XEROX
PALO ALTO RESEARCH CENTER
Computer Sciences Laboratory
April 11, 1978

To: Orbit and Dover owners and debuggers

From: Bob Sproull

Subject: Orbit/Dover Test Software

Filed on [Ivy]<Spruce>OrbitTest.Press

This memo outlines the present software for testing and debugging Orbit and Dover. Although it is not extensive, it have proven adequate.

Orbit Hardware Diagnostic

An Orbit diagnostic program is saved on [Ivy]<Spruce> as the file OrbitTest.Run in the "dump file" <Spruce>Orbittest.Dm. You will also need the file OrbitTest.Parameters from the same dump file.

The program, invoked with the command "OrbitTest," has a very simple user interface based on the mouse. You are first expected to enter a "test number" (see below for a description of each test). You may execute the test by "bugging" the item "single step" or "repeat" with any mouse button. When you are finished and wish to proceed to another test, bug the "test number" item, and provide a new number.

Many of the tests require that parameters be provided. A standard set of parameters is described on the file OrbitTest.Parameters, which the test program will read automatically. Bug the "parameter set" item, and provide the number of the parameter set--this simply avoids having to bug each parameter and type it separately. You may, of course, enter parameters by hand: simply bug the entry in the parameter window and type in the value: octal values are typed with a trailing "b", decimal values with a trailing ".", hexadecimal values with a trailing "x". (If the value is printed in only one format, as in the test number, the default radix is the same as the printing format.)

Restarting tests. If you are obtaining funny results, you may wish to re-start the test by bugging the "test number" field and supplying the test number again. You will also need to re-establish the appropriate parameters. Re-starting the test will sometimes reset some Orbit or adapter state that is crucial to the proper operation of the test.

Adapter cable. Some of the tests require that the adapter cable be jumpered so that adapter commands are looped back as adapter status signals. It is helpful to have a connector handy with the jumpers permanently installed.

Test 0. This test checks whether Orbit can be reset properly, and requires that adapter power be off or that the cable be jumpered. It will signal an error if INCON=0, BEHIND=0, stableROS=1, badROS=0 or IACS=0 after the reset. A tight loop will reset Orbit as fast as possible.

Test 1. Check the refresh timer and branch logic for discovering whether refresh is needed. The test prints out an estimate of the number of microseconds between "ticks" of the refresh timer (it is not very precise). If the refresh logic does not se to operate properly, an error message is printed: Refresh timeout -- refresh needed indication is absent." This will occur if the refresh timer does not tick within 600 ms. The detailed test:

```
Do steps 1-2 1000 times in order to time it carefully:
   1. Clear the refresh timer (CLRFRESH)
   2. wait until OrbitHeight branches, indicating refresh needed
```

Test 2. Send individual commands to the adapter, and report status. If the cable is jumpered, you should expect status word 0 to be the same as the ROS command, status word 1 to be the complement of the command, and status words 2 to 11 to be 0. If the adapter is connected and powered up, the interpretation of the status bits will of course be determined by the adapter. Each time the loop is repeated, the XOR value is XORed with the ROS command. A tight loop will repeatedly send the command shown on the screen.

Test 3. Check adapter command/status logic with cable jumper installed. This test cycles 1000 times with random numbers, and expects the jumper to arrange that the status word 0=the command, status word 1=complement of the command, and the remainder of the status words are zero. If this condition is not met, an error is indicated. A tight loop will loop sending the command very fast.

Test 4. Check FA logic. This test lets you set various values of FA and make sure that the output addressing logic is working properly. FA values of 0 to 377b should check out properly. An error is indicated if the wrong number of words is read (should be (400b-FA)*16), or if BEHIND fails to set. The basic test is to set FA, and then read words with OrbitOutputData until a buffer switch happens.

Test 5. Read buffer memories and display on the screen. Each time the test is repeated, a different buffer is read. The main point of the test is to verify that, aft several iterations, the memory becomes "white." An error is indicated if the buffers have switched at the wrong time (FA is set once whenever it changes, and thereafter Orbit is expected to stay in synchrony).

Test 6. Set height. This test simply sets the height register, and makes no error checks. A tight loop will set the register repeatedly.

Test 7. Set X, Y. This test simply sets the x and y registers, and makes no error checks. A tight loop will set the registers repeatedly.

Test 8. Set the Ink memory. This test sets the Ink memory from a vector of 16 values (vector no. 0 is all 0, vector no. -1 is all -1's, vector no. 1 is 16 words, each of whi 1 (i.e., 15 zeroes and 1 one bit)). A tight loop will set the memory quite fast, loopin through all 16 x locations.

Test 9. Set DBC and width, and read them back. An error is indicated if the stored and re-read values do not match or if IACS does not come on. A tight loop will execute the register-setting command repeatedly.

Test 10. Reads DWC. The test resets Orbit, and then reads DWC. An error is caused if it reads a non-zero value. A tight loop will cause repeated reading. (Note: The proper counting of DWC is checked by test 12)

Test 11. Reads DBC and width, and displays them. A tight loop will cause repeated reading.

Test 12. This is the main test that checks the image-generation parts of Orbit. It is usually used in conjunction with the "parameter-setting" features to set up vast amounts of state. The input values are:

X:	X coordinate of first scan-line of character
Y:	Y coordinate of bottom of character
Width(minus1):	Width-1 of character
Height(negative):	-Height of character, in bits
FA:	Setting of FA for reading buffers back into Alto
FontVector:	"Vector" to use as font data
InkVector:	"Vector" to use as ink data
dX:	Amount by which X should be incremented each iteration
dY:	Similar increment for Y
Tasking:	See below

The tasking flag is 1 if Orbit is to be run in the normal fashion (TASK instructions executed in the inner loop as font data is being passed to Orbit) or 0 if it is to be ru without tasking (helpful in certain cases for debugging because ’scoping is easier).

The test feeds the specified character to Orbit, and then reads back various registers (DWC, DBCWID, IACS) and checks to verify that they have the proper ending values. Finally, the test reads the other buffer memory (the one that does not contain the character most recently imaged) and displays it on the Alto screen. A tight loop will repeatedly ship the character to Orbit, and will not do checking or screen updating. After each iteration, the value of INCON displayed is the value when the character was shipped to Orbit (i.e., INCON=0 means that buffer A received the character, and B was displayed on the screen).

There are several "standard" parameter sets that are used with Test 12:

1. Upper case A which moves across the screen, changing X and Y as it goes.
2. Vertical line which moves to the left across the screen
3. Slanted line that moves to the left and up and down
4. 16x16 black square that moves to the left
5. 16x128 black rectangle that moves to the left
6. 16x16 black square that moves to the left (checks shifter)

Test 13. Adapter register test. This test (slowly) sends all possible commands to the adapter (requires 16 passes to complete a full cycle), and checks that the registers in the adapter are updated properly by checking the status coming back. (Warning: When the test stops, the "ROS Command" display contains the value of the command word that is about to be sent, whereas the status will show the last command actually sent.) An error will be caused if some register in the adapter is not responding properly.

Test 14. This test is designed to send the adapter a sequence of commands and to wait specified amounts of time in between commands. The "Command vector" is a vector of pairs of entries: the first of the pair is the command to send; the second is the number of milliseconds to wait before proceeding.

Test 15. This test is intended to exercise the adapter data-fetching logic. It puts th adapter in test mode, and therefore generates a fake line sync, a fake bit clock (with 4096 pulses/scan-line). The basic test loop is:

1. Do a buffer reset
2. Enable Orbit to send packets the adapter
3. Start page sync (with TestPageSync)
4. Wait for SendVideo to come on and then go off again,
   counting Orbit buffer switches as we go.
5. Disables Orbit sending of packets
6. Re-synchronizes Orbit buffers

The number of scan-lines that should be generated is 4*(4096.-VideoGate). Consequently, there should be (4*(4096.-VideoGate))/16 Orbit buffer switches.

This is a good test for "scoping" the adapter video logic. Put a non-zero number in "Do video stripe?" to enable a single band of video, 4 bits wide, near the end of the video data (which may be of whatever length, depending on FA).

Test 16. Test for bit-clock stability in the adapter on a working machine. This test is not an Orbit test, but rather tests whether the start/end of scan detectors, bit cloc generator and servo are operating properly. For this test, Orbit must be connected to a working ROS with the "TTL" adapter connected, and with a ROS installed (and beam interlocks, if any, closed).

The test lets you provide a bit-clock frequency and observe stability with an oscilloscope. The test requires specifying a time period for a scan-line (650 microseconds, the default, is appropriate for our systems) in order to set the polygon speed appropriately. The number of facets is also required (32 for Dover and Sequoia, 12 for Pimlico). You should specify the bit-clock frequency you want to test, in units of .1 MHz (e.g., 135 means 13.5 MHz). Note that this refers to the raw bit clock frequency; remember that some printers will divide this down by a power of 2.

The scope should be hooked up as follows: display on one channel (and trigger from) the signal LineSync. On the other channel (alt sweep), display EOLCount’, the signal that indicates the bit clock counter has expired. Both signals are available on the back plane.

For any given frequency, the bit clock will be in one of three states:

1. It is servoing, i.e., the frequency is within the range of the operation of the bit clock servo. In this case, the two scope signals will look the same. To observe what happens when the bit clock is not servoing, choose a very low frequency (e.g. 5 MHz): note that EOLCount’ falls before end-of-scan, i.e. the bit clock cannot go slowly enough. You may also try a very high frequency (e.g., 29 MHz), and observe that EOLCount’ falls after end-of-scan (or may never fall).

2. It is dynamically unstable. The error signal to the servo system is wavering all ove the place. This can be observed by watching the falling edge of EOLCount’: if it waver in time, the bit clock is unstable.

3. It is statically unstable. If line sync is stable when the beam is forced on solid, reverts to detecting every other scan line when the beam is turned off and left to servo by the "end scan beam on" method of forcing beam on for start/end detection. To test this, try "single stepping" the test: you will notice that the "beam on" indication alternates between forced on (1) and left to servo (0). If the LineSync trace divides by two (i.e., becomes twice as slow) with beam off, the servo is statically unstable. This situation can perhaps be improved by making the adapter use LeadSync; or perhaps we should make a change so that EndScanBeamOn was turned on earlier.

Parameter File

Index of vectors:

Test	Number	What
all	0	16-word vector of words=0
all	-1	16-word vector of words=-1
all	1	16-word vector of words=1
12	64	128-word vector of words=-1
12	65	54-word vector for upper case A font (height=48, width=18)
12	66	54-word vector for 16x16 square in white (height=48, width=18)

Index of parameters:

Test	Number	What
4	1	FA=0
4	2	FA=370b
9	1	DBC=0, Width-1=0
9	2	DBC=17b, Width-1=7777b
12	1	Upper case A (height=48, width=13, x=0, y=7720b, FA=300b, FontVector=65, InkVector=-1, dX=1, dY=-1)
12	2	1-bit vertical (height=16, width=16, x=0, y=7760b, FA=100b, FontVector=1, InkVector=-1, dX=0, dY=-1)
12	3	1-bit slant (height=15, width=16, x=0, y=7760b, FA=100b, FontVector=1, InkVector=-1, dX=1, dY=-1)
12	4	16x16 square (height=16, width=16, x=0, y=7760b, FA=0, FontVector=-1, InkVector=-1, dX=0, dY=-16)
12	5	128-bit black slug (height=128, width=16, x=0, y=7760b, FA=0, FontVector=64, InkVector=-1, dX=0, dY=-16)
12	6	16x16 square (height=48, width=13, x=0, y=7720b, FA=0, FontVector=66, InkVector=-1, dX=0, dY=-1)

Format of OrbitTest.Parameters. The best way to see how it is put together it to look at it, and mimic its contents. An entry is preceded by the noun VECTOR: or PARAMETER:. Each is followed by two numbers: the first is the test number to which it applies (or ALL if it applies to all tests); the second is the identifying number of the parameter set (or of the vector set) within that test. Note that octal is the default radix for all numbers. Characters between the character $/ and CR are ignored.

For a vector, we give the total number of elements in the vector, followed by the numbers that are to be placed in the vector.

For a parameter set, we provide pairs: parameter number, value. The parameter numbers are best discerned by looking at previous examples, or by consulting the function OrbitSetP in OrbitTestUtils.Bcpl, saved in OrbitTest.Dm.

Handy things to remember for now. Test 14, parameter set 1 should set up the adapter and Dover for 350 bit/inch operation (it also sets a large delayed line sync so that quad pattern printing works). The polygon motor should begin spinning, the bit clock should synchronize, etc. If you want to force the beam on for checking sync, send 17712 to the adapter (use Test 2 to send individual commands).

Here are other parameter sets for Test 14 (all except 1 and 2 set the delayed line sync the way Spruce sets it):

Set No.	Engine	Resolution	Scan line mod made (serial no’s ge #400)
1	Dover	350	No
2	Dover	350	Yes
3	Dover	384	No
4	Dover	384	Yes
5	Pimlico	350	Yes
6	Pimlico	384	Yes
7	Sequoia	350	No
8	Sequoia	350	Yes
9	Sequoia	384	No
10	Sequioa	384	Yes

Dover Hardware Diagnostic

There is no Dover diagnostic program per se. Several of the functions of OrbitTest can be harnessed to test out various parts of the adapter hardware. However, there is a simple printing program that will print unformatted text files in one font. The program, Orbit.Run, is saved in <DOVER>OrbitTest.Dm. You will probably want some fonts that can be found in the same dump file: Demo350.ac (350 bits/inch) and Demo300.ac (300 bits/inch).

Initialization. The Orbit program builds a temporary file that must be initialized with a font. The initialization procedure is invoked as follows:

```
   Orbit/i fontname
```

where fontname is the name of a file on your disk that has the same format as CDtemp produced by PrePress (see PrePress documentation for more information). The fonts Demo350.ac and Demo300.ac meet this requirement.

To install Orbit for printing on Durango, use "Orbit/ai fontname".

Printing. The simple printing commands are:

```
   Orbit/xxx filename
```

where xxx are possible switches, and the filename specifies what is to be printed. If a switch is a digit, this is the number of copies to print. So

```
   Orbit/6 demo.txt
```

will print a 1-page demonstration file, demo.txt.

If you include the "d" switch, it leaves the display on and prints > every time a sheet is fed. The switch "r" will "reprint" the previous file (i.e., you can omit the filename

The switch "x" will not print at all, but will set up the adapter in "test mode" to simulate the exact situation (about 280 microseconds per scan line) so that the video can be scoped. This switch will "print" the page 1000 times (by turning on TestPageSync), so that you can get a bunch of good looks at it with the scope. Simple video patters (such as a single horizontal line going the full width of the page) are particularly effective here. See below for how such lines can be produced.

(Additional switches that you probably don’t need to know about are given below. They are mostly for reference:

/I filename	Install new font and exit (see above)
/digit	Copies (see above)
/R	Reprint (see above)
/D	Leave display on (see above)
/X	Adapter test mode (see above)
/L	Limit text line length
/M	Use maximum buffering strategy
/T	Two up
/O	Only print one page
/S number	Line spacing change
/AI filename	Install for Durango
/B	Read bin-list from Orbit.Bins (Durango only)

/N Inhibits setting ROS adapter parameters.

Uglies. Unless /N is included, the program always sets up Dover to print at 350 lines/inch, regardless of the font you provide. (In the case of Durango, the program does not touch the adapter settings at all; they must be established correctly with some other program.) If the ROS serial number is > #400, a page delay value is used that assumes the page delay modification has been made (see OrbitTest parameter descriptions.)

Making the Orbit demo font. The fonts can be made in the standard way with PrePress. However, if you want the demonstration font with the Logo, Pellar signature, and the "Woman" halftone, you can make it as follows:

1. Bring <pressfonts>Helvetica.Sd to your disk as SdTemp.
2. Add in the splines for the logo and signature by executing "PrePress Readsf/u 2/x 2/y demo.sf". The /x and /y terms are needed to "blow up" the signature and logo appropriately.
3. Convert the font for a particular size with PrePress. For example, "PrePress convert 12/p 3000/d".
4. Merge in the woman character with the command "Mergeac CdTemp Woman.ac Demoxxx.ac". This reads CdTemp (left behind by PrePress convert) and adds the characters in Woman.Ac (only one of them) on the end of the font, and writes the new font in Demoxxx.ac.

Embedding special commands in the text file to be printed. There are some arcane methods for including "directives" in the text file to get various effects. Consult Demo.txt for an example if the discussion below is terse. There are three control characters (^A, ^B, and ^C) that receive special interpretation.

A coordinate system is established for a page: (0,0) is at the lower left corner of a protrait page. X is to the right, Y up. Numeric arguments to control directives can be in one of two forms: an integer, terminated with a comma or semicolon; or a decimal number (with a decimal point, e.g. 4.0), terminated with a comma or semicolon. In the first case, the number is taken literally; in the second, it is multiplied by the resolution, in bits/inch, to get the number to use. The idea of the second format is to permit easy specification of dimensions in inches, without having to compute the corresponding dimension in bits.

^A. Puts out a black "rectangle"; a very narrow rectangle looks like a line. There are four arguments: ^An1,n2,n3,n4; puts out a rectangle with left edge at n1, width n2, bottom edge at n3, height n4.

^B. Changes current position on page for subsequent text characters. There are three arguments: ^Bn1,n2,n3; sets current x position to n1, current y position to n2, and left margin to n3 (only if n3 is not zero).

^C. A single argument is used as an offset to add to every subsequent ASCII character read from the file (except for control characters). The reason for this is that the "woman" character is numbered 177 in Demoxxx.ac, so the sequence ^C62;A^C0; will print character 177b=62.+101b.