VisiCom Vigra VGS-882
Graphics Server

In Stock

Used and in Excellent Condition

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source
for quality pre-owned
equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Plane masking vs. acceleration features

Note: You can use plane masking with the VGS-882, but not if you have also enabled acceleration (which is on by default.) Use the standard X server arguments (such as -cp2) to disable acceleration if you need to use plane masking.

Miscellaneous questions and answers

Q. Can I use the overlay memory with X?

A. The overlay memory is only used with the terminal emulator. Use with X is not supported.

Q. What is the status of VGS-882 support for Solaris 2.x?

A. Version 3.1.5 was released in July 1996. New features include:

- The postinstall script runs correctly instead of looping forever.
- The rewritten driver handles multiple boards correctly.
- All new configuration script is more helpful and groks multiple boards.
- The package contains a newer and more complete set of fonts.
- The new driver does not gratuitously hammer the host CPU.
- The new package gives significantly higher performance.
- The "loadvgs" script recognizes multiple boards and offers usage help.
- Most importantly, it's all checked into CVS and buildable from source.

It is available from the Vigra ftp server: see the INDEX at ftp.vigra.com/pub/vgs The files you need begin with "vgs-3.1.5-solaris2"

Q: In attempting to run several VGS-882 cards under Solaris, the system seems to be creating device entries out of sequence.

A. This is due to a peculiarity in the way Solaris numbers boards.

Basically, any time you tell it you have a VGS board at some VME address, it allocates a board "instance" for you; just a number.

The very first board you create is /dev/vgs0, for example. If you re-jumper this board to be at another address and tell the kernel about the new address, it DOESN'T REDEFINE vgs0. Instead, it creates a brand new /dev/vgs1 at the new address. The old vgs0 sits around forever.

So it's possible that you created vgs0, vgs1, vgs2 whatever, then moved things around and created some more. There is really no way for the configuration program to know what number the kernel will assign to the new boards, so it assumes that they are number from 0 on up. If boards have moved, this may not be the case since there may now be gaps in the numbering.

The Solution:

There are two ways you can approach fixing this problem:

1. Tell the VGS configure program about the correct device names. (It assumes that the boards are numbered sequentially starting from zero.) First, use "dmesg" to find out what instances the kernel assigned to the devices. You will see lines such as these:

```
NOTICE: VGS Solaris 2.x device driver - V1.1.0  (C)1996 Visicom Labs.
NOTICE: vgs0: VME address 0x9a000000, level 5, vector 0xc9
NOTICE: vgs2: VME address 0x9b000000, level 5, vector 0xca
NOTICE: vgs3: VME address 0x9c000000, level 5, vector 0xcb
```

Note the vgs identifiers, which may not be 0 through 2 (notice how it skips vgs1 above). Then run

```
/opt/VIGRAvgs/bin/configure
```

and make sure that all the vgs boards (vgs0, vgs1, vgs2) have the corresponding correct device name, for example:

```
/dev/vgs0 /dev/vgs2 /dev/vgs3
```

-or-

5. Flush out all the old instances and start over from 0. The file that keeps track of these things is /etc/path_to_inst. Follow these steps:

   1. Uninstall the VGS driver from the kernel:

      ```
      /usr/sbin/rem_drv vgs
      ```

   2. Remove old vgs minor nodes:

      ```
      rm -f /dev/vgs*
      ```

   3. Clear out instance numbers by removing any vgs line from path_to_inst:

      ```
      grep -v "/vgs@" /etc/path_to_inst > /etc/path_to_inst.new
      mv /etc/path_to_inst /etc/path_to_inst.old
      mv /etc/path_to_inst.new /etc/path_to_inst
      ```

   4. Reinstall the driver:

      ```
      /usr/sbin/add_drv -m '* 0666 root sys' vgs
      ```

   5. Re-run /opt/VIGRAvgs/bin/configure and verify that the device names are all /dev/vgs0 and up.

I hope this clears things up. Sorry for the runaround but this is just one of those Solaris things...

Q: I have a NEC LCD flat panel display that supports 1280x1024. How can I use it with the VGS-882?

A: The exact details vary among operating systems; see the VGS-882 installation booklet for your specific software (available from Vigra).

Before even installing the VGS882, make sure that the DIP switches are set properly for the terminal emulation mode. This information is in the hardware installation guide.

Then install the board as specified in that manual.

After that, you need to select the appropriate installation booklet and follow the software installation instructions there.

For the display being discussed (NEC LCD1280) you will need to modify the vg882.db file to activate the command:

```
video = 1280x1024-60-nec
```

The token on the right hand side of the = keys into another file in the same directory called "vmodes.db" you can use to create parameters for special video modes.

VGS882 Front panel LEDs

The green LEDs on the front panel of the VGS-882 are arranged and labeled as follows:

```
|         |
|  o   o  |
|  4   2  |
|         |
|  o   o  |
|  3   1  |
|         |
| KEYBOARD |
|  ......  |
```

displaying a 4-bit quantity (i.e., LED labeled 1 represents the LSB (2^0), 4 represents

2^3)

1. When the board powers up, it goes through 6 stages of reset as shown in the table below (under LED USAGE). If it passes all the internal tests, it will display 0110 steadily. Otherwise it will halt and display an error indication as described further down in the table.

   If terminal emulation is ON, it executes these 6 steps in less than one second. If terminal emulation is OFF, you can see the LEDs count up. It will take about 4 seconds in that case.

2. When running the internal X server, LED 1 and 3 will be on, and LED 4 will blink at about 1/sec. Mouse activity will cause LED 4 to flicker.

Here are the meanings of the lights as displayed during the normal power up ("reset") cycle and the error codes displayed in case of failure.

```
*       LED USAGE:
*
*       The LEDs indicate the various stages of the reset code.
*           If a test in that stage fails, the LEDs blink quickly.
*
*       Binary   Decimal  Description
*       ============================================================
*       0000     0        Reset
*       0001     1        EPROM checksum test
*       0010     2        Cache flush, jump to kseg0 (cached)
*       0011     3        DRAM tests
*       0100     4        Main VRAM tests
*       0101     5        Overlay VRAM tests
*       0110     6        Jump to C main( ) routine
*
*       In the event of an unexpected exception, the LEDs blink
*       slowly with the following code:
*
*       Binary   Decimal  Description
*       ============================================================
*       0000     0        External interrupt (all LEDs remain off)
*       0001     1        Debug exception
*       0010     2        Undefined
*       0011     3        Undefined
*       0100     4        Address Error Exception (load or instruction)
*       0101     5        Address Error Exception (store)
*       0110     6        Bus Error Exception (instruction fetch)
*       0111     7        Bus Error Exception (data load or store)
*       1000     8        SYSCALL Exception
*       1001     9        Breakpoint Exception
*       1010     A        Reserved Instruction Exception
*       1011     B        Coprocessor Unusable Exception
*       1100     C        Arithmetic Overflow Exception
*       1101     D        Undefined
*       1110     E        Undefined
*       1111     F        Undefined
```