <B015>        ÷
STAT INV/N<11>    BIT<05AF,-->    ÷
CONTROL DT<01,--> BIT<05AF,-->    ø

 LIST DESCR TO :07FF   LIST DATA      Listing options
ALARM FLG$<A000>API2B---S-------      Alarm flag bits
     TRIPS:   0  *RESET ALARMS        Reset local station alarms
```

The format is designed for fill-in-the-blanks interaction. On the second line, the node/channel is entered for the device whose analog descriptor is to be accessed; alternatively, the name can be entered on the third line. Interrupt after typing to request the current database information for that device to be displayed. The following lines include the various fields of the analog descriptor:

1. 18-character title
2. 6-character name
3. 4-character engineering units text
4. 1-byte "conversion type"
5. 4 floating point scaling constants
6. 4-byte analog control field
7. 2-byte family word
8. 4-byte associated digital status field
9. 6-byte associated digital control field

Field descriptions

Title text

This 18-character text field is a short description of the channel device in addition to the name. Due to the limited size of the small screen consoles, only 14 characters of the title are displayed by the parameter page. Those selected are the 14 characters following the first *word* of the title. The first word is often used to indicate the node of the device, which is often implied in the name. In the case that associated status / control fields are used, the last 6 characters of the title text (characters 13–18) are used to store two 3-characters status state texts. In the case that two bits of associated status / control are used, the entire title is preempted, as the second bit's status text is stored in characters 6–11 of the title. So, in this case, the name alone must suffice to represent the device on the parameter page.

Name text

The 6-character name of a device must be unique on the network, not only within a single node. When a name is entered and the interrupt given to enter all fields into a given channel's descriptor, a broadcast request is simultaneously issued to find out whether the name already exists among the nodes of the network. If it does exist in some node, then its node / channel designation is displayed just to the left of the `NAME` prompt. The setting has already taken place, so at that point there is at least one duplication of the entered name. One may wish to remove the one indicated or use a different name for the channel.

Units text

The 4-character field is placed into the 4-byte units text field of the analog descriptor. The parameter page aligns this field if possible.

Conversion type

This field is a byte that includes flags that mark the channel for various special treatments with certain Data Access Table entries. Bit #1 (mask=$02) is set to mark capture data, which is used by DAT type $16. Bit #2 (mask=$04) is set to mark zero-subtracted data, which is used by DAT type $07. Bit#3 (mask=$08) is used to mark channels which need linearization, which is used by DAT type $06. See the document entitled "RDATA Entry Formats" for more details on this.

Scale factors

Four floating point constants are used for engineering units scaling. A reading, nominal or tolerance value is scaled to engineering units by using the first fullscale and offset values. The setting value is scaled using the last two fullscale and offset values. The linear formula used is:

```
eng:= Float(raw)/32768.*fScale + offset;
```

The `raw` value is the 16-bit raw data word. The value (`raw/32768`) is the fraction of fullscale. Both `fScale` and `offset` are in engineering units. More

complex conversion formulae can be handled by pre-processing via a Data Access Table entry, for example. The raw data values are always scaled linearly.

In the case of tolerance, the offset term is omitted, since a tolerance is like a difference value.

Analog control field

The first byte of this 4-byte field is the analog control type#. Types currently supported are:

00 No analog control (parameter page will not mark it with a "–")
01 Datel Multibus board (obsolete)
02 Motor (setting value is desired reading, relative setting is #steps)
03 Bipolar multiplex D/A (obsolete)
04 Unipolar multiplex D/A (obsolete)
05 Memory word (accessed as two bytes)
06 i8253 timer (obsolete)
07 M6840 timer (obsolete)
08 1553 D/A (12-bit)—used in rack monitor
09 Analog Devices RTI-602 D/A board
0A Memory word (accessed as a word)
0B Message queue setting to another cpu (co-processor)
0C Unsigned 12-bit D/A (in short I/O space)
0D Burr-Brown MPV904 12-bit D/A board
0E 1553 D/A (16-bit)
0F AMD9513 timer (32-bits from pair of channels)
10 Memory byte (single byte no shift)
11 Memory byte (single byte w/shift in short I/O space)
12 Same channel reading word w/mask
13 Smart Rack Monitor analog control (12-bit)
14 Smart Rack Monitor analog control (16-bit)

Details of the meaning of the other 3 bytes available in the analog control field are found in the document entitled "Analog Control Types."

Family word

The family word is a delta channel# used to reference another channel which is in some way related to this one. By including the appropriate delta value for a set of related channels, one can form a family of channels which is accessible from any member of the family. Listype #49 may be used to request the list of channel numbers that comprise the family to which a given channel belongs. This is more fully described in the document entitled "Related Groups of Channels."

Associated digital status/control

One or two bits can be associated with an analog channel to show related

status information such as on/off and allow digital control as well. This topic is more fully discussed in the document entitled "Digital Control Pulse Delays."

Database entry

After selecting the channel, change any of the fields displayed to the desired values and press interrupt (or the ESC key) with the cursor anywhere in the range of the 7 lines following the NAME line. The program reads *all* the fields that are alterable from the screen, including the node/channel number on the second line and the name on the third line, and it encodes the information into the internal format and issues a setting to install the descriptor record into that channel's local database. (Since it reads the channel number each time, one can easily move one channel's descriptor to another channel with minor editing.) It then requests the descriptor to be read back and displays the results. The date-of-last-change, displayed on the second line, should then reflect the current date. This field is not directly settable, but it is automatically updated whenever a setting is made to any of the fields in the channel's analog descriptor.

Separate from the analog descriptor fields for a channel device, there are alarm flag bits that can be set on the next to last line of the display. This flags word is shown in hexadecimal. Single characters are shown to the right as a reminder of each bit's meaning. The current set of flag bits available for setting are:

15 Active (1=enabled for alarm scan)
14 Pattern (1=composite status word)
13 Inhibit (1=inhibit beam while bad)
12 Two times (1=require twice in succession for change-of-state)
11 Beam (1=alarm scan only on beam pulses)
10 Bypass control—used internally
9 Two times counter—not settable
8 Good/bad (1=bad)—not settable
7 Silent (1=inhibit sending of alarm messages)

To modify these flag bits for a given channel whose descriptor is being displayed, enter the desired hex pattern and interrupt on that line.

To facilitate the inspection of families of channels, one can interrupt under the FAM area on the display to sequence to the next channel in the family (if there is one). Note that a value of $0000 in the family word means there is no next channel in the family.

Setting enable flag—important!

When the page is entered, an internal flag is set to disabled to prevent any settings from being made accidentally while browsing. To toggle this flag to enable settings, interrupt in the –SE field in the middle of the second line. The

response to show that the flag is enabled is *SE. If an attempt is made to change a descriptor without the flag enabled, the –SE field will flash a few times as a reminder that settings are not enabled.

Raise/lower switches

One can use the raise/lower push-buttons to increment/decrement the current channel# in order to scan through a sequence of channels. A delay is introduced so the data displayed can be seen by the human eye. Incrementing beyond the maximum channel# shown on the listing line wraps to channel 0.

Error status

An error status code number is displayed as a single character at the end of the second line. If there are no errors, this character will be blank; otherwise, its value is displayed in inverse video. Likely values that may be displayed are:

8 At least one node is not responding to the request for data.

When no reply is received from a name lookup request, the same code will be shown immediately following the name field on the third line.

Alarm info

The last line shows the alarm trip count for the channel, which is the number of times the channel has been observed by the alarm scanning logic to change state from good to bad. This value is limited on the display to 9999.

Interrupt on the right side of this line to reset the alarms for the target node, resulting in zeroing the good/bad alarm flag bits for all analog channels and binary bits in that node. On the next alarm scan, then, every device which is in the alarm scan and in the bad state will report an alarm message. It is more usual for a host system to issue a broadcast setting to reset alarms in all nodes.

Listing options

Two listings of analog database entries are provided on a listing line. On the left-hand part of the line, an interrupt produces a listing of each channel's ADESC entries, just as they are shown on the page. On the right-hand part of the line, an interrupt produces a listing of the ADATA table entries (the reading, setting, nominal and tolerance values). The listing starts with the channel currently selected on the page and ends with the channel# (in hex) indicated on the listing line, which is by default set to the maximum channel# available in that node The data for each channel is encoded into a single line, and a heading is output at the beginning that shows the target node# and the current time-of-day. The display is updated as the channels are scanned in sequence over the selected range. When a channel is accessed that is not used (meaning the name starts with a blank or a null byte, and the number of associated status bits is zero, and the analog control type is zero, and the channel is not in the alarm scan), it is skipped

and omitted from the listing. The channel's data is displayed very briefly to at least give a visual hint of what's there.

The output of the serial port can be plugged into any RS-232 device, so that one can archive the contents of a node's local database in this way. It could be stored on a host's disk or captured by a Macintosh terminal emulator program or whatever. The serial output from a Local Station is spooled through dynamic memory to the serial output port. This means that the listing process may finish before all the characters have been transferred out the serial port. To stop the listing process, merely interrupt again on the listing line while it is active.

Here is an example of the two forms of listing output:

Analog descriptors

```
 NODE=0611  ANALOG DESCRIPT 02/25/92 1254
 CHAN#  NAME   TITLE       ..... UNIT  ANLG CTRL  CVT ADFS   OFFS   DAFS   OFFS    FMLY   DATE    DIGITAL
STATUS/CONTROL
  :0500 IN2PHS T2-1 INTERTKMANAUT V     00 .. .... 00  10     0      0      0            02/10/92  11 05AB,--   01,--
05AB,--
  :0501 GR2LO  RF2 PICKUP LOOP #1 NRM   00 .. .... 08  8.34   0.166  10     0.199        01/15/92  00
  :0502 GR2MID RF2 GRDIENT MANAUT NRM   02 A3 B015 08  8.34   0.166  18000  0.199        02/13/92  11 05AF,--   01,--
05AF,--
  :0503 GR2HI  RF2 PICKUP LOOP #3 NRM   00 .. .... 08  8.34   0.166  0      0.199        01/15/92  00
  :0504 PA2F   RF2 PA FWD     CBR MW    00 .. .... 09  31.15  0      0      0.265        02/14/92  11 0515,--   84,--
0595,--
  :0505 PA2R   RF2 PA REV POWER   MW    00 .. .... 09  22.61  0      0      0.24         01/15/92  00
  :0506 DR2F   RF2 DRVR FWD POWER KW    00 .. .... 09  831    0      0      0.215        01/15/92  00
  :0507 DR2R   RF2 DRVR REV POWER KW    00 .. .... 09  309    0      0      0.291        01/15/92  00
  :0508 AD2CK8 RF2 A/D ZERO CHK 8  V    00 .. .... 00  10     0      0      0            01/15/92  00
  :0509 AD2CK9 RF2 A/D ZERO CHK 9  V    00 .. .... 00  10     0      0      0            01/15/92  00
  :050A LL2F   RF2 LL FWD POWER   W     00 .. .... 09  18.12  0      0      0.245        01/15/92  00
  :050B LL2R   RF2 LL REV POWER   W     00 .. .... 09  23.53  0      0      0.337        01/15/92  00
  :050C IPA12F RF2 IPA1 FWD PWR   W     00 .. .... 09  2144   0      0      0.307        01/15/92  00
  :050D IPA12R RF2 IPA1 REV PWR   W     00 .. .... 09  1517   0      0      0.174        01/15/92  00
  :050E IPA22F RF2 IPA2 FWD PWR   KW    00 .. .... 09  19.5   0      0      0.233        01/15/92  00
  :050F IPA22R RF2 IPA2 REV PWR   KW    00 .. .... 09  20.28  0      0      0.253        01/15/92  00
  :0510 PH2ADJ RF2 PHASE ADJUST   DEG   02 A3 B013 00  400    0      2500   0            01/30/92  00
  :0511 TU2POS RF2  LOW    HGH    IN    00 .. .... 00  4      0      0      0            02/11/92  12 050A,09   00,00
0500,00
  :0512 RF2LML RF2 LOSS MON LOWER V     00 .. .... 00  10     0      0      0            01/15/92  00
  :0513 RF2LMU RF2 LOSS MON UPPER V     00 .. .... 00  10     0      0      0            01/15/92  00
  :0514 TO2IN  T2 TOROID IN       MA    00 .. .... 04  100    0      0      0            01/15/92  00
  :0515 TO2OUT T2 TOROID OUT      MA    00 .. .... 04  100    0      0      0            01/15/92  00
  :0516 TK2IPL TK2 VIPPS2 LEE     TOR   00 .. .... 00  10     0      0      0            01/15/92  00
  :0517 TK2IPH TK2 VIPPS2 HEE     TOR   00 .. .... 00  10     0      0      0            01/15/92  00
  :0518 MD2IV  RF2 MOD INPUT VLTS V     00 .. .... 00  10     0      0      0            01/15/92  00
  :0519 MD2OV  RF2 MOD OUTPUT VLT KV    00 .. .... 00  100    0      0      0            01/15/92  00
  :051A MD2OI  RF2 MOD OUTPUT CUR A     00 .. .... 00  1000   0      0      0            01/15/92  00
  :051B DR2SV  RF2 DR SCREEN VOLT V     00 .. .... 00  4000   0      0      0            01/15/92  00
  :051C RF2HV  RF2 HI VLTS  ONOFF KV    00 .. .... 00  100    0      0      0            02/10/92  11 0516,--   86,--
0597,--
  :051D RF2PAI RF2 7835 FIL. CURR A     00 .. .... 00  10000  0      0      0            01/15/92  00
  :051E DR2PAV RF2 4616 FIL. VOLT V     00 .. .... 00  10     0      0      0            01/15/92  00
  :0520 QPS201 T2 QUAD PS1    RST A     13 A3 0200 00 -312.5  0      312.5  0            02/11/92  11 0518,--   84,--
0594,--
  :0521 QPS202 T2 QUAD PS #2      A     13 A3 0201 00 -312.5  0      312.5  0            01/30/92  00
  :0522 QPS203 T2 QUAD PS #3      A     13 A3 0202 00 -312.5  0      312.5  0            01/30/92  00
  :0523 QPS204 T2 QUAD PS #4      A     13 A3 0203 00 -312.5  0      312.5  0            01/30/92  00
  :0524 QPS205 T2 QUAD PS #5      A     13 A3 0204 00 -312.5  0      312.5  0            01/30/92  00
  :0525 QPS206 T2 QUAD PS #6      A     13 A3 0205 00 -312.5  0      312.5  0            01/30/92  00
  :0526 QPS207 T2 QUAD PS #7      A     13 A3 0206 00 -312.5  0      312.5  0            01/30/92  00
  :0527 QPS208 T2 QUAD PS #8      A     13 A3 0207 00 -312.5  0      312.5  0            01/30/92  00
```

Analog data

```
 NODE=0611  ANALOG DESCRIPT 02/25/92 1255
 CHAN#  NAME   TITLE       ..... UNIT  READNG SETTNG NOMINL TOLRNC  AFLG TRIP
  :0500 IN2PHS T2-1 INTERTKMANAUT V     -0.0012        0      0.08   3400    0
```

```
:0501 GR2LO  RF2 PICKUP LOOP #1 NRM   -0.002         0.981  1      3400    0
:0502 GR2MID RF2 GRDIENT MANAUT NRM    0.0041        0.9789 0.0499 3400    0
:0503 GR2HI  RF2 PICKUP LOOP #3 NRM    0.0021        0.983  1      3400    0
:0504 PA2F   RF2 PA FWD     CBR MW     0.0124        2.1655 0      0400    0
:0505 PA2R   RF2 PA REV POWER   MW     0.0062        0.1028 0.1649 1400    0
:0506 DR2F   RF2 DRVR FWD POWER KW     0.3297        174.96 12.097 3400    0
:0507 DR2R   RF2 DRVR REV POWER KW     0.1226        0.9524 0.7921 0400    0
:0508 AD2CK8 RF2 A/D ZERO CHK 8  V     0.0055        0      0      0000    0
:0509 AD2CK9 RF2 A/D ZERO CHK 9  V     0.0043        0      0      0000    0
:050A LL2F   RF2 LL FWD POWER   W      0.01          4.5648 0.9998 0400    0
:050B LL2R   RF2 LL REV POWER   W      0.0072       -0.0043 0.1996 0400    0
:050C IPA12F RF2 IPA1 FWD PWR   W      0.9814        351.36 24.929 0400    0
:050D IPA12R RF2 IPA1 REV PWR   W      0.6944        2.5462 2.4999 0400    0
:050E IPA22F RF2 IPA2 FWD PWR   KW     0.0065        4.0639 0.4166 1400    0
:050F IPA22R RF2 IPA2 REV PWR   KW     0.0043        0.1232 0.1999 0400    0
:0510 PH2ADJ RF2 PHASE ADJUST   DEG    255.86        254.88 5.0293 1400    0
:0511 TU2POS RF2  LOW    HGH    IN     1.297         1.963  0      0400    0
:0512 RF2LML RF2 LOSS MON LOWER V     -9.9994        0      0      0000    0
:0513 RF2LMU RF2 LOSS MON UPPER V     -9.9991        0      0      0000    0
:0514 TO2IN  T2 TOROID IN       MA     0      2.1118 0      0      0000    0
:0515 TO2OUT T2 TOROID OUT      MA     0     -0.0336 0      0      0000    0
:0516 TK2IPL TK2 VIPPS2 LEE     TOR   -3.1671        0      0      0000    0
:0517 TK2IPH TK2 VIPPS2 HEE     TOR   -9.9979        0      0      0000    0
:0518 MD2IV  RF2 MOD INPUT VLTS V      0.0024        2.8296 0      0400    0
:0519 MD2OV  RF2 MOD OUTPUT VLT KV     0.0275        20.145 0      0400    0
:051A MD2OI  RF2 MOD OUTPUT CUR A      0.3052        171.2  0      0400    0
:051B DR2SV  RF2 DR SCREEN VOLT V      3.2959        507.08 24.902 0400    0
:051C RF2HV  RF2 HI VLTS  ONOFF KV     0.0366        0      0      0400    0
:051D RF2PAI RF2 7835 FIL. CURR A      6742.2        6747.1 50.659 0400    0
:051E DR2PAV RF2 4616 FIL. VOLT V      0.8441        0      0      0000    0
```

Command file

The source code files are:

SYSTEM.SA	System routines glue
EDAD.SA	Pascal Analog Descriptor application
CVGN.SA	Numeric conversion—floating pt to ascii
UPPERCAS.SA	Adjust text string to upper case

There are about 1000 lines of source in EDAD.SA. The total size of the application is about 9K bytes.

Associated Status of a Channel

Alarm info addition

Aug 31, 1989

One can request associated status of an analog channel to obtain up to 3 bits of digital status that is related to the channel. This status is intimately tied in with the job of updating bits of text on a parameter page line. This note describes an enhancement to the status information that is returned to include alarm information as well. In this way, a host program can decide whether to display the status in green or red, for example, based upon whether the status bit is in alarm.

Each status bit, independent of its possible association with an analog channel, can optionally be in the alarm scan. If a bit is in the alarm scan, it can have a good/bad attribute. (When it is not in the alarm scan, it has no good/bad-ness, for the purpose of this discussion.) A bit in the alarm scan can also optionally "inhibit beam" when it is in the bad state. (Inhibit beam here refers to the assertion of a control line external to the local station whenever any channel or bit with that option selected is in alarm.)

The current use of listype #5, using a channel-type ident, provides the status information for up to 3 associated bits in a single byte. The bits used begin with the most-significant bit and work down from there. Bit#7 refers to the status for the first associated bit, bit#6 is used for the second, and bit#5 is used for the third. With the recent use of expanded Bit numbers (more than 8 bits wide), only 1 or 2 associated status bits are supported. Further details on the database entries that support this feature can be found in the document "Digital Pulse Delays."

In order to provide alarm information in addition to the status data, one merely requests more than one byte of data using listype #5. The additional data will be supplied in the second and following bytes. The bits used in the byte are the same ones used for the status byte.

The complete information available requires a data request of 4 bytes. The meaning of the hi order bits in the bytes are:

0: status (same as always)
1: active bit (1= Bit is in the alarm scan)
2: good/bad bit (1= Bit is bad *and* in alarm scan)
3: inhibit bit (1= Bit is inhibiting beam *and* bad *and* in alarm scan)

Binary Descriptor Page

Local Station Application

Oct 12, 1989

Introduction

The Binary Descriptor Page provides for general binary bit and byte database and I/O manipulation. One can enter the 16-character descriptions of each bit, the alarm flags for each bit, set or clear individual bits, set binary bytes, and monitor the current state of bits and associated trip counts. It can also output a listing of a range of bits in a node.

Display layout

```
 0   B BINARY DESCRIPT 06/22/88 1214
 1   ___BIT#<07:000>-<:3FF>    _LIST      node/bit, range
 2   000  00 00 00 00 00 00 00 00         32 raw data bytes
 3   040  00 00 00 00 00 00 00 00
 4   080  00 00 00 00 00 00 00 00
 5   0C0  00 00 00 00 00 00 00 00  #T
 6   007<TITLE OF BIT    >0 <10010> 0     bit titles,
 7   006<TITLE OF BIT    >0 <10010> 0    bit data,
 8   005<TITLE OF BIT    >0 <10010> 0    alarm flags,
 9   004<TITLE OF BIT    >0 <10010> 0    trip count
10   003<TITLE OF BIT    >0 <10010> 0
11   002<TITLE OF BIT    >0 <10010> 0
12   001<TITLE OF BIT    >0 <10010> 0
13   000<TITLE OF BIT    >0 <10010> 0
14   *ZERO TRIPS  ACT,NOM,INH,2X,BEAM     clear trips
15
```

Displayed data

There are several areas of interest. Select a node and bit# on line 1 and interrupt to select a byte that contains that bit and show the text for all bits in that byte. The ending bit# gives a range of bits that are used for generating a listing (by interrupting under `LIST`) and causes a wrap when using the raise/lower buttons to advance the selected byte. The 3 hex digit bit# allows for 4096 bits in a system, which is 512 bytes. The three characters just before the `BIT#` prompt are used to display the data request status code for the three relevant requests that provide the data for the page. The character just before the word `LIST` is used for a blinking "*" while a listing is active. Interrupt again to terminate a listing.

The four lines starting at line 2 show 32 bytes of binary data. The first bit# for the data on that line is shown at the left. Enter a new bit# at the start of line 2 to change the set of 32 bytes displayed, or use the raise/lower buttons with the cursor on any of these 4 lines to move

through successive 32-byte chunks of binary data. The range given in line 1 determines when this logic wraps. If the current byte is in the range of 32 bytes shown, the current byte value is shown in inverse video.

The 16-character bit titles are shown on lines 6–13. The bit# is at the start of the line. To the right of the text is the present bit value. To the right of the bit value is the alarm flags for that bit, where the significance of the bits displayed are listed on line 14. They are:

ACT	Active (=1)
NOM	Nominal state
INH	Inhibit beam if bad (=1)
2X	Require two times for change of state (=1)
BEAM	Only scan for alarms on beam cycles (Bit $9F = 0)

The last value shown on a beam title line is the trip count. If the trip count for a bit is > 99, the count overwrites the “>” symbol. In any case, the largest value is 255, as it is a byte-size value.

Control actions

Interrupt under the LIST word to start / stop a listing of the range of bits indicated on line 1. The format produced is as follows:

```
BINARY DESCRIPT 03/29/89 1417
 BIT#<20:000>-<:2FF>
 BIT TITLE            V  ANI2B #T    BIT TITLE            V  ANI2B #T
 007<SPARE           >1 <00000> 0    00F<STANDBY         >0 <00000> 0
 006<INTERLOCK FAULT >0 <00000> 0    00E<TIMING          >0 <00000> 0
 005<AMP TEMP FAULT  >1 <00000> 0    00D<AMP AC ON       >1 <00000> 0
 004<AMP LOAD FAULT  >0 <00000> 0    00C<VAC OK          >1 <00000> 0
 003<ALL ON          >1 <00000> 0    00B<FANS OK         >1 <11000>82
 002<AMP BIAS ON     >1 <11000255    00A<TEMP ON         >1 <00000> 0
 001<LLRF ON         >1 <00000> 0    009<DOORS OK        >1 <00000> 0
 000<AMP DC ON       >1 <00000> 0    008<REMOTE          >1 <10000>68
```

Enter a new binary byte value over the displayed value on lines 2–5 and interrupt to cause that *byte* to be set.

Enter a new title and interrupt to set the new title for that bit.

Change the alarm flags and interrupt to establish new alarm flags.

Interrupt on line 14 to cause a clear trips command to be set to the current node whose binary data is displayed on the page.

Bit control

Interrupt in the area where the bit value is displayed to control the bit in several ways:

1. Interrupt with the cursor under the bit value (0 or 1) to toggle the bit value to the other state.

2. Type a new bit value (0 or 1) and interrupt with the cursor just after the character to set the new value.

3. Type a new value (2–9,A–Z, which implies 2–35) to pulse the bit hi for that number of 15 Hz cycles.

4. Type a period (“ . ”) to issue a short (20 µ sec) pulse.

Clock Events Page

Diagnostic events display

Wed, Aug 24, 1994

Introduction

Using the new Tevatron clock event detection hardware on the digital IP board that is part of the Internet Rack Monitor (IRM), it is useful to view the occurrence of clock events in real time. Software support for clock event inter rupts maintains event diagnostic data that can be displayed. This page shows most of that diagnostic data in addition to an on-line picture of clock event activity.

Display layout

```
7 CLOCK EVENTS    08/24/94 1415
NODE<0561> E<18> N= 1383 T=44       Events on captured cycle:
               DELTA=   .066679
..2...6........F.1......8.A.....    02, 06, 0F, 11, 18, 1A
.....5.........................     25
............................
.........................A.C...     7A, 7C

.1.3.............2.............     81, 83, 92
............................
............................
............................
NEVTS=     16110    213 EVTS/SEC
EVTS-HIST:          0 15006   273
      59    37     1    38     0
```

(The picture shown was captured at a random time to illustrate the page.)

Enter the node# on the second line and interrupt to change the display. (It is also activated with the previous node used when the page is first invoked.) On the same display row, the E field is the current selected clock event# of interest. For each event interrupt, when the detected event matches that event#, the IRM software counts the event and captures the time (in ms) that it occurred relative to the cyclic activity of that IRM. These data are displayed on the remainder of that line. On the next line is displayed the delta time in seconds between the last two selected events.

Two areas of four 32-character lines exhibit the clock event activity. The first area shows clock events 00–1F, 20–3F, 40–5F, and 60–7F on successive lines. The least digit of the clock event# is shown whenever that clock event occurs during the previous 15 Hz cycle. These data are updated on the screen at 15 Hz. Events 80–9F, A0–BF, C0-DF, E0-FF above are shown on consecutive lines in the second area.

The statistics toward the bottom of the page show the total number of all events, the number of events detected over the previous second, and the event distribution.

The event distribution shows the number of events detected in the hardware FIFO during each event interrupts. The first three bins (0, 1, 2) are shown on one line, with 3, 4, 5, 6, ≥ 7, shown on the other line. In the example shown, it is clear that almost all event interrupts result in only one event present in the 64-deep FIFO. Two events are found in the FIFO about 1.7% of the time.

Additional details

All statistics are displayed relative to the statistics that were in place when the first reply was received to the data request of that kind of memory data. The total number of events, for example, is the number of events that occurred since the time the data request was sent. A new data request is sent when the node# or selected event# field is changed, or when the page is entered. This feature allows multiple users to view the clock event information without interference. The only resource that is shared is the selected event#. When another user changes the selected event#, it affects the selected-event results obtained by all users. Changing the selected event# on this page, however, restarts the viewed selected event statistics.

Note that if this page is viewed via a "Page G" facility, such as the "VME Screen Image" option of the Macintosh Parameter Page, the data rate and screen update rate should be 15 Hz, or some event activity data may not be seen.

This page application is called EVTS and is written in 680 lines of Pascal that compile into less than 4K bytes. Execution time during 15 Hz updates as measured on two different local station/IRMs is as follows:

CPU board	*CPU*	*Clock,MHz*	*Maximum time/cycle, ms*
MVME-133A	68020	20	2.5
MVME-162	68040	25	1.0

CRT Image Page

Local Station application

Dec 28, 1989

Overview

Local station user interaction occurs via the local console. Each 15 Hz cycle, the console data is polled for any changes. The raw console input data consists of 16 switch inputs from the pushbuttons, 14 lamp outputs as pushbutton indicators, an 8-bit keyboard character code, and an 8-bit knob counter. A 4800 baud serial interface is used to access these data under interrupt control. The Console Task processes the inputs for the system. Some of the pushbuttons are handled with mutually exclusive logic. An application usually keys on the *light* activity for actions it makes in response to pushbutton activity. Note that a station with no local console attached can still run application pages, as the display interface resides on the Crate Utility board. Only the composite video and the serial send and receive cables are attached to the local console hardware.

Display layout

```
G CRT IMAGE        10/31/89 1336
SYSTEM<  >
```

The system's node# whose console is to be monitored is entered in hex. Upon interrupting, the display changes to the current local console display appearance of the system under study. Interrupt again to return to the original page.

Basic approach

This particular application presumes to know a lot about the system—more than most application pages. This is in part because it was written without any special hooks provided by the system to support its implementation. Some of the features supported are fortuitous.

Console-related input data are only 4 bytes and are stored in the system global variables. Two are for the switch input data, one is for the keyboard, and one is for the knob. The switch input data is automatically processed to turn on related lights behind the switches. These two light bytes can be read also.

The basic plan is to request memory data for the display image buffer in the remote station. This buffer is 512 bytes of ascii for all the characters on the screen. (It isn't the real video buffer because that uses a 6-bit ascii encoding. The reason for having an image buffer is to support "reading" from the display.) The image buffer is requested to be returned at 15 Hz. Any characters which have been changed are then updated on the local display.

Additional data requested along with the display image buffer is the remote cursor position, the remote lights, and the remote knob change. The program logic proceeds as follows:

The local keyboard input character is checked. Whenever a new character has been typed locally, it is sent to the remote system, which will in turn respond to it just as if the character were typed on the remote console (assuming it has one). A special case is made for activating the keyboard interrupt on the remote station. If a control-Q is typed locally, then an ESC character is sent to the remote station, which is interpreted the same as a keyboard interrupt. This must be done since a keyboard interrupt used locally will produce local actions, such as returning to the local display or exiting the page. (If that is done inadvertently, merely recall the CRT Image page, and the remote screen will re-appear just as it was.)

If the location of the remote cursor changes, the local cursor is moved to keep pace with it. This allows the local user to monitor a remote user's typing and cursor motions.

If the remote lights change, the local lights are set to match them. This allows the local console lights to indicate what a user at the remote console is doing. Of course, such light changes at the remote console can arise from the actions of the application that is being run by the local user as well.

If the local lights are changed, the remote lights are set to match them. This allows the local pushbuttons to work to control the remote application.

If the local knob is adjusted, the same count of knob clicks is passed to the remote station. This allows the local knob to work.

To summarize:
The monitoring of the remote station's display image buffer and cursor location allows monitoring remote console activity.

Local typing, except for ESC, is sent to be acted upon remotely. Local switch changes that result in lights being lit are passed to the remote station. Local knob action is passed to the remote station. The combination of these actions allows control of a remote application.

Currently the cursor control (etch-a-sketch) buttons do not work on the remote station because the local knob counter is continuously read into the global copy.

D0 Data Requests Test Page

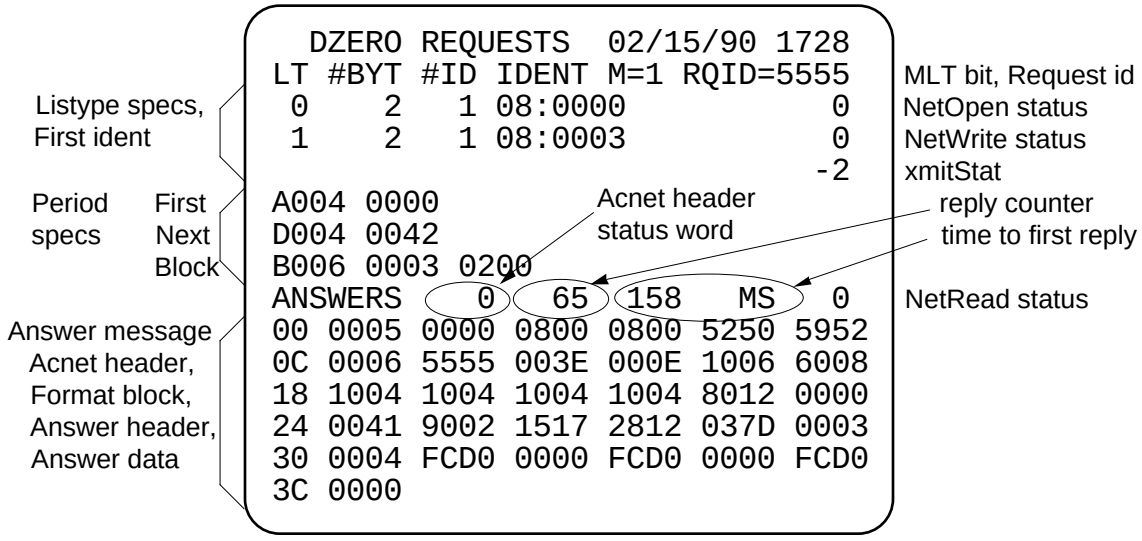
Local Station Application

Mar 21, 1990

Introduction

A new protocol for data request network messages was designed for use with the D0 detector's CDAQ (Controls Data Acquisition) software. The VME local stations support this message format for data requests. This application page exercises the new protocol by issuing data requests to a local station and displaying the responses that are returned. It is written using the local station's Network Layer interface routines.

Display page layout



This snapshot of the test program display serves as an example of several of the features it provides. It shows an example of requesting a 2-byte reading from channel 0 in node 08 and a 2-byte setting from channel 3. The period specification indicates that the data is to be collected the first time right away and at intervals of 15 Hz (66 msec) thereafter, accumulating (blocking) 3 sets of data (with a 512 msec timeout) before returning a response.

Parameter entry

The `M` bit is set for periodic requests. The request remains active until a cancel message is sent. If the `M` bit is zero, the request is automatically cancelled after update of the first set of reply data. The `RQID` is the request id that serves to uniquely identify one request from another.

Enter up to three listype specifications each with an associated starting ident. The listype#, the #bytes requested/ident, and the #idents are specified in decimal. Enter the starting ident in hex. The ident length is determined by number of

digits typed. When the #idents > 1, the additional idents are added sequentially

using a step size of 1. (For memory addresses, the step size is 2.) To omit a listype spec, blank out the listype# field.

The period specifications occupy three lines for the First, Next, and Block specs and are entered in hex. (See Al Jonckheere's document entitled "D0 Control Data Acquisition Network Data Transmission Protocol" for a complete presentation of the entire protocol including the period specification parameters.) To omit a period spec, blank out the first word of the line. The other words will remain, but the spec will not be used.

Interrupt anywhere in the parameter specification area of the screen (on rows 2–8) to initiate the data request. The word ANSWERS in the response area of the screen should be hi-lited to indicate the request is active. (If it is a one-shot request, or if an error is returned, it may not stay hi-lited for long.) When the first (or only) response is received, the elapsed time is displayed on the ANSWERS line in milliseconds. This time is measured from just *before* the call to `NetWrite` until just *after* the call to `NetRead` that returns the first response in the test program.

Just to the left of the elapsed time to first response is the count of replies received. The Acnet header status word is also shown in decimal just after the word ANSWERS. This is done for convenience in interpreting error codes, which tend to be negative numbers.

Other status replies that are shown are toward the right end of the screen. The status return from NetOpen, which is called upon entry to the page, is given at the end of the third line. Below that is the return from the call to NetWrite when the request message was issued. Below that is the transmit status word that gives the success of the network transmission. At the end of the ANSWERS line is the status return from the call to NetRead which returns the reply data.

Cancel an active request by an interrupt on the ANSWERS line. The hi-liting will be removed as the cancel message is sent (a USM with bit#9 set in the Acnet header flags word).

Answer data viewing

Six lines are used to display answer data in hex with 6 words per line. The byte value at the left shows the offset in bytes of the first word on the line. There are three ways to adjust the starting offset for the block of answers.

The easiest way is to use the raise/lower buttons on the local console. It will adjust the offset by 72 bytes (6 lines of 6 words each) at a time. If you advance too

far, it wraps to the start.

To adjust the offset so that the first word is one of the displayed data words, merely interrupt under the word you want to be the starting word displayed. Obviously, you can only move ahead in this way.

The third way to adjust the offset is by typing in the desired offset in the first characters of the first line of answer data and interrupting. Three characters can be entered, even though only the least significant two characters can be displayed due to the screen's limited number of characters per line.

The entire response message is displayed, beginning with the 9-word Acnet header. (For an error response, this is all you get.) This is followed by the format block, beginning with the size word of that block. That is followed by the 9-word answer message header. The remainder of the response message is the answer data itself.

Details of the example response

The first 9 words are the Acnet header. The first word shows that it is a reply message with the `MLT` bit set. The next word is the reply status word, and it is also shown in decimal on the line above. The destination and source bode of the request are both node 08 in this case. This example illustrates use of the network to make a request to itself. (If it didn't do that, it would always require two nodes to do the test.) A by-product of this is that the xmitStat value shown at the end of the fifth line is -2, indicating "address not recognized," which in this case is normal.

The next two words of the Acnet are the destination task name, which for D0 data requests is `RPYR`. The source task id is 6, which denotes the table index in the Network Connect Table returned by the call to `NetOpen`. The request id is followed by the message size word, which is the total size of the entire response message including the Acnet header itself.

The format block length of 14 bytes including the length word shows six format specs. The first three describe the format of the answer message header and are always present. They indicate 3 words, one 8-byte time, and 2 words. The last three format specs describe the format of the three sets of answers as given by the blocking specification. Each indicates the two words (a reading and a setting) that were requested in this example.

The answer message header starts with the `8012` word. The status word is zero. The only nonzero status here would be 4, indicating a bus error, possible when accessing arbitrary memory locations. All other errors are detected during request initialization and return error responses as status-only replies, consisting

of only the Acnet header.

The next word is the sequence counter. Note that its value matches the reply counter shown in decimal above, which is good. It is followed by the BCD time field. Note that this matches the time of day that is shown on the top line of the display. (The seventh byte of this 8-byte field is a cycle count, also in BCD, and the last byte is a residual half-millisecond count.) The time is followed by the number of data sets (=3 from the blocking specs) included in this response and the size of each set (=4 bytes in this case).

At long last we get to the answer data. Each of the three sets shows the same values of the reading and setting words. As to why the reading and setting don't match in this case, remember that node 08 is a test station, in which almost anything is possible.

The time response value for this example shows 158 msec, which may seem long. But remember that the example specified that 3 sets of 15 Hz data were to be accumulated before returning a response. The first set will be collected right away as soon as the request is processed, and the third set will be collected two cycles later. Since this station actually runs at 12.5 Hz in the absence of 15 Hz interrupt input signal, two more cycles would be 160 msec. The request was issued by the application program in the same node, and it runs soon after the data collection and update time for that node. (The 66 msec value on the period specs is rounded to one 80 msec cycle.) So that's why. A more typical simple example without blocking would result in an elapsed time about 5–6 msec for the first response. Interestingly, this matches the analogous response time that can be measured for the "classic" data request protocol used by the Local Stations.

D0 Data Settings Test Page

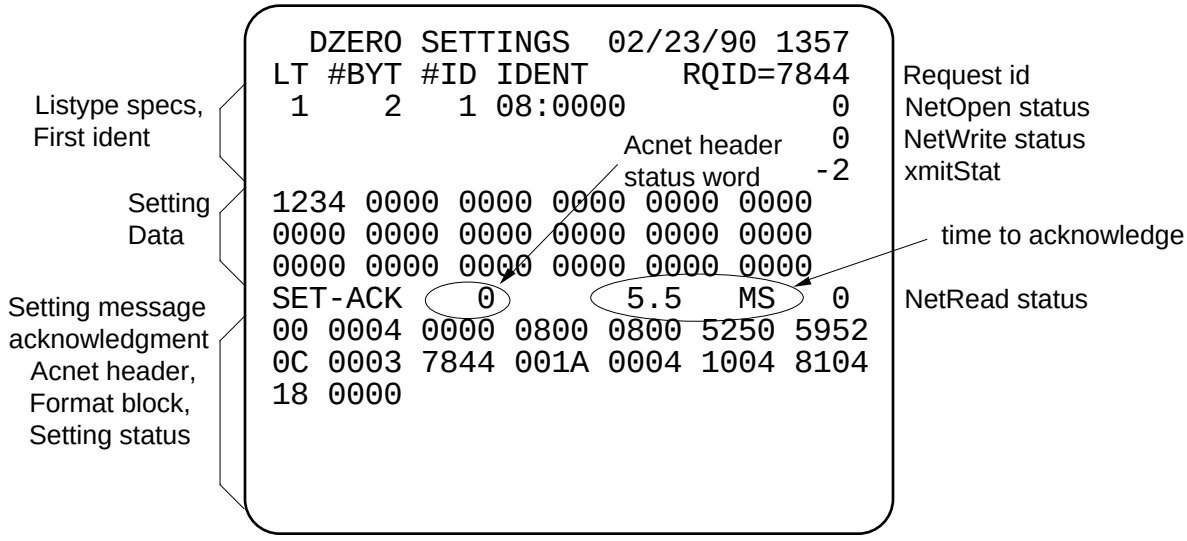
Local Station Application

Feb 23, 1990

Introduction

A new protocol for data setting network messages was designed for use with the D0 detector's CDAQ (Controls Data Acquisition) software. The VME local stations support this message format for data settings. This application page exercises the new protocol by issuing data setting messages to a local station and displaying the response that is returned. It is written using the local station's Network Layer interface routines. This document and test program itself borrows heavily on the document entitled "D0 Data Requests Test Page."

Display page layout



This snapshot of the test program display shows an example of sending a 2-byte data word setting value of $1234 to channel 0 in node 08.

Parameter entry

The RQID is the request id that serves to uniquely identify the setting acknowledgment with the setting message that was issued.

Enter up to three listype specifications each with an associated starting ident. The listype#, the #dataBytes/ident, and the #idents are specified in decimal. Enter the starting ident in hex. The ident length is determined by number of digits typed. When the #idents > 1, the additional idents are added sequentially using a step size of 1. (For memory addresses, the step size is 2.) To omit a listype spec, blank out the listype# field.

The setting data is entered in the three line field that allows for up to 36 bytes.

amount of the data needed is determined by the listype specs. (To send only one byte, remember that the byte order is strictly left-to-right in 68K systems.)

Interrupt anywhere in the parameter specification area of the screen (on rows 2–8) to initiate the data setting. The word SET-ACK in the response area of the screen is hi-lited to indicate the setting is active. (If the response time is quick enough, it may not be hi-lited long enough to be seen on the 60 Hz refresh display screen.) When the response is received, the elapsed time is displayed on the SET-ACK line in milliseconds. This time is measured from just *before* the call to NetWrite until just *after* the call to NetRead that returns the response.

The Acnet header status word is shown to the left of the elapsed time in decimal just after the word SET-ACK. This is done for convenience in interpreting the Acnet header error status codes, which tend to be negative numbers.

Other status replies that are shown are toward the right end of the screen. The status return from NetOpen, which is called upon entry to the page, is given at the end of the third line. Below that is the return from the call to NetWrite when the request message was issued. Below that is the transmit status word that gives the success of the network transmission. At the end of the SET-ACK line is the status return from the call to NetRead which returns the acknowledgment.

An interrupt on the SET-ACK line merely turns off the hi-liting.

Response data viewing

Six lines are used to display answer data in hex with 6 words per line. The byte value at the left shows the offset in bytes of the first word on the line. There are two ways to adjust the starting offset for the block of answers. (This feature was borrowed from the data request test page and is overkill for the short setting acknowledgments.)

To adjust the offset so that the first word is one of the displayed data words, merely interrupt under the word you want to be the starting word displayed. Obviously, you can only move ahead in this way. The other way to adjust the offset is by typing in the desired offset in the first characters of the first line of answer data and interrupting.

The entire response message is displayed, beginning with the 9-word Acnet header. This is followed by the 2-word format block, beginning with the size word of that block. That is followed by the 2-word acknowledgment message whose second word is the status code.

For cases in which an error is detected before actually invoking the setting

Details of the example setting response

The first 9 words are the Acnet header. The first word shows that it is a reply message. The next word is the reply status word, and it is also shown in decimal on the line above. The destination and source bode of the request are both node 08 in this case. This example illustrates use of the network to make a setting to itself. (If it didn't do that, it would always require two nodes to do the test.) A by-product of this is that the xmitStat value shown at the end of the fifth line shows the value -2, indicating "address not recognized," which in the case of sending a message to the same node is normal.

The next two words of the Acnet header are the destination task name, which for D0 data requests / settings is `RPYR`. The source task id is 3, which denotes the table index in the Network Connect Table returned by the call to `NetOpen`. The request id is followed by the message size word, which is the total size of the entire response message including the Acnet header itself.

The format block length of 4 bytes (including the length word) shows a single format spec which describes the format of the response message header word and the response status word as two 16-bit integers.

The header is the `8104` word. The status word is zero, indicating no error. The nonzero status values currently range from 1–55 decimal. All other errors are detected during request initialization and return error responses as status-only replies, consisting of only the Acnet header.

The time response value for this example shows 5.5 msec, which is typical for setting acknowledgments when the target node is not busy at the time the setting message arrives.

Downloading Programs

On a first-name basis

Aug 28, 1990

Previous application management

Application page programs heretofore have been allocated manually in the VME local stations. One found some free space in non-volatile memory and downloaded the code for an application page into it. Then the starting address was entered on the index page to associate a page with the program.

Difficulties with that scheme arose when the number of downloaded programs got to be large. (Once upon a time, there were but four.) It was hard to manage the memory available efficiently. One tended to select starting addresses on 4K-byte boundaries just to keep it simpler, resulting in fragmentation.

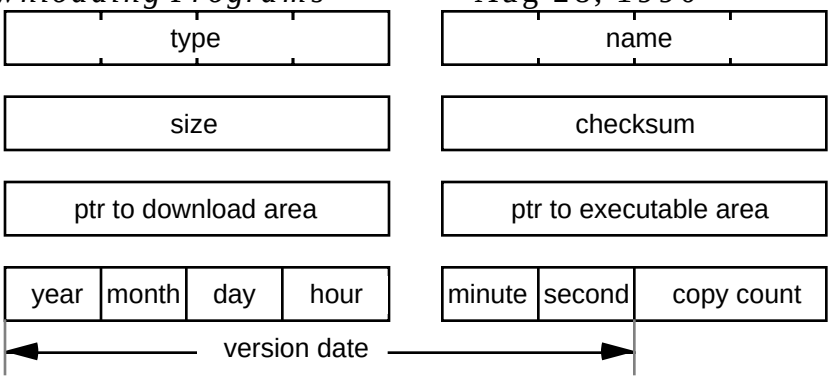
Another difficulty arose when downloading to another system. One must take care that a previous version of the program was not currently executing on that system. If it were, a crash would likely result as the new code comes raining down upon the executing code. This crash is not easily recoverable, since the system will try to recall the same page after it aborts. After 15 times, the system aborts more thoroughly by exiting to SBug, requiring local manual intervention at that station to recover.

A third problem with the previous scheme was that updating new versions of code required copying to the various local stations one at a time. Only strict adherence to conventions can insure that the program is downloaded in the same area of memory on separate nodes. (There is no system requirement for such.) Different nodes may have different needs for a repertoire of page applications.

A fourth problem was lack of checking to insure that a program residing in non-volatile memory was not corrupted by an errant happenstance in a local station.

Named-based programs

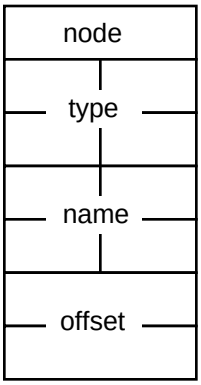
By supporting communications by program name, we insulate the user from having to deal with manual memory allocation and at the same time, we can be free of concern for locating a given program at the same address in multiple nodes. A new system table called `CODES` is used to maintain information about each downloaded program. Its layout is as follows:



The 4-character program type distinguishes programs used for different purposes. Values such as PAGE, LOOP, TASK can be used to distinguish page application programs, local application programs and task initialization programs. The 4-character program name serves to distinguish programs of the same type. The program size is next, followed by the longword checksum, which is a sum of words. The download ptr is a ptr to the allocated area in non-volatile memory. This is followed by the ptr to the area in on-board ram where the program was copied for execution. The date serves to identify the version of the program. The copy count shows the number of times the program was copied into on-board ram for execution. It is cleared when a new version is downloaded.

New listype to support downloading

Support for name-based downloading requires a new listype (#76). The ident format is as follows:



Both the type and name codes are included to fully identify the program being referenced. The offset is a longword to permit downloading of large programs of the future, when 16-megabyte ram chips are commonplace. Special offset values are used for beginning and ending the download process.

To download a program, an offset value of –1 is used to send the type, name and size information to the receiving node. If the receiving node has the program in

successfully, although a currently-executing version can still be running, since it runs in on-board memory, which is separate from the download area. The size is used to allocate memory to receive the downloaded program, and that download ptr is placed in the entry.

Settings are delivered with offset values ≥ 0 (but not +1) to fill the download area with the program. After that is finished, which may require multiple settings, a special setting is made with the offset value of +1. The data sent with that setting is the checksum longword and optionally the version date. The next time the page is called up, in the case of a PAGE-type program, the new program will be checked for corruption using the checksum longword and copied into on-board ram for execution. In the case of a LOOP-type program, the next time the closed loop is enabled, the new version will be used. If such a program is enabled during the download operation (and executing out of on-board ram), it will automatically be disabled and automatically re-enabled on the next cycle, resulting in automatic and orderly transition to the new version.

A data request using this listype deals with the same data as the setting does. Notice that no program header is required, as all the size and name information is carried in the CODES table entry. The data returned for an offset=–1 are the entire contents of the CODES table entry. Offset values ≥ 0 return arbitrary portions of the program code itself.

CODES Table Management

An enhanced version of the download application page allows inspection of the various CODES table entries. One can disable a downloaded program from further invocations and free its memory, much as one would delete a file in a disk operating system to get more room for storing new files.

To maintain the free space in one contiguous piece, the system moves allocated blocks around at system reset time to eliminate fragmentation before any programs are put to use. It has no side effects, since only the CODES table knows where they are. Any other references to these programs are by name.

Entering New Parameters on a Local Station

Parameter page lore

Mar 16, 1989

There are several options available for entering new channels to be displayed on a Local Station's parameter page. This note attempts to itemize them.

There are 14 lines available on the parameter page display. The line after the page title is called the "first" line. It has no line immediately preceding it. When entering any text on the screen, the program does not "watch" the characters being typed. It only "reads" from the screen when the user presses the keyboard interrupt button (or ESC key). So it doesn't know what the user typed; it only knows what is displayed on the screen. (It also cannot tell which characters are displayed in inverse video.)

Analog device name

If one types a name of up to 6 characters starting in column 0 (the left-most column) and presses the interrupt button, the parameter page program will try to translate the name into a channel number. It first searches its own local database for a match on the name; if it's not found, it sends the name out to all network nodes. Each node searches its own database for a match. A node which finds a match (there should only be one) returns its CHAN ident (including its node number). If no match is found, the name is unknown and the line is blanked. If there is a match, the CHAN is accepted for that line, and a new request for database information and data is issued to rebuild the page. The name that is entered must be at least two characters in length, and it must not include a colon character. The program will use the cursor position to limit the characters that it accepts as user input. Hence, one should not move the cursor after entering the name and before pressing the interrupt.

Analog channel ident

The channel ident is represented in ASCII as `NN:CCC`, where `NN` is the hex node number and `CCC` is the hex channel number within that node. If the user types this form at the start of a line, it will be interpreted as a CHAN ident. This case is distinguished from the name case just described by the presence of the colon separator.

There are several variations of this form supported by the parameter page. If a node number is typed alone (including the colon) on an active line (one which currently shows a channel), the new channel ident is formed from the node number typed and the same channel number currently shown. In other words, the same channel in another node is selected. But that's not all. If there are lines *after* this line on the page which are displaying the same node number as is *presently* displayed on this line, then each of *those* lines will be

similarly adjusted to refer to the new node number typed. As a degenerate case of this, suppose all 14 lines are displaying a set of channels which are all from the same node. Entering a new node number alone on the first line will cause all the lines to switch to the new node but show the same channel numbers as before.

As a throwback to earlier systems, a single character typed in column 0 will be assumed to select a new node number also. In this case, the letters A–Z map into nodes 0A–23. In order to be recognized, the character typed must not be the same as the first character of any name displayed on that line. Users are advised to try to forget this older method.

If the user enters a channel number alone without a node part, the local node is the node which is selected. In this case, one must enter a colon followed by the channel number to prevent interpretation as a name. This is especially convenient for a local test station in which there may not even be a network connection, or at least no interest in other network stations.

Clear a line

To clear a line to blanks that is active, merely interrupt with the cursor in column 0 of the line. The line will be blanked and no data will be displayed on that line. The program does not, however, revise its data request at this point. It will continue to receive the data that was displayed until an addition or change is made to the channels displayed on the page. A by-product of this detail is illustrated by the case when a line is displaying the only channel on the page that is from a node which is not supplying data to the request. (Perhaps it does not exist or is not open onto the network.) If one interrupts at the start of that line, there will still be the error indication (8) displayed reflecting the fact that some data is “missing.”

To clear a set of successive lines, enter a name that is all blanks. (Two blanks are sufficient.) Then that line *and all following lines* will be blanked.

Sequence of channels

This case is similar the the one just described for clearing a line, but with rather different effect. If the user interrupts in column 0 of a *blank* line which is preceded by an active line, The CHAN accepted is the next channel number in the same node as that of the preceding active line. In addition, the cursor is advance to the following line; thus, by interrupting at the start of a blank line which is followed by more blank lines, one gets a sequence of channels entered all from the same node as the active line just before the first blank line. Also, since an interrupt at the start of an active line blanks that line

(without advancing the cursor to the next line), one can enter a sequence of channels *through* a set of lines which include active lines. It just takes an extra interrupt to get rid of each active line. What's more, one can take advantage of the repeat action feature of the keyboard and merely press the ESC key continuously to achieve this effect.

Sequence of nodes

One can also enter a sequence of nodes using the same channel number. In this case, an interrupt is made with the cursor in column 1 (the second character position) of a blank line that is immediately preceded by an active line. The CHAN entered is then the same channel number as the preceding active line combined with the next node number (node+1) of the one on the preceding line. In addition, the cursor is advanced to column 1 of the following line. This allows the user to enter a sequence of nodes all with the same channel number on a sequence of blank lines.

If the user interrupts in column 1 of an *active* line, that same active line is changed to select the same channel number in the next node. Successive interrupts step through successive nodes on that same line, as the cursor is not moved.

Saving a channel selection

All the changes described above are *temporary* changes. In order to make the present set of channels displayed into the *permanent* set (the ones which will be used the next time that page is invoked), the user merely types an X in the home position (use the home key on the keyboard) and interrupts with the cursor just after the X. (It would seem that one should get page X by following this procedure, but not in this special case.) The parameter page merely refreshes itself, and does not exit the page. The next time the same page gets called up from another page or the index page, this newly appointed set of channels will be displayed immediately.

Local Application Parameters

Page application

Dec 12, 1991

Local applications are a method of adding new features to the local station software without being linked in with the system code. Each entry in the Local Applications Table LATBL contains the context for an invocation of a local application. This can permit a single LA to be invoked multiple times using different parameter values. It is analogous to calling a procedure with different arguments passed. This page application allows viewing and modify ing the parameters of LATBL entries in any local station.

Page layout

An example of the LA used for rf conditioning is as follows:

```
E LOC APPL PARAMS 12/12/91 1228
NODE<062A>  NTRY< 6>
NAME=KRMP   CNTR=0FE8
TITL"RF CAVITY CONDITIONING  "
SVAR=000473F8
ENABLE  B<0530> KLY RF RAMP ENBL
SPARKS  B<01A8> WG REFL ENERGY
RFONST  B<01A5> LLRF DISABLED
RFINTLK B<010C> A0 INTERLK RESET
PPWR    C<01F0> KAWG1P      MW
VACUUM  C<008E> KAV1        V
RADTION C<0006> SACVRD      R/H
RFRESET B<0086> RF RESET
EVENTS  B<0180> CLK EVT 18
OTHERS  C<01F1> KATRGP      MW
```

Enter the target node and LATBL entry# (range 0–31) and interrupt. That entry is displayed and presented for editing. If the CNTR word is counting, the LA is currently active and is being called at 15 Hz. The name of the LA , a title that describes its function, and the current ptr to the static variables of that LA used by the entry are shown. The remainder of the display shows up to 10 words used as parameters.

Each parameter line shows some prompt text which is a reminder of its use, where a B or C suffix means that the parameter is actually a Bit# or Chan#, respectively. In either of these two cases, the Bit text or the Chan name / units are shown to the right to verify the parameter word's significance.

Text database

As each new local application is written, another portion of a "text file" should be added. The text file is actually a named "program" called HELPLOOP. It is prepared with the MPW assembler and downloaded in the usual way. A key is

used at the start of each program's text info to allow variable length entries. Each entry has a fairly rigid structure organized into 8-character units. An example from the portion used for the KRMP example is as follows:

```
DC   'L***KRMP','RF CAVITY CONDITIONING  '
DC                          'ENABLE B','SPARKS B'
DC   'RFONST B','RFINTLKB','PPWR   C','VACUUM C'
DC   'RADTIONC','RFRESETB','EVENTS B','OTHERS C'
```

The first 4 characters are the key used to identify the presence of the 4-character name of the LA. After the name is a 24-character LA title to denote the LA's function. Then each parameter uses an 8-character prompt text, which denotes the parameter's function. If the 8th character is a B or a C, then the parameter is either a Bit# or a Chan#; otherwise, all 8 characters can mean the prompt text. (The layout of the parameters above corresponds to the usual appearance of an LATBL entry as viewed on a memory display page.) Unused parameters on the end need not be included.

The text file is first looked for in the local station which is running this page application. If it is not found there, it looks up the copy in node 0576. It searches for a match on the name and displays the prompt text for each parameter.

Making changes

Type in new values for the parameter words and interrupt on each one. The display will be updated to reflect any changed Chan or Bit text. type in a new name and interrupt, and the title and parameter prompt text will be changed accordingly. Type a new node# and/or entry# to move to a new LATBL entry. The raise/lower buttons can be used to adjust the entry#.

Program stats

The LAPP page application is about 800 lines of Pascal running in 5K bytes.

Local Console Alarm Display

Alarms on bottom line

R. Goodwin

Aug 9, 1989

Local Consoles can optionally (via option switch on Crate Utility Board) display alarm messages on the bottom line of the small screen display. Since there is only a single line available for this purpose—shared by the Small Memory Dump feature—there are complications which arise due to the fact that the message must be displayed long enough to be noticed by a human. This note describes the current scheme for handling such display messages.

Option switches

Three option switches relate to local alarms encoding and display.

4: Enable *analog* alarms to be included for local display.
3: Encode and output alarm message to serial port.
2: Encode and display alarm message on bottom line of screen.

For no local alarms encoding, switches #2 and #3 should be off. When switch #4 is off, only binary and comment alarms are encoded for local handling. For messages which are written to the serial port, the time-of-day associated with the alarm message is included. (The small screen does not have room to include the time-of-day, but it does indicate a “>” mark to show that the single message shown is fresh.)

Alarm message queuing

All messages which are destined to be sent to the network pass through the Output Pointer Queue, or `OUTPQ`. This includes data requests, settings, answers, and alarm messages generated by the Local Station. It can also include alarm messages generated by another station if those alarm messages are received by the Local Station. In that case, they are merely passed to the `OUTPQ` but marked (by setting the “used” bit in the queue entry) as having already been sent to the network so they aren’t sent again.

The QMonitor Task monitors all entries placed in the `OUTPQ`. It first monitors the entries to see that they have been transmitted to the network. If they take too long, they time out, in order that the queue traffic is not stalled. The time-out period is about 1 second.

After the entries have been delivered to the network, QMonitor scans the entries to check for alarm messages that should be locally displayed. If the message is an alarm message that should be displayed on the small screen, the message is encoded and passed to `ADspQEnt`. This routine could be declared as follows:

```
Function ADspQEnt(row,col: Integer;
VAR dspText: Chars32;
nChars, nCycles: Integer): Integer;
```

The row and col parameters indicate the position on the screen for the displayed

message. The dspText is the array of characters to be displayed. The nChars is the number of characters in the text. The nCycles is the delay in cycles to allow time for the message to be read before it is permitted to be overwritten by a new message. For alarm messages, the following constant values are used:

row= 15
col= 0
nChars= 32
nCycles= 30

The function value is an error status code with these possible values:

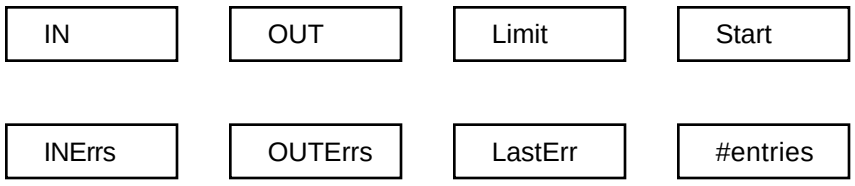
0: No errors.
1: Cannot allocate ADSPQ.
2: nChars zero or negative.
3: Cannot allocate text message block.
4: Bad ADSPQ header.
5: ADSPQ full. Cannot accept message.

The other call that QMonitor makes every cycle is to ADspQMon, which monitors the Alarm Display Queue to update the screen as needed and show the queued messages on the screen for the time designated. It has no arguments and no function return, although it can report errors through the ADSPQ header as follows:

6: Invalid OUT ptr.
7: Unexpected memory block type.

Structure of ADSPQ

The queue is created by ADspQEnt the first time it is called, and a system global variable is set to point to it. It currently has room for 124 entries. At 2 seconds per message displayed (30 cycles @ 15 Hz), it would take 4 minutes to empty a queue that was full. The queue header has the following structure:



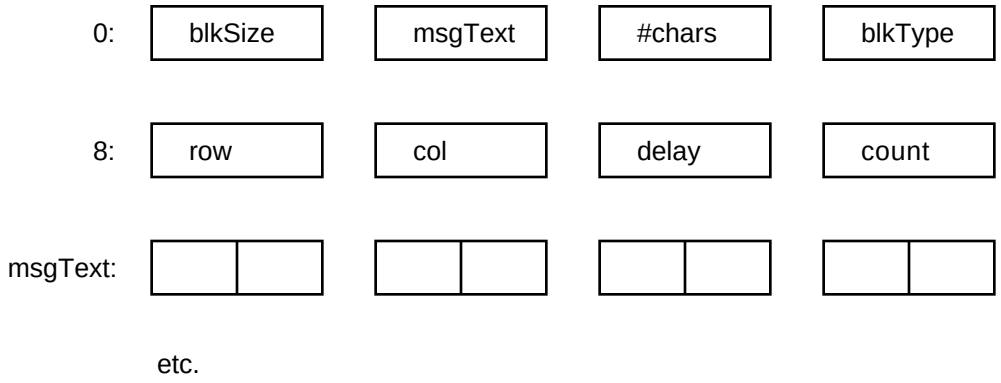
When the queue is initialized, IN and OUT and START are all set equal to the size of the header (16 bytes). LIMIT is set to the size of the queue, which is the size of the header plus the product of the number of entries (124) and the size of each entry (4 bytes). With these values, the size of the queue is 512 bytes. The other 4 words in the header are cleared.

ADspQEnt places a new entry into the queue and advances the queue's IN ptr, an offset to the next entry to be used. ADspQMon processes and removes entries from the queue as required and advances the OUT ptr. When the OUT ptr reaches LIMIT, it is

reset to the START value. When it reaches equality with IN, the queue is empty.

Diagnostics are included which count errors detected by both ADspQEnt and ADspQMon. The last error code is also recorded as is the total number of entries placed into ADSPQ.

The entry that is placed into ADSPQ is a pointer to a text message block whose format is as follows:



The first word is the allocated block size, the second is the offset to the message text, the third is the length of the text, and the fourth is the block type (=10). The next three words denote the position for the text on the screen and the delay in cycles allowed for viewing. The next word is used to count down the delay. The text to be displayed completes the contents of the block.

ADspQMon processing

Each entry placed into the ADSPQ by ADspQEnt points to a memory block that it allocated. As each block is processed by ADspQMon and displayed for the indicated delay time, the block is freed. If the option switch is turned off during the display of messages, the current message will time out its delay, and no more messages will appear. Any waiting queue entries will be skipped and the associated blocks freed.

When a new message is about to be displayed, the first two characters from the screen are read to see if the display area is available. The convention is that the first two characters should be blanks to declare this. If they are not blank, the new message is discarded and the block freed. This allows the use of the Small Memory Dump when alarm messages on the bottom line are enabled without concern that a new message will overwrite the memory data that is displayed. When the first character of the address whose memory is to be displayed is typed, no new alarms will be written there. When the feature is disabled and the line is blanked, by interrupt at the start of the line, new alarms will again be allowed to appear.

After a new message has timed out its delay, two blanks are written at the start of the display area to indicate that the message shown is no longer “fresh.” Just before the blanks are written, a check is made that the first two characters are still the same characters that were originally written there; if they are not, the blanks are not written. This again allows the user to “take over” and use the memory dump feature. Current

new messages are marked with "> " in the first two characters to indicate a "fresh" message.

Code modularization

In the modern spirit of OOP, "information hiding" principles have been followed in the implementation of this alarms display handling. The layout of `ADSPQ` and the layout of the associated message blocks are known only within the new module `ADSPQMON` which includes both routines described herein. QMonitor only knows to pass the messages it wants displayed to `ADspQEnt` and that it must call `ADspQMon` every cycle.

Serial alarms output

When QMonitor senses that a serial port version of an alarm message is to be produced, it only has to call the `PrintLn` routine with the line of text to be output. Since the serial characters are spooled to memory, there is no delay in processing the serial output.

Memory Dump Page

Local Station Application

R. Goodwin

Nov 13, 1991

Function

This application is a venerable workhorse of Local Station diagnostics, maintenance and debugging. It displays 64 bytes of memory from either the local node or any set of nodes, updated at 15 Hz. This makes it possible to actually see what is happening in memory and thereby give a more complete understanding of the internal operation of system software. It allows setting memory words or bytes by keyboard entry. It can transfer memory blocks of any length from any location in any node to any other loca tion in any node. It can also fill a block of memory with any longword pattern, and it can dump a block of memory in standard S-record ascii format to any node's serial port.

Display layout and tour

```
M MEMORY DUMP      11/12/91 0814
0576:100000 0400 0010 0014 0000
0576:100008 0400 0040 0015 0000
0576:100010 0800 0004 0019 0000
0576:100018 0800 0010 0019 8000
0508:000788 9111 1208 1449 1120
0751:000788 9111 1208 1449 117F
0752:000788 9111 1208 1449 110D
0753:000788 9111 1208 1449 1114
ACCESS BY BYTES

MW 0576:00120000  0508:00120000
                        0000F000
MB       00000000  0576:FFFFA205
CACHE ON                00000001
```

This snapshot shows an example of the Memory Dump page application display. It shows 32 bytes of memory from the beginning of node 0508's system table directory, followed by 8 bytes of memory data from each of 4 nodes showing the current time-of-day. (Note that the time-of-day is well synchronized, as it is multicast on the network every minute by node 0516, which has access to a clock that is itself synchronized by reception of NBS satellite signals.)

Address entry

Up to 8 different addresses can be entered on the page. When an address is entered, that line and all of the following lines are automatically set to show consecutive memory data. So set different node/address combinations on different lines from the top down.

Bytes/words access mode

The data bytes presented on the page are accessed by bytes, which means that `MOVE.B` instructions are executed in the relevant contributing nodes to collect the data.

Another option is access by words, which causes `MOVE.W` instructions to be used instead. Normally one wouldn't expect there to be any difference, but some hardware boards are designed to be accessed in only one way. A keyboard interrupt in the first 18 columns of this line toggles between access by bytes and access by words.

Memory settings

One can enter new data to be stored at the memory locations displayed. The size of the setting can be one or two bytes, or it can be one word. To enter a word, type the new value over the data value that is currently displayed at that address. (The display will not overwrite while the cursor remains in the field.) With the cursor still positioned immediately to the right of the value just typed, press the keyboard interrupt button. Expect to see the new value displayed in place of the old one. The setting of the new value word will be set as two bytes if the display is using byte access and as a word if the display is using word access.

To make a single byte memory setting, type the two digits of the desired setting in place of the current byte value and use two spaces to cover up the other byte of the word on the display. Interrupt just after the word and only the single byte value will be set.

There is only the ability to set bytes and words manually. Other amounts of data can be set using the features described next. As an example, a longword setting could be set using the fill option, although the actual setting to the hardware would take place accessed by either two words or 4 bytes.

Copy, fill, and dump

The next four lines allow for copying memory blocks, filling them with constant data, or dumping them to a serial port. The first one is set to copy 60K bytes of memory words ("`MW`") from address $00120000 in node 0576 to the same address in node 0508. (This would copy the local station system program from node 0576 to node 0508.) The second one is set to copy a single 00 byte to the crate utility board 68901 chip data direction register. This results in disabling the heartbeat trigger, forcing the station to be reset by the watchdog timeout circuit on that board. The difference in specifying the fill option rather than the copy option is denoted by the absence of the colon that must be present in the first `nnnn:address` field. A keyboard interrupt action with the cursor in the area of either pair of lines will cause the indicated operation to be performed.

Other options available on these lines are `DB` and `DW`, which cause bytes or words of data from a memory block to be encoded into standard Motorola S-record ascii format and spooled to a serial port. To select the dump option, interrupt with the cursor in the first column, under the "`M`". Another interrupt switches back to the move/fill option. Byte or word access for these operations is similarly toggled by interrupt in the second column. The dump option provides a way of archiving memory data on the disk of a pc or host computer without using the network. Suggested future enhancements to these options might be to implement a find function and a verify function.

Cache enable

The last line displays the fact that the 68020 instruction cache is enabled in the local

station. A keyboard interrupt on the left of this line toggles this enable.

Pointers

When examining software data structures, it is sometimes necessary to follow a pointer chain, or linked list. A facility for doing this is included on the Memory Dump page. An interrupt with the cursor located at either of the two middle hex digits of the upper word of a memory address that is currently displayed (as two words in the data area) causes the contiguous region displayed (usually 64 bytes) which includes that address to be replaced by the same size region starting at the address indicated. *It's much easier to do than to say.* Additional interrupts under addresses displayed can follow a linked chain of data structures. Each time this happens, the address (and node) is saved in an internal table. One can navigate through this table by interrupts with the cursor toward the right end of the line immediately following the 8 data lines, where an address will be shown. Interrupt under the left half of the address to back up in the table; interrupt in the right half to move forward through the saved addresses. If one backs up all the way to the start, the displayed address vanishes. Up to 100 addresses are saved internally, all of which is lost when leaving the page.

Freeze display

When the display is being updated at 15 Hz, it is sometimes difficult to catch values that change often, or to correlate them with other changing values. To assist in viewing a snapshot of the memory displayed, press the Hex button on the Local Console. The contents of the screen will be frozen as of that moment, and the "NO UPDATE" message will be displayed. Release the Hex button to resume normal updating. Note that while the display is frozen, the data is still being collected; it is simply not being updated on the display. Another way to make snapshots is via the serial port printing option below.

Volts units

If the data words being displayed are A/D readings, they can be shown in voltage units by selecting the Volts button on the Local Console. This assumes that the scaling used is ± 10 volts full scale. The value $8000 corresponds to –10 volts, and the value $7FFF represents +10 volts. Zero volts is $0000.

Entering addresses

To enter addresses in the memory data part of the display, there are several conventions in use. Typing an address alone at the start of a line, overwriting the node number, causes only the address part to be changed. But take care to terminate the address with a space, so that additional digits starting at the current cursor position are not accepted as part of the new address. If one merely changes the node digits, leaving the address alone, then only the node will be changed to match what has been modified. If both node and address are entered, both parts will be accepted.

The small local displays have only 32 characters per line, so there is not enough room to show a complete node#, a 32-bit address, and 4 data words with enough spacing to make it readable. Since it is common to use addresses within the 24-bit range, or to use node#s in which the hi byte is always the same, there is a toggle between two options of displaying the node:addresses. To alternate between the two options, interrupt in the first column of the second row—under the first character of the first node#.

If the interrupt is given with the cursor in the first column, then the address accepted is the address from the previous line but with the node number incremented by one. If there is no previous line, or if the cursor is on the last line, the current address is not changed; only the node number is incremented. And if the cursor is not on the last line, then the cursor is moved to the start of the next line. This makes it easy to look at the same 8 bytes of memory in successive nodes.

The raise and lower push buttons can be used to move forward and backward through memory. When the raise button is pressed, the address on the current line is advanced by the number of bytes of data that is displayed on that line plus the following lines, and the addresses are forced to be contiguous memory in the same node as on the cursor line. This allows one to scan through a region of memory displaying up to 64 bytes at a time. If the cursor is on a line *other* than one of the eight data lines, memory addresses are advanced 64 bytes at a time as if the cursor were on the first data line.

Smart Rack Monitor (SRM) access

Local stations connect to SRMs via Arcnet. A special range of node#s is reserved for this purpose. Arcnet nodes use node#s in the range 7AA0–7ABF, allowing for a maximum of 32 SRMs on a single Arcnet network. New Linac control stations will typically have 3–4 SRMs accessible via Arcnet. Although the memory request protocol is different for SRMs than for local stations, special support is built into the memory dump page to permit viewing Arcnet-connected SRM memory in a natural way.

Small Memory Dump

This is a system function and is therefore available independent of what application page is running. Still, it affords one more line of 8 bytes that is available for display using the bottom line of the small screen display. (The easiest way to find it is to press the Home key and then the Up arrow key.) Enter an address at the start of the line; the node number is not required, as this line only displays local memory in bytes. If a bus error is found, the displayed data bytes are displayed in inverse video. Local memory settings can be made in the same way as is done via the Memory Dump page. The raise

and lower push buttons can be used to move through local memory 8 bytes at a time while the cursor is on the bottom line.

Printing to the serial port

This is also a system function, but it bears including here. Any application page display can be written to the local serial port by a keyboard interrupt under the page title on the top line (actually the 3rd through the 18th column). Sometimes it's useful to save copies of the memory display for later perusal.

Error codes

In response to data requests, there can be an error code returned. The Memory Dump page displays this code on the top line in the second column (between the current page number and the page title) if it nonzero. Only a few codes are commonly seen. An **8** code means that some node whose data is requested in the list of addresses on the display is not returning any data in response to the request. This may mean that it's down or is off-line (not on the network), or the local station operating the memory dump page is off-line. A **7** code means the same thing, except that data was received from that node at least once since the request was last issued. This code appears often when nodes are not running in synchronization and is generally ignored. The other common code that may appear is a **4** code. It is described next.

Bus errors

The system has no means of predicting whether a given address from which data is requested will produce a bus error when it is accessed. When the access is made by the system code in processing the data request, it treads carefully. If a bus error is encountered while accessing the memory requested (by bytes or by words as selected) an error **4** is returned with the data in response to the request.

Memory copy block sizes

When moving memory blocks around, it is done with 1024-byte packets at 15 Hz. This means that 60K bytes of memory—which happens to be about the size of the system code—can be transferred in about 4 seconds. In the case of memory transfers to Arcnet-based SRMs, the maximum packet size is 480 bytes, in conformance with the smaller maximum fame size available with Arcnet.

SRM task names

Normal SRM communications are addressed to the network destination task name SRMD. But to download a new version of the SRM system code, the task name NEWV must be used. The memory dump page program checks for the special case of a new version download by the range of memory used in the memory copy. If it is in the range used for the system code, then it uses the NEWV task name; otherwise, it uses SRMD. After the transfer is complete, in the case that NEWV was used, a reset message is sent using the task name REST. This resets the SRM and causes it to begin execution of the new version.

Access to data from other nodes

Every data request or setting request includes a node number along with the memory address being referenced. The application program pays very little attention to whether the node number used is the local node or not. The system code that is invoked to make a data request does whatever is required to accomplish the task. If network communication is required, then it is used; otherwise, it is not. The job of writing application programs would likely be much more difficult if this were not the case. Again, to the user, this makes no difference at all. She needs only to specify the nodes she is interested in.

Command file

These modules are required to build the MDMP program:

SYSTEM TRAP calls to system library routines
MDMP Main Pascal object module
CACR Library routine to access 68020 Cache Control Register

The program size is about 10K bytes.

Memory Plot Page

Diagnostic application

Dec 28, 1989

Introduction

This application page was developed for use with the Graphic Strategies, Inc., color graphics interface board. It plots memory data words in histogram fashion. The data words can be consecutive or separated by any constant spacing. The data that is plotted can be located in any node, as the data request services of the system software are used to collect the data for plotting.

Display layout

```
0         P MEMORY PLOT     12/22/89 1047
1         AUTO-PLOT  #CY=1
2                    INC=2
3         N=64       MAX=$7FFF
4         C=4        MIN=$8000
5         TXTC=1             *SAVE AUTO
6
```

Interrupt on lines 1–5 to start plotting data. Interrupt under AUTO-PLOT to stop the plot (or exit the page). While plotting is active, AUTO-PLOT is shown in inverse video.

The #CY parameter is the number of *cycles* of the period of the data request and must be in the range 1–250. The INC parameter gives the *incremental* separation for successive data words in bytes. Its value must be even. Use the value 2 to specify contiguous data words. Using a negative value allows plotting data in reverse order.

The N parameter specifies the *number* of data words to plot. They are given equal spacing along the horizontal axis on the plot. The fewer the data words, the wider the slots. The value of N can range from 1–512.

The C parameter is the *color* used for the data plot. The TXTC parameter is the color to use for the *text* that appears on the plot. The color codes are 3-bit values that turn on the Red-Green-Blue color guns. (The value 6 should be yellow, representing red+green. The value 1 is blue.) Text in the upper right of the plot gives the selected plotting parameters and the current data max and min values.

The MAX and MIN values give the scaling (in hexadecimal) for the data values that are plotted. The values shown are appropriate for mapping the entire range of data values onto the 512x512 graphics display. When the value zero lies between MAX and MIN, a blue line is shown on the display to mark its vertical location.

Network Frames Page

Built-in network diagnostics

Mar 24, 1992

Whilst debugging network communications, it can be helpful to see what network frames a local station has recently processed. The token ring chipset DMA's frames into a large (currently 64K bytes) circular network receive buffer. Each frame takes only the space needed for its size. Similarly, token ring frames are assembled in another circular frame buffer. A 16-byte record is written into data stream #0 about each frame processed by the system software. Enable this feature by creating DSTRM table entry #0 for a data stream called NETFRAME and resetting. (Nearly all local stations have this data stream installed already.) The system support for data streams makes this information available to a requester. The Network Frames page, usually installed as page F, provides a user interface to access the data from this data stream.

```
0  F NETWORK FRAMES  03/24/92 1358
1  NODE<0601> #RCVD= 237 LIST<0000>
2  NODE=0000 - SIZE=0000 TIME=0000
3  0624 000C R 06E86E 1357:50-05+33
4  0616 000C R 06E890 1357:50-06+ 5
5  0623 000C R 06E8B2 1357:50-06+15
6  0610 0016 R 06E8D4 1357:50-06+18
7  0611 000C R 06E900 1357:50-06+19
8  0610 000C R 06E922 1357:50-06+21
9  061E 01F8 R 06E944 1357:50-06+23
10 0617 000C R 06EB52 1357:50-06+24
11 061E 000C R 06EB74 1357:50-06+24
12 0614 000C R 06EB96 1357:50-06+25
13 0624 000C R 06EBB8 1357:50-06+32
14 0053 02B8 T 08AD42 1357:50-06+41
```

Specify the node from which information about recent network frames is to be collected and interrupt on row 1 to request and capture the data. The number of frame records collected is shown in the RCVD field. The filter specs on row 2 restrict the resulting listing by sending / target node and / or frame contents size and / or time received. One may use the operators = (equal), ! (not equal), > (greater than or equal) and < (less than) to specify the filtering logic. Disable a filter by using the value 0000. One may also enter a T or R in place of the – to restrict the list to Transmitted or Received frames only. The resulting listing, scrollable by pages, starts on row 3.

Interrupt on row 2 to regenerate the listing (and reset the scroll offset) based upon any changed filtering options. If the field after the LIST prompt on row 1 is nonzero, the entire listing is also written to the serial port of that node.

The list shown includes the node that sent the received frame (or the target node of the transmitted frame), the size of the frame, the T/R indicator, the address of the frame in the circular buffer, and the time-of-day the frame was processed. The time-of-day is specified as `HrMn:Sc-Cy+ms` to include the 15 Hz cycle (range 00–14) and the milliseconds into that cycle. Note that this marks the time that the frame was processed at the task level, not the time of the network interrupt. For a received frame, it is the time that the frame is being processed by the task that was made active in response to the receive interrupt. For a transmitted frame, it is the time that the frame is assembled in the circular buffer and being passed to the chipset for transmission. Thus, the elapsed time between a request frame and a reply frame can be seen as the software turn-around time.

In the case of the example shown above, node 0601, running its parameter page application locally, had requested some 15 Hz single channel readings and settings from 9 different nodes. The part of the list shown shows the frames from each node received during cycle 06 of time 1357:50 one week after St Patrick's Day, 1992. In each case, the size of the frame was 000C bytes, which is correct for a Classic protocol reply of a reading word and a setting word. The earliest contributing node was 0616, and the latest was 0624. There is an additional frame coming from node 0610 and also a larger one from 061E. At the bottom of the part of the list shown, there is a transmitted reply to node 0053 that is sent at 41 msec into the cycle, which correlates with the fact that the Server Task is scheduled to run at 40 msec into the cycle; hence, this is likely a server reply to an accelerator (RETDAT) request. With the rate of frame processing shown, the 237 frames covers only about 2 seconds of time on that sunny spring afternoon.

Interrupt on a listing line to change the listing area to show the frame contents itself à la scrollable memory dump. Interrupt again to switch back to redisplay the frame list. This is successful only if the circular buffer, either for received or transmitted frames, has not "wrapped" since the data was collected.

For Acnet header frames, the destination and source node words and the destination task name words are byte-swapped when viewed as received frames. They appear in network order when viewed as transmitted frames.

Network Layer

Ping Pong test program

Aug 2, 1989

The Network Layer software supports task-to-task communication across the network. A set of four Pascal-callable interface routines are provided that invoke the Network Layer routines to provide network service to application programs in the VME Local Stations. The PingPong application program is a demonstration of the use of these routines.

Upon program initialization PingPong connects to the network using the two names PING and PONG. The PING "task" accepts operator input to send a request to PONG. The PONG "task" will respond to any requests that are sent to it. The same application is simultaneously a PING task and a PONG task. It can be loaded on two different nodes for the purpose. This allows exercising sending requests and receiving replies between two nodes. One can be PING and the other will be PONG. Or, they can be both simultaneously.

In order to provide for some example "work" to be done by the replying task, the request is interpreted as a request for memory data that should be returned in response to the request. Both the number of bytes and the starting memory address are specified in the last three words of the request message. With the 9-word Acnet header, the size of the request message is therefore 24 bytes. The size of the response message to such a request is 20 plus the number of bytes requested, since the first word of the response (following the Acnet header) is a status word with the value 'OK' or 'BE' as an indication of whether a bus error was encountered when accessing the requested memory. The format of the display showing the PING activity is as follows:

```
J NET PING PONG   08/01/89 1057
PING   OPEN     0   *CLOSE
      *WRITE    0    0   N<9000>
   0002 0000 7304 0804 504F4E47
   0000 1234 0018 0104 00102008
       READ     0 T=  14 N=9000
   0004 0000 7304 0804 504F4E47
   0003 1234 0118 "OK" 8908 0109
PONG   OPEN     0   *CLOSE
       READ              N=   0


       WRITE             N=   0
```

The upper part of the display shows the PING message traffic. Note the fields of the Acnet header. The 0002 signifies a request, the destination node is $73, the destination task name is 'PONG', the message id is $1234, and the message size is 24 bytes. The source node and the source task id for a request message

are filled in by the Network Layer software. The three additional words of the request message following the header specify that PONG should reply with 260 bytes of memory data beginning at address $102008. This request was executed 9000 times.

The next section of the display shows the response that was received from the PONG task in node $73. The Acnet header is mostly identical in the response message, indicated by the first word value of 0004. The replier had to specify the message size and a status word, but the other fields are left the same as were received in the request. Note that the source task id (actually the destination task id for a reply) has the value $03. That value was also part of the received request and serves to route the reply back to the requesting task, PING in this case. The time for the response is shown as 14 counts in units of 0.5 msec; hence, it means 7 msec. This time is measured from just before PING's call to NetWrite until just after PING's call to NetRead in response to the application's invocation due to the Network event that results from the arrival of the response message. There were 9000 responses received.

A example of the display from the perspective of PONG is as follows:

```
J NET PING PONG   08/01/89 1125
PING   OPEN     0   *CLOSE
      *WRITE              N<9000>
   0002 0000 7304 0804 504F4E47
   0000 1234 0018 0104 00102008
       READ       T=     N=   0


PONG   OPEN     0   *CLOSE
       READ     0         N= 535
   0002 0000 0804 7304 504F4E47
   0005 5678 0018 0004 0010200C
       WRITE    0         N= 535
   0004 0000 0804 7304 504F4E47
   0005 5678 0018 "OK" 25310024
```

The lower part of the display shows the PONG activity. (The upper part merely shows the example request message last used that is saved across invocations of the page.) Note the fields of the Acnet header sent by PING from node $73 this time. The value $5678 was used for the message id in this case and the source task id $05 was filled in by node 73's Network Layer software. Four bytes of data were requested from location $10200C in node 08. The reply message sent by PONG is also shown, The status word preceding the four returned memory data bytes is displayed here in Ascii and indicates that there was no bus error accessing memory. A total of 535 requests were replied to.

The two lines of the display which show the NetOpen status return allow

invoking NetClose to test the status return from that call. One normally won't do this, as the tasks will no longer be connected to the network. Leaving the page also closes the network connection for both PING and PONG.

Overview of Network processing

It may be helpful to understand some of what is going on behind the scenes while PingPong is "doing its thing." PING sends a request message by making a call to NetWrite, specifying the message that it wants to send, including the Acnet header in the first 9 words. It also provides a variable which will be set later to indicate the success of the transmission to the network. NetWrite actually allocates a dynamic memory "message block" to house the some control information used by the network. A pointer to the message is put into the message block, and a pointer to the block is passed to NetQueue, which in turn invokes OUTPQX to place the pointer onto the Output Pointer Queue OUTPQ. NetWrite then calls NetSend (which calls NetXmit) to build the network frame in a circular frame buffer and pass it to the token ring chipset. At this point NetWrite returns to the user application.

Meanwhile, the chipset uses DMA to transfer the frame buffer into its own high speed memory which is able to keep up with the 4 Mbps token ring bandwidth. When it obtains the token from the ring, it transmits the frame. When the frame has circulated around the ring, the transmitting chipset strips it from the ring and emits a new token. At this point the success of the transmission is known, and the chipset generates a transmit interrupt. The network transmit interrupt routine (in module NetInt) records a status code in the user's variable that was passed earlier via the call to NetWrite. The way PING is written, no particular notice of this value is made except at the usual 15 Hz invocations of the application, when the screen is updated with the current value of the variable if it has changed.

On the PONG side of the equation, the arrival of the network message to the chipset results in a DMA transfer into a circular frame receive buffer and an interrupt being delivered to the system. The receive interrupt routine uses the destination SAP to obtain a message queue id from the NETCT table of connected SAPs. In this case, it sends a frame reference message containing a pointer to the frame contents to the ANet message queue. Writing a message into this queue wakes up the ANet Task.

The ANet task analyzes the Acnet header and looks up the destination task name in the NETCT Table. It sends a message reference to the associated message queue (whose name is PONG in this case). Another field in the NETCT entry indicates that it should also send event #4 to the application task to signal it that a network event has occurred. (This was arranged automatically by PONG's NetOpen call.) When the application task is invoked with the

Network event, it calls NetRead (as both PING and PONG, since the Network event may signal the arrival of either a request or a response or both). In this case, a message is received by PONG's call to NetRead, and the request, including the Acnet header, is copied into the user's buffer.

PONG interprets the request and calls its own MemData routine to collect the requested memory data into a reply buffer. The Acnet header is copied to the start of that same buffer, the first word is set to denote a reply message, and the last word of the header is set to indicate the total message size. NetWrite is called to deliver the response. As before, NetWrite allocates a message block and puts a pointer to the user's message into it. Then NetQueue and NetSend are called in turn to "get it out the door." NetSend builds the network frame in the circular frame transmit buffer and hands it off to the chipset. The chipset DMA's the frame into its own fast memory and transmits the frame to the token ring.

Back on the PING side, the response frame is received, and the receive interrupt passes a reference to it to the ANet task, which in turn passes a reference to the message to the application and signals the application task via event #4. The application is invoked and finds a message for PING. The call to NetRead results in the response message being copied from the frame buffer into the user's buffer. The cycle is complete. If the count is not yet exhausted, then NetWrite is called to send the request message again to PONG.

The Network Layer implementation, as is seen from the above discussion, does copy messages in memory. The received data is copied from the circular receive frame buffer into the user's buffer. The transmitted data is copied from the user's buffer into the circular transmit frame buffer.

Statistics have been collected on the performance of the Ping Pong test vehicle. They are listed in the following table:

Cache off

#bytes	Requester Tx to Rx	Requester Rx to Tx	Requester Appl Prog	Replier Tx delay	Rep rate	Frames/sec	KBytes/sec
44.5	2.04.3	1.36.5	154	1			
256	5.02.1	4.32.0	7.2	139	36		
512	5.52.1	4.43.5	7.7	130	66		
1024	8.42.2	4.65.0	10.6	94	97		
2048	14.02.5	5.09.0	16.6	60	123		
3072	19.52.8	5.213.0	22.5	44	137		
4096	25.53.0	5.416.5	28.5	35	144		

Cache on

#bytes	Requester Tx to Rx	Requester Rx to Tx	Requester Appl Prog	Replier Tx delay	Rep rate	Frames/sec	KBytes/sec
42.7	2.03.2	1.34.8	208	1			
256	3.62.0	3.22.0	5.7	175	45		
512	5.02.0	3.23.5	7.0	143	73		
1024	7.42.2	3.35.0	9.6	104	107		
2048	12.02.5	3.69.0	14.5	69	141		
3072	17.03.7	3.713.0	19.5	51	158		
4096	21.53.0	4.016.5	24.5	41	167		

The reference to “Cache on” refers to the 68020 instruction cache in both nodes. The “Tx to Rx” refers to the time from the Tx interrupt of the request message to the Rx interrupt of the response message. The “Rx to Tx” refers to the time from the Rx interrupt of a response to the Tx interrupt of the next request. The application program timing is the timing of the `PING` application. The “Replier Tx Delay” is the time from the replier’s `NetXmit` routine handing the frame off to the chipset to the time of the Tx interrupt generated by the chipset when the response message has been completely transmitted around the ring. The “Rep Rate” is the time from one request to the next and should equal the sum of “Tx to Rx” and “Rx to Tx” times. It also corresponds to the measured time to make the request and receive the reply from the application’s viewpoint. The last two columns are derived from the measured data. The “Frames/sec” only counts the response frames, not the request frames. The “KBytes/sec” only counts the requested memory data bytes, not the total bytes in the frame.

Nonvolatile Memory Analysis

Diagnostic aid page application

Fri, Oct 14, 1994

Local station system software supports a memory-resident read-only “file” system that allows for downloading “named programs.” These programs can be page applications, or local applications or data files. The only write provision is that of writing a new version of an entire file. This can happen by using the Download Page for copying a program between stations or downloading S-records via the serial port. It can also happen via the TFTP server, commonly used to initially download a program from the Macintosh-based MPW development environment. Nonvolatile memory is used for storing these “files” of program/data, in order that they can survive a power-off condition. Because this area is a very long-term storage, corruption can occur from not-quite-bug-free programs. Part of this potential problem is alleviated by use of a checksum of the “file” contents. The checksum is checked when the file is read out. But there have been occasions when allocated space has been in a state other than “free”. The diagnostic tool herein described is designed to help analyze the integrity of this nonvolatile file system.

Each file is referenced by an entry in the file directory, which consists of the entries in the CODES table. Each 32-byte entry specifies an 8-character name, such as LOOPECHO or PAGEPARM, a size, a checksum, a download ptr, an execution ptr, and a version date. The nonvolatile memory area itself includes a header that specifies a ptr to the linked list of free spaces, the total size of the area, and the maximum contiguous free space. At system reset time, all free spaces are coalesced into a single free space. A free space includes in its first 8 bytes two longwords, one a ptr to the next free space (or NIL) and the other the size of its own free space. An allocated block includes a header that specifies the allocated size of that block.

The plan for this diagnostic tool is to scan through the contents of the nonvolatile area, identifying the type of entry, and producing a list of ranges of addresses containing each type. One can select to look for allocated blocks, or free spaces, or both. The resultant list will show holes where a non-matching type is found.

A free space is identified by its inclusion in the free space linked list. An allocated space is identified by an entry in the CODES table whose download ptr references it. An entry that is neither free nor allocated is unavailable for use. Such areas need to be identified so that they can be converted into free spaces for subsequent use. This diagnostic tool is especially aimed at finding such unavailable areas. A nonvolatile area that is unknown may be converted into a free space. For the moment, this must be done manually.

Display layout

```
(put picture here)
```

Enter the node# of the station of interest. An interrupt causes the following sequence of actions:

1. Request global variable at $F50, the ptr to the nonvolatile area. Request non-volatile area header. Request size of CODES table from its table directory entry (#9). Request first

part of CODES table. (Minimum size = 64 entries.) Show nonvolatile area ptr on display after NV=. Show total size of area after SZ=. Display maximum contiguous free space after MF=.

2. Request rest of CODES table, using info about its size, for local reference. When collected, show number of valid entries after CODES=.

3. Request first two longwords of each free space block in nonvolatile area, and save for local reference, following linked list of free blocks. Display total #free blocks after FREE=.

4. Request first two longwords of each block in area, beginning with base address + 0010. For each such block, determine type of block, whether free (in free space list), allocated (has ptr in CODES table), or unknown. If matches requested type of interest (A, F or – on second row), then add into range coalescing logic and update display. Update #allocated blocks found after ALLOC=. Update #unknown spaces after UNKN=.

Diagnostic results

Many stations were found to have unknown blocks. Almost all of them were blocks with no data inside. This could happen by use of the download page to start a transfer from S-records via a serial port, when the transfer was not completed. Such a transfer could be targeted at a group address such as, say, all Linac stations. This leaves an invalid CODES table entry that, when replaced by a subsequent successful transfer of the same program, leaves an allocated block that is orphaned; *i.e.*, no CODES table ptr references it.

In order to "fix" this problem, find a free entry in the CODES table, and fill it with an entry using the name AAAAAAAA, say. Enter the size – 8 as found from the bottom line of this diagnostic page, and also enter the download ptr as the address + 8 from the bottom line. Go to the download page, display entries starting with A, and you will get this single entry. Type zeros on top of the size given and interrupt immediately following the zeroed size value. The problem block will be freed. Return to the diagnostic page to again check for more problems. This method assures that consecutive free blocks will be coalesced and the linked list maintained in order.

Page G Plan

Remote access to local console applications

Mar 10, 1992

The local station console interface is simple and can be emulated from any host platform. It allows remote emulation of many page applications already written without replicating their functionality on the host. It is already supported from a Macintosh, workstation, Vax, or another local station console. This note describes the logic used to facilitate implementation on other platforms.

The local console display consists of 16 lines of 32 characters each. These 512 characters are kept in an image buffer starting at address $000400 in the target station. The first 32 characters are the top line, the next 32 characters are the second line, etc. Alpha betic characters are upper case only; lower case characters need not apply. In addition, the sign bit of each character byte signifies that the character is dis played in inverse video.

The basic scheme for capturing the display is simply to make a periodic request for the 512 bytes of image buffer memory. Any characters which change are then updated on the host's display or window. The cursor location can be monitored by the memory word at $000F7C for the column# 0–31 and the word at $000F7E for the row# 0–15. The host display may update to show this cursor position on its own display, if needed. Writing to these two words can change the cursor position to allow a mouse click to set the cursor location, for example.

Typing on the screen is supported by sending the character code to memory address $000F69 using a one-byte memory setting. Note that the sign bit of the new character *must be set* to announce to the local station system that the character is new. The following character codes are used:

$1E	é¾	Home (top line, left column)
$0B	↑	Up arrow
$0A	↓	Down arrow
$08	←	Left arrow
$0C	→	Right arrow
$0D		Return (to start of current line)
$1B	ESC	Keyboard interrupt

To each code add $80 to produce the byte value used in the setting.

The state of the push-button lites is given by the two bytes at $000F78 (units) and $000F79 (modes). The bit assignments of these lites are as follows:

Bit#	*Units*	*Bit#*	*Modes*
7	(unused)	7	(unused)
6	Eng	6	A/D
5	Volts	5	D/A
4	Hex	4	Nom
3	Raise	3	Tol
2	Lower	2	Set
1	Left-right	1	Keyswitch
0	Up-down	0	Keyboard int

To make a change in the lites selection, note that there are two mutually exclusive groups of lites, the A/D-D/A-Nom-Tol-Set group and the Eng-Volts-Hex group. Selection of any lite in a group means that the other lites in that group should be turned off. The local station only updates these latching mutually-selectable lites when a push-button is pressed, leaving these bits open to remote access.

The other lites are DC. Turning on a given bit turns on the lite; turning off the bit turns off the lite. The page applications monitor the state of the *lites* to know what to do. With the local console in place, pressing the switches is one way to control these lites; settings from a remote host to the proper lites byte is another.

Since these other lites *are* DC, however, the local station updates the lites to reflect the current state of the switches every cycle. This makes it hard for a remote host to confidently change the lites between the last update and the page application's monitoring of it. A solution to this problem lies in recognizing that the hi bit of each lites byte is unused, due to the hardware interface design. When a remote host makes a setting to one of these bytes, it should also set the hi bit to signal that it is new from the host, much like the keyboard character logic. When the local station processing is about to update the lites byte, it should check the hi bit of the present lites byte value; if it is set, merely clear that bit, else update the lites byte normally. This gives a chance for the page application to notice the remote setting, but it also builds in an automatic reversion to maintaining the lites according to the current console switch states.

Knob control is not at present supported, due to the way it was originally implemented. Every cycle, the latest reading of the knob counter from the console interface is monitored for changes. If a remote setting changed the value of this byte, it would only be overwritten the next cycle, at least if there is a local console attached. Without a local console, manipulation of the knob byte reading could effect knob control. If it is important to implement knob control, the system

should be changed to work with knob count differences. Then the remote host could write to this difference count byte, which would be cleared when used.

In summary, the data acquisition is to periodically collect the 512 bytes of image buffer memory, and also the 8 bytes from $000F78–000F7F, to obtain the lites bytes and the cursor position words. Settings are accomplished by writing to either of the two lites bytes or to the keyboard character input byte, or perhaps to the cursor position words.

Parameter Page

Local Display Application

Sep 19, 1989

Introduction

The Parameter Page is a generic application that has been implemented on many platforms of the various accelerator control systems at Fermilab. It is designed to display a list of devices along with their current reading values (and other related values) scaled to engineering units and updated at a rate of about once per second. One can also control those devices which are controllable. A means is provided to select what devices make up the list of devices and to save such sets of devices. One can usually plot device data in real time at a rate of 15 Hz and observe the effect of knob adjustments of a device upon the data which is being plotted. These features are implemented in the Parameter Page application that runs on the Local Station display; in fact, the Parameter Page was the first applications to be written for the Local Station system and remains the mainstay.

Display layout

```
1 PREAMP PS        09/19/89 1404
TEMPT1 T1 TEMPERATURE  17.23  C
TEMPT2 T2 TEMPERATURE  16.71  C
TEMPT3 T3 TEMPERATURE  17.19  C
TEMPT4 T4 TEMPERATURE  16.61  C
TEMPT5 T5 TEMPERATURE  16.65  C
TEMPT6 T6 TEMPERATURE  16.74  C
TEMPT7 T7 TEMPERATURE  17.5   C
TEMPIN H20 TEMP IN     1.566  C
TMPOUT H20 TEMP OUT    19.14  C
DIDEOT DIODE TEMP OUT  16.93  C
DIDEIN DIODE TEMP IN   16.78  C
CTL+15 CONTROL +15V    15.15  V
CTL+5  CONTROL +5V     4.908  V
CTL-15 CONTROL -15V   -14.83  V
```

There are up to 14 display lines available for displaying device data. Information that can be displayed about one device is:

1. The 6-character device name
2. A minus sign used to indicate that the device is controllable
3. Up to 14 characters of the device title (skipping the first word)
4. Associated digital status bits (reduces title space)
5. Associated digital control bits interrupt buttons
6. Numeric value: reading, setting, nominal, tolerance, channel#, “set”
7. Alarm status: good / bad, beam inhibit
8. Engineering units text

Numeric field

The various modes of display are reading, setting, nominal, tolerance and “set” values, and they are selected in turn by the buttons labeled “A/D, D/A, NOM, TOL,

and SET" on the small console. The first four options are obvious. The last one requires explanation. The name "SET" comes from its use as displaying the setting value of a controllable device saved when the line was last initialized—either upon entry to the page or upon manually entering that device on the line. If there has been no (net) change in the setting since that time, the node and channel# is displayed instead in the form "NN:CCC," where the node# NN and the channel# CCC are displayed in hex. If a previous setting is displayed and one interrupts anywhere in the numeric field, the device will be restored to the saved value, and one should expect to see the saved value replaced by the node/channel representation. Note that the saved value that is restored is not read from the screen in order to prevent round-off problems.

The three unit selections available are engineering units, volts and hex, available for selection by the buttons labelled "ENG, VOLT, and HEX" on the small console. While engineering units or volts are selected, the values displayed are averages of the 15 Hz values, over 13 cycles, with preference given to cycles that have beam status asserted (see below). While hex is selected, the values displayed are merely sampled every 13 cycles. The reason for this is that hex may be used for checking for "stuck" or "missing" bits on an A/D reading, for example. The reason for normally displaying averages is simply to cut down on the noise. Whenever volts or hex units is selected, the engineering units text field for each line is changed to read " V." instead.

The alarm status is indicated by displaying the current value field in inverse video to indicate "bad" alarm status. If beam is inhibited because the inhibit option is selected for that device, the inverse video action is flashing.

Access to the alarm flag bits is not provided by the Parameter Page, except that a device can be observed to not be in the alarm scan (active flag bit=0) if the numeric field is blank when displaying nominal or tolerance values. Alarm flags for an analog channel can be accessed from the Analog Descriptor Page.

Engineering units scaling

For each analog channel there are 4 floating point constants that are used for engineering units conversion. The values are normally entered from the Analog Descriptor Page. The first two are used for scaling the reading, nominal and tolerance values; the third and fourth constants are used for scaling setting values. For each pair of constants, the first is a fullscale value, and the second is an offset, both in engineering units. All the scaling formulae are linear. To handle nonlinear cases, the raw data must be linearized first, perhaps by use of a Data Access Table entry that is designed for the purpose.

To scale a reading or a nominal value to engineering units, use the following formula:

```
eng:= Float(raw)/32768.*F1 + F2;
```

F1–F4 are the four floating point constants, and raw is the two-byte value.

To scale a tolerance to engineering units, use the following formula:

```
eng:= abs(Float(raw)/32768.*F1);
```

Note that the offset value is not used, as the tolerance is like a first derivative. Tolerances have no sign.

To scale a setting to engineering units, use the following formula:

```
eng:= Float(raw)/32768.*F3 + F4;
```

To derive the raw values from engineering units values, merely work the above linear formulae backwards.

A special case exists for motors. When entering a setting for a motor, one is really entering a *desired reading*, so the first two constants get used, rather than the last two. The system software actually uses the third constant in this case as the number of motor steps required to make a 10 volt change in the reading, in order to compute the number of steps to issue to the motor.

Beam status

Beam cycles are indicated by status bit `09F` in the local station having the value zero. The averaging logic attempts to select those cycles which have beam status asserted for inclusion in the averaging accumulation. In the event that no cycles have beam status, then the averages include all those non-beam cycles. If all cycles have beam status, then the averages include all those beam cycles. This means that if the status bit never changes, the averages simply include the data from all the cycles. It's when the beam status varies that it gets interesting. If the average has started with a non-beam cycle, but a beam cycle occurs later in the series of 13 cycles that make up the average set, then the average is started over with that beam cycle, and only data from beam cycles will be included over the next 12 cycles. A result of this is that a beam cycle occurring at 1 Hz, say, will result in numeric values being updated with that beam cycle's values at 1 Hz. Or, a burst of 5 beam cycles repeating every 2 seconds will result in averages of the 5 beam cycles alternating with averages of 13 non-beam cycles. The updating attempts to track beam cycles.

Analog control

Analog devices can be controlled by typing a value into the numeric field and pressing the keyboard interrupt button, with the cursor positioned immediately after the last character typed. (This is referred to as entering a value and "interrupting." The significance of the term "keyboard interrupt" is steeped in history.) In this way, setting, nominal, or tolerance values can be entered in the currently selected units. The updating of a device's current value is inhibited while the cursor remains within the numeric field in order to prevent overwriting the value being typed.

A special form of input is permitted for stepping motor control. A number followed by an "`S`" within the numeric field is interpreted as the number of steps to issue to the motor. Without the "`S`" included, the value entered is accepted as a desired reading in the current units. The system uses the 3rd floating point constant in the analog descriptor, which for a motor-controlled device is the number of steps to make a 10 volt change in the reading, in order to decide how many steps to issue to change the reading from its current reading value to the desired value. The process is not iterated. A

subsequent adjustment of a motor which is moving will override its current control. One can therefore stop a motor that is moving to some far-ranging destination by either typing a "0S" or by adjusting the knob slightly.

Another means of analog control is through the use of the knob on the console, with the cursor positioned anywhere on the line of the device to be adjusted. The knob adjusts only the setting no matter what value is displayed. The settings are issued and the numeric value is updated at 15 Hz (with instantaneous values, not averages) while the knob is being turned so that one can more precisely follow the adjustments as they are being made. The update period for numeric values on lines whose devices are *not* being adjusted is about 0.8 seconds (13 cycles @ 15 Hz). An alternative to using the knob for analog control is the use of the raise/lower buttons on the console. They act to control the device at a rate of 16 "knob clicks" every 15 Hz cycle, which is about one turn of the knob per second. Knob control is related to the adjustment of the setting of a device through automatic scaling that depends only upon the analog control type that is used by the given device. An internal table in the system software (in the SETAC module) relates the knob sensitivity to the analog control type#. For control of stepping motors, one knob click corresponds to one motor step.

Knob adjustments are made in the station that is interfaced to the device. That node always knows the most recent setting value, as all setting commands to that device must pass through that station. The knob adjustment is scaled and added to the most recent setting to produce the new setting. If the allowed ± 10 volt range is exceeded, the setting value is clamped to the appropriate end of the range. Some analog control types, such as timers, may be clamped to zero at the low end.

One other special form of analog control is applied to the nominal value of a device. An interrupt in the engineering units text field copies the present reading of the device to the nominal value. If one has just tuned a device to a new setting, this interrupt updates the nominal value to match.

Digital status

Associated digital status bits can be displayed on the same line as the analog value. If there is one associated digital status bit, the last 6 characters of the title field are replaced by two 3-character fields to indicate which of the two states of status is current, such as " ON..." or "...OFF," for example. (The periods here indicate that the two states represent control buttons described in the next section. If no control is available related to the status, then spaces are shown in place of the periods.)

For two associated status bits, the first one (as described in the analog descriptor digital control field) is displayed as just described. The second one is displayed in another 6-character field just to the left in a similar manner, and no title is displayed as there is no room for it (either on the display or in the descriptor title field).

Digital control

Digital control buttons can be designed to work in conjunction with the corresponding digital status. Interrupt under the 3-character field that is represented by

periods to execute the control that would presumably result in the desired change of status. In the above example, with the status field displayed as "`...OFF`," for example, interrupt under the periods to make the device turn `ON`. It is also possible to have digital control without a corresponding display of status, such as might be used for a "`RESET`" control. In this case, the field would always display the same way, and an interrupt anywhere in the field would issue the `RESET` control to the device. Details on how to set up the analog descriptor to accomplish the many and various types of digital control that are supported by the system are covered in a separate document entitled "Digital Control Pulse Delays."

Error status

An error status code number is displayed in the second character position of the top (title) line. If there are no errors, this character will be blank; otherwise, its value is displayed in inverse video. Likely values that may be displayed are:

8 At least one node is not responding to the request for data.

7 The requested data seems to be tardy, but data has been received at least once from all nodes. This is common if nodes are not synchronized.

Device entry

There are two principle means of entering a device to be displayed on a line. One is by entering the device name (from 2–6 characters) at the beginning of the line, over any name already displayed there, and interrupting. If the name is not the name of a local device, the system uses the network to find another station that has a device by that name. (This is done by a broadcast message to the network and results in almost no perceptible delay.) If there is no response from the network, a blank line results.

The other means of device entry is by entering the node/channel representation and interrupting. If the device with that node/channel does not exist or if the node is unreachable via the network, then a generic form of parameter line results that includes the node/channel designation. If the node is down and subsequently comes back up, the line will automatically re-initialize with the proper data showing. This happens because the system automatically reissues active requests to silent nodes every 2 seconds.

All such changes made to the list of displayed devices are temporary unless they are "saved." To make the current list of channels the "permanent" list—the one displayed when the page is subsequently re-invoked—type an "`X`" in the upper left corner of the display (use the Home key on the keyboard) and interrupt. The current list will be saved and the display refreshed.

More details of variations of device entry are found in the document entitled "Entering New Parameters on a Local Station."

Plotting

A somewhat crude form of plotting is supported using the "semi-graphics" mode of the display chip used on the Crate Utility board. This mode allows display of a pattern of 6 square dots (2 wide by 3 high) at each character position on the screen. For the Parameter Page support, an 11-line portion of the display is used for the graphics plot

using this mode. This gives a resolution of 64 dots of width and 33 dots of height. The remaining three lines (out of the 14 normally used for parameters) are used for entering scale ranges and two parameter lines. The two parameter lines are used to select the channels to be plotted and are also available for control in the usual way.

There are two forms of plotting control supported. One is a time plot—plotting one channel's reading against time—and the other is a parameter plot—plotting two parameters against each other.

To enter the plotting mode, place the cursor at the start of the 12th parameter line. (Pressing the Home key and 4 up-arrows will do this.) Type a "T" or a "P" and interrupt to enter the time plot or the parameter plot mode respectively.

Upon entering the time plot mode, the 12th line will read as follows:

```
YI,F,TIME=0,10,60
```

or something like that. The three values shown are the initial y-value, which will be plotted at the line above; the final y-value, which will be plotted at the top of the line after the page title; and the number of seconds to cross the screen from the left edge to the right edge. The three values can be altered by typing in free form with the cursor position respected. Whenever new values are entered and the interrupt is given, the plotting area is blanked and data from the y-channel (displayed on the next line) is plotted at 15 Hz. When the x-position reaches the right edge of the screen, the data continues to plot, although the x-position no longer changes. To restart the plot with the same scaling, interrupt to the right of the values shown. To continue the plot at the left edge without erasing what is already plotted, interrupt with the cursor to the left of the equal sign (but not at the start of the line). As the plotting of data points progresses, an attempt is made to show the most recent data value plotted as a bright dot and the old data as a dim(mer) dot. The word "attempt" here refers to the fact that the bright / dim option in the semi-graphics mode can only be applied on a character position basis (2 dots by 3 dots).

Upon entering the parameter plot mode, the 12th line will read as follows:

```
YI,F,XI,F=0,10,0,10
```

or something like it. Here the scaling values are the initial and final y-values followed by the initial and final x-values. The y-parameter is the device displayed on the next line, and the x-parameter is the device displayed on the following (last) parameter line. In this mode the current x-data and y-data values are plotted continuously at 15 Hz. The knob can be used to adjust, say, the x-parameter to observe the effect of its changes on the y-parameter. If the selected display mode is settings, then the x-data value (but not the y-data) plotted is the setting value, rather than the reading value. Other variations are similar to those of the time plot.

In either mode, device data can be plotted in either engineering units or in volts. To exit the plotting mode, interrupt at the *start* of the 12th line. The lines of devices that were preempted for the plotting area will reappear. One can make the plotting mode "permanent" by typing an "X" in the corner as described above for saving the current list of channels.

Internal matters

Data requests and setting commands first check the size of the channel numbers involved to see if the short ident format, which can be used for channels in the range `00–FF`, can be used. If so, it uses short format idents. This was done to insure compatibility with systems that do not recognize the more recently-implemented long format idents. This logic can be removed if this backwards compatibility is not required.

The use of the `conv` byte in the analog descriptor has changed. It used to have values of 1–4 to signify (`normal,rfGrad,rfPower,zeroData`) attributes of a channel. It is now coded in a bit-significant way, where bit #1 (mask=$02) means `captureData` and bit #2 (mask=$04) means `zeroData`. In this way, a channel can have both attributes using a value of $06. The Parameter Page was written prior to this change. There may be adjustments that should be made in the light of the new `conv` usage.

Many details not discussed here can be found in the source code, which consists of about 1100 lines of Pascal compiled into about 10K bytes of code.

Ping Client Test

Page application

Sun, Aug 9, 1992

This application page aids in testing IP connectivity and response. It sends the usual ICMP Echo Request message to the given target IP address and displays the response. Any node supporting IP is likely to include such a ping client.

Page layout

```
     I PING/ECHO/UDP   08/09/92 0733
*PING<131.225.129.120>   N<   2>   target IP addr, #times
SIZE<  64>       0  PERIOD<  15>   data size, period in 15 Hz
REPLY -12     T=   5 MS  N=   2    elapsed time, #replies
0000 2767 0011 0001 089C 0A0B      data received
0C0D 0E0F 1011 1213 1415 1617
1819 1A1B 1C1D 1E1F 2021 2223
2425 2627 2829 2A2B 2C2D 2E2F
3031 3233 3435 3637 3839 3A3B
3C3D 3E3F
```

Operation

Enter the target IP address and #times to execute on the second line. As each request message is sent, the sequence field in the ICMP header is incremented.

Enter the data size of the request message, including the 8-byte ICMP header size, on the third line. A size value less than 12 is changed to the default value of 64, as a minimum of 10 bytes is needed for the ICMP header and the time words. The maximum value allowed for the data size is about 9000 bytes, consistent with the largest IP datagram supported by the local station system.

Enter the period in 15 Hz cycles at the right of the third line. (If the period is 15, test requests will be sent at 1 Hz.) A period value of zero is a special case that means “send the next request as soon as a reply is received from the last request.” This allows for testing much faster than 15 Hz for target nodes that respond quickly.

An interrupt anywhere on the second or third lines initiates the test sequence. (The exception is the area of the 4 characters of PING, where an interrupt is used to select the UDP Echo mode described below.) Requests are sent to the target node using ICMPSend, according to the period specified, where the identification field (third word of the ICMP header) is set to the portId returned from UDPOpen. (As this program was partly writ ten to test the UDP layer routines, a new UDPOpen is made for each new test sequence.) The PING characters are hi-lighted to show activity for the duration of the test sequence, after which the hi-lighting is removed. Another interrupt during this activity will abort the test sequence. Leaving the page will also abort the test, of course.

When a reply is received, details of the response are shown on the following lines. The field after REPLY is a status return code from the call to UDPRead. An ICMP Echo Reply message is passed to the page application by the system’s ICMP processing that interprets the identifier word of the ICMP header as a portId, which via the net connect

table NETCT leads to the message queue that was created by UDPOpen. It usually shows the value "–12" that means the message queue is empty, which is expected after all messages have been read from it.

After the "T=" is the response time in ms. When the Echo Request message is sent to the target IP address, part of the data is used to hold a time value, so that when the reply is received, the elapsed time can be computed. This simplifies the processing of elapsed time in cases where replies are tardy such that multiple requests are outstanding.

The number of replies received since the last request is shown by the value after "N=". The data received is shown on the following (up to 8) lines. The first four words are the ICMP header, followed by the time word. The other words are a fixed increasing pattern. Expect to see the sequence number in the fourth word increment for each new reply. The time word can be also be seen to change.

UDP Echo option

The same page can also serve as a UDP Echo client. Select this option by an interrupt on the word PING, which will cause it to change to ECHO. (Interrupt again to change back to PING mode.) In this case, a UDP message is sent (via UDPWrite) to the standard UDP echo port (7) of the target IP node. The UDP header is not included in the reply message displayed in this case. The fourth word of the data is used as a sequence# and the fifth word is the time. The rest of the data words are the fixed pattern. A UDP Echo server returns the same UDP datagram in response to any UDP datagram received. For the local stations, the local application ECHO provides this function. It can also serve as a simple example of writing a UDP server program for the local station.

Usefulness

This page was invaluable during testing of the implementation of IP and UDP for the local stations. When used in conjunction with the network frames page, usually accessible via Page F on the local station consoles, it allows viewing of the complete frames that are sent or received by the local station. IP fragments can also be examined, along with the timing of all frames to 1 ms resolution. See the document "Network Frames Page" for more details.

Broadcast

It is possible to issue a broadcast ping message if any routers along the path permit it. When using a target IP address such as 131.225.129.255 to do this, all replies will be displayed, which at last count added up to 63. Depending upon the token ring network configuration, some replies may be lost due to congestion at the bridges. The local station system can get very busy showing the replies, and the response time displayed will include a lot of this display updating. Also, as the ARP table is filled by any IP frame received, it can get filled up very quickly. As a result, while not disallowed, the use of broadcast ping is generally discouraged.

Print Memory Utility

Page application

Fri, Jun 13, 1997

It is often of interest to know some bit of information from an entire set of nodes. The information desired may be located at the same address in memory in all nodes in the set. The usual Memory Dump page application can be used of course, but typing node#s one by one can be tedious. The Print Memory page application can make this process easier.

Page Layout

```
H PRINT MEMORY    06/13/97 1634
FILE<NBRF>    LIST<0576>
ADDR<00000E00>W
SIZE<   16>
LINE<   32>
NODE=06CF
CNTR=   21
```

The FILE parameter is a data file name. Here the file name NBRF refers to a local file called DATANBRF—the file type name is always assumed to be DATA—that contains a list of all the IRM nodes in the Booster HLRF control system. This file can easily be created using the MPW assembler and linker and the MPW tool called TFTPtool to download the resultant CODE resource in the usual way.

The LIST parameter is the node# whose serial port is used to target the lines of print output. Typically, this node would have its serial port connected to a terminal emulator program on a Macintosh or PC so the text can be captured for printing.

The ADDR parameter is the base address of memory whose contents are to be printed for each node in the node list file. The letter B, W, or L that follows the address indicates that the memory data is to be accessed by bytes, words, or longwords, respectively.

The SIZE parameter is the amount of memory starting at the base address that is to be printed, in the range 1–256 bytes. The LINE parameter specifies the format of how many bytes are to be printed on each line, in the range 1–32 bytes.

The NODE field and CNTR fields show the current node# and count of nodes that have already been processed during memory data acquisition as each node is queried every 15 Hz (or 10 Hz) cycle. Provision is made for nodes that deliver data late, say, because they reside in Germany, for example, as well as for retrying a couple of times in case no response is forthcoming for the current node being queried. These two fields are output fields. Normally, after all the data has been collected and printed from all the nodes in the list of nodes, the NODE field will match the last node# in the list from which data was collected, and the CNTR field will show the total number of nodes from

Printed data format

```
FILE<NBRF>  06/13/97 1625
 06B1:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06B2:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06B3:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06B4:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06B5:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06B6:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06B7:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06B9:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06BA:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06BB:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06BC:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06BD:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06BE:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06BF:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06C0:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06C1:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06C2:00000E00   0030 0048 0005 165D FFF5 FF3D 000E F3B6
 06CA:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06CB:00000E00   0040 0048 0004 E827 7300 7F80 000E F896
 06CE:00000E00   0010 0048 0000 0001 FFFF FC10 000E BCB8
 06CF:00000E00   0030 0048 0005 0025 7300 7F80 000E F896
```

The above is an example of the printed output result of operating the program with the set of parameters shown. For each node# represented in the data file, the node# is shown followed by the memory address. After space for an error status indicator, if any, the memory data is displayed in a format that reflects the mode of access. In this case the access was by words, so the data is shown as 16-bit words. If byte access were specified, the data would be shown as separate 8-bit bytes. If longword access were specified, the data would be shown as 32-bit longwords. Values that may be shown to reflect error status are likely to be only these two:

4 Bus error detected during memory data access

8 No response received from the node

Additional error codes 1–3, 5–7, indicate internal errors that should not occur.

Request-reply Page

Diagnostic application

Dec 21, 1989

Introduction

This application was originally developed to measure the timing for satisfying one-shot requests for data, especially across the network. It permits specification of a general data request using a given listype#, initial ident (including node#) , #idents and #bytes/ident. The one-shot request is repeated at 15 Hz, and the measured time from the before request until after the reply is measured and updated as a histogram in 0.5 msec bins over a 20 msec range.

Display layout

```
 0 R REQUEST-REPLY   12/19/89 1022
 1 *READ ID<08:103>          N<  1>
 2  LT< 0>#B<   2>= 0000
 3
 4     0  0    0  5    0 10    0 15
 5     0     156       0       0
 6     0  1  267  6    0 11    0 16
 7     0      30       0       0
 8     0  2    3  7    0 12    0 17
 9     0       0       0       0
10     0  3    0  8    0 13    0 18
11     0       0       0       0
12     0  4    0  9    0 14    0 19
13     0       0       0       0
14
```

The ID field on line 1 is the ident in hexadecimal. In this case, it means channel $103 in node $08. The N field is the count of sequential idents beginning with the given ident. The LT field is the listype# in decimal. The #B field is the number of bytes requested per ident in decimal. The value shown in hexadecimal after the equal sign is the first part of the returned data. Up to 16 bytes can be displayed on two lines.

Ident format

The format of an ident can be one of these four types:

Format entered	Internal form	Example of use
08:23	0823 short “chan”	08:0204
0008 0204	long “chan”	
08:105C00	0810 5C00 short “addr”	
08:001A0000	0008 001A 0000 long “addr”	

The only ident type for which entry is not supported is the analog channel name. This support might be a good addition to the program, since name searches are currently performed in simple linear fashion and can take awhile to execute.

Timing histogram

The time delay for the one-shot data requests made each 15 Hz cycle is tallied and displayed in histogram form on 10 lines, where the 40 bins of data counts are listed in 4 columns. Every other bin is labelled with a number in msec units, since the time values are measured in 0.5 msec units.

A free-running counter on the 68901 chip on the Crate Utility Card that counts at 2000 Hz is used to measure the elapsed time. The counter is sampled just before the call to `ReqData` that makes the data request, and it is sampled again just after the call to `Collect` which delivers the response data to the caller. The difference is what is histogrammed.

By varying the parameters such as the #bytes requested per ident and the number of idents, one can gain some good insight into the detailed timing dependencies of data requests.

Typical timing results for a simple data request such as requesting a single two-byte reading of an analog channel from a node on the token ring give a typical elapsed time of about 6 msec. One should consider that the network chipset requires about 1 msec to pass the shortest frame, and there are 4 passes through the chipset that are involved in this data request. If one makes the same request for a local channel's reading, the timing result is about 1 msec.

It is also of interest to observe the build-up of the distribution of elapsed times. When the nodes are not operating synchronously—driven by the same external interrupt—the one-shot data request can be used as a "probe" to get and idea of the amount of activity that occurs in the target station. Of course, a more direct means of getting that information is to examine the signals from the task lights and interrupt lights on the Crate Utility Board's front panel connector.

Another use for this application is to test new listypes which do not have local applications that use those listypes. Such listypes may be implemented for the use of Host computers. In fact, this application is currently the only one which allows user entry of a listype#. Other applications use fixed listypes as needed for the data they are designed to access.

Settings Log Page

Page application

Fri, Sep 9, 1994

The settings log for the local station/IRM is implemented using a data stream. A 16-byte record is written into the data stream when a setting is successfully made. This page application facilitates viewing the setting log contents. It is based strongly upon a similar page application that views network frame diagnostics, also written into a data stream, in that case, for each network frame received or transmitted. A new feature supports display of Acnet setting log records in the same data stream.

Display layout

```
C SETTINGS LOG    08/12/94 0810
NODE<0508> #RCVD=  32 LIST<0000>
NODE=0000 - LTNB=0000 TIME=0000
0508 2904 002241F6 0806:06-10+28
0508 2904 002241F6 0806:06-11+28
0508 2904 002241F6 0806:06-12+28
0508 2904 002241F6 0806:06-13+28
0508 2904 002241F6 0806:06-14+28
0508 2904 002241F6 0806:07-00+28
6041 1502 00C10100 0806:07-00+31
0623 2302 0006002D 0806:11-12+29
0610 2306 00000812 0807:00-00+29
0610 2306 00000812 0808:00-00+29
0610 2306 00000812 0809:00-00+29
0610 2306 00000812 0809:59-14+44
```

Fields listed above are the source node# of the setting, the listype# and #bytes shown as two bytes forming a word, the channel# and data shown as two words forming a long word, and the time of completion of the setting action. As in the network frame diagnostics, the time is expressed in hours, minutes, seconds, 15Hz cycles, and milliseconds within the current cycle.

In the example of node 0508 setting diagnostics shown, node 0508 itself initiated settings for analog channel 0022 using listype#41 (engineering units floating point) with 4-byte data, where the first word of the floating point setting data was 41F6. These settings were set every 15 Hz cycle until a setting to clear the enable bit for the local application causing the settings was disabled by using listype#21 to toggle Bit# 00C1. (01 is the digital control type# for toggle.) Node# 6041 is a pseudo node# that indicates the setting command was received via UDP/IP. The next line shows a setting of “generally interesting data” with byte offset 06 that is used for synchron iz ing the FTPMAN cycle counter with Tevatron clock event 02. Four minutes of time-of-day settings from node 0610 also access “generally interesting data.”

Operation

Interrupt on the second row to initiate a one-shot data request in order to capture the most recent setting records from data stream #1 from the node# between the "<" and ">". The number after RCVD shows the number of records received in response to that one-shot request. If the LIST field is nonzero, then send a serial listing of the same diagnostic records displayed to that indicated node#.

Interrupt on the third row to modify the filter parameters that select from the captured diagnostics above. Filtering can be done on the source node#, on the listype#/#bytes, and on the time. (Obviously, only hours and minutes can be specified for the time filter.) Characters that denote the type of filtering for each filter are used where the "=" sign is indicated. Use "=" to mean equality, "!" to mean inequality, ">" to mean greater than or equal to, and "<" to mean less than. When a filter value field is 0000, that filter is disabled. If more than one filter is enabled, the "AND" of the filter conditions is meant.

When a list of diagnostic lines is displayed on the screen, use the raise/lower buttons to scroll by pages forwards or backwards through the list. Using a one-shot request suitable for a data stream of size 2K bytes, one can capture and display/list up to 124 setting diagnostic records.

Listing format

```
C SETTINGS LOG    08/12/94 0856
NODE<0508> #RCVD= 124 LIST<0576>
NODE=0000 - LTNB=0000 TIME=0000
SrcN LtNb ChanData HrMn:Sc-Cy+ms
0508 2904 002241F6 0806:06-10+28
0508 2904 002241F6 0806:06-11+28
0508 2904 002241F6 0806:06-12+28
0508 2904 002241F6 0806:06-13+28
0508 2904 002241F6 0806:06-14+28
0508 2904 002241F6 0806:07-00+28
6041 1502 00C10100 0806:07-00+31
0623 2302 0006002D 0806:11-12+29
0610 2306 00000812 0807:00-00+29
0610 2306 00000812 0808:00-00+29
0610 2306 00000812 0809:00-00+29
0610 2306 00000812 0809:59-14+44
```

This is an excerpt from the complete 124-line listing showing the same records displayed in the example above.

Acnet settings log

A different type of settings log is maintained for Acnet. Support must be provided at the application level for logging setting activity of Acnet devices. For local stations/IRMs, this means such support is needed for page applications such as

the parameter page and the analog descriptor page. The parameter page can initiate settings for D/A's, nominal and tolerance values, and digital control. The analog descriptor page can initiate settings of alarm flags. Both pages have the device name available for log reporting. The method is more fully described in the document Settings Log Implementation as well as the document Settings Log LA.

The afore-mentioned Acnet records are written into the same data stream. This settings log page application displays them with a slightly different format, as shown in the following example:

```
C SETTINGS LOG    09/09/94 1004
NODE<0622> #RCVD=   4 LIST<0000>
NODE=0000 - LTNB=0000 TIME=0000
0622 0102 0424F333  L2GADJ-SET
0622 1602 00900101  K2LPGN-BCTL
0622 1602 00900100  K2LPGN-BCTL
0622 0102 00900021  K2LPGN-SET
```

In place of the time, which is not recorded for the Acnet record format, is the device name and Acnet property. The display, or listing, shows the device name and the Acnet property using these abbreviations:

Setting	SET
Binary control	BCTRL
Analog alarm block	ANAB
Digital alarm block	DGAB

In the example shown, device L2GADJ was set to a value F333, two binary control actions were performed on device K2LPGN, followed by a setting of the same device to the value 0021.

With the normal settings records entered into the same data stream, including closed loop setting activity, it's expected that the queue will wrap in a short time. The Acnet records are logged to a central setting log server, so such entries will more likely be viewed there. The other setting records will be useful as a diagnostic tool for observing settings happening very recently, much as the network frame diagnostic is used for observing recent (often *very* recent) frame activity.

Station Survey Page

Diagnostic application

Dec 28, 1989

Introduction

This application page was developed to assist in keeping track of a collection of Local Station network nodes. Some key system configuration parameters and statistics are collected and displayed for any given node, or they may be listed to the serial port for a selected set of network nodes.

Display layout

```
0 Q STATION SURVEY  12/22/89 1047
1 *SCAN  *EDIT  *LIST TO NODE<08>
2 NODE<2A>  VERSION=12/11/89  PSOS
3 CPU=68020 BOARD=133  CY=66 SW=00
4 CHANS=1024 BITS=2048 UP= 1.01  D
5 REQUESTS=4   MXFREE=4000 CERR=00
6 PAGE=2 24MW BEAM STICK  FFF26000
7 <KLYSTRON PFN    >
8
```

Enter the node number on line 2 and interrupt to get this display for any node. The above example shows the system parameters for node 2A at this writing.

The version date of the executing system is shown along with the PSOS tag if the system uses the pSOS kernel. The cpu chip# and cpu board# are shown on the next line. If the system is running in prom, an upper-case P is shown immediately after the cpu board#. On the same line are also shown the cycle length in milliseconds and the reading of the option switches (in hexadecimal) from the station's Crate Utility Board.

The number of analog channels, the number of binary bits, and the amount of time since the node was last reset (in days) is shown on line 4. The number of current data requests that are active is given, where only locally initiated requests and data server requests are counted. (Ordinary network requests are not included here; including them would be a enhancement to the program.)

The maximum contiguous block of free memory is shown next, limited to $4000 bytes for pSOS systems. This is because the dynamic memory for such systems is more than 64K bytes, and the system variable that holds that information is only a word. It would be better if it were a longword; this is another possible enhancement. Also, the pSOS systems don't provide a system call to get that maximum contiguous value directly; the system code tries to allocate successively smaller (by halves) blocks of memory in order to get an estimate of it. Some applications reference this word to gauge whether they should hold up sending output to the serial print queue, for example, since dynamic memory must be allocated for each line of text that is spooled for printing.

The console serial I/O error counter is shown next. The local console is

interrogated every cycle to get the latest reading of the buttons, keyboard and knob. Errors are tallied for the received data from that serial port.

The current display page# and page title is shown on line 6, followed by the entry point address of the application that is currently running. The above example shows an entry point that is in prom memory on the cpu board. On line 7 is an editable field that describes the location / use of that station. Enter a 16-character description and interrupt here to change it. (It is stored internally in the place where the title would be kept for page#0. It isn't used when page#0 is displayed, since the index page's title is fixed.) The description was implemented for the use of this survey application.

Selected stations

Three operations that refer to a selected set of nodes are supported by line 1 on the page. They are denoted by the keywords SCAN, EDIT and LIST.

To prepare the list of selected nodes for use by the SCAN and LIST options, interrupt under *EDIT. The word EDIT will change to SAVE, and the last 6 lines may look like this example:

```
 9 02 03 04 05 06 07 08 09 0A .. 0D
10 12 .. 16 .. 19 .. 21 .. 23 .. 24
11 30 31 .. .. 34 35 .. 37 .. .. ..
12 .. 46 47 .. 4A 4B .. .. .. .. ..
13 .. 54 55 56 57 .. 5A 5B 5C 5E 5F
14 .. .. .. .. .. .. .. .. 73
```

This is the editable list of nodes that are used by the SCAN and LIST options. While in this mode, with the SAVE word shown in inverse video, enter any node#s in any of the available fields. To remove a node from the list, enter a double period. Then interrupt anywhere in the area of those 6 lines (or under the SAVE word) to save the updated list of nodes. The node list will immediately be blanked, and the SAVE word will be changed to the EDIT prompt again.

The SCAN option is used to automatically sequence through the selected node list and display the system parameters that are collected from each node. As it does this, the node# is displayed in the 6-line area used for the node list. If the station does not respond, its node# is shown in inverse video. This mode is typically used to find out what nodes (of the selected set) are "up."

A special use of the SCAN option is to collect a list of all the nodes which are "up" to initialize the selected node list. To do this, interrupt under SCAN after having interrupted under EDIT. The set of on-line nodes will appear quickly. To *save* this list, interrupt under the SAVE word. To *prevent* overwriting the selected node list, exit the page.

The LIST option collects the same data as the SCAN option, but it prepares it for listing to the serial port of the local station (or of any other station on the network). The list may then be examined at leisure. The format of the listing produced might look like this:

```
STATION SURVEY  12/22/89 1106 *
Node Version p  Chip Board Cy Sw Chan Bits  Up-d Rq Free System_title
 03 03/07/88   68000  101P 80 40  255  248  0     0 2520 LINAC ANNEX TEST
 04 03/07/88   68000  110P 80 1C  255  248  0     0 1DA0 WH1 MUON TEST
 05 07/05/89   68000  101  80 00  255  248  57.1  2 3EA8 XGAL TEST
 08 12/14/89 p 68020  133  80 14  255  248  0.82  2 4000 LINAC SUN ROOM
 0D 09/15/89 p 68020  133  80 40  255  248  7.83  4 4000 D0 MCH3
 12 09/06/89   68020  133  66 00  255  248  35    4 3DA8 LINAC TEST BENCH
 16 03/07/88   68000  110P 66 00  255  248  0     0 12A0 MCR STATION
 21 07/05/89   68020  133  80 40  512  768  65    0 3F70 D0 MCH2
 23 10/26/89 p 68020  133  66 1C 1024 2048  13.9 35 4000 LINAC KLYSTRON
 2A 12/11/89 p 68020  133  66 00 1024 2048  1.06  4 4000 KLYSTRON PFN
 30 07/05/89   68020  133  80 00 1536 2048  10.9  4 3DA8 D0 MUON TEST
 31 07/05/89   68020  133  80 00  255  248  58.9  3 3E30 D0 FCH1
 34 08/06/89 p 68020  133  80 40 1024 2048  16.9  0 4000 IB4 BLAZEY TEMPS
 35 09/06/89 p 68020  133  80 40 1024 2048  9.02  0 4000 D0 MUON MCH R305
 37 09/22/89 p 68020  133P 80 40 1024 2048  0.79  0 4000 NWA TEST CELL
 46 08/18/89 p 68020  133P 80 0C 1024 2048  2.07  0 4000 D0 MUON MCH R305
 47 10/26/89 p 68020  133  80 1C 1536 2048  38.9  4 4000 D0 MUON MCH R305
```

The lower-case p after the system version date indicates the use of the pSOS kernel. The upper-case P after the cpu board# means the system is executing in prom.

***Not* an unassuming application**

This application presumes to know more about the memory layout of the system software than most applications would. Because of this, it may have to be updated or enhanced along with certain system modifications.

Swift Commands Log Page

Page application

Thu, Jul 11, 1996

The swift digitizer commands log for the IRM is implemented as a data stream. A 16-byte record is written into the data stream when a swift digitizer command is entered into the queue. This page application facilitates viewing the log. It is based strongly upon a similar page application that views settings log diagnostics, also written into a data stream for each setting performed by, or initiated from, the node.

Display layout

```
6 SWIFT DIG CMDS  07/11/96 1413
NODE<0576> #RCVD=   2 LIST<0576>
EVNT=00   - RATE=   0 TIME=0000
14  100 16384 4096 1303:18-13+56
12  200  1024 4096 1310:09-10+51
```

Fields listed above are the trigger clock event# in hex, the rate in KHz, the delay in µ s, #points, and the time of entry into the command queue. As in the settings log diagnostics, the time is expressed in hours, minutes, seconds, 15Hz cycles, and milliseconds within the current cycle.

In the example of node 0576 swift digitizer diagnostics shown,

Operation

Interrupt on the second row to initiate a one-shot data request in order to capture the most recent diagnostic records from data stream #2 from the node# between the “<” and “>”. The number after RCVD shows the number of records received in response to that one-shot request. If the LIST field is nonzero, then send a serial listing of the same diagnostic records displayed to that indicated node#.

Interrupt on the third row to modify the filter parameters that select from the captured diagnostics above. Filtering can be done on the event#, on the rate, and on the time. (Obviously, only hours and minutes can be specified for the time filter.) Characters that denote the type of filtering for each filter are used where the “=” sign is indicated. Use “=” to mean equality, “!” to mean inequality, “>” to mean greater than or equal to, and “<” to mean less than. When a filter value field is 0000, that filter is disabled. If more than one filter is enabled, the “AND” of the filter conditions is meant.

When a list of diagnostic lines is displayed on the screen, use the raise/lower buttons to scroll by pages forwards or backwards through the list. Using a one-shot request suitable for a data stream of size 2K bytes, one can capture and display/list up to 124 swift digitizer diagnostic records.

Listing format

```
6 SWIFT DIG CMDS  07/11/96 1413
NODE<0576> #RCVD=   2 LIST<0576>
```

```
            14  100 16384 4096 1303:18-13+56
            12  200  1024 4096 1310:09-10+51
```

This is an excerpt from the complete 124-line listing showing the same records displayed in the example above.

Tesla Cavity Conditioning

Peak power driver

Wed, Nov 16, 1994

Coupler cavity testing requires conditioning, which is careful control of the applied rf power level to the cavity, backing off when rf system trips occur. The main idea is to advance the peak power demand voltage from an initial value to a maximum value, using an initial short rf pulse length. When the target demand voltage is reached, the pulse length is slowly increased until a maximum value. When a trip occurs, reduce the power level by a fraction toward the minimum value, and also reduce the pulse length by a fraction toward its minimum short value. As conditioning improves, the rate of rise of the peak power demand voltage as well as the rate of rise of the pulse width can be adjusted to more quickly respond to a trip.

Parameters needed:	*PROMPTS*
0. Enable Bit# for this closed loop activity	`ENABLE B`
1. Other parameters:	`OTHER C`
Minimum peak power demand voltage	
Maximum peak power demand voltage	
Power demand adjustment per time interval	
Time interval in cycles for power adjustment	
Minimum pulse width	
Maximum pulse width	
Pulse width adjustment per time interval	
Peak power demand restart % after trip	
Pulse width restart % after trip	
2. rf 'on' status Bit#, state	`RF ON B`
3. rf system reset control Bit#	`RESET B`
4. Peak power demand voltage Chan#	`PPOWER C`
5. Pulse width Chan#	`PWIDTH C`
6–9. (spare)	

Backoff logic

The minimum and maximum values for the power demand specify the range of values to be used during conditioning, beginning with the value of power demand when the local application is first enabled. The same is true for the pulse width minimum and maximum values. In other words, starting the application does not cause the power demand and pulse width to be reset to their minimum values. But when a trip occurs, both the power demand and the pulse width will be reduced by a fraction toward their minimum range values. For example, if the power demand range were 1–6 volts, the pulse width range were 100–1000 μ s, the restart fractions for both were 50%, the current power demand was 5 volts, and the current pulse width were 500 μ s. If a trip occurs, then the power demand will be reduced to 4 volts, and the pulse width would be reduced to 300 μ s. If another trip occurred right away, the power demand would be further reduced to 2.5 volts, and the pulse width to 200 μ s. Contin

uing frequent trips would drive both parameters toward their minimum values.

1553 Test Page

Diagnostic application

Oct 16, 1989

Introduction

This application is used for diagnostic testing of 1553 hardware attached to a Local Station. Sequences of 1553 commands including output data can be composed and run multiple times at 15 Hz to increase the usage rate. Errors are detected and counted, retaining the last error status word. The 1553 hardware driven by this program must be attached to the local station. (To initiate testing remotely, one could use the "Page G" facility that allows running a station's application from another station's console.)

Display layout

```
0     9 1553 TEST PAGE  10/13/89 1430
1     *NOW      450 ADDR=200000 #=  10
2      RT T SA #W DATA  CMD STAT ERRC
3     * 2 1 18  1 5555 1641 1000 0000
4     * 2 1 19  1 AAAA 1661 1000 0000
…
```

1553 commands (up to 10 lines)

This example shows reading the two digital input words from a rack monitor selected as RT#2 accessed via controller #0, after running for 3 seconds.

Display areas

Enter the 1553 controller base address on line 1. Each 1553 controller board includes two 1553 controllers, and each controller occupies 64K bytes of memory space. The most significant 7 bits of the board's base address are set via switches on the controller board.

Enter a multiplier count at the right end of line 1 to specify the number of times per 15 Hz cycle that the sequence of 1553 commands specified on the page is to executed. Using a value larger than 1 gives bright scope traces and can generate better error statistics.

Starting with line 3 is a list of up to ten 1553 commands. The first column is a flag character (blank=inactive, period=one-shot, star=continuous). The 4 components of the command word are shown in the next 4 fields of each line. The first data word is shown next, followed by the encoded command word, the status word returned from the RT, and the error counter. The error counter field can be changed to the controller's error status word by interrupting under the ERRC word on line 2 to change it to ERRS. (Interrupt again to change it back.) The data word only gives the first word of data associated with the 1553 command transfer. To access the other data words, either use the raise/lower buttons, or enter a "group#" in the range 0-8 after the word DATA on line 2. Group 0 is displayed as above. Group 1 displays the first 4 data words (starting with the first word that was shown in group 0), group 2 displays the next 4 words, etc.

Entering 1553 commands

Specify each command by entering decimal values for the remote terminal (RT), the transmit/receive bit (1=Transmit), the subaddress (SA) and the word count (#W). An interrupt at the start of the line toggles through the flag sequence (space, period, star).

To activate, interrupt on line 1 under the word RUN. To deactivate, interrupt again.

While entering changed values of the fields which make up the command or data, it is not necessary to interrupt, as all the displayed fields will be read when the RUN is activated the next time. While it is active, the commands cannot be altered by typing new values and interrupting; however, the flags can be toggled to enable / disable commands individually, and the data word values used for output can be altered.

Status bit display

There are 1553 standard conventions for the use of the bits in the status word received from an RT in response to a command sent by the controller. To display a status word shown in hex from the STAT column, whether the RUN is active or not, interrupt under the status word value. The display is then rewritten to show the bit-by-bit breakdown of the status word. The top 5 bits are the RT address and are shown as one value, whereas the other bits are each shown on a separate line. Interrupt anywhere on these 12 lines to return to the main display.

Similarly, there are specific meanings for the error word received from the controller. These can also be broken out by interrupting under the error status word (or error count). Another interrupt returns to the main display.

Retained parameters

Upon exit from the page, the base address, multiplier count, and all 1553 commands including the execution flags are saved for recall when the page is re-invoked. Only the first data word, however, is saved. When the page is next called up, the other 31 data words for each command will be set to zeros.

1553 data acquisition

This program uses a diagnostic routine called EXEC1553, included with the system software, to access the 1553 hardware. It executes one command at a time by setting up a 1553 command block at the base address plus $80. The Pascal declaration of this routine is as follows:

```
Procedure Exec1553(cmd: Integer;      {  1553 command word }
      adr: Longint;       {  controller base address }
      VAR data: Integer;  {  data array }
      VAR stat: Integer;  {  returned RT status }
      VAR errs: Integer); {  returned error status}
```

Token Ring Page

Network initialization

Dec 20, 1989

Introduction

The token ring chipset must be initialized and opened onto the network before any network communication can take place. This application page contains that necessary logic. It can be run manually step-by-step, or it can be run automatically as an "auto-page" application, which is the normal case. The system initialization code checks to see if the network is already open; if it is not, it checks for the presence of the Token Ring Page application as page "T" with the title `TOKEN RING INIT` and schedules it for auto-page execution.

Display layout

```
 0 T TOKEN RING INIT 12/19/89 1022
 1 *RESET
 2 *INITIALIZE  IR:0040
 3 *OPEN       SSB:
 4 *RECEIVE  $105100: PARLIST
 5 *TRANSMIT $XXXXXX: PARLIST N<  >
 6 COMPLETION STATUS
 7  1      RING STATUS
 8  2      COMMAND REJECT
 9  3      OPEN
10  4      TRANSMIT
11  6      RECEIVE
12 $FFFC10: BOARD ADDRESS
13 $105000: SCB ADDRESS
14 $105008: SSB ADDRESS
```

To operate the program manually, interrupt under `*RESET` to reset the chipset. After about 1 second, the Interrupt Register, shown in hexadecimal after the `IR:`, should settle to a value of `0040`, meaning that the bring-up diagnostics built into the chipset have completed successfully. If bit#5 and bit#4 (mask=`0030`) are set, the bring-up diagnostics have detected an error indicated by the code in the low 4 bits of the register as follows:

0 Initial Test Error
1 Adapter ROM CRC Error
2 Adapter RAM Error
3 Instruction Test Error
4 Context/Interrupt Test Error
5 Protocol,Handler Hardware Error
6 System Interface Register Error

These error codes are listed in the "TMS380 Adapter Chipset User's Guide" by Texas Instruments, the manufacturer of the token ring chipset.

Interrupt under the `*INITIALIZE` to initialize the chipset. This completes

rapidly and should result in a value of `0000` in the Interrupt Register, and the value of the System Status Buffer, shown in hexadecimal after `SSB:`, should be `FFFFD1D7C5D9C3D4`. If a value appears with bit#4 set (mask=`0010`), the low 4 bits may exhibit one of the following initialization error codes:

1 Invalid initialization block
2 Invalid options
3 Invalid receive burst count
4 Invalid transmit burst count
5 Invalid DMA abort threshold
6 Invalid SCB
7 Invalid SSB
8 DIO parity
9 DMA timeout
A DMA parity error
B DMA bus error
C DMA data error
D Adapter check

This table was copied from the TMS380 manual referred to above. Further details are found therein.

Interrupt under `*OPEN` to open onto the token ring network. This takes about 20–30 seconds and should result in the first word of the SSB to change to `0000` after having been set to `0003` when starting the `OPEN` operation.

Interrupt under `*RECEIVE` to allow reception of token ring network frames. A count should show up on line 11 indicating the successful initiation of the Receive command, and a completion status value of `8000` should appear. The receive logic of the node is now enabled. The transmit logic will be automatically initialized when the first frame is transmitted.

In summary, one can initialize the chipset manually by interrupts on lines 1–4 in sequence with appropriate delays for each action to complete.

Completion status

The program monitors the SSB to observe network chipset activity. Since it only executes every 15th second cycle, it can easily miss such activity. It displays the completion status on the appropriate line and keeps a count of the number of such occurrences. For the case of a received frame, the `CSTAT` from the Receive Parameter List is shown instead, along with the received frame size.

Automatic operation

While the program is being operated automatically, following system reset, one can monitor the actions of the program as they are taken in sequence. The interrupt register is checked to make sure the chipset is behaving as expected. If

it is not, retries will be made. If all retries fail, it just stops, and the program awaits manual recovery.

During automatic sequencing, the rest of the system is running normally, collecting data and scanning for alarms, etc. But network transmissions are held off while network initialization takes place. Messages to be sent to the network are queued through the usual `OUTPQ` mechanism. When the Receive command has been issued, following a successful Open, any messages queued for transmission will be sent to the intended destination nodes. The normal timeout imposed by logic in the `OUTPQMON` routine of the QMonitor Task, is also disabled during network initialization.

Transmit testing

This documentation describes how Page T performs in the pSOS version of the system, in which a chain of Transmit Parameter Lists is used. A previous version of the system used only a single TPL and thus could not queue multiple frames to the chipset for more efficient network transmission. In that system, one could use the line beginning with `*TRANSMIT` to send multiple copies of the same frame that was last sent. It was a means of kludging up network transmission activity for diagnostic purposes. It wasn't used much.

Token Ring boards

The program automatically determines which token ring board is in use and acts accordingly to accomplish that board's initialization. The two board types supported are the Fermilab board and the Proteon board. Each board uses the same TI chipset, but the Proteon board has some special board registers that must also be set up.

Local board only

This is one of the few application programs that must be used locally; i.e., one cannot initialize another station's token ring board over the network, of course. (Another page that has a local-only characteristic is the 1553 Test Page.)

The program assumes knowledge of the location of the Token Ring system table, where it obtains some of its parameters. A byte variable in the Transmit Parameter List table header called `xmtActiv` is used to hold off network activity of the token ring driver while the chipset is being initialized and opened onto the network. The program also assumes knowledge of the addresses of the token ring chipset registers. It uses pointer references to access the `TRING` table and the chipset. At this writing, these addresses are as follows:

`TRING` table base address	`$00105000`
xmtActiv byte variable	`$00105C20`
Base of chipset registers	`$FFFFFC10`

Off-network operation

It may occasionally be useful to run the system without being on the network. Some means of inhibiting the automatic invocation of the Token Ring Page is needed to do this. There are at least four approaches that can accomplish this.

One can remove the token ring interface board from the VME crate. The system should still run in its stand-alone fashion without using the network. Data requested from other nodes will not be found, just as if the other nodes were not on the network. But data is still collected, applications are run, and alarm scanning is performed. To make use of such alarms, one would probably want to enable the local alarms generation on the bottom line of the screen and/or through the local serial port.

Another way to inhibit the use of the network is to disable the automatic call-up of the Token Ring Page by changing the entry point address for Page T, or by moving the application to a different page. It could still be used manually.

A third approach is to unplug the token ring cable from the token ring board. This would cause the Open to fail, since there is no longer a connection to the network media. Page T would eventually discover this and stop.

A fourth option is to simply manually exit from the token ring page before it completes its job of opening onto the network. Local console activity is still permitted even while the page is running automatically. Just press the Home key on the keyboard and exit the page.