PIClib/RAYlib AGENDA

Overview

- Hardware (15 min)
  - Pixel Machine
  - Sun Platform
- Initializing the Machine (20 min)
  - Setting the Environment
  - Hyptools
- Programming Environment (20 min)
  - PIClib
  - RAYlib
  - DEVtools

BREAK (15 min)

AGENDA, cont'd.

PIClib/RAYlib (90 min)

- Environment/Initialization
- Directory Structure/Important Files
- Compiling and Linking a Demo Program
- Library Description by Function
  - Program Examples
- Summary of Helpful Hints

Lunch (60 min)

Hands on PIClib Programming (60 min)

Break (15 min)

RAYlib Introduction (40 min)

Programming (rest of day)

PIClib OVERVIEW

- high resolution, 24-bit color
- lots of interactivity
- fast point, line and polygon rendering
- very high level of image quality, if desired
- easy to use, easy to learn
- can use as frame buffer or graphics device
- conspicuous goodies - texture mapping, antialiasing and lots of off-screen memory

SUN COMPUTING ENVIRONMENT

- Environmental Variables

HYPER_MODEL	964, 964dX, 964dn, 964n
HYPER_PIPE	serial / parallel
HYPER_UNIT	0, 1, 2, ... 8
HYPER_PATH	/usr/hyper

- $PATH and $MANPATH changes
- Adding system commands and demo executables to path

  /usr/hyper/bin
  /usr/hyper/demo/piclib/bin
  /usr/hyper/demo/raylib/bin

- Adding Pixel Machine man pages to MANPATH

  /usr/hyper/man

SUN COMPUTING ENVIRONMENT

- C shell modifications - add *source .hyper_login* to *.login*
- */usr/hyper/.hyper_login*

```
setenv HYPER_MODEL      964dX
setenv HYPER_PIPE       parallel
setenv HYPER_UNIT       0
setenv HYPER_PATH       /usr/hyper
set path = ( ${path}
        $HYPER_PATH/bin
        $HYPER_PATH/demo/piclib/bin
        $HYPER_PATH/demo/raylib/bin )
if ( $?MANPATH ) then
   setenv MANPATH     ${MANPATH}:$HYPER_PATH
   else
    setenv MANPATH     /usr/man:$HYPER_PATH/m
    endif
```

SUN COMPUTING ENVIRONMENT

- C shell modifications - add *source .hyper_cshrc* to *.cshrc /usr/hyper/.hyper_cshrc*

```
alias   hypmodel    'setenv HYPER_MODEL *'
alias   hypipe      'setenv HYPER_PIPE  *'
alias   hypunit     'setenv HYPER_UNIT  *'
alias   hypath      'setenv HYPER_PATH  *'
```

- Also: Korn Shell - use *.hyper_env* and *.hyper_profile*

INITIALIZATION AND BOOTING

- Initialization
  - hypinit, hypboot -p and hypboot -r
  - *booting* the machine loads code into the pipe and pixel nodes
  - reboot machine when switching between PIClib and RAYlib
  - initialize and boot when changing any of the hyper variables
- System Commands
  - hypinit, hypstat, hypenv, hypid
  - hyplock, hypfree, hypload, hyprun
- PIClib Program Creation
  - *#include "/usr/hyper/include/piclib.h"*
  - *cc src.c /usr/hyper/lib/piclib.a -lm -o src*
  - floating point accelerator, co-processor and profiling versions
  - can link both PIClib and RAYlib together in same C program

SYSTEM COMMANDS

- *hypenv* - displays current setting of environment variables
- *hypfree* - releases a locked machine
- *hypid* - displays node ID data
- *hypinit* - initializes the hardware
- *hyplock* - locks a machine
- *hypstat* - displays system status

SYSTEM UTILITIES

- *picboot* - loads PIClib software into Pixel Machine
- *picgamma* - sets the color lookup tables
- *picinit* - initializes and resets the PIClib library
- *piclear* - clears front and back buffers to black
- *picrt* - linearizes monitor response

PIClib COMMAND SETS

- Control Functions
- Drawing and Rendering Primitives
- Modeling, Viewing, and Projection Transformations
- Lighting, Shading, and Depth-Cueing
- Viewports and Buffers
- Overlay Control
- Mice and Cursors
- Text and Raster Ops
- Frame Buffer and Data Memory Manipulation
- Antialiasing and Filtering
- Texture Mapping
- Picking and Selecting
- Video Control

CONTROL FUNCTIONS

- Initialization
  - initialize - start up hardware, set default variables and modes, invoke new signal handler
  - resume - start up hardware, leave all variables and modes untouched
  - exit - halt machine, detach mouse, reinstate standard signal handler
- Pipe Nodes
  - swap pipe - send data to *alternate* pipe
- Pixel Nodes
  - vertical synchronization
  - processor synchronization

DRAWING AND RENDERING PRIMITIVES

- Basic Primitives
  - points
  - lines
  - polygons - colors, normal vectors and texture indices at vertices
  - integer and floating point
  - 2-D and 3-D
- Primitives
  - rectangles
  - arcs
  - circles
- High Level Objects
  - cubic curves
  - bicubic patches
  - quadrics
  - superquadrics - ellipsoids, toroids, and hyperboloids
  - modes - point, line, polygon and texture

TRANSFORMATION COMMANDS

- Two Current Matrices [*MV*] and [*P*]
- Two Current Stacks [*MV_STACK*] and [*P_STACK*]
- Transformation Stack Control [*MV*]
  - put - current matrix, identity matrix
  - get - current matrix, inverse matrix, normal matrix (inverse transpose)
  - pre and post-multiply current matrix
  - push and pop between stack and current matrix

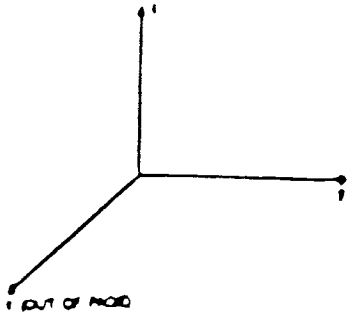
TRANSFORMATION COMMANDS

- Modeling Transformations [M]
  - get - put identity matrix, then get [MV]
  - rotate - incremental, absolute, and arbitrary axis
  - translate - incremental and absolute
  - scale - incremental and absolute
  - incremental is optimized
- Viewing Tranformations [V]
  - get - put identity matrix, then get [MV]
  - Camera view, Lookup view, Lookat view, Polar view

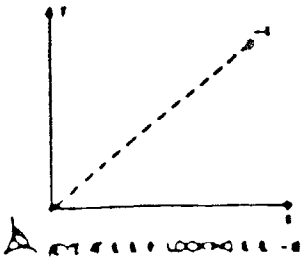
PROJECTION TRANSFORMATIONS

- Projection Transformation Stack Control [P]
  - put - current matrix, identity matrix
  - get - current matrix, inverse matrix
  - pre and post-multiply current matrix
  - push and pop between stack and current matrix
- Perspective Projections [P]
  - 3-D perspective viewing pyramid - view via pyramid base center
  - 3-D perspective viewing window - view via lower left of pyramid
- Orthographic Projections [P]
  - 3-D orthographic
  - 2-D orthographic

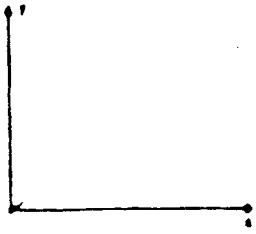
Coordinate Systems



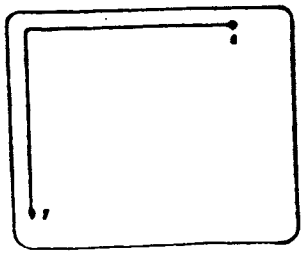
World Space [M]



Eye Space [V]



Screen Space [P]



Pixel Space

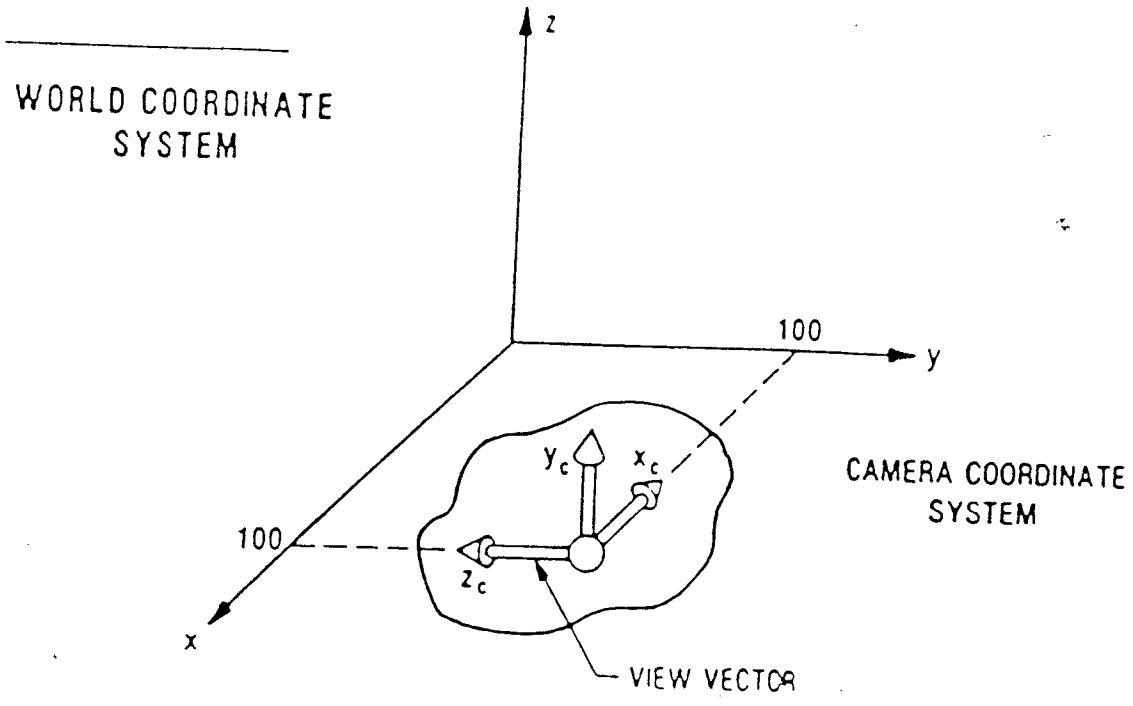


Figure 3-8. PICcamera_view(100.0, 100.0, 0.0, 0.0, 0.0, 0.0)

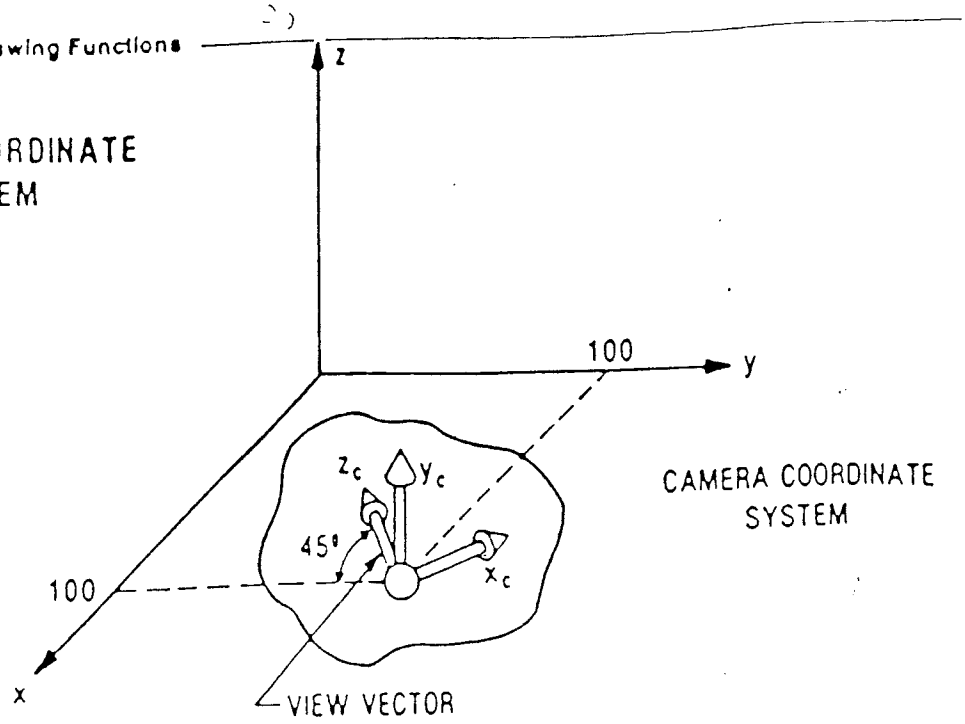


Figure 3-9. PICcamera_view(100.0, 100.0, 0.0, 45.0, 0.0, 0.0)

LIGHTING

- Light Properties
  - type (directional, point, spot)
  - intensity
  - position
  - direction and other attributes
  - beam concentration
- Lighting Controls
  - array of lights, selected lights turned on or off
  - directional, point and spot arrays (50 lights of each type)

VIEWPORTS AND BUFFERS

- Viewports
  - specified in pixel coordinates - put / get
  - viewport stack - push / pop
  - specify current drawing color - RGB values [0.0 - 1.0]
  - clearing rgb clears to current rgb color
  - depth range associated with each viewport - put / get
  - clearing z-buffer clears to *far* z-depth
- Buffer Control
  - double buffering - enable / disable, get current buffer, get current mode
  - front/back buffers, front displayed, back used for drawing
  - swapping buffers - exchanges front buffer pointer and back buffer pointer
  - extended buffering - see frame buffer and data memory manipulation

OVERLAYS

- The *Alpha* Plane
  - overlay control - enable / disable
  - specify *alpha* color - (0 <= *alpha* <= 255)
  - clearing *alpha* plane clears to current *alpha* color
  - overlay mode 0 - disable overlays
  - overlay mode 1 - display rgba (*alpha* < 255) or invert all rgb values (*alpha* = 255)
  - overlay mode 2 - (*alpha* >= 128) display *alpha* channel only, else display rgb

MICE AND CURSORS

- Mouse Control
  - attach / detach mouse process
  - get button, get locator *xy*, get valuator
  - enable / disable event queueing
  - get event and value - mouse, keyboard button sampling
  - event queue control
  - process spawned to handle *real-time* events
  - PIClib and event monitor processes communicate via shared memory
- Cursor Control
  - user definable cursors - 32 by 32 bit array
  - enable / disable cursor display
  - position cursor
  - default cursors in */usr/hyper/cursors*

TEXT AND RASTER OPS

- Text
  - Vector fonts - *Hershey* fonts
  - Raster fonts - suntools fonts
  - open a font file, select a font type, write a string using current font
  - vector fonts can be used in 2-D or 3-D
- Raster Ops
  - add to all pixels within a viewport - used for dropped shadows
  - multiply to all pixels within a viewport

RASTER PRIMITIVES

- Atoms
  - directional light source vector
  - specify surface characteristics - ambient, diffuse and specular coefficients
  - renders much *faster* than polygonally generated spheres
  - phong shaded
  - should be used as a building block for 3-D modeling

FRAME BUFFER AND DATA MEMORY MANIPULATION

- Scan Lines
  - put and get
  - packed rgba and abgr, unpacked rgb and rgba, encoded rgb
- Buffer to Buffer Copies
  - front rgba to back rgba
  - back rgba to off-screen rgba
  - off-screen rgba to back rgba
  - on-screen z to off-screen z
  - large image scrolling and rgba flipbooks
- Image or Floating Point Data Write
  - broadcast data to Dynamic RAM - z-buffer or texture maps
  - broadcast data to Video RAM - texture maps
  - image *cache* memory - textures are replicated in every node

ANTIALIASING

- Antialiased Lines
  - enable / disable
- Antialiasing by Supersampling
  - specify filter kernel contents, kernel size and x and y scale
  - enter supersampling pass, exit supersampling pass
  - filter can be box, pyramid, gaussian ...
  - some image processing with kernel scale > 1 - edge detection, blurring

TEXTURE MAPPING

- *poly_point_nv_uv* - polygon vertex with a texture map index and vertex normals
- texture maps replicated in every pixel node's VRAM
- texture maps loaded with broadcast data (*picbroadv*)
- textures can be antialiased with supersampling
- current size limit - 256 squared images in 24-bit color
- high level generation of texture mapped primitives - bicubic patches, etc.

PICKING AND SELECTING

- Picking
  - set picking region size
  - attach / detach picking process
  - enter / exit picking mode
  - identifier stack - initialize, put id, push and pop id
- Selecting
  - attach / detach selecting process
  - select region is defined by a 3-D projection command
  - enter / exit selecting mode
  - identifier stack - initialize, put id, push and pop id

VIDEO CONTROLS

- update color map - enable / disable
- rgb color map table - put / get
- rgb color map table entry - put / get
- rgb alpha map table - put / get
- rgb alpha map table entry - put / get

PIClib DEMOS

- *Atoms, Lsd, Torus*: phong-shaded atom primitives
- *Flip0, Flip1, Scroll*: flipbooks and scrolling
- *Lighttool, RGB*: interaction between PIClib and *suntools*
- *Cursors, Paint, Pick, Event*: mice, cursors, picking and event handling
- *Doggy*: rgbaz 3D compositing
- *RasterText, VectorText*: raster and vector fonts
- *Super*: high level 3D object generation

PIClib DEMOS, cont'd

- *Objects*: rendering public-domain databases
- *Tmaps*: texture mapping onto different surfaces
- *Curves, Patches*: parametric cubic curves and bicubic patches
- *Moma, Logo*: simple backgrounds
- *Curves, Patches*: parametric cubic curves and bicubic patches
- *ImageIO*: simple frame buffer I/O stuff
- *FlyBy*: Bsplined camera position animation
- *Cue*: depth-cueing for lines
- *Example*: a simple PIClib program to render a still frame
- *Bounce*: shows what a creative computer animator can do in a day

RAYlib FEATURES

RAYlib is a library of C functions grouped into the following categories:

- Control Functions
  - initialize the machine
  - begin ray tracing
  - terminate a ray tracing session

- Graphics Primitives
  - generate 3-D polygons with normals and/or textures
  - superquadrics render spheres, cylinders, ellipsoids, toroids, and hyperboloids of one and two sheets

RAYlib FEATURES, cont'd.

- Bounding Volumes
  - initialize, terminate, and record 3-D extents of objects
  - proper use can increase ray tracing execution speed
- Transformations
  - viewing and projection functions
  - control size, position, and orientation of objects and scenes
  - control the transform matrix, modeling, scaling, and rotating of objects

RAYlib FEATURES, cont'd.

- Shading and Lighting Functions
  - control the position, orientation, and intensity of light sources
  - individual light switch control
  - handle ambient light intensity
  - define surface properties for objects
- Viewport Functions
  - create and manipulate viewports
- Antialiasing
  - eliminates jagged edges by using stochastic sampling
- Video Functions
  - load and retrieve color rgb maps
  - load and retrieve alpha overlay color maps
  - enable and disable video maps from shadow maps

DIFFERENCES BETWEEN PIClib AND RAYlib

Internal operation of RAYlib is fundamentally different from PIClib:

- PIClib renders each geometric primitive as it is received by the Pixel Machine
- RAYlib maintains a database of all geometry being rendered
- therefore, when using RAYlib, rendering takes place after all objects have been defined

SIMILARITIES BETWEEN PIClib AND RAYlib

Despite their internal differences, PIClib and RAYlib share:

- common syntax
- common functionality
  - however, not every RAYlib function has a corresponding PIClib function and vice-versa
- common structure definitions (typedefs)

Because of these similarities:

- programs can be easily ported from PIClib to RAYlib and vice-versa
- PIClib users will find it easy to use RAYlib

FUNCTIONS UNIQUE TO RAYlib

The following functions exist only in RAYlib:

- RAYtrace()
  - begins the ray tracing process
  - nothing is rendered until RAYtrace() is called
- RAYstatistics()
  - enables/disables printing of ray tracing statistics
- RAYopen_bounding_volume()
  - begins computation of a bounding volume
  - proper use of bounding volumes improves the performance of RAYlib

FUNCTIONS UNIQUE TO RAYlib, cont'd.

- RAYclose_bounding_volume()
  - ends computation of a bounding volume
- RAYambient_intensity()
  - sets the intensity of the white ambient light
- RAYbackground_color()
  - sets the color of a primary ray when it does not intersect any object in a 3-D scene
- RAYclear_viewport()
  - clears the current viewport to a specified rgbα color
  - this function is primarily used to clear the entire screen or to create drop shadows

FUNCTIONS UNIQUE TO RAYlib, cont'd

- RAYsamples()
  - defines the minimum and maximum number of samples to take within a pixel when antialiasing
  - defines the threshold to be used to determine the amount of antialiasing needed
- RAYput_texture()
  - allocates host memory for virtual textures or regions of resident texture memory
- RAYset_texture()
  - sets the current texture map to a specified texture id

COMMON STRUCTURE DEFINITIONS

Using common structure definitions:

- maintains compatibility between RAYlib and PIClib
- accommodates the differences in how the structure is used by each library
- particular elements of a structure may be meaningful to one library but ignored in the other

COMMON STRUCTURE DEFINITIONS, cont'd

The following structure definitions are used differently in RAYlib than in PIClib:

- RAYsurface_model()
  - contains the same elements as PICsurface_model(), but they are used slightly differently
  - in RAYlib, the *a_** and *s_** color components are ignored, and the *specularity*, *reflectivity*, and *refraction_index* elements are used; the reverse is true in PIClib
- RAYlight_source()
  - the structure element, *intensity*, applies to RAYlib point and area light sources, and is ignored in PIClib
  - the fields: *samples*, *vertices*, and *vertex* support area light sources

FUNCTIONS DIFFERENT FROM PIClib

The following RAYlib functions behave differently than their PIClib counterparts:

- RAYput_surface_model()
  - unlike PIClib, in RAYlib a call to RAYput_surface_model() allocates memory for each surface model
  - to reuse a surface model, RAYset_surface_model() should be called with the value returned by the call to RAYput_surface_model()
- RAYatom()
  - the radius of the atom primitive defined by RAYatom() is scaled by the average scale factor determined by the current transform
  - in PIClib no modeling transformations are applied to the radius of an atom, even though the projection transform is applied

FUNCTIONS DIFFERENT FROM PIClib, cont'd.

- RAYshade_mode()
  - controls shading effects such as shadows, reflections, and antialiasing
- RAYlight_ambient()
  - in PIClib sets the color of the ambient light for a 3-D scene
  - in RAYlib defines the color when a reflected or transmitted ray does not intersect any objects

DEVtools Training Outline

- Introduction and overview
- Pixel Machine Architecture
- DSP32 Tools (compiler, assembler, etc.)
- DEVtools Host Library
- DEVtools Pixel Machine Library
- Using DEVtools
- Runtime skeleton
- Sample programs
- Debugging tools
- Lab session

Pixel Machine Architecture - Overview

You should already know:

- Pixel Machine connects to Sun host via VME bus
- Uses DSP32 processors
- 1 or 2 pipe boards
  - Each board has 9 processors, each processor has:
    - an input FIFO
    - 36k bytes of static RAM
    - output to the FIFO of the next processor
  - with 2 pipe boards pipes can operate
    - serially
    - in parallel
- 16, 20, 32, 40 or 64 pixel boards
  - Each board has 4 processors, each processor has:
    - an input FIFO
    - 36k bytes of static RAM
    - 256k bytes of DRAM
    - 512k bytes of video memory
    - communication with 4 neighboring processors

Pipe Node Architecture

Memory areas:

- Startup code
- Static RAM
- Input FIFO
- Output FIFO
- Flags

Last node on a board:

- Feedback FIFO
- More flags

Pipe Node Architecture (continued)

Flags:

- For input FIFO:
  - empty flag
  - half-full flag
- For output FIFO:
  - half-full flag
  - full-flag

Pipe Node Architecture - Last Node

Flags on the last node of a pipe board:

- Output FIFO is the broadcast bus to the pixel node input FIFOs
- For feedback FIFO:
  - half-full flag
  - full flag
- Broadcast bus flags:
  - bus request
  - bus release
  - bus grant signal
- Pixel node signals
  - Pixel nodes - all vsync flags set
  - Pixel nodes - all psync flags set

Pipe Nodes - FIFO Rules

- Don't read from an empty FIFO
- Don't write to a full empty FIFO
- Always read or write all four bytes of each FIFO entry

Pipe Nodes - Using the Flags

Testing Flags:

- Connect the signal to the DSP32 sync pin
  - write to the memory location designated for each flag
  - wait at least one instruction cycle (two for the last pipe node on a board)
  - test using the the sys and syc flags

Obtaining the bus:

- Write to the bus request address
- Test the bus grant flag

Releasing the bus:

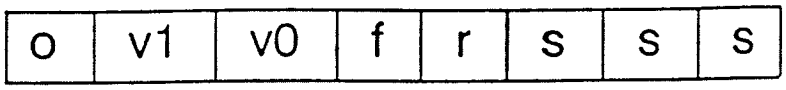
- Write to the bus release address

Pixel Node Architecture

Memory areas:

- Startup code
- Static RAM
- DRAM (via page registers)
- Video memory (via page registers)
- Input FIFO
- Flag register
- Page registers

Pixel Node - Flag Register



Flag register contains:

- sss
  - Sync signal selection flags
- f r
  - the nodes psync (f for flag) and vsync (r for rdy) flags
- v0 v1
  - video buffer selection flags
- o
  - overlay flag

Pixel Node - Flag Register (continued)

Sync signal values:

- 010
  - draw empty
- 011
  - draw half-full
- 100
  - vertical blanking
- 101
  - horizontal blanking
- 110
  - all processors have vsync set
- 111
  - all processors have psync set

Pixel Node - Mode Register

- Mode register must be set by host
- Contains:
  - overlay mode
  - video shift flag
  - gate enable flag
  - serial I/O direction

Overlay mode:

- Overlay off
  - rgb values always used
- Overlay on
  - if overlay = 0 use rgb
    if overlay = 255 use ~rgb
    otherwise use overlay
- Overlay force
  - always use overlay
- Overlay mask
  - if high order bit of overlay is set (overlay & 0x80) use the overlay value
    otherwise use rgb

Pixel Nodes - FIFO Rules

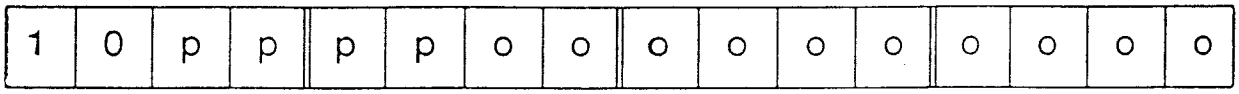
- Don't read from an empty FIFO
- Always read all four bytes of each FIFO entry

Pixel Nodes - Testing the Flags

- Connect the signal to the DSP32 sync pin
  - update the sync signal field of the flags register
  - wait at least two instruction cycles
  - test using the the sys and syc flags

Pixel Node - Page Registers

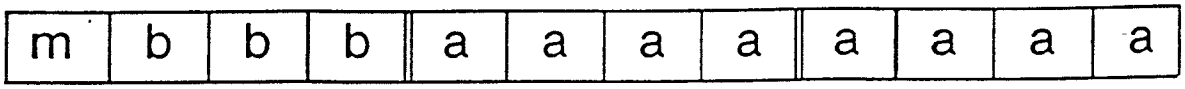
- Needed because DSP32 has 16 bit address space
- Used for:
  - DRAM
  - video memory



Format of a Paged Memory Address

- 10 in the first two bits = a page register memory reference
- pppp is the page register number
- oooooooooo is the offset from the location described by the page register

Pixel Node - Page Register Structure



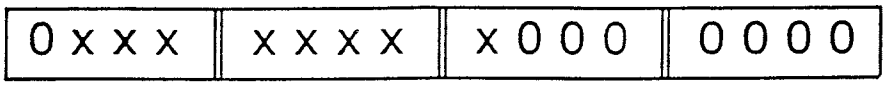
Format of a Page Register

- m is the mode bit
  - 0: fixed row
  - 1: fixed column
- bbb is the bank selection code
  - 001: DRAM
  - 100: RG0
  - 101: BO0
  - 110: RG1
  - 111: BO0
- aaaaaaaa is the extended address

Pixel Nodes - Using DRAM

- Must use page register to access
- Once a page register has been established:
  - other locations in the same row (or column in fixed column mode) can be accessed using a normal pointer
  - pointer references can not wrap around the end of a row (or column)
- Can be accessed as floats, shorts, or bytes

Pixel Nodes - Using Video Memory



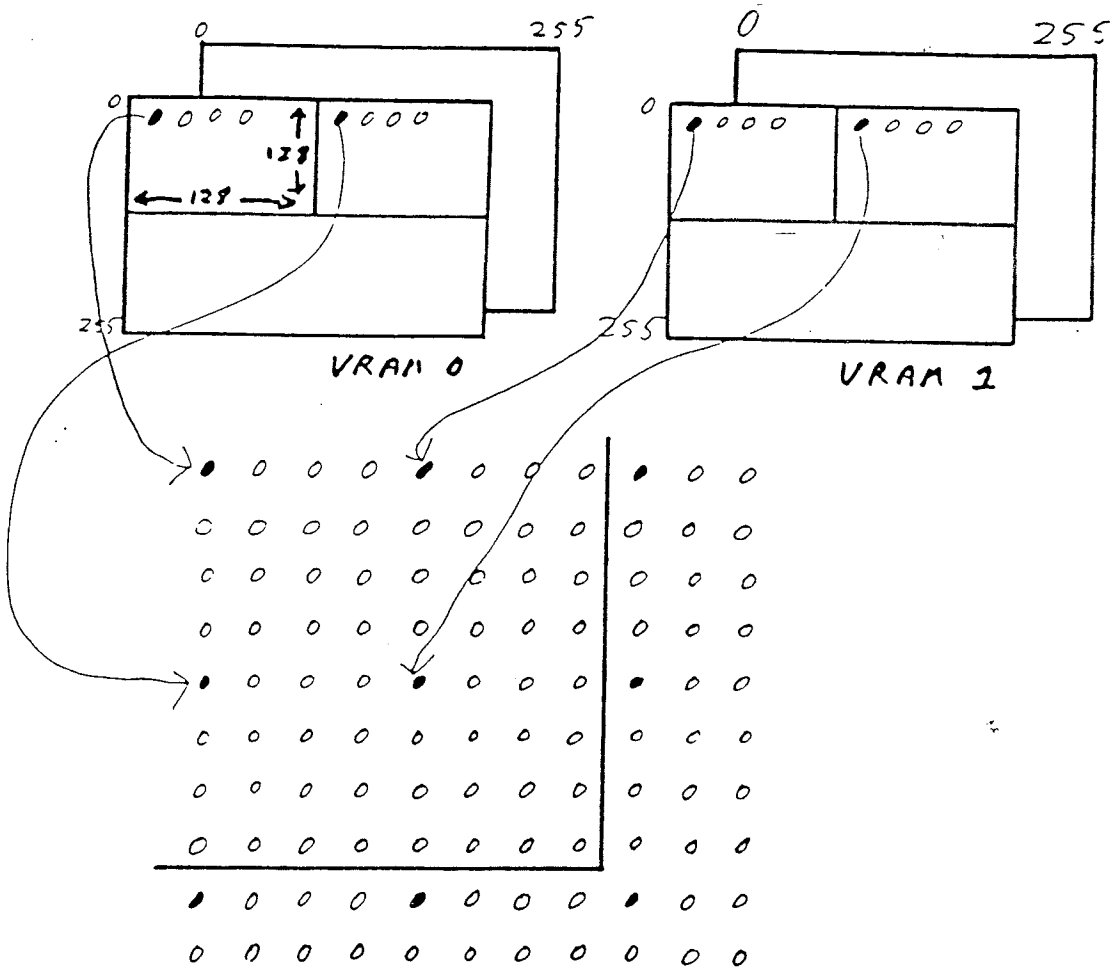
How Pixels are Stored in Memory

- Each pixel occupies 2 bytes of address space
  - simplifies saturation processing
  - 512k bytes of video memory occupies 1 megabyte of address space

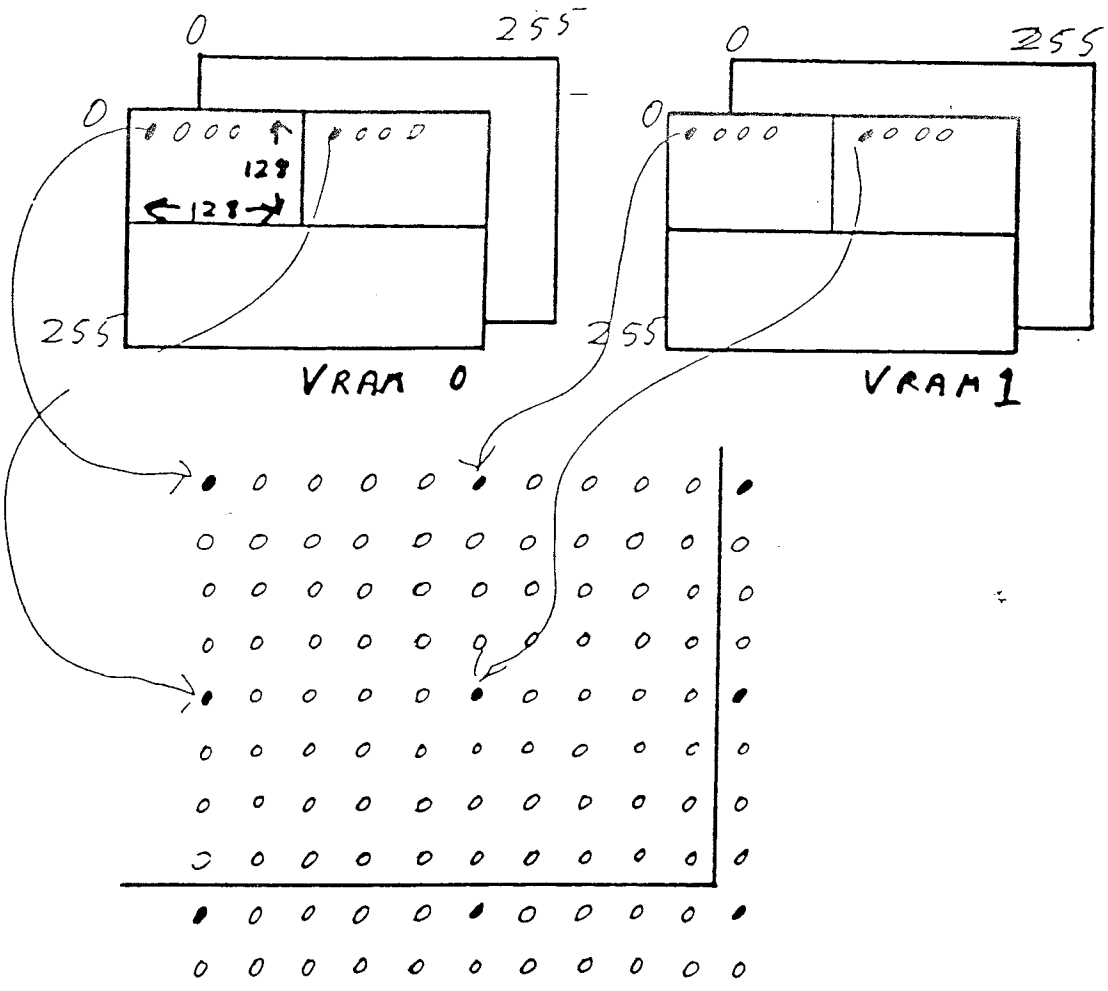
Pixel Nodes
Video Memory Organization

- Two banks
  - VRAM0
  - VRAM1
- Each bank consists of two sections:
  - red and green components
  - blue and overlay components
- This is because:
  - each section is addressed as 256 x 256 x 32 bits
    - 16 bits of red, 16 bits of green, even though only 8 bits are used of each word
  - page register offsets are 0 to 1023 bytes
    - 256 x 32 bits (4 bytes) is 1024 bytes

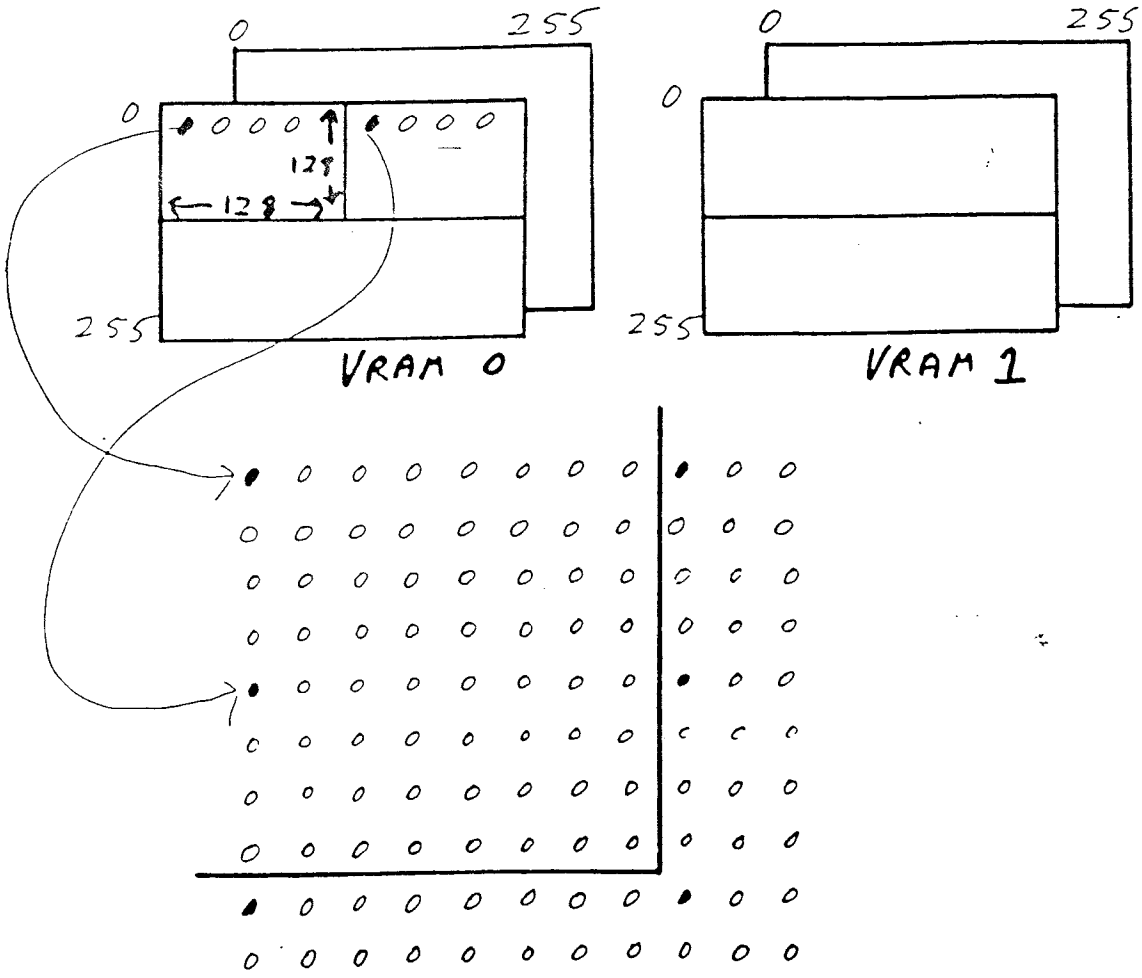
Subscreens for the 916
Processor 0, Buffer 0



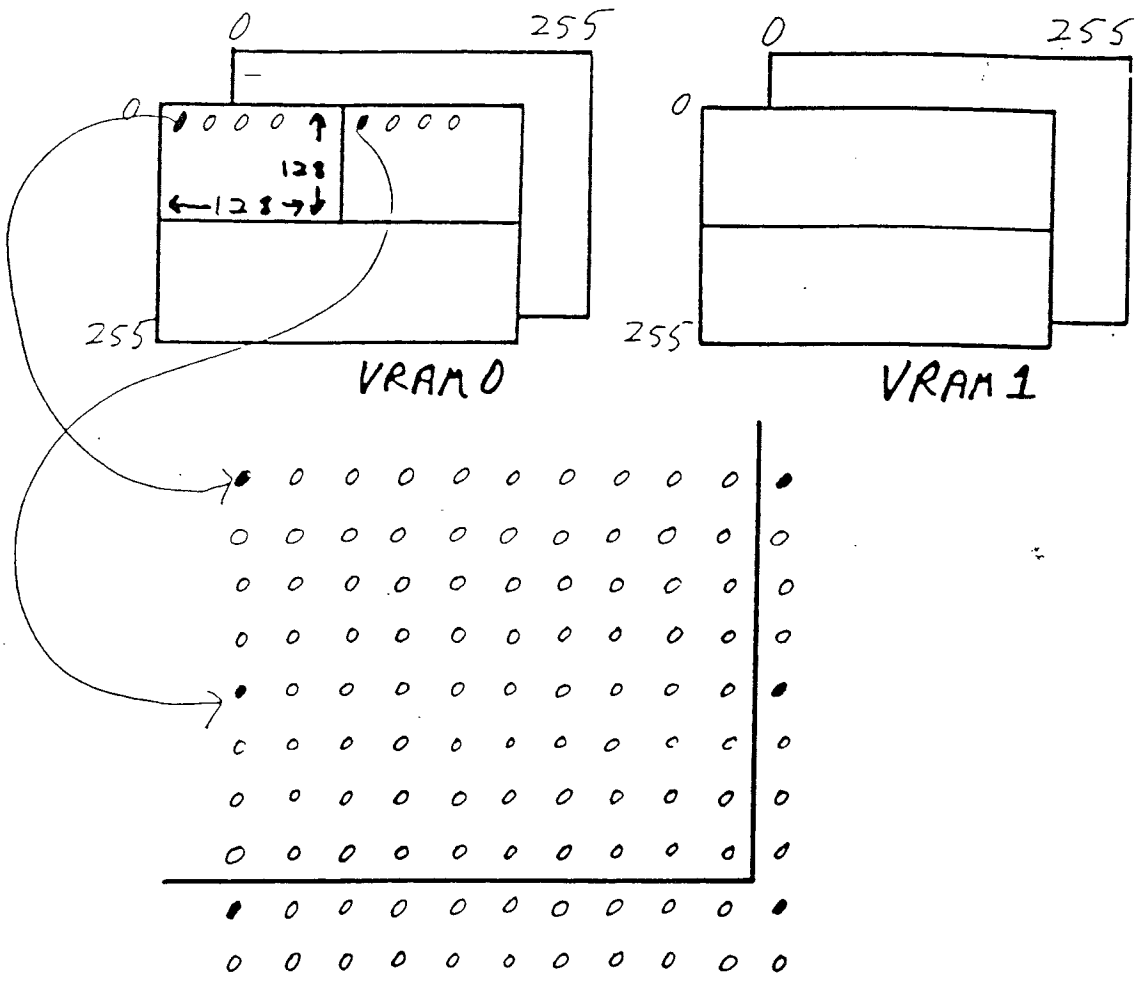
Subscreens for the 920
Processor 0, Buffer 0



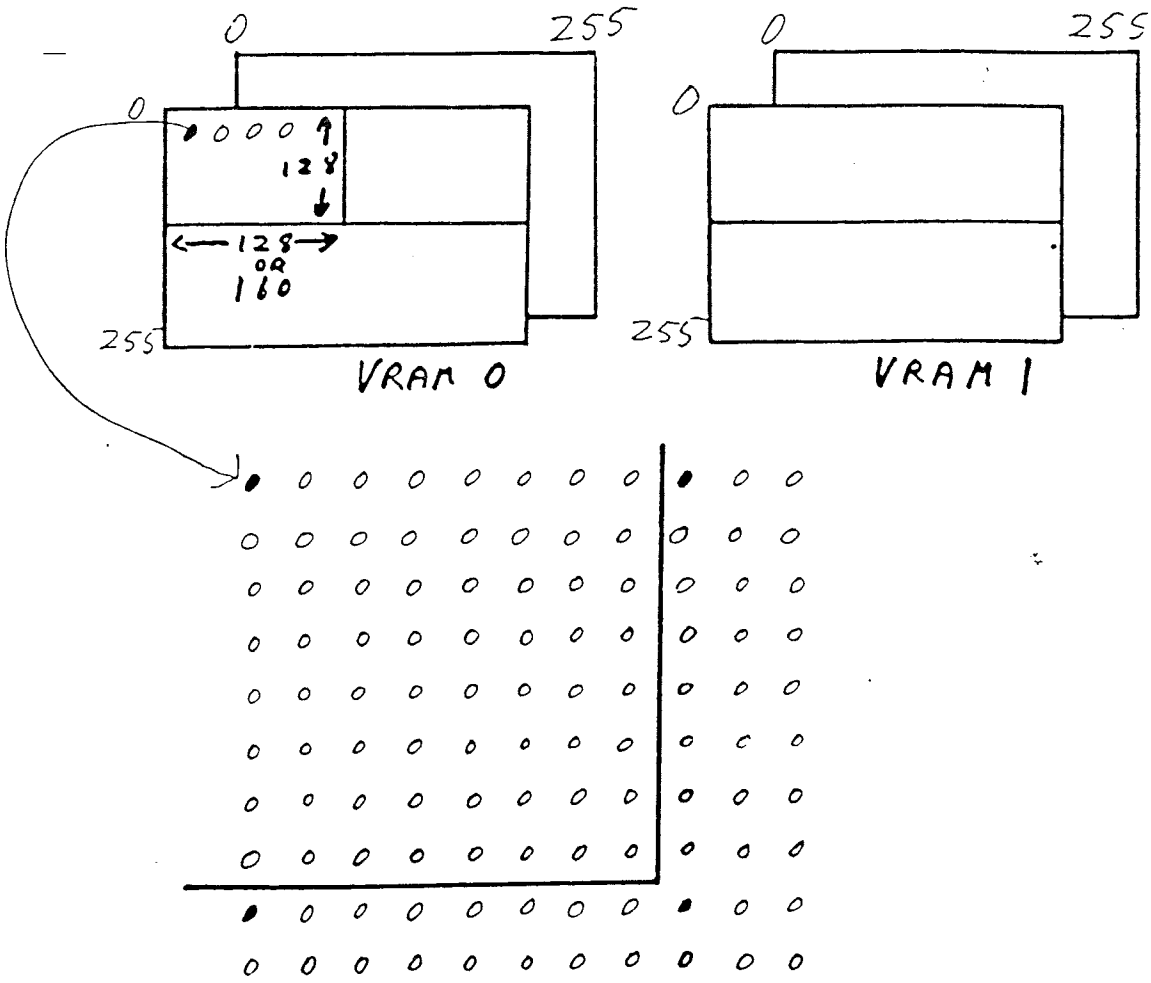
Subscreens for the 932
Processor 0, Buffer 0



Subscreens for the 940
Processor 0, Buffer 0



Subscreens for the 964
Processor 0, Buffer 0



DSP32 Architecture - Overview

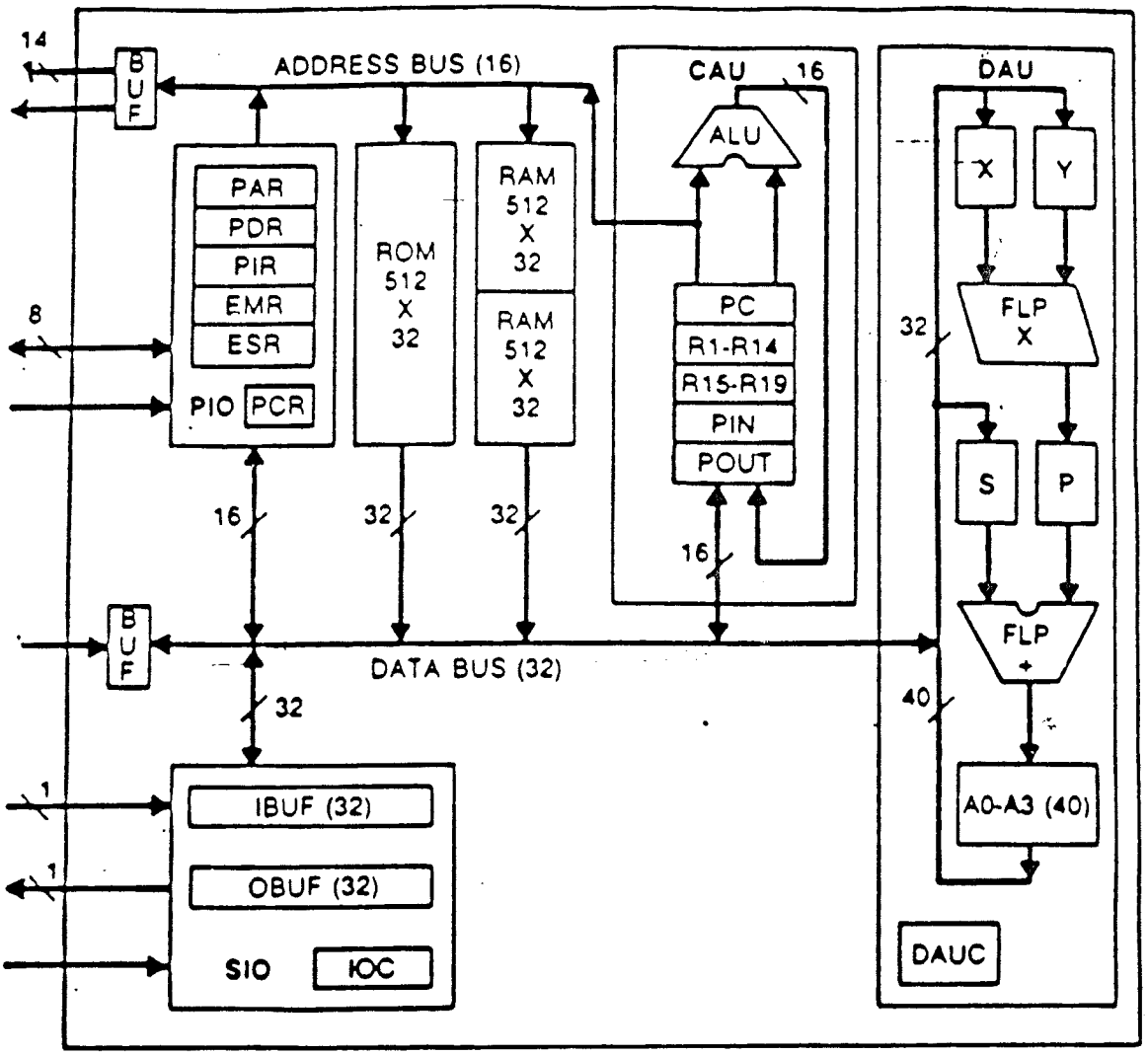
- 5 MIPS
- Up to 10 MFLOPS
- DAU (data arithmetic unit)
  - 4 40 bit accumulators
  - highly pipelined - can execute 5 million floating point multiple/add instructions per second
- CAU (control arithmetic unit)
  - 21 16 bit integer registers
- Parallel I/O
  - DMA
  - program controlled
  - used to communicate with host

DSP32 Architecture - Overview (continued)

- Serial I/O
  - DMA
  - program controlled
  - used for communication between DSP processors

- Data format - least significant byte first
  - reverse of Sun
  - same as VAX

DSP32 Architecture



DSP32 Architecture - DAU

- Highly pipelined
  - several instructions executing at once
  - assembler programmers must know about pipelining and latency
  - not as hard as it sounds
- General form of instruction is:

$$A = [-]B \{+,-\} C * D$$

- Operands can be:
  - floating point register
  - indirect via a general register (r1-r14)
  - indirect with a post-increment or decrement
  - indirect with a post-increment from a register (r15-r19)

DSP32 Architecture - Registers

- General registers
  - r1-r14 - general purpose
  - r15-r19 - general purpose - can be used as postincrement registers
  - a0 - a3 - 40 bit floating point accumulators
  - PIN, POUT - general purpose - serial I/O input/output pointers
  - PCR - parallel I/O control register
  - PDR - parallel data register
  - PIR - parallel interupt register
  - PAR - parallel address register
  - EMR - error mask register
  - ESR - error source register
  - IBUF, OBUF - serial I/O input and output registers
  - IOC - I/O control register

DSP32 Architecture - PCR Contents

- DMA mode
- autoincrement mode
- PIF flag
  - set when the DSP writes to the PIR
  - cleared when the host reads from the PIR
- PDF flag
  - set when the host writes to the PDR
  - cleared when the DSP reads from the PDR

DSP32 Architecture - Parallel I/O

- Options:
  - DMA or program control
  - For DMA - autoincrement after read/write
- DMA to/from host:
  - host sets address in PAR
  - host reads from or writes to PDR
- Program control from host:
  - host writes to PDR
  - waits for PDF flag in PCR to be reset
  - DSP reads from PDR
- Program control from DSP:
  - DSP writes to PIR
  - host checks PIF flag in PCR
  - host reads PIR

DSP32 Tools

- located in /usr/hyper/devtools/dsp32/{bin, lib, include}

- set shell environment variable:
  DSP32SL=/usr/hyper/devtools/dsp32
- set path:
  PATH=$PATH:/usr/hyper/devtools/dsp32/bin
- see documentation
  - C Language Compiler User Manual
  - C Language Compiler Library Reference Guide
  - Software Support Library User Manual

DSP32 Commands

- d3ar
  - archive (library) management utility
- d3as
  - assembler
- d3cc
  - C compiler
- d3ld
  - linker
- d3nm
  - object and executable file map listing utility
- d3sim
  - simulator

DSP32 Libraries

- libc
  - subset of UNIX system libc
  - printf - for simulator only libnode version used for Pixel Machine
  - memcpy, strlen, ctype, etc.
- libap
  - matrix multiplication
  - filter functions
  - fast trig functions
- libm
  - standard trig functions, etc.

↓ cut ↓

DEVtools Training Outline

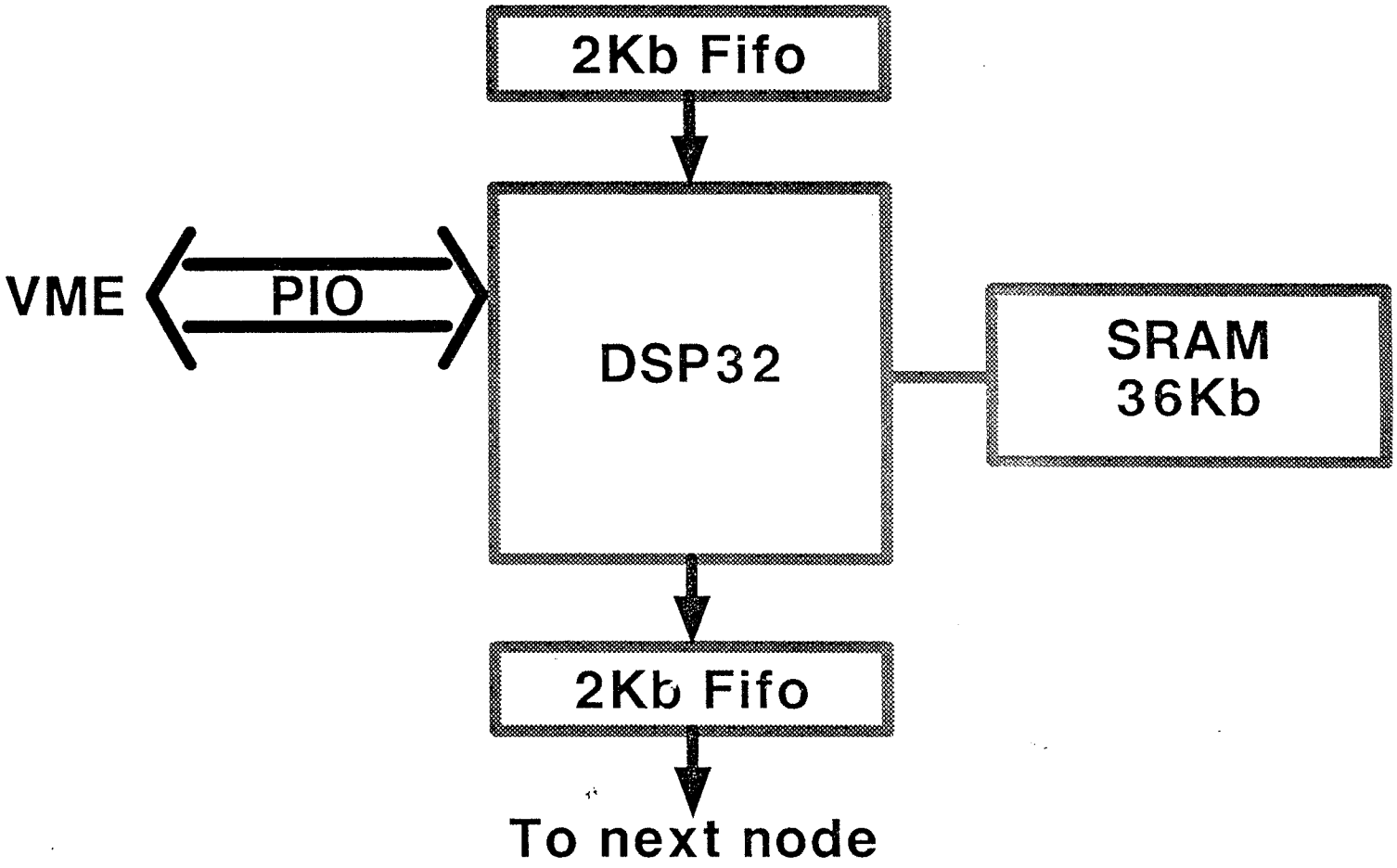
- Introduction and overview
- Pixel Machine Architecture
- DSP32 Tools (compiler, assembler, etc.)
- DEVtools Host Library
- DEVtools Pixel Machine Library
- Using DEVtools
- Runtime skeleton
- Sample programs
- Debugging tools
- Lab session

Pixel Machine Architecture - Overview

You should already know:

- Pixel Machine connects to Sun host via VME bus
- Uses DSP32 processors
- 1 or 2 pipe boards
  - Each board has 9 processors, each processor has:
    - an input FIFO
    - 36k bytes of static RAM
    - output to the FIFO of the next processor
  - with 2 pipe boards pipes can operate
    - serially
    - in parallel
- 16, 20, 32, 40 or 64 pixel boards
  - Each board has 4 processors, each processor has:
    - an input FIFO
    - 36k bytes of static RAM
    - 256k bytes of DRAM
    - 512k bytes of video memory
    - communication with 4 neighboring processors

PIPE NODE



HGA93212.015

Pipe Node Architecture

Memory areas:

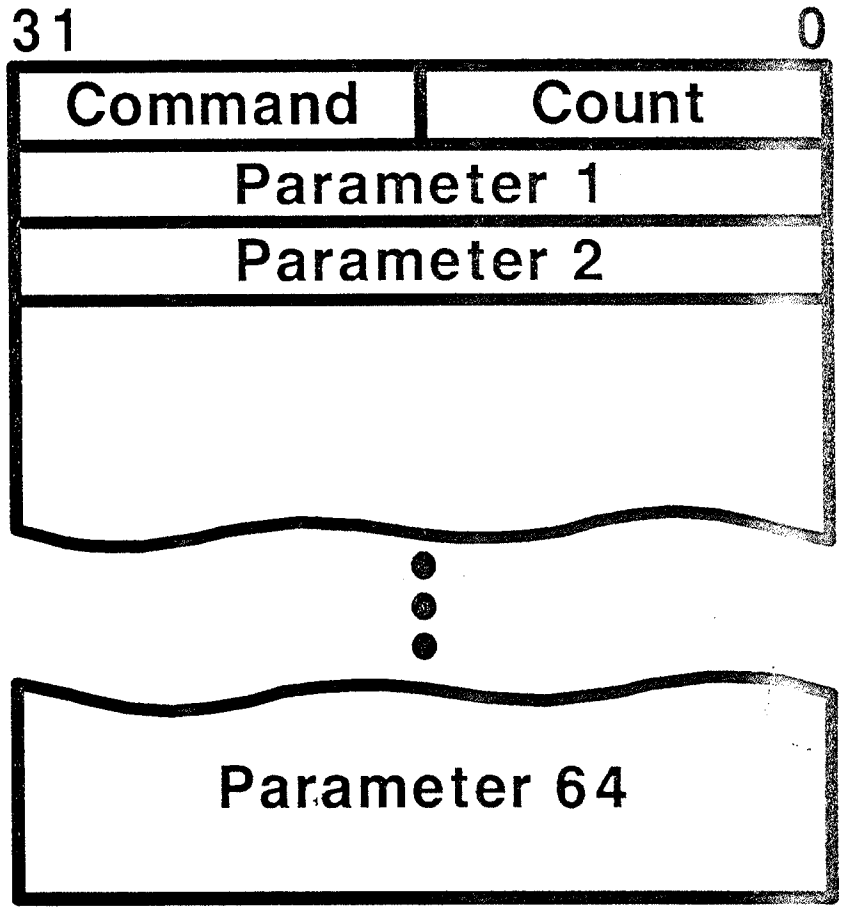
- Startup code
- Static RAM
- Input FIFO
- Output FIFO
- Flags

Last node on a board:

- Feedback FIFO
- More flags

FIFO

Command passing



HGA93212.017

Pipe Node Architecture (continued)

Flags:

- For input FIFO:
  - — empty flag
  - — half-full flag
- For output FIFO:
  - — half-full flag
  - — full-flag

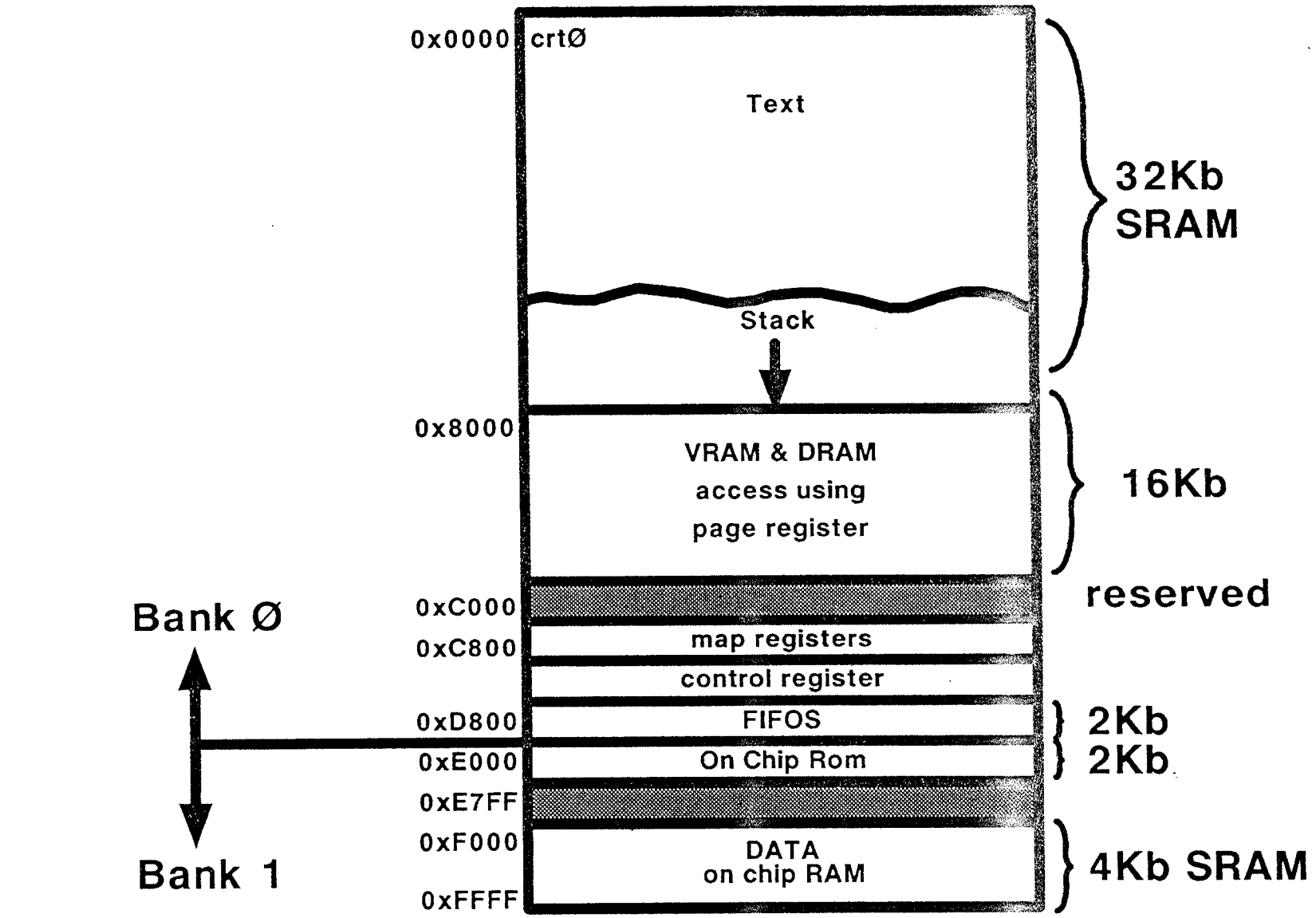
Pipe Node Architecture - Last Node

Flags on the last node of a pipe board:

- Output FIFO is the broadcast bus to the pixel node input FIFOs
- For feedback FIFO:
  - half-full flag
  - full flag
- Broadcast bus flags:
  - bus request
  - bus release
  - bus grant signal
- Pixel node signals
  - Pixel nodes - all vsync flags set
  - Pixel nodes - all psync flags set

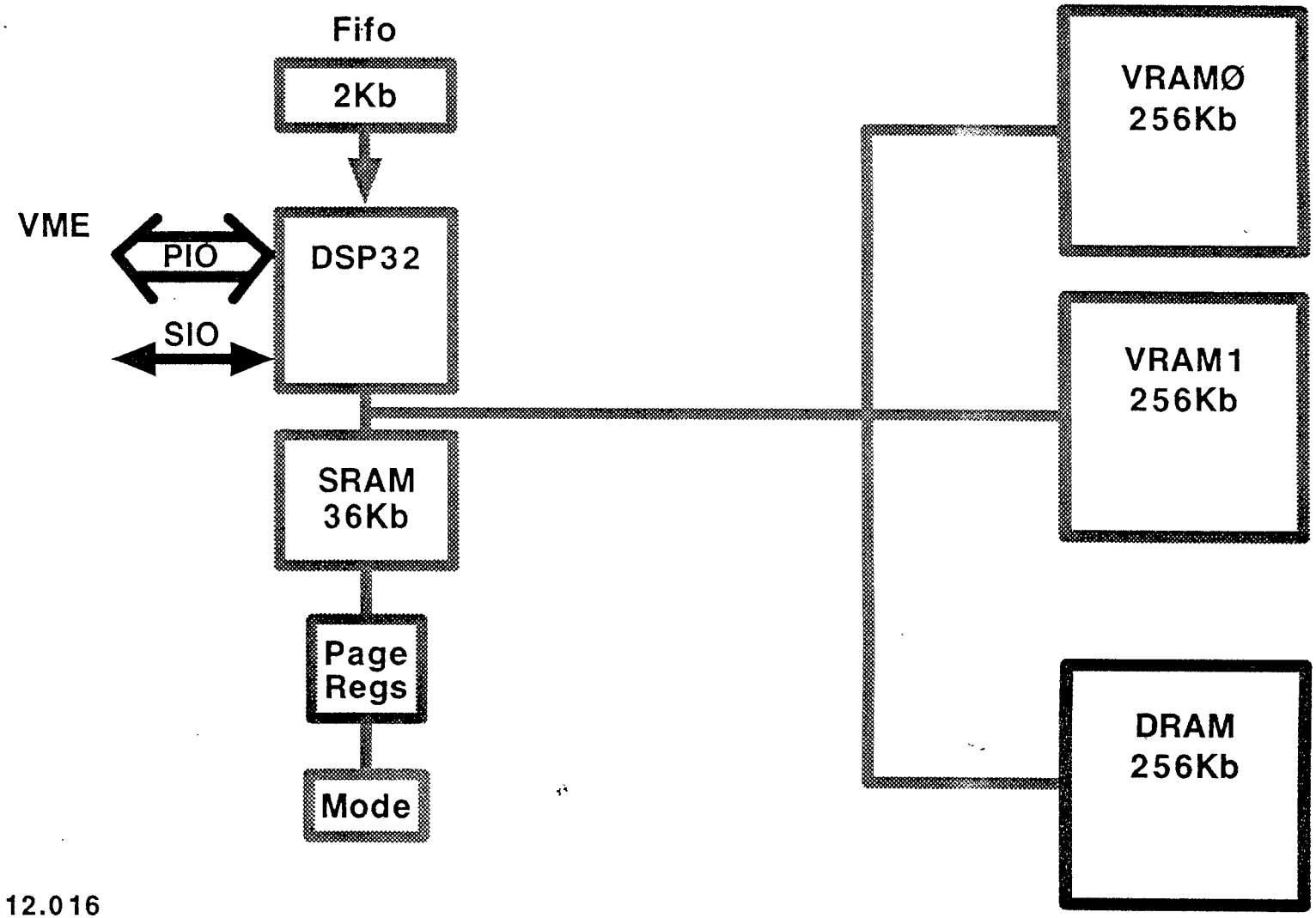
Pipe Nodes - FIFO Rules

- Don't read from an empty FIFO
- Don't write to a full ~~empty~~ FIFO
- Always read or write all four bytes of each FIFO entry



HGA93212.013

PIXEL NODE



HGA93212.016

PIXEL NODES

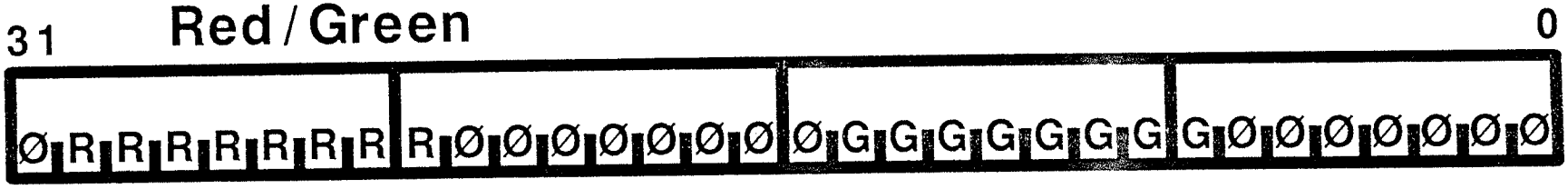
Video memory organization

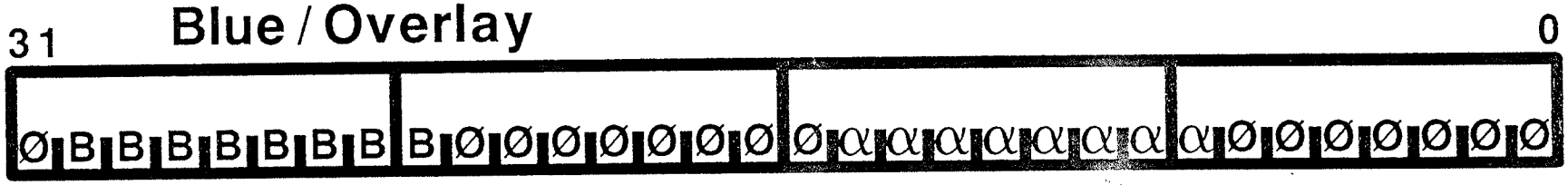
- Two banks
  - VRAM0
  - VRAM1
- Each bank consists of two sections:
  - red and green components
  - blue and overlay components
- This is because:
  - each section is addressed as 256 x 256 x 32 bits
    - 16 bits of red, 16 bits of green, even though only 8 bits are used of each word
  - address offsets are 0 to 1023 bytes
    - 256 x 32 bits (4 bytes) is 1024 bytes

HGA93212.001

HOW PIXELS ARE STORED

- Each pixel occupies 2 bytes of address space
  - simplifies saturation processing
  - 512Kb of video memory occupies 1 megabyte of address space



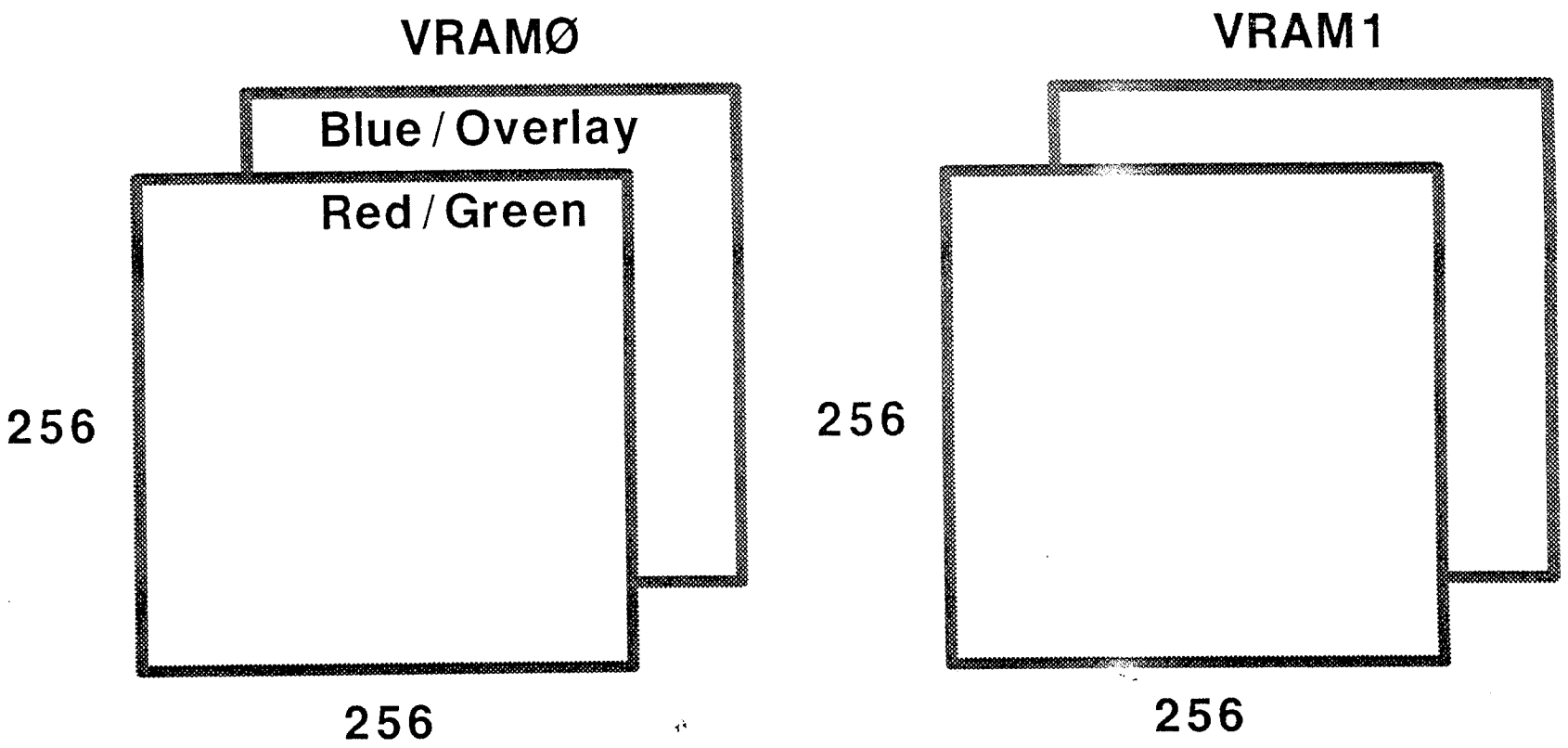


- One instruction int ↔ float

HGA93212.002

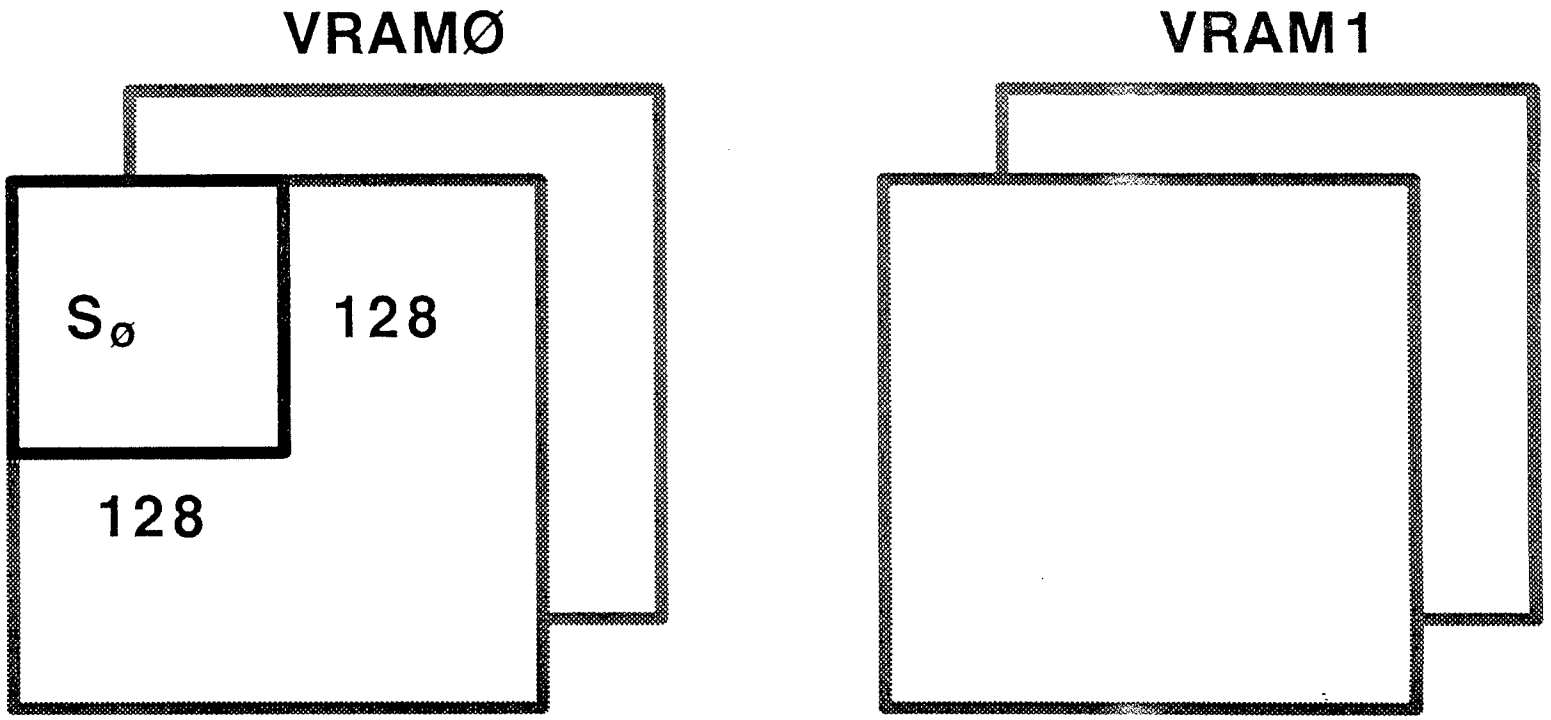
PIXEL NODES

Video memory organization



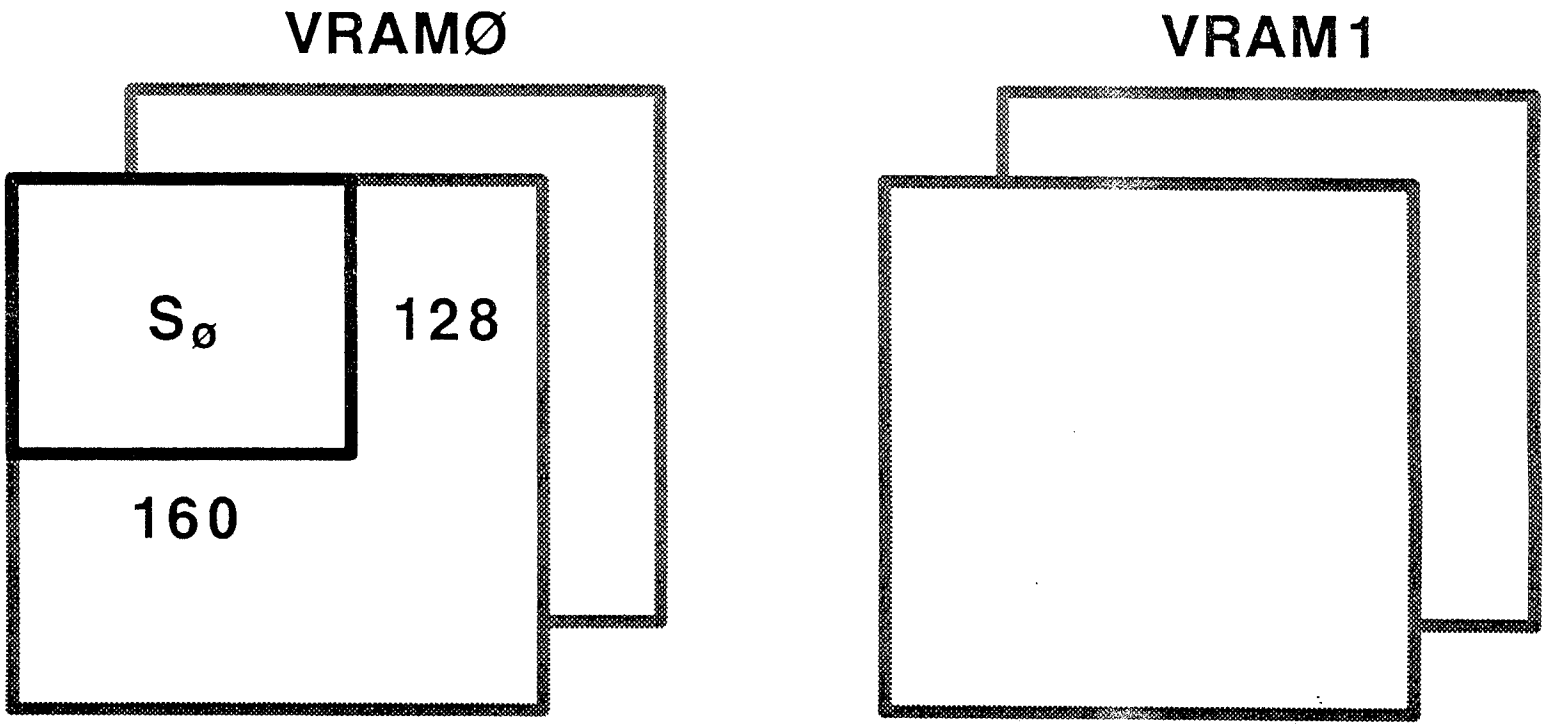
HGA93212.021

FRAME BUFFER ORGANIZATION
Model 964



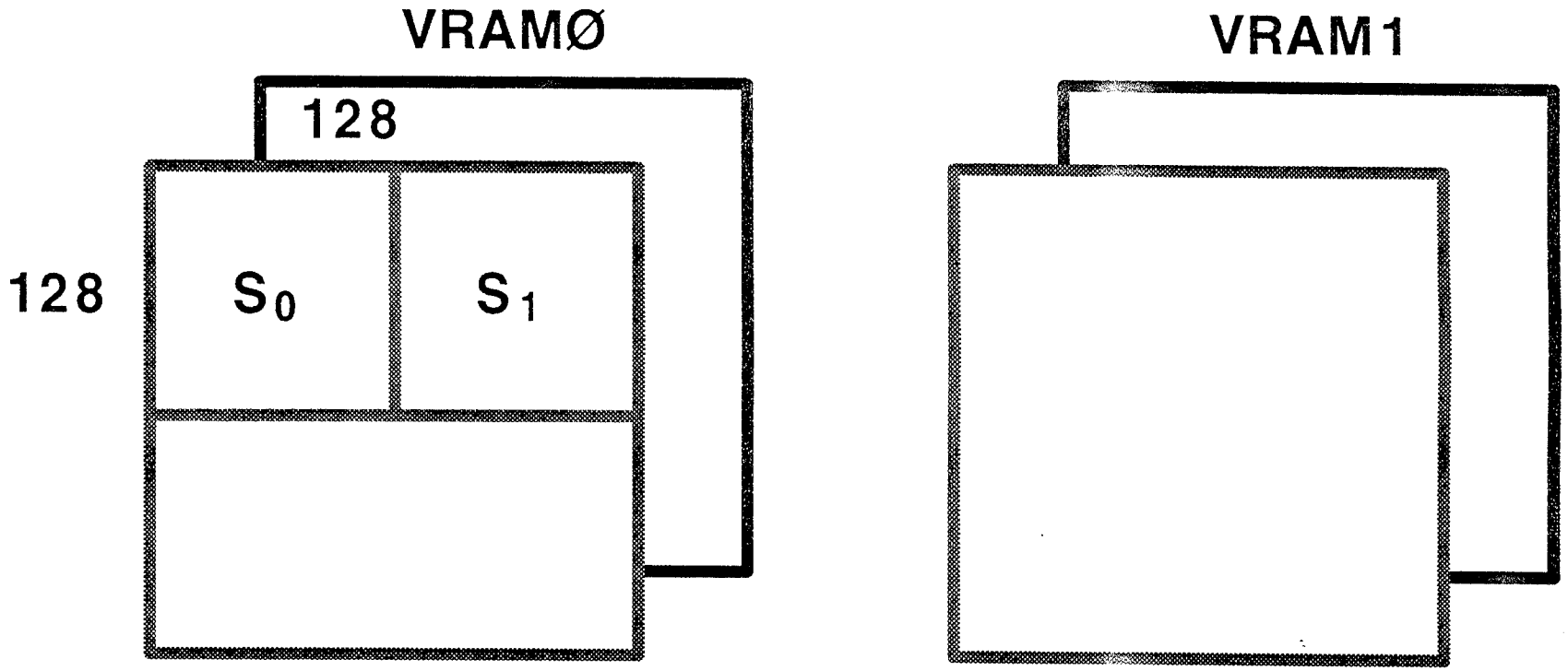
HGA93212.022

FRAME BUFFER ORGANIZATION
Model 964X



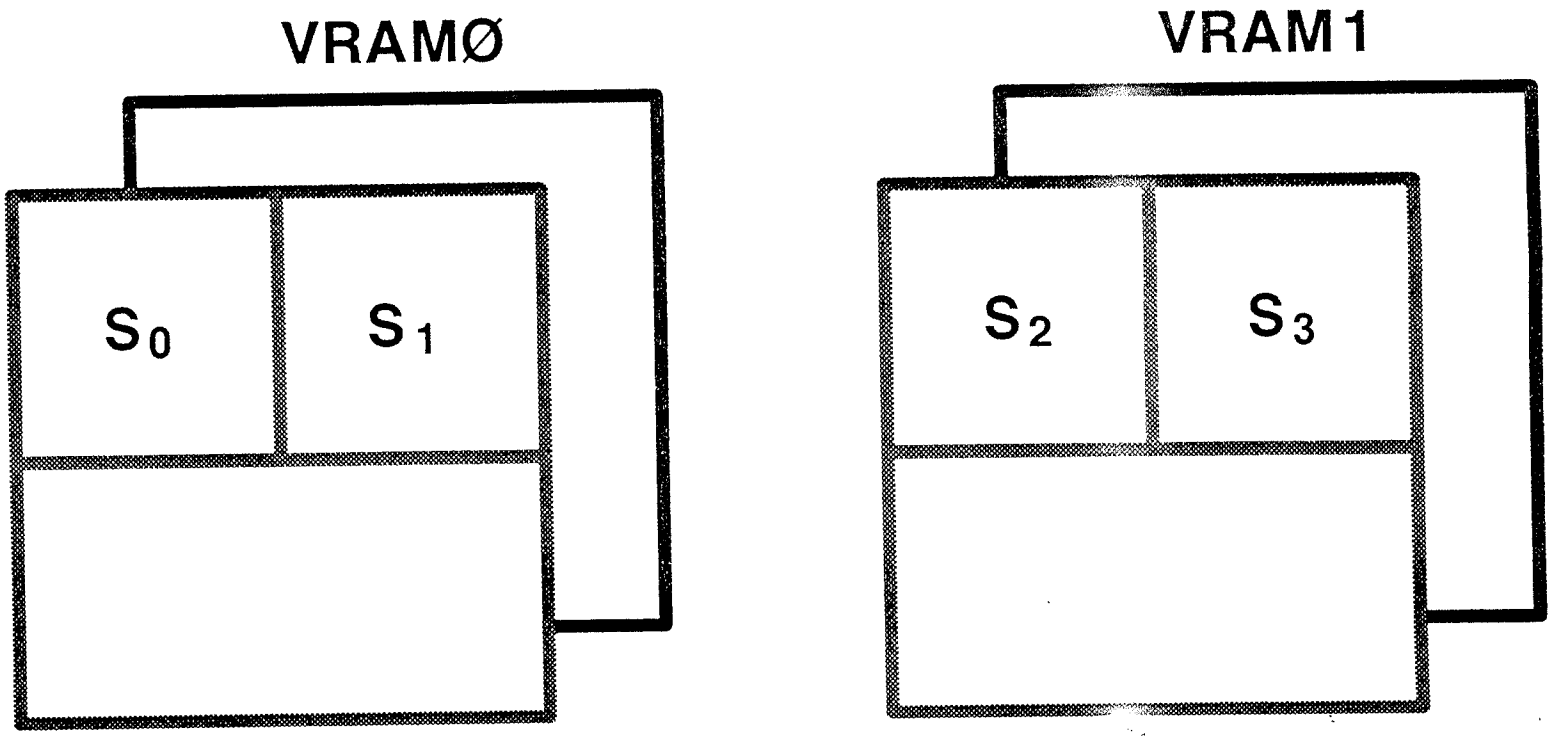
HGA93212.023

FRAME BUFFER ORGANIZATION
Model 940/32



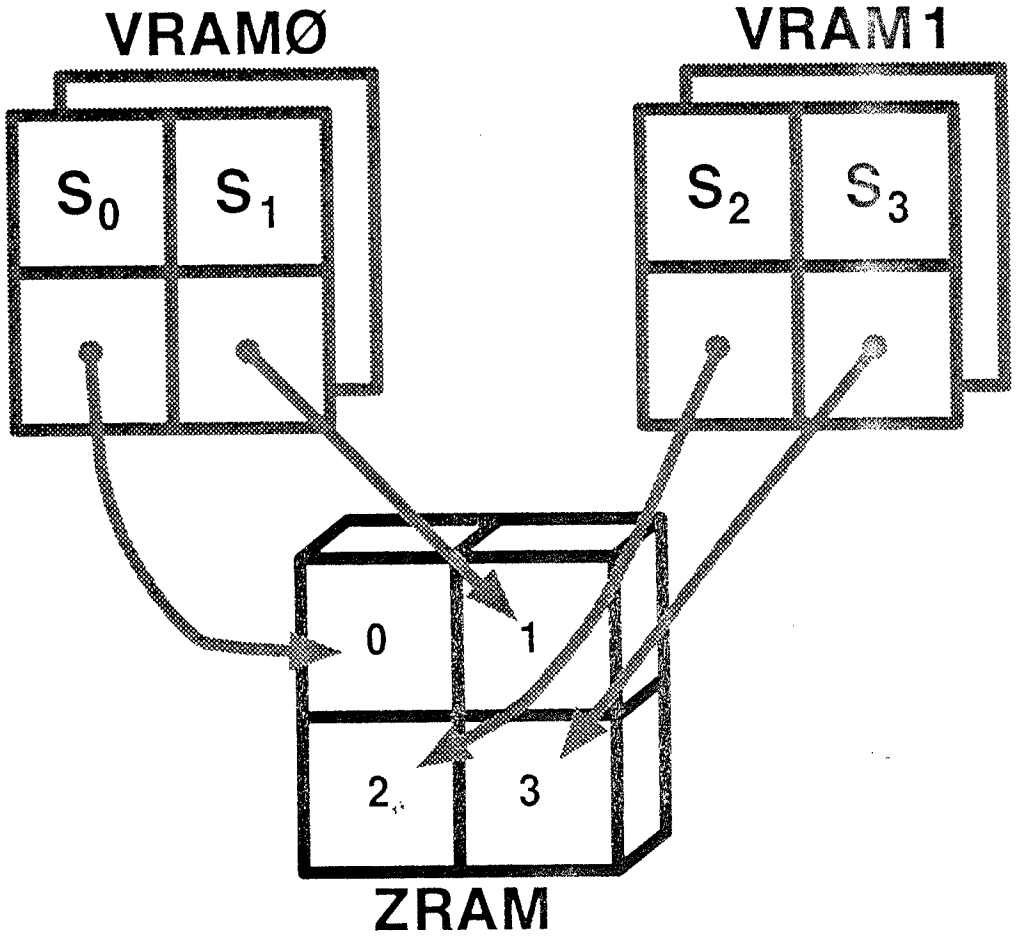
HGA93212.025

FRAME BUFFER ORGANIZATION
Model 920/16



HGA93212.024

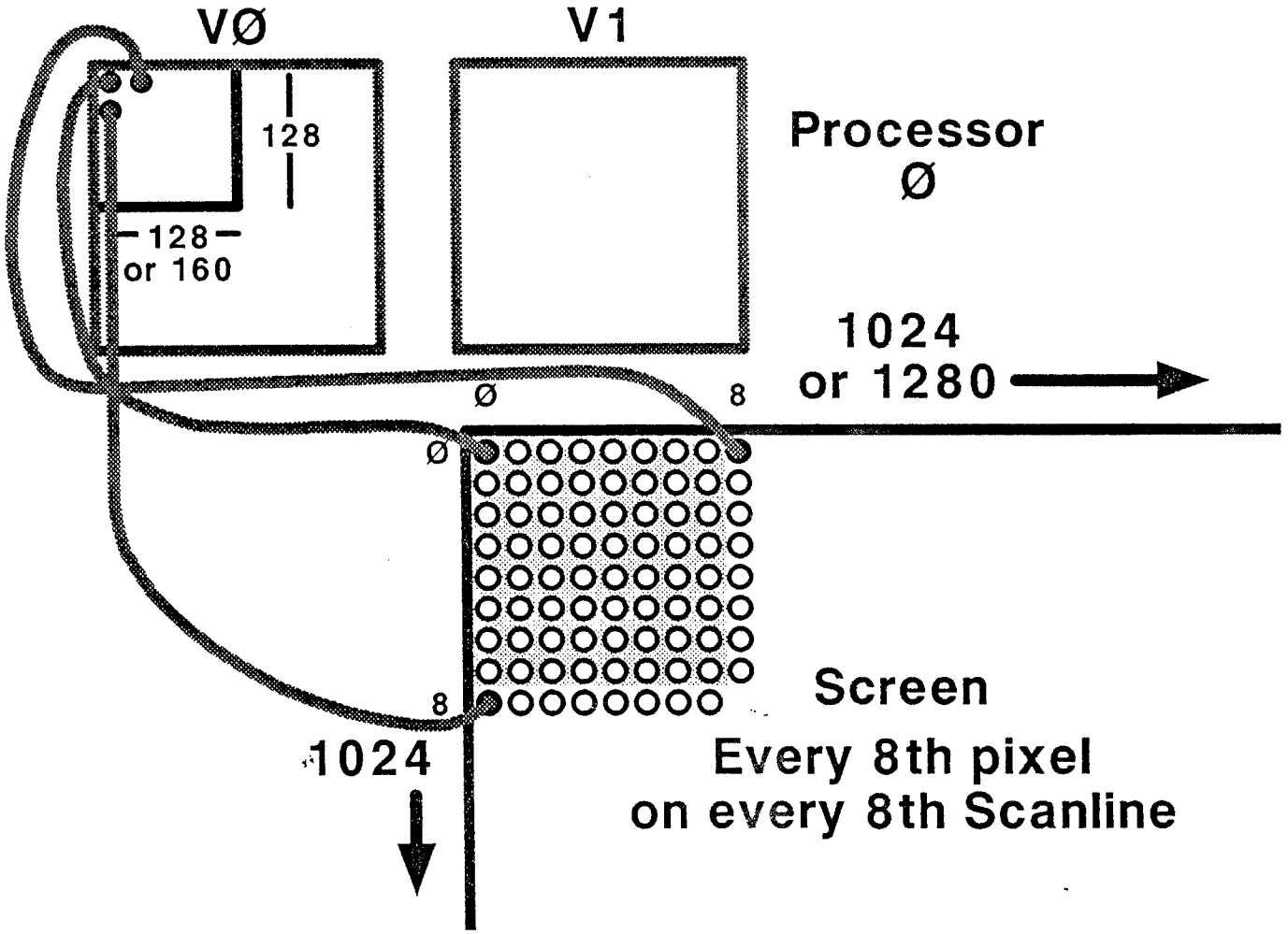
Z BUFFER MAPPING
Model 916/20



HGA93212.026

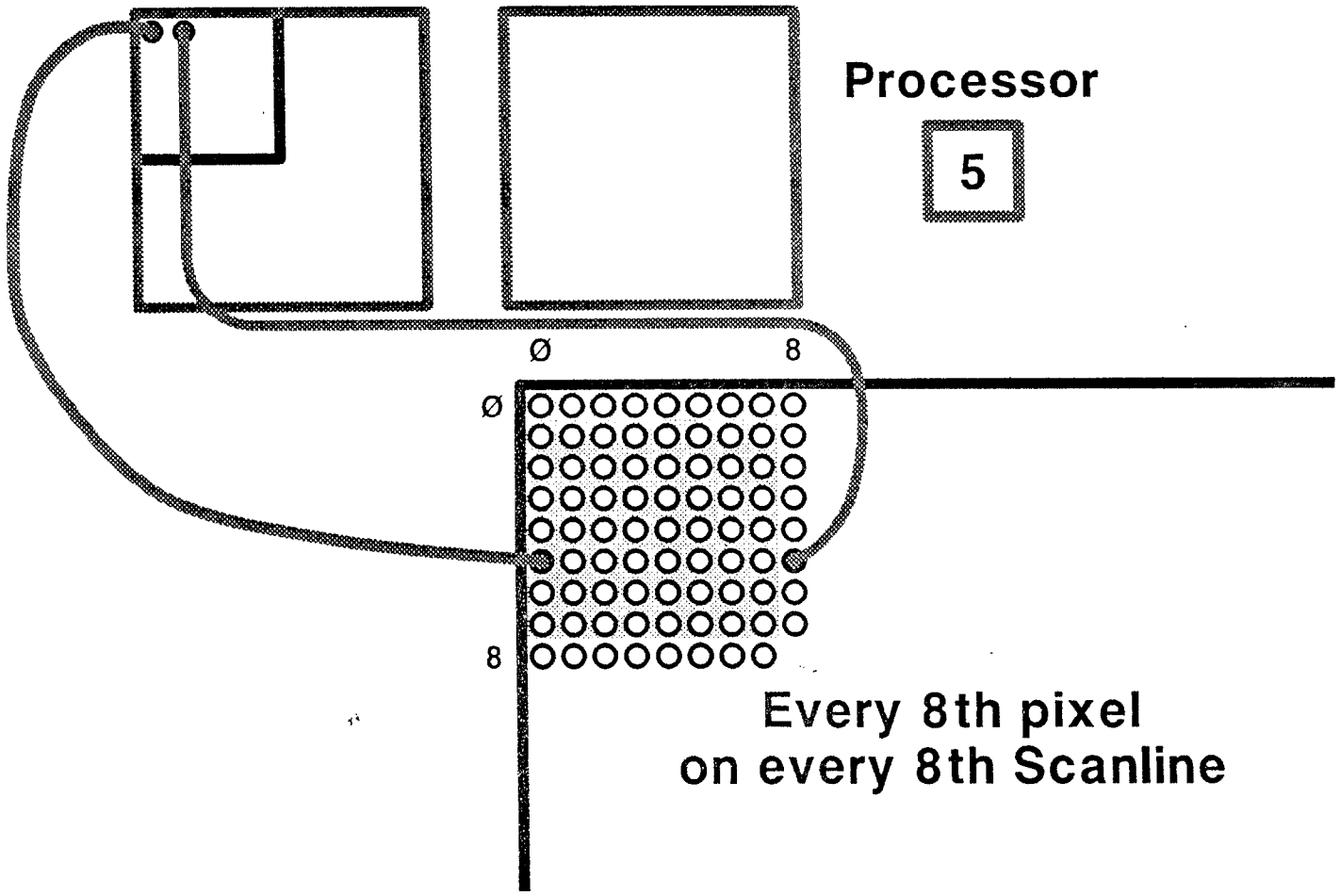
PROCESSOR TO SCREEN MAPPING

Model 964



HGA93212.027

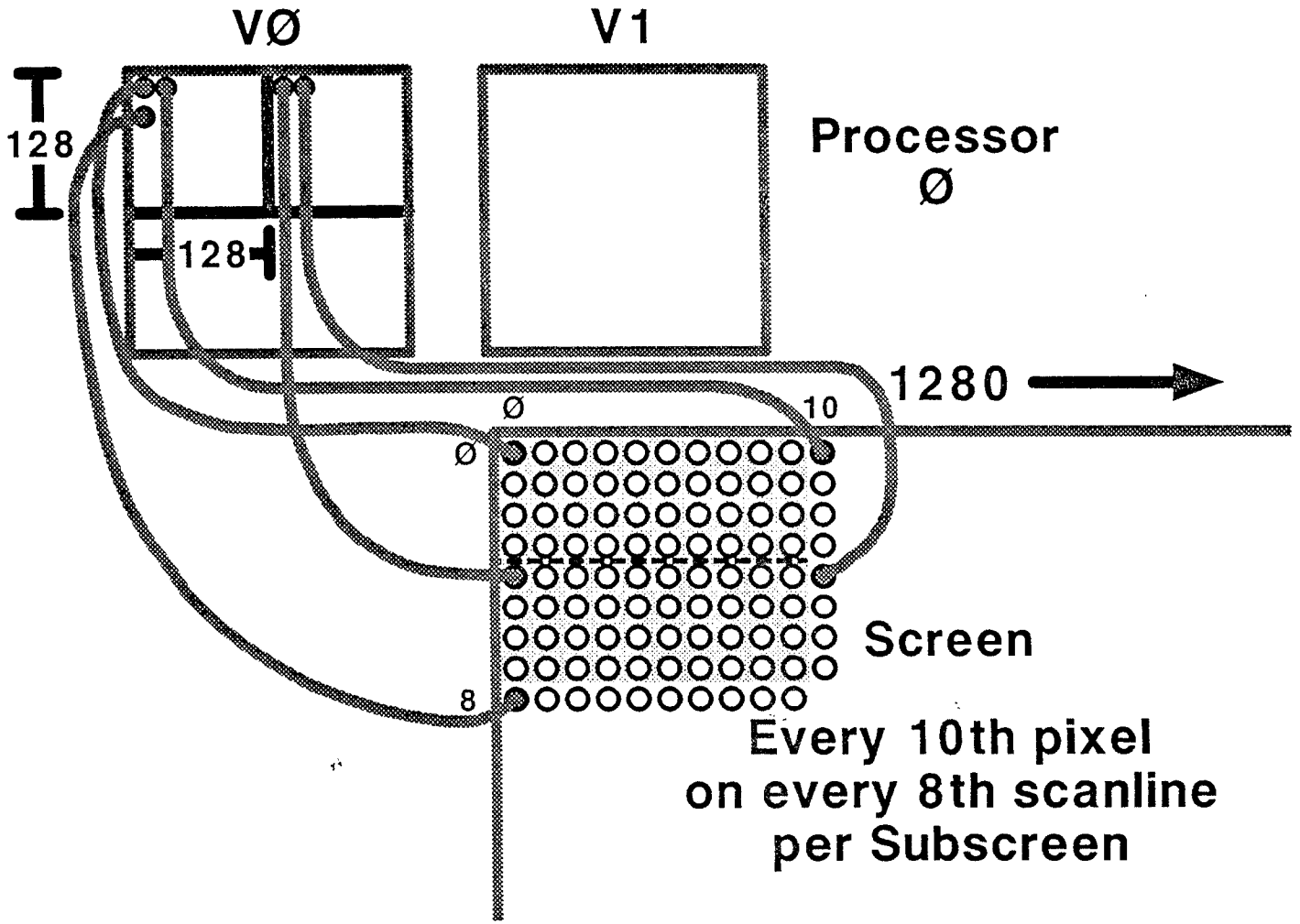
PROCESSING TO SCREEN MAPPING
Model 964



HGA93212.009

PROCESSING TO SCREEN MAPPING

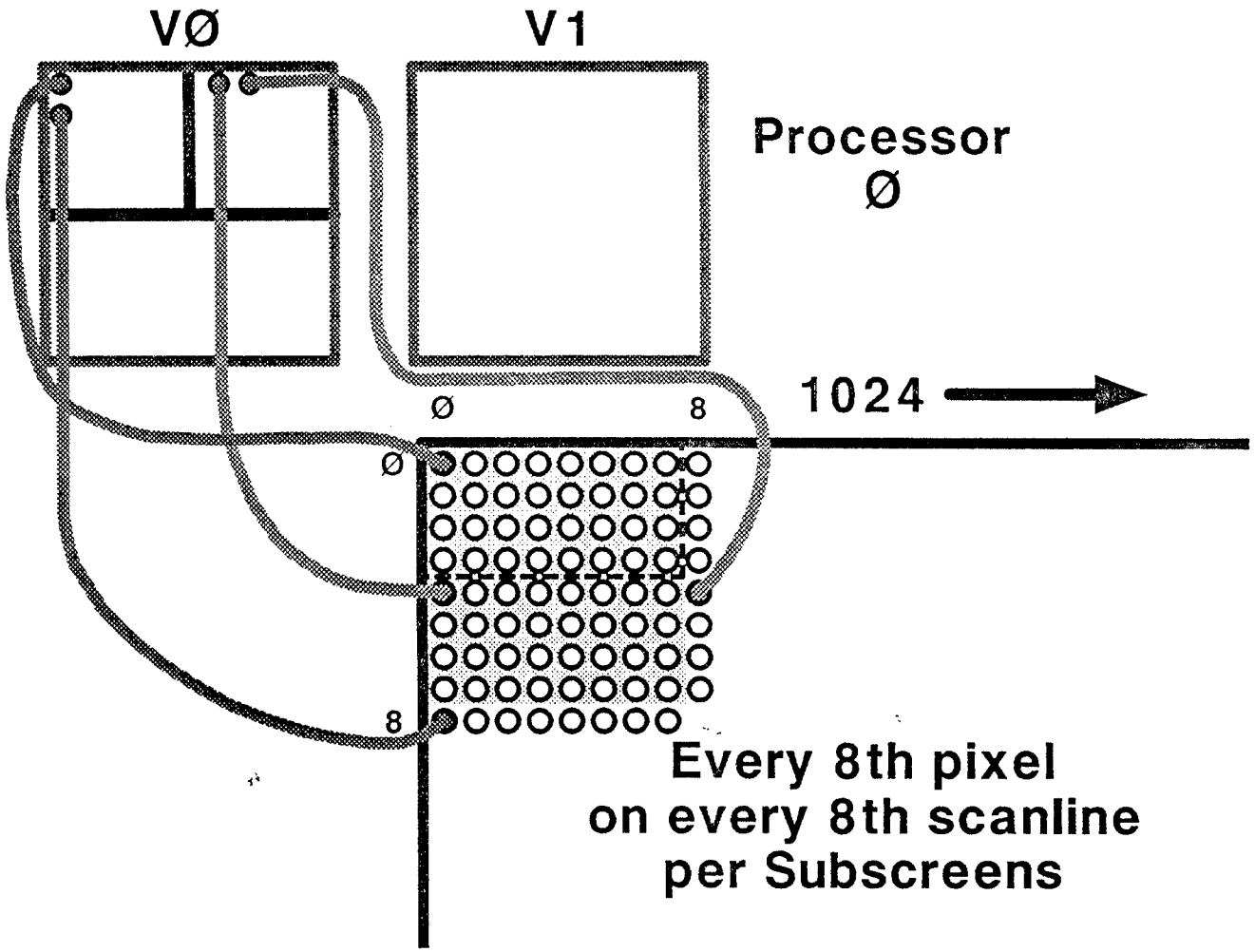
Model 940



HGA93212.010

PROCESSING TO SCREEN MAPPING

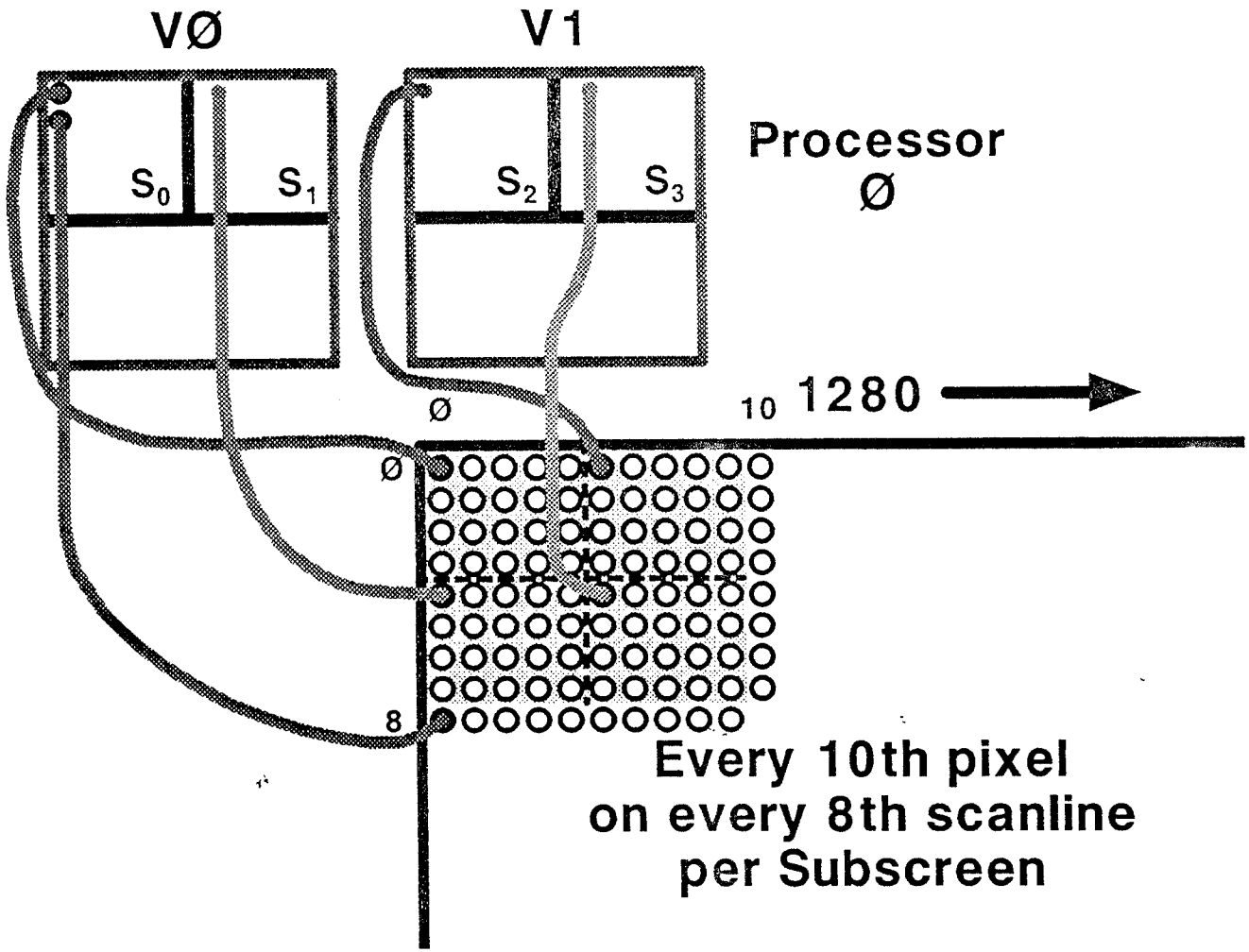
Model 932



HGA93212.030

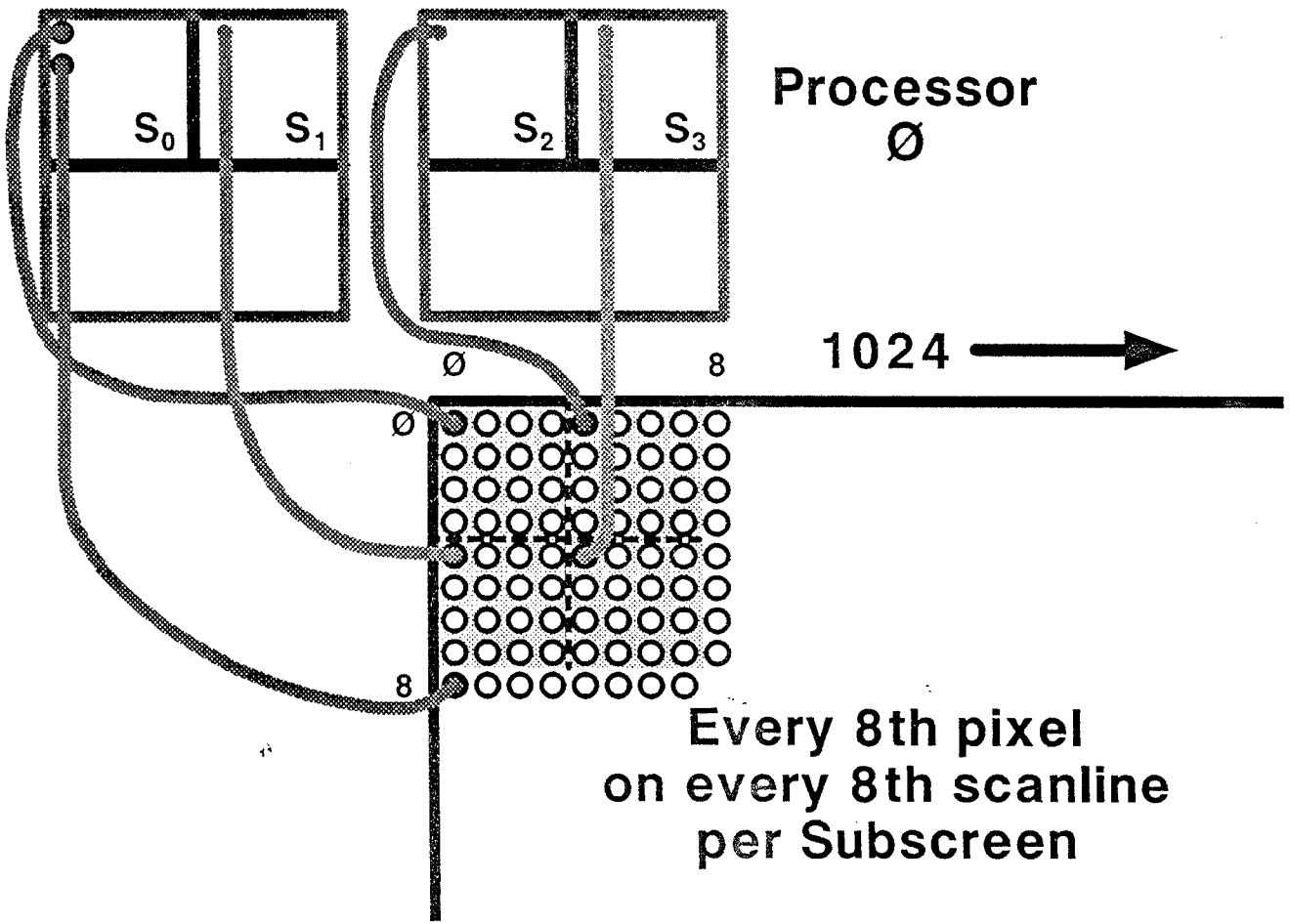
PROCESSING TO SCREEN MAPPING

Model 920



HGA93212.029

PROCESSOR TO SCREEN MAPPING
Model 916



HGA93212.028

Pixel Node - Flag Register

Flag register contains:

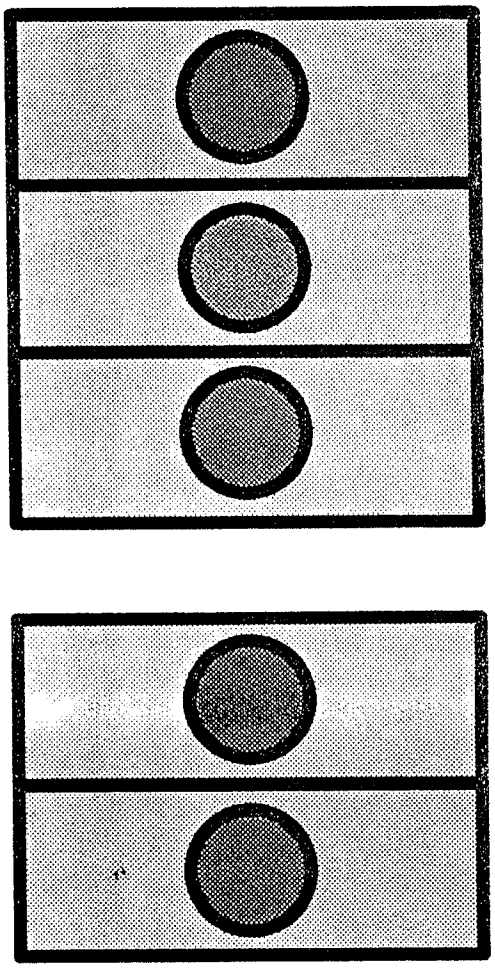
- sss
  - — Sync signal selection flags
- f r
  - — the nodes psync (f for flag) and vsync (r for rdy) flags
- v0 v1
  - — video buffer selection flags
- o
  - — overlay flag

Pixel Node - Flag Register (continued)

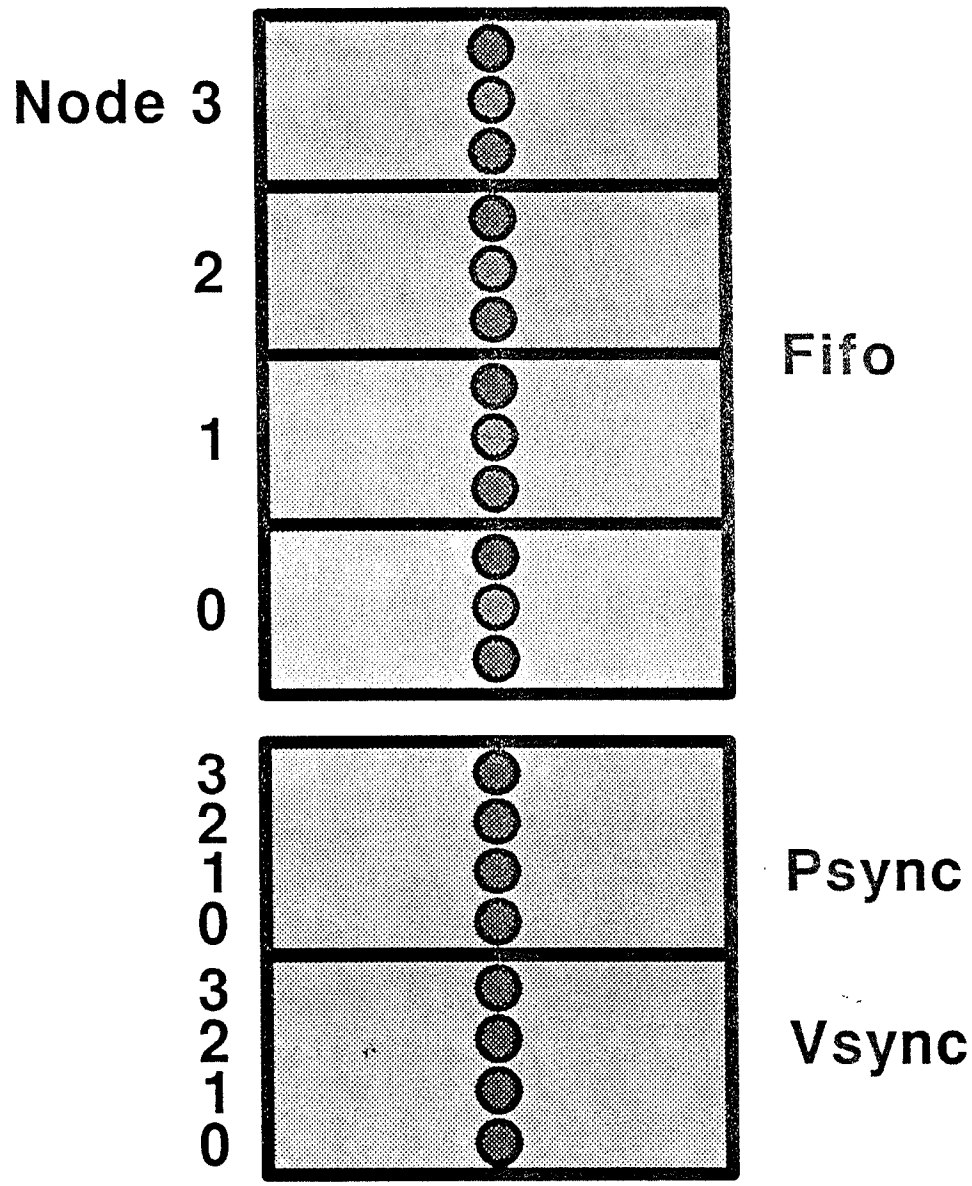
Sync signal values:

- 010
  - — draw empty
- 011
  - — draw half-full
- 100
  - — vertical blanking
- 101
  - — horizontal blanking
- 110
  - — all processors have vsync set
- 111
  - — all processors have psync set

LEDs



HGA93212.019



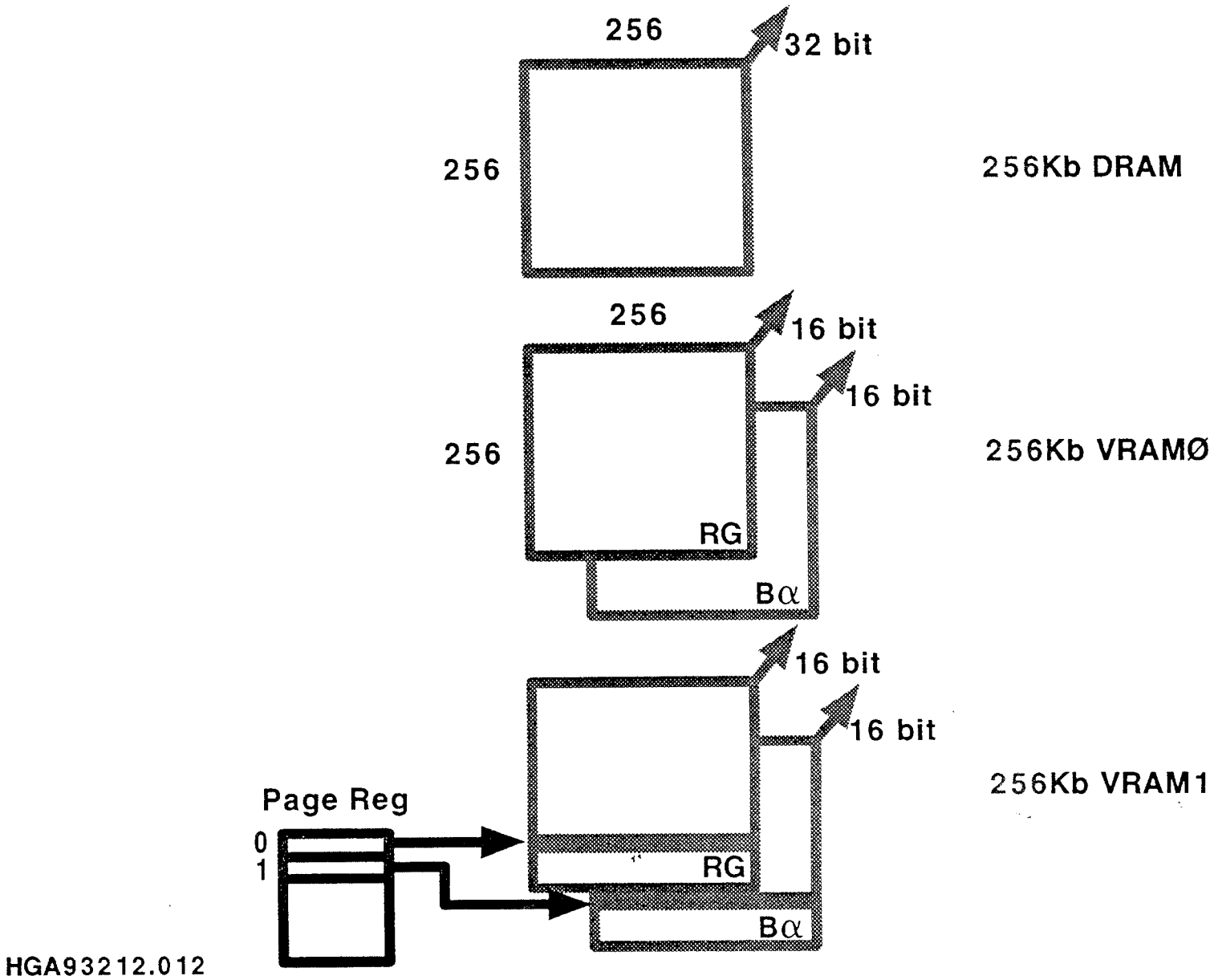
HGA93212.020

Pixel Node - Mode Register

- Mode register must be set by host
- Contains:
  - overlay mode
  - video shift flag
  - gate enable flag
  - serial I/O direction

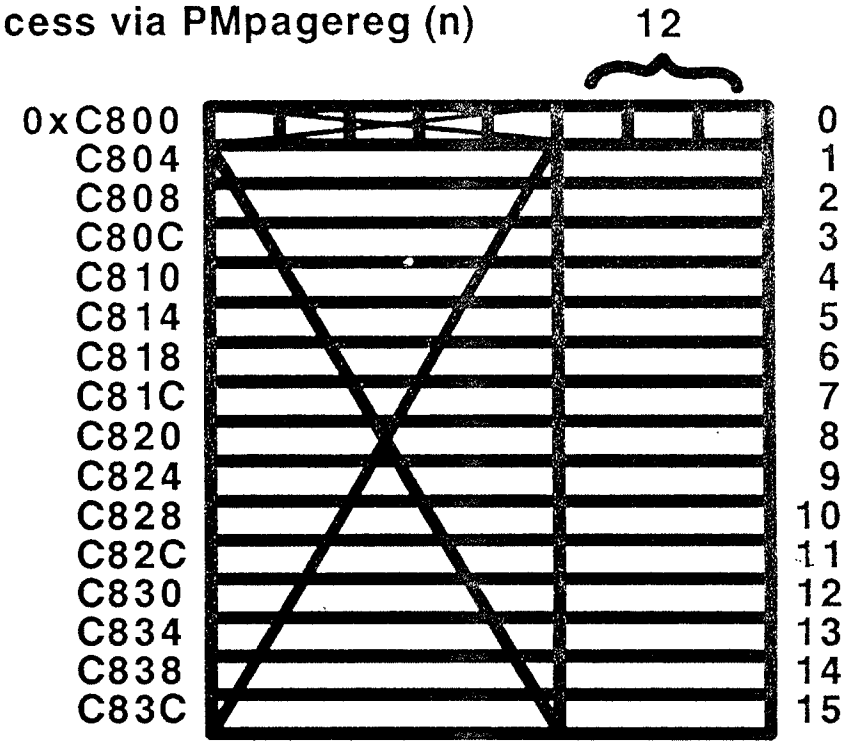
Overlay mode:

- Overlay off
  - rgb values always used
- Overlay on
  - if overlay = 0 use rgb
    if overlay = 255 use ~rgb
    otherwise use overlay
- Overlay force
  - always use overlay
- Overlay mask
  - if high order bit of overlay is set (overlay & 0x80) use the overlay value
    otherwise use rgb


256
32 bit
256
256Kb DRAM
256
16 bit
16 bit
256
256Kb VRAMØ
RG
Bα
16 bit
16 bit
256Kb VRAM1
Page Reg
0
1
RG
Bα
HGA93212.012

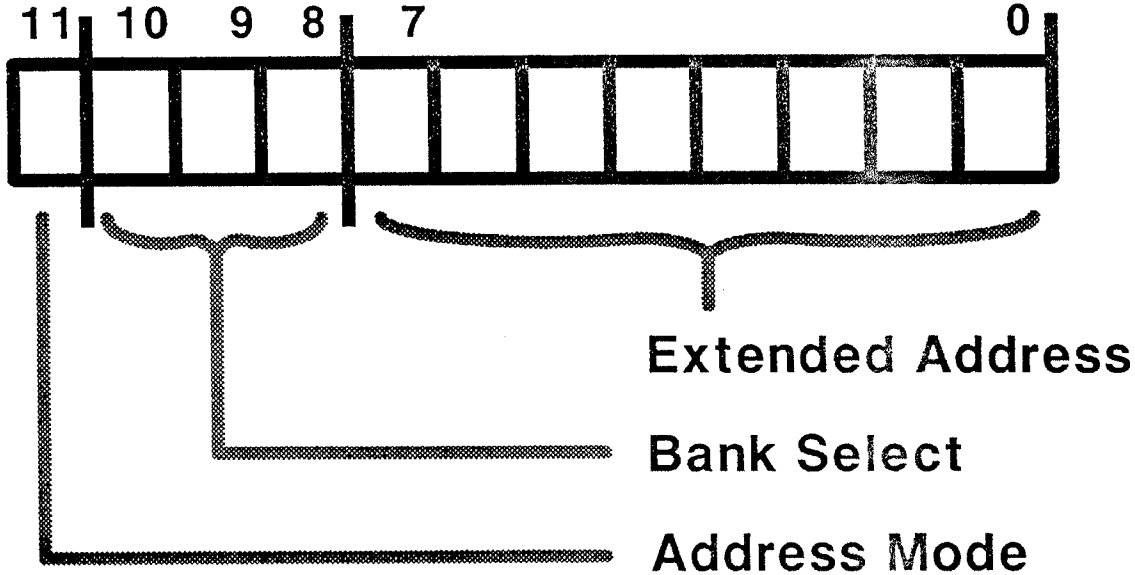
PIXEL NODE-PAGE REGISTERS

- Needed because DSP32 has 16 bit address space
- Used for:
  - DRAM
  - video memory
- Access via PMpagereg (n)



HGA93212.008

PAGE REGISTER FORMAT



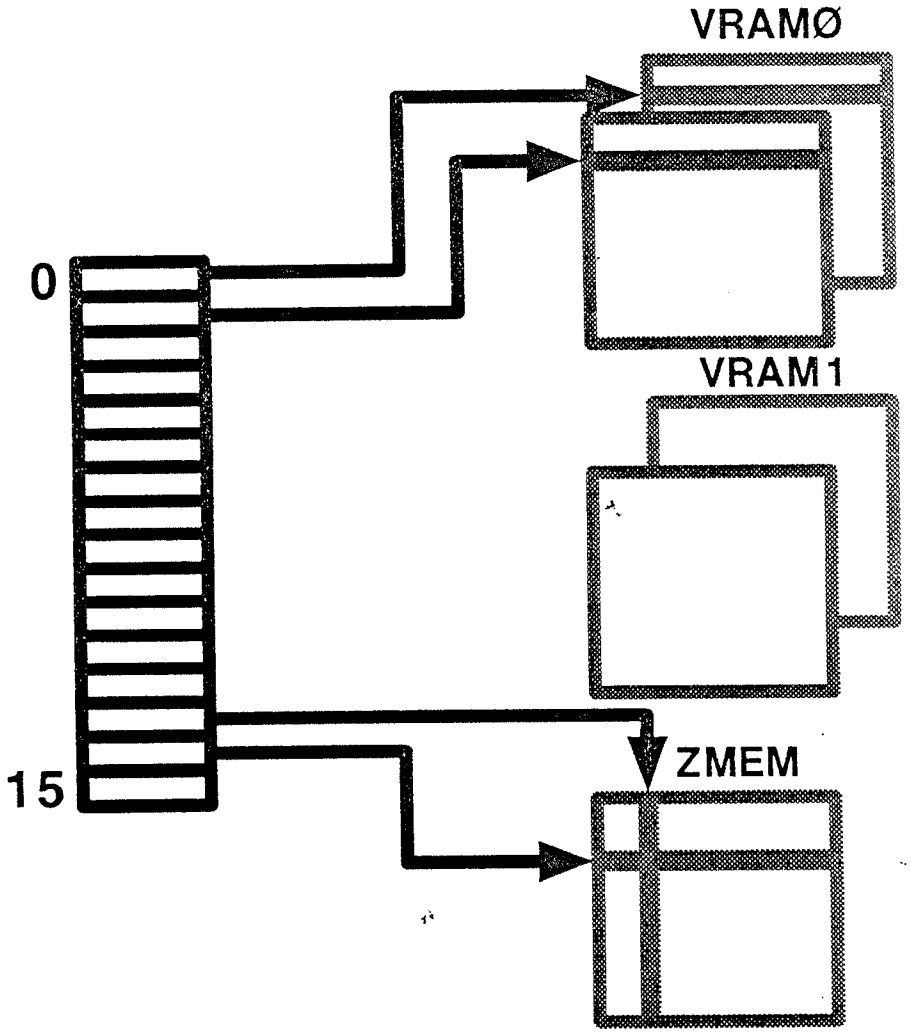
Banks		Modes	
001	ZMEM	0	Fixed Row
100	RGØ	1	Fixed Column
101	BOØ		
110	RG1		
111	BO1		

HGA93212.007

PAGE REGISTER REFERENCE

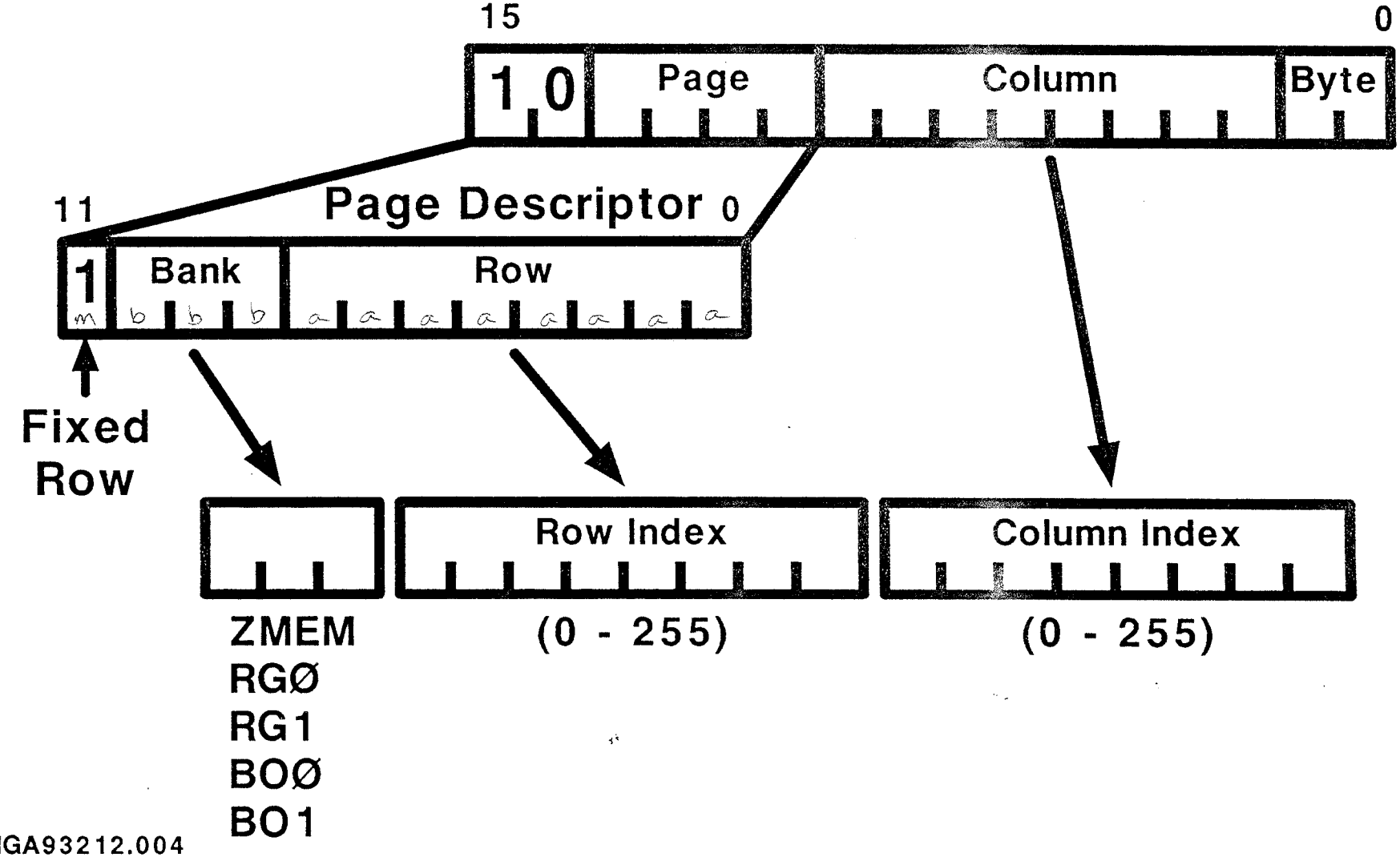
Page Registers address the 16 1Kb blocks from Ox8000 - OxBFFF

VRAM & DRAM REFERENCING



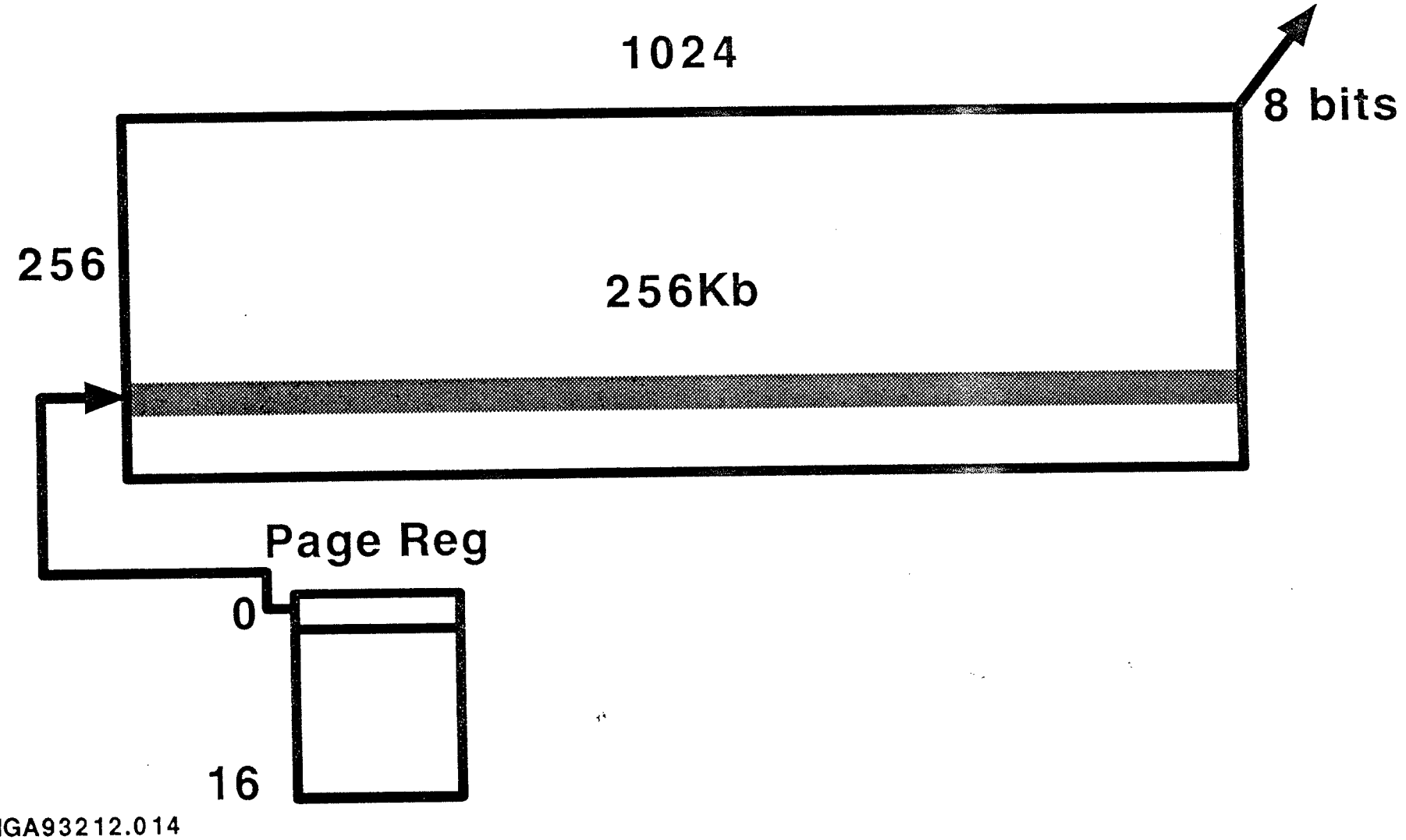
HGA93212.005

FIXED ROW ADDRESSING



HGA93212.004

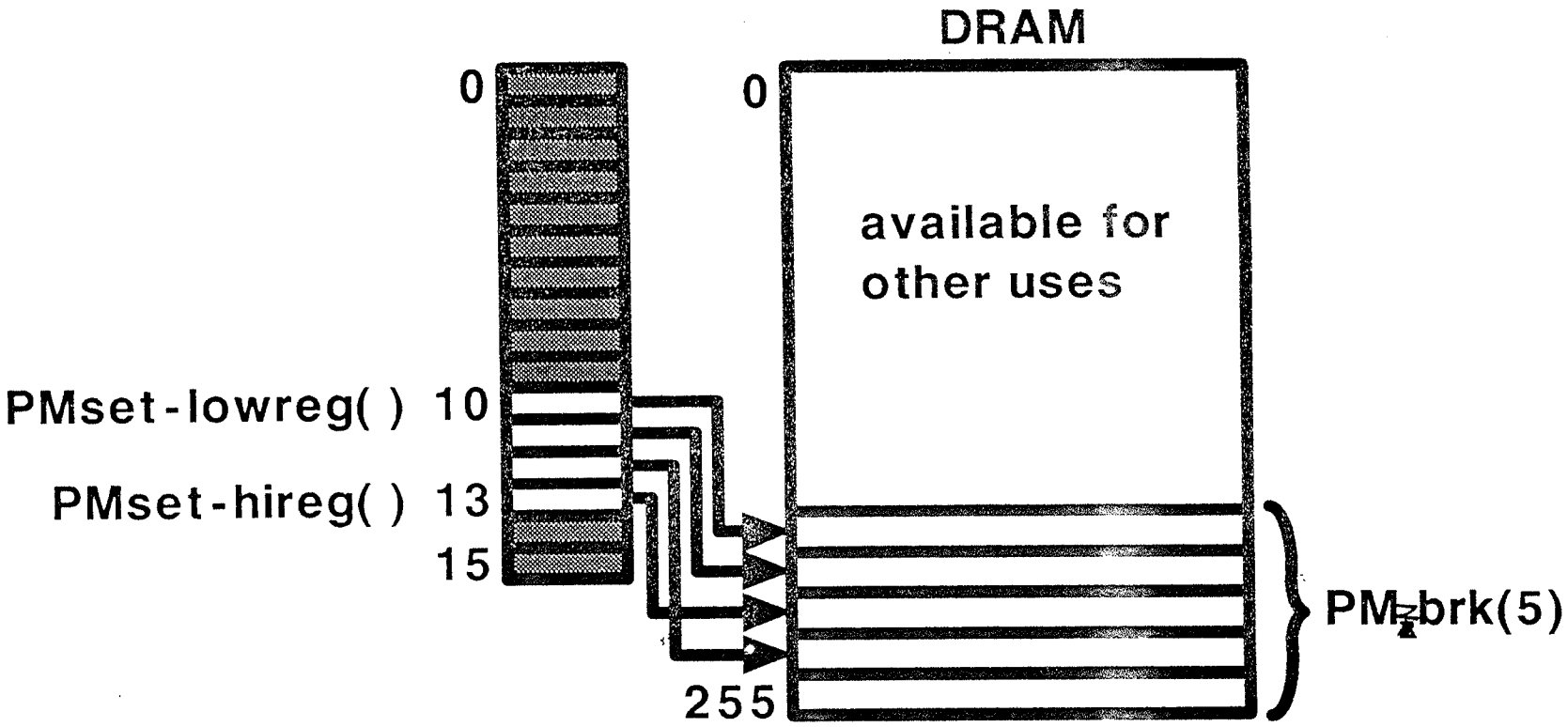
ANOTHER VIEW OF DRAM



HGA93212.014

DRAM ALLOCATION

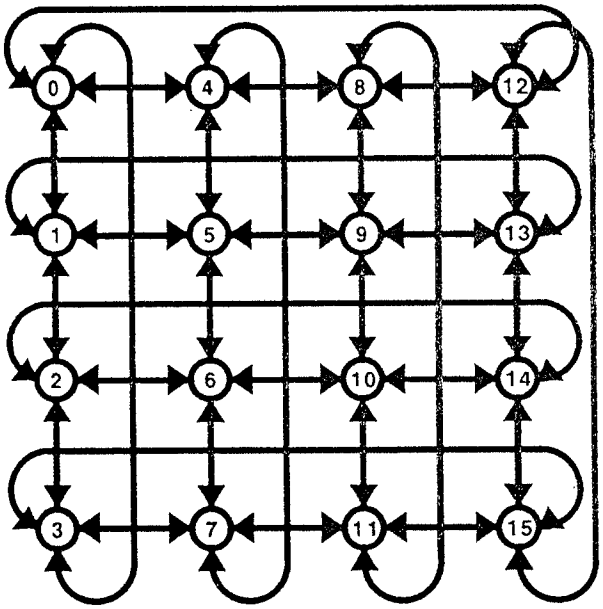
- alternate method of DRAM and page register usage
- simulates malloc( ) type operations
- allocates a subset of page registers



HGA93212.011

SERIAL I/O TRANSFERS

- All nodes simultaneously must send & receive in the same direction
- Same size packets
- Programmed I/O



MODEL 916

- Host controls direction
- ~1.2 MB/sec raw thruput

HGA93212.003

Pixel Nodes - FIFO Rules

- Don't read from an empty FIFO
- Always read all four bytes of each FIFO entry

Pixel Nodes - Testing the Flags

- Connect the signal to the DSP32 sync pin
  - — update the sync signal field of the flags register
  - — wait at least two instruction cycles
  - — test using the the sys and syc flags

DSP32 Architecture - Overview

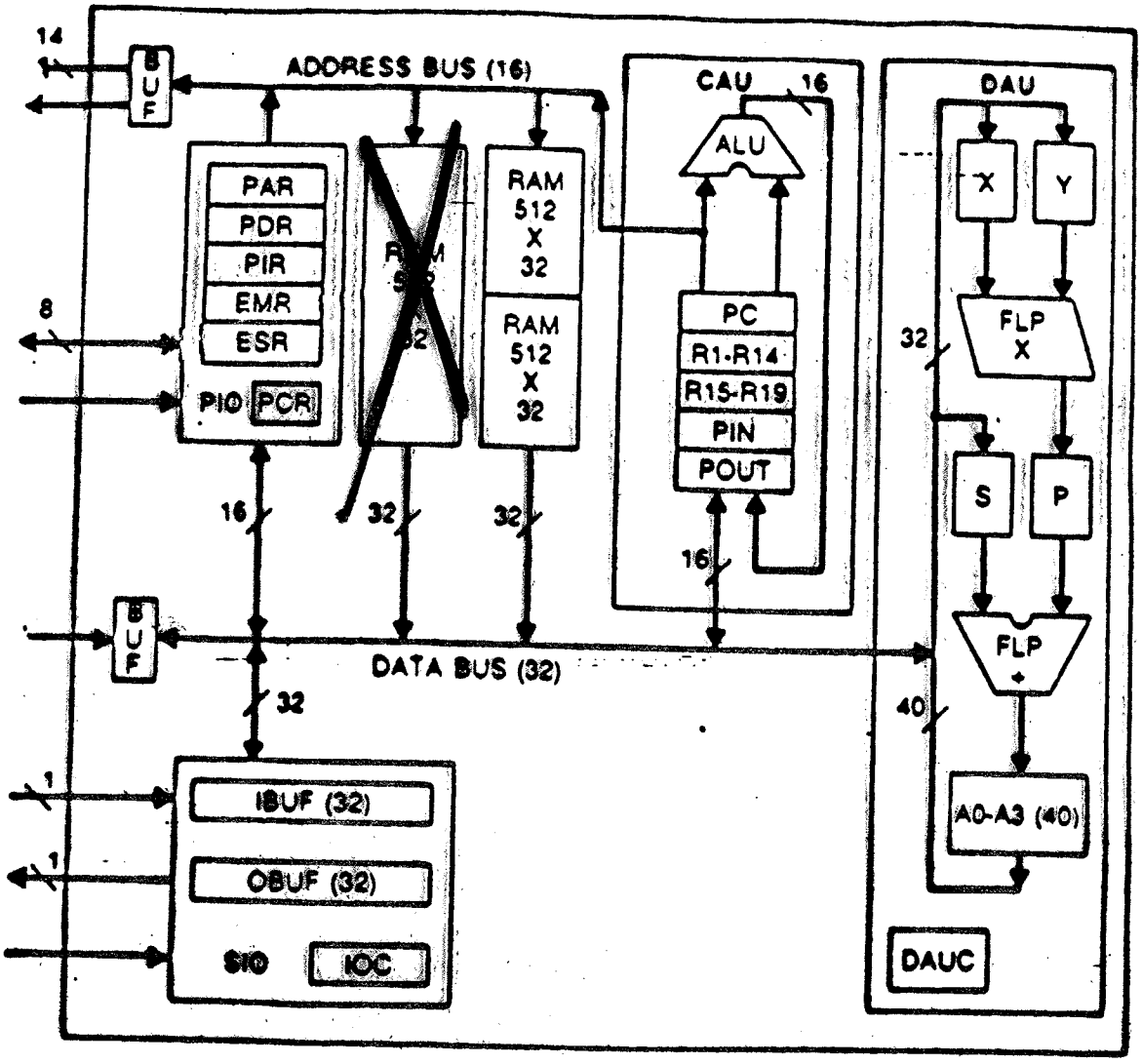
- 5 MIPS
- Up to 10 MFLOPS
- DAU (data arithmetic unit)
  - — 4 40 bit accumulators
  - — highly pipelined - can execute 5 million floating point multiple/add instructions per second
- CAU (control arithmetic unit)
  - — 21 16 bit integer registers
- Parallel I/O
  - — DMA
  - — program controlled
  - — used to communicate with host

DSP32 Architecture - Overview (continued)

- Serial I/O
  - — DMA
  - — program controlled
  - — used for communication between DSP processors

- Data format - least significant byte first
  - — reverse of Sun
  - — same as VAX

DSP32 Architecture



TOP
DO NOT AFFIX OVERLAYS ALONG THIS SURFACE

NOTES:

DSP32 Architecture - DAU

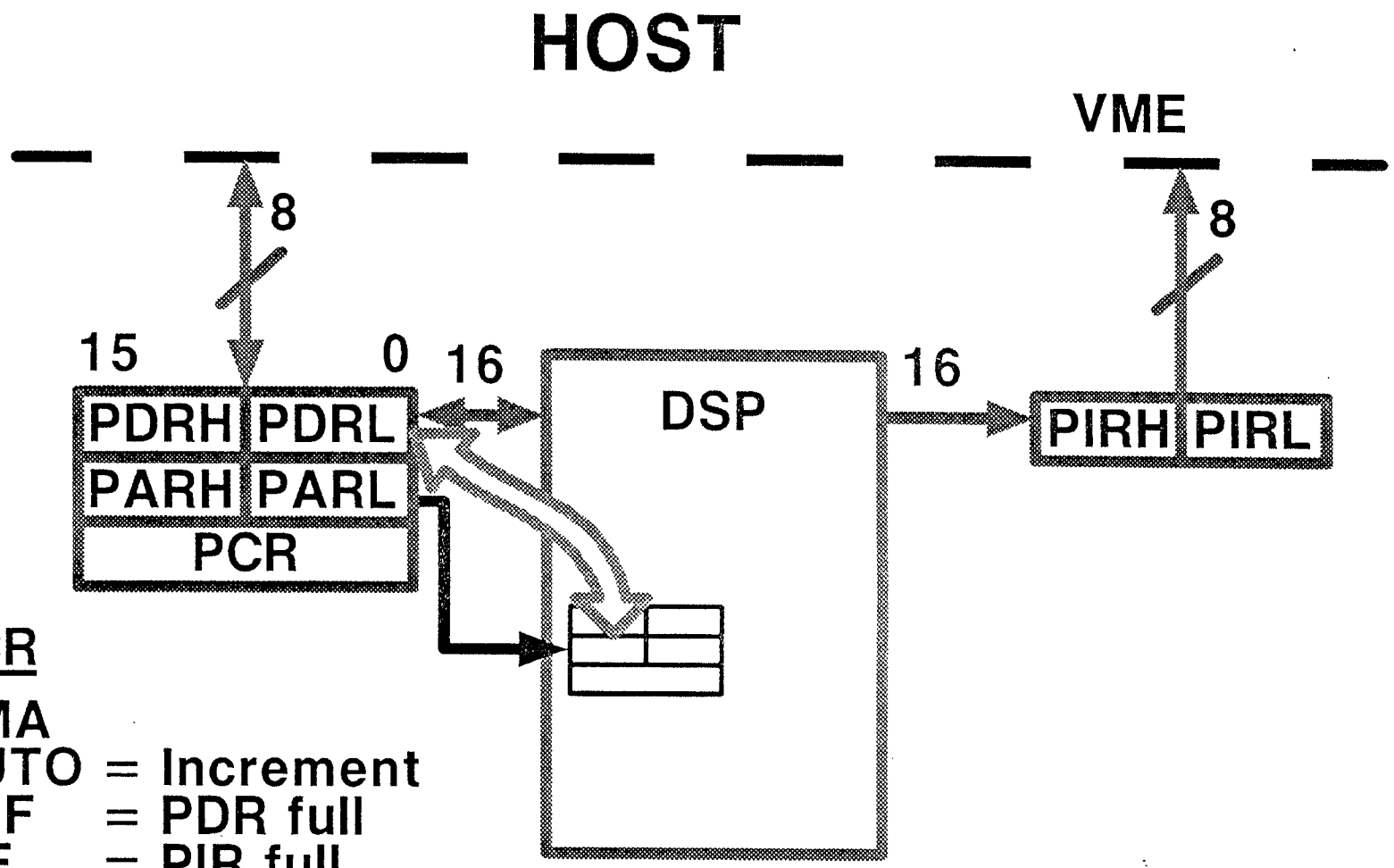
- Highly pipelined
  - — several instructions executing at once
  - — assembler programmers must know about pipelining and latency
  - — not as hard as it sounds
- General form of instruction is:

$$A = [-]B \ \{+,-\} \ C * D$$

- Operands can be:
  - — floating point register
  - — indirect via a general register (r1-r14)
  - — indirect with a post-increment or decrement
  - — indirect with a post-increment from a register (r15-r19)

DSP32 Architecture - Registers

- General registers
  - r1-r14 - general purpose
  - r15-r19 - general purpose - can be used as postincrement registers
  - a0 - a3 - 40 bit floating point accumulators
  - PIN, POUT - general purpose - serial I/O input/output pointers
  - PCR - parallel I/O control register
  - PDR - parallel data register
  - PIR - parallel interrupt register
  - PAR - parallel address register
  - EMR - error mask register
  - ESR - error source register
  - IBUF, OBUF - serial I/O input and output registers
  - IOC - I/O control register



PCR

DMA
AUTO = Increment
PDF = PDR full
PIF = PIR full

•
•
•

HGA93212.018

DSP32 Architecture - PCR Contents

- DMA mode
- autoincrement mode
- PIF flag
  - — set when the DSP writes to the PIR
  - — cleared when the host reads from the PIR
- PDF flag
  - — set when the host writes to the PDR
  - — cleared when the DSP reads from the PDR

DSP32 Architecture - Parallel I/O

- Options:
  - DMA or program control
  - For DMA - autoincrement after read/write
- DMA to/from host:
  - host sets address in PAR
  - host reads from or writes to PDR
- Program control from host:
  - host writes to PDR
  - waits for PDF flag in PCR to be reset
  - DSP reads from PDR
- Program control from DSP:
  - DSP writes to PIR
  - host checks PIF flag in PCR
  - host reads PIR

DEVtools

Components of DEVtools:

- DSP tools
  - C compiler
  - assembler
  - linker
  - math and application libraries
- Host Library
  - Pixel Machine control and communication
  - image input/output
- Pixel Machine Library
  - pipe node I/O
  - pixel node I/O
  - host communication
  - internode communication
  - memory management and access functions

DSP32 Tools

- located in /usr/hyper/devtools/dsp32/{bin, lib, include}

- set shell environment variable: DSP32SL=/usr/hyper/devtools/dsp32
- set path: PATH=$PATH:/usr/hyper/devtools/dsp32/bin
- see documentation
  - — DSP32 C Language Compiler User Manual
  - — DSP32 C Language Compiler Library Reference Guide
  - — DSP32 Software Support Library User Manual
  - — DSP32 Information Manual
  - — DSP32 Application Software Library Reference Manual

DSP32 Commands

- d3ar
  - — archive (library) management utility
- d3as
  - — assembler
- ~~d3cc~~ replaced by devcc (startup code included)
  - — C compiler
- d3ld
  - — linker
- d3nm
  - — object and executable file map listing utility
- d3sim
  - — simulator

DSP32 Libraries

Described in detail in "DSP32 C Language Compiler Library Reference Manual"

- libc
  - — automatically linked in by C compiler
  - — subset of UNIX system libc
  - — printf - for simulator only libpm version used for Pixel Machine
  - — strlen, ctype, etc.
  - — internal functions like mod operations & integer multiply
- libap
  - — Application Library
  - — optimized utility routines for DSP32
  - — matrix multiplication, filter functions, fast trig functions
- libm
  - — standard trig functions, etc.
  - — very slow and big math functions, but they do bounds and error checking

DEVtool Libraries Overview

Host library

- devlib.a
  - — contains all host callable functions

Three DSP libraries and one Pixel Machine library for node programs

- DSP tool libraries
  - — libc
  - — libap
  - — libm
- Pixel Machine Library
  - — libpm

DSP Tools Libraries

DSP libraries

- described in detail in "DSP32 C Language Compiler Library Reference Manual"
  - libc
    - automatically linked in by C compiler
    - for internal functions like mod operations
  - libap
    - Application Library
    - optimized utility routines like matrix multiply, log, etc.
  - libm
    - slow and big math functions, but they do bounds and error checking

DEVtools Library

Pixel Machine Library

- libpm
  - — DSP functions for Pixel Machine architecture
  - — contains Pipe, Pixel, Math and Node (either Pipe or Pixel) functions

DEVtools Host Functions

- initializes machine
- loads executable files into the nodes
- host program interacts with user and communicates with the Pixel Machine
- provide a message polling system - receiving a message triggers a system or user defined action
  - system defined actions include
    - servicing printf requests
    - handling requests for changing serial I/O message-passing direction
    - causing the host program to terminate when the Pixel Machine program has completed
  - user-defined actions could include
    - requests for environment information such as host command line arguments
    - page fault requests for data to be sent to device through pipe or DMA for applications that require more memory than is available on a given node
    - general DMA requests from individual pipe/pixel nodes for SRAM, VRAM, DRAM

DEVtools Message Protocol
Initializing Message System

- Host system maintains table of functions to be called for each message code. For each message code two function pointers are provided:
- user supplies:
  - — function to call when message is received from a pipe node
  - — function to call when message is received from a pixel node
- Process on host polls nodes periodically

DEVtools Message Protocol
Sending a Message

- Node loads message code into PIR register
- Node sets semaphore indicating a message is pending
- Node continues execution
- Host polling process checks PIR status
- Gets PIR value and calls appropriate routine
- Message handling function does its job, then returns to the polling routine
- Host polling routine resets the nodes semaphore
- Node must test semaphore to see if the message has been completed

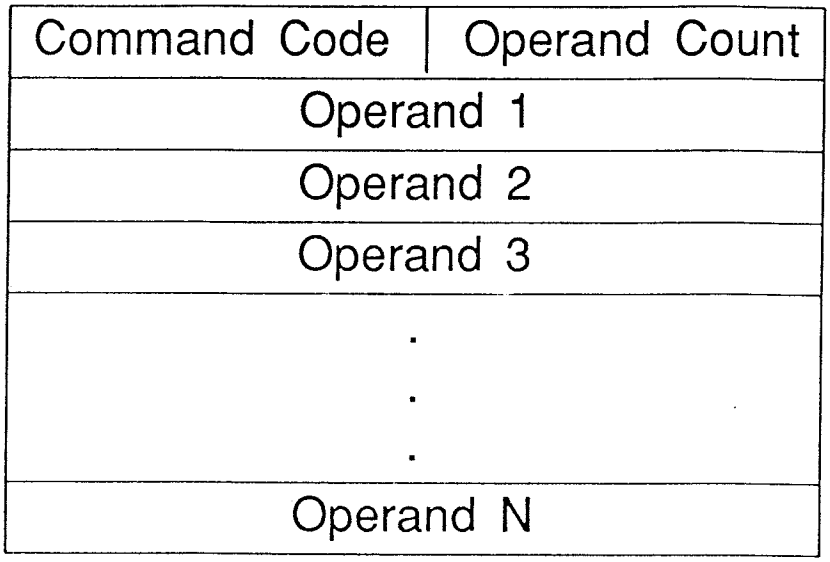
DEVtools Host Device Control

- DEVinit()
  - — opens the device associated with the Pixel Machine
  - — creates a lock file
  - — resets the processors
  - — initializes the message system
- DEVpipe_boot
- DEVpixel_boot
  - — checks whether the specified executable is already loaded
  - — loads an executable into a group of pipe or pixel nodes if needed
- DEVrun()
  - — starts execution of the Pixel Machine
- DEVexit()
  - — halts the Pixel Machine processors
  - — removes the lock file
  - — closes the device

DEVtools Host Message Service

- DEVuser_msg_enable()
  - — initializes entries in the message table
  - — user supplies pointers to the functions to be invoked
- DEVpoll_nodes()
  - — polls nodes looking for messages to be served
  - — provide first and last pipe nodes to be polled
  - — provide first and last pixel nodes to be polled
  - — iteration count
  - — usleep delay time between polls

DEVtools Command Protocol



- Command Code - 16 bits
  - — values > 0 for user use
  - — values <= 0 reserved for system commands
- Operand Count - 16 bits
  - — number of operands to follow
  - — stored as negative number for internal reasons
- Operands - 32 bits
  - — floats or long integers

Command Protocol - Macros

- DEVcommand(command_code, count)
  - — encodes command_code and count into a 32 bit value
- DEVcwrite0 - DEVcwrite9
  - — writes a command with a fixed number of operands in the range of 0 to 9
    - DEVcwrite2(DEVcommand(code, 2), int, i, j)
- DEVwrite0 - DEVwrite9
  - — writes operands associated with a previously written command code
    - DEVcwrite2(DEVcommand(code, 4), int, i, j)
      DEVwrite2(float, x, y)
- DEVwriten
  - — writes an array of operands

Command Protocol - Macros (continued)

- DEVreadn
  - — reads an array of values from the feedback FIFO
- DEVcommand_opcode
  - — extracts the opcode from a command read from the feedback FIFO
- DEVcommand_length
  - — extracts the operand count from a command read from the feedback FIFO
- _alt on the end of the macro name writes to the alternate pipe

System Commands

- Used to implement host functions that cause DSP routines to be invoked implicitly
- Functions that use system commands:
  - — Image upload and download
    - DEVget_scan_line()
    - DEVput_scan_line()
  - — Swapping pipes
    - DEVswap_pipe()
  - — Sync the Pixel Machine and exit
    - DEVwait_exit()
- System commands must be "enabled" before use
  - — only enabled functions are loaded by the linker
    - conserves memory
  - — pipe and pixel programs enable commands using PMenable()

How System Commands Work

- PMenable()
  - — creates an entry in a table of pointers to system command handlers
- Host writes command
  - — negative opcodes reserved for system commands
- PMgetop (pipe nodes) and PMgetcmd (pixel nodes):
  - — read an opcode
  - — check for system opcodes (negative values)
  - — looks for entry in table created by PMenable
  - — calls the system command routine
  - — transparent to user of PMgetop and PMgetcmd
- System commands that are not enabled are:
  - — passed to PMcopycmd by pipe nodes
  - — ignored by pixel nodes

Enabling System Commands

- PMenable(PM_ENABLE_SWAP_PIPE)
  - — required for use of DEVswap_pipe()
  - — must be called by pipe nodes 8 and 17 on dual parallel pipe systems
- PMenable(PM_ENABLE_WAIT_EXIT)
  - — Required for use of DEVwait_exit()
  - — must be called by all pixel nodes

Enabling Upload/Download System Commands

- Called by pixel node routines
  - — Required for use of DEVget_scan_line() and DEVput_scan_line()
- PMenable(PM_ENABLE_GET_SCAN_LINE)
  - — enable upload from VRAM and ZRAM
- PMenable(PM_ENABLE_GET_VRAM)
  - — enable upload from VRAM only
- PMenable(PM_ENABLE_GET_DRAM)
  - — enable upload from DRAM only
- PMenable(PM_ENABLE_PUT_SCAN_LINE)
  - — enable download to VRAM and ZRAM
- PMenable(PM_ENABLE_PUT_VRAM)
  - — enable download to VRAM only
- PMenable(PM_ENABLE_PUT_DRAM)
  - — enable download to DRAM only

Image Upload and Download

Transfers image data between the host and the Pixel Machine

- DEVput_scan_line()
  - — downloads an image from the host to the Pixel Machine
- DEVget_scan_line()
  - — uploads an image from the host to the Pixel Machine

Arguments:

- Buffer code specifies location of image in pixel node memory
  - — DEV_FRONT_BUFFER: the front (currently displayed) portion of VRAM
  - — DEV_BACK_BUFFER: the back (currently non-displayed) portion of VRAM
  - — DEV_VRAM0_BUFFER: to the VRAM0 portion of VRAM
  - — DEV_VRAM1_BUFFER: the VRAM1 portion of VRAM

Color Pixel Formats

- DEV_RGBA_PIXELS:
  - — Red, green, blue, and alpha.
  - — Fastest mode for color download.
  - — 4 bytes per pixel.
- DEV_RGB_PIXELS:
  - — Red, green, and blue.
  - — 3 bytes per pixel.
  - — Alpha channel is cleared.

8 Bit Monochrome Pixel Formats

- DEV_MONO_PIXELS:
  - — The same value is loaded into red, green, and blue to create a monochrome display image.
  - — 1 byte per pixel.

- DEV_MONO_R_PIXELS
  DEV_MONO_G_PIXELS
  DEV_MONO_B_PIXELS
  DEV_MONO_A_PIXELS:
  - — Monochrome stored in the specified color field.
  - — 1 byte per pixel.

16 and 32 Bit Monochrome Pixel Formats

- DEV_MONO_16_PIXELS:
  - Monochrome stored as 16 bit integer.
  - Valid only with DEV_ZRAM_BUFFER.
  - 2 bytes per pixel.

- DEV_DSP_FLOAT_PIXELS:
  - Monochrome stored as 32-bit DSP floating point values on both the host and Pixel Machine.
  - Valid only with DEV_ZRAM_BUFFER.
  - 4 bytes per pixel.

- DEV_IEEE_FLOAT_PIXELS:
  - Monochrome stored as 32-bit IEEE floating point on the host and as DSP floating point on the Pixel Machine.
  - Valid only with DEV_ZRAM_BUFFER.
  - 4 bytes per pixel.

Image File Format

User extensible image file header

- DEVget_image_header()
  - — reads image header and positions file at start of data
- DEVwrite_image_header()
  - — writes image header and positions file at start of data

DEVtools System Library Overview

Types of routines:

- DSP Input/Output
  - — Read/write using DMA
  - — Read/write using PIR
- Conversion to/from IEEE/DSP floating point
- Low level machine control
  - — not usually used by DEVtools users
  - — Open and initialize Pixel Machine
  - — Start/stop processors
  - — Configure pipes
  - — Initialize mode registers based on system configuration

DMA Input/Output

- DEVpixel_read
  DEVpipe_read
  DEVpixel_write
  DEVpipe_write
  - — Read and write blocks of data to/from a designated address
  - — Works without intervention of the program executing on the DSP
  - — Do not perform byte order changes
  - — Address must be an even (word-aligned) memory location.

PIR/PDR Input/Output

Reads and writes data under program control of the DSP

Requires intervention of the program executing on the DSP

- DEVpixel_get DEVpipe_get
  DEVpixel_put DEVpipe_put
  - — use a timeout parameter
- DEVpixel_get_msg DEVpipe_get_msg
  DEVpixel_put_msg DEVpipe_put_msg
  - — does not use a timeout parameter
  - — can perform byte order changes
- DEVpixel_get_pir
  DEVpipe_get_pir
  - — does not use a timeout parameter
  - — reads a single 16-bit quantity

Data Conversion

- DEVdsp_ieee
  - — Converts a DSP floating point number to host format.
  - — Input must have bytes in correct host order.
- DEVieee_dsp
  - — Converts host floating point to DSP format.
  - — Output bytes are still in host order.
- DEVbswaps macro
  - — byte swap short
  - — Performs byte order changes on a 16-bit quantity
  - — Input must be an unsigned integer
- DEVbswapl macro
  - — byte swap long
  - — Performs byte order changes on a 32-bit quantity
  - — Input must be an unsigned integer

Video Look-up-table Routines

- DEVload_color_tables
  - — loads gamma-corrected color table
  - — called automatically by DEVopen based on HYPER_GAMMA environment variable
- DEVload_linear_ramp
  - — used if HYPER_GAMMA is null
- DEVget_color_map
  - — gets current contents of the color tables
- DEVput_color_map
  - — updates contents of the color tables
  - — shadow palate updating should be turned off before calling
- DEVshadow_off
  - — turns off updating of the shadow palate
- DEVshadow_on
  - — turns on updating of the shadow palate

Pixel Machine Libraries
Node Routines

Functions that work in both Pipe and Pixel nodes.

Highlights:

- PMhost_exit
  - — signal host polling function (DEVpoll_nodes) to return to caller
- PMoutpir
  - — output a value to the PIR register
  - — read on host using DEVpipe_get_pir or DEVpixel_get_pir
- printf
  - — up to ten arguments
  - — extensions:
    - %if - used to print numbers that are in the host floating point format (IEEE)
    - %b - binary format

Pixel Machine Libraries
Node Routines (continued)

- PMsetsem
  - set the semaphore
  - used by the message handling system
  - waits for the semaphore to be cleared before setting
- PMusermsg
  - send a user-defined message to the host
  - waits for the semaphore to be cleared before sending
  - does not wait for semaphore to clear before returning
- PMwaitsem
  - waits for the semaphore to be cleared by the host

Pixel Machine Libraries
Node Routines (continued)

- Color conversion macros
  - PMcolor_float
  - PMcolor_int
  - PMint_color
  - PMfloat_color

Pixel Machine Libraries - Pipe Functions

```
typedef struct
{
        short int       opcode;
        short int       count;
        float           *data_ptr;
} PMcmdtype;
```

Command Structure Used by node functions

Pipe Functions - Reading Commands

Functions to read and write opcodes and data to and from the FIFOs

Take care of flag checking and FIFO rules

- PMgetdata
- PMgetop
- PMputcmd
- PMputdata
- PMputop
- PMcopycmd

PMgetop transparently processes system commands

Pixel Machine Libraries - Pixel Functions

Pixel Functions provides functions for:

- accessing pixels through subscreens
- general VRAM access without subscreens
- reading commands from the FIFO
- serial I/O and deinterleaving
- processor synchronization
- ZRAM memory allocation
- miscellaneous

Pixel Machine Libraries - Pixel Functions
Pixel Access Using Subscreens

- PMgetpix / PMputpix
  - — read/write a pixel to/from a subscreen
  - — returns a pointer to the next pixel
- PMqget / PMqput
  - — read/write the next pixel of a scan line
  - — uses pointer returned by PMgetpix/PMputpix
  - — returns a pointer to the next pixel
  - — must not wrap around the end of a scan line
- PMgetscan / PMputscan
  - — read/write a number of pixels of a subscreen for a given scan line
- PMpixaddr
  - — generate a pointer to to a specific pixel

Pixel Machine Libraries - Pixel Functions
VRAM Access Without Subscreens

All functions return a pointer to the next pixel.

- PMv0get / PMv0put
  - — read/write a pixel from video buffer 0
- PMv1get /PMv1put
  - — read/write a pixel from video buffer 1
- PMqget / PMqput
  - — read/write the next pixel of a scan line
  - — uses pointer returned by PMv0get, PMv0put, PMv1get, PMv1put
  - — must not wrap around the end of a row

Pixel Machine Libraries - Pixel Functions
DRAM Access without subscreens

- PMzget / PMzput
  - — read/write a float in DRAM (z memory)
  - — returns a pointer to the next float of the specified row
  - — can use and increment the pointer returned by PMzget/PMzput directly

```
zptr = PMzget(i, j, zorig);
value = *zptr++
*zptr++ = value
```

Pixel Machine Libraries - Pixel Functions
DRAM Access Using Subscreens

Allows access to DRAM corresponding to each VRAM subscreen

Not usually used

- PMgetzbuf / PMputzbuf
  - — read/write a float of a DRAM subscreen
  - — returns a pointer to the next float that can be used directly and incremented up to the subscreen boundary

Pixel Machine Libraries - Pixel Functions
Reading Commands From FIFO

- PMgetcmd
  - — copy opcode, count, and parameters from the FIFO to global PMcommand structure
  - — used at the top of main loop
  - — transparently processes system commands

Pixel Machine Libraries - Pixel Functions
Synchronization and LEDs

- PMpsync
  - synchronize processors
  - returns when all processors have called PMpsync
- PMvsync
  - synchronize processors and waits for vertical blanking
- PMrdyoff
  - call after PMvsync to clear RDY bit
- PMflagled
  - toggle FLAG bit and LED
  - can't be used with PMpsync
- PMrdyled
  - toggle RDY bit and LED
  - can't be used with PMvsync

Pixel Machine Libraries - Pixel Functions
Serial I/O

- PMsioinit
  - — initialize serial I/O
  - — called once in programs that use serial I/O
- PMsiodir
  - — set serial link direction
- PMmsg_setup
  - — loads the address of the serial input buffer
- PMmsg_exchange
  - — sends data
  - — processors should PMpsync before executing

Pixel Machine Libraries - Pixel Functions
Serial I/O - Deinterleaving

- PMinterleave
  - — deinterleave a rectangular region of the screen
  - — row, column or both
  - — produces contiguous blocks in each node
  - — works in VRAM0, VRAM1 or ZRAM

Pixel Machine Libraries - Pixel Functions
ZRAM Allocation

Alternate use of ZRAM for malloc() like operations

- reserves a number of ZRAM rows and page registers for future allocation
- frees user from explicit page register operations and management
- four functions:
  - PMzbrk()
  - PMgetzdesc()
  - PMgetzaddr()
  - PMfreezaddr()
- macros to specify range of page registers
  - PMblock_reg()
  - PMavail_reg()
  - PMset_lowreg()
  - PMset_hireg()

Pixel Machine Libraries - Pixel Functions
Miscellaneous

- PMapply
  - — invokes a function once for each subscreen
  - — can be used with any function that uses subscreen as the first argument
  - — function should not change its parameters
  - — return value is ignored
- PMclear
  - — fill a rectangular region of the screen
- PMdblbuff
  - — enable double buffering mode
- PMswapbuff
  - — switches the video buffer that is being displayed
  - — uses PMvsync and PMrdyoff

Pixel Machine Libraries - Math Functions

- PMx_exp_n
  - computes x**n
  - x is a float, n is an integer from 1 to 20
- PMfdiv
  - floating point division
  - for use by assembler programs
- PMieee_dsp
  - converts an array of host floats to DSP floats
- PMlong_dsp
  - converts an array of long integers to DSP floats
- PMldot
  - special dot product for light sources
- PMnorm
  - normalize a vector
  - returns inverse of magnitude before normalization

Processor Space Mapping - Macros

- PMilo(x) returns the smallest processor space value that, when mapped to screen space, will be $\geq x$.

- PMihi(x) returns the largest processor space value that, when mapped to screen space, will be $\leq x$.

- PMjlo(y) returns the smallest processor space value that, when mapped to screen space, will be $\geq y$.

- PMjhi(y) returns the smallest processor space value that, when mapped to screen space, will be $\leq y$.

↓ cut ↓

Processor Space Mapping

- $i = \frac{1}{N_x}(x - O_x)$
- $j = \frac{1}{N_Y}(y - O_y)$
- $x = i\ N_x + O_x$
- $y = j\ N_y + O_y$

Pixel Node Macros (continued)

- PMmyx
  - test if a given x screen coordinate is in processor space
  - evaluates to (PMilo(x) == PMihi(x))
- PMmyy
  - test if a given y screen coordinate is in processor space
  - evaluates to (PMjlo(x) == PMjhi(x))
- PMxat
  - map subscreen i coordinate to screen space x
- PMyat
  - map subscreen j coordinate to screen space y

Processor Space Mapping

Draw a set of vertical and horizontal lines screen space defined by xmin , xmax , ymin , and ymax .

```
for (x=xmin; x<xmax; x+=delta)
for (y=ymin, y<ymax; y++)
        PUTPIX(x, y, RED);

for (y=ymin, y<ymax; y+=delta)
for (x=xmin; x<xmax; x++)
        PUTPIX(x, y, GREEN);
```

Processor Space Mapping

Code that will run on a 964

```
for (x=xmin; x<xmax; x+=delta)
for (y=ymin, y<ymax; y++)
if ((i=PMilo(x))==PMihi(x)&&(j=PMjlo(y))==PMjhi(y))
        PMputpix(PMputscrns[0],i, j, RED);

for (y=ymin, y<ymax; y+=delta)
for (x=xmin; x<xmax; x++)
if ((i=PMilo(x))==PMihi(x)&&(j=PMjlo(y))==PMjhi(y))
        PMputpix(PMputscrns[0],i, j, GREEN);
```

Processor Space Mapping

A better method is to iterate over processor space

```
imin = PMilo(xmin);
imax = PMihi(xmax);
jmin = PMjlo(ymin);
jmax = PMjhi(ymax);

for (i=imin; i<=imax; i+=delta)
for (j=jmin, j<=jmax; j++)
        PMputpix(PMputscrns[0],i, j, RED);

for (j=jmin, j<=jmax; j+=delta)
for (i=xmin; i<=imax; i++)
        PMputpix(PMputscrns[0],i, j, GREEN);
```

Julia Set Example - Uniprocessor

```
a1 = (rehi - relo) / (xmax - xmin);
b1 = relo - a1*xmin;
a2 = (imhi - imlo) / (ymax - ymin);
b2 = imlo - a2*ymin;

for (y = ymin; y<=ymax; y++)
        for (x=xmin; x<=xmax; x++) {
            re = a1*x + b1;
            im = a2*y + b2;
            done = FALSE;

            for (n=0; n<nmax && !done; n++){
                if ((z = re*re + im*im) <= zmax){
                    temp_im = 2*re*im + Q;
                    re = re*re - im*im + P;
                    im = temp_im;
                }
                else done = TRUE;
            }
            if (done)
                write_pixel(x, y, value_based_on_n);
            else write_pixel(x, y, value_based_on_z);
        }
```

Julia Set Example - Pixel Machine Implementation

```
a1 = (rehi - relo) / (xmax - xmin);
b1 = relo - a1*xmin;
a2 = (imhi - imlo) / (ymax - ymin);
b2 = imlo - a2*ymin;

imin = PMilo( xmin );     jmin = PMjlo( ymin );
imax = PMihi( xmax );     jmax = PMjhi( ymax );
PMfxtoi(a1, b1);          PMfytoj(a2, b2);

for (j=jmin; j<=jmax; j++)
        for (i=imin; i<=imax; i++) {
            re = a1*i + b1;
            im = a2*j + b2;
            done = FALSE;
            for (n = 0; n<maxn && !done; n++) {
                if ((z = re*re + im*im) <= zmax) {
                    temp_im = 2*re*im + Q;
                    re = re*re - im*im + P;
                    im = temp_im;
                }
                else done = TRUE;
            }
            if (done)
                PMputpix(i, j, value_based_on_n);
            else PMputpix(i, j, value_based_on_z);
        }
```

Pixel Node Macros (continued)

- PMfxtoi
  - — map a linear function of x to a linear function of i
- PMfxytoij
  - — map a linear function of (x,y) to a function of (i.j)
- PMfytoj
  - — map a linear function of y to a linear function of j

Processor Space Mapping - Functions

1) $f(x) = A_{xy}x + B_{xy} \quad \rightarrow \quad f(i)\ A_{ij}i = + B_{ij}$

2) $f(x,y) = A_{xy}x + B_{xy}y + C_{xy} \quad \rightarrow \quad f(i,j) = A_{ij}i + B_{ij} + C_{ij}$

3) $f(y) = A_{xy}y + B_{xy} \quad \rightarrow \quad f(i) = A_{ij}y + B_{ij}$

Using DEVtools

- Compiling Pixel Machine programs
- Linking Pixel Machine programs
- Compiling host programs
- Linking host programs
- Loading programs on the Pixel Machine
- Executing Pixel Machine programs
  - — with an associated host program
  - — without an associated host program

Using DEVtools
Compiling Pixel Machine Programs

- Use devcc command
- Typical command to compile a Pixel Machine program:

```
devcc -c ctest.c
```

DSP32 Libraries

- DSP32 libraries that must be included on the link command:
  - -lap
    - if libap is used
  - -lm
    - if libm is used

Using DEVtools

Linking Pixel Machine Programs - Example

- A typical command used to link a Pixel Machine program is:

```
devcc -o ctest ctest.o
-lap
```

Using DEVtools
Compiling Host Programs

- Use devcc cc command
- Command must include the options:

```
-I/usr/hyper/devtools/include
```

- Typical command to compile a host program is:

```
devcc -c
 -I/usr/hyper/devtools/include
 host.c
```

Using DEVtools
Linking Host Programs

- Must include:
  - — devlib.a

- Typical command to link a host program:

```
devcc -o host host.o /usr/hyper/devtools/lib/devlib.a
```

Using DEVtools
Library Options

Sun-3 Host Libraries		
Library Name	Floating Point Option	Profiling Code
devlib.a	68881	no
devlib_p.a	68881	yes
devlib_ffpa.a	fpa	no
devlib_ffpa_p.a	fpa	yes

Sun-4 Host Libraries	
Library Name	Profiling Code
devlib.a	no
devlib_p.a	yes

Using DEVtools
Running a Program

- If there is a host program:
  - just run the host program - it will:
    - load the nodes with the appropriate DSP executable
    - start execution on the Pixel Machine
  - all nodes must have a valid program loaded
  - machine is locked automatically
  - the machine is halted when you call DEVexit
- If there is no host program
  - use hypload to load the nodes
  - use hyprun to start execution
  - use hyphalt to stop execution
  - hyplock and hypfree can be used to lock the machine while your program is running

Using DEVtools
Running a Program
With a Host Program

~~hypload -dall prog.dsp~~ not needed with a host
~~hypload -gall pipe.dsp~~
hostprog

- prog.dsp is the name of the DSP32 executable (for the pixel nodes)
- pipe.dsp is the DSP32 executable (for the pipe nodes)
  - — a program must be loaded into every processor
- hostprog is the name of the host executable

Using DEVtools
Running a Program
Without a Host Program

optional with one user

```
hyplock
hypload -dall prog.dsp
hyprun -dall
hyphalt -dall
hypfree
```

- prog.dsp is the name of the DSP32 executable

Skeleton

Sample program illustrates:

- Host process that sends commands to the pipe
  - source in devtools/skeleton/host/devtest.c
- Pipe process that reads the command and generates other commands
  - source in devtools/skeleton/pipe/p_skel.c
- Pixel node program that reads the commands and creates an image
  - source in devtools/skeleton/pipe/skeleton.c

- Executables are in devtools/skeleton/boot
  - script to run program is called Skel

Sample Programs

- Located in devtools/sample
- Executables in devtools/sample/boot
- Scripts to run programs start with upper case letters

Pipe nodes

- pipe1.dsp
  - — write a string of floats to output FIFO
- pipe2.dsp
  - — read a string of floats from input FIFO
- pipeio.dsp
  - — pass opcodes and data while writing opcodes to host

Sample Programs (continued)

Pixel nodes

- blue.dsp
  - — clear screen to blue
- hello.dsp
  - — printf to host
- julia_set.dsp
  - — Julia set
- led.dsp
  - — turn off RDY and FLAG leds
- mand_set.dsp
  - — growing mandelbrot set
- ntsc_bars.dsp
  - — draw test pattern
- send.dsp
  - — internode message passing via host system
- shift.dsp
  - — serial I/O message passing

Debugging

- Use printf
  - — can limit output to a single processor by using
    if (_nid == 0) printf(...);
- To inspect or change data in a pipe or pixel processor:
  - — hypeek -d0 -a1000

    Displays the contents of location 0x1000 of pixel node 0.
- To update the value:
  - — hypoke -d0 -a1000 -D123

    Sets the contents of location 0x1000 of pixel node 0 to the value 123.

Debugging - Using Feedback

- To inspect the data being received by the FIFO of a processor:
  - load pipe_fb.dsp or pixel_fb.dsp into the processor whose input is to be displayed.
  - execute the program
  - run devfb on the host
- To display the feedback information as commands:
  - follow steps described above
  - pipe output of devfb into devcmd

Debugging - hypdead

- Displays the location at which a DSP process halts
  - hypdead -d0

```
/*
 *      Version: @(#)scrn.c     1.1
 *
 *      Sample program showing 3 different methods for dealing
 *      with subscreens at run time.
 *      This sample illustrates the use of subscreens only.
 *      In general using PMputpix is not a fast way to clear the screen.
 */

#include        <pxm.h>

PMpixeltype     backgnd = {
        PMint_color(0),         /* red intensity */
        PMint_color(117),       /* green intensity */
        PMint_color(140),       /* blue intensity */
        PMint_color(0)          /* overlay value */
};
main( )
{
register int    i, j;
register int    cnt;

        /*
                .... clear the screen to backgnd color
                using processor coordinates ( i, j ) ....
         */
        for ( j = 0; j <= PMjmax; j++ )
        {
                for ( i = 0; i <= PMimax; i++ )
                {
#ifdef FASTEST
                        PMputpix( PMscrns[0], i, j, &backgnd );
                        if (PMmx)
                        {
                                PMputpix( PMscrns[1], i, j, &backgnd );
                                if (PMmy)
                                {
                                        putpix( PMscrns[2], i, j, &backgnd );
                                        putpix( PMscrns[3], i, j, &backgnd );
                                }
                        }
#endif
#ifdef FASTER
                        for (cnt =0; cnt< PMnindex; cnt++)
                        {
                                PMputpix( PMscrns[cnt], i, j, &backgnd );
                        }

#else /* FAST : best all around choice */

                        PMapply(PMputpix, i, j, &backgnd );
#endif
                }
        }
}
```

```
/*
 *      Version: @(#)point.c    1.1
 */


#include        "pxm.h"


void
point( screen, x, y, color )

register PMsubscrn      *screen;
float                   x, y;
PMpixeltype             *color;

{

        register short  i, j;

        /*
                ...     render a point
        */


        i = PMilo( screen, x );
        if ( i == PMihi( screen, x ) )
        {
                j = PMjlo( screen, y );
                if ( j == PMjhi( screen, y ) )
                        PMputpix( screen, i, j, color );
        }

}
```

```
From pxm.h ...
        /*

                ..... C language definitions and macros .....
        */

typedef struct {
                short   r;      /* red value */
                short   g;      /* green value */
                short   b;      /* blue value */
                short   o;      /* overlay value */
        } PMpixeltype;

typedef struct {

                /*
                        .... Floating Point Subscreen Variables ....
                */

                float   Nx;     /* Number of processors in X dimension */
                float   Ny;     /* Number of processors in Y dimension */
                float   Ox;     /* Offset of processor in X */
                float   Oy;     /* Offset of processor in Y */

                /*
                        .... Floating Point Subscreen Constants ....
                */

                float   ai;             /*   ai = (1.0 / Nx)         */
                float   aj;             /*   aj = (1.0 / Ny)         */
                float   bilo;           /*   0.5 - (Ox + 1.0) * ai   */
                float   bihi;           /*  -0.5 - (Ox  * ai)        */
                float   bjlo;           /*   0.5 - (Oy + 1.0) * aj   */
                float   bjhi;           /*  -0.5 - (Oy * aj)         */

                /*
                        .... Fixed Point Subscreen Constants ....
                */

                short   ifix, jfix;     /* current buffer info */

        } PMsubscrn;

/*
        ..... FIFO I/O Buffer definition .....
*/

typedef union
{
        short   word[128];
        float   flt[64];
} PMbufftype;

/*
        ... PMcommand buffer definition
*/

typedef struct
{
        short int       opcode;
        short int       count;
        float           *data_ptr;
}PMcmdtype;
```

```
/*
 *      Global variables
 */


/*      all nodes                                   */
extern int      PMnode;         /* node identification number   */
extern int      PMnx;           /* number of drawing nodes in x */
extern int      PMny;           /* number of drawing nodes in y */
extern int      PMox;           /* drawing node's offset in x   */
extern int      PMoy;           /* drawing node's offset in y   */
extern char     PMsid[16];      /* software name (16 chars)     */
extern int      PMsem;          /* semaphore                    */
extern int      PMmodel;                /* model code                   */
extern int      PMvideo;                /* video format code            */
extern int      PMpipe;         /* pipe mode code               */
extern int      PMxmax;         /* maximum x value              */
extern int      PMymax;         /* maximum y value              */


extern  PMcmdtype PMcommand;    /* PMcommand struct with Opcode,Count and
                                 * DataPtr for reading and writing FIFO's
                                 */


/*      for Pixel nodes  only   */
extern int      PMimax; /* max pixels in I direction in processor space */
extern int      PMjmax; /* max pixels in J direction in processor space */
extern int      PMmx;   /* more processing in x direction? boolean      */
extern int      PMmy;   /* more processing in y direction? boolean      */

        /*
         *      Table of Values for PMmx and PMny
         *
         *
         *              ----------------------
         *              |model  |PMmx   |PMmy  |
         *              |-----  |----   |---   |
         *              |964    | 0     | 0    |
         *              |940    | 1     | 0    |
         *              |932    | 1     | 0    |
         *              |920    | 1     | 1    |
         *              |916    | 1     | 1    |
         *              ----------------------
         */

extern int      PMnindex; /* no. of subscreens (2*PMmy+PMmx+1)          */
extern int      PMmodel;        /* coded pixel machine model    */

extern PMsubscrn *PMscrns[4];   /* initialized array of SubScreen pointers */
```

Quick synopsis of DEVtools library functions
(see corresponding man page for more information)
1.0ga

Host Functions (3H)

DEVexit(3H) halts processors, closes Pixel Machine device
DEVinit(3H) opens and initializes Pixel Machine device
DEVput_scan_line(3H) download an image or a portion of an image to a Pixel Machi
DEVrelease_pipe_semaphore(3H) clear the software semaphore in the memory of one c
DEVswap_pipe(3H) switch primary and alternate pipes of a dual-pipe system
DEVuser_msg_enable(3H) define a message code and specify functions to be called
DEVwait_exit(3H) flush commands in pipe, then call DEVexit
DEVwrite(3H) macros to write to the Pixel Machines pipelines and read commands

Math Functions (3M)

PMcos(3M) fast cosine for -PI/2 <= theta <= PI/2
PMfdiv(3M) perform floating point division
PMieee_dsp(3M) convert IEEE float to DSP float
PMldot(3M) specialized dot product for light sources
PMlong_dsp(3M) convert an array of longs to float
PMnorm(3M) normalize a vector and return its length
PMpow(3M) power function
PMsin(3M) fast sine for -PI/2 <= theta <= PI/2
PMsqrt(3M) sqare root function
PMx_exp_n(3M) integer power function

Pixel or Pipe Node Functions (3N)

PMcolor_float(3N) macro that converts internal color value to floating point
PMcolor_int(3N) macro that converts internal color value to an integer
PMcommand(3N) data structure used for FIFO commands
PMdelay(3N) do nothing for a specified time
PMenable(3N) enable processing of selected system commands
PMfloat_color(3N) macro that converts floating point value to internal color
PMhost_exit(3N) signal DEVpoll_nodes to return to caller
PMint_color(3N) macro that converts an integer to an internal color value
PMoutpir(3N) output a value to the PIR register
PMsetsem(3N) set the semaphore
PMusermsg(3N) send a user defined message to the host
PMwaitsem(3N) wait for semaphore to clear
printf(3N) formatted output conversion on host

Pipe Node Functions (3P)

PMbus_wait(3P) waits until control of the broadcast bus is granted
PMcopycmd(3P) copy opcode, parameter count, and data from input to output FIFO of
PMfb_off(3P) direct output commands to the regular output FIFO
PMfb_on(3P) direct output commands to the feedback FIFO
PMgetdata(3P) get data from a pipe node FIFO
PMgetop(3P) get opcode and parameter count from input FIFO of a pipe node
PMputcmd(3P) write opcode, parameter count, and parameters to the output FIFO of
PMputdata(3P) write parameters to the output FIFO of a pipe node
PMputop(3P) write opcode and parameter count to the output FIFO of a pipe node
PMswap_pipe(3P) release the broadcast bus and request it again

Pixel Node Functions (3X)

PMapply(3X) apply a function to all subscreens
PMclear(3X) fill a rectangular region of the screen
PMcopy_f(3X) fast but dangerous 32 bit D/VRAM copy
PMcopy_s(3X) safe 32 bit DRAM or VRAM copy

DEVclear_buffer(3S) clear pixel memory of the frame buffer to specified value.
DEVclose(3S) closes the Pixel Machine
DEVdsp_ieee(3S) convert from the DSP32 floating-point format to the IEEE floating-p
DEVerror(3S) generate an error message on standard error
DEVfifo_parallel(3S) configure a pipe board to operate in parallel mode
DEVfifo_read(3S) reads a block of four byte values from a pipe FIFO
DEVfifo_reset(3S) resets all FIFOs on a pipe board
DEVfifo_serial(3S) configure a pipe board to operate in serial mode
DEVfifo_write(3S) writes a block of four byte values to a pipe FIFO
DEVget_color_map(3S) reads the color tables from video controller board and retu
DEVget_image_header(3S) read the DEVtools image header from a file
DEVget_pixel(3S) read a pixel from the frame buffer
DEVget_scan_line(3S) read one or more scan lines from a frame buffer
DEVieee_dsp(3S) convert from the host's floating-point format to the DSP32 floating
DEVload_color_tables(3S) reads file of gamma calibration values and sets col
DEVlock(3S) manage Pixel Machine locks
DEVopen(3S) make a Pixel machine available to a user program
DEVpipe_boot(3S) load a Pixel Machine executable into specified set of pipe
DEVpipe_enable_error_halt(3S) set DSP to halt on hardware errors
DEVpipe_get(3S) reads a block of memory from a pipe DSP
DEVpipe_get_msg(3S) read a block of memory from a pipe DSP
DEVpipe_get_pir(3S) read the PIR register of a pipe DSP
DEVpipe_halt(3S) halt a pipe node processor
DEVpipe_id_check(3S) check status of node's ID
DEVpipe_id_print(3S) reads the node ID from a processor
DEVpipe_put(3S) sends a block of data to a pipe DSP
DEVpipe_read(3S) reads a block of memory from a pipe DSP
DEVpipe_run(3S) begins execution of programs loaded into specified pipe nodes
DEVpipe_write(3S) write a buffer to a pipe DSP
DEVpixel_boot(3S) load a Pixel Machine executable into specified set of pixel
DEVpixel_buffer(3S) selects the frame buffer to be displayed
DEVpixel_enable_error_halt(3S) set DSP to halt on hardware errors
DEVpixel_get(3S) reads a block of memory from a pixel DSP
DEVpixel_get_msg(3S) read a block of memory from a pixel DSP
DEVpixel_get_pir(3S) read the PIR register of a pixel DSP
DEVpixel_halt(3S) halt a pixel node processor
DEVpixel_id_check(3S) check status of node's ID
DEVpixel_id_print(3S) reads the node ID from a processor
DEVpixel_id_write(3S) writes a node id block to a reserved location in a pixel no
DEVpixel_mode_init(3S) initialize pixel board mode register
DEVpixel_mode_overlay(3S) set overlay mode in the pixel mode register
DEVpixel_overlay(3S) update overlay mode in pixel processors
DEVpixel_put(3S) sends a block of data to a pixel DSP
DEVpixel_read(3S) reads a block of memory from a pixel DSP
DEVpixel_run(3S) begins execution of programs loaded into specified pixel no
DEVpixel_system(3S) macros used to fetch system description information from th
DEVpixel_write(3S) write a buffer to a pixel DSP
DEVpoll_nodes(3S) polls DSP processors for messages
DEVput_color_map(3S) update color tables from video controller board and return
DEVput_image_header(3S) write a DEVtools image header to a file
DEVput_pixel(3S) write a pixel into the frame buffer
DEVread_z(3S) reads a buffer of bytes from the Z memory of a pixel node
DEVserial_direction(3S) updates the serial I/O link direction
DEVshadow_off(3S) turns off updating of color lookup tables from shadow table
DEVshadow_on(3S) turns on updating of color lookup tables from shadow tables
DEVsswapl(3S) convert between DSP32 long integer and host long integer
DEVswap_long(3S) convert from DSP32 long integers to host long integers
DEVswap_short(3S) convert from DSP32 short integers to host short integers
DEVwrite_z(3S) writes a buffer of bytes into the Z memory of a pixel node