RDS 3000 USER'S GUIDE

April 1982

Ikonas Graphics System, Inc.
a subsidiary of
Adage, Inc.
Copyright 1982, IKONAS GRAPHICS SYSTEMS, INC.

The information contained herein is subject to change without notice.

IKONAS GRAPHICS SYSTEMS, INC., makes no warranty of any kind, express or implied, with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IKONAS shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

This document is proprietary to IKONAS GRAPHICS SYSTEMS, INC., and shall not be reproduced in whole or in part without the written authorization of IKONAS GRAPHICS SYSTEMS, INC.

Part Number 10-301-095-10A
# TABLE OF CONTENTS

**THE IKONAS BUS**
- IKONAS Conventions ........................................ 2
- IKONAS Bus Addressing ........................................ 4
- IKONAS Memory Map ........................................... 5
- IKONAS BUS Function Codes .................................. 6

**THE IKONAS DR64**
- Configuring the DR64 Image Memory ......................... 9
- Addressing the DR64 in LORES Mode ......................... 11
- Addressing the DR64 in HIRES Mode ......................... 14
- Accessing the DR64 in WORD Mode .......................... 15
- The DR64 Write Mask .......................................... 16

**THE GM64 GRAPHICS MEMORY**
- Mask Mode Writes ............................................ 19
- The Sender ID .................................................. 20
- Z-buffer Option ............................................... 21

**THE VIDEO DATA PATH** ........................................... 24

**THE FRAME BUFFER CONTROLLER**
- Definitions of Video Terms ................................ 25
- FBC Video Formatting - Sync Control ....................... 27
- FBC Video Formatting - Viewport ............................ 29
- FBC Video Formatting - Window .............................. 31
- Zoom .................................................................. 32
- Cursor .................................................................. 33
- FBC Flag Bits and Mode Settings ............................ 34
- Summary of FBC Control Registers ......................... 35
- The Video Data Path as it Leaves the FBC ................ 36

**THE IKONAS XBS34 VIDEO SWITCH**
- XBS34 Model ..................................................... 37
- Transparent Mode ............................................... 38
- Red Channel to All Three Outputs ......................... 39

**THE LUVO COLORMAP AND VIDEO OUTPUT MODULE**
- Colormaps ......................................................... 40
- The colormap used for color correction .................... 41
- Colormaps used for pseudocolor ............................. 42
- Colormap Pages and the Cursor ............................... 43
- The Channel Switch ............................................ 44
- LUVO data routing - full-color operation .................. 45
- LUVO data routing - pseudocolor operation ............... 46
- Setting up the colormaps ...................................... 47
- LUVO data routing - IKONAS bus access .................... 48
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>THE MPC 68000 COMPUTER</td>
<td>123</td>
</tr>
<tr>
<td>MPC Memory Map</td>
<td>125</td>
</tr>
<tr>
<td>IKONAS Bus Access</td>
<td>126</td>
</tr>
<tr>
<td>IKONAS Address Translation Tables</td>
<td>127</td>
</tr>
<tr>
<td>Image Memory Address Translation</td>
<td>128</td>
</tr>
<tr>
<td>Segment Memory Address Translation</td>
<td>129</td>
</tr>
<tr>
<td>Serial Port Setup</td>
<td>130</td>
</tr>
<tr>
<td>Serial Port Status Register</td>
<td>131</td>
</tr>
<tr>
<td>Serial Port Programming Sequence</td>
<td>132</td>
</tr>
<tr>
<td>Programmable Timer (MPC256)</td>
<td>133</td>
</tr>
<tr>
<td>IKONAS Bus Access to the MPC</td>
<td>134</td>
</tr>
<tr>
<td>Accessing MPC Memory from the IKONAS Bus</td>
<td>135</td>
</tr>
<tr>
<td>Table of I/O Addresses</td>
<td>136</td>
</tr>
</tbody>
</table>
THE IKONAS BUS

---

CAMERA

DR64 IMAGE MEMORY

FBC VIDEO CONTROL MODULE

ISS34 VIDEO CROSS BAR SWITCH

LUVG COLOR MAP & VIDEO OUTPUT

---

---

BPS GRAPHICS MEMORY

PROGRAM MODULE

SRS FAST MEMORY

MA1024 IROTATION MODULE

IF/IK HOST INTERFACE

---

external interfaces<--> S8000 COMPUTER
and peripherals MODULE

1256K RAM

---

---

/IKONAS/BUS/
THE IKONAS BUS

The IKONAS BUS connects all elements of the RDS3000 system.

The IKONAS bus:

- is synchronous
- starts a bus cycle every 200 ns
- checks for a new bus master every cycle

Each IKONAS bus cycle transfers the following information:

- 24 bits of address
- 5 bits of function code
- 32 bits of data
IKONAS Conventions

The IKONAS is a 32-bit machine, so we have some conventions for representing 24-bit addresses and 32-bit data items:

- `yyyyy` to represent a 24-bit number whose upper 14 bits are `yyyyy` and whose lower 10 bits are `xxxx`
- `hhhhh` to represent a 32-bit number whose upper 16 bits are `hhhhh` and whose lower 16 bits are `11111`

All numbers are octal. Examples:

`20200$, 100$100$, 202\0`
IKONAS Bus Addressing

The following rules apply to IKONAS addresses in general:

- The lower half of memory (bit 23=0) is for DR64s - image memory.
- The upper eight of memory (bit 21-23=111) is for MPC RAM and I/O.
- Everything in between is for IKONAS modules and fast RAM.
- The upper eight bits of address are used to distinguish between IKONAS modules.
<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3770001777</td>
<td>reserved for multiple-MPC systems</td>
</tr>
<tr>
<td>34100000</td>
<td></td>
</tr>
<tr>
<td>3407701777</td>
<td></td>
</tr>
<tr>
<td>34040000</td>
<td></td>
</tr>
<tr>
<td>3403701777</td>
<td>(upper 16 bits unused by MPC)</td>
</tr>
<tr>
<td>34000000</td>
<td></td>
</tr>
<tr>
<td>3370001777</td>
<td>reserved for future use</td>
</tr>
<tr>
<td>30300000</td>
<td></td>
</tr>
<tr>
<td>3020001777</td>
<td></td>
</tr>
<tr>
<td>302000042</td>
<td></td>
</tr>
<tr>
<td>302000041</td>
<td></td>
</tr>
<tr>
<td>302000040</td>
<td></td>
</tr>
<tr>
<td>3010001777</td>
<td>XBS34 reserved</td>
</tr>
<tr>
<td>301000046</td>
<td></td>
</tr>
<tr>
<td>301000045</td>
<td></td>
</tr>
<tr>
<td>301000040</td>
<td></td>
</tr>
<tr>
<td>3000007777</td>
<td>FBC reserved</td>
</tr>
<tr>
<td>3000000100</td>
<td></td>
</tr>
<tr>
<td>3000000777</td>
<td></td>
</tr>
<tr>
<td>3000000400</td>
<td></td>
</tr>
<tr>
<td>3000000377</td>
<td></td>
</tr>
<tr>
<td>3000000310</td>
<td></td>
</tr>
<tr>
<td>3000000377</td>
<td></td>
</tr>
<tr>
<td>3000000300</td>
<td></td>
</tr>
</tbody>
</table>
IKONAS MEMORY MAP (continued)

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2777741777</td>
<td>reserved for future use</td>
</tr>
<tr>
<td>20700000</td>
<td></td>
</tr>
<tr>
<td>2067741777</td>
<td>CGM4</td>
</tr>
<tr>
<td>20600002</td>
<td>reserved</td>
</tr>
<tr>
<td>20600001</td>
<td>address of CMCH</td>
</tr>
<tr>
<td>20600000</td>
<td>CGM4 Base Control Block</td>
</tr>
<tr>
<td>2057741777</td>
<td>SPS</td>
</tr>
<tr>
<td>20500000</td>
<td>writes as control register</td>
</tr>
<tr>
<td>2047741777</td>
<td>reads as current PC</td>
</tr>
<tr>
<td>20402000</td>
<td>MA1024</td>
</tr>
<tr>
<td>20402002</td>
<td>reserved</td>
</tr>
<tr>
<td>20402000</td>
<td>control registers</td>
</tr>
<tr>
<td>20401000</td>
<td>multiplier microcode memory</td>
</tr>
<tr>
<td>20400000</td>
<td>transformation matrix memory</td>
</tr>
<tr>
<td>2037741777</td>
<td>LUDC</td>
</tr>
<tr>
<td>20300001</td>
<td>reserved</td>
</tr>
<tr>
<td>20300000</td>
<td>channel crossbar control</td>
</tr>
<tr>
<td>20300000</td>
<td>colormap memory</td>
</tr>
<tr>
<td>2027741777</td>
<td>SRF</td>
</tr>
<tr>
<td>20200000</td>
<td>BK (1024) words per SRF - maximum of 8</td>
</tr>
<tr>
<td>2017741777</td>
<td>reserved</td>
</tr>
<tr>
<td>20100000</td>
<td></td>
</tr>
<tr>
<td>2007741777</td>
<td>MCM4</td>
</tr>
<tr>
<td>20000000</td>
<td>BK (1024) words per MCM4 - maximum of 8</td>
</tr>
<tr>
<td>1777741777</td>
<td>DR64</td>
</tr>
<tr>
<td>0000</td>
<td>framebuffer memory - up to 12 DR64s</td>
</tr>
<tr>
<td></td>
<td>each DR64 provides 312x512x8 pixels LORRES</td>
</tr>
<tr>
<td></td>
<td>1024x1024x2 pixels HIRRES</td>
</tr>
</tbody>
</table>
IKONAS BUS FUNCTION CODES

The 5-bit function code is present for each IKONAS bus operation. The function code tells:

- whether the operation is a read or a write
- how the address will be interpreted by the addressed module
- what operation will be performed by the addressed module

For example, if the address is 100:100, then:

If the function code is: 0 2 3 20 22 23 32
The operation will be: nothing — invalid address

LORES read from pixel (40, 40)
HIRES read from pixel (100, 100)
nothing — invalid address
LORES write to pixel (40, 40)
HIRES write to pixel (100, 100)
set write mask in all LORES cards
THE DR64 IMAGE MEMORY

Features:

- provides 512x512 pixels of 8 bits each or 1024x1024 pixels of 2 bits each
- configurable into images of up to 512x512x32 or 1024x1024x24
- software selectable between 512x512 and 1024x1024 display
- may be used as 64Kx32 bulk memory
- separate port for input at video rate
- allows each bitplane to be protected from modification
Configuring the DR64 Image Memory

A system is configured by specifying how each DR64 contributes its 8 bits at 512x512 or 2 bits at 1024x1024 to the pixel memory.

When a pixel is addressed, more than one DR64 will respond to the address. Each DR64 which is configured to contribute to the pixel will respond. All DR64s may respond to the same pixel address.

In 512x512 (LORES) display, three DR64s provide a 24-bit full-color image. If you have more than three DR64s, you can have more than one full 512x512 image stored.
Example of configuration for LORES

Cards 0, 1, and 2 all respond to LORES accesses in image 0.

Card 0 (IMAGE 0 RED)

Card 1 (IMAGE 0 GREEN)

Card 2 (IMAGE 0 BLUE)

Card 3 (IMAGE 1 RED)

Card 3 responds to LORES accesses only in image 1.

LORES pixel bits: 16-23 8-15 0-7
Example of configuration for HIRES

All cards respond to HIRES accesses

Card 0

Card 1

Card 2

Card 3

HIRES pixel bits 6-7 4-3 2-3 0-1
Addressing the DR64 in LORES mode

In 512x512 mode, pixels are addressed as \((x, y)\) coordinates. The IKONAS address is \(y\times x\). In LORES mode the \(y\) and \(x\) coordinates must be multiplied by 2. The IKONAS bus function code is 2 (read) or 22 (write).

The following diagram gives the \(y\times x\) addresses of pixels on the screen, as addressed and displayed in LORES mode.

<table>
<thead>
<tr>
<th>Top left pixel</th>
<th>DR64 PIXEL ADDRESSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 060 1 062 1 064 (\Rightarrow) 0617721 0617741 0617761</td>
<td></td>
</tr>
<tr>
<td>1 260 1 262 1 264 (\Rightarrow) 2617721 2617741 2617761</td>
<td></td>
</tr>
<tr>
<td>1 460 1 462 1 464 (\Rightarrow) 4617721 4617741 4617761</td>
<td></td>
</tr>
</tbody>
</table>
| \(\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdOTS B ottom ri ght pixel
Addressing the DR64 in HIRES mode

In 1024 x 1024 mode, pixels are addressed as \((x,y)\) coordinates. The IKONAS address is \(y \times x\). The IKONAS bus function code is 3 (read) or 23 (write).

The following diagram gives the \(y \times x\) addresses of pixels on the screen, as addressed and displayed in HIRES mode.

<table>
<thead>
<tr>
<th>Top left pixel</th>
<th>DR64 PIXEL ADDRESSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 080</td>
<td>1 081</td>
</tr>
<tr>
<td>1 180</td>
<td>1 181</td>
</tr>
<tr>
<td>1 280</td>
<td>1 281</td>
</tr>
<tr>
<td>1 380</td>
<td>1 381</td>
</tr>
</tbody>
</table>

The diagram shows the relationship between the pixel coordinates and their corresponding DR64 addresses in the HIRES mode.
Accessing the DR64 in WORD mode

In WORD mode each DR64 is a 64Kx32-bit bulk memory. The WORD mode addresses start at 1000$0 and go for as far as you have DR64 memory.

Therefore, the first DR64 has word mode addresses from 1000$0 to 1077$1777; the second DR64 has addresses 1100$0 to 1177$1777; and so on.

The IKONAS bus function code is 0 (read) or 20 (write).
Notes on different addressing modes

Remember, even though there are three different addressing modes for the DR64, only one set of data bits is stored. Addressing in the different modes only gives you different views of the same data.

In particular this means:

- when you store an image in HIRES you cannot meaningfully display it in LORES
- when you store an image in LORES you cannot meaningfully display it in HIRES
- when you store an image, you are overwriting any data you previously stored in those DR64s, even if you stored the previous data using a different addressing mode.

In the configuration example given above, storing into LORES image 0 will destroy HIRES bits 0-5; storing into the HIRES image will destroy all LORES bits for images 0 and 1; storing into card 0 in word mode will destroy the red component of image 0 (bits 0-7).
The **DR64 Write Mask**

The **write mask** allows you to protect bitplanes of the image memory from modification.

There is one bit in the write mask to correspond to each bit of a pixel. When a write is performed, only those bits which have 1s in the corresponding bit positions of the write mask will be modified. Bits which have 0s in the corresponding positions of the write mask will be unchanged.

The write mask is always active.

Whatever is in the write mask will always be used to mask any write to a DR64.

To set the write mask:

1. **use the IKONAS bus function code corresponding to the kind of writes you will be using:** 32 for LORES, 33 for HIRES.
2. **use the IKONAS address of any pixel in the image you want to set the write mask in.** For example, 0000 for image 0; 20000 for image 1; etc.
3. **write the data which will become the write mask.**

**Example (8 bits per pixel):**

| Old data | 11111111 |
| Write mask | 00001111 |
| Data written | 01100110 |
| Result data | 11110110 |

Because of the write mask, only bits 0-3 are modified; bits 4-7 are unchanged.
The GM64 Graphics Memory

The GM64 Graphics Memory Module is largely compatible with the DR64 Image Memory.

- Features compatible with the DR64
  - 512x512 8-bit pixels or 1024x1024 2-bit pixels per card
  - Programmably switched between 512x512 and 1024x1024 mode
  - HIRES and LORES read and write identical to DR64
  - WORD mode read identical to DR64
  - Video input port allows real-time image capture

- New features
  - Mask-mode writes: any or all of 32 pixels may be written simultaneously with a preset value
  - Enhanced video input port allows real-time generation of polygons
  - Z-buffer option: up to 24 bits may be used to store z; z-comparison and conditional write occur with no performance degradation.
Mask Mode Writes

Mask mode allows you to write up to 32 pixels with one operation. The operation of function code 20 (write word mode) is altered, and new function codes 24, 36, and 37 are added.

Mask mode consists of two steps:

1. Use a set shade function to store the data value to be used by subsequent mask-mode writes. Function codes to use are

   36 for LORES

   37 for HIRES

2. Use the mask-mode write function to write up to 32 pixels (HIRES) or 16 pixels (LORES). Function codes to use are

   24 for LORES. Bits 0-15 of the IKONAS bus data will be the mask.

   20 for HIRES. Bits 0-31 of the IKONAS bus data will be the mask.

Each of the 32 data bits (for LORES, the low 16 bits) of the mask-mode write data corresponds to one pixel in the GM64 memory. A one-bit in a pixel position causes that pixel to remain unchanged; a zero-bit in a pixel position causes the previously set shade to be stored in that pixel. All GM64 cards respond to the set-shade and mask-mode write operations.
The Sender Id

The sender id is a three-bit field, set in each IKONAS bus write, indicating the originator of the write. Each IKONAS module appends its sender id to each write operation. Most modules allow the sender id to be set under program control; for example, the BPS32, the IF/IK, and the MPC allow the user to specify the sender id. Other modules use a fixed sender id: the FBC always uses sender id 0.

The sender id is useful in cases where an operation is a two-step process -- for example, in mask-mode writes or use of the write mask. The second step of the process uses the data stored by the first step which had the same sender id as the second step.

In the GM64, the sender id is used by all write operations. Each sender id has its own write mask and mask-mode shade. A write operation uses the write mask and shade corresponding to the sender id for that write.

This means that you may have up to eight write masks and mask-mode shade values active at one time. Each process using the memory may be allocated its own write mask and shade value, which will not interfere with any other.
Example of Mask-mode Write - LORES

Step 1. Use function code 36 to set the shade.

Step 2. Issue the mask-mode write using function code 24. Note the setting of the top four address bits:

<table>
<thead>
<tr>
<th>address</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110000000110000110101</td>
<td>0000000000000000001110000110010</td>
</tr>
</tbody>
</table>

- bits 20-23 = 0110 for mask mode
- bits 15-19 = 0
- bits 6-14 = address (here, line 303 octal)
- bits 1-5 = top 5 bits of address. Pixels numbered 101010000 to 101011111 will correspond to the mask. These numbers are 320 to 337 octal.

bit 0 is unused.

Image memory

<table>
<thead>
<tr>
<th>columns</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0123456701</td>
<td>0123456701234567</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>01234567</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

line 0
line 302
line 303
line 304
line 776
line 777

these pixels correspond to the mask
x'd pixels have the preset shade stored in them
Example of Mask-mode Write - HIRES

Step 1. Use function code 37 to set the shade.
Step 2. Issue the mask-mode write using function code 20. Note the setting of the top four address bits:

<table>
<thead>
<tr>
<th>address</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4 0 1 4 0 3 3 2</td>
<td>0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0110000001100001101010</td>
<td>00000000000000001111000011001010</td>
</tr>
</tbody>
</table>

bits 20-23-- | I | I |
mask mode |
bits 15-19=0 | |
bits 5-14= address (here, line 607 octal) |
bits 0-4=top 3 bits of address. Pixels numbered 010100000 to 01011111 will correspond to the mask. These numbers are 500 to 537 octal.

Image memory

<table>
<thead>
<tr>
<th>columns</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>0123456701</td>
<td>0123456701</td>
</tr>
<tr>
<td>line 0</td>
<td></td>
</tr>
<tr>
<td>line 1</td>
<td></td>
</tr>
<tr>
<td>line 607</td>
<td></td>
</tr>
<tr>
<td>line 610</td>
<td></td>
</tr>
<tr>
<td>line 611</td>
<td></td>
</tr>
<tr>
<td>line 1776</td>
<td></td>
</tr>
<tr>
<td>line 1777</td>
<td></td>
</tr>
</tbody>
</table>
these pixels correspond to the mask
x's pixels have the preset shade stored in them
Z-buffer Option

When the Z-buffer option is installed, an additional memory function is added: Certain bitplanes are configured as z-planes by wire-wrap jumpers on the QM64. When a conditional write is performed for a given function, the value to be written into those bitplanes is compared against the value currently stored there.

If the old value is less than the new value, the write is suppressed and the location remains unchanged.

If the old value is greater than or equal to the new value, the write is performed as a normal memory write operation.

The conditional write operation takes no more time than a normal write.

The QM64 is wire-wrap jumpered to allow LORES conditional write or HIRES conditional write. Use the proper function code for the cards in your system:

26 for LORES conditional write

27 for HIRES conditional write.

Normal operation of the QM64 is unaltered by the Z-buffer option. HIRES and LORES normal writes (unconditional) are unchanged.
THE VIDEO DATA PATH

---

[Diagram of video data path]

---

IKONAS BUS

---

[Diagram of IKONAS bus]

---

external interfaces <-> 68000

---

IKONAS BUS

---
THE IKONAS VIDEO CHAIN

The video chain is the set of modules which process data at pixel rates. This consists of three modules:

- FBC - figures out what data to display, fetches it from the DR64, and starts it on its way through the video chain. Also inserts sync, blanking, cursor.

- XBS - rearranges bits of a pixel at pixel rates.

- LUVD - colormap performs a table lookup on each color of the pixel, and sends the result of the lookup to the digital-to-analog converters which produce standard video.
The Frame Buffer Controller

The Frame Buffer Controller (FBC) controls the format and content of the display. Items controlled by the FBC are:

- video sync rates — number of lines per field and time per line
- viewport — the size and position on the screen of the displayed data
- window — the position of the frame buffer data within the viewport
- zoom — the magnification factor for each pixel
- cursor — the shape and location of the 32x32-bit programmable cursor
- aspect ratio — the ratio of the width of the picture to the height
- automatic erase — automatically erases the screen after display
- and others.

The FBC produces correct video sync information to conform to the EIA RS-170A standard or the RS-343 standard, selectable programmably.
Definitions of Video Terms

Since the FBC produces a video signal, you must understand what the video signal looks like and how it can be controlled by the FBC.

The diagram on the next page shows the video signal as a sequence of lines, just as it appears on the color monitor. One such sequence, from the top of the monitor to the bottom, is called a field. Note that part of each line, and several lines at the top and bottom of the picture, are not displayed at all. This part of the picture is called the blanked portion of the picture. Blanked portions of the picture are denoted by / in the diagram.

Embedded in the blanked portion of the picture are reference pulses used by the monitor. The horizontal drive pulse occurs during each line. The vertical drive pulse occurs during the top of each field. The horizontal and vertical drive pulses are denoted by \ in the diagram.
The FBC controls the sync information in the video signal by controlling the number of lines per field, and the length of time for each line. This information is in the FBC register at 30000#4.
FBC Video Formatting - Viewport

The unblanked portion of the picture corresponds to the face of the color monitor (roughly), but the FBC allows you to display less than the full 1024 lines and columns, and allows you to position the displayed data anywhere on the monitor. Two control registers, at 30000$0$ and 30000$1$, control the start of viewport and length of viewport respectively.
The viewport registers, defined above, specify where on the color monitor the displayed picture will fall; the window controls what data from the DR64 framebuffer memory will be displayed in the viewport. The window is controlled by the register at 30000H.

In the diagram, let yw be the y window setting, xw the x window setting, yv the y viewport setting, and xv the x viewport setting. The diagram shows the line which will appear on the screen at each location. Note that as the viewport changes, the visible picture moves, but the actual scene does not appear to move. Instead, data at the moving edges is revealed or obscured.
Zoom

Zoom is a magnification of the picture by repeating pixels and lines. The effect of zoom is to make each pixel grow, thus making the image grow.

On the IKONAS, zoom in the x and y directions is independent.

You may zoom only in integral multiples; so the smallest zoom is a factor of 2. The largest zoom is a factor of 256.

Zoom is controlled by the register at 30000H3.
Cursor

The IKONAS cursor is a 32x32 programmable overlay. There are two steps to displaying a cursor:

- first, define the cursor by specifying the 32x32 matrix
- then specify the location of the 32x32 overlay on the screen

Once you define the cursor, you can move it around on the screen by changing the location register. You do not have to redefine the cursor unless you want the overlay pattern to change.
Defining the cursor

The cursor is a 32x32 matrix, for a total of 1024 bits. You define these 1024 bits by writing 256 4-bit numbers into the cursor definition table at 30000400-30000777.

The diagram below shows the correspondence of addresses and bits to pixels in the 32x32 cursor matrix.

The numbers 0, 1, 2, and 3 denote the bit position in the 4-bit word at the address specified. Each digit corresponds to one pixel of the cursor.
Positioning the Cursor

The cursor is positioned by setting the cursor position register, 30000h6, to the desired (x,y) coordinates. The cursor must be turned on in the flag register before it will be displayed.

When the cursor is active, the 32x32 cursor matrix appears as an overlay upon the framebuffer picture.

The cursor location register gives the location (x,y) of the top left point of the cursor. Pixels x to x+31 on lines y to y+31 then correspond to the 32x32 cursor matrix.

Any of these pixels which corresponds to a 1-bit in the 32x32 cursor matrix will be 'in the cursor'; any which corresponds to a 0 will 'outside the cursor'. Any pixel outside of the 32x32 overlay will also be outside the cursor.

Pixels in the cursor are denoted by a tag bit appended to them as they pass through the video chain. This bit is used by the LUVD to select the shade displayed at the pixel.
Example of cursor display

Cursor matrix:

```
100000000000000000000000000000000001
0100000000000000000000000000000010
00100000000000000000000000000000100
000100000000000000000000000000001000
0000100000000000000000000000000010000
etc.
0000100000000000000000000000000010000
000100000000000000000000000000001000
00100000000000000000000000000000100
01000000000000000000000000000000010
100000000000000000000000000000000001
```

Cursor location set to (100, 100)

Screen will be:

```
100*100
\ 
* * 
** *
* 
** *
* * 
137*137
```

The *s represent pixels in the cursor.
FBC Flag Bits and Mode Settings

The register at 30000$5$ contains flag bits which specify modes of operation of the FBC. This register is allocated as follows:

```
Bits 33222222221111111111110
10987654321098765432109876543210
_____________________________________________________________________
| pixel clock | IR|IJP| X|E| |MIC| |
```

Definitions of these bits are as follows:

C (bit 2) - ON to cause the cursor to be displayed, OFF to suppress the cursor.

H (bit 3) - ON for 1024x1024 (HIRES) display, OFF for 512x512 display.

E (bit 5) - ON for automatic erase, OFF for normal display. When automatic erase is in effect, the data in the DR64 framebuffer will be erased every field.

X (bit 6) - ON if external sync information (from the LUVDO) is to be used, OFF if video sync is to be generated by the FBC.

PG (bits 7-8) - colormap page. These bits are appended to the pixel data as two tag bits for processing in the video chain.

S (bit 9) - ON if RS-343 sync is to be generated, OFF if RS-170A sync is to be generated. This bit should be off for 512x512 30 Hz NTSC standard video, on otherwise.

R (bit 10) - ON for repeat field operation, OFF for interlaced.

Pixel clock (bits 16-22) - number of nanoseconds per single pixel in LLORES, or per two pixels in HIRES. This field may be used to change the aspect ratio.
## Summary of FBC Control Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000000</td>
<td>3322222222111111111111</td>
<td>y viewport start</td>
</tr>
<tr>
<td>3000001</td>
<td>1098765432109876543210</td>
<td>x viewport start</td>
</tr>
<tr>
<td>3000002</td>
<td>1098765432109876543210</td>
<td>y viewport size</td>
</tr>
<tr>
<td>3000003</td>
<td>1098765432109876543210</td>
<td>x viewport size</td>
</tr>
<tr>
<td>3000004</td>
<td>1098765432109876543210</td>
<td>y window location</td>
</tr>
<tr>
<td>3000005</td>
<td>1098765432109876543210</td>
<td>x window location</td>
</tr>
<tr>
<td>3000006</td>
<td>1098765432109876543210</td>
<td>y zoom</td>
</tr>
<tr>
<td>3000007</td>
<td>1098765432109876543210</td>
<td>x zoom</td>
</tr>
<tr>
<td>3000008</td>
<td>1098765432109876543210</td>
<td>number lines per frame</td>
</tr>
<tr>
<td>3000009</td>
<td>1098765432109876543210</td>
<td>time per line</td>
</tr>
<tr>
<td>300000a</td>
<td>1098765432109876543210</td>
<td>pixel clock</td>
</tr>
<tr>
<td>300000b</td>
<td>1098765432109876543210</td>
<td>R13:PG 1X1E: 1X1C</td>
</tr>
<tr>
<td>300000c</td>
<td>1098765432109876543210</td>
<td>y cursor position</td>
</tr>
<tr>
<td>300000d</td>
<td>1098765432109876543210</td>
<td>x cursor position</td>
</tr>
</tbody>
</table>

### Notes:

1. The two low-order bits of the x and y viewport start and size values are ignored, and 0 is used for them, so you can set the starting position and size in multiples of four lines. The size used is actually the size given +4, so the size ranges from 4 to 1024.

2. When the REPEAT FIELD bit is set in 3000005, the y viewport size should be multiplied by two.

3. In LORES display, the x window should be multiplied by four.

4. Refer to the FBC Programming Guide for an exact discussion of the relationship between y window and y viewport.

5. Zoom counts should be set to one less than the magnification desired. Zoom count of 0 = unmagnified display, 1 = zoom by factor of 2, etc.

6. The number of lines per frame is more precisely the number of half-lines per field plus one. This number should be even for interlaced operation, odd for repeat-field operation.

7. The pixel clock can range from 20 to 77. Operation at settings below 26 is not guaranteed by IKONAS.
The Video Data Path as it Leaves the FBC

The FBC starts data down the video chain. The pixel data as it leaves the FBC is as follows:

<table>
<thead>
<tr>
<th>Bit number</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pixel data bit 0</td>
</tr>
<tr>
<td>1</td>
<td>Pixel data bit 1</td>
</tr>
<tr>
<td>2</td>
<td>Pixel data bit 2</td>
</tr>
<tr>
<td>3</td>
<td>Pixel data bit 3</td>
</tr>
<tr>
<td>4</td>
<td>Pixel data bit 4</td>
</tr>
<tr>
<td>5</td>
<td>Pixel data bit 5</td>
</tr>
<tr>
<td>6</td>
<td>Pixel data bit 6</td>
</tr>
<tr>
<td>7</td>
<td>Pixel data bit 7</td>
</tr>
<tr>
<td>8</td>
<td>Pixel data bit 8</td>
</tr>
<tr>
<td>9</td>
<td>Pixel data bit 9</td>
</tr>
<tr>
<td>10</td>
<td>Pixel data bit 10</td>
</tr>
<tr>
<td>11</td>
<td>Pixel data bit 11</td>
</tr>
<tr>
<td>12</td>
<td>Pixel data bit 12</td>
</tr>
<tr>
<td>13</td>
<td>Pixel data bit 13</td>
</tr>
<tr>
<td>14</td>
<td>Pixel data bit 14</td>
</tr>
<tr>
<td>15</td>
<td>Pixel data bit 15</td>
</tr>
<tr>
<td>16</td>
<td>Pixel data bit 16</td>
</tr>
<tr>
<td>17</td>
<td>Pixel data bit 17</td>
</tr>
<tr>
<td>18</td>
<td>Pixel data bit 18</td>
</tr>
<tr>
<td>19</td>
<td>Pixel data bit 19</td>
</tr>
<tr>
<td>20</td>
<td>Pixel data bit 20</td>
</tr>
<tr>
<td>21</td>
<td>Pixel data bit 21</td>
</tr>
<tr>
<td>22</td>
<td>Pixel data bit 22</td>
</tr>
<tr>
<td>23</td>
<td>Pixel data bit 23</td>
</tr>
<tr>
<td>24</td>
<td>Pixel data bit 24</td>
</tr>
<tr>
<td>25</td>
<td>Pixel data bit 25</td>
</tr>
<tr>
<td>26</td>
<td>Pixel data bit 26</td>
</tr>
<tr>
<td>27</td>
<td>Pixel data bit 27</td>
</tr>
<tr>
<td>28</td>
<td>Pixel data bit 28</td>
</tr>
<tr>
<td>29</td>
<td>Pixel data bit 29</td>
</tr>
<tr>
<td>30</td>
<td>Pixel data bit 30</td>
</tr>
<tr>
<td>31</td>
<td>Pixel data bit 31</td>
</tr>
<tr>
<td>32</td>
<td>In-cursor bit</td>
</tr>
<tr>
<td>33</td>
<td>Colormap page bit 0</td>
</tr>
<tr>
<td>34</td>
<td>Colormap page bit 1</td>
</tr>
</tbody>
</table>
THE IKONAS XBS34 VIDEO SWITCH

---

**IKONAS BUS**

---

external interfaces<->: HPC  
and: COMPUTER  
peripherals: MODULE  
256K RAM

---

ILCOM

---

COLOR  
MONITOR

---

INPUT  
MEMORY  
MODULE

---

DR64  
IMAGE  
CONTROL

---

FBC  
VIDEO  
/VIDEO/

---

/VIDEO+/  
/CROSS-/  
/SWITCH/  
/OUTPUT

---

BPS  
GRAPHICS  
PROCESS

---

MCH4  
PROGRAM  
MEMORY

---

SRB  
FAST  
MEMORY  
/ROTATION

---

MA1024  
HOST  
INTERFACE

---

IF/II

---

->to host
XBS34 VIDEO CROSSBAR SWITCH

- allows arbitrary rearrangement of pixel data bits
- allows bitplanes to be suppressed or repeated
- processes at video pixel rates
- switching path can be quickly modified by software for frame-to-frame changes
- handles 35 inputs and produces 34 outputs
The XBS34 accepts up to 35 bits from the previous stage of the video chain and produces 34 bits of output. Each output bit may be selected from any of the 32 input pixel data bits or the constant zero, according to the setting in a register; in addition, the four most-significant outputs may be selected from any of the 35 input bits (which include the in-cursor bit and the colormap page bits).

To set up the XBS34, you specify, for each output bit, the number of the input bits is to appear at that output bit. A number of 77 is used to specify that zero is to appear at the output bit.

For each output bit $x$, the number controlling that output is stored at 30200$\times x$. 
**XBS34 Set Up in Transparent Mode (no change of pixel data)**

<table>
<thead>
<tr>
<th>Input bit selected</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel data 0</td>
<td>&gt;0</td>
</tr>
<tr>
<td>Pixel data 1</td>
<td>&gt;1</td>
</tr>
<tr>
<td>Pixel data 2</td>
<td>&gt;2</td>
</tr>
<tr>
<td>Pixel data 3</td>
<td>&gt;3</td>
</tr>
<tr>
<td>Pixel data 4</td>
<td>&gt;4</td>
</tr>
<tr>
<td>Pixel data 5</td>
<td>&gt;5</td>
</tr>
<tr>
<td>Pixel data 6</td>
<td>&gt;6</td>
</tr>
<tr>
<td>Pixel data 7</td>
<td>&gt;7</td>
</tr>
<tr>
<td>Pixel data 8</td>
<td>&gt;8</td>
</tr>
<tr>
<td>Pixel data 9</td>
<td>&gt;9</td>
</tr>
<tr>
<td>Pixel data 10</td>
<td>&gt;10</td>
</tr>
<tr>
<td>Pixel data 11</td>
<td>&gt;11</td>
</tr>
<tr>
<td>Pixel data 12</td>
<td>&gt;12</td>
</tr>
<tr>
<td>Pixel data 13</td>
<td>&gt;13</td>
</tr>
<tr>
<td>Pixel data 14</td>
<td>&gt;14</td>
</tr>
<tr>
<td>Pixel data 15</td>
<td>&gt;15</td>
</tr>
<tr>
<td>Pixel data 16</td>
<td>&gt;16</td>
</tr>
<tr>
<td>Pixel data 17</td>
<td>&gt;17</td>
</tr>
<tr>
<td>Pixel data 18</td>
<td>&gt;18</td>
</tr>
<tr>
<td>Pixel data 19</td>
<td>&gt;19</td>
</tr>
<tr>
<td>Pixel data 20</td>
<td>&gt;20</td>
</tr>
<tr>
<td>Pixel data 21</td>
<td>&gt;21</td>
</tr>
<tr>
<td>Pixel data 22</td>
<td>&gt;22</td>
</tr>
<tr>
<td>Pixel data 23</td>
<td>&gt;23</td>
</tr>
<tr>
<td>Pixel data 24</td>
<td>&gt;24</td>
</tr>
<tr>
<td>Pixel data 25</td>
<td>&gt;25</td>
</tr>
<tr>
<td>Pixel data 26</td>
<td>&gt;26</td>
</tr>
<tr>
<td>Pixel data 27</td>
<td>&gt;27</td>
</tr>
<tr>
<td>Pixel data 28</td>
<td>&gt;28</td>
</tr>
<tr>
<td>Pixel data 29</td>
<td>&gt;29</td>
</tr>
<tr>
<td>Pixel data 30</td>
<td>&gt;30</td>
</tr>
<tr>
<td>Pixel data 31</td>
<td>&gt;31</td>
</tr>
<tr>
<td>In-cursor bit</td>
<td>&gt;1c</td>
</tr>
<tr>
<td>colormap #0</td>
<td>&gt;p0</td>
</tr>
</tbody>
</table>
XBS34 Set Up to Transfer Red Channel Input to All Three Outputs

Input bit selected:

Pixel data 0 |

Pixel data 1 |

Pixel data 2 |

Pixel data 3 |

Pixel data 4 |

Pixel data 5 |

Pixel data 6 |

Pixel data 7 |

Pixel data 8 |

Pixel data 9 |

Pixel data 10 |

Pixel data 11 |

Pixel data 12 |

Pixel data 13 |

Pixel data 14 |

Pixel data 15 |

Pixel data 16 |

Pixel data 17 |

Pixel data 18 |

Pixel data 19 |

Pixel data 20 |

Pixel data 21 |

Pixel data 22 |

Pixel data 23 |

Pixel data 24 |

Pixel data 25 |

Pixel data 26 |

Pixel data 27 |

Pixel data 28 |

Pixel data 29 |

Pixel data 30 |

Pixel data 31 |

In-cursor bit: | 1

colormap pg 0: | 1

colormap pg 1: | 1

Output cursor bit: | 1

Output colormap: | 1

Output 0: | 1

Output 1: | 1

Output 2: | 1

Output 3: | 1

Output 4: | 1

Output 5: | 1

Output 6: | 1

Output 7: | 1

Output 8: | 1

Output 9: | 1

Output 10: | 1

Output 11: | 1

Output 12: | 1

Output 13: | 1

Output 14: | 1

Output 15: | 1

Output 16: | 1

Output 17: | 1

Output 18: | 1

Output 19: | 1

Output 20: | 1

Output 21: | 1

Output 22: | 1

Output 23: | 1

177: zero
THE LUVQ COLORMAP AND VIDEO OUTPUT MODULE

<table>
<thead>
<tr>
<th>CAMERA</th>
<th>COLOR</th>
<th>MONITOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIB</td>
<td>DR64</td>
<td>FBC</td>
</tr>
<tr>
<td>VIDEO</td>
<td>IMAGE</td>
<td>VIDEO</td>
</tr>
<tr>
<td>INPUT</td>
<td>MEMORY</td>
<td>CONTROL</td>
</tr>
<tr>
<td>MODULE</td>
<td></td>
<td>MODULE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SWITCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IKONAS BUS

<table>
<thead>
<tr>
<th>BPS</th>
<th>MCM4</th>
<th>SR8</th>
<th>MA1024</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FAST</td>
<td>HOST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEMORY</td>
<td>3D &amp; 2D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MODULE</td>
<td>ROTATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>INTERFACE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

External interfaces (---) and peripherals:

- MPC
- 68000
- COMPUTER
- 1286A RAM

---to host

---to host
The LUVQ Colormap and Video Output Module

- performs a color lookup on all three video channels, ten bits each
- full 30-bit output at video rates
- contains three independent lookup tables, one for each video channel
- fast access from IKONAS bus
- channel crossbar switch supports full-color images and pseudocolor
- NTSC standard video output and sync output
- accepts externally generated sync
- options to allow overlay bitplanes
- data-dependent cursor — allows solid, transparent, complementing, or more complex cursor functions
Colormaps

A colormap is a device which converts an input pixel value into an intensity to be used for one component (red, green, or blue) of the video output.

The LUVO contains three independent colormaps, one for each video output channel (red, green, and blue).

A colormap treats each pixel independently. Whenever a particular value appears at the input to the colormap, the output value is always the same.

The value produced by a colormap for each input value can be set by the programmer.

Colormaps have two main classes of use:

1. When the input pixels are in full color, with a red, green, and blue component, each component can be passed through a colormap which compensates for the characteristics of the display monitor and the observer’s eye. This is called color correction.

2. When the input pixels are eight bits or fewer, each value of the input pixels may correspond to a different color. The same pixel value is passed to the three different colormaps, one for each output component (red, green, and blue); the resulting mixture of components gives the desired color for that input pixel value. This is called pseudocolor.
The colormap used for color correction

Since the colormap implements a function of the inputs, color correction can be shown in that way:

The pixel value for a particular channel consists of eight bits of pixel data, the in-cursor bit, and colormap page bit 0, as follows:

(colormap page bit 0, C=in-cursor bit, both inserted by the FBC)

\begin{align*}
\text{red channel:} & \quad \text{PIC: data bits 0-7} \\
\text{green channel:} & \quad \text{PIC: data bits 8-15} \\
\text{blue channel:} & \quad \text{PIC: data bits 16-23}
\end{align*}
Colormaps used for pseudocolor

Colormaps used for pseudocolor resemble a translation table memory more than a mathematical function. The same pixel value is applied as the address of each memory, and the values stored at that address are used for the components of the color displayed.

In the example following, the colormap has been loaded with a palette of random colors.

Input
pixel value

0

addr
v

v

0

1

2

3

etc. /

-RED TABLE

GREEN TABLE

BLUE TABLE

<table>
<thead>
<tr>
<th>0</th>
<th>.5</th>
<th>.5</th>
</tr>
</thead>
</table>
RED GREEN BLUE

Output color (cyan)
In pseudocolor, the pixel value is taken from some one of the red, green, or blue inputs, and the same value is sent to all three colormaps. The pixel value will be:

(P=colormap page bit 0, C=cursor bit, both inserted by the FBC)

```
bit 9 8 7 6 5 4 3 2 1 0

|P|C| data bits *** |
```

where *** is 0-7 for the red channel input, 8-15 for the green channel input, or 16-23 for the blue.
Colormap Pages and the Cursor

As can be seen from the format of the inputs to the colormaps, the colormap page bit and the in-cursor bit from the FBC become part of the pixel value applied to each colormap.

The in-cursor bit is applied as bit 8 of each colormap input. This bit is 1 for each pixel defined to be in the programmable cursor, and is 0 for all other pixels. In effect, you have two separate colormaps, one for pixels in the cursor and one for pixels not in the cursor. You may use this feature to provide a cursor whose value depends on the pixel value; for example, a complementing cursor, or a 'transparent' cursor which merely brightens the pixels.

The colormap page bit (bit 7 of 30000$5$) is applied as bit 9 of each colormap. This gives you the effect of having two colormaps, one with the page bit on and the other with the page bit off. You may switch between the colormaps merely by changing the value of the page bit in the FBC.
The channel switch

The channel switch on the LUVO allows you to select either full-color or pseudocolor operation. The channel switch is similar to the XBS34 crossbar switch, but you route entire video channels instead of single bits.

The channel switch has one output channel for each of the three colormaps; each output channel may be selected from any of the three input channels.
LUVD data routing - Full-color Operation

LUVD data routine - Full-color Operation

Data from FBC or XBS Channel switch Color Lookup

Digital-to-
Analog
Conversion

red pixel data in-cursor bit colormap page

RED

COLOMAP

DAC

green pixel data in-cursor bit colormap page

GREEN

COLOMAP

DAC

blue pixel data in-cursor bit colormap page

BLUE

COLOMAP

DAC

IKONAB bus address bits 0-9

0-9 10-19 20-29

IKONAB bus data bits
**LUVO data routing - Pseudocolor Operation**

(Example shows pseudocolor from the green channel)
Setting up the colormaps

The colormaps are initialized by writing for each input pixel value the output which the colormaps are to provide when that input is applied.

When a value is written to the LUVO, one entry in each of the three colormaps is modified with a single IKONAS bus write. When a value is read from the LUVO, one entry from each of the three colormaps is read with a single read.

The address used in the IKONAS operation is $20300vvvv$ where $vvvv$ is the input pixel value whose entries you want to read or modify.

Of the 32 IKONAS bus data bits, bits 0-9 correspond to the RED colormap; bits 10-19 correspond to the GREEN colormap; bits 20-29 correspond to the BLUE colormap; and bits 30-31 are reserved for future use.

Example:

The address is $20300$50, the IKONAS bus data is $4402\backslash 57720$. The entries in the colormaps corresponding to input pixel value 50 are:

\[
4402\backslash 57720 = \begin{array}{cccc}
\text{blue} & \text{green} & \text{red} \\
220 & 227 & 1720 \\
\end{array}
\]

The entries in the colormaps range from 0 (dark) to 1777 (bright).
LUVO data routing = IKONAS bus access

Data from
FBC or KSB
\[\text{Channel switch} \rightarrow \text{Color Lookup} \\]

Digital-to-
Analog
Conversion
\[\text{DAC} \rightarrow \]

red pixel data
in-cursor bit
colormap page

\[\text{RED} \rightarrow \text{COLORMAP} \rightarrow \text{DAC} \rightarrow \]

green pixel data
in-cursor bit
colormap page

\[\text{GREEN} \rightarrow \text{COLORMAP} \rightarrow \text{DAC} \rightarrow \]

blue pixel data
in-cursor bit
colormap page

\[\text{BLUE} \rightarrow \text{COLORMAP} \rightarrow \text{DAC} \rightarrow \]

IKONAS bus
address bits
0-9

IKONAS bus data bits
0-9 10-19 20-29
THE IKONAS 8PS32 GRAPHICS PROCESSOR
The BPS32 Graphics Processor

- 32-bit processor for high precision
- Can simultaneously perform two 16-bit operations for high speed
- 5 million instructions per second
- Writable control store - fully user-programmable
- Assembler is provided, written in standard FORTRAN
- Control store can be used as scratch memory
- Powerful instruction set allows overlapped arithmetic, branches, and I/O
- Fast multiply option allows 5 million multiplies per second
Microcode Development Tools

Microcode development tools consist of the BPS32 assembler (IKASM), the object-file loader (IKLOD), and the debugging tool (IKDEB).

All these programs run on the user's host computer.

IKASM converts a symbolic assembler-language file into an object file containing BPS32 machine code.

IKLOD is a subroutine which loads an IKASM object file into the IKONAS address space.

IKDEB allows you to start and start the processor, load files, and examine locations in the IKONAS address space.
IKASM Assembler Language Syntax

The IKASM assembler has the following syntax rules:

1. each statement corresponds to one BPS32 microcode instruction
2. one instruction per line
3. one line per instruction unless continued by slash in column 1 of next line
4. symbols are separated by spaces
5. symbols may be up to eight characters long
6. the first symbol may be followed by a colon, denoting a label; the value of the current location counter will be assigned to the label
7. fields may appear in any order
8. all characters after a semicolon are comments and are ignored by the assembler
The IKLOD subroutine

The object file output of IKASM is an unformatted FORTRAN file described in the IKONAS Loader Format and BPS32 Host Programming Guide documents.

To load an object file into the IKONAS, your FORTRAN program should do the following:

1. open the file for input

2. call the IKLOD subroutine, passing it the unit number of the file. On return your program should close the unit.

Example: CALL IKLOD(IUNIT)
IKASM Statements

IKASM statements fall into two categories:

1. Pseudo-operations: directives to the assembler which do not generate BPS32 instructions
2. Instructions: any line representing a BPS32 instruction.

The syntax of instructions and pseudo-operations is identical. Pseudo-operations are distinguished by the first symbol in the statement.

Pseudo-operations are:

1. assignment - explicitly assign a value to a symbol
2. ORG - set location counter
3. DEFAULT - establish default settings for fields
4. END - end the IKASM run
Instructions

Any IKASM statement which is not a pseudo-operation represents a BPS32 instruction.

In the hardware, BPS32 instruction words are composed of fields. Since the BPS32 is a microprogrammed machine, each field corresponds to some section of the hardware: a data path, a register, or a control signal.

An IKASM instruction is a specification of all the settings of all the fields.

Each field in the hardware microinstruction corresponds to a set of keywords in IKASM, one keyword for each possible setting of the field. For example, the hardware component called the 'R selector' can be loaded from the MDR or the MAR; the keywords 'RMAR' and 'RMDR' correspond to the appropriate settings of the 'R selector' field of the microinstruction.

In IKASM, you code the keywords which indicate the desired settings of the microinstruction fields. Each keyword specifies simultaneously the field it refers to and the value for that field. You must avoid conflicts: two keywords which specify different value for the same field.

Symbols may be defined in your program, by the assignment pseudo-operation and by coding the symbol as the label of a statement. Symbols other than keywords may be used in instructions; they will be assembled in the data field of the instruction. The data field is a 16-bit field which holds immediate data and branch addresses.

If you code a symbol which is not defined (has not been defined in your program and is not a keyword), an error message is printed.
Examples of statements

LABEL: RA1 RPS B2 BD JMPDF FRED ; r2 <- r1+r2

The keywords are: RA1 - R operand from register 1; RPS - add R operand to S operand; B2 - S operand from register 2; BD - write result to register 2; JMPDF - branch to the value in the data field.

The symbol FRED, which must be defined elsewhere in the program, will be assembled into the data field as the branch target.

The symbol LABEL will be assigned a value equal to the location of this instruction. This label can be the target of branch instructions.

RA1 RA2 PR ALUMDR ; MDR <- rxx

This statement contains a conflict: the R operand is specified as register 1 and register 2.
The Assignment Pseudo-operation

Example:

GBASE = 10 ; define offset to base of queue

The assignment statement defines the first symbol, and gives it a value equal to the value of the symbol following the equals sign.

The symbol thus defined may be used as a data field in IKASM instructions.

A symbol need not be defined before it is referenced.
The DEFAULT Pseudo-operation

DEFAULT NANOP CCNOP LDNOP PR YD CARO SSO ALUBR SB MDRIKD MARIKA

The DEFAULT statement gives the default setting for fields in the BPS32 instruction.

The DEFAULT statement should be the first statement in a program.

When an IKASM instruction is assembled, all keywords appearing in the instruction will set the corresponding fields in the microinstruction; fields which are not set by a keyword in the instruction will assume the setting of the DEFAULT statement.

The DEFAULT statement saves a lot of typing, since you don't have to specify fields whose values equal the default.
The ORG Pseudo-operation

ORG value
ORG 256
ORG PGMSTRT

The ORG pseudo-instruction specifies a new value for the location counter. The location counter is automatically incremented by every IKASM instruction; the ORG statement may be used to skip a block of memory or start a program at a desired address.
The END Pseudo-operation

END

The END pseudo-operation is the last statement in an IKASM program.
BPS32 Organization

The BPS32 hardware consists of two components: the processor section and the sequencer section.

The processor section executes instructions, performs arithmetic and I/O, and saves results.

The sequencer section figures out what instructions to execute. During each instruction it computes the address of the next instruction.
The processor section contains the following components:

1. 17 32-bit registers
2. a 32-bit ALU including shifter
3. a 32-bit Memory Data Register (MDR), to hold data to be written to the IKONAS bus
4. a 32-bit Memory Address Register (MAR), to hold the IKONAS bus address for IKONAS bus operations
5. (optional) a 16x16-bit two’s complement multiplier

The connections within the processor section are illustrated in the diagram on the next page.
The sequencer section consists of the following components:

1. a 16-bit program counter to hold the address of the next instruction
2. a 16-bit register to hold the data field of the previous instruction
3. a four-word by 16-bit stack to hold subroutine and loop addresses
4. a 16-bit loop counter
5. condition-code-testing logic

During each instruction, the microcode word and condition codes are examined to determine the address of the next instruction, which will be one of the following:

1. the current program counter plus one
2. the value in the data field of the current instruction (branch instruction)
3. the value at the top of the subroutine stack (subroutine return)
4. the value in the data field of the previous instruction (two-way branch)
5. the value on the IKONAS bus during the previous instruction (indirect branch)

Based on the microcode word, the sequencer may decrement the loop counter and set a flag if the count becomes zero.
The Sequencer Subroutine Stack

The subroutine stack in the sequencer has room to hold four addresses, to allow for subroutines nested up to four deep. If subroutines are nested more than four deep, the results are unpredictable.

A subroutine call consists of pushing the address of the current instruction plus one onto the stack (this will be the return address, the address of the instruction following the subroutine call), and branching to the subroutine. A return from subroutine consists of branching to the address contained in the top of the stack, and popping the stack. If the stack is empty when a return from subroutine is executed, the results are unpredictable.

The subroutine stack may be used to mark the top of loops by pushing the address of the top of the loop onto the stack. Special instructions may be used to branch to the address on the top of the stack without removing it from the stack.

You must be careful to remove any addresses pushed onto the stack, since no address except the one on the top of the stack can be referenced by the sequencer.
BPS32 QUICK REFERENCE CARD

LOAD CONTROL

LONGP DF not loaded to any other register
LDCON control reg = DF
LDCT CTR = DF
LDDR UDR = 0
Y BUS CONTROL src & dst
ALUDY ALUMMAR
ALUMPxy ALUMPxy
IMPlxy IMPlxy
* wait 1 cycle for completion
IMMDR IMMDR
MPZy MPZy
MARIA addr = MARO-23
MASHY addr = MARO-29MARIO-9
IKRAS BUS FUNCTIONS
IKRD word mode read
IKWR word mode write
HIERSD HIRES read
HLESRD LORES read
HIERSW HIRES write
HLESGR LORES write
HIIASKW HIRES set write mask
LIIASKW LORES set write mask
HISHADR HIRES set shade
LISHADR LORES set shade
USRSTMA write & start MA1024
CONTMA restart MA1024
STOPMA stop MA1024

GLOSSARY

ALU shifter output
BR B register
CTR loop counter
DF data field
I program counter
IB IKONAS bus
MPX M PLA X input
MPY M PLA Y input
MPZ M PLA Z output
Q 0 register
STK stack
UDR upper data register
 operand one's complement
operand two's complement
AND
OR
EXCLUSIVE OR

COND CODE SELECT

Preset M to negate condition
CCONP always true
CCNTZ CLR = 0 after JMPCNT or ULPNT
CCBLK vertical drive pulse
CCMAAC R1 = 024 busy
CCHOST host request
CCDOV ALU overflow
CCNEG ALU negative
CCZERO Y bus zero
CCCAR carry from ALU1
CCY16 Y-bus 16 = 1
CCGT2 ALU >= 0 & Y bus = 0 =
* invalid during read from IB or MPLA

SPECIAL FUNCTIONS

INCPS BR = 0 - 1
SMTC convert BR sign-magnitude to two's complement
SLNML single length normalize
RLNML double length normalize
ULMPL unsigned multiply
TCLMPL signed multiply
DMLPL end of signed multiply
TCDIV divide
TCDIV end of divide

UNION SHIFTS

MULTIPLY - Rx = Ry
Product = RxY
ALUZ BR BD
RA PR QD LDCNT 1-32
UMPY RA R1 LSO JMPCNT NCCNTZ
THS'S COMPLEMENT MULTIPLY - RX + RY
RMS LSO JMPCNT NCCNTZ
THS'S COMPLEMENT DIVISION
Ry\\/Q = dividend (64 bits)
R = divisor (32 bits)
Q = quotient, n bits
Ry = remainder
(n = degree of precision wanted)

HIAR'S COMPLEMENT

DLNML RA By LR LDCNT -1:n
TCDIV RA By LR CARZ JMPCNT NCCNTZ
TCMLPL RA By CARZ LSO
THS'S COMPLEMENT DIVISION

ARITHMETIC COMPARISONS

Test Fmt CC (32 bits/16 bits)
R = S RMS NCCNEG/NCCY16
R = S SMR CCNEG/CCY16
R = S SMR NCCNEG/NCCY16
R = S RMS CCNEG/CCY16
R = S SMR CCNEG
R = S RMS CCNEG

LOGICAL COMPARISONS

R = S RMS CCCAR
R = S SMR NCCCAR
R = S SMR CCCCAR
R = S RMS NCCCAR
R = S RMS CCCCAR

SAMPLE DEFAULT STATEMENT

DEFAULT NANOP CCONP LONPO PR YD CARO SSO ALUAYD SB MDRMAD MARIAK
Operation: Move register 2 to register 3
Instruction: RA2 PR B3 BD
Operation: register 3 ← register 3 + 200
Instruction: RLIMM 200 RPS B3 BD
Operation: Write MDR to the IKONAS bus at addr in MAR; subtract R4 from MAR.

Instruction: IKWR RMAR RMS R4 CAR1
Operation: Increment R5 by 2
Instruction: B5 INCRS CAR1

MAR BUS

A addr→ 16 32-bit registers <- B addr

R OPERAND SELECTOR

S OPERAND SELECTOR

i carry in→ ALU

32-bit shifter

data field

MDR MAR MPLIER

UPPER DATA REGISTER

IKOMA data IKOMA address
Operation: Read IKONAS bus into MDR; load G from R5
(second half of dynamic memory read)
Instruction: IKMDR RA5 PR QD
The MA1024 3-D Transformation Unit

- Performs 3-D rotation of vector endpoints and polygon vertices
- Fast - 300,000 transformations per second
- Allows storage of 64 rotation angles simultaneously
- Option allows rotation with perspective
- Calculates dot-product for lighting simulation
- Calculates cross-product of vectors for normal-vector computation
- Fully user-programmable with writable control store
- Full parallel processing - does not require access to IKONAS bus during operation
- Uses up to seven SR8 dual-port memories for data input and output
MA1024 Inputs and Outputs

- Input points can be 3-D [x y z] or homogeneous [x y z w]
- Output is same format as input
- 4x4 coefficient matrix controls rotation

The MA1024 accepts a list of input points, matrix multiplies them by a matrix representing the desired rotation (the coefficient matrix), and stores the resulting output points.

The MA1024 input and output points, which are stored in the SR8 memory, represent points in three-space or points in homogeneous coordinates. A point in three-space is a vector [x y z]; a point in homogeneous coordinates is a vector [x y z w].

The MA1024 always uses homogeneous coordinates in its multiplication, so a point in three-space is extended by setting w=1: [x y z 1].

The output points produced by the MA1024 are the same format as the inputs: either points in three-space or points in homogeneous coordinates. Input points in three-space carry with them a tag field which is copied unchanged to the output.

The examples below will all use input points in three-space.
The Rotation Operation in the MA1024

Each input point to the MA1024 is transformed to produce a corresponding output point by a matrix multiplication defined as follows:

\[
\begin{bmatrix}
C_{xx} & C_{xy} & C_{xz} & 0 \\
C_{yx} & C_{yy} & C_{yz} & 0 \\
C_{zx} & C_{zy} & C_{zz} & 0 \\
T_{x} & T_{y} & T_{z} & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
X_{o} \\
Y_{o} \\
Z_{o}
\end{bmatrix}
\]

is the output point;

\[
\begin{bmatrix}
X_{i} \\
Y_{i} \\
Z_{i}
\end{bmatrix}
\]

is the input point;

\(C_{ij}\) is the contribution of input coordinate \(i\) to output coordinate \(j\);

\(T_k\) is the amount of translation (lateral movement) in coordinate \(k\).

Data Formats

All numbers in the MA1024 are 16-bit two's-complement fractions ranging in value from -1.0 to +0.9999. The most-significant bit of such a number (bit 15) is the sign bit; bits 0-14 are the fractional part. The implied binary point lies between bits 14 and 15.

An [x y z] point or an [x y z w] point requires four 16-bit numbers, or two 32-bit IKONAS data words. The two words for a point occupy consecutive locations in the SRB memory, as follows:

For [x y z] points:

<table>
<thead>
<tr>
<th>Bits</th>
<th>3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 1</th>
<th>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>first word</td>
<td>y coordinate</td>
<td>z coordinate</td>
</tr>
<tr>
<td>second word</td>
<td>E</td>
<td>tag field</td>
</tr>
</tbody>
</table>

E (bit 30) is the end-of-list bit, set in the last point to be rotated.

For [x y z w] points:

<table>
<thead>
<tr>
<th>Bits</th>
<th>3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1</th>
<th>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>first word</td>
<td>y coordinate</td>
<td>z coordinate</td>
</tr>
<tr>
<td>second word</td>
<td>w coordinate</td>
<td>z coordinate</td>
</tr>
</tbody>
</table>
Coefficient Matrix Format

The coefficient matrix to be used for a rotation is stored in the coefficient memory of the MA1024.

The MA1024 can hold up to 64 coefficient matrices simultaneously, though only one may be used for a single list of points.

Each coefficient matrix occupies 16 consecutive words of the coefficient memory, as follows (refer to the diagram above for the meaning of $C_{xx}$ etc.):

Word 0  $C_{xx}$
Word 1  $C_{xy}$
Word 2  $C_{xx}$
Word 3  $T_x$
Word 4  $C_{yx}$
Word 5  $C_{yy}$
Word 6  $C_{yz}$
Word 7  $T_y$
Word 8  $C_{zx}$
Word 9  $C_{zy}$
Word 10 $C_{zz}$
Word 11 $T_z$
Word 12 0
Word 13 0
Word 14 0 (used by the perspective division option)
Word 15 .9999
MA1024 Operating Sequence

The steps to use of the MA1024 are the following:

1. Load the correct MA1024 microcode routine into the program memory. Microcode routines to perform operations described in this document are supplied by IKONAS and may be loaded using the IKLOD subroutine. The routine to perform rotation is named MMEOL.

2. Load the coefficient matrix for the rotation into the coefficient memory.

3. Load the points to be transformed into the SRB. Be sure E (bit 30) is set in the last word.

4. Load the correct values into the MA1024 control registers for the following:
   1. MA1024 program address
   2. Coefficient matrix offset
   3. Input data address
   4. Output data address

See below for the formats of these control registers.

5. Start the MA1024 by storing 0 into address 20402#3 using a function code of 25.

Note

The procedure described above is the proper sequence for using the MA1024 to perform rotation. It does not use all the features of the MA1024. If you want to write custom MA1024 microcode, you should learn about the full power of the MA1024 by reading the MA1024 Programming Guide.
The registers described below control the operation of the MA1024. All the registers may be accessed from the IKONAS bus in the address range 20402#0-20402#2.

<table>
<thead>
<tr>
<th>Bits</th>
<th>3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2</th>
<th>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>20402#0</td>
<td>coefficient offset</td>
<td>W</td>
</tr>
<tr>
<td>20402#1</td>
<td>output list offset in SRB</td>
<td>input list offset in SRB</td>
</tr>
<tr>
<td>20402#2</td>
<td></td>
<td>loop O counter value</td>
</tr>
</tbody>
</table>

**Program address** is the offset to the MA1024 microprogram to be used. For the IKONAS rotation program MMEOL, this value will be 0. For custom microcode, the value will be the same as the value of the ORG statement in the IKASH assembly of the microcode.

**W** (bit 13) should be set to 0 for MMEOL. When this bit is 1, the MA1024 will expect input points in homogeneous coordinates: \([x \ y \ z \ w]\). IKONAS does not supply microcode which requires \(W\) set to 1.

**Coefficient address** is the offset to the coefficient matrix to be used. If this offset is specified as cccc, the coefficient matrix must start at 20400#cccc.

**Input and output offsets** are the offsets to the input and output areas in the SRB. If one of these offsets is specified as xxxx, the area at 20200#xxxx will be used for input or output. The last point in the input list must have the end-of-list (E) bit set.

**Loop O counter value** is a value which may be used by user microcode. IKONAS microcode does not use the loop O counter, which may be left uninitialized.
Diagram of MA1024 Control Flow

Coeficient Memory

Control Registers

Program Memory

SRS MEMORY

INPUT LIST

OUTPUT LIST
THE MA1024 3-D TRANSFORMATION UNIT WITH CLIPPING SUPPORT

Diagram showing the connections and components of the MA1024 3D transformation unit with clipping support.
The MA1024 3-D Transformation Unit with Clipping Support

- Performs 3-D rotation of vector endpoints and polygon vertices
- Fast - 300,000 transformations per second
- Allows storage of 64 rotation angles simultaneously
- Allows rotation with perspective
- High-precision perspective divider allows perspective rendering of complex scenes
- Clipping assist feature detects and flags points out of view in six clipping planes - allows savings of 50-80 percent of time spent in clipping
- Calculates dot-product for lighting simulation
- Calculates cross-product of vectors for normal-vector computation
- Fully user-programmable with writable control store
- Full parallel processing - does not require access to IKONAS bus during operation
- Uses up to seven SR8 dual-port memories for data input and output
MA1024 Inputs and Outputs

- Input points can be 3-D \([x \ y \ z]\) or homogeneous \([x \ y \ z \ w]\)
- Output is same format as input
- 4x4 coefficient matrix controls rotation

The MA1024 accepts a list of input points, matrix multiplies them by a matrix representing the desired rotation (the coefficient matrix), and stores the resulting output points.

The MA1024 input and output points, which are stored in the SRB memory, represent points in three-space or points in homogeneous coordinates. A point in three-space is a vector \([x \ y \ z]\); a point in homogeneous coordinates is a vector \([x \ y \ z \ w]\).

The MA1024 always uses homogeneous coordinates in its multiplication, so a point in three-space is extended by setting \(w=1\): \([x \ y \ z \ 1]\).

The output points produced by the MA1024 are the same format as the inputs: either points in three-space or points in homogeneous coordinates. Input points in three-space carry with them a tag field which is copied unchanged to the output.

The examples below will all use input points in three-space.
The Rotation Operation in the MA1024

Each input point to the MA1024 is transformed to produce a corresponding output point by a matrix multiplication defined as follows:

\[
\begin{bmatrix}
C_{xx} & C_{xy} & C_{xz} & 0 \\
C_{yx} & C_{yy} & C_{yz} & 0 \\
C_{zx} & C_{zy} & C_{zz} & 0 \\
T_x & T_y & T_z & 1
\end{bmatrix}
\begin{bmatrix}
X_o \\
Y_o \\
Z_o \\
1
\end{bmatrix}
= \begin{bmatrix}
X_i \\
Y_i \\
Z_i \\
1
\end{bmatrix}
\]

\([X_o Y_o Z_o] \text{ is the output point;}\)

\([X_i Y_i Z_i] \text{ is the input point;}\)

\(C_{ij}\) is the contribution of input coordinate \(i\) to output coordinate \(j\);

\(T_k\) is the amount of translation (lateral movement) in coordinate \(k\).

Data Formats

All numbers in the MA1024 are 16-bit two’s-complement fractions ranging in value from -1.0 to +0.9999. The most-significant bit of such a number (bit 15) is the sign bit; bits 0–14 are the fractional part. The implied binary point lies between bits 14 and 15.

An [x y z] point or an [x y z w] point requires four 16-bit numbers, or two 32-bit IKONAS data words. The two words for a point occupy consecutive locations in the SR8 memory, as follows:

For [x y z] points:

<table>
<thead>
<tr>
<th>Bits</th>
<th>3 3 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>first word</td>
<td>y coordinate</td>
</tr>
<tr>
<td>second word</td>
<td>xEi clip flags! tag field</td>
</tr>
</tbody>
</table>

E (bit 30) is the end-of-list bit, set in the last point to be rotated.

The clip flags are set by the clipping assist feature, if it is enabled. More about this feature under 'perspective transformation'.

For [x y z w] points:

<table>
<thead>
<tr>
<th>Bits</th>
<th>3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>first word</td>
<td>y coordinate</td>
</tr>
<tr>
<td>second word</td>
<td>w coordinate</td>
</tr>
</tbody>
</table>
Coefficient Matrix Format

The coefficient matrix to be used for a rotation is stored in the coefficient memory of the MA1024.

The MA1024 can hold up to 64 coefficient matrices simultaneously, though only one may be used for a single list of points.

Each coefficient matrix occupies 16 consecutive words of the coefficient memory, as follows (refer to the diagram above for the meaning of Cxx etc.):

Word 0  Cxx
Word 1  Cxy
Word 2  Cxx
Word 3  Tx
Word 4  Cyx
Word 5  Cyy
Word 6  Cyz
Word 7  Ty
Word 8  Czx
Word 9  Czy
Word 10 Czz
Word 11 Tz
Word 12 0
Word 13 0
Word 14 0 (used by the perspective division option)
Word 15 .9999
The Perspective Transformation and Clipping

The MA1024 allows perspective transformation by dividing the coordinates by a value dependent upon the distance of the object from the observer.

The clipping assist feature of the MA1024 may be used to compare the transformed coordinates of each point against six clipping planes; the MA1024 will set six flags, one for each clipping plane, to show whether the point is beyond the plane or not.

The steps in perspective transformation with clipping are:

1. The input points are defined to be in 'world coordinates', with values ranging from -1 to +.9999.

2. Use the MA1024 to rotate the input points and simultaneously convert them to 'clipping coordinates'. Clipping coordinates are similar to world coordinates, except that w (the perspective coordinate) is computed, and x, y, and w are scaled to allow clipping of x and y against w. Examples of this transformation follow. During this step the clipping assist feature is activated to cause the MA1024 to set flag bits if a coordinate is outside the clipping planes. The z-coordinate is left in world coordinates, except that the distance of the near clipping plane is subtracted from it.

3. Use the BPS32 to handle the clipped points. This may involve discarding them or replacing clipped vectors.

4. Use the MA1024 again to perform the perspective division by w. Since w is not stored in the point list, it is recalculated from z.
Perspective and Clipping Sequence

Initial World Coordinates

Clipping Coordinates

Out-of-bounds Points Clipped by the BPS32

Final Perspective View
The Conversion from World Coordinates to Clipping Coordinates

The MA1024's conversion from world coordinates to clipping coordinates consists of three components:

1. Calculation of the transformed x, y, and z coordinates in the same manner as described above for simple rotation

2. Calculation of w - the perspective coordinate - as a function of the transformed z and the desired viewing pyramid

3. Adjustment of z by subtracting the distance to the near clipping plane

W (the perspective coordinate) is significant because the final operation in the perspective transformation is division of x and y by w. W represents the apparent distance of the object from the observer. If the transformation is to simulate the view of a telescopic lens, i.e., if the object is to be made to appear close to the observer, w will be made a fraction of z. If the transformation is to give a wide-angle view, i.e., the object is to appear far away, x and y will be scaled down accordingly. x and y are scaled down instead of scaling up w, since it is impossible to scale up using the MA1024. Since x and y will be divided by w later, scaling down x and y is equivalent to scaling up w.

Z is the absolute distance from the object to the observer. Z is adjusted by subtracting a constant, which represents the nearest object which may be viewed.
The Clipping Assist Feature

Conversion to clipping coordinates defines a viewing pyramid within which all points are visible, and outside of which all points are off the screen. This viewing pyramid is defined by the equations:

\[-w < x < +w\]
\[-w < y < +w\]
\[0 < z < z\text{range}\]

That is, all points for which \(x\) or \(y\) are greater than \(w\) in absolute value are off the screen. When the viewing pyramid is defined in this way, the division by \(w\) results in numbers in the range \(-1\) to \(+0.9999\), the legal range for the MA1024.

The \(z\text{range}\) given in the equations above is the distance between the nearest object to be displayed and the farthest. It will be remembered that \(z\) was adjusted by subtracting the distance to the nearest object; so clipping against 0 and the \(z\text{range}\) will discard all points which are either too near or too far away.

The clipping assist feature sets six flags in the tag field of each point. The six bits are defined as follows:

\[
\begin{array}{ccccccc}
\text{Bits} & 3 & 0 & 9 & 1 & 0 & 9 \\
\text{Value} & 3 & 2 & 2 & 2 & 2 & 1 \\
\text{Flags} & & & & & &
\end{array}
\]

- \(x_l\) (bit 24) is set if \(x\) is less than \(-w\)
- \(x_h\) (bit 25) is set if \(x\) is greater than \(w\)
- \(u_l\) (bit 26) is set if \(y\) is less than \(w\)
- \(u_h\) (bit 27) is set if \(y\) is greater than \(w\)
- \(z_l\) (bit 28) is set if \(z\) is less than \(z\text{range}\)
- \(z_h\) (bit 29) is set if \(z\) is greater than \(z\text{range}\)

\(z\text{range}\) must have been previously stored at 20402$3. The clipping assist bit (bit 15) must have been set in 20402$0. If \(w\) is negative, the sense of each test is reversed: \(x_l\) is set if \(x\) is greater than \(-w\), etc.
The Perspective Division

Perspective division is simply division of x, y, and z by w. The MBINW operation in the MA1024 is used to perform the division. The following observations apply:

1. w must be greater than the number being divided. Any x, y, or z values greater than w must be clipped by the BPS before division.

2. w is not stored with the point, but must be recalculated from z and rescaled. If w is scaled down from z, z must also be scaled so that it is less than w.

3. After the perspective division, a translation (to the center of the screen, usually) may be performed.
MA1024 Operating Sequence

The steps to use of the MA1024 are the following:

1. Load the correct MA1024 microcode routine into the program memory. Microcode routines to perform operations described in this document are supplied by IKONAS and may be loaded using the IKLOD subroutine. The routine to perform rotation is named MHEOL.

2. Load the coefficient matrix for the rotation into the coefficient memory.

3. Load the points to be transformed into the SRB. Be sure E (bit 30) is set in the last word.

4. Load the correct values into the MA1024 control registers for the following:
   1. MA1024 program address
   2. Coefficient matrix offset
   3. Input data address
   4. Output data address
   5. Clipping assist, if needed
   6. Zrange for clipping, if needed

See below for the formats of these control registers.

5. Start the MA1024 by storing 0 into address 20402#3 using a function code of 25.

Note

The procedure described above is the proper sequence for using the MA1024 to perform rotation. It does not use all the features of the MA1024. If you want to write custom MA1024 microcode, you should learn about the full power of the MA1024 by reading the MA1024 Programming Guide.
MA1024 Control Registers

The registers described below control the operation of the MA1024. All the registers may be accessed from the IKONAS bus in the address range 20402#0-20402#2.

<table>
<thead>
<tr>
<th>Bits</th>
<th>3 3 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 0 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20402#0</td>
</tr>
<tr>
<td></td>
<td>20402#1</td>
</tr>
<tr>
<td></td>
<td>20402#2</td>
</tr>
<tr>
<td></td>
<td>20402#3</td>
</tr>
</tbody>
</table>

**Program address** is the offset to the MA1024 microprogram to be used. For the IKONAS rotation program MMEOL, this value will be 0. For custom microcode, the value will be the same as the value of the ORG statement in the IKASAM assembly of the microcode.

**W** (bit 13) should be set to 0 for MMEOL. When this bit is 1, the MA1024 will expect input points in homogeneous coordinates: \([x y z w]\). IKONAS does not supply microcode which requires \(W\) set to 1.

**N** (bit 14) is set to suppress normalization during perspective division and computation of \(w\). \(N\) should be off unless perspective division is being performed; in particular, it should be off when the clipping assist feature is on (see bit 15 below). For compatibility with the IKONAS RDS-2000 MA1024, \(N\) should be off.

**C** (bit 15) is set to engage the clipping assist feature. When this bit is on, bits 24-29 of the tag field of each output point will contain the clipping flags, calculated as described under the 'perspective transformation' page.

**Coefficient address** is the offset to the coefficient matrix to be used. If this offset is specified as \(cccc\), the coefficient matrix must start at 20400#cccc.

**Input and output offsets** are the offsets to the input and output areas in the SRB. If one of these offsets is specified as \(xxxx\), the area at 20200#xxxx will be used for input or output. The last point in the input list must have the end-of-list (E) bit set.

**Loop O counter value** is a value which may be used by user microcode. IKONAS microcode does not use the loop O counter, which may be left uninitialised.
Zrange for clipping assist is the distance between the nearest and farthest objects to be displayed. The setting is meaningful only if clipping assist is engaged.
Diagram of MA1024 Control Flow

Coefficient Memory

2040060

Control Registers

20400960

Program Memory

20401960

ISR out ISR in

ISR out ISR in

2040061777

SRS MEMORY

2020060

20200699

111 tag

end

20200899

111 tag

20200899

y

101 tag

(more points)

y

111 tag

20200899

y'

101 tag

(more points)

y'

111 tag

INPUT LIST

OUTPUT LIST
THE CGM4 CHARACTER GENERATOR

IKONAB BUS

external interfaces <-> 68000 and peripherals

---

IKONAB BUS
The CGM4 Character Generator

- Writes characters directly into the DR64 framebuffer
- Character shade is set by user
- Fast — writes 1 million pixels per second after startup
- Allows variable character size — up to 32x32
- Allows variable spacing between characters
- Uses SR8 for font memory to allow user-programmed character set
- Up to 32 full ASCII fonts available at one time with one SR8
- Allows magnification of characters without changing font
- Allows variable direction: left, right, up, down
- Can fill background of character with specified shade, or leave it unchanged
- On-board-font option gives standard 7x9 font without requiring an SR8 or font programming
CGM4 Programming

The CGM4 converts a character string in memory into pixels in the DR64 framebuffer. The pixels drawn into the framebuffer will be a visible representation of the character string when viewed on the display monitor.

When the CGM4 starts to draw a character, it consults a font table which indicates which pixels should be written for the character code. The font table is stored in an SR8 memory in coded form.

To draw characters with the CGM4, the following information is needed:

1. Description of the font:
   1. the font table - a map of each character
   2. the address of the font table
   3. the width of each character
   4. the height of each character
   5. the spacing between characters

2. Color of the characters, and background color if applicable

3. Magnification factor for the characters

4. Address of the character string to be displayed

5. Address in the DR64 framebuffer at which the characters are to be drawn

6. The character string itself

7. Mode settings, including the GO bit to start the operation (see CGM4 Base Control Block)
Coding the Font Table Entry for a Character

The font table consists of one entry for each character in the character set. Each entry indicates which pixels should be written for the corresponding character.

To create a font table entry for a character, follow this procedure:

1. Note the width and height (in pixels) of the characters in the font. Call the width W and the height H; in this example W=7 and H=9. Each character in the font will occupy the same amount of space on the screen.

(continued)
Font-table Entry Example - continued

2. Draw a picture of the character, \( W \) pixels wide and \( H \) pixels high, noting which pixels should be written for the character:

```
!(-7->!)
- * *
^ **
| * *
| * *
9 * *
| ######
| * *
v * *
- * *
```

3. Change each cell within the \( W \times H \) grid to a 0 if it is a space, or a 1 if it is not:

```
0001000
0010100
0100010
1000001
1111111
1000001
1000001
1000001
1000001
```

4. Join the rows of the grid into one long string of bits. There will be \( W \times H \) bits. The last bit in each row will be immediately followed by the first bit of the next row.

```
00010000001010001000101000001111111100000110000011000001
```

5. Break the string into 32-bit words. If there is not an even multiple of 32 bits in the string, add bits to the end to fill out the last 32-bit word. Copy the string from left to right, and fill up the 32-bit words from high-order bit to low-order bit.

Word 0: 000100000010100010001010000011111
Word 1: 11110000001100000011000001100001

6. You now have the font entry for the character. Install it in the font table at the proper address, as shown below.
Installing Font Table Entries Within the Font Table

After you have created the font table entry for a character, you must still store the entry into the font table at the address expected by the CGM4.

The CGM4 uses three pieces of information to calculate the address of the font table entry for a character it is about to display: the font table address, the number of words per font table entry, and the ASCII code for the character.

The address of the font table entry for a character should be equal to:

\[
\text{font table address} + (\text{number of words per entry} \times \text{ASCII code})
\]

In other words, the CGM4 uses the ASCII character code to index into the font table.

Example. If the base address of a 7x9 font is 20100$0$, what is the address of the entry for uppercase A? Solution. The ASCII code for A is 101 octal. There are 2 words per font table entry, so the address of the entry for A will be 20100$0 + (2 \times 101) = 20100$202.

```
font table
20100$0          ---------
    |    |
    |---/\---|
20100$202| entry |     |
    | for A |
    |------|
20100$204| entry |     |
    | for B |
    |------|
20100$206| entry |     |
    | for C |
    |---/\---|
20100$376         ---------
```
Installing the Character String to be Displayed

The character string is stored into memory as a sequence of 8-bit characters, four characters per IKONAS 32-bit word.

Within each IKONAS word, the first character to be displayed is the low-order character (bits 0-7); then bits 8-15; etc.

When all four characters from an IKONAS word have been displayed, the next sequential IKONAS word is fetched and the characters in it are displayed.

The end of the character string is indicated by a character consisting of eight zero bits (character code 000, the ASCII NUL character). This may come in the middle of an IKONAS word, so that the display need not be a multiple of four characters.

The characters will be displayed in the order shown:

```
20200$200 | 4 | 3 | 2 | 1 |
20200$201 | 8 | 7 | 6 | 5 |
20200$202 | 12 | 11 | 10 | 9 |
```

eetc.
Starting the CGM4

After the font has been loaded and the character string has been loaded, all that remains is to set up the CGM4 control blocks. There are two: the CGM4 base control block at 206000$0$ and the CGM4 auxiliary control block (CGMCB), which can be anywhere in memory and is pointed to by the CGM4 base control block.

Format of the CGMCB:

<table>
<thead>
<tr>
<th>Bits</th>
<th>3</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CGMCB-0</td>
<td>char. height</td>
<td>char. width</td>
<td>font table offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGMCB-1</td>
<td>shade to use for pixels in the character</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGMCB-2</td>
<td>shade to use for background pixels, if bit 12 of 206000$0=$1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGMCB-3</td>
<td>char. spacing</td>
<td>starting pixel address for output (yyyyyyyy)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGMCB-4</td>
<td>address of character string to display</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

char. height: height of a character in pixels

char. width: width of a character in pixels

font table offset: offset of the font table within the SR8 memory. If the SR8 base address is 201000$0$, the font table will be at 201000$0$+font-table-offset.

character shade: shade to be written to pixels written in the character

background shade: shade to use for background pixels. This applies only if NON-TRANSPARENT mode is selected (see the CGM4 base control block)

char. spacing: number of pixels to skip between characters

starting output address: pixel address to use for the top-left pixel of the first character

character string address: address of characters to be displayed
CGM4 Base Control Block

Format of the CGM4 base control block (except for bit 8 of 20600 on, these locations are read-only):

<table>
<thead>
<tr>
<th>Bits</th>
<th>3</th>
<th>3</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>20600</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>206001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DIR (bits 13-14): gives the orientation to be used.
00=normal left-to-right, 01=top-to-bottom, 10=bottom-to-top, 11=right-to-left.

I (bit 12): if on, transparent mode is used: background pixels are left unchanged. If off, non-transparent mode is used: the background shade is written to background pixels.

W (bit 11): on for word mode writes, off (normal) for pixel mode.

E (bit 10): on to use the optional on-board 7x9 font, off to use to SBR font memory.

G (bit 8): the GO bit. When this bit is set, the CGM4 starts to draw the character string. All information needed by the CGM4 should be ready when the GO bit is set. When 20600 on is read from, bit 8 will be on if the CGM4 is busy, off if it is idle and can start another operation.

x zoom (bits 4-7): the magnification factor in the x-direction: a setting of 1 (normal) is no magnification; a setting of 2-17 octal magnifies the x-direction by that amount; a setting of 0 magnifies the x-direction 16 times.

y zoom (bits 0-3): magnification in the y-direction, similar to x zoom.
THE IF/IK HOST INTERFACE

- Direct DMA connection to user's host computer
- Fast - 2 million bytes per second transfer rate
- Allows non-DMA I/O of single words to avoid host DMA overhead
- Multiple modes: 32-bit word, 16-bit halfword, and 8-bit byte:
  - 32-bit word mode allows easy access to 32-bit data
  - 16-bit halfword mode allows efficient access of 16-bit and smaller items
  - 8-bit byte mode allows efficient access to a single red, green, or blue image component
- Vectored interrupts to host based on IKONAS events
- Allows host-programmable system reset
- Allows transfer to be inhibited except during blanking interval
IF/IK Transfer Modes

The IF/IK is the connection from the IKONAS to the user's host computer. Use the IF/IK to start a transfer between the IKONAS bus and the host computer's memory.

The examples here will assume a host computer with a 16-bit word; for example, a DEC PDP-11 or VAX-11 with a UNIBUS.

The IF/IK supports three transfer modes:

1. **WORD mode:** two host 16-bit words correspond to one IKONAS 32-bit words.

2. **HALFWORD mode:** one host 16-bit word corresponds to the low-order bits (bits 0-15) of one IKONAS 32-bit word. Bits 16-31 of the IKONAS 32-bit word are zero.

3. **BYTE mode:** each 8-bit byte of host data corresponds to one IKONAS 32-bit word. The IKONAS 32-bit word is addressed as four bytes (bits 0-7, 8-15, 16-23, and 24-31); the user specifies which byte receives or transmits with the host.
Transfer Mode Examples

Word Mode

HOST

| word 0 | <----------------------------- |
|--------|
| word 1 | <----------------------------- |
|        |                                  |
|        |                                  |
|        |                                  |
|        | v                                 |
|        | v                                 |

IKONAS: | bits 16-31 | bits 0-15 |

Halfword Mode

HOST

| word | <----------------------------- |
|      |                                  |
|      |                                  |
|      | v                                 |

IKONAS: | 0 | bits 0-15 |

Byte Mode (example using byte 1)

HOST

| Hi byte!Lo byte! | <----------------------------- |
|                 |                                  |
|                 | v                                 |

IKONAS: | word 0: | x | x | data | x | x |

IKONAS: | word 1: | x | x | data | x | x |

The bytes marked x are undefined.
Notes on the Transfer Modes

1. In word mode, the even host words correspond to the low-order half of the IKONAS words; the odd host word correspond to the high-order half of the IKONAS words. This is in accordance with the host computer's numbering of words in INTEGER*4 variables.

2. In halfword mode, each host word corresponds to the Low-order half of an IKONAS word. However, during write operations, all 32 bits are written; bits 16-31 are written as 0.

3. In byte mode, each host byte corresponds to a selected byte of an IKONAS word. However, during write operations, all 32 bits are written; bits in bytes other than the selected byte are undefined. If you want to write to a single channel of the DR64 framebuffer (for example, the green component, byte 1), you should set the write mask to protect all bits except the channel to be written.
Starting an I/O Operation

The steps to starting an I/O operation with the IF/IK are:

1. Set up the host half of the interface, as specified in the programming guide for your host interface. (See the IKONAS PROGRAMMING REFERENCE MANUAL.)

2. Set up the IKONAS CONTROL REGISTER, described below.

3. Start the I/O operation as described in the programming guide.

4. Wait for completion interrupt from the IF/IK.
The IKONAS Control Register

The IKONAS control register defines the transfer mode for all I/O operations using the IF/IK. The format is as follows:

```
<table>
<thead>
<tr>
<th>Bits</th>
<th>1 1 1 1 1 1 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYT</td>
<td>PI</td>
</tr>
</tbody>
</table>
```

**func** (bits 0-4): the IKONAS bus function code. Bit 4 is always set for write operations, off for read operations. For most RDS3000 modules, the write function code in octal 20, read is 00; but each module may accept special function codes. Refer to the individual module descriptions. In particular, the DR64 uses many function codes.

**H** (bit 5): set for halfword mode, off for word or byte mode. When bit 5 is set, bit 11 must be reset.

**D** (bit 6): set for DMA transfers, reset for programmed I/O. The IKONAS I/O driver in the host will normally set this bit automatically without special action by the user.

**I** (bit 7): set for invisible I/O, reset normally. Invisible I/O occurs during the blanking portion of the picture. It is useful mainly for accesses to the LUVO, since any access to the LUVO during the picture causes an interruption of the video signal.

**±** (bit 8): set to increment the IKONAS address between words. This bit should always be set.

**X** (bit 9): set to allow the BPS32 to execute. When this bit is reset, the BPS32 will stop; setting the bit again will cause the BPS32 to resume from the point at which it stopped.

**R** (bit 10): set to force an IKONAS system reset.

(continued)
The IKONAS Control Register - continued

B (bit 11): set for byte mode, off for word or halfword mode.

Specific to read operations:

E (bit 12): this bit is read-only. When it is on, the video signal is in the vertical blanking interval.

P (bit 13): this bit is read-only. When it is on, the IKONAS system has requested service from the host. The request may be from the MPC or from the BPS32.

BYT (bits 14-15): the byte number to be selected for byte mode transfers. This field is meaningful only when bit 11 (B) is set. 00=byte 0 (bits 0-7), 01=byte 1 (bits 8-15), 10=byte 2 (bits 16-23), 11=byte 3 (bits 24-31). Set by user for read operations.

Specific to write operations:

(bits 12-14): these 3 bits contain the sender id for write operations; they specify the write mask to be written or used for the DR64 or DR256 and the write mask and shade register for the QM64. Set by user for write operations.

(bit 15): write-only; reserved for future use.
The MPC 68000 Computer Module

- Powerful 68000 processor to give minicomputer flexibility and power
- Allows standalone processing in the RDS3000 system
- 256K bytes of memory
- Up to 32K bytes of ROM for fixed-function systems
- Four RS-232 serial ports to allow terminals, data tablets, and other devices
- Programmable event timer
- Connects to VERSAbus and Multibus - gives full range of device support
- Connects to the IKONAB PCP:
  - 4 RS-232 ports
  - 16 analog ports (dials, joysticks, etc.)
  - 16 parallel ports (buttons, LEDs, etc.)
### MPC Memory Map

*(All Addresses in hex)*

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC0000-FFFF7FF</td>
<td>IKONAS bus space segment 15</td>
</tr>
<tr>
<td>FB0000-F8FFFF</td>
<td>IKONAS bus space segment 14</td>
</tr>
<tr>
<td>C40000-F7FFFF</td>
<td>IKONAS bus space segments 1-13</td>
</tr>
<tr>
<td>C00000-C3FFFF</td>
<td>IKONAS bus space segment 0</td>
</tr>
<tr>
<td>800000-8FFFFF</td>
<td>IKONAS image space - 1024x1024 pixels</td>
</tr>
<tr>
<td>4FFE02-7FFFFF</td>
<td>reserved for VERSAbus devices</td>
</tr>
<tr>
<td>4FFE00-4FFE01</td>
<td>reserved</td>
</tr>
<tr>
<td>4FFD00-4FFDFF</td>
<td>reserved for VERSAbus devices</td>
</tr>
<tr>
<td>4FFD80-4FFD8F</td>
<td>MPU translation and control registers</td>
</tr>
<tr>
<td>4FFD40-4FFD7F</td>
<td>reserved for VERSAbus devices</td>
</tr>
<tr>
<td>4FFD3C-4FFD3F</td>
<td>Timer no. 3 value</td>
</tr>
<tr>
<td>4FFD38-4FFD3B</td>
<td>Timer no. 2 value</td>
</tr>
<tr>
<td>4FFD34-4FFD37</td>
<td>even byte reserved</td>
</tr>
<tr>
<td>4FFD32-4FFD31</td>
<td>Timer no. 1 value</td>
</tr>
<tr>
<td>4FFD30-4FFD2F</td>
<td>Ctrl. reg. 2/status</td>
</tr>
<tr>
<td>4FFD10-4FFD2F</td>
<td>Ctrl. reg. 0 or 3</td>
</tr>
<tr>
<td>4FFD0F</td>
<td>reserved for VERSAbus devices</td>
</tr>
<tr>
<td>4FFD0C</td>
<td>reserved</td>
</tr>
<tr>
<td>4FFD08</td>
<td>reserved</td>
</tr>
<tr>
<td>4FFD07</td>
<td>reserved</td>
</tr>
<tr>
<td>4FFD04</td>
<td>reserved</td>
</tr>
<tr>
<td>4FFD00</td>
<td>reserved</td>
</tr>
<tr>
<td>4FFC80-4FFCFF</td>
<td>PCP registers, as defined by PCP</td>
</tr>
<tr>
<td>4FFC20-4FFC7F</td>
<td>reserved for VERSAbus devices</td>
</tr>
<tr>
<td>4FFC00-4FFC1F</td>
<td>reserved for VERSAbus devices</td>
</tr>
<tr>
<td>040000-04FF8F</td>
<td>IKONAS space address translation table</td>
</tr>
<tr>
<td>008000-03FFFF</td>
<td>reserved for VERSAbus devices</td>
</tr>
<tr>
<td>000000-007FFF</td>
<td>program ROM and data RAM-same address</td>
</tr>
</tbody>
</table>
IKONAS Bus Access

- IKONAS bus is directly mapped to 68000 address space

- Two kinds of access:
  - Image mode: the image memory (1024x1024 pixels) is mapped to locations 800000-8FFFFF
  - Segment mode: C00000–FFFFFF is divided into 16 segments, each mapped to an area of IKONAS control space

- IKONAS bus function codes and sender IDs can be specified

- Fast interface allows 68000 to run while write cycles complete
IKONAS Address Translation Tables

There are 16 address translation table entries, one for image mode and segment 0, and one for each segment 1-15. Each translation table entry specifies:

The upper IKONAS address bits to use
The IKONAS bus function code to use
The IKONAS bus sender id to use
Whether the segment is to be mapped to the IKONAS bus or the VERSAbus

The translation table looks like:

```
1 1 1 1 1 1
5 4 3 2 1 0

4FFC1E (segment 15) | func | sid | !V!IKONAS ad 16-23!
---------------------
4FFC1C (segment 14) |     |    | !V!IKONAS ad 16-23!
4FFC02 (segment 1)  |     |    | !V!IKONAS ad 16-23!
4FFC00 (image & seg 0) | func | sid | !V!IKONAS ad 16-23!
```

`func` will be used for IKONAS bus function bits 0-3
`sid` will be used for the IKONAS sender id

Y should be set to cause this segment not to be mapped to the IKONAS bus. It should correspond to VERSAbus devices, or a bus error will result.

IKONAS ad 16-23 will be supplied as the upper bits of the IKONAS bus address
Image Memory Address Translation

Bit numbers: 2222111111111
32107865432109876543210
68000 address: B1F8E2
binary: 10000001100110001110010

<table>
<thead>
<tr>
<th>bit 22-23 = 10</th>
<th>select image mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>image mode translation table</td>
</tr>
<tr>
<td></td>
<td>4FFC00: 0010000000000000</td>
</tr>
<tr>
<td></td>
<td>IOKMAS address 120-23</td>
</tr>
<tr>
<td></td>
<td>IOKMAS address 0-19</td>
</tr>
<tr>
<td></td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td>must be 0</td>
</tr>
</tbody>
</table>

68000 read/write:

v v v v
v v v v

v 010 000 00000000011001100011000 = 3191070
4 0-3 0-2 0-23
IOKMAS IOKMAS IOKMAS
function sender address code (bits 20-33=LORES image number)
Segment Memory Address Translation

Bit numbers: 22221111111111

68000 address: 321098765432109876543210
binary: C 4 0 0 1 0
1100010000000000000010000

<table>
<thead>
<tr>
<th>bits 22-23=11</th>
<th>select segment mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits 18-21=segment number</td>
<td></td>
</tr>
</tbody>
</table>

4FF00: 0000000000000000

---
<table>
<thead>
<tr>
<th>IKONAS address</th>
<th>16-23</th>
</tr>
</thead>
<tbody>
<tr>
<td>IKONAS address</td>
<td>0-15</td>
</tr>
</tbody>
</table>

---
68000 read/write

---
| 0000 000 | 110000000000000000000000 |
| 0-3 0-2 0-23 |

IKONAS function | sender |
code | id |
IKONAS address
Serial Port Setup

To start a serial port, you must:

- Write the baud rate to the baud rate address
- Issue reset through the control register
- Initialize the control register

<table>
<thead>
<tr>
<th>Baud Rate Bits</th>
<th>Baud Rate Bits</th>
<th>Baud Rate Bits</th>
<th>Baud Rate Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>0-3</td>
<td>0-3</td>
<td>0-3</td>
</tr>
<tr>
<td>50 1111</td>
<td>150 1011</td>
<td>1800 0111</td>
<td>4800 0011</td>
</tr>
<tr>
<td>75 1110</td>
<td>300 1010</td>
<td>2000 0110</td>
<td>7200 0010</td>
</tr>
<tr>
<td>110 1101</td>
<td>600 1001</td>
<td>2400 0101</td>
<td>9600 0001</td>
</tr>
<tr>
<td>134 1100</td>
<td>1200 1000</td>
<td>3600 0100</td>
<td>19200 0000</td>
</tr>
</tbody>
</table>

Bits: 7 6 5 4 3 2 1 0

- Set 11 to reset the serial port
- Store 01 for normal operation
- Set as follows to select number of bits per character and parity:
  - 000 = 7 bits, even parity, 2 stop bits
  - 001 = 7 bits, odd parity, 2 stop bits
  - 010 = 7 bits, even parity, 1 stop bit
  - 011 = 7 bits, odd parity, 1 stop bit
  - 100 = 8 bits, 2 stop bits
  - 101 = 8 bits, 1 stop bit
  - 110 = 8 bits, even parity, 1 stop bit
  - 111 = 8 bits, odd parity, 1 stop bit
- Set to 01 to enable interrupts after characters are transmitted. 00 to suppress
- Set to enable interrupts when character received, reset to disable.
## Serial Port Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **7**: Set when character available to read
- **6**: Set when character may be written
- **5**: Set when modem carrier lost (should be 0)
- **4**: Set when modem can accept a character
- **3**: Set when framing error has occurred, or BREAK key pressed
- **2**: Set when receiver overrun occurred (a character was received before the program had read the previous character)
- **1**: Set when parity error detected
- **0**: Set when interrupt request pending
Serial Port Programming Sequence

1. Setup
   1. Store control register=03 to reset
   2. Set character length, receiver interrupt enable
      (example: control reg=8D for 7 bits, odd parity,
      one stop bit, all interrupts enabled)
   3. Set baud rate (example: baud rate generator=01 for
      9600 baud)

2. Read sequence
   1. Interrupt to level 2 (serial port interrupt level)
   2. Read status registers to determine source of
      interrupt - detect a received-character interrupt
   3. Read the character from the data register. This
      clears the interrupt request
   4. Process the character
   5. Wait for interrupt

3. Write Sequence
   1. Write character to data register and enable
      transmitter interrupt
   2. Wait for interrupt
   3. Interrupt to level 2 - decode and detect transmitter
      interrupt
   4. If another character is ready to send, write it and
      wait; else disable transmitter interrupt
Programmable Timer (MPC256)

The programmable timer consists of three separate timers, each with several modes of operation possible. For a complete description of the timer, see the Motorola MC6840 data sheet. The following is an example of timer use.

Setting a real-time clock or interval timer

Problem: to create a timer interrupt every 16 milliseconds (60 Hz interrupt rate) using timer number 1.

1. Select control register 1 by storing 01 at 4FFD33 (control register 2).

2. Set the timer mode by storing C2 into control register 1, location 4FFD31. Bit 0 of each timer control register has special meaning; for example, bit 0 in the control register for timer 1 is a master-reset bit; so if you wanted to set a different timer, the setting of bit 0 in the timer control register would depend on the timer being set.

3. Set the timeout value: 16000 (the number of 1-microsecond clock ticks in 16 milliseconds), stored into the timer 1 value, locations 4FFD35 (upper byte) and 4FFD37 (lower byte). The MOV instruction could be used for this.

4. The timer is started. Wait for an interrupt

5. When the timer expires, interrupt level 3 will be entered.

6. The interrupt routine should read the timer status register (4FFD33) to determine which timer interrupted. The three low-order bits of the status register indicate which timers have interrupted: bit 0 on=timer 1, bit 1=timer 2, bit 2=timer 3.

7. For each timer which has interrupted, the interrupt routine should read the counter value. Reading the counter value clears the interrupt from that counter.

8. The interrupt routine performs its processing.

9. If a continuous real-time clock is needed, the interrupt routine exits; the timer will interrupt again when the time expires. If a single interval is needed, the interrupt should be disabled by storing 0 into control register 1.
IKONAS bus access to the MPC

- RAM occupies IKONAS addresses 34000H-34377H
- I/O space occupies IKONAS addresses 34777H-34777H
- Each IKONAS access transfers 16 bits (two bytes) to bits 0-15 of the IKONAS bus
- IKONAS accesses pass through the MMU for translation
**Accessing MPC memory from the IKONAS BUS**

Bits: 2222111111111111
IKONAS addr: 32109876543210 9876543210

---

<table>
<thead>
<tr>
<th>select MPC</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>select memory</td>
<td>168000</td>
</tr>
<tr>
<td>space</td>
<td>1-18</td>
</tr>
<tr>
<td>--- odd/even bit</td>
<td></td>
</tr>
<tr>
<td>68000 addr</td>
<td>00000 000000110100100000</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0 4 A 4 x</td>
</tr>
<tr>
<td>1A40&lt;=IKONAS bus bits 8-15</td>
<td></td>
</tr>
<tr>
<td>1A41&lt;=IKONAS bus bits 0-7</td>
<td></td>
</tr>
</tbody>
</table>

**Accessing MPC I/O space from the IKONAS BUS**

Bits: 2222111111111111
IKONAS addr: 32109876543210 9876543210 (PCP register 0)

---

<table>
<thead>
<tr>
<th>select MPC</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>select I/O</td>
<td>168000</td>
</tr>
<tr>
<td>space</td>
<td>1-18</td>
</tr>
<tr>
<td>--- odd/even bit</td>
<td></td>
</tr>
<tr>
<td>68000 addr</td>
<td>01001 11111111010000000</td>
</tr>
<tr>
<td></td>
<td>4 F F 0 8 x</td>
</tr>
<tr>
<td>4FFC80&lt;=IKONAS bus bits 8-15</td>
<td></td>
</tr>
<tr>
<td>4FFC81&lt;=IKONAS bus bits 0-7</td>
<td></td>
</tr>
</tbody>
</table>
# Table of I/O Addresses

<table>
<thead>
<tr>
<th>Device</th>
<th>MPC address</th>
<th>IKONAS bus address</th>
</tr>
</thead>
<tbody>
<tr>
<td>IKONAS translation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>4FFC00-4FFC1F</td>
<td>34777$1000-34777$1037</td>
</tr>
<tr>
<td>PCP</td>
<td>4FFC80-4FFCFF</td>
<td>34777$1100-34777$1177</td>
</tr>
<tr>
<td>Serial ports</td>
<td>4FFD00-4FFD0F</td>
<td>34777$1200-34777$1217</td>
</tr>
<tr>
<td>Timer</td>
<td>4FFD30-4FFD3E</td>
<td>34777$1230-34777$1247</td>
</tr>
<tr>
<td>MMU</td>
<td>4FFD80-4FFDBF</td>
<td>34777$1300-34777$1337</td>
</tr>
<tr>
<td>Host interrupt</td>
<td>4FFE01</td>
<td>34777$1400</td>
</tr>
</tbody>
</table>