VINO Design Specification 099-8937-001

Linda Bender
Chris Kogelnik
Mark Troeller

Digital Sight and Sound

Version 2.0

Table of Contents

1.0 Introduction....1
- 1.1 General Overview....1
- 1.2 Features....1
  - 1.2.1 Block Diagram....2

2.0 System Bus Interface....3
- 2.1 Address Map....3
- 2.2 Programmed I/O....5
- 2.3 DMA....5
  - 2.3.1 DMA Operation....5
  - 2.3.2 DMA Register Summary....10
- 2.4 Arbiter....11
- 2.5 Interrupt Handling....11
- 2.6 Reset....12
- 2.7 I^2C Bus....12
  - 2.7.1 I^2C Basic Protocol....12
  - 2.7.2 I^2C Implementation....13

3.0 Video Inputs....13
- 3.1 YUV Bus....13
- 3.2 SGI Digital Video Bus....13
  - 3.2.1 Data Format....14
  - 3.2.2 Control Codes Embedded in Data Format....14
  - 3.2.4 SGI Digital Video Timing Information....15
  - 3.2.3 Error Protection Bits....15

4.0 Video Processing....16
- 4.1 Data Formats....16
- 4.2 Filtering/Decimation....16
- 4.3 Dithering....17
- 4.4 Clipping....18
- 4.5 Color Space Conversion....18
- 4.6 Synchronization Control....19
- 4.7 Frame/Field Rate Control....19
- 4.8 Alpha Value....19

5.0 Software Issues....20
- 5.1 Data Formats....20
- 5.2 Graphics Synchronization....20
- 5.3 Frame Assembly....20

6.0 Register Descriptions....20

7.0 Signal Descriptions and Pinout....25
- 7.1 Pin Assignments....27
- 7.2 Test Modes....29

8.0 Packaging....29

9.0 DC Characteristics 31
10.0 AC Characteristics 31
10.1 I/O Timing Table 31
10.2 PLL Characteristics 32
11.0 Fault Coverage 33
12.0 Timing 33
13.0 Indy Related Issues 40
13.1 Application Block Diagram 40
13.2 Video Input Notes 40
14.0 Bugs 41

List of Tables

TABLE 1. Register Address Map 4
TABLE 2. Digital Video Control Byte Format 14
TABLE 3. Hamming Code Words for Error Protection Scheme to work 15
TABLE 4. ID and Revision Register 20
TABLE 5. Control Register 21
TABLE 6. Interrupt and Status Register 22
TABLE 7. Alpha Register 22
TABLE 8. Clipping Start Register 22
TABLE 9. Clipping End Register 22
TABLE 10. Frame Rate Register 22
TABLE 11. Field Counter Register 23
TABLE 12. ChX Line Size Register 23
TABLE 13. ChX Line Count Register 23
TABLE 14. ChX Page Index Register 23
TABLE 15. ChX Pointer to Next Four Descriptors Register 23
TABLE 16. ChX Pointer to Start of Descriptor Table Register 23
TABLE 17. ChX DescriptorX Data Register 24
TABLE 18. ChX Fifo Threshold Compare Value Register 24
TABLE 19. ChX Fifo GIO Pointer Register 24
TABLE 20. ChX Fifo Video Pointer Register 24
TABLE 21. I^2C Control and Status Register 24
TABLE 22. I^2C Data Register 25
TABLE 23. Pin Description 26
TABLE 24. 208 PQFP Pin Assignments 27
TABLE 25. Test Mode Pin Functions 29
TABLE 26. DC Characteristics (T_j =-20 to 75^o C) 31
TABLE 27. AC Characteristics 31
TABLE 28. DMSD Selection of Composite and S-Video Analog Inputs 41

List of Figures

FIGURE 1. Descriptor Cache Management....................7
FIGURE 2. *Page_index* Roll Over Flow Diagram....................7
FIGURE 3. Invalidate Descriptors Flow Diagram....................8
FIGURE 4. *Line_count* Flow Diagram....................8
FIGURE 5. DMA Control Flow Diagram....................9
FIGURE 6. SGI Digital Video Interface Transmitted Clock Waveform....................15
FIGURE 7. Logical pin diagram of VINO....................25
FIGURE 8. 208 Plastic Quad Flat Pack Package Dimensions....................30
FIGURE 9. VINO PLL Network....................33
FIGURE 10. MC read from VINO register....................34
FIGURE 11. Write MC to Vino....................35
FIGURE 12. VINO Descriptor Fetch....................36
FIGURE 13. VINO fifo empty....................37
FIGURE 14. DMA with Page Index Roll....................38
FIGURE 15. DMA with End of Field Interrupt....................39
FIGURE 16. VINO System Application Block Diagram....................40

Revision History

8.12.93 Added Logical Pin Diagram of Vino, Added SGI Digital Port Description, Enhanced Interrupt description, YUV Data Format corrected, Data Formats extended to 64 bit word, added CSC accuracy, added Page Index behavior.

7.16.93 Added DC and AC Timing, Package Drawing Fig, INDY System Application Fig, PLL Description. and Video Input Notes Section

7.15.93 Added Pin List, Signal Description, Test Pin Mode Table, Updated Frame Rate Control

1.0 Introduction

The purpose of this chip is to provide an inexpensive video input into the system enabling various applications such as video conferencing, video capture and TV in a window. It is a video input only device accepting NTSC, PAL, and SGI's D1 formats and has some video processing functions. On the system side the chip interfaces to the GIO64 bus, moving data directly to system memory, rather than a graphics back end. The video stream is now a general system resource manipulatable and redirectable by software for various applications.

1.1 General Overview

The chip is partitioned into a few subsections, which are the system interface, video processing, Philips 7191 and I^2C interface and the D1 interface. The chip receives a digital video stream from the Philips video chip set, which does the analog to digital conversion and standard decoding, or the D1 interface. The video data is then processed and DMA'd into main system memory. System software is then responsible for moving the image from memory to the display. This architecture has two major benefits: it is not constrained to a specific graphics back-end, making it portable to other systems and the video stream is a general system resource leveraging main memory for storage rather than an additional frame buffer.

The chip has two independent DMA channels. Each channel has a number of configurable options: video image clipping, decimation with filtering for a number of sizes, YUV to RGB color space conversion, dithering and frame rate control. Each of these options is independently selectable, controlled by bits in the register file.

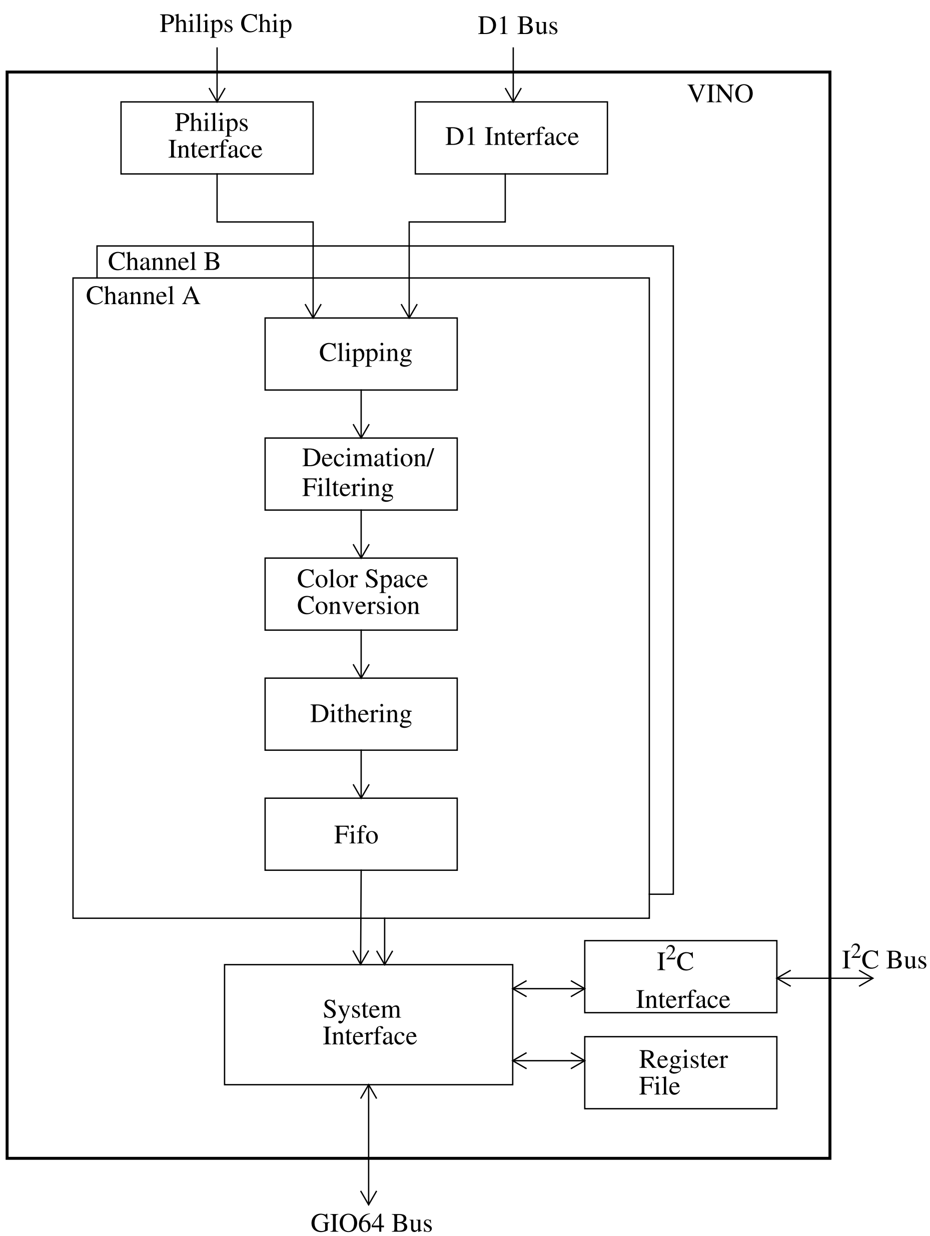
On the system side, the interface is to the GIO64 bus. Non-pipelined DMA and doubleword Programmed IO are supported. The register file is accessible to the GIO interface, which controls the functionality. System software maintains a descriptor table per channel in main memory describing the video buffer space. Each channel maintains its DMA pointers by traversing its descriptor table. The hardware is able to interlace fields or pack fields into main memory.

1.2 Features

- Selectable full size video input from NTSC, PAL, or SGI's D1
- Two independent DMA channels
- GIO64 bus master
- DMA striding to allow a frame to be assembled in system memory
- Horizontal and vertical decimation with filtering of sizes 1/2, 1/3 and1/4. "Field to Frame" for 1/2
- General purpose clipping

- YUV to RGB color space conversion
- 24-bit to 8-bit RGB dithering
- Frame/field rate control
- 4:2:2 YUV, 32-bit RGBA, 8-bit RGB and 8-bit Monochrome pixels to memory
- I^2C interface

1.2.1 Block Diagram



2.0 System Bus Interface

VINO is a 64 bit real-time device that supports the non-pipelined GIO64 bus protocol. It is a bus slave for all programmed I/O transfers and a bus master for all DMA transfers. VINO can not be preempted from the GIO64 bus when it is bus master.

In Indy, VINO interfaces to the GIO64 bus using the EISA_Req_n, EISA_Gnt_n and EISA_AS_n signals. The EISA_Req_n is the highest priority request of the GIO64 devices. Real-time devices are serviced in the order given below:

1) EISA (video on Indy)
2) HPC3
3) Refresh (maximum time on bus is 800ns)
4) GIO64 Expansion Slot 0 (if configured as real time, maximum time on bus is 5us).
5) GIO64 Expansion Slot 1 (if configured as real time, maximum time on bus is 5us).

VINO requests the GIO64 bus whenever an internal fifo needs to be serviced or descriptors need to be fetched. VINO stays on the GIO64 bus until both fifos have been drained and descriptors have been fetched. VINO does not have a hardware timer to limit the amount of time it is on the GIO64 bus or the frequency of requests.

In a lightly loaded system, a descriptor fetch cycle (total time vino grant is low) was measured as 780 ns. A DMA cycle with the fifo threshold set to half-way (64 double-words) was measured as 3 us. The DMA cycle consisted of 17 cycles of overhead and 83 cycles of double-word transfers. Based on these measured values, VINO could theoretically be on the bus for 10.27 us (10.27 us is the time to drain both fifos and perform two descriptor fetch cycles). Considering bus usage, will VINO ever need to drain both fifos and perform two descriptor fetch cycles? A simulation model of the fifos in Indy was developed to answer this question. The model made the following assumptions:

GIO bus clock rate 33 MHz
Video Fifo1 Fill Rate 60.00 MBytes/sec
Video Fifo2 Fill Rate 30.00 MBytes/sec
HPC Ethernet Fill Rate 1.25 MBytes/sec
HPC SCSI Fill Rate 20.00 MBytes/sec
HPC PBUS Fill Rate 6.00 MBytes/sec
Video Fifo1 Setpoint 512 Bytes
Video Fifo2 Setpoint 512 Bytes
MC Buffer Content 3488 Bytes

This represents full motion RGBA PAL television (15MHz peak pixel rates) coming into Video Fifo1 and YUV full motion PAL television coming into Video Fifo2. These two streams are then sent across the GIO bus into system DRAM. The MC Buffer Content is the water mark of the RGBA PAL television buffer that is being sent to Graphics for display of TV in a window. After 1 million cycles of the GIO bus approximately 0.03 seconds of time has elapsed during which 1.8 million bytes were transferred from VINO Channel 1 video into system DRAM and 1.8 million -3488 of those bytes have reached the display window in the graphics subsystem.

The longest time that VINO was on the bus was 6.24 usec.

2.1 Address Map

The address space starts at 0x00080000. The following figure shows the accessible registers in the address space. All accesses are on a double-word (64 bit) boundary regardless of the actual data size of the register.

For 32 bit access use 4/8 ending address for 0/8 addresses respectively.

TABLE 1. Register Address Map

GIO Addr	Function	Read/Write
0008 0130	Channel B FIFO Write Pointer (video)	Read
0008 0128	ChB FIFO Read Pointer (GIO)	Read
0008 0120	ChB FIFO Threshold	Read/Write
0008 0118	ChB Descriptor 3 Data	Read/Write
0008 0110	ChB Descriptor 2 Data	Read/Write
0008 0108	ChB Descriptor 1 Data	Read/Write
0008 0100	ChB Descriptor 0 Data	Read/Write
0008 00F8	ChB Pointer to Start of Descriptor Table	Read/Write
0008 00F0	ChB Pointer to Next Four Descriptors	Read/Write
0008 00E8	ChB Page Index	Read/Write
0008 00E0	ChB Line Count	Read/Write
0008 00D8	ChB Line Size	Read/Write
0008 00D0	ChB Field Counter	Read
0008 00C8	ChB Frame Rate	Read/Write
0008 00C0	ChB Clipping End	Read/Write
0008 00B8	ChB Clipping Start	Read/Write
0008 00B0	ChB Alpha	Read/Write
0008 00A8	Channel A FIFO Write Pointer (video)	Read
0008 00A0	ChA FIFO Read Pointer (GIO)	Read
0008 0098	ChA FIFO Threshold	Read/Write
0008 0090	ChA Descriptor 3 Data	Read/Write
0008 0088	ChA Descriptor 2 Data	Read/Write
0008 0080	ChA Descriptor 1 Data	Read/Write
0008 0078	ChA Descriptor 0 Data	Read/Write
0008 0070	ChA Pointer to Start of Descriptor Table	Read/Write
0008 0068	ChA Pointer to Next Four Descriptors	Read/Write
0008 0060	ChA Page Index	Read/Write
0008 0058	ChA Line Count	Read/Write
0008 0050	ChA Line Size	Read/Write
0008 0048	ChA Field Counter	Read
0008 0040	ChA Frame Rate	Read/Write
0008 0038	ChA Clipping End	Read/Write
0008 0030	ChA Clipping Start	Read/Write
0008 0028	ChA Alpha	Read/Write
0008 0020	I^2C Data	Read/Write
0008 0018	I^2C Control	Read/Write

TABLE 1. Register Address Map

GIO Addr	Function	Read/Write
0008 0010	Interrupt Status	Read/Write
0008 0008	Control	Read/Write
0008 0000	Rev/ID	Read

2.2 Programmed I/O

A PI/O operation is a single 64-bit doubleword register access; block transfers are not supported. PI/O operations are always doubleword aligned. All unused bits in a register are read as zero. The GIO64 *byte_count* field is ignored during a PI/O operation, so eight bytes are always transferred.

2.3 DMA

VINO is bus master during DMA operations. The DMA logic supports two modes:

1) field capture
2) frame assembly in memory (interleaving)

During field capture, VINO packs the first, then second field and so on into the buffer space described by the descriptor table. A field always starts on a 4K-byte address, so more than likely there will be gaps of memory between consecutive fields.

During frame assembly, frames are assembled into the buffer space described by the descriptor table. The first line of the first field is placed in memory, a line of memory is skipped, and then the second line is placed in memory. This process of placing a line and skipping a line continues until the entire first field is in memory. The same process is used for the second field, starting at the top of the buffer and filling in the lines of memory skipped during the first field. The procedure is repeated for the next frame, starting at the address described by the next quad descriptor group (explained in detail in the next section).

2.3.1 DMA Operation

This section describes the DMA operation in greater detail. A detailed description of a field capture operation is presented first, followed by a description of frame assembly which builds upon the field capture operation. Flow diagrams of the DMA operation are presented at the end of the description.

In order to keep DMA running, a continuous supply of descriptors must be available. A descriptor contains the physical address of a 4K-byte memory region or the physical address of the next descriptor. DMA logic transfers data to the memory region addressed by a descriptor. In addition to the physical address, a descriptor contains a valid, jump and stop bit. A supply of descriptors is maintained using a cache of four descriptors. Descriptor zero is the active descriptor and is used as the upper bits of the DMA address. *Page_index* forms the lower bits of the DMA address. *Page_index* is incremented by 8 bytes after each doubleword transfer, maintaining a doubleword index into the current 4K-byte memory region. When 4K-bytes of data has been transferred, *page_index* rolls over, causing each descriptor in the cache to shift up one location, and the bottom entry to be invalidated. When an invalid descriptor is shifted into location zero, a new set of descriptors is fetched from the address in *next_descriptor*. If a valid descriptor with the jump bit set is shifted into descriptor location zero, a new set of descriptors is fetched from the address in the jump descriptor. *Next_descriptor* is incremented by 16 bytes after every descriptor fetch operation. If a valid descriptor with the stop bit set is shifted into descriptor location zero, DMA operation is stopped immediately and an interrupt is generated. The DMA initialization routine must be run (set page_index to zero, write next_descriptor etc.), and this bit must be cleared before DMA can continue.

To begin a field capture operation, *page_index* must be set to zero. The address of the start of the descriptor table must then be written into *next_descriptor*. As soon as *next_descriptor* is written, the descriptor cache is invalidated causing a descriptor fetch to occur. It is important that *page_index* precede *next_descriptor*. As stated in the previous paragraph, descriptors shift when *page_index* rolls over, defined as *page_index* going from a non-zero value to zero. Depending on initial conditions, setting *page_index* to zero can cause a *page_index* roll over and hence a shift in descriptors. If *next_descriptor* is written first, descriptors are fetched, and a subsequent write to *page_index* might cause the descriptors to shift. After these registers have been initialized, DMA can be enabled.

Data is not written to the FIFO until DMA is enabled. When data reaches the threshold level in the FIFO (this can only occur if DMA is enabled), the DMA logic makes a request to the internal arbiter for the GIO64 bus (arbitration is explained in Section 2.4 on page 11). A grant from the arbiter begins the actual data transfer. The address for the DMA transfer is generated using bits 11:3 of the *page_index* register, bits 29:12 of the active descriptor, and with bits 2:0 and 31:30 set to 0. The GIO *byte_count* field is set to the maximum value. The data transfer continues until the FIFO is empty or *page_index* rolls over. If *page_index* rolls over, the data stream is terminated, but the GIO64 bus is not released. The *page_index* roll over causes the descriptors to shift in the descriptor cache, and the address from the new active descriptor is used to generate a new address/byte count data stream. A FIFO empty condition completely terminates the DMA operation. The DMA logic de-asserts it request line, and the internal arbiter responds by de-asserting the device's grant line and the GIO64 request line, releasing the GIO64 bus. The process is started over when data once again reaches the threshold level in the FIFO.

In order to ensure that the last part of a field is not sitting in the FIFO during vertical retrace, a DMA operation is initiated as soon as an end-of-field condition occurs. An end-of-field condition indicates that the last pixel in a field has been placed in the FIFO. The DMA operation proceeds as described above until the FIFO is empty. A FIFO empty condition in conjunction with an end-of-field signal indicates that the last pixel of the field has been transferred. After the last pixel of a field is transferred, an end-of-field interrupt is generated, and the descriptors are shifted up one location so that the next field starts on a 4K-byte address.

Frame assembly is more complicated than field capture, but descriptor cache maintenance as well as the basic DMA operation is the same. To begin a frame assembly operation, *page_index* and *line_count* must be set to zero, *line_size* must be set with the appropriate value (discussed in Section 2.3.2 on page 10), and *start_descriptor* and then *next_descriptor* must be set to the starting address of the descriptor table. Writing to *next_descriptor* causes a descriptor fetch to occur. After these registers have been initialized, DMA can be enabled.

The actual DMA transfer is not started until data in the FIFO reaches the threshold level. The address and byte count are generated in the same manner described above. Data transfer continues until the FIFO is empty, *page_index* rolls over, or *line_count* equals *line_size* (a line of data has been transferred). FIFO empty and *page_index* conditions are explained above. After a line of data has been transferred, the current address, byte count, data stream is terminated, the GIO64 bus mastership is retained, and *page_index* is incremented by *line_size* + 1. A new address (produced from the incremented *page_index*), byte count, data stream is generated. If incrementing *page_index* causes a *page_index* roll over, descriptors are shifted up one location, and the lower bits of the new address come from the incremented *page_index* while the upper address bits come from the new active descriptor. This procedure causes a line of memory to be skipped between every line in the first field.

Just like the field capture operation, a DMA operation is initiated as soon as an end-of-field condition occurs. After the last pixel of the first field is transferred across the bus, *start_descriptor* is loaded into *next_descriptor*, causing a descriptor fetch and resetting DMA to the top of the frame. *Page_index* is set to *line_size* + 1, so that the first line of the second field is placed in the line of memory skipped during the first field. Using the lines from the second field, DMA continues to fill in the lines of memory previously skipped. This continues until the next end-of-field condition. After the last pixel in the second field has been transferred across the bus, *next_descriptor* is loaded into *start_descriptor*, the current descriptors are invalidated (skipping any unused descriptors), and a new set of descriptors are fetched. The next frame starts on the 4K-byte address described by the new active descriptor. More than likely, there will be gaps of memory between consecutive frames. *Start_descriptor* is loaded with *next_descriptor* to ensure that the first descriptor in a frame is located at a quad-word address. *Next_descriptor* and *start_descriptor* must always be quad-word aligned, because the descriptor logic can only fetch from a quad-word address.

FIGURE 1. Descriptor Cache Management

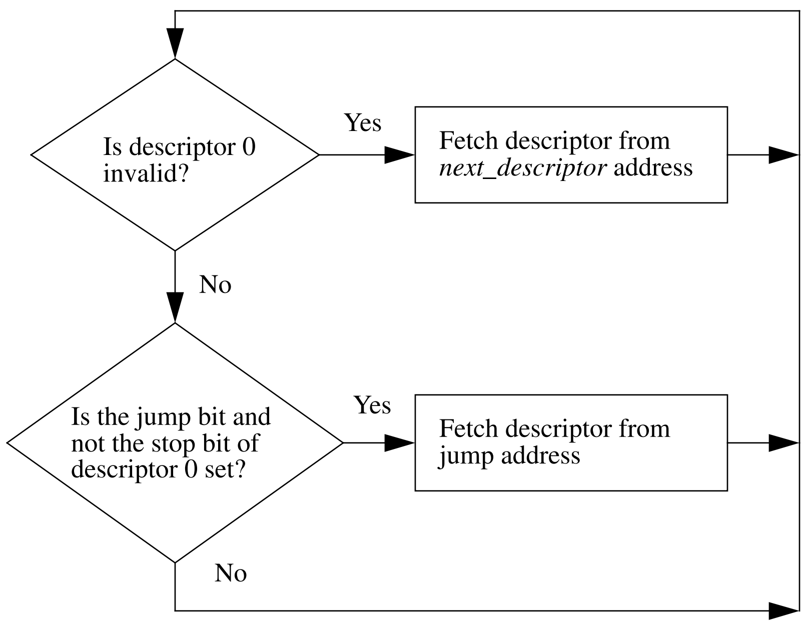


FIGURE 2. ***Page_index*** **Roll Over Flow Diagram**

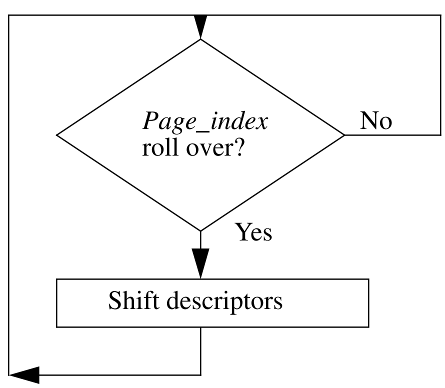


FIGURE 3. Invalidate Descriptors Flow Diagram

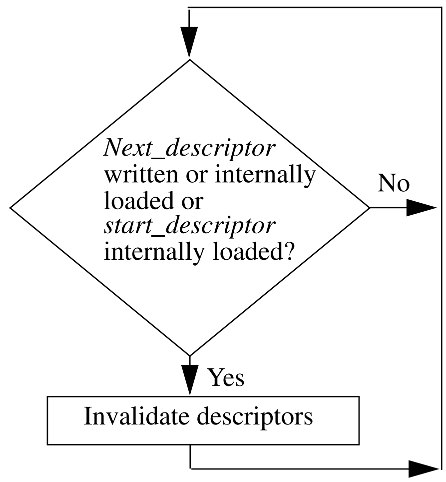


FIGURE 4. *Line_count* Flow Diagram

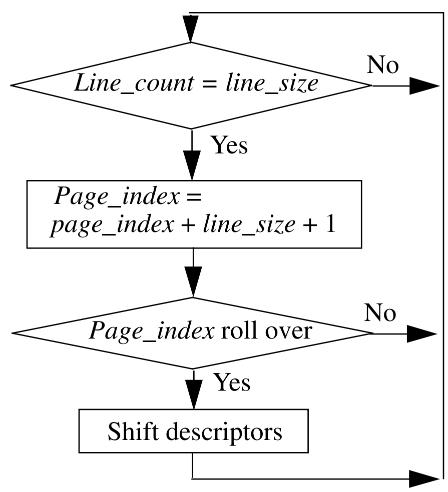
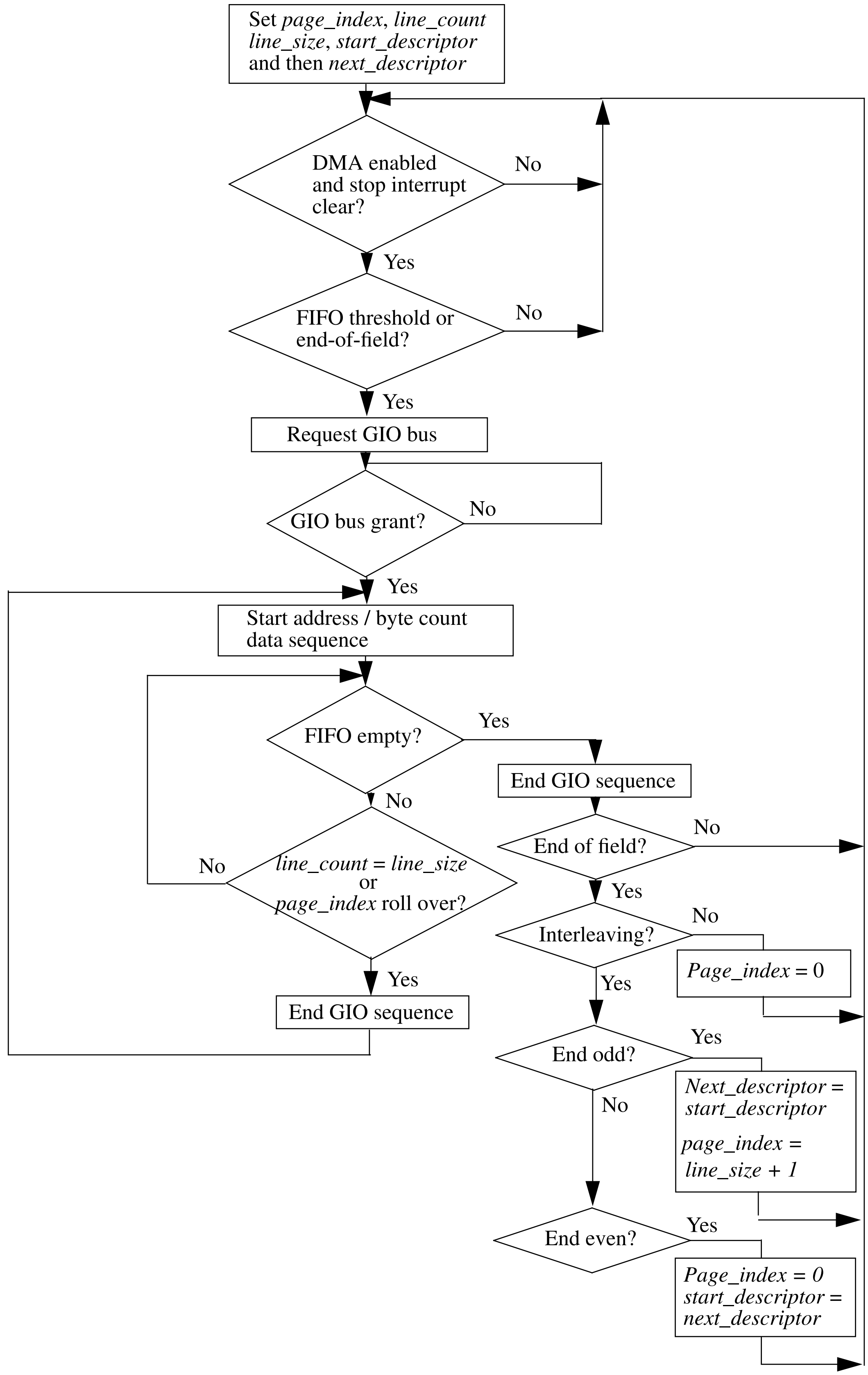


FIGURE 5. DMA Control Flow Diagram



2.3.2 DMA Register Summary

Only the register bits pertinent to the DMA description are discussed. A detailed description of the registers can be found in Section 6.0 on page 20. Enabled bits are set to ‘1’ and disabled bits to ‘0’.

Control Register - Each channel has the following control enable bits. The bits in this register are set and cleared by software.

> Endianess - This bit is common to both channels (1 - little, 0 - big).
>
> Channel enable - Setting this bit enables the DMA channel. This bit must be disabled when setting up related DMA registers. When this bit is disabled, the FIFO pointers are held in reset (0).
>
> Interleave - When enabled, fields are assembled into frames in system memory.

Interrupt Register - The bits in this register are set (1) by hardware and cleared (0) by software.

> End-of-field - Set after the last byte of a field has been transferred across the bus.
>
> FIFO overflow - Set when the FIFO has overflowed. DMA operation is stopped immediately. The DMA bit must be disabled, resetting the FIFO pointers, the DMA initialization routine must be run (set page_index to zero, write next_descriptor etc.), and this bit must be cleared before DMA can continue.
>
> End of descriptor table - Set when the stop bit of the active descriptor is set. DMA is stopped immediately. The DMA initialization routine must be run (set page_index to zero, write next_descriptor etc.), and this bit must be cleared before DMA can continue.

Line_Size Register - Lines must be a multiple of 8 bytes. This requirement makes all DMA transfers doubleword aligned and eliminates the need for alignment hardware in the FIFO. The appropriate value for *line_size* is calculated by taking the total number of bytes in a line, and then subtracting 8 bytes. As an example, *line_size* is 20 hex for a 40 byte line.

Line_count Register - Contains current byte in line count. After every transfer this register is incremented by 8 bytes. When *line_count* = *line_size*, *line_count* is reset to zero.

Page_index Register - The nine bits in this register index into a doubleword in the current page buffer, and form bits 11:3 of a DMA address with the active descriptor providing the high bits. Bits 2:0 of the DMA address are always zero, since transfers are always doubleword aligned.

Next_Descriptor Register - Contains the physical quadword address of the next four descriptors. This register is incremented by sixteen after each descriptor fetch.

Start_Descriptor Register - Contains the physical quadword address of the first descriptor in a frame. It should be initialized to the same address as *next_descriptor.*

Descriptor Cache - The cache has four 33-bit registers containing the current four descriptors. All locations in the descriptor cache can be read or written. The valid bit can only be read. Writing to a cache location produces a valid descriptor. Descriptor zero is the active descriptor. When page_index rolls over, all descriptors are shifted up one location, and the bottom descriptor is invalidated. A new set of descriptors are fetched from the address in *next_descriptor* when an invalid descriptor is shifted into location zero. If a valid descriptor with the jump bit set is shifted into location zero, a new set of descriptors are fetched using the address from the jump descriptor. If a valid descriptor with the stop bit set is shifted into descriptor location zero, DMA operation is stopped immediately and an interrupt is generated. The DMA initialization routine must be run (set page_index to zero, write next_descriptor etc.), and this bit must be cleared before DMA can continue. All descriptors are invalidated when *next_descriptor* is written, causing a new set of descriptors to be fetched.

Each descriptor contains a physical address of the start of a 4K-byte page of system memory or the address of the next descriptor. Descriptors are packed into a descriptor table located in memory, and the start of the table must be quad word aligned. The last descriptor in a table must have the stop bit or jump bit set to prevent runaway DMA's. The jump bit and stop bit are "OR'd" with the physical address to produce a descriptor. These bits, 31 and 30, are driven low during the GIO64 address cycle to produce the correct physical address. The figure below shows the descriptor format:

32	31	30	29 — 0
valid	stop	jump	pointer

Stop - When a stop bit is detected, DMA operation is stopped immediately, and an interrupt is generated. The DMA initialization routine must be run (set page_index to zero, write next_descriptor etc.), and this bit must be cleared before DMA can continue.

Jump - When this bit is set, the pointer field contains the quad word address of the next descriptor. Jumps can only be to quad word address. Restricting descriptors to quad word addresses avoids complicated alignment logic in the descriptor fetch circuitry.

Pointer to a Page Buffer - If neither stop nor jump is set, then the pointer field is the address of a 4K-byte page of buffer space.

DMA FIFO - Each channel FIFO is 1Kbytes configured as 128 x 64 bits. The read pointer (GIO) and write pointer (video) are read only. The pointers are held in reset (zero) when DMA is disabled. The threshold value is programmable.

2.4 Arbiter

There are four devices on VINO that require the use of the GIO64 bus: two DMA channels and two descriptor fetch units. The arbiter receives requests from the devices, and provides grants based on the fixed priority shown below. Upon receiving a request, the arbiter requests the GIO64 bus. When the arbiter receives the GIO64 bus grant, it provides a grant to the device with the highest priority. When the highest priority device is done with the bus, it de-asserts its request. The arbiter responds by de-asserting the device's grant, and provides a grant to the next highest requesting device. This process continues until all requesting devices have used the bus. The arbiter then releases the GIO64 bus by de-asserting the GIO64 bus request.

1) channel A descriptor fetch
2) channel A DMA
3) channel B descriptor fetch
4) channel B DMA

2.5 Interrupt Handling

The following interrupts cause the GIO_INT_N signal to become active:

1. DMA channel A end of descriptor table
2. DMA channel A FIFO overflow
3. DMA channel A end-of-field
4. DMA channel B end of descriptor table
5. DMA channel B FIFO overflow
6. DMA channel B end-of-field

All interrupts are level sensitive. This means that the condition that caused the interrupt must be cleared before the interrupt can be cleared. The GIO_INT_N signal is active as long as there are active interrupts.

If an *end of descriptor table* interrupt occurs, DMA operation stops immediately. The DMA initialization routine must be run (set page_index to zero, write next_descriptor etc.), and this bit must be cleared before DMA can continue.

If a *FIFO overflow* interrupt occurs, DMA operation is stopped immediately. The DMA bit must be disabled, resetting the FIFO pointers, the DMA initialization routine must be run (set page_index to zero, write next_descriptor etc.), and this bit must be cleared before DMA can continue.

If an *end-of-field* interrupt occurs, DMA continues. This interrupt does not have a condition to clear, so once received, this interrupt can be cleared immediately.

The *end-of-field* interrupt will only occur on a DMA channel that has been activated by setting the chA/B DMA enable bit in the Control Register. The end-of-field interrupt will not occur on fields that have been turned off in the Frame Rate Register.

To ensure proper operation for two-channel DMA interrupt handling it is recommended that when servicing interrupts for a given channel, the other channel's interrupt be disabled (using the control register bit) when resetting the interrupt for the channel that has interrupted. This will ensure that the edge sensitive IOC chip in the INDY system will see the single physical interrupt line from VINO de-assert when an interrupt condition is cleared in VINO. There is a pathological condition that could exist which would have the second DMA channel produce and *end-of-field* interrupt just after the VINO chip has interrupted for the first DMA channel and the software has already polled the VINO to find which channel is interrupting and is about to reset that first DMA channel's interrupt. If the second channel interrupts at this instance, then the physical interrupt attached to the IOC chip will never de-assert and so the second interrupt will be ignored as will all subsequent interrupts which could cause the interrupt handler to never be invoked again and cause unknown system behavior.

2.6 Reset

VINO reset comes from the Philips 7197 clock generation chip. The output reset signal from the clock chip is held low for 200 ms (determined by a capacitor) after the input reset is de-asserted. During this extended reset period, the video clock from the Philips 7197 is running. A valid video clock during reset is needed to reset the VINO video subsection. The Philips 7197 input reset is controlled by a register bit in the IOC chip (refer to the IOC specification for a complete description). This bit is set low to reset VINO, and high for normal operation. This bit is cleared on reset.

2.7 I^2C Bus

VINO provides an interface between the GIO64 bus and the I^2C bus used by the Philips chip set. The I^2C bus is a very slow serial bidirectional bus (64KHz for VINO). Data must be converted back and forth between serial and parallel data formats depending on the data direction. VINO is always a master on the I^2C bus. The Philips 1992 Desktop Video Data Handbook provides detailed timing diagrams and information on the I^2C bus.

2.7.1 I^2C Basic Protocol

The bus consists of two wires: serial clock (SCL) and serial data (SDA). Both wires are open-collector with a resistive pull-up creating a wired-AND bus. Both lines float high when the bus is idle. For data transfer, SDA is stable when SCL is high and SDA transits when SCL is low. The sequence is started and stopped when these constraints are not met. A start signal is generated by pulling SDA low while SCL is high, and a stop signal is generated by floating SDA high while SCL is high. All data is transferred as 8-bit bytes with the MSB first. Each byte has to be followed by an active low acknowledge sent by the receiver. The number of bytes that can be transmitted per transfer (between start and stop conditions) is unrestricted. A receiver can hold SCL low to force the transmitter into a wait state. Data transfer continues when the receiver releases SCL.

2.7.2 I^{2}C Implementation

The I^{2}C circuitry contains a *I^{2}C Control* register and a *I^{2}C Data* register. Reading/Writing from/to the *I^{2}C Data* register initiates a read/write cycle. The *I^{2}C Control* register provides software with a means of controlling the I^{2}C bus, and I^{2}C status information. Detailed register information is provided in Section 6.0 on page 20.

There is no buffering of data between the GIO64 bus and the I^{2}C bus, so only single byte transfers are supported. Software must poll the *I^{2}C Control* register to see if a transfer has been completed before proceeding to the next transfer.

3.0 Video Inputs

The VINO ASIC has two video input ports. One port accepts a 16 bit YUV bus and timing information that connects without glue logic directly to a Philips 7191A/B DMSD (Digital Multistandard Decoder). The second input is compatible with the SGI Digital Video Interface and is an 8wire Data 1 wire Clock Interface that is compatible with *Indy Cam* as well as the *Galileo* product for the *Indigo^2* and the *Indy Video* product for *Indy* as a means of passing digital video into the VINO chip and onto the GIO bus.

3.1 YUV Bus

The timing and sequence information for this bus is best described in the Philips "Desktop Video Data Handbook 1993". It can also be found quite nicely documented in the Kaleidoscope engineering documentation. It should be noted that the information to detect field 1 vs. field 2 in this interface is accomplished by the level of the HREF pin on the 7191 at the de-assertion of VS. For field 1 HREF is a logical 1 for field 2 HREF is a logical 0. This has the added complication of confusing the line count circuit inside of VINO that is used in conjunction with the ClipY registers. The line counter is unable to set the clip start at exactly the first visible line that the DMSD sends as the counter is not incremented until the next HREF edge.

It is also interesting to note that the 1993 Philips book shows the relationship of the actual transmitted Composite video line numbering with the VS and HREF synchronization signals that are provided to VINO by the 7191.

For 625 50 Hz video (PAL)VS de-asserts at line 7 of the Composite Signal. VINO will not be at line 1 until the Composite Signal hits line 8. For field2 line 320 becomes line 1 of vino's field 2 counter.

For 525 60 Hz video (NTSC) VINO calls line 1 line 8 of the Composite Signal in field 1 and line 270 becomes VINO line 1 of field 2. This information is useful when calculating Y clip start and Y clip end of a visible NTSC or PAL signal for capture.

3.2 SGI Digital Video Bus

A high speed port that can run up to 30 MHz is provided on the VINO chip as a means of bringing in Digital Data that has the logical format of D1 video. This is a single ended TTL connection that is used by Indy Cam to bring in 4:2:2 digital video and synchronization information. It can also be used as a means of bringing Digital Video data in from the Indy Video add-in board for use on the GIO64 bus. Additionally external products can be designed to digitize video and deliver it to this interface for VINO to place on the GIO 64 bus.

3.2.1 Data Format

The data on this bus complies with the logical format of the SMPTE Recommended Practice 125-1984 and is as follows:

U0 Y0 V0 Y1 U2 Y2 V2 Y3 U4 Y4 V4 Y5 . .. . UN YN VN YN+1

It is anticipated that all data sets on this input will be divisible by 4 for the total count of a line of video in order to preserve the Color and Luminance relationship of 4:2:2. The VINO chip expects this to be so.

3.2.2 Control Codes Embedded in Data Format

Horizontal, Vertical and Field identifiers are embedded in the data stream as follows:

UxYxVxYx+1 FF 00 00 SS 80 10 ... blanking ... 80 10 FF 00 00 SS V0 Y1 U2 Y2 V2 Y3 . . . .

The SS byte of the string above can represent either and EAV (end of active video) or SAV (start of active video) on a given horizontal line. The SS byte is always preceded by the FF 00 00 sequence and the FF and 00 values are not allowed as valid levels in the active video stream. During blanking the U and V components are clamped to 128 (80H) while the Y component is clamped to 16 (10H). The SS byte contains Hamming code to correct any single bit error that may occur during transmission. The error correction applies only to the SS byte and not the video bytes or the preamble sequence leading up to the SS byte.

The Data Format of the SS bit is as follows:

TABLE 2. Digital Video Control Byte Format

Bit Position	Bit ID	Function
D7	1	Always set to 1
D6	F	Field Type, 0=Odd 1=Even
D5	V	Vertical Blanking 1=Blanking, 0=Active Can only change when H=1 (EAV)
D4	H	Horizontal Blanking 1=Start Blanking, 0=Start Active Video
D3	P3	Hamming Code error protection
D2	P2	Hamming Code error protection
D1	P1	Hamming Code error protection
D0	P0	Hamming Code error protection

3.2.3 Error Protection Bits

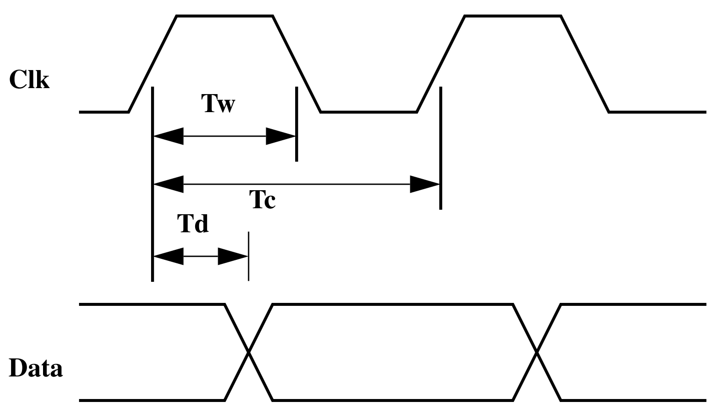
The following code is used to correct single bit errors that may occur in the transmission of the Timing Control words on the SGI Digital Video Interface.

TABLE 3. Hamming Code Words for Error Protection Scheme to work

F	**V**	**H**	**P3**	**P2**	**P1**	**P0**
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	1	0
1	1	0	1	1	0	0
1	1	1	0	0	0	1

3.2.4 SGI Digital Video Timing Information

The typical clock frequency for the SGI Indy Cam is 24.5454 MHz. The VINO interface has been simulated to work up to 30 MHz. It is expected that the Clock to Data relationship for the Digital Video Input Port have the following characteristics for the 24.5454 typical case. The intent is to place the rising edge of the clock as near as possible to the time the data is stable.



Tw = 20 +/- 3 nsec

Tc = 24.5454 Mhz, 40 nsec nominal

Td = 20 +/- 3 nsec

FIGURE 6. SGI Digital Video Interface Transmitted Clock Waveform

4.0 Video Processing

Each of the processing blocks described below are independently selectable in the two channels.

4.1 Data Formats

The data received from the Philips and D1 interface is in 4:2:2 YUV form, with 8-bits per component. This can be changed into 24-bit RGB. The 24 bits can also be reduced to 8-bit RGB. The 24-bit RGB is presented to the system as 32-bit RGBA by appending the value in the Alpha Register. These pixels are then packed into 64 bit words as shown in the diagram below.

Big Endian Arrangement in GIO64 Word

format	63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0
RGBA	A0	B0	G0	R0	A1	B1	G1	R1
YUV	U0	Y0	V0	Y1	U2	Y2	V2	Y3
8-bit Mono or RGB	P0	P1	P2	P3	P4	P5	P6	P7

	7 6	5 3	2 0
8-bit RGB	B	G	R

	7 0
8-bit MONO	Y

For YUV data, the U and V components are shared for the two Y samples. In order too reduce the hardware complexity, only a doubleword number of pixels are DMA'd per line. If the little endian bit is set the RGBA format above will swap the positions of the upper and lower 32 bit words but preserve the byte ordering within a 32 bit word. All other formats swap bytes in the normal big-endian to little-endian convention across a 64 bit word. Note that this feature while present in the hardware is an untested function in the -001 silicon.

Little Endian Arrangement in GIO64 Word

format	63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0
RGBA	A1	B1	G1	R1	A0	B0	G0	R0
YUV	Y3	V2	Y2	U2	Y1	V0	Y0	U0
8-bit Mono or RGB	P7	P6	P5	P4	P3	P2	P1	P0

	7 6	5 3	2 0
8-bit RGB	B	G	R

	7 0
8-bit MONO	Y

4.2 Filtering/Decimation

Decimation reduces the video window from full size to 1/2, 1/3, 1/4, 1/5, 1/6, 1/7 or 1/8 in both horizontal and vertical directions on a field basis; independent x or y decimation is not supported. Pixels are averaged horizontally, stored and then averaged vertically. If the decimation size does not evenly divide the window size, missing pixels will be treated as black. This artifact will only occur on the right- and bottom-most edges of an image. Bits X of register X control the decimation factor. The table below shows the weights applied both horizontally to average pixels and vertically to average lines

It is recommended that only the 1/2, 1/3 and 1/4 settings be used as the algorithm implemented in Vino -001 version simply divides each source pixel by the total decimation factor and does not preserve any fractional information from that operation. For example in the 1/4 setting each pixel in the 4 by 4 block of source pixels that will become a single decimated pixel is divided by 16 and then added together. This means each pixel loses 1/2 of its color precision. It is possible that this will be fixed in a future version of the chip.

TABLE: Decimation Weights

Pixel/Line	0	1	2	3	4	5	6	7
Weight	1/2	1/2	1/2	1/2	1/2	1/2	1/2	1/2
Decimation 1/2		N0		N1		N2		N3
Weight	1/4	1/2	1/4	1/4	1/2	1/4	1/4	1/2
Decimation 1/3			N0			N1		
Weight	1/4	1/4	1/4	1/4	1/4	1/4	1/4	1/4
Decimation 1/4				N0				N1

4.3 Dithering

In order to reduce the video bandwidth for some applications where image quality is not paramount, but full size is still important, a dithering with a matrix which minimizes introduced texture and beating, is performed on a field basis. Each 24-bit RGB pixel is reduced to an 8-bit (BBGGGRRR) pixel. Note, for dithering to work properly, YUV to RGB conversion should be enabled. Initially the intensity level of the pixel is scaled to evenly repartition existing values into the fewer number of bits. These bits are then modulated according to a comparison of the remainder of the scaling and the value returned by indexing into the dither matrix by the pixel position. This method has the nice feature that overflow will never occur on the resultant pixel value. The pseudo-code below describes the operation. The initial 8-bit color component is ‘I’, the scaled component is ‘S’ and the result is ‘R’.“>>” is the logical shift operator. Note, using a Bayer matrix results in beating between fields as the values are not evenly divided incrementally between the two fields. The matrix below

Dithering eight bits to three for the red and green components:

```
S[6:0] = I[7:1] - (I[7:4] >> 3) >> 1
if (S[3:0] > M[y mod 4][x mod 4]) then R = S[6:4] + 1;
else R = S[6:4];
```

Dithering eight bits to two for the blue component:

```
S[5:0] = I[7:2] - (I[7:4] >> 2) >> 1
if (S[3:0] > M[y mod 4][x mod 4]) then R = S[5:4] + 1;
else R = S[5:4];
```

Dithering is turned on or off through the status register.

Dither Matrix

12	7	11	0
10	1	13	6
5	14	2	9
3	8	4	15

4.4 Clipping

The incoming video stream contains a different number of pixels per line and lines per field depending on the standard selected. Not all of the data contains live video, for example during the vertical blanking period. In order to remove this data from the video stream, a general mechanism is provided which can also be used to clip the active video image. For horizontal clipping, registers specify the starting and ending pixel position on a line. Registers specify the starting and ending line numbers for vertical clipping. There are two sets of registers. One is intended for even fields and another for odd fields, but no restrictions are made to their application. The line count begins on the falling vertical retrace of even fields.

There is an important relationship is setting the clip register starting Y value of the two fields. First field1 is always one line lower in clipping than the equivalent setting for field 2 because of the way the falling edge of vertical retrace and HREF behave in the two different fields (see YUV Bus description for details). Second is that since the two fields may be re-interleaved by software at some future point, the spacial relationship must be preserved or the image will be distorted. So if both fields are set to clip at the same number then when they are re-interlaced field 2 should have its first line placed above field 1 in order to maintain the correct physical relationship of the fields within a frame. If field 2 is set to n+1 the Y start clip of field 1 then field 1 should have the first line placed above the first line of field 2. Experimentation is greatly encouraged here to verify these rules.

It should also be noted that the clipping regions for Y start and Y end will vary between various consumer and professional analog Composite and Y/C sources since there is room for interpretation in the Specifications for the exact number of blank lines before start of active video in vertical. This will also be different for our INDY CAM which is encoded onto the SGI Digital Interface input to VINO.

Clipping Rule. The Line Size for clipping must be set so that the total number of pixels clipped AND the downstream filter decimation (1/2, 1/3, 1/4) produce enough pixels to fill an integer number of 8 byte GIO 64 words. This problem is further complicated when combined with the color space mode possibilities that allow from 8 to 2 pixels packed into an 8 byte GIO word.

An example of an legal clip setting would be 600 pixels in the line with 1/2 decimation and RGB-8. To verify that this is legal take 600 apply the 1/2 decimation factor which produces 300 pixels. Divide 300 pixels by 8 and see if an integer number of GIO words is produced. Since this yields 37.5 GIO words this is not a legal CLIP/DECIMATE/COLORSPACE combination. However if the color space is 24 RGBA (2 pixels per GIO64 Word, then 300/2 is 150 GIO words and that would be a legal clip/decimate/colorspace combo.

4.5 Color Space Conversion

Graphics pixels are RGB, while video pixel are YUV based, requiring conversion to allow the system to manipulate and display the video stream. Channel A is set by bit11 and channel B by bit23 in the Control Register. For conversion to 8-bit RGB the pixels are first converted from YUV to 24 bit RGB pixels and then dithered down to 8 bit pixels. This mode is invoked by setting the Control Register bit 18 for Channel A and bit 30 for Channel B. One final mode of color (or lack there-of) is supported which is a Monochrome only mode. In this mode the U and V components are simply discarded and only the Y component which represents a gray-scale value is preserved. This mode is extremely efficient in that it can pack a full intensity shaded image into the least amount of Bus Bandwidth and destination memory size.

The required conversion equations for 4:2:2 YUV to 24 bit RGB are shown below. The hardware does not interpolate from 4:2:2 to 4:4:4 before performing color space conversion. The UV value for a given Y value is simply repeated for the next Y value which does not have its own UV value.

$$R = 1.596027 \times (V' - 128) + 1.164384 \times (Y' - 16)$$

$$B = 2.017232 \times (U' - 128) + 1.164384 \times (Y' - 16)$$

$$G = 1.164384 \times (Y' - 16) - 0.8129676 \times (V' - 128) - 0.3917623 \times (U' - 128)$$

The color space conversion is performed by shifts and adds which introduces an error of less that +/-0.6 for the value range of 0 to 255 for a precision of 99.5%. The color space conversion is clamped to 0 and 255 to ensure illegal color values are not computed. The vhdl model of the Color Space computational hardware was verified against a C program using floating point arithmetic and the above equations to verify the accuracy of the function.

4.6 Synchronization Control

The video stream from Vino is raw: no control or tag information is included. This requires each field to be properly placed in memory or the display and interleaving of the fields will be incorrect. When DMA is enabled, the video stream will start on the next available field. Setting the synchronization bit in the Control Register will ensure the first field captured for every frame is the odd field. The second field can be either odd or even. Multiple odd or even fields in a row can be caused by some VCR's in pause mode. Note that no fields will be captured if synchronization is enabled and only even fields are produced. All fields will be captured if synchronization is disabled.

4.7 Frame/Field Rate Control

In order to reduce the number of fields a channel produces, a coarse limiting mechanism is provided. The Frame Rate Control Register is a twelve field (six frame) mask with a mode bit. Each mask bit has an associated field and when the bit is set the field is captured, otherwise it is dropped. The mode bit controls the number of mask bits used. When the mode bit is 0 (NTSC) all 12 mask bits are used, when it is 1 (PAL mode) only the first 10 mask bits are used. The mask is loaded into a shift register with the low-order bit indicating the capture status of the next field. The shift register is clocked after a valid field completes. Once the all the bits are used, the shift register is reloaded with the mask. The register has the following format:

Bit	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	11	10	9	8	7	6	5	4	3	2	1	0	mode
Frame	5		4		3		2		1		0		

When interleaving is enabled, the mask must be enabled on a frame, not field, basis (both fields in the frame should have the same mask), or the DMA mechanism will not function properly. The six frame stencil applies in the NTSC mode. The five frame stencil is used in PAL mode to more easily mesh with the 50Hz frequency producing rates of 5, 10, 15, 20 and 25 frames per second. NTSC input can make use of both NTSC mode, producing 5, 10, 15, 20, 25 and 30 frames per second, or PAL mode, producing 6, 12, 18, 24 and 30 frames per second. Neither mode is constrained to a particular standard. Software can additionally extract frames from this subset to produce a rate not provided.

4.8 Alpha Value

The Alpha Register contains a double-buffered byte value which is included in the pixel stream in RGBA mode. The buffer is loaded with the Alpha Register at the end of every field. This allows alpha values to be smoothly changed on field boundaries.

5.0 Software Issues

5.1 Data Formats

Unfortunately pixel data formats, as with so many other supposed standards, vary among machines. YUV data is now a coherent collection of bytes, making it incompatible with Starter Video. The 8-bit RGB format produced by dithering is incompatible with Indigo, due to Indigo's RRRBBGGG format. Applications sharing video data across platforms will need to either shift bits or use a non-standard color map. VINO's 8-bit data is compatible with Rex3.

5.2 Graphics Synchronization

The video stream must be synchronized to the graphics display using the graphics vertical retrace interrupt to prevent image tearing. This implies the DMA moving the image from memory to the frame buffer must wait for the vertical retrace interrupt before beginning the DMA. The DMA to Rex3 need not be performed only during vertical retrace, just started then. Care must be taken to ensure the DMA will remain ahead of the scan. This method eliminates the need for double buffering. The vertical interrupt from VC2 can be adjusted for system and software induced latency.

5.3 Frame Assembly

Frame assembly can be achieved a number of ways varying in bandwidth and quality. No frame assembly is done on the chip, as there is no field or line store. The video image is de-interlaced and synchronized by moving it to main memory. Once a field is in memory, it can be presented to the frame buffer with black inserted in between the field lines, resulting in a crisp, half-brightness picture with no artifacts. A more ideal method is to create a frame from every field by line replication. This results in a full intensity picture with some aliasing due to the replication, which is outweighed by intensity gain. The bandwidth from memory to the frame buffer is doubled, making this solution attractive to only certain users. A reduced bandwidth method, which has inter-field motion artifacts but is a full brightness image, is to DMA each new field, leaving the older alternate field in the frame buffer, displaying the two most recent fields simultaneously. If a field or frame is dropped, this method results in an unacceptable non-coherent mixture of fields. A third reduced bandwidth method is to drop a field and create a frame with the remaining field. The motion may be jerky between frames, but there will be no inter-field artifacts visible on screen. The user should be allowed make a selection among these or other display methods as preferences are subjective.

6.0 Register Descriptions

In the following 64-bit register descriptions, all bits not explicitly defined are read as 0

TABLE 4. ID and Revision Register

Bits	Function	Reset Value
7:4	VINO ID value	B
3:0	VINO revision number	0000

The *Control* register is read/write. The bits in the *Control* register are set and cleared by software. The enable bits are active high. The register is cleared on reset.

TABLE 5. Control Register

Bits	Function
30	chB dither 1 - enabled 24-bit to 8-bit, 0 - disabled
29	chB decimation horizontal only enable
26-28	chB decimation scale factor
25	chB decimation enable
24	chB luma only enable
23	chB color space conversion 1 - RGB, 0 - YUV
22	chB selection 1 - D1 interface, 0 - Philips chip
21	chB sync enable
20	chB enable interleave
19	chB DMA enable
18	chA dither 1 - enabled 24-bit to 8-bit, 0 - disabled
17	chA decimation horizontal only enable
14 -16	chA decimation scale factor
13	chA decimation enable
12	chA luma only enable
11	chA color space conversion 1 - RGB, 0 - YUV
10	chA selection 1 - D1 interface, 0 - Philips chip
9	chA sync enable
8	chA enable interleave
7	chA DMA enable
6	chB enable end of descriptor table interrupt
5	chB enable fifo overflow interrupt
4	chB enable field transferred interrupt
3	chA enable end of descriptor table interrupt
2	chA enable fifo overflow interrupt
1	chA enable field transferred interrupt
0	endianess of GIO interface 1 - little endian, 0 - big endian

The *Interrupt* register is read/write. All bits are active high. The bits are set by hardware and cleared by software (write 0 to clear). The register is cleared on reset. Writing a one to any of the bits has no effect on the value of the bit.

TABLE 6. Interrupt and Status Register

Bits	Function
5	chB end of descriptor table interrupt
4	chB fifo overflow interrupt
3	chB end of field transferred interrupt
2	chA end of descriptor table interrupt
1	chA fifo overflow interrupt
0	chA end of field transferred interrupt

.

TABLE 7. Alpha Register

Bits	Function	Read/ Write	Reset Value
7:0	Alpha value	R/W	not reset

.

TABLE 8. Clipping Start Register

Bits	Function	Read/ Write	Reset Value
27:19	Y Even start	R/W	not reset
18:10	Y odd start	R/W	not reset
9:0	X start	R/W	not reset

.

TABLE 9. Clipping End Register

Bits	Function	Read/ Write	Reset Value
27:19	Y Even end	R/W	not reset
18:10	Y odd end	R/W	not reset
9:0	X end	R/W	not reset

TABLE 10. Frame Rate Register

Bits	Function	Read/ Write	Reset Value
12:1	frame rate mask value	R/W	not reset
0	mode 0 - NTSC, 1 - PAL	R/W	not reset

Enabling DMA will reset the Field Counter Register.

TABLE 11. Field Counter Register

Bits	Function	Read/ Write	Reset Value
15:0	field count value (incrementing counter)	R	0x00

TABLE 12. ChX Line Size Register

Bits	Function	Read/ Write	Reset Value
11:3	line size in bytes/ 8 lines must always be a multiple of 8 bytes	Read/Write	0x00

TABLE 13. ChX Line Count Register

Bits	Function	Read/ Write	Reset Value
11:3	incrementing counter - maintains current byte in line count	Read/Write	0x00

TABLE 14. ChX Page Index Register

Bits	Function	Read/ Write	Reset Value
11:3	Index to doubleword in the current page (bits 11:3 of DMA address)	Read/Write	0x00

To begin a field capture operation, *page_index* must be set to zero. The address of the start of the descriptor table must then be written into *next_descriptor*. As soon as *next_descriptor* is written, the descriptor cache is invalidated causing a descriptor fetch to occur. It is important that *page_index* precede *next_descriptor*. As stated in the previous paragraph, descriptors shift when *page_index* rolls over, defined as *page_index* going from a non-zero value to zero. Depending on initial conditions, setting *page_index* to zero can cause a *page_index* roll over and hence a shift in descriptors. If *next_descriptor* is written first, descriptors are fetched, and a subsequent write to *page_index* might cause the descriptors to shift. After these registers have been initialized, DMA can be enabled.

TABLE 15. ChX Pointer to Next Four Descriptors Register

Bits	Function	Read/ Write	Reset Value
31:4	pointer to next four descriptors (quad word address)	Read/Write	0x0000

TABLE 16. ChX Pointer to Start of Descriptor Table Register

Bits	Function	Read/ Write	Reset Value
31:4	pointer to start of descriptor table (quad word address)	Read/Write	0x0000

TABLE 17. ChX DescriptorX Data Register

Bits	Function	Read/ Write	Reset Value
32	valid bit	Read	1
31:0	descriptor data	Read/Write	0x0000

TABLE 18. ChX Fifo Threshold Compare Value Register

Bits	Function	Read/ Write	Reset Value
9:3	fifo threshold compare value	Read/Write	0x00

TABLE 19. ChX Fifo GIO Pointer Register

Bits	Function	Read/ Write	Reset Value
9:3	fifo GIO pointer	Read	0x00

TABLE 20. ChX Fifo Video Pointer Register

Bits	Function	Read/ Write	Reset Value
9:3	fifo video pointer	Read	0x00

TABLE 21. I^2C Control and Status Register

Bit	Function	Read/ Write	Reset Value
7	bus error status 1 - bus error 0 - no bus error	Read	0
5	acknowledge status 1 - acknowledge not received 0 - acknowledge received	Read	0
4	transfer status 1 - transfer busy 0 - transfer done	Read	0
2	last byte control 1 - more bytes hold onto bus 0 - last byte release bus	Read/Write	0
1	bus direction control 1 - read data 0 - write data	Read/Write	0
0	force idle state control 1 - write - no effect 0 - write - force idle state 1 - read - not idle 0 - read - idle	Read/Write	0

TABLE 22. I^{2}C Data Register

Bits	Function	Read/ Write	Reset Value
7:0	bus data write to this address initiates a write cycle if bit 1 of the *I^{2}C Control* register is 1, a read of this address initiates a new read cycle	Read/Write	0x00

7.0 Signal Descriptions and Pinout

The Logical Pin diagram below shows the Pins on VINO arranged by functional grouping. The following table covers detailed characteristics of each VINO pin (Drive Strength, direction and description) while the final table in this section provides the detailed pin number assignments for the 208 PQFP package that is home to the VINO silicon sliver.

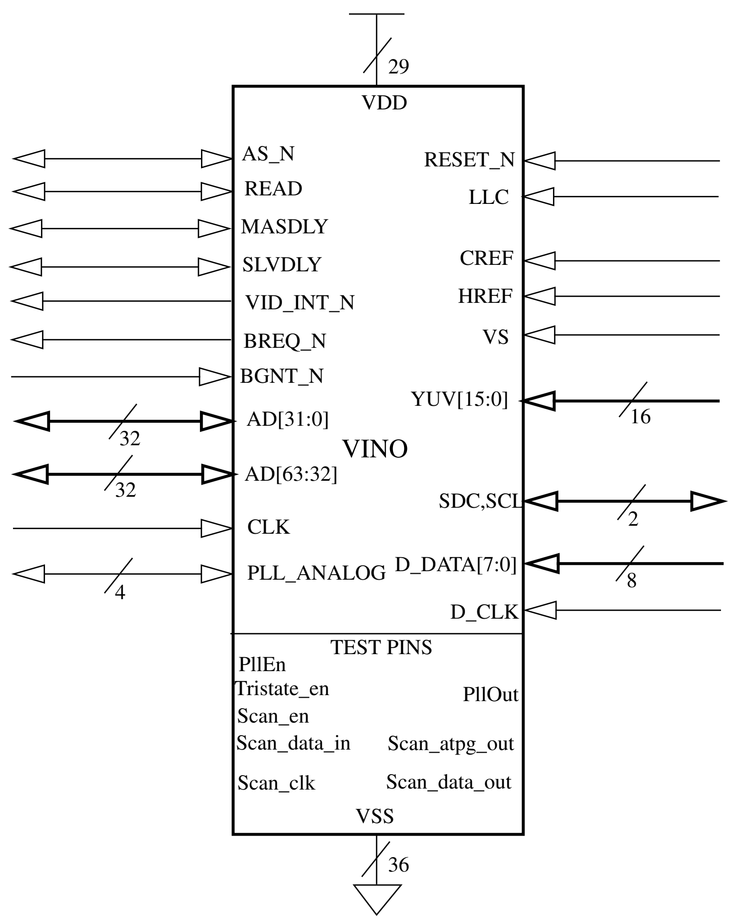


FIGURE 7. Logical pin diagram of VINO

TABLE 23. Pin Description

Pin Name	Type	Description	Output Drive
GIO_AS_n	I/O	I- Slave - GIO64 Address Strobe Valid O - Master - GIO64 Address Strobe Valid	8 mA
GIO_Read	I/O	I - Slave - GIO64 Read/Burst In Progress O - Master - GIO64 Read/Burst In Progress	8 mA
GIO_MasDly	I/O	I - Slave - GIO64 Read - Data Valid, Write - Accept Data O- Master- GIO64 Read - Data Valid, Write - Accept Data	8 mA
GIO_SlvDly	I/O	O - Slave - GIO64 Read - Data Valid Write - Accept Data I - Master - GIO64 Read - Data Valid Write - Accept Data	8 mA
GIO_AD(63:0)	I/O	64 Bit Bi-Directional Multiplexed Address/Data GIO64 Bus	8 mA
GIO_Clk	I	Input Clock 33MHz. VINO has PLL to bring clock inside VINO to any flip flop to within 3 nsec of this clock rising edge	
GIO_BReq_n	O	GIO Bus Request	4 mA
GIO_BGnt_n	I	GIO Bus Grant	
GIO_Int_n	O	Interrupt to IOC Chip	4 mA
Reset_n	I	Reset Signal from Phillips 7197 Chip synchronous to P_llc clock. GIO Section of VINO re-clocks this with GIO_CLK before using as a reset for GIO_CLK flops	
SCL	I/O	I2C Clock Bi-Directional - Vino Drives Low Only - Open Drain emulation	4 mA
SDA	I/O	I2C Data Bi-Directional - Vino Drives Low Only - Open Drain emulation	4 mA
P_Y(7:0)	I	8 bit Luma Samples from Analog Decoder	
P_UV(7:0)	I	8 bit Chroma Samples from Analog Decoder	
P_href	I	1 - Horizonatal Line Active Pixel 0 - Horizontal Blanking	
P_vs	I	1 - Vertical Sync Active 0 - Vertical Active Pixel	
P_cref	I	1 - P_llc / 2 Indicates Valid Pixel Sample 0 - Invalid Pixel	
P_llc	I	2x Video Pixel clock	
D_data(7:0)	I	SGI D1 Video Input Bus Expects 4:2:2 YUV	
D_clk	I	SGI D1 Video Pixel Clock ~ 24.5454 MHz	
Scan_clk	I	Input for ATPG Scan. Selected per Matrix in the Clock Selection Section.	
Scan_data_in	I	Data Input Pin for ATPG Scan Test.	
Scan_data_out	O	I/O Boundary Scan Chain Data Output (Board Test Data Out)	4 mA
Scan_atpg_out	O	Entire Scan Chain Data Output (Chip Test Data Out)	4 mA
Scan_en	I	Mode Pin to Put Chip in Various Clock/Test Modes	
TristateEn	I	1 - Ouputs Active, PLL Enabled and Powered On 0 - Outputs 3-State, PLL Disabled and Powered Off	
PllEn	I	Mode Pin to Put Chip in Various Clock/Test Modes	

TABLE 23. Pin Description

Pin Name	Type	Description	Output Drive
PllOut	O	Internal PLL 33 Mhz divided by 16 for test	4 mA
Pll_RC1	Anal	Analog Phase Lock Loop Filter Pin 1.	
Pll_RC2	Anal	Analog Phase Lock Loop Filter Pin 2.	
Pll_VDD	Anal	Analog Power for internal PLL	
Pll_AGND	Anal	Analog Ground for internal PLL	
Pll_VSS	Anal	Pll VSS Gnd tied to Board Digital Ground Outside Chip	
VDD	PWR	29 Pins Total Digital +5V Power	
VSS	GND	36 Pins Total Digital Ground	

7.1 Pin Assignments

The following table provides the physical pin to signal name assignment for the 208 PQFP VINO component.

.

TABLE 24. 208 PQFP Pin Assignments

Pin #	Signal	Pin #	Signal	Pin #	Signal
1	VSS	2-23	No Connect	24	VSS
25	Scan_en	26	Scan_data_in	27	Scan_atpg_out
28	Scan_data_out	29	TristateEn	30	No Connect
31	Reset_n	32	VDD	33	VSS
34	GIO_MasDly	35	VDD	36	VSS
37	VSS	38	GIO_BReq_n	39	GIO_Int_n
40	GIO_SlvDly	41	GIO_AS_n	42	GIO_BGnt_n
43	VDD	44	VSS	45	VSS
46	GIO_Read	47	VSS	48	GIO_AD0
49	VDD	50	GIO_AD1	51	VSS
52	VDD	53	GIO_AD2	54	GIO_AD3
55	GIO_AD4	56	GIO_AD5	57	VSS
58	GIO_AD6	59	GIO_AD7	60	GIO_AD8
61	GIO_AD9	62	VDD	63	VSS
64	GIO_AD10	65	GIO_AD11	66	GIO_AD12
67	GIO_AD13	68	VDD	69	GIO_AD14
70	GIO_AD15	71	GIO_AD16	72	GIO_AD17
73	VDD	74	VDD	75	VSS
76	GIO_AD18	77	GIO_AD19	78	GIO_AD20
79	GIO_AD21	80	VSS	81	GIO_AD22
82	GIO_AD23	83	VSS	84	GIO_AD24
85	GIO_AD25	86	VDD	87	GIO_AD26

TABLE 24. 208 PQFP Pin Assignments

Pin #	Signal	Pin #	Signal	Pin #	Signal
88	GIO_AD27	89	GIO_AD28	90	GIO_AD29
91	GIO_AD30	92	VSS	93	VDD
94	GIO_AD31	95	GIO_AD32	96	GIO_AD33
97	GIO_AD34	98	VSS	99	VDD
100	GIO_AD35	101	GIO_AD36	102	VDD
103	VDD	104	VSS	105	GIO_AD37
106	GIO_AD38	107	VSS	108	GIO_AD39
109	GIO_AD40	110	GIO_AD41	111	VDD
112	GIO_AD42	113	GIO_AD43	114	GIO_AD44
115	VSS	116	VDD	117	GIO_AD45
118	GIO_AD46	119	GIO_AD47	120	VDD
121	GIO_AD48	122	GIO_AD49	123	VSS
124	GIO_AD50	125	GIO_AD51	126	GIO_AD52
127	GIO_AD53	128	GIO_AD54	129	VSS
130	VDD	131	GIO_AD55	132	GIO_AD56
133	GIO_AD57	134	VSS	135	GIO_AD58
136	GIO_AD59	137	GIO_AD60	138	GIO_AD61
139	VDD	140	VSS	141	VSS
142	GIO_AD62	143	GIO_AD63	144	VDD
145	VSS	146	PLLEn	147	Scan_clk
148	VSS	149	VDD	150	VSS
151	Pll_AGND	152	Pll_RC1	153	Pll_VSS
154	PLL_RC2	155	Pll_VDD	156	GIO_Clk
157	VSS	158	VDD	159	SDA
160	SCL	161	VSS	162	VDD
163	D_data0	164	D_data1	165	D_data2
166	VSS	167	VDD	168	D_data3
169	D_data4	170	D_data5	171	D_data6
172	D_data7	173	VDD	174	VSS
175	D_clk	176	VDD	177	VSS
178	P_Y0	179	P_Y1	180	P_Y2
181	P_Y3	182	P_Y4	183	P_Y5
184	P_Y6	185	P_Y7	186	No Connect
187	No Connect	188	VDD	189	VSS
190	VSS	191	P_UV0	192	P_UV1
193	P_UV2	194	P_UV3	195	P_UV4
196	P_UV5	197	P_UV6	198	P_UV7
199	P_href	200	P_vs	201	VSS
202	P_cref	203	VDD	204	P_llc

TABLE 24. 208 PQFP Pin Assignments

Pin #	Signal	Pin #	Signal	Pin #	Signal
205	VSS	206	No Connect	207	PllOut
208	VDD				

7.2 Test Modes

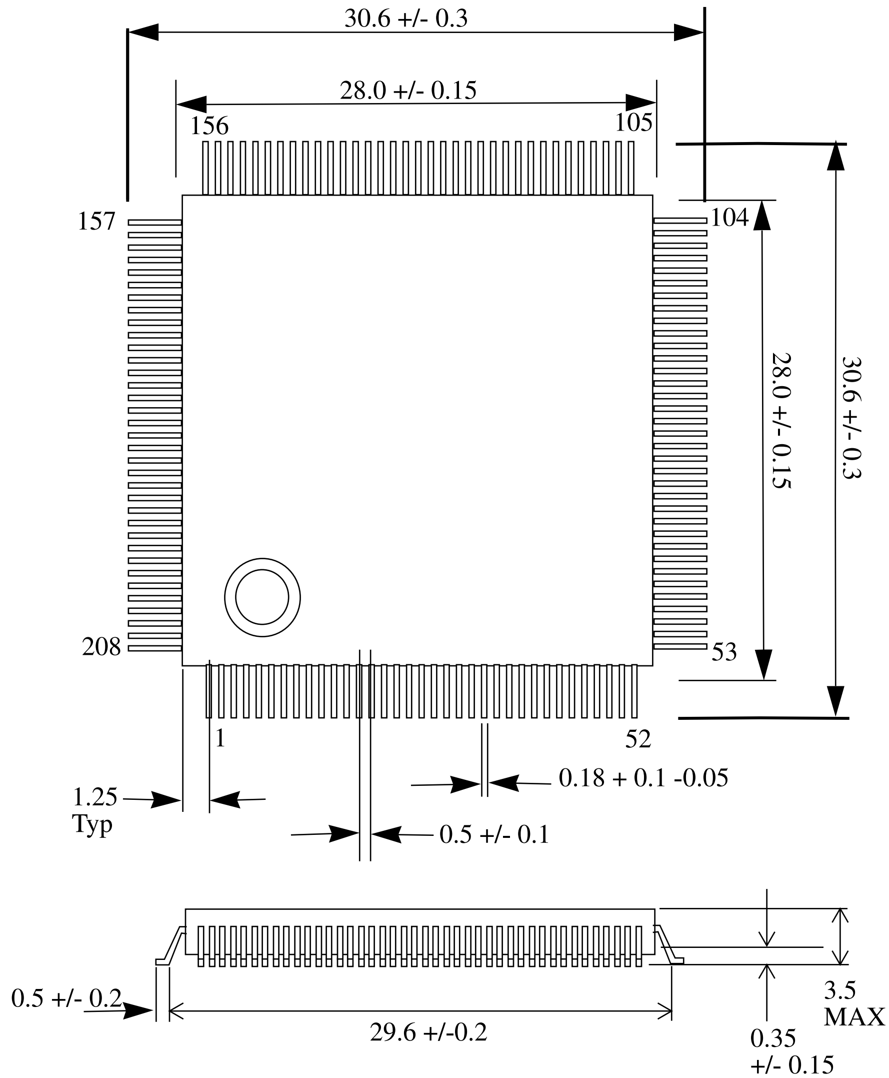
The input pins SCAN_EN, PLL_EN and TRISTATE_EN control various test and functional modes of the VINO chip per the table below. Note that in the course of implementing the control that the pin labeled PLL_EN influences which clock the GIO Flops are connected to but that TRI_EN actually disables the Phase Lock Loop.

TABLE 25. Test Mode Pin Functions

VINO Function	TRI_ EN	SCAN_ EN	PLL_E N	GIO Flop Clock	Video Flop Clock
Normal Operation w/ PLL and during RESET	H	L	H	GIO_CLK-PLL	Video Clks
Normal Operation No PLL Functional Vectors	H	L	L	GIO_CLK-PAD	Video Clks
Scan Flops Use D Input	H	H	L	Scan_Clk	Scan_Clk
Scan Flops Use SI Input	H	H	H	Scan_Clk	Scan_Clk
PC, VCO, RAM Disabled Functional Vectors Parametric Test	L	L	L	GIO_CLK-PAD	Video Clks
PC, VCO, RAM Disabled Scan Flops Use D Input	L	H	L	Scan_Clk	Scan_Clk
PC, VCO, RAM Disabled Don't Use PLL Clock is Z and drives Clock Mux down GIO Clock Tree	L	L	H	Undefined	Video Clks
PC, VCO, RAM Disabled Scan Flops Use SI Input	L	H	H	Scan_Clk	Scan_Clk

8.0 Packaging

VINO is packaged in a 208 PQFP Heat Spreader Package to accommodate up to 2 Watts of Power Dissipation.



All Units in mm

FIGURE 8. 208 Plastic Quad Flat Pack Package Dimensions

9.0 DC Characteristics

TABLE 26. DC Characteristics (T_j =-20 to 75^o C)

Parameter	Description	Conditions	Min	Max	Units
V_{IL}	Low Input Voltage	V_{DD} = 4.75V	0	0.8	V
V_{IH}	High Input Voltage	V_{DD} = 5.25V	2.2	5.25	V
V_{OL}	Low Output Voltage	V_{DD} = 5V I_O = 0 mA	----	0.05	V
V_{OH}	High Output Voltage	V_{DD} = 5V I_O = 0 mA	4.95	----	V
I_{IH}, I_{IL}	Input Leakage Current	V_I = V_{DD}, V_{SS}	-1	+1	uA
I_{OZH}, I_{OZL}	Output Leakage Current	V_O = V_{DD}, V_{SS}	-1	+1	uA
C_{OUT}	Output Pin Capacitance 1 - 8 mA	f = 1 MHz V_{DD} = 0V	----	15	pF
C_{OUT}	Output Pin Capacitance 16 mA	f = 1 MHz V_{DD} = 0V	----	20	pF
Power	Power Dissipation	VDD = 5V		1.48	W

10.0 AC Characteristics

10.1 I/O Timing Table

TABLE 27. AC Characteristics

Description	Min	Max	Units
GIO_AS_n setup to GIO_Clk rising	7.55	----	ns
GIO_AS_n hold from GIO_Clk rising	1.59	----	ns
GIO_AS_n delay from GIO_Clk rising	----	12.01	ns
GIO_Read setup to GIO_Clk rising	4.13	----	ns
GIO_Read hold from GIO_Clk rising	0.93	----	ns
GIO_Read delay from GIO_Clk rising	----	12.01	ns
GIO_MasDly setup to GIO_Clk rising	4.58	----	ns
GIO_MasDly hold from GIO_Clk rising	0.96	----	ns
GIO_MasDly delay from GIO_Clk rising	----	12.01	ns
GIO_SlvDly setup to GIO_Clk rising	3.5	----	ns
GIO_SlvDly hold from GIO_Clk rising	0.3	----	ns
GIO_SlvDly delay from GIO_Clk rising	----	11.75	ns
GIO_AD(63:0) setup to GIO_Clk rising	6.58	----	ns
GIO_AD(63:0) hold from GIO_Clk rising	2.98	----	ns
GIO_AD(63:0) delay from GIO_Clk rising	----	13.36	ns
GIO_BReq_n delay from GIO_Clk rising	----	10.29	ns
GIO_BGnt_n setup to GIO_Clk rising	4.16	----	ns

TABLE 27. AC Characteristics

Description	Min	Max	Units
GIO_BGnt_n hold from GIO_Clk rising	0.96	----	ns
GIO_Int_n delay from GIO_Clk rising	----	9.51	ns
Reset_n Low Pulse Width	10.0	----	us
SDA refer to Philips I^2C Specification	----	----	----
SCL refer to Philips I^2C Specification	----	----	----
Scan_data_in	NA	NA	
Scan_data_out delay from Scan_Clk rising	----	11.95	ns
Scan_atpg_out	NA	NA	
Scan_en setup to Scan_Clk rising	----	29.27	ns
Scan_en hold from Scan_Clk rising	7.85	----	ns
PllEn setup to Scan_Clk rising	----	15.68	ns
PllEn hold from Scan_Clk rising	9.95	----	ns
PllOut delay from GIO_Clk rising	----	14.93	ns
D_data setup to D_Clk	3.0	----	ns
D_data hold to D_Clk	2.0	----	ns
P_href, P_vs, P_cref setup to P_llc	3.0	----	ns
P_href, P_vs, P_cref hold to P_llc	2.0	----	ns
P_Y, P_UV setup to P_llc	3.0	----	ns
P_Y, P_UV hold to P_llc	2.0	----	ns

10.2 PLL Characteristics

VINO contains an analog Phase Lock Loop. This section utilizes the GIO_Clk input as its reference and is designed to put the clock edge at all the GIO_Clk flip flops inside of VINO within 3 nsec of the clock provided to VINO at the GIO_Clk pin. This PLL is characterized to run at the nominal 33 MHz GIO bus speed and has not been characterized at 25 MHz or frequencies higher than 33 MHz.

It is necessary to connect a resistor capacitor network to VINO on the PLL RC1 and RC2 pins as per the figure below.

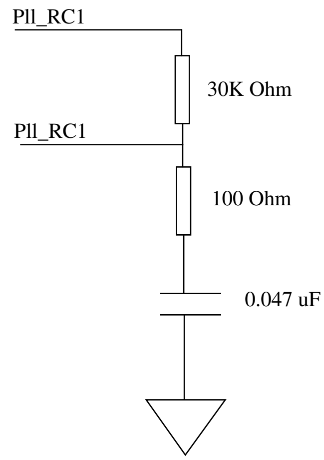


FIGURE 9. VINO PLL Network

11.0 Fault Coverage

Running the 1.5 Million vectors generated by the Sunrise ATPG tool results in a 92.12% fault coverage. Since this number was lower than desired, a limited number of functional vectors were fault graded to obtain an overall fault coverage of 92.79%.

12.0 Timing

FIGURE 10. MC read from VINO register

Insert from Post Script File **099-8937-001.ps.01** to be printed out separately.

FIGURE 11. Write MC to Vino

Insert from Post Script File **099-8937-001.ps.02** to be printed out separately.

FIGURE 12. VINO Descriptor Fetch

Insert from Post Script File **099-8937-001.ps.03** to be printed out separately.

FIGURE 13. VINO fifo empty

Insert from Post Script File **099-8937-001.ps.04** to be printed out separately.

FIGURE 14. DMA with Page Index Roll

Insert from Post Script File **099-8937-001.ps.05** to be printed out separately.

FIGURE 15. DMA with End of Field Interrupt

Insert from Post Script File **099-8937-001.ps.06** to be printed out separately.

13.0 Indy Related Issues

13.1 Application Block Diagram

VINO as implemented in the INDY product CPU Mother Board is shown below.

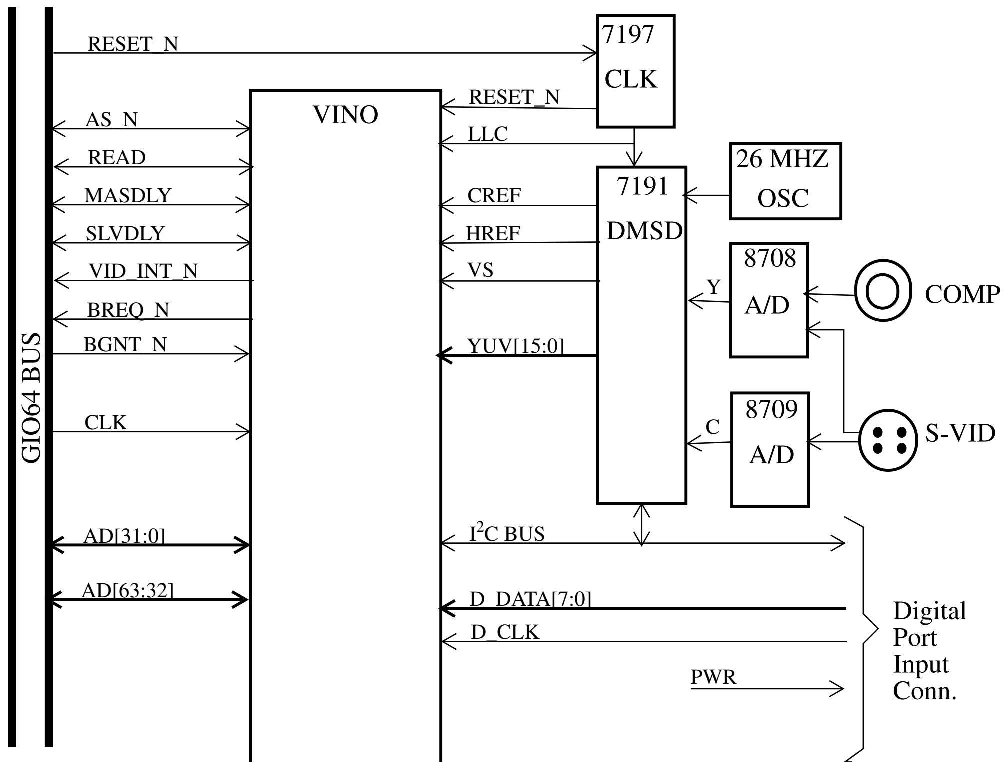


FIGURE 16. VINO System Application Block Diagram

13.2 Video Input Notes

VINO can support any combination of two video inputs simultaneously with the exception of the two analog inputs. Since the 7191 DMSD can only process the Composite OR the S-Video input at any given time, only one is available to the VINO input at any given time. Any single Video input is allowed to drive both Channels (both video processing and GIO DMA) of VINO at once.

The Selection of the two analog inputs is accomplished by writing to a control register in the 7191 DMSD chip specifically to bits GPSW1 and GPSW2. The INDY implementation of these bits produces the following table.

TABLE 28. DMSD Selection of Composite and S-Video Analog Inputs

Analog Input Selected	**GPSW2**	**GPSW1**
Composite	L	L
S Video	H	L
Not Used	L	H
Not Used	H	H

It is possible to determine if an analog video source is present by polling the HPLL bit inside the 7191 DMSD part. If the PLL is locked then a device is “broadcasting” video into the selected analog port. This is useful for diagnostics to query if an analog device is present and active prior to asking VINO to perform DMA from that source.

14.0 Bugs

1) Jump Descriptor:
A jump descriptor does not update *next_descriptor*. Subsequently, when the four descriptors fetched because of the jump are exhausted, the descriptor fetch logic uses the stale (incorrect) address in *next_descriptor* to fetch new descriptors.

The work around is to make the 4th descriptor in each set of four (4) descriptors a jump descriptor. This will ensure that *next_descriptor* will never be used to fetch descriptors except for the very first fetch.

2) End-of-field
The DMA state machine does not properly handle an end-of-field condition when the FIFO is empty. If an *x_clip_end* value is less than the active area of the horizontal line of the source video, the video clip subsystem places the last pixel of a field in the FIFO and waits for the true end of line (as opposed to the clipped end of line) before setting end-of-field. In some cases, the last pixel of a field is transferred across the bus (emptying the FIFO) before the end-of-field signal is received by the DMA state machine.

The work around is to set *x_clip_end* to the end of the active area of the horizontal line of the source video. *x_clip_start* works and both endpoints of *y_clip* work.

3) Filtering sizes 1/4, 1/5,..., 1/8
Loss of precision in accumulation in these sizes causes image quality to degrade to an unacceptable level (banding and color shift to green). This can be fixed by averaging horizontally first and then averaging vertically. Until this is fixed, these sizes should not be used. 1/4 size should be generated by filtering down from a 1/2 sized image in software.