LA-UR--85-3582

DE86 002397

10/85

TITLE: PROTON STORAGE RING - THE MAN/MACHINE INTERFACE

MASTER

AUTHOR(S): Robert F. Lander and Peter N. Clout

SUBMITTED TO: 2nd International Workshop on Accelerator Control Systems
Los Alamos, New Mexico, October 7-10, 1985

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

By acceptance of this article the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes.

The Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

PROTON STORAGE RING—THE MAN/MACHINE INTERFACE*

ROBERT F. LANDER and PETER N. CLOUT, MS H808

Los Alamos National Laboratory, Los Alamos, NM 87545

The human interface of the Proton Storage Ring Control System at Los Alamos is described in some detail, together with the software environment in which operator interaction programs are written. Some examples of operator interaction programs are given.

1. Introduction

In implementing the Control System for the Proton Storage Ring (PSR) a balance was struck between investigating and using state-of-the-art techniques in real-time computing and minimizing the technical risk and the cost of the control system. Thus new techniques were used only where a clear advantage was seen. The resultant man/machine interface has been highly successful. Presented here is a brief description of the PSR control system, with emphasis on the graphics interface, the ease with which new programs can be added, and some experience (after six months of operation) of the human interface. The overall control system has been described elsewhere [1,2,3].

2. Graphics hardware

At the start of the PSR controls project, observations at existing control systems indicated that, although keyboards are widely used for computer interaction, at an accelerator control system they are clumsy to use and take a large amount of valuable console space. A decision was thus made early in the design of the PSR [4] not to have keyboards in use in the control room. It was also decided to use high resolution color-graphics devices to present information to the operator. This left the decision on what operator input device to use. The options were to use a mouse, lightpen, touchscreen or trackball. Lightpens and trackballs were eliminated because they were

*Work supported by the US Dept. of Energy.

judged to be harder to use, and lightpens have a wire attached. We also noted that trackballs and mice use either specialized hardware or CPU cycles to keep the operator informed as to where the current position is between hits. Touchscreens and lightpens do not suffer from this overhead. We finally chose touchscreens, with optional use of the mouse where it was necessary to indicate individual pixels. For the graphics devices we chose two Lexidata 3400 systems with six 19 in. 50 MHz noninterlaced color monitors. These gave a high quality, flicker-free display. This system gave us 512 by 640 resolution, fast drawing characteristics, a choice of 256^3 colors, and 4 planes of memory. In front of each CRT we placed an Elographics touchscreen.

The memory of each of the Lexidata controllers is configured as four arrays (640 x 512 bits) for each of the three CRTs attached to it; that is, each CRT has four memory-planes to make up its pixel memory. (See fig. 1, Position A.) Each of these planes contains 1 bit for each pixel on that screen. Thus one has a range of 16 possible "color indexes" available to describe any given pixel. This color index serves as a pointer into each of the three Color LookUp Tables (CLUTs) for that CRT and yields an 8-bit "intensity level" from each, one for each of the three primary colors (red, green, and blue), (fig. 1, Position B). Each of these three intensity levels is sent to the DAC for the appropriate color-gun, producing the color of the particular pixel involved, (fig. 1, Position C.) Pixels should not be confused with the individual phosphor dots on the face of the monitor screen. Depending on the resolution of the image processor, a single pixel might encompass many phosphor dots on the display. Although one indexes into the CLUT with a 4-bit number (using 4 planes of memory per CRT, giving 16 immediately available colors), the CLUT actually can be indexed with up to 10 bits, yielding over 1000 colors "on tap" at any one time if 10 memory planes are used. Although it is possible to choose from a pallet of 24 million colors, our experience is that small changes in intensity level are not noticeable, and a more realistic figure would be around 700 easily distinguished colors.

As an example of CLUT use, if the fifth value in the red section of the CLUT is 255, and the fifth value in each of the other 2 sections of the CLUT is 0, and if a pixel is to be illuminated with the color (or z-value) of 5,

that pixel would be displayed as red. Another example has the third value of each section of the CLUT as 255. A pixel with a z-value of three would be displayed as white. By convention, the first value in the three sections of each CLUT is the background color, normally three zeros, or black, for each CRT.

Although a Color LookUp Table gives the programmer a wide choice of colors, it also introduces other possibilities because it gives one the ability to interpret any one of the 16 colors being sent by a program as any color at all. It is therefore possible to create the illusion of movement on the screen while not rewriting the screen at all, but merely rewriting the CLUT. If, for example, one draws a series of dots in the color-pattern 12, 13, 14, 12, 13, 14, etc, and then sends, at quarter-second intervals, a CLUT with the interpretations 12:blue, 13:black, 14:black; 12:black, 13:blue, 14:black; 12:black, 13:black, 14:blue, and then repeats, blue dots will be seen to move across a black screen.

Another possibility for using the CLUT allows a program other than the one that wrote the screen to draw an icon on the screen at any point and move it around the screen without erasing the background display. This is done by setting the bottom eight colors in the CLUT in the normal way and setting the top eight colors to be white. Now, by simply setting the most significant bit (MSB) of the color of any pixel on the screen to 1, that pixel will appear as white. By setting the MSB back to 0, the former color will reappear. This technique is used to draw and move the mouse cursor on the screen in response to the movement of the mouse in the pad.

The Elographics touchpanel placed in front of each graphics screen is a high-resolution pressure-sensitive resistive screen. Each of these touchscreens has an entry in our database, just like any other equipment in the Storage Ring. The touchscreen produces character-string input, which is interpreted in all cases by the command module [3], and the coordinates stored in the database for that screen. Because the touch screen is a separate device from the display screen, each set of raw touch coordinates has to be converted to graphics coordinates (pixel locations). To provide the necessary conversions, we have written a calibration program, which is one of the very few programs in the system that is not started by touching the screen. The need for this exception is clear: recalibration is needed at those times when the touchpanel is not returning the correct coordinates. A button under each

of the six CRTs will start the calibration program on that screen. The program presents a pattern of 20 dots. The operator touches each dot, in whatever order he chooses. A least-squares algorithm is used to determine which dot the operator is touching, and the values returned by the touch panel are translated into this location. All of these translations are, finally, placed in the database under the entry for that screen. From then on, until the characteristics of the touchscreen change, touches will be correctly translated into coordinate positions.

3. Graphics software

One of our primary goals was to create a library of routines that would make it simple to add new process modules to our system. A process module is a program running on the VAX, through which an operator can interact with some equipment. We wanted it to be easy, for example, for physicists to query our database and to display information in whatever format they desired, without ever knowing the intricacies of the system. (All interaction is through the database, discussed below.)

Table 1 shows a short FORTRAN program fragment that does the following:

- Performs necessary initialization (including starting the mouse program for this screen)
- Places itself on the "wake-up list" for a change to a particular datapoint
- Puts two buttons on the screen
- Reacts properly to a button-touch or a change to the selected datapoint

One of the buttons will cause the "keypad" to be displayed on the screen (see fig. 2). This keypad will then accept a value from the operator and place this value in the database entry for the analog control of a particular equipment.

The first call is made to pminit (process module initialization): This initializes the new subprocess's task id and other parameters required by service subroutines. Next is tpinit (touch panel initialization): This routine initializes the touch panel on the CRT owned or shared by this process, so that a touch to that panel will wake up the process.

Then the integer variable "crtid" is initialized, so that the program knows which CRT it is running on, after which (optionally) a call is made to the mouse subroutine, which starts the mouse program on this CRT so that all future interaction with this program can be done either through the mouse or through touches to the screen itself.

After that, a call is made to subroutine addwake, on a particular equipment and channel. The result of this call is that after the program "goes to sleep" (see below), any change to this datapoint will "wake it up". (The program will also be awakened by a touch to the screen, since tpinit has been called.) The next line verifies that the addwake call has succeeded.

After initialization is completed, two buffers are defined using gradefinebuf so that "deferred-mode" commands can be sent to the screen. (Most commands are sent in deferred mode, making it possible to paint the entire screen at once, with one call to the DMA routine grasendbuf.) Buffers can be saved and reused, speeding up the redrawing of the screen. The third and fourth parameters to gradefinebuf deal with the "touch" facility. They can be set to zero, if no touch areas are to be defined.

The next group of subroutine calls define the picture to be drawn on the graphics screens and the touch-sensitive areas. The first one (grac1r) clears the screen. Next, the call to buttonpar sets up button parameters. This command will remain in effect until another call to buttonpar is made. The parameters to this call define the size, button color and text color of buttons written. A logical variable defines whether or not a new touch area is required for the buttons. Note that the picture of the button and the touch area are quite independent; buttons are often redrawn to change the color or text, with no need to redefine its touch area.

Now we are ready to define any buttons that may be required. A string variable is initialized with the label to be displayed in the first button and the call to writebut made. The first parameter to writebut is the desired location of the button on the screen. The actual position depends on the size defined in the buttonpar call. Three sizes are defined. Size 2 buttons, which we are using in this example, are 90 by 60 pixels, yielding a screen full of 8 rows of 7 buttons each, or 56 possible button locations. In this example, Location 11 is the third button in the second row. Size 2 buttons can have up to 40 characters of text in them (4 rows of 10 characters).

After another button is defined, the buffer is sent to the Lexidata controller (grasendbuf). The next call (to grawaitio) prevents the program from overwriting the buffers until the output to the Lexidata controller is complete.

The call to sys\$hiber, a VMS system service routine, suspends this process. A touch to any part of the screen will restart the process, as will a change to any datapoint (database value) that it happens to be watching. It is therefore necessary, when the process is awakened, that the process distinguish between the various possibilities and, if the program has indeed been awakened by a touch, determine its validity (on a button or not) and, if it is a button, which one.

The routines we have developed for this purpose make the solution quite simple. The first line after the call to sys\$hiber (the first line to be executed upon awakening), is a call to function test_mdt (test modified data-tag) for a particular equipment/channel--the one referred to in the addwake call (see above). If this call returns the logical value "true," the process will make a call to the get subroutine for the channel in question, which gets the value from the database and places it in the third parameter (y). The next few lines (through "end if") will write this new value on the screen, in red Size 3 letters (very large), starting at pixel location (60,60). Location (0,0) is the upper-left corner.

The call to touchcheck will return "true" if the screen has been touched. Then a call to readtou (readtouch) yields the "number" of the touch. A zero returned in "touchnum" indicates that the touch was not within the bounds of a currently defined button; whereas any other number indicates the number of the button touched. For this the buttons are numbered in the sequence in which they were declared.

Once the program knows which button has been pressed, it takes the following actions: The first button was the "exit" button, so the program acts accordingly. The second button indicates the desire on the operator's part to enter an analog value, so he is presented with the keypad (see fig. 2). The various parameters to the keypad select its colors, location, and limits of the acceptable input values. The input value is returned in the last parameter, rval. This value is then placed in the database by the call to put, (see below), and the program returns to the hibernating state.

Touch areas can be defined independently of buttons, making it easy to draw various icons, etc., that are to be touched when particular responses are desired. If the program is not hibernating when a touch is received, the touch is saved and the program will be awakened immediately upon its next call to sys\$hiber.

We now discuss the process for adding a mouse to our system. We chose to use an optical mouse from Mouse Systems Inc., in Santa Clara, California. The only unfortunate feature of this mouse is that it requires a small optically cross-hatched panel to run around on. This panel is not popular with our staff because it takes up valuable space in the console area, and is often to be found in a drawer when the mouse is not in use.

All movement of the mouse is relative to its previous location, and our software keeps track of the present location, maintaining a cursor on the screen in that position. The three buttons on the mouse can be set by the software to indicate whatever command is desired. One button, of course, must be usable to indicate a "touch" at the present location. The software simply places the present location in the database for that touchscreen when the "touch" button is pressed, and the rest proceeds in the normal way. That is, the hibernating program awakens, notes the touch, and proceeds accordingly. The second button is used to toggle coarse/fine movement of the cursor, and the third is used by different programs for various other purposes.

The touchscreen is most popular for general use, and the mouse is only used when a particularly fine (specific pixel) touch is required. Thus the mouse is used to define waveforms on the screen to control the 2.8-Mhz buncher or the orbit-bump magnets. Using the mouse, precise waveforms can be drawn simply and quickly. These waveforms can then be written to the CAMAC waveform generator to control this equipment. Naturally, this software is a little different from that described above for the use of buttons. In this case the program reads the X- and Y-coordinates directly from the database.

So far we have discussed analog input only with respect to the keypad. A more interesting alternative, available for all equipment, is the use of knobs. As distinguished from the old style control room, where there would be a knob dedicated to each analog control point, we have what we call "soft knobs." For each pair of screens there are four knobs. Each of these knobs

can be assigned to control any desired datapoint simply by touching the "assign-to-knob" button on the screen that has to do with the desired equipment. The name of the equipment and channel to be controlled, and its present control and read-back values will appear on the 40-character 1-line display next to the knob that has been assigned. Each knob is, of course, simply connected to a shaft encoder, which is connected to the VAX through a CAMAC module designed in-house. The assignments to a knob can be "stacked," up to 15 deep, and the various equipment "scrolled" into view, yielding much greater flexibility than is possible with many other systems. These units are described in more detail elsewhere [3].

4. The Database

As has been mentioned above, all communication between the process modules and the equipment takes place through the database. This design has many advantages, among which are portability, ease of programming, and generalized error checking and reporting. The run-time database includes, among other things, the present values for all channels, their CAMAC addresses, and any limits and convert codes that may be in effect. The control scheme is that the process modules (running on the VAX) place values in the database for a control channel on some equipment. This part of the database is migrated to the appropriate Instrumentation Subsystem (ISS) [1,2,3] together with the modified data-tags (mdt). The fact that the mdt is set causes the appropriate task running on that ISS to be awakened (much like a screen touch awakens a hibernating process on the VAX). This task then reads the present value for that channel and acts accordingly. The same scheme is used for input channels but the data is migrated in the reverse direction. The important part of all this is that all writing to and reading from the database is done through the use of the two routines mentioned above: get and put. The author of a new module is required to know nothing about the details of what he is doing other than the name of the equipment and the channel sequence numbers. Furthermore, all configuration changes are made within the database; therefore, it is not necessary to modify codes when this changes.

All values pass through the database on their way from equipment to display screen; thus, it is a simple matter to check each value to see if it is outside the safe range. Those seen to be beyond their Warning or Alarm limits are posted on the Alarm Screen. We plan to write an event logger, which will record the time and details of each such event.

5. The resulting system

Using the facilities described, some very useful and descriptive diagnostics displays, as well as many control screens, have been written. Some of these will be described in broad terms here.

Figure 3 shows a diagram of the Ring, with the activation protection monitors (APs) represented by dots. These radiation detectors monitor the spill in any location, and the resulting signal is represented on this screen by a bar graph, which changes from green to yellow to red as the spill increases. As the tuning of the PSR progresses it is often necessary to move the monitors in the tunnel. This display allows the location of the monitors on this screen to be easily changed together with the spill bar graph and the label. This is a case where the precision of the mouse is required. For each move, the upper left button on the screen must first be touched. Then, touching the element to be moved will cause it to be erased and then rewritten at the point of the next touch. These edits are temporary until the upper-right button is touched, when they are saved on disk.

Another invaluable diagnostic tool is the beam position monitor system (BPM). This system consists of 61 monitors at various positions in the beam pipe and a common multiplexer and signal-processing electronics. Figure 4 shows the display of the current data from the system. Another screen is used to select the monitors to be used and the order in which to display them. Up to 20 monitors may be selected at a time. This screen is also used to set up the operating parameters for the system. The horizontal and vertical positions of the centroid of the beam, relative to the pipe center, as measured at selected BPMs, are shown in fig. 4 in centimeters. Shown here is the histogram plot, with the newest reading displayed on the right for each monitor. The beam intensity (at each monitor) is displayed on the third graph. For the

convenience of the operators, the readings of all 65 APs are also displayed near the bottom of the screen as a histogram. This display has an example of a touch area without a button being shown. To change the vertical scale of each of the centroid graphs, the lower scale values of each graph should be touched (-0.05 in the case of the upper graph).

Another diagnostic that measures the beam is the wire scanner system. The current on a wire is measured as it passes through the beam. The graph of horizontal- and vertical-wire current as a function of position provides integrated profile information about the beam as well as the position of the centroid. Figure 5 shows the display of the results of a scan by LDWS04. It will be seen that the position of the centroid and the width of the beam for both sections has been calculated and displayed on the graph. This software will display the results of scans by up to six wire scanners and will also display previously obtained data. The harp system provides similar data but obtains it by placing an array of wires, 32 in each direction, in the beam. A fast multiplexer and ADC then digitize the stored charge on each wire after sufficient beam has passed. A data display screen similar to the one shown in fig. 5 is provided for the harp system.

A number of subsidiary systems are monitored by the control system and the current status is displayed to the operator on demand. One of these systems is the deionized water cooling system. Figure 6 shows the status display in its current form. The program generating this display determines through which pipes the water is flowing by reading the status of the pumps and valves from the database. These water pipes are then drawn so that water is seen to actually flow. This is done by rewriting the Color LookUp Table as described above. With this program running, the CPU usage for the water flow is less than 4%. Similar programs have been written for the ventilation system, the target cooling system, and the vacuum system. In some of these programs, more detail on particular subsystems can be obtained by touching that system on the top display.

At the injector to LAMPF, the linear accelerator that provides beam for the PSR, a chopper prepares the beam so that it has the correct waveform for successful injection and storage in the PSR. This chopping involves nanosecond and subnanosecond edges and waveforms. (To observe these

waveforms on an oscilloscope in the PSR control room, over a kilometer away, the oscilloscope must be close to the chopper.) A TV camera looks at the oscilloscope screen so that the PSR operator can observe the waveforms, but it is also necessary that the operator has full control over the oscilloscope and the multiplexer that selects the input signals. The oscilloscope used, a Tektronix 2465, has a GPIB interface and this is connected to a CAMAC GPIB controller on the PSR CAMAC crate adjacent to the chopper. The multiplexer is directly controlled by CAMAC. Figure 7 shows the display that controls this oscilloscope, and in use it is found to be only slightly less convenient than using the oscilloscope directly. In many of the buttons, the current value is displayed and touching the button will highlight it for a moment and toggle the value to the next one. Where there are more than a few values possible, touching the button will cause it and the two buttons labeled up and down to be highlighted. The up and down buttons can then be used to adjust the value to the one required and then touching the original button will remove the highlights.

Figure 8 shows the waveform-editor screen that enables the operators to create precise waveforms with the mouse. Figure 8 shows a rather simple waveform where the mouse has been used to add a third point to the waveform. Touching the detach button at the bottom of the display will make the modified waveform the new waveform that then can be further edited. At present, the use of straight lines to join the points is quite adequate and the waveform is smoothed by the electronics. Other buttons on the screen allow the operator to expand the display as required and to edit the waveform with greater precision. The horizontal axis shows start of injection (SOI) and end of injection (EOI) when it is on-scale. The numbers on the horizontal axis are microseconds from SOI.

Figure 9 shows the status screen, a VT220, that displays a brief summary of the current operating mode of the PSR, some key operating parameters, and messages from the operating staff. The terminal that generates this display is part of the PSR control console. Slave displays make this information available at a number of points on-site, and a copy of the display can be brought up at any terminal logged on to the PSR control VAX.

Lastly, for those who may not completely trust computers, TV monitors constantly view a variety of glowstick currents and full stripmaps from which the operator can determine if there is laser at some location, where it is in the beam-pipe, its size, and its shape. There are also several oscilloscopes in the control room other than the choppy scope mentioned above, which give the operators a computer-independent view of their equipment.

6. Conclusions

The human interface to the PDR control system has been most successful. In particular, the use of touch screens has satisfied over the early criticism of the idea, and the lack of keyboards on the table surface of the console has been accepted. For the few occasions where a small amount of alphanumeric entry is needed, a keyboard is drawn on the display. Where resolution greater than that obtainable with the end of a finger is needed, the mouse has met the need. Improvements to the human interface that might be considered are the following.

- Replace either the mouse with a mechanical device, or the console computer box with a touch-sensitive surface that the optical mouse will recognize.
- Upgrade the graphics system to a 1024 x 1024 resolution system that is now available. This upgrade will improve the quality of the displays.
- Expand the use of different colors to fully exploit the graphics hardware.

There are no features of the human interface hardware that we can no longer justify, in the light of experience.

Acknowledgments

A control system such as the one described is the work of numerous people. Many of these are acknowledged by the references, but one must also

acknowledge those who designed and constructed the console and its components. In particular, Gary Carlson designed and built the knob unit CAMAC interface as well as the knob units themselves. George Vaughn and Terry Weisgerber did much of the rest of the assembly work on the console.

References

- [1] P. Clout, S. Bair, A. Conley, R. Ford, M. Fuka and N. Greene, "The Proton Storage Ring Control System," IEEE Trans. Nucl. Sci. 30 (4), 2305 (1983).
- [2] P. Clout, "Proton Storage Ring Diagnostics and Control System," Proc. VII Meeting of the Int. Collaboration on Advanced Neutron Sources, Chalk River Nuclear Laboratory report AECL-8488, 128 (1983).
- [3] Peter Clout, Andrew Conley, Samuel Bair, Mary Fuka, Elsie Sandford, Robert Lander, and David Wells., "The PSR Control System," these proceedings.
- [4] P. Clout, "PSR Control Console Conceptual Design Document," Los Alamos National Laboratory, Accelerator Technology Division PSR Technical Note 88, (1981).

Table 1

Demonstration Program Fragment

```

program demol
implicit none

real*4      y
character*7  msg
integer*2    grabuflen, touchbuflen, crtld, mycrt, touchnum
parameter    (grabuflen = 2500)
parameter    (touchbuflen = 2)
integer*2    grabuf(0:grabuflen), touchbuf(5, 0:touchbuflen)
integer*4    stat, addwake, touchcheck, test_mdt
character*40  txt

c
call pminit
c
call tpinit
c
crtld = mycrt()
c
call mouseme(crtld)
c
stat = addwake(srtm02, 2)
c
if (.not.stat) type *, 'demol: Addwake failure.  stat = ', stat
c
call graopen(crtld)
c
call gralut(crtld, color16)
c
c
call gradefinebuf(grahuf, grabuflen, touchbuf, touchbuflen)
c
call graclr
c
call buitonpar(2, g_red, g_blue, g_touch)
c
txt = 'EXIT'                ! Text to be placed on a button
c
call writebut(3, %ref(txt))
c
      1234567890123456789012345678901234567890
txt = '  ENTER      VALUE      FOR      SRTM02  '
c
call writebut(4, %ref(txt))  ! Define a button in position #4
                             (immediately to the right of #3)
c
call grasendbuf(crtld, grabuf)

```

```

C      call grawaitio
C
C      20      call sys$hiber()
C
C      if (test_mdt(srtm02, 2)) then
C
C          call get(srtm02, 2, y)
C
C          write(msg, 100)y
C
C          100      format(f7.2)
C
C          call gradefinebuf(grabuf, grabuflen, 0,0)
C          call grarao(g_red, 60, 60, 3)
C              Grarao erases the
C              previous contents of the specified location, while
C              grasao simply over-writes.
C
C
C          call gratxt(7, %ref(msg))
C              This routine will write the contents of "msg"
C              on your crt, according to the parameters specified
C              in grarao (or grasao). The first parameter (of gratxt)
C              is the length (in characters) of the message to
C              be written.
C
C
C          call grasendbuf(crtid, grabuf)
C          call grawaitio
C      end if
C
C      if (touchcheck(crtid)) then
C
C          call readtou(touchbuf, touchnum)
C
C          if (touchnum .lt. 1) stop
C
C          if (touchnum .eq. 2) then
C              stat = .false.
C              do while (.not. stat)
C                  stat = keypad(crtid, 250, 250, g_blue, g_red,
1                  g_yellow, 0., 6., rval)
C              end do
C              call put(srtm02, 2, rval)
C          end if
C      end if
C
C      end if
C      goto 20
C      end

```


Figure Captions:

Fig. 1. A schematic diagram of the use of a Color LookUp Table to give a wide choice of colors from only four planes of video memory.

Fig. 2. The keypad as drawn on the graphics display to allow the operator to enter a particular number.

Fig. 3. The console display of the positions and current readings of the activation protection (AP) system monitors. Because it is quite normal to move the monitors in the ring tunnel as tuning progresses, the program driving this display allows the operator, with the touch screen or mouse, to move the dots showing the positions of the APs and the corresponding histogram and alphanumeric information.

Fig. 4. The Beam Position Monitor (BPM) data display. This shows position and intensity information from the selected monitors as well as a histogram display of the current readings from the 65 APs.

Fig. 5. The data display from a wirescanner.

Fig. 6. The deionized water system facility display. The current status of pumps and valves determines which pipes show water flowing.

Fig. 7. The oscilloscope control display which allows the operator to control a remote oscilloscope through CAMAC and a GPIB.

Fig. 8. The waveform editing screen which allows the operator to create and edit control waveforms that are applied to some of the PSR equipment. Waveforms can be up to 4096 points long and can be updated at rates of up to 2 MHz.

Fig. 9. The PSR status display. This is generated on a normal terminal and monitors around the facility repeat this information.

REPRODUCED FROM
BEST AVAILABLE COPY

Color Look-Up Table

*on dig. copy,
use initial p. only*

(Z)

13-SEP-1985 11:13:15.50

35.4			
7	8	9	-
4	5	6	E
1	2	3	E N T
0		.	
CLEAR			

RING APS

REPRODUCED FROM
BEST AVAILABLE COPY

Fig 3

REPRODUCED FROM
BEST AVAILABLE COPY

RESET

702
402
102

CONTINUOUS
MODE-
TOUCH
TO HOLD

SAVE

RECALL

FILENUM
6

TUNING
VALUES

SPARE

OFFSETS
ENABLED

RAW DATA
FILE
SAVE

TOUCH
FOR
TEXTPLOT

RETURN

4

LDWS04
C=-2.0
V= 3.0

Retract

Rescan

Return

Fig. 5

13-SEP-1985 10:29:22.14

PSR DEIONIZED WATER SYSTEM

REPRODUCED FROM
BEST AVAILABLE COPY

Original Not Found

13-SEP-1985 10:33:08.99

RESET SAVE/RECALL

TEK 2465 CONTROL
SWEET

CHOP

H MODE

A B

VERTICAL

ASWEEP

MAG
OFF

POS
-0.20

TRACE
1.00

B END
OFF

DELAY
FINE
4.00

DELAY
COARSE
4.00

INTENSE
FINE
1.00

INTENSE
COARSE
3.00

SPEED SPEED
50 USEC 1 MS

TRIGGER

MODE MODE
NORMAL TRIGABLE

SOURCE SOURCE
CH2 CH2

COUPLING COUPLING
AC AC

SLOPE SLOPE
PLUS PLUS

LEVEL LEVEL
0.20 0.00

UP DOWN

CH1 COUPLING MAG
FIFTY INPUT 2
ON VOLTS / DIV
2 Volts POS
3.00

CH2 COUPLING MAG
FIFTY INPUT 2
ON VOLTS / DIV
2 Volt POS
4.00

CH3 CH4
OFF ON
VOLTS / DIV
.5 .1
POS POS
-1.00 1.40

EXIT

REPRODUCED FROM
BEST AVAILABLE COPY

Copy Oscilloscope

13-SEP-1985 10:30:40.20

CAVITY VOLTAGE

EOI-LOC1
000

CHOOSE
SAMPLE
LOCATION

EXIT
NO SAVE

REPRODUCED FROM
BEST AVAILABLE COPY

ADJUST
HORIZ

SCALE
UP

SCALE
DOWN

EXIT
AND
SAVE

DETACH

SCROLL
MINUS

SCROLL
PLUS

SET
S01-VAL1
11976

SET
EOI-VAL1
6117

Sample Location is -200

slot: 929 val: 4725
Its VALUE is 0.00 (on Cav Volt Wave)

7/2 2.8 - 14Hz *[Handwritten signature]*
[Handwritten signature]

19

REPRODUCED FROM
BEST AVAILABLE COPY

PSR/WNR STATUS 13-SEP-1985 10:48

	BEAM	RATE	LENGTH		PSR CHOPPING PATTERN
SRBC:	OFF	4	200 us	Pattern Width:	200 ns
LEBC:	OFF	0	750 us	Countdown:	10
LDBC:	OFF	0	200 us	LBOG Width:	100 usg

RUN PERMIT	LINE-D MODE:	PSR-TUNE
	PSR MODE:	TUNEUP BS

LDCM01	0.05 us	RICM01	0.03 us
LDCM02	-0.05 us	RICM02	0.00 us
LDCM04	0.03 us	RICM03	0.01 us
LDCM05	0.03 us	RICM04	0.02 us

PSR TUNING RESUMES AT 0700 THURSDAY
LINE E OPERATION 2000-0700 NIGHTLY UNTIL THE WEEKEND
Schedule uncertain for rest of week due to target installation

19

PSR/WNR Status Screen