Kyouko

"Imizu Chip Series"

Programmer's Reference
Manual

January 26, 2016

Contents

1 Device Overview **4**
1.1 Introduction . . . 4
1.2 Extensions . . . 4
1.3 Architectures . . . 4
1.4 Terminology . . . 5

2 MMIO Registers **6**
2.1 Device ROM . . . 6
2.1.1 ROM.Version.Major . . . 6
2.1.2 ROM.Version.Minor . . . 6
2.1.3 ROM.Manuf.ID . . . 6
2.1.4 ROM.Manuf.Week . . . 7
2.1.5 ROM.Manuf.Year . . . 7
2.1.6 ROM.Manuf.Lot . . . 7
2.1.7 ROM.Manuf.Location . . . 7
2.1.8 ROM.Device.VRAM . . . 7
2.1.9 ROM.Device.Features . . . 8
2.1.10 ROM.Device.Frames . . . 8
2.1.11 ROM.Device.Encoders . . . 8
2.2 Configuration Registers . . . 9
2.2.1 Config.Reboot . . . 9
2.2.2 Config.ModeSet . . . 9
2.2.3 Config.Interrupt . . . 9
2.2.4 Config.Acceleration . . . 9
2.2.5 Info.Fifo.Start . . . 10
2.2.6 Info.Fifo.End . . . 10
2.3 Info and Status . . . 11
2.3.1 Info.Fifo.Head . . . 11
2.3.2 Info.Fifo.Tail . . . 11
2.3.3 Info.Status . . . 11

3 FIFO Commands **12**
3.1 Interaction . . . 12
3.2 Rasterization . . . 13
3.2.1 Raster.Primitive . . . 13
3.2.2 Raster.Emit . . . 13
3.2.3 Raster.Clear . . . 13
3.2.4 Raster.TargetFrame . . . 13
3.2.5 Raster.Flush . . . 14
3.3 Drawing State . . . 15
3.3.1 Draw.Vertex.Coord4f . . . 15
3.3.2 Draw.Vertex.Color4f . . . 15
3.3.3 Draw.Vertex.Transform16f . . . 16
3.3.4 Draw.Clear.Color4f . . . 16

3.4 Stream Control 17
3.4.1 Stream.BufferA.Address 17
3.4.2 Stream.BufferA.Config 17
3.4.3 Stream.BufferB.Address 17
3.4.4 Stream.BufferB.Config 17

4 Objects **18**
4.1 Frames 19
4.1.1 Limitations 19
4.1.2 Frame.Columns 19
4.1.3 Frame.Rows 19
4.1.4 Frame.RowPitch 19
4.1.5 Frame.PixelFormat 20
4.1.6 Frame.StartAddress 20
4.2 Encoders 21
4.2.1 Encoder.Width 21
4.2.2 Encoder.Height 21
4.2.3 Encoder.OffsetX 21
4.2.4 Encoder.OffsetY 22
4.2.5 Encoder.Frame 22
4.3 Setting a Display Mode 23
4.4 Example Configuration for 1024x768 32bpp 23
4.5 Mode Changes 23

5 Rasterization Sequences **24**

1 Device Overview

1.1 Introduction

The Kyouko Graphics Accelerator is a modern, high performance device designed for use in mid-tier desktop systems. It consists of 2D and 3D subsystems, both of which are highly configurable via an extensive set of registers. The 2D subsystem supports legacy software via an optinal emulation interface designed to mimic an IBM-compatible VGA adapter. A full set of 2D acceleration features are present, including hardware bitblit and cursor drawing operations.

This Programmer's Reference Manual (PRM) is intended for hardware, software, and firmware designers who seek to implement or utilize the graphic functions of the Kyouko Graphics Accelerator. Familiarity with basic 2D and 3D graphics programming is assumed.

1.2 Extensions

A particular hardware implementation of the Kyouko device may not support all of the features described in this document. A properly implemented driver must be capable of detecting those extensions the present device supports and possibly emulating for user code those features which are not available. Registers or functionality that requires one of these extensions will be marked in the tables of the following sections. The presence of these features may be detected by reading certain ROM fields.

1.3 Architectures

The PCI bus is commonly used on a number of architectures, each with different properties. The bus is CPU ignorant, meaning that data must be marshalled by software into the correct forms for the device - little endian PCI devices must be handled properly on big endian machines, and vice-versa. All chips from the Takaoka series are little endian to provide maximum performance on x86-derived architectures. A portable device driver should take this into account to work on platforms that are big endian, such as PowerPC.

1.4 Terminology

Term	Description
VGA	Video Graphics Array, a de-facto standard for computer display hardware introduced 1987 by IBM
MM	Memory Mapped
MMIO	Memory-Mapped I/O
QW	Quad-Word consisting of 64 bits
DMA	Direct Memory Access
APIC	Advanced Programmable Interrupt Controller, a device used on modern systems to control and distributed interrupts on multiprocessor systems
PCI	Peripheral Component Interconnect
OOB	Out of Band
MSI	Message Signaled Interrupt, a method of multiplexing a vast number of virtual interrupt lines to the relatively small number of hardware interrupt lines available in modern I/O APIC systems
HDMI	High-Definition Multimedia Interface
FIFO	First In First Out (a queue)
RAMDAC	Random Access Memory Digital-to-Analog Converter
ROM	Read Only Memory, commonly used to store firmware or hardware information
BPP	Bits Per Pixel
OpenGL	Open Graphics Library, an interface defined by the Khronos Group for building high performance 3D applications
R/W (register)	A register that may be both read and written multiple times
RO (register)	A register that may only be read from
WO (register)	A register that may only be written to
uintZ	An unsigned integer type of Z bits
uintZ[X:Y]	The portion defined by the bits [X,Y] of an unsigned integer type Z bits in precision
ieee754-*	Represents a numerical type as described by IEEE 754 floating-point technical standard

2 MMIO Registers

All device configuration is controlled via a set of 32-bit registers, mappable into the host address space. Reads and writes to the area are processed by a special unit of the device. They do not go directly to device memory. Most operations depend on and modify state variables, so the driver must ensure that memory accesses are properly ordered. This usually means caching should be disabled for the physical address range.

In this section, all the user-accessible registers are detailed. Their names, offsets, initial values, and valid operations are listed.

2.1 Device ROM

2.1.1 ROM.Version.Major

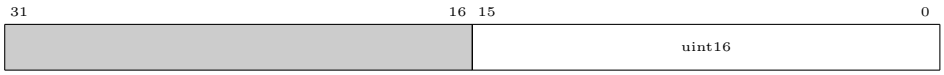
I/O Offset: 0x0000, Default: Hardware Dependent, Attributes: RO



Bits 0:15 Chip Major Revision
Bits 16:31 Reserved (Read as zero)

2.1.2 ROM.Version.Minor

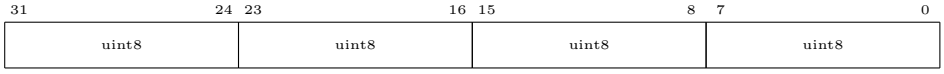
I/O Offset: 0x0004, Default: Hardware Dependent, Attributes: RO



Bits 0:15 Chip Minor Revision
Bits 16:31 Reserved (Read as zero)

2.1.3 ROM.Manuf.ID

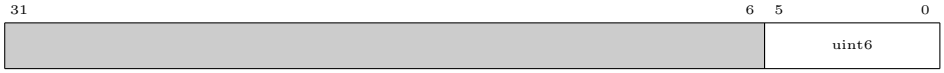
I/O Offset: 0x0008, Default: Hardware Dependent, Attributes: RO



Bits 0:7 ASCII character 0
Bits 8:15 ASCII character 1
Bits 16:23 ASCII character 2
Bits 24:31 ASCII character 3

2.1.4 ROM.Manuf.Week

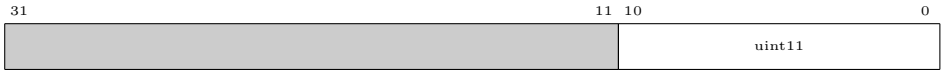
I/O Offset: 0x0010, Default: Hardware Dependent, Attributes: RO



Bits 0:5 Week of the year, 0-indexed
Bits 6:31 Reserved (Read as zero)

2.1.5 ROM.Manuf.Year

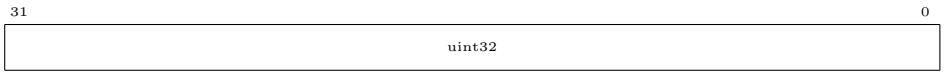
I/O Offset: 0x0014, Default: Hardware Dependent, Attributes: RO



Bits 0:10 Year
Bits 11:31 Reserved (Read as zero)

2.1.6 ROM.Manuf.Lot

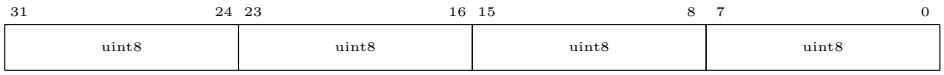
I/O Offset: 0x0018, Default: Hardware Dependent, Attributes: RO



Bits 0:31 Lot Identifier (varies)

2.1.7 ROM.Manuf.Location

I/O Offset: 0x001C, Default: Hardware Dependent, Attributes: RO



Bits 0:7 ASCII character 0
Bits 8:15 ASCII character 1
Bits 16:23 ASCII character 2
Bits 24:31 ASCII character 3

2.1.8 ROM.Device.VRAM

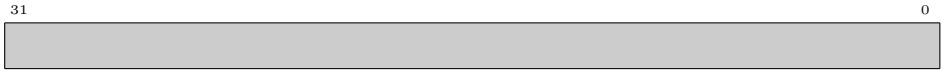
I/O Offset: 0x0020, Default: Hardware Dependent, Attributes: RO



Bits 0:13 Global VRAM, in Mebibytes
Bits 14:31 Reserved (Read as zero)

2.1.9 ROM.Device.Features

I/O Offset: 0x000C, Default: Hardware Dependent, Attributes: RO



Bits 0:31 Reserved (Read as zero)

2.1.10 ROM.Device.Frames

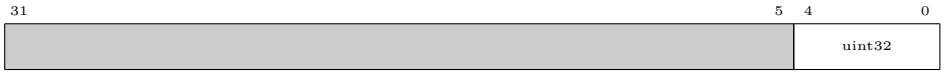
I/O Offset: 0x0024, Default: Hardware Dependent, Attributes: RO



Bits 0:4 Number of Frame Objects supported by device
Bits 5:31 Reserved (Read as zero)

2.1.11 ROM.Device.Encoders

I/O Offset: 0x0028, Default: Hardware Dependent, Attributes: RO



Bits 0:4 Number of Encoder Objects supported by device
Bits 5:31 Reserved (Read as zero)

2.2 Configuration Registers

2.2.1 Config.Reboot

I/O Offset: 0x1000, Default: N/A, Attributes: WO



N/A Any write to this register will immediately reboot the device. Configuration changes will be lost, and the device will return to legacy VGA-compatible mode.

2.2.2 Config.ModeSet

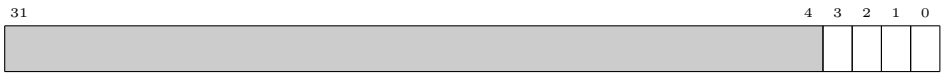
I/O Offset: 0x1008, Default: N/A, Attributes: WO



N/A Any write to this register will flush encoder frame configuration. If chosen frame configurations are invalid, device may reset or maintain previous values.

2.2.3 Config.Interrupt

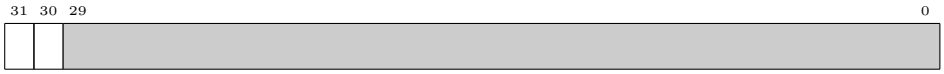
I/O Offset: 0x100C, Default: 0x00000000, Attributes: RW



Bit 0	Interrupt on device error
Bit 1	Interrupt on Buffer A flush
Bit 2	Interrupt on Buffer B flush
Bit 3	Interrupt on V-Sync
Bits 4:31	Reserved (Read as zero)

2.2.4 Config.Acceleration

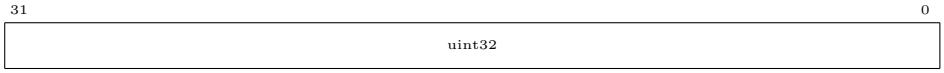
I/O Offset: 0x1010, Default: 0x80000000, Attributes: RW



Bits 0:29	Reserved (Read as zero)
Bit 30	Accelerated 3d pipeline enable 0 – Only 2D-mode active 1 – Use 3D-context
Bit 31	Legacy VGA enable 0 – Native Kyouko interface 1 – Emulate IBM VGA adapter

2.2.5 Info.Fifo.Start

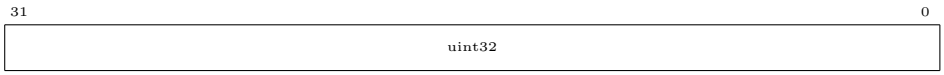
I/O Offset: 0x1020, Default: N/A, Attributes: WO



Bits 0:31 Physical memory location for start of FIFO memory

2.2.6 Info.Fifo.End

I/O Offset: 0x1024, Default: N/A, Attributes: WO

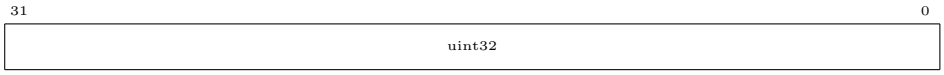


Bits 0:31 Physical memory location for one byte past end of FIFO memory

2.3 Info and Status

2.3.1 Info.Fifo.Head

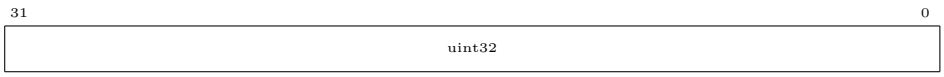
I/O Offset: 0x4010, Default: N/A, Attributes: WO



Bits 0:31 Position of FIFO head, as an index

2.3.2 Info.Fifo.Tail

I/O Offset: 0x4014, Default: N/A, Attributes: RO



Bits 0:31 Position of the FIFO tail, as an index

2.3.3 Info.Status

I/O Offset: 0x4008, Default: 0x00000000, Attributes: RW



Bit 0 Device error (stalled)
Bit 1 Buffer A flushed
Bit 2 Buffer B flushed
Bit 3 V-Sync
Bits 4:31 Reserved (Read as zero)

3 FIFO Commands

3.1 Interaction

In order to increase the bandwidth and concurrency of command execution, commands are sent to the device via a host memory ring buffer. The driver should initialize the FIFO with a 32-bit memory region of physical memory. Each entry in this FIFO is 8 bytes, consisting of a 32-bit command type and a 32-bit command operand. Some commands do not require an operand, in which case the operand is ignored.

The FIFO follows a single producer, single consumer model. Entries are placed at the head by the host and removed by the device from the tail. The FIFO is never allowed to be full, as this would create ambiguity in the case where the head and tail are at the same position - it could be either full or empty. Thus if the head and tail values are equal, this assumption lets us assume the FIFO is empty. As reads to a device mapped address are expensive, the driver should maintain its own copy of the FIFO state and check withe device for the current tail value only when necessary. It is not possible to read the head position back from the device.

When configuring the device FIFO, the CfgFifoStart and CfgFifoEnd registers need to be written with the first byte and one past the final byte of the configured memory region, respectively. In practice this means CfgFifoEnd shall be equal to CfgFifoStart plus the size of the memory in bytes. After these registers are changed, the device will internally reset both its head and tail pointers to 0.

For commands which have multiple 32-bit fields, each can be addressed as an offset of the command's address. For example, updating the Y coordinate used on the next vertex emit would use a command of 0x5004 and an operand of the new Y value.

3.2 Rasterization

3.2.1 Raster.Primitive

I/O Offset: 0x3000, Default: N/A, Attributes: FIFO

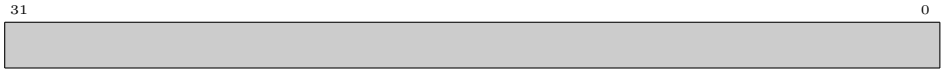


Bits 3:0 Change card rasterization mode
- 0 – End rasterization sequence
- 4 – Begin triangles
- 5 – Begin triangle strip
- 6 – Begin triangle fan

Bits 31:4 Reserved (Write zero)

3.2.2 Raster.Emit

I/O Offset: 0x3004, Default: N/A, Attributes: FIFO



N/A Any write to this register will cause a single vertex to be emitted to the rasterizer subsystem. The state of the Draw registers at execution are used.

3.2.3 Raster.Clear

I/O Offset: 0x3008, Default: N/A, Attributes: FIFO



Bit 0 Color Buffer
- 0 – Do not modify color buffer
- 1 – Fill color buffer with Draw clear color

Bit 1 Depth Buffer
- 0 – Do not modify depth buffer
- 1 – Fill depth buffer with zeroes

Bits 2:31 Reserved (write zero)

3.2.4 Raster.TargetFrame

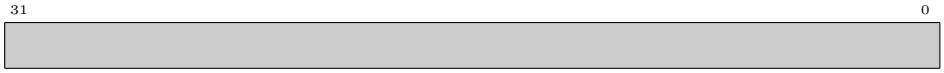
I/O Offset: 0x3100, Default: N/A, Attributes: FIFO



Bits 4:0	Raster target (frame object index)
Bits 31:5	Reserved (Write zero)

3.2.5 Raster.Flush

I/O Offset: 0x3FFC, Default: N/A, Attributes: FIFO



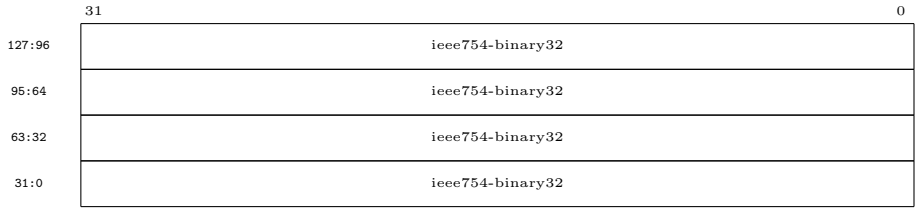
N/A	Execution will flush the hardware's internal raster queue, ensuring the frame data in VRAM is consistent with all prior FIFO commands.

3.3 Drawing State

The device state set by these registers is not reset after issuing a draw command. For instance, a particular value of Draw.Vertex.Color4f need only be set once if multiple vertices should be emitted with the same color.

3.3.1 Draw.Vertex.Coord4f

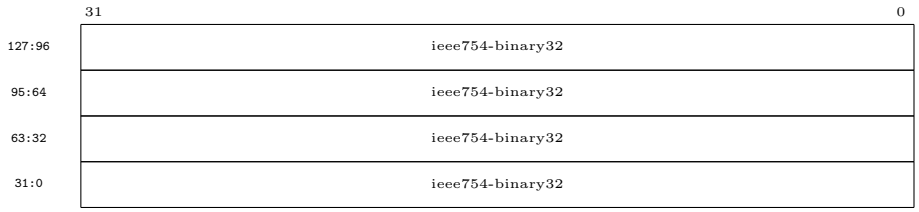
I/O Offset: 0x5000, Default: 0x00000000, Attributes: FIFO



Bits 31:0 X coordinate
Bits 63:32 Y coordinate
Bits 95:64 Z coordinate
Bits 127:96 W coordinate

3.3.2 Draw.Vertex.Color4f

I/O Offset: 0x5010, Default: 0x00000000, Attributes: FIFO



Bits 31:0 Color red component (range [0.0,1.0])
Bits 63:32 Color green component (range [0.0,1.0])
Bits 95:64 Color blue component (range [0.0,1.0])
Bits 127:96 Color alpha component (range [0.0,1.0])

3.3.3 Draw.Vertex.Transform16f

I/O Offset: 0x5080, Default: N/A, Attributes: FIFO

	127	96	64	32	0
1023:768	ieee754-binary32	ieee754-binary32	ieee754-binary32	ieee754-binary32	
767:512	ieee754-binary32	ieee754-binary32	ieee754-binary32	ieee754-binary32	
511:256	ieee754-binary32	ieee754-binary32	ieee754-binary32	ieee754-binary32	
255:0	ieee754-binary32	ieee754-binary32	ieee754-binary32	ieee754-binary32	

Bits 255:0 Transform X row
- 31:0 – $X[0]$
- 63:32 – $X[1]$
- 95:64 – $X[2]$
- 127:96 – $X[3]$

Bits 511:256 Transform Y row
- 31:0 – $Y[0]$
- 63:32 – $Y[1]$
- 95:64 – $Y[2]$
- 127:96 – $Y[3]$

Bits 767:512 Transform Z row
- 31:0 – $Z[0]$
- 63:32 – $Z[1]$
- 95:64 – $Z[2]$
- 127:96 – $Z[3]$

Bits 1023:768 Transform W row
- 31:0 – $W[0]$
- 63:32 – $W[1]$
- 95:64 – $W[2]$
- 127:96 – $W[3]$

3.3.4 Draw.Clear.Color4f

I/O Offset: 0x5100, Default: 0x00000000, Attributes: FIFO

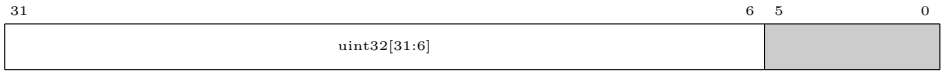
	31	0
127:96	ieee754-binary32	
95:64	ieee754-binary32	
63:32	ieee754-binary32	
31:0	ieee754-binary32	

Bits 31:0 Color red component (range [0.0,1.0])
Bits 63:32 Color green component (range [0.0,1.0])
Bits 95:64 Color blue component (range [0.0,1.0])
Bits 127:96 Color alpha component (range [0.0,1.0])

3.4 Stream Control

3.4.1 Stream.BufferA.Address

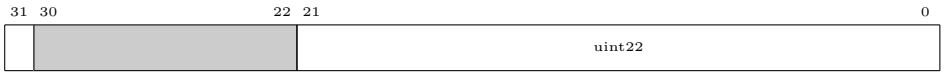
I/O Offset: 0x2000, Default: 0xFFFFFFFF, Attributes: FIFO



Bits 0:5 Ignored (treated as zero)
Bits 6:31 Physical source address for stream buffer A

3.4.2 Stream.BufferA.Config

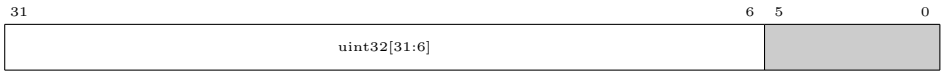
I/O Offset: 0x2008, Default: 0xFFFFFFFF, Attributes: FIFO



Bits 0:21 Buffer size in bytes
Bits 22:30 Reserved (write zero)
Bit 31 Source address space type
0 – Host physical address space (BM)
1 – Device VRAM

3.4.3 Stream.BufferB.Address

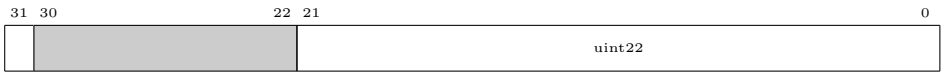
I/O Offset: 0x2004, Default: 0xFFFFFFFF, Attributes: FIFO



Bits 0:5 Ignored (treated as zero)
Bits 6:31 Physical source address for stream buffer A

3.4.4 Stream.BufferB.Config

I/O Offset: 0x200C, Default: 0xFFFFFFFF, Attributes: FIFO



Bits 0:21 Buffer size in bytes
Bits 22:30 Reserved (write zero)
Bit 31 Source address space type
0 – Host physical address space (BM)
1 – Device VRAM

4 Objects

The functionality of some device subsystems are abstracted to the driver and modified via state objects. There are two types of state objects in current devices: Encoders and Frames. The number of instances of these objects varies from model to model, but can be determined via ROM fields. They live in the same address space as other MMIO registers, with the base address being the offset to the first instance of the object. Other instances of that object may follow, depending on the number of physical susbystems present on the device. All current objects are 64-bytes and therefore instances are separated by a stride of 64-bytes.

4.1 Frames

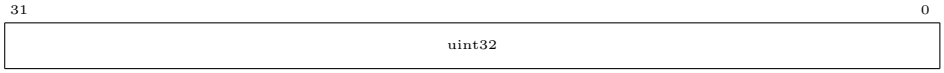
Frames represent buffers on the device consisting of pixel data. Frames are used as sources for encoders and targets for rasterization, blitting, and other drawing operations. The base address for frame objects is 8000h in MMIO space.

4.1.1 Limitations

Current implementations only support frames with 32-bit pixel depths, with the alpha channel being ignored. This corresponds to a pixel format of F888h on the device. Furthermore, render targets must be tightly packed (row pitch equal to the columns times bytes per pixel). Encoders do support variable row packing, making the functionality available for 2D-only clients. Additionally, row pitch and the start address must be a multiple of 16-bytes. Frames that do not fit entirely within VRAM may cause device memory or state corruption.

4.1.2 Frame.Columns

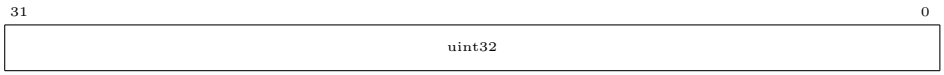
Instance Offset: 0x0000, Default: 0x00000000, Attributes: RW



Bits 0:31 Number of pixels per row in frame

4.1.3 Frame.Rows

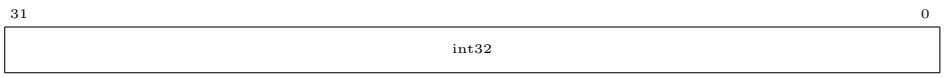
Instance Offset: 0x0004, Default: 0x00000000, Attributes: RW



Bits 0:31 Number of pixels per column in frame

4.1.4 Frame.RowPitch

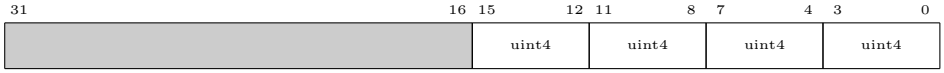
Instance Offset: 0x0008, Default: 0x00000000, Attributes: RW



Bits 0:31 Bytes between successive rows of pixels

4.1.5 Frame.PixelFormat

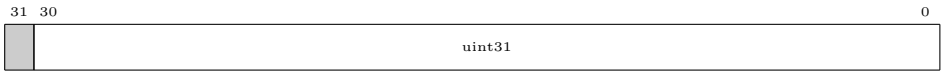
Instance Offset: 0x000C, Default: Hardware Dependent, Attributes: RO



Bits 0:3	Red channel bits per pixel
Bits 4:7	Blue channel bits per pixel
Bits 8:11	Green channel bits per pixel
Bits 12:15	Alpha channel bits per pixel
Bits 16:31	Reserved (Write zero)

4.1.6 Frame.StartAddress

Instance Offset: 0x0010, Default: 0x00000000, Attributes: RW



Bits 0:30	VRAM address of first pixel in frame buffer
Bit 31	Reserved (Write zero)

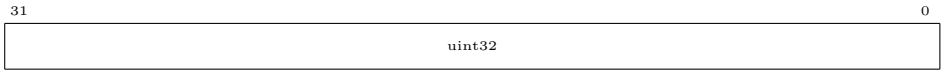
4.2 Encoders

Encoders are the abstraction for physical output ports on the device, e.g. DisplayPort or HDMI. Each is capable of sourcing a Frame for pixel data. It is best to think of encoders as a window into a frame. This window can be smaller than or equal to the size of the frame it views. If the encoder is configured for a size smaller than its source frame, the window can be panned around the various sections of the frame. The most common use cases for this are virtual desktops that span multiple monitors with a single render context. The base address for encoder objects is 9000h in MMIO space.

The size chosen for an encoder's window is also the resolution that the target display device will be set to. You must ensure that this resolution is supported natively by the device, or enable automatic display scaling support on the encoder (certain models only).

4.2.1 Encoder.Width

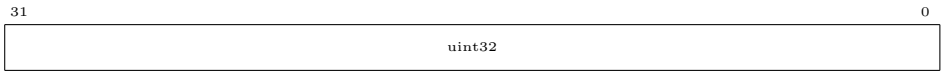
Instance Offset: 0x0000, Default: 0x00000000, Attributes: RW



Bits 0:31 Width of target mode, in pixels

4.2.2 Encoder.Height

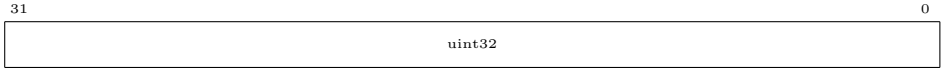
Instance Offset: 0x0004, Default: 0x00000000, Attributes: RW



Bits 0:31 Height of target mode, in pixels

4.2.3 Encoder.OffsetX

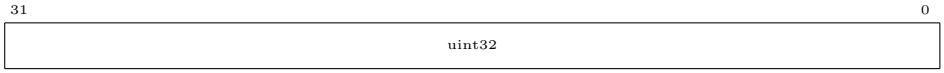
Instance Offset: 0x0008, Default: 0x00000000, Attributes: RW



Bits 0:31 Number of skipped columns from source frame, in pixels

4.2.4 Encoder.OffsetY

Instance Offset: 0x000C, Default: 0x00000000, Attributes: RW



Bits 0:31 Number of skipped rows from source frame, in pixels

4.2.5 Encoder.Frame

Instance Offset: 0x0010, Default: 0x00000000, Attributes: RW



Bits 0:5 Frame index used as scanout buffer
Bits 6:31 Reserved (write zero)

4.3 Setting a Display Mode

Due to the on-device firmware being able to understand the OOB messaging system used by modern display devices, configuration of the video unit is much easier than in previous devices. Timing values and mode detection are automatically controlled. It is recommended that a proper driver will take advantage of these systems. However, due to legacy compatability constraints, certain modes are guaranteed to be supported by the Kyouko device on all display devices. These modes mirror a subset of the standard VGA modes.

4.4 Example Configuration for 1024x768 32bpp

Register	**Required Value**
Config.Acceleration	0x00000000
Frame[0].Columns	1024
Frame[0].Rows	768
Frame[0].RowPitch	4096
Frame[0].PixelFormat	0x0000F888
Frame[0].StartAddress	0x00000000
Encoder[0].Width	1024
Encoder[0].Height	768
Encoder[0].OffsetX	0
Encoder[0].OffsetY	0
Encoder[0].Frame	0

4.5 Mode Changes

In order to change the mode or resolution of the device, the encoder and linked frame configuration must be refreshed. This involves sending the device a Config.ModeSet message via any MMIO write to the register. The device will load the settings written by the driver and configure its physical encoder for the requested mode. During this time, the FIFO will continue to be processed, but commands related to drawing may not be properly issued. In order to ensure that no processing of drawing commands occurs during this time, the driver should wait for the first vertical refresh interrupt to be issued, queue a Raster.Flush and wait for the FIFO to empty, or wait a minimum of 5 seconds.

As the chip is not capable of recovering from errors on its own in most cases, it is important that an unsupported mode not be configured during the device's mode set sequence.

5 Rasterization Sequences

The device maintains an internal state necessary to render most graphics primitives on its own when given a stream of vertex state information. A rasterization sequence begins when a non-zero Raster.Primitive write is processed by the device, which indicates how the device should interpret future Raster.Emit writes. Any number of Raster.Emit commands may be issued during the sequence. Upon processing Raster.Emit, the device will take snapshot of the state of all Draw state registers and queue them into the raster stream. Once enough vertices have been processed in this manner by the device for good performance, the primitive will be rasterized to the currently configured frame. In order to end the sequence and issue non-drawing commands again, the driver must write a zero to the Raster.Primitive register.

The state of the Draw register space at the time a Raster.Emit is issued determines how that particular vertex is drawn. The Draw registers may be updated any number of times, including none, between different instances of Raster.Emit. All writes to Draw registers pass through unmodified to the device's raster stream with the exception of Draw.Vertex.Coord4f. The Draw.Vertex.Coord4f register will be multiplied by the matrix in Draw.Vertex.Transform16f before leaving the command queue. Thus, issuing Raster.Emit prior to a full setup of Draw.Vertex.Transform16 may lead to unexpected results.