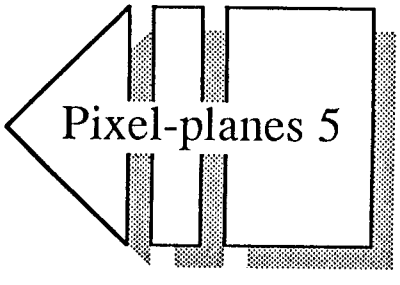
# PART I General

1. Design Objectives
2. Hardware Overview
3. Algorithms and Performance



# Part I Chapter 1

# DESIGN OBJECTIVES

## I.1.1 System Goals

We are designing Pixel-planes 5 to be an interesting research vehicle for high performance 3-D graphics architecture, an experiment in design and construction of high performance digital systems, and a platform for graphics algorithms research. We describe here the overall objectives in the Pxpl5 design, listed roughly in order of importance:

- **Fast Polygon Rendering.** For Pxpl5 to be an interesting research vehicle it must offer substantially higher performance than available commercial machines. We believe that performance in excess of 1M Phong-shaded z-buffered triangles per second (we assume 100-pixel triangles embedded in triangle strips) will be required to meet this objective by mid 1989. In addition to advanced lighting models with multiple lights, texturing and anti-aliasing will be considered essential in interactive raster graphics systems; we expect to provide these enhancements with no worse than 10x penalty in performance.

- **Programmability.** To be useful as a research vehicle, Pxpl5 must be easy to program, and all of its power should be available from a high-level language. We believe that C-language support and a graphics support library are essential to the development of new graphics algorithms. We are becoming convinced that compatibility with a subset of the PHIGS+ graphics standard will be useful; we have found from the Pxpl4 experience that familiar standards are very helpful for ease of use. Network extensibility via a subset of X.11 is desirable.

- **Packaging.** Pxpl5 will be modular and re-configurable in a variety of forms that trade performance for cost. A very high-performance configuration that meets our primary performance goals must fit within a workstation pedestal and should consume no more than 2KW of power. These limits allow the machine to live in an office environment.

- **Curved Surfaces.** While the machine's primary performance goal is expressed in terms of processing polygonal data, it is desirable that more expressive, more efficient graphics primitives be supported. We intend to provide a machine that can directly render certain curved surfaces and that provides at least rapid tesselation of surfaces described by B-splines.

- **Constructive Solid Geometry.** The new platform should provide a basis for greatly extending the speed and usefulness of our current work in editing and displaying objects described by CSG.

- **Volumetric and Image Processing Algorithms.** There is considerable local research in volumetric displays and in image processing; we want to support this work by providing a fast and flexible platform on which images can be displayed rapidly and also transformed in a variety of ways.

## I.1.2 Architectural Features

Since Pxpl5 is intended as an experimental platform for parallel processing in graphics and image processing. Its architecture is quite general, consisting of a collection of two types of processing elements, Graphics Processors (GP's) and Renderers, that communicate over a Ring network. The Ring also connects these elements to a display device and a Host workstation. A Pxpl5 system may be configured with varying numbers of GP's and Renderers to meet diverse computational and budget needs.

We describe here how the GP's and Renderers carry out, respectively, the object-oriented and pixel-oriented computation tasks in a graphics system. These processors are not, however, limited to these tasks. For example, a Renderer may be viewed as a very powerful SIMD machine capable of attacking other computing problems encountered in Ray Tracing, Image Processing, and elsewhere.

The "front end" of typical raster graphics systems deals with compact descriptions of objects to be displayed. It traverses hierarchical display lists that describe these objects, performs geometric transformations and clipping, and computes various lighting effects. Pxpl5 performs these computations with a collection of GP's operated in MIMD fashion. The general purpose nature of these processors is necessary for a variety of algorithms, including ones that sort primitives into bins corresponding to small contiguous regions of the display screen, a step crucial in applying the Renderers in parallel to the "back end" problem.

The "back end" or "renderer" of a raster system is responsible for the pixel-oriented operations of scan conversion (determining which pixels are inside a given graphics primitive), hidden-surface elimination (determining which pixels of an object are visible and which are hidden by other objects), and shading (painting color onto the pixels in such a way as to give the appearance of a "real" surface). In most raster systems the pixel color values from these computations are written into a "frame buffer" memory, from which the display is refreshed. Pxpl5's set of Renderers performs these computations with SIMD parallelism at the pixel level and coarse-grain, MIMD parallelism at the level of small, disjoint regions of the screen.

The major elements, then, of the Pxpl5 design are:

- **Graphics Processors** (GP's), floating point engines, each with considerable local code and data storage.
- **Renderers**, each a small SIMD array of pixel processors with its own controller.
- **Frame Buffer**, double-buffered, built from conventional Video RAM's, from which the display is refreshed.
- **Host Interface**, which supports communications to/from a UNIX workstation.

The initial configuration of the system will likely have 32 GP's, 8-16 Renderers, a Host Interface and a Frame Buffer. A useful system can be built with as few as one GP, one Renderer, a Host Interface and a Frame Buffer. We discuss these elements in more detail below.

## Ring Network

The bandwidth required to move primitive object descriptions from the "front end" to the renderer in our system will reach 40 MWords/second, even for relatively simple rendering algorithms. Simultaneously, pixel values must be moved from the renderer to a frame buffer at rates up to 60 MWords/sec for real-time interactive applications. Pxpl5 will have a single, unified communications structure to support both kinds of traffic. For the very large bandwidth required, a simple bus is not practical for technological reasons, so we employ a ring network in Pxpl5. The Ring is physically concentrated onto a single printed circuit board, with only short point-to-point communications paths between Ring Nodes, and can therefore be operated at quite high speeds between nodes. Initial specifications call for a one-word-wide (32 bits of data) Ring operating at 160 MHz.

Each Ring Node is a separate custom integrated circuit that serves a client board (a Ring Device) with a single 20 MHz bi-directional port. The Ring protocol allows multiple messages to be present on the Ring at one time, so each Ring Device can send a message to another Device at 20 MWords/second, even when many pairs of Devices are communicating simultaneously.

For many applications, the multiple GP's in a system will be used mainly to implement the "front end" functions, while the Renderers perform pixel-oriented operations. However, the Ring connects these elements in a very general way, so that the machine should be thought of as a heterogeneous array of parallel processors that can be used in a variety of ways.

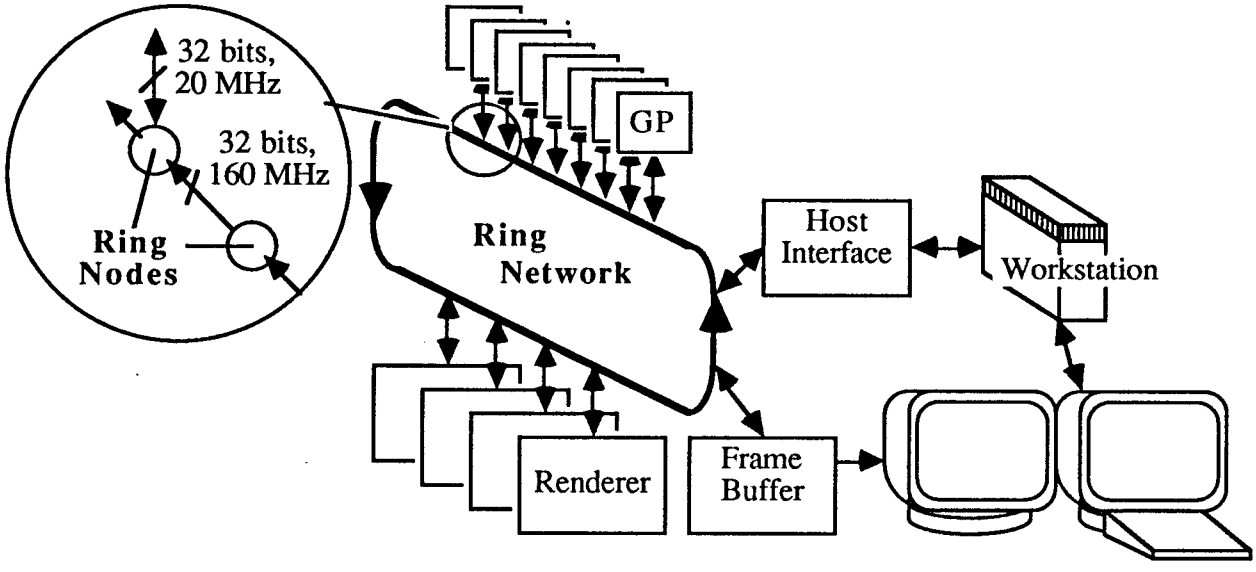



**Figure 1:** Overview of Pxpl5 System, showing Ring Network and Ring Devices that include Graphics Processors (GP's), Renderers, Host Interface, and Frame Buffer.

## Graphics Processors (GP's)

The performance levels we have set require sustained computation rates in the "front end" of several hundred MFlops, feasible today only in parallel or vector architectures. Vector processing offers considerable savings in hardware complexity and cost, but building programming tools for a vector unit is a major undertaking, far beyond the means of our small research group. We have therefore elected to build a MIMD processor based on currently available Weitek "XL" parts. This approach offers three strong advantages:

- The Weitek XL chip set comes with off-the-shelf software support in the form of a C compiler and micro-code assembler.
- Our group has considerable experience with the Weitek XL from building and programming the existing Pxpl4 Graphics Processor (also based on the Weitek XL).
- The general-purpose processors in our MIMD organization can handily sort primitives into bins for Pxpl5's parallel renderer architecture.

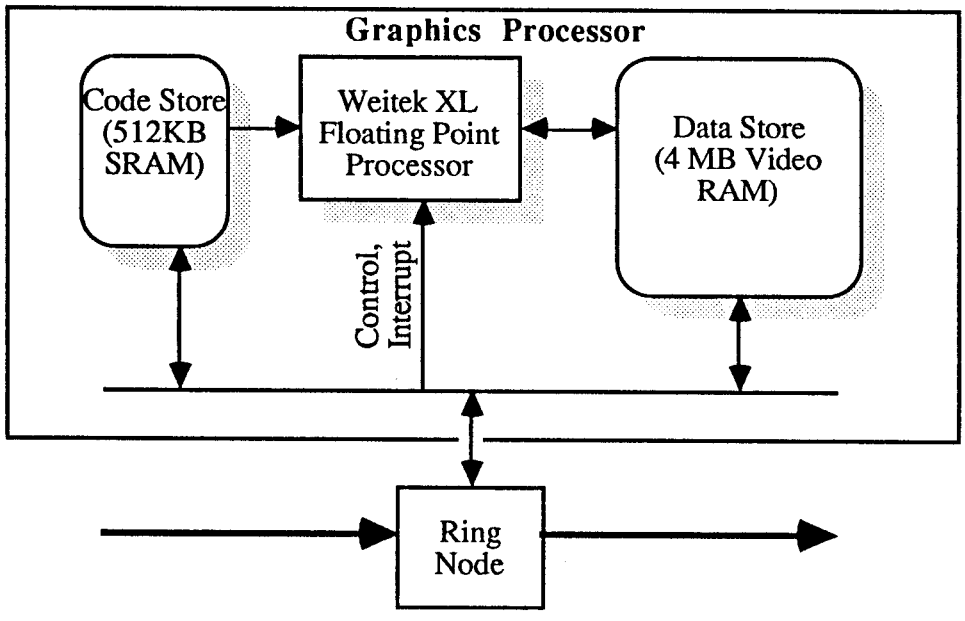



**Figure 2:** Simplified block diagram of Pxpl5 Graphics Processor.

The programming model can be relatively simple. Load sharing is accomplished by dividing an object database across the GP's, in one of several simple ways, and each GA may run exactly the same code. GP's will be programmable in the C language. So far as possible, users will have access to the machine's full capability without resorting to writing programs in machine code.

## Renderers

The "back end" of a traditional graphics machine suffers from the well-know memory bandwidth bottleneck into the frame buffer. Even for the simplest shading algorithms, the performance levels we want to attain will require memory bandwidth of 1 GWord/sec or higher (this is based on a simple analysis of memory references needed to render 1M 100-pixel triangles per second). Since this problem has been the focus of our research for many years, we feel that it is essential to demonstrate clearly the strengths of our approach, which is based on frame buffers built from logic-enhanced memories, and to extend this approach as far as possible.

Pxpl4 and previous generations are "fully instantiated" frame buffers—there is a processing element with memory, built onto a custom chip, for each displayed pixel. The screen is refreshed directly from this frame buffer. This simple realization attains performance levels in the range of 50K Gouraud-shaded triangles per second, but it requires over 2,000 custom chips even for a display of modest resolution. To achieve a 20x increase in performance for Pxpl5, to adapt the machine to a variety of new tasks, and to make the design commercially feasible, we require an approach that leverages the speed improvements of better chip technology with an improved renderer architecture:

- **Virtual Pixel Architecture.** The design of the Renderer in Pxpl5 is intended to overcome the most serious difficulties of the strict SIMD organization of Pxpl4: high cost and poor processor utilization. Improving the utilization of a SIMD array implies getting fewer processors to do more work. We are taking the approach of building multiple Renderers, each a small (perhaps 128x128) array of SIMD pixel processors with its own Image Generation Controller (IGC). Pxpl5 is organized so that each Renderer can operate independently in MIMD fashion on separate stream of graphics primitives, and it is thus parallel by graphics primitive in the Renderer as well as in the "front end". To the extent that successive geometric primitives fall into regions belonging to different Renderers, the system can render those primitives simultaneously. Simulations suggest that this approach may yield 4x or better speedups for complex scenes, which, combined with chip technology improvements, puts us within reach of our performance goals.

  This general idea was described in earlier publications as "buffered" Pixel-planes. What is new is the idea of "virtual pixels", a flexible mapping of a Renderer's virtual pixel space onto the space of the display screen. Virtual pixels are supported by a memory hierarchy, whose principal element is the "backing store" described below. A cost of this approach is the sorting problem added to the front end phase: primitives must be sorted into bins corresponding to Renderer-sized regions of the screen, then routed to the appropriate Renderer. The parallel GP's with their large data stores handle this problem nicely.

- **Backing Store.** To reduce cost, while still achieving very high performance, we will use the Renderers in a completely different way from the single Renderer of Pxpl4. Pxpl5 Renderers are not rigidly assigned to some particular region of the display screen, but can be flexibly mapped to any region. To support this mode of operation, each Renderer has a "backing store" memory, composed of commercial VRAM's, tightly coupled to the EMC's through the VRAM video port. The backing store is available, through the random access port, to the rest of the system, which can read and write pixel values in the conventional way. Soon-available 1MB VRAM's will allow storage of 4K bits per pixel with this organization. A Renderer uses this memory to save and retrieve pixel indicia, effectively allowing "context switches" when the Renderer ceases operating on one part of the screen and moves to another. A typical context switch takes about 0.5 msec, the time required to process a hundred or so

triangles, and can be completely overlapped with processing in the Renderer. For simple applications, the backing store will be used to store pixel color values for sub-regions of the screen as the Renderer completes them. When the entire image has been rendered, each of these regions is transferred in a block to the display memory in the Frame Buffer, from which the display screen is updated. Pxpl5's rendering, then, depends on a three-tiered *memory hierarchy*: The local, fast memory associated with each pixel processor on the Enhanced Memory Chip is the top-most element in this hierarchy, the backing store is the second tier, and the Frame Buffer is the third. This memory structure allows useful systems to be built with any number of Renderers; a single Renderer with only 64 custom chips, for example, would provide better performance in many applications, and more generality than does Pxpl4.

- **Quadratic Expression Evaluator.** In Pxpl5, we are replacing the linear expression tree on Pxpl4 with a quadratic expression evaluator tree that produces the function $Ax+By+C+Dx^2+Exy+Fy^2$ simultaneously at each pixel. We expect that quadratic expressions will be especially useful in rendering CSG objects and for direct rendering of quadratic patches and objects composed of conic sections.

- **Faster, Denser Chips.** Performance can readily be improved by using better semiconductor technology. The Enhanced Memory Chips (EMC's) for Pxpl5 use 1.25μ CMOS technology and are specified to run at 40 MHz for a 4-5x improvement in speed. Algorithm developers who use Pxpl4 are constantly frustrated by the relatively small amount (72 bits) of local memory at each pixel processor, so we are providing 256 bits of memory at each pixel on the Pxpl5 EMC.

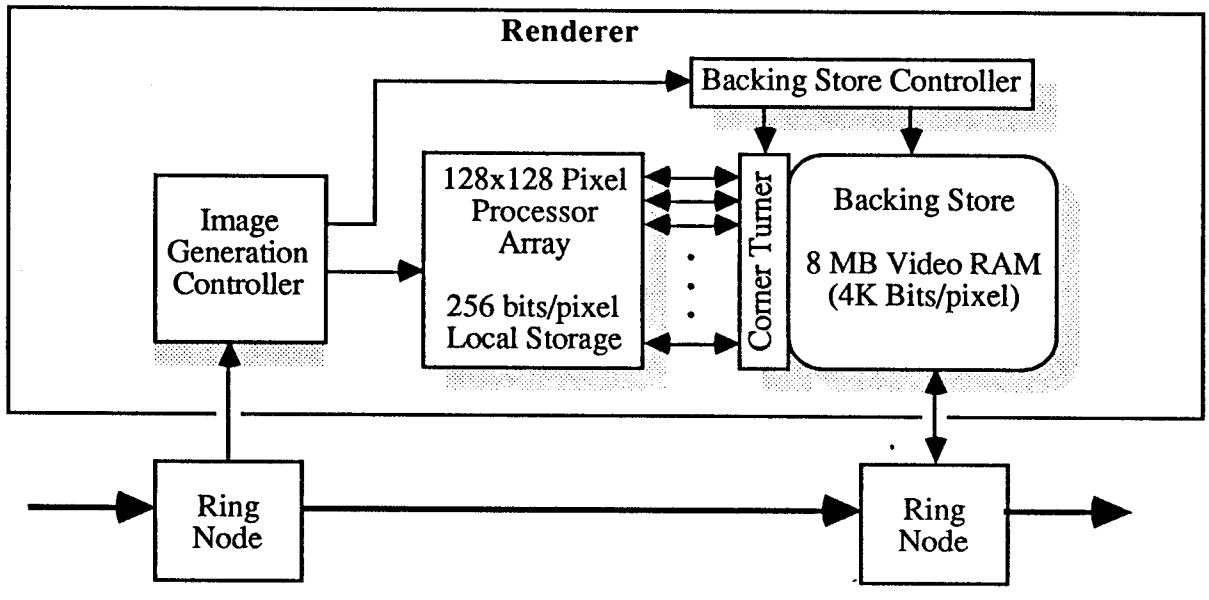



**Figure 3:** Simplified block diagram of Pxpl5 Renderer.

## Frame Buffer

The Frame Buffer is built in a fairly conventional way using Video RAM's. The Video port of this RAM array sends pixel three (R-G-B) color values (8 bits each) and a user-defined fourth value through a color lookup table to video DAC's. The video circuitry is sufficiently flexible to handle a variety of display hardware, and it provides a "gen-lock" input to support video recorders and cameras. The Frame Buffer is accessed through four Ring Nodes, to provide an aggregate bandwidth of 80 MWords/sec into the buffer. For real-time screen update (up to 60 Hz), the Video RAM random I/O port is connected to these Ring Nodes in such a way that it appears four-ported, *i.e.*, messages arriving at any of the four Nodes can go to any memory location.

## Host Interface

In its first implementation, Pxpl5 will be hosted by a UNIX workstation. Communications to the Host will be supported by a DEC DR11-style DMA interface at the Host end and the Host Interface at Pxpl5 end. The Host Interface simply executes the protocol expected by a DMA device, and allows blocks of memory data to be passed between any Device on the Ring and the Host.

# Part I Chapter 2

# HARDWARE OVERVIEW

In this chapter, we outline the architecture for Pixel-planes 5, which we believe satisfies the objectives given in Chapter I.1, with the exception of the unrealistic power requirement of that chapter.

Figure I.2.1 shows Pxpl5 as a set of devices plugged into the ports of a communications network called the *Ring*.

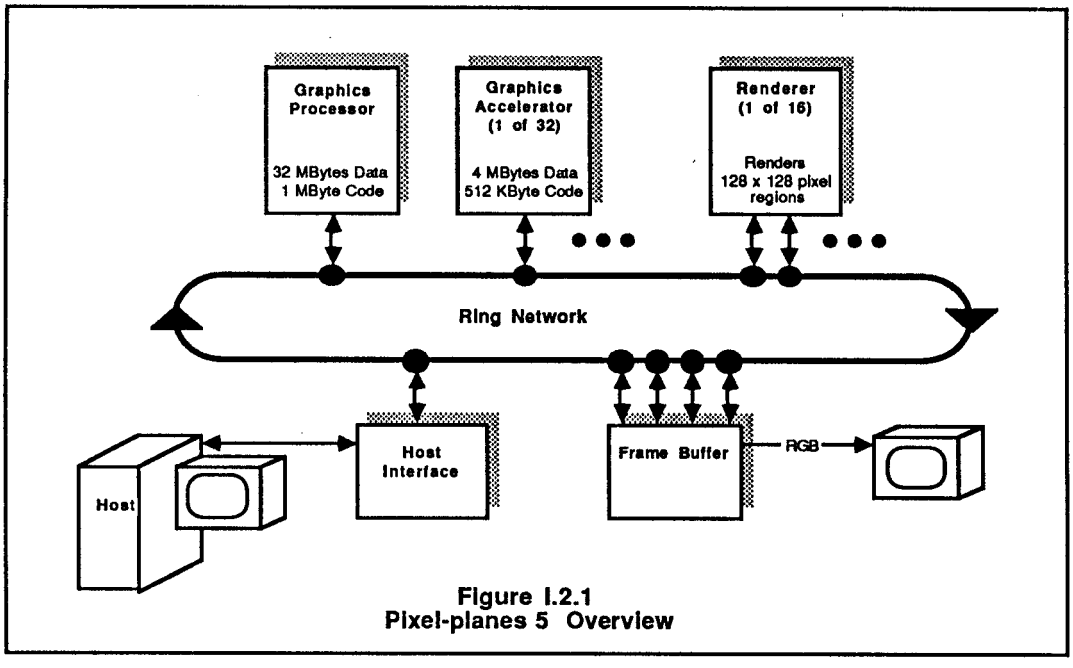



**Figure I.2.1**
**Pixel-planes 5 Overview**

## I.2.1. THE RING

The ring is a 160 MHz ring network which distributes data from port to port in a circular fashion. The ring can be thought of as a collection of cars, which move one stop clockwise every 6.25 nsec. Optimally, a port can place data in every eighth car of the Ring, giving a maximum transmission rate from any one port of 20 MWords / sec. Transmissions from different ports may interleave, as shown in figure I.2.2.

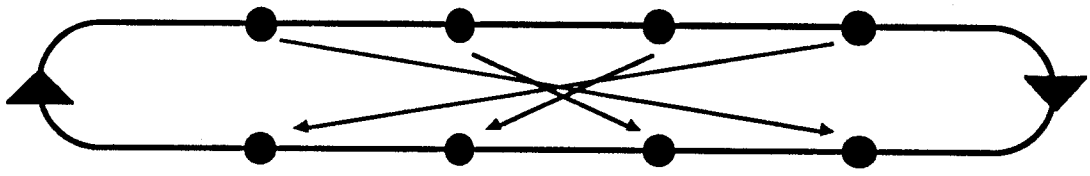


**Figure I.2—2**
**Ring Bandwidth Utilization Example**
Each thin line represents 20 MWds / sec ring traffic.

The ring has a 32-bit address space. The low 25 bits give a word address on any selected ring device. The high order 7 bits specify one of 128 ring nodes.

A device on the ring sends a packet of data to another device by writing a 32-bit destination address into the transmit buffer of its ring interface, followed by a 32-bit source address, then the data. The high order 6 bits of the destination address indicate a ring node; the low order 25 bits give a word address on the device connected to that node. The low order 25 bits of the source address optionally indicate a word address on the device sending the packet.

When the system is powered up, the Host reads a "device id" register from each ring node to determine the ring configuration. Application programs use this information to take best advantage of the devices available on the ring.

## I.2.2 RING DEVICES

The following sub-sections describe the ring devices and their interfaces to the ring.

### I.2.2—1 Host Interface (HIF)

The Host Interface allows the host workstation to broadcast and receive packets on the ring. This link is supported by a DRV-11WA DMA interface card in the host. The Host Interface also initializes the ring upon system power-up.

When a host application wishes to send a packet to a device on the ring, it sends the packet to the Host Interface via the DR-11 DMA interface. The Host Interface then places the packet on the ring, where it is routed to the intended device.

When the Host wishes to read data from a ring device, it must first send a packet to that device as outlined above. That packet instructs the device to return a packet encapsulating the requested data. The Host sets up the DR-11 to transfer the return packet back to the Host when it arrives at the Host Interface.

Devices can send asynchronous packets to the HIF which cause the HIF to interrupt the Host. This facility is intended for reporting errors and debug packets. The HIF also reports ring timeout and hardware errors to the Host.

## I.2.2—2 Graphics Processor (GP)

The Graphics Processor (GP) is intended to be the local manager in the Pixel-planes 5 system. It is based on the 10 MHz Weitek XL chip set, a Harvard processor optimized for fast floating point calculation. The GP is capable of 10 million multiply-accumulates per second. Although the GP is considered the overall system supervisor, it has no special privileges on the Ring.

When a packet arrives at the GP, its Interface Controller looks at the destination address of the packet and routes the packet to the appropriate location on the GP. Packet types received by the GP include:

Data Write: Write to data store. The packet body is written into Data Store at an address given in the low 25 bits of the packet destination address. The processor is then interrupted, and may examine a registers to determine the packet length, it's source device, and where it was written.

Data Read: Read from data store. The packet body contains two addresses *source* and *dest,* and an integer *count. Count* words are read from data store address *source* and transmitted as a packet on the ring to address *dest.*

Code Write: Write to code store. The processor is stopped, then the packet body is written into Code Store at at an address given in the low 25 bits of the packet destination address.

Code Read: Read from code store. The packet body contains two addresses *source* and *dest,* and an integer *count.* The processor is stopped, then *Count* words are read from code store address *source* and transmitted as a packet on the ring to address *dest.*

Stop: The processor is stopped.

Start: The processor is reset and begins execution at location zero.

Status Read: Read the GP device id and status. The packet body contains an address*dest.* The device id and status are transmitted in a two word packet to address *dest.*

The GP can send packets out of its data store by writing the source address, destination address, and length of the packet into control registers of the interface controller. The controller interrupts the GP upon completion of the packet transmission.

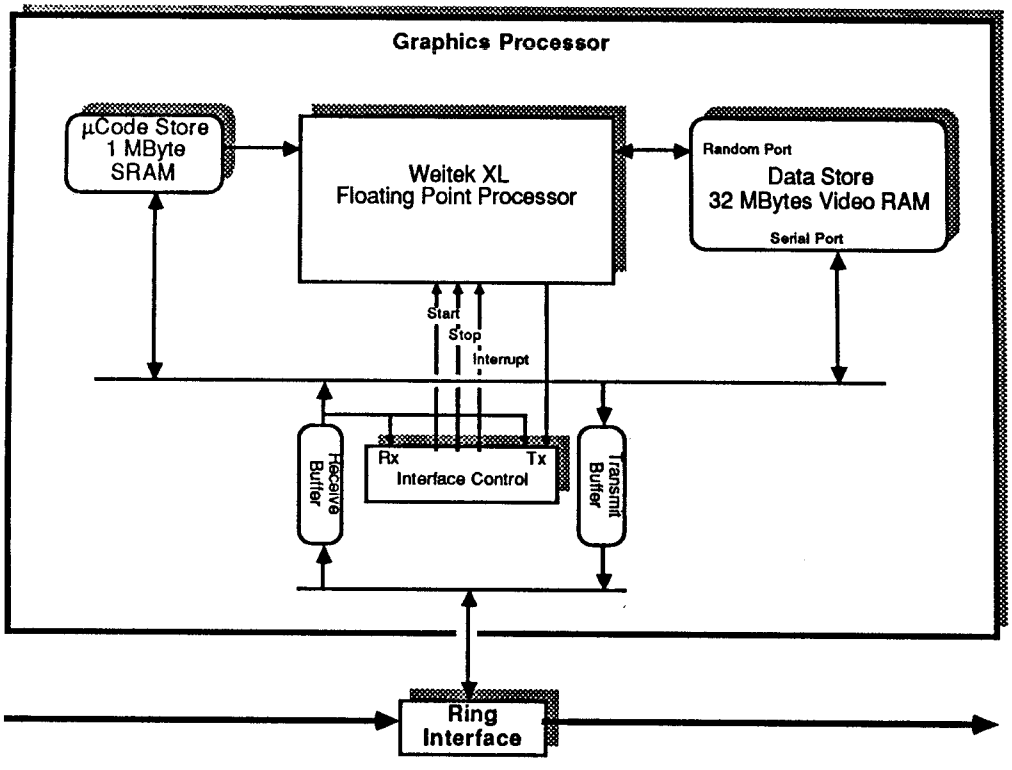



**Figure I.2.3**
**Graphics Processor**

## I.2.2—3 Graphics Accelerators

The Graphics Accelerators (GA's) are copies of the GP with smaller code and data stores. Each GA has 4 MBytes of data store and 512 KBytes of code store.

In triangle rendering applications, the GA's are used in parallel to transform geometric primitives into linear expression representations, and to sort those transformed primitives into bins corresponding to regions of the frame being rendered. The GA's may also be used to apply texture table lookups to images.

A high end Pixel-planes 5 system will have 32 GA's on the ring. At 32 KΔ's/sec per GA, this will give the target performance of one million Δ's/sec.

## I.2.2—4 Renderers

A Renderer is a 128 x 128 array of 40 MHz bit serial ALU's, each with 256 bits of tightly coupled local data store. A quadratic expression evaluator (the Tree) delivers bi-quadratic expressions in x and y simultaneously to each ALU. The ALU's and the Tree and are

controlled in SIMD fashion by the Image Generation Controller (IGC).

The local store of each ALU is supplemented with 4096 bits of loosely coupled backing store, organized as 128 32-bit registers per ALU. The IGC controls the parallel loads and stores of 32-bit words from backing store to local store. A single backing store operation, dispatched by the IGC, loads or stores the low order 32 bits of local store to a specified backing store register array. After dispatching a backing store operation, the IGC can continue to execute ALU commands as the backing store operation proceeds concurrently. It can wait for completion of the backing store operation by monitoring a done flag from the BSC. One backing store operation will take slightly more than 4096 image generation cycles, about 100 μsec.

A Renderer has two ports onto the ring. The first port receives commands for the IGC; the second is used to read and write the backing store. From the ring, the backing store appears as 128 separate 128 x 128 arrays of 32-bit values. In a typical rendering application, these arrays will be transferred over the ring to the Frame Buffer to become pixel regions.

Packet types received by a Renderer's IGC port include:

Command Buffer Write: The packet body is placed in the IGC command buffer.

Status Read: Read the IGC port id and status. The packet body contains an address *dest*. The port id and status are transmitted over the ring to address *dest*.

Packet types received by a Renderer's backing store port include:

Backing Store Write: Write to backing store. The packet body is written into Backing Store at an address given by the low order bits of the packet destination address. This address specifies one of the 128 backing store register arrays.

Backing Store Read: Read from backing store. The packet body contains two addresses *source* and *dest*. A register array given by *source* is read from backing store and transmitted as a packet on the ring to address *dest* (typically a Frame Buffer address).

Status Read: Read the backing store port id and status. The packet body contains an address *dest*. The port id and status are transmitted over the ring to address *dest*.

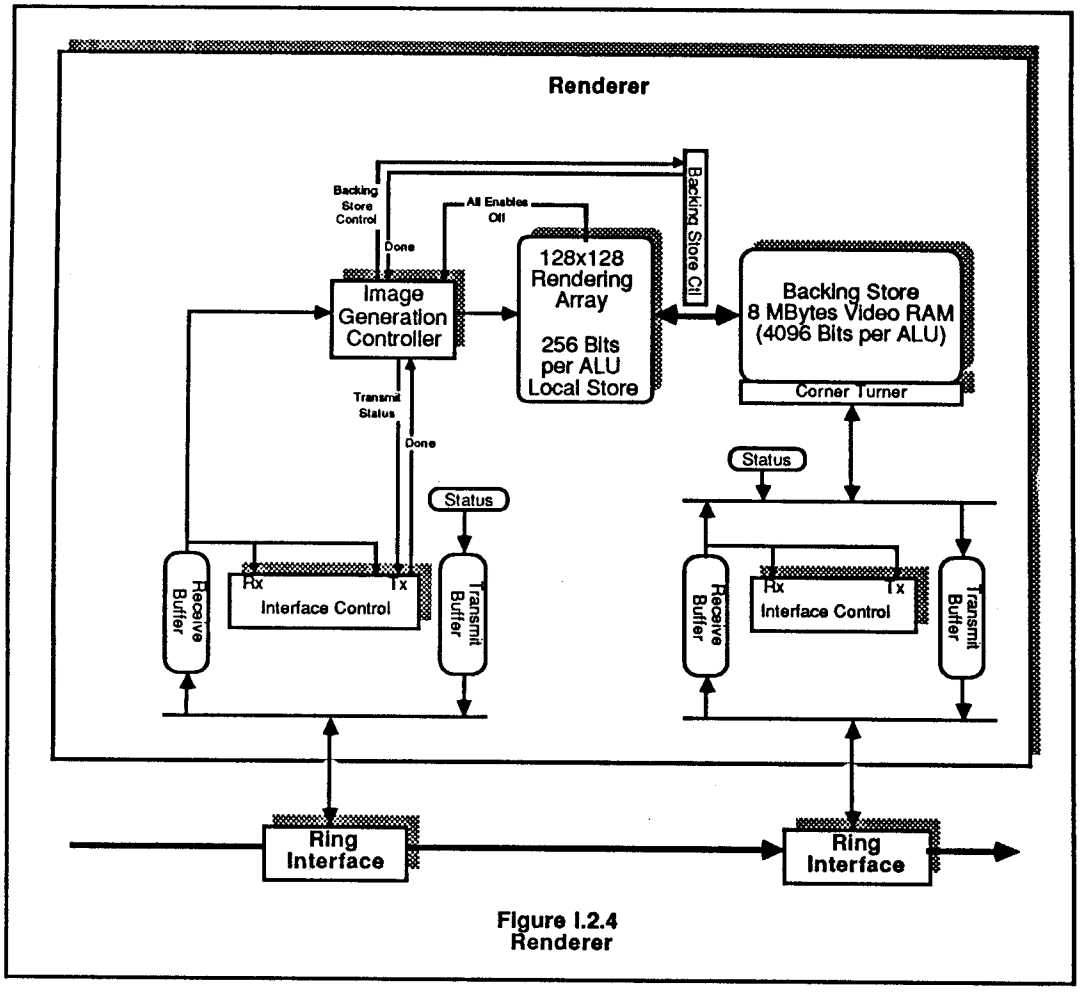



**Figure I.2.4**
**Renderer**

## I.2.2—5 Frame Buffers

A Frame Buffer is a 2048 x 2048 x 32 bit memory array, with four ports on the ring and a video port out to a 1280 x 1024 monitor. The four ring ports allow up to four ring devices to simultaneously read or write 20 MWords / sec into the Frame Buffer Store. A Pixel-planes 5 system will support multiple Frame Buffers; two Frame Buffers will support stereo imaging nicely.

From the ring, a Frame Buffer Store is organized as a 32 x 16 array of 64 x 128 x 32 bit regions. These regions are the same size as the ALU arrays in the Renderers, so a Renderer register array can be copied directly to or from a Frame Buffer region over the ring.

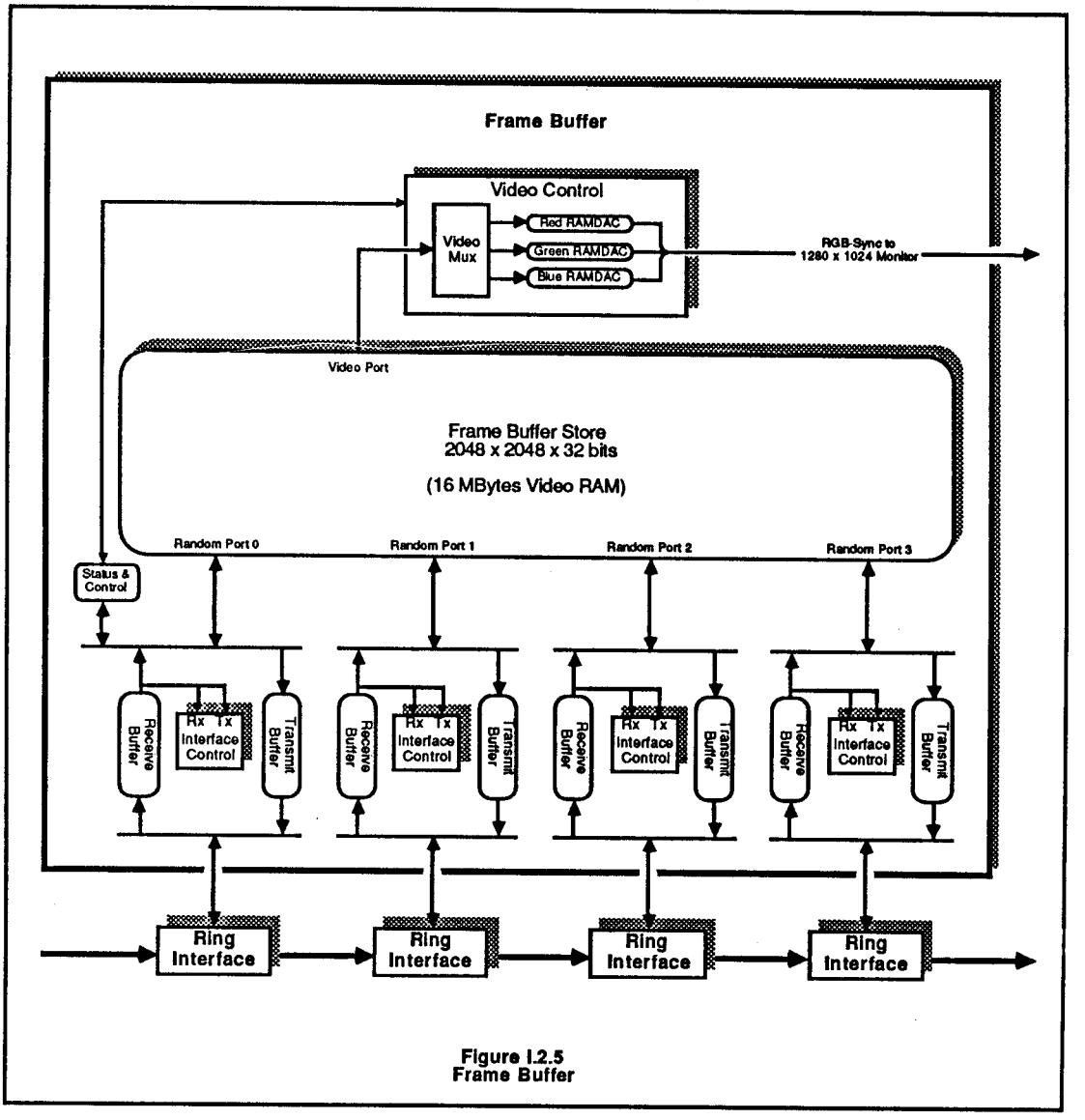



**Figure I.2.5**
**Frame Buffer**

Bytes 0, 1, and 2 of the Frame Buffer are denoted the Red, Green, and Blue bytes, respectively. These three bytes are streamed from a window in the Frame Buffer Store into the video RAMDAC's, which perform the red, green, and blue color lookups and drive the video output.

The size and location of the display window within the Frame Buffer is controlled by a control register written from the ring. The window can be one of three sizes: 1280 x 1024, 640 x 512, or 320 x 256. Its location within the Frame Buffer, also determined by a control register, must be aligned on region boundaries.

The RAMDAC's are 12 bit lookup tables driving 12 bit DAC's. The three video bytes can be mapped into the video RAMDAC's in three different ways, depending on the lookup mode:

RGB: The red, green, and blue bytes are streamed into the red, green, and blue RAMDAC's, respectively. The low order nibbles of the RAMDAC inputs are set to zero.

Mono8: The red byte is streamed into all three RAMDAC's for 8 bit monochrome. The low order nibbles of the RAMDAC inputs are set to zero.

Mono12: The red byte and the low order nibble of the green byte are streamed together into all three RAMDAC's for 12 bit monochrome. The green bits are the least significant.

## I.2.3 PACKAGING

The backplane has 18 slots, 4 ring ports to a slot. Each ring node is implemented in a custom CMOS part on the backplane, connected into a ring as shown in figure 1.2.6. The ring nodes (squares in the diagram) are connected to the rectangular connectors to ring devices. The connectors are arranged in 18 slots of four connectors. The 18 slot limit is imposed on us by a maximum backplane size; the ring node parts themselves will support a ring with 128 nodes.

A ring device may be built on a quarter, half, three-quarter, or full height board. A full height device will occupy a full slot, and may have up to four ring ports. A quarter height device will occupy one fourth of a slot, and may have only one ring port. The half and three-quarter height devices work similarly. For mechanical integrity, ring devices of less than full height must be bolted together to form a full height board before being plugged into the backplane.

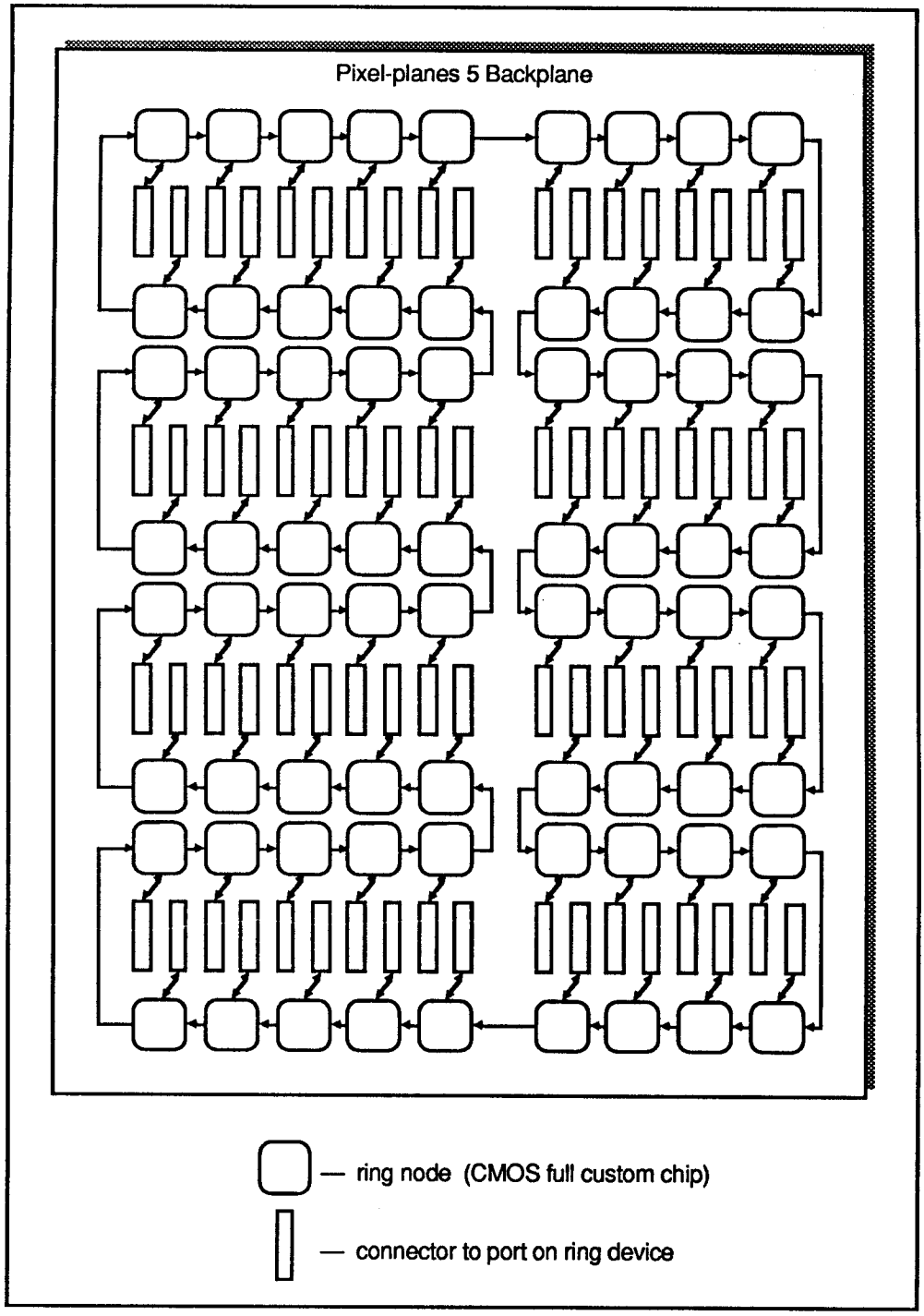



Figure I.2.6 Backplane

The Host Interface occupies a quarter height board; The GP occupies a half height board. Together, these two devices occupy one slot. A Graphics Accelerator is implemented on a

quarter-height board, a Renderer occupies one half-height board, and the Frame Buffers each occupy a full-height board.

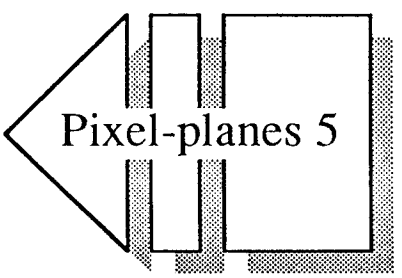
A full blown system might have the following ring devices:

| | |
|---|---|
| Host Interface / GP | 1 slot |
| 32 GA's | 8 slots |
| 16 Renderers | 8 slots |
| 1 Frame Buffer | 1 slots |
| total: | 18 slots |

Other organizations are certainly possible. For example, an additional Frame Buffer could be added at the expense of two Renderers for stereo imaging.

# PART II Power, Packaging, & Cooling

1. Ring Network
2. Backplane
3. Power Supplies
4. Cabinet Mechanicals



# II.1.3. RING ARCHITECTURE

This section describes the Pxpl5 ring network as follows:

| | |
|---|---|
| Subsection 1 | Describes the ring as a "multi-ring." |
| Subsection 2 | Packet format and ring addresses. |
| Subsection 3 | Error packet format & Board ID's. |
| Subsection 4 | Hardware interface to the ring. |
| Subsection 5 | Port connector pinout. |

## II.1.3—1 The Ring

The Pxpl5 ring is an 8-way multi-ring with a 160 MHz control path and 8 32-bit channels time-multiplexed onto a 160 MHz 32-bit data path. All device interfaces to the ring are synchronous. The ring's behavior is defined in the previous section on the multi-ring.

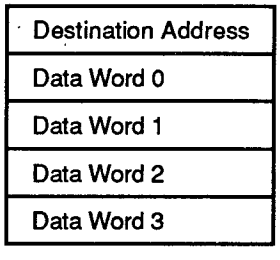
A device's interface to the ring is called a *port*. A device may have more than one port, depending on its board height and needs. A single height board has one port, a double height board has two ports, and one can imagine schemes with sandwiched boards allowing devices with more than two ports.

One device sends data to another by clocking a *packet* into the *transmit interface* of one of its ports. The packet format includes the destination port address of the packet, allowing the ring to transfer the packet to the *receive interface* of the intended port on the receiving device.

## II.1.3—2 Packet Format and Ring Addresses

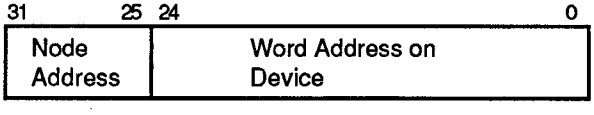
A ring packet consists of a 32-bit destination address, then zero or more 32-bit data words.

*Example 4 word Ring Packet:*




Ring addresses have two fields: a 7-bit node address and a 25-bit word address. Thus the ring supports up to 128 nodes, each with a 32 Mword address space.

*Ring Address:*



## II.1.3—3 Error Packets and Board ID's

This sub-section specifies the format of an *error packet* to be sent in the event of a low level system error. Errors falling into this category might include

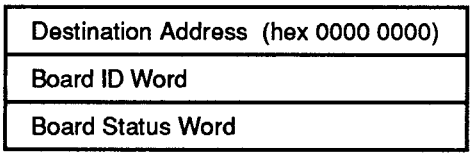
- Receiver buffer overflow (message too long).
- IGC Hung on a renderer.
- Invalid Code or Data Address (Bus Error) on a GP.

This error handling scheme is not to be invoked upon high level errors such as an overflowing bin in the Rendering Control System, or an invalid PPHIGS command detected at a GP.

When a device detects an error, it should send a packet to the Host Interface device, which always resides at ring address zero. The Host Interface and host software should recognized packets sent to address zero as error packets.

The following error packet format should be used:

*Error Packet:*




The low-order byte of the Board ID word is an eight bit board serial number. The high 24 bits of the Board ID word are undefined. Each application board is assigned a unique serial number when it arrives from fabrication and assembly. The serial numbers are stored in a system file for access by system software.

## II.1.3—4 Ring Port Specification

A double-height board has two independent ring ports. The signals for one port are carried from the ring board to the application board via two 160-pin DIN connectors, one per port. These signals can be broken into three groups: miscellaneous, receive, and transmit signals.

*Signal Timing*
All ring interface signals are synchronous with respect to the 20 MHz clock Ring20ClkH. The two numbers following device-originating signals are the setup and hold times in nsec with respect to Ring20ClkH. The two numbers following ring-originating signals are the min and max prop delays in nsec for those signals with respect to Ring20ClkH. Note that these times are with respect to an ECL clock signal, and must be adjusted to account for translating the clock to TTL levels.

*Signal Termination*
Application boards may be mounted on 15" extender boards for debugging. To this end, the application board should terminate all TTL signals received from the ring with 100Ω terminators (160/240 Thevenin Equivalent). The ring board will do likewise for all TTL signals it receives from the application board.

The application board should route the ECL clocks on 50Ω traces and terminate them to VTT through 50Ω.

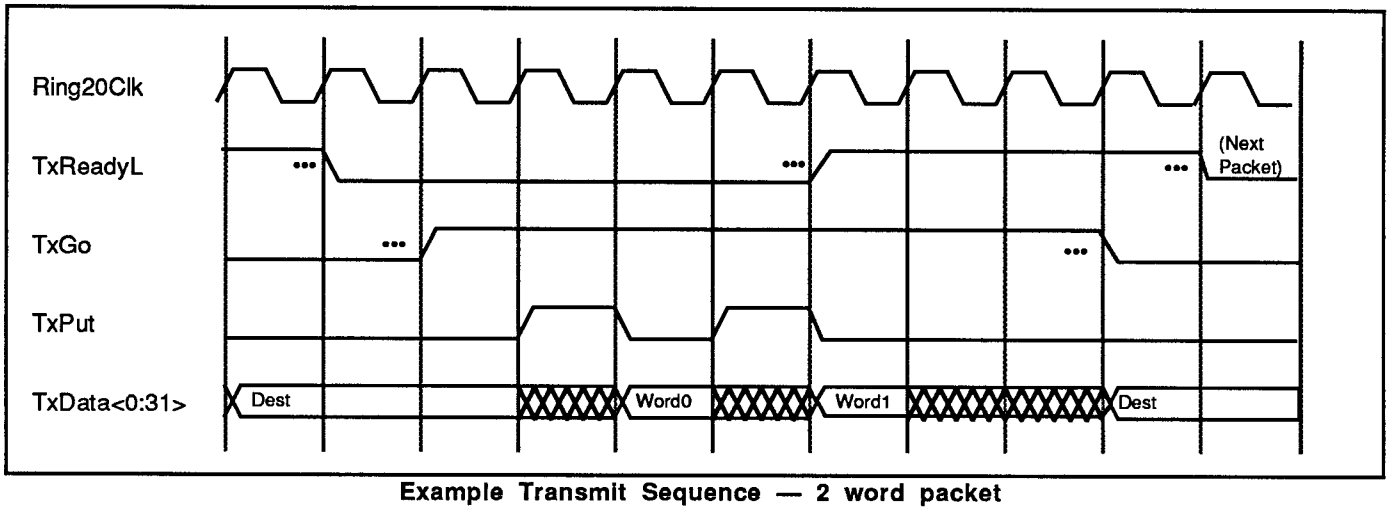
*Signal Polarity*
All ring signals are active high unless suffixed with 'L', in which case they are active low.

### II.1.3—4.1 Miscellaneous Signals

| | | |
|---|---|---|
| Ring40ClkH<br>Ring40ClkL . . . | | 40 MHz differential ECL clock, provided for 40 MHz devices such as the Renderer. |
| Ring20ClkH<br>Ring20ClkL . . . | | 20 MHz differential ECL clock, the basis for all ring interface timing. Ring20ClkH is phase-aligned with Ring40ClkH. |
| Reset . . . . | [5:14] | Asserted by ring to reset devices. Will be asserted for at least 2 µsec. After the ring de-asserts Reset, ring devices should be in an idle state, ready to receive a packet. |
| ResetGenL . . | [Async] | The Host Interface asserts this signal on node0 for at least 2 µsec to boot the ring and all attached devices. All other devices leave this signal unconnected. |
| ResetEnabL . | [Static] | The Host Interface ties this signal to GND on node0; all other devices leave this signal unconnected. (ResetGenL and ResetEnabL are ignored on node1.) |
| OccupiedL . . | [Static] | All devices tie this signal to GND. |

## II.1.3—4.2 Transmit Signals

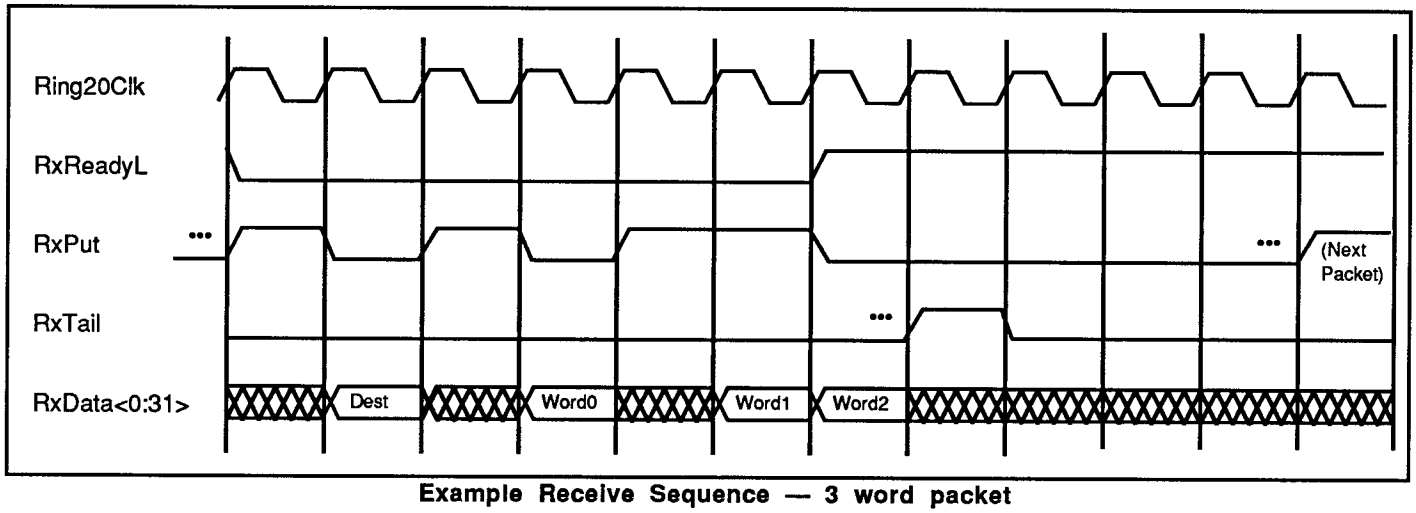
| | | |
|---|---|---|
| TxReadyL | [28:4] | The transmitting device asserts this signal when it is ready to transmit a packet, and de-asserts it on or after the last data word of the packet.<br><br>If the intended receiver device is hung (never ready for a packet), the ring will never assert TxGo to the transmitting device. In this case, the transmitting device can report the problem by de-asserting TxReady to abort the failed transmission, then sending an error packet to the Host Interface in the usual fashion. |
| TxGo | [5:22] | The ring asserts this signal when it has acquired the receiver and a channel for data transmission. The device may then clock in data with the TxPut signal.<br><br>The ring deasserts TxGo no sooner than three cycles after the device deasserts TxReady. The device must not reassert TxReady (for a new packet) until the cycle after TxGo is deasserted. |
| TxPut | [15:0] | The device clocks the rest of the packet across the interface by asserting this signal one cycle before each data word. TxPut is a level sensitive signal, and may be asserted on consecutive cycles for consecutive data words. TxPut must not be asserted after TxReady has been deasserted for a packet. |
| TxData<0:31> | [9:0] | Transmit data asserted by device— consists of packet destination address and data words.<br><br>The device places the packet destination address on TxData one or more cycles before asserting TxReady, and leaves it there until the ring asserts TxGo. |




**Example Transmit Sequence — 2 word packet**

## II.1.3—4.3 Receive Signals

| | | |
|---|---|---|
| RxReadyL | [28:4] | The device asserts this signal to indicate that it can receive a "maximum length" packet. This length varies from device to device.<br><br>RxReady is not monitored by the ring after the node begins receiving a packet. Typically this signal is a mousetrapped HalfFull flag from a IDT FIFO part. |
| RxPut | [2:7] | The ring asserts this signal one cycle before the destination address and each data word. The ring may deassert RxPut on any cycle in the receive sequence to skip that cycle. |
| RxTail | [2:7] | The ring asserts RxTail to indicate the end of the packet. It may be asserted as soon as the cycle after the last data word.<br><br>The ring will assert RxPut for the next packet no sooner than 4 cycles after RxTail. |
| RxData<0:31> | [2:7] | Receive data asserted by ring— consists of packet destination address and data words. |




**Example Receive Sequence — 3 word packet**

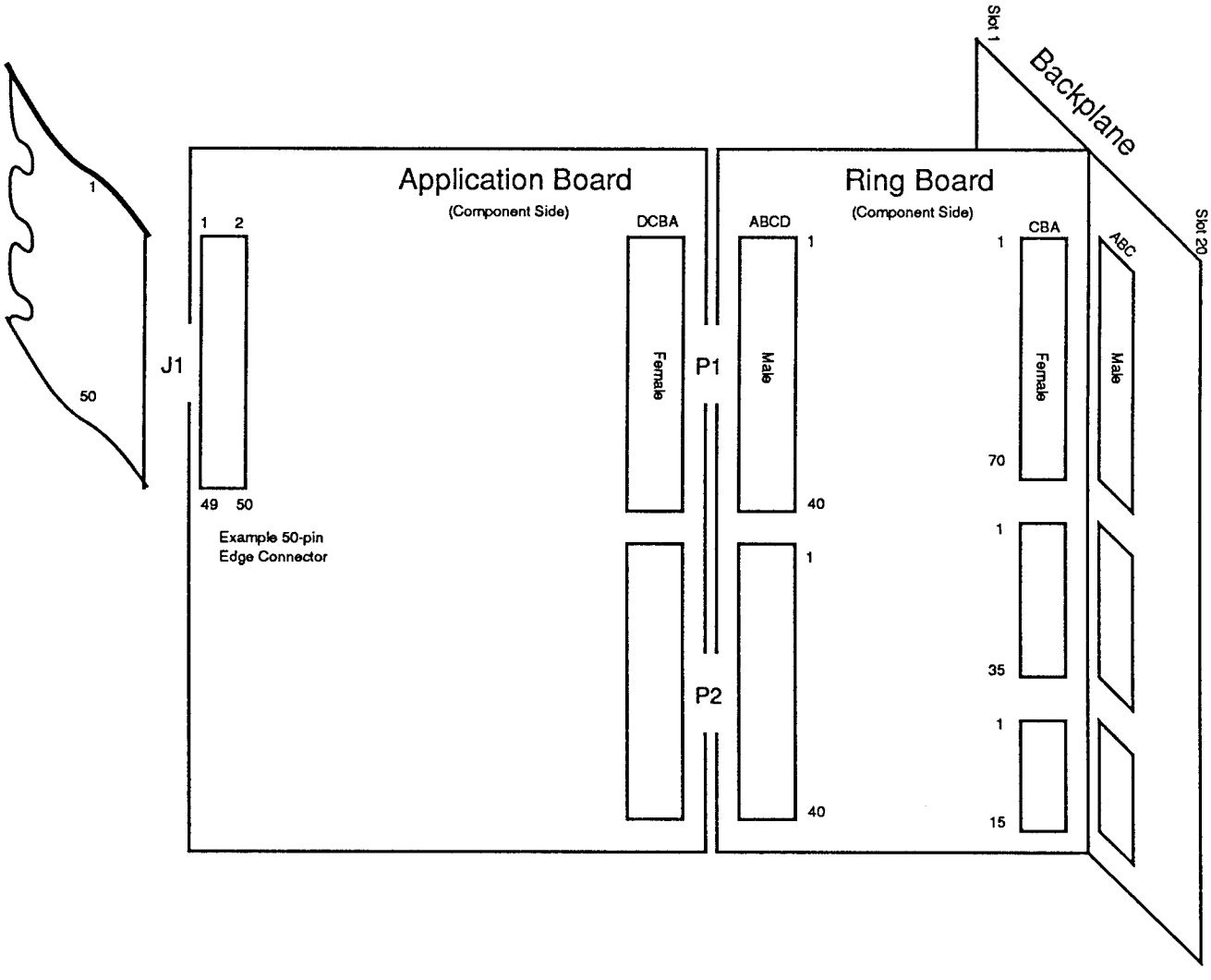
## II.1.3—5 Ring Port Connector

The following pinout applies to the connection to both ring nodes, on connectors P1 and P2 for node0 and node1, respectively.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | RxData0 | VCC | RxData1 | GND |
| 2 | VCC | RxData2 | GND | RxData3 |
| 3 | RxData4 | VCC | RxData5 | GND |
| 4 | VCC | RxData6 | GND | RxData7 |
| 5 | RxData8 | VCC | RxData9 | GND |
| 6 | VCC | RxData10 | GND | RxData11 |
| 7 | RxData12 | VCC | RxData13 | GND |
| 8 | VCC | RxData14 | GND | RxData15 |
| 9 | RxData16 | VCC | RxData17 | GND |
| 10 | VCC | RxData18 | GND | RxData19 |
| 11 | RxData20 | VCC | RxData21 | GND |
| 12 | VCC | RxData22 | GND | RxData23 |
| 13 | RxData24 | VCC | RxData25 | GND |
| 14 | VCC | RxData26 | GND | RxData27 |
| 15 | RxData28 | VCC | RxData29 | GND |
| 16 | VCC | RxData30 | GND | RxData31 |
| 17 | TxData0 | VCC | TxData1 | GND |
| 18 | VCC | TxData2 | GND | TxData3 |
| 19 | TxData4 | VCC | TxData5 | GND |
| 20 | VCC | TxData6 | GND | TxData7 |
| 21 | TxData8 | VCC | TxData9 | GND |
| 22 | VCC | TxData10 | GND | TxData11 |
| 23 | TxData12 | VCC | TxData13 | GND |
| 24 | VCC | TxData14 | GND | TxData15 |
| 25 | TxData16 | VCC | TxData17 | GND |
| 26 | VCC | TxData18 | GND | TxData19 |
| 27 | TxData20 | VCC | TxData21 | GND |
| 28 | VCC | TxData22 | GND | TxData23 |
| 29 | TxData24 | VCC | TxData25 | GND |
| 30 | VCC | TxData26 | GND | TxData27 |
| 31 | TxData28 | VCC | TxData29 | GND |
| 32 | VCC | TxData30 | GND | TxData31 |
| 33 | RxPut | VCC | RxTail | GND |
| 34 | VCC | RxReadyL | GND | TxPut |
| 35 | TxReadyL | VCC | TxGo | GND |
| 36 | VCC | Reset | OccupiedL | ResetGenL |
| 37 | ResetEnabL | VTT | VEE | VEE |
| 38 | VTT | VTT | VEE | VEE |
| 39 | Ring20ClkH | GND | Ring40ClkH | GND |
| 40 | Ring20ClkL | GND | Ring40ClkL | GND |

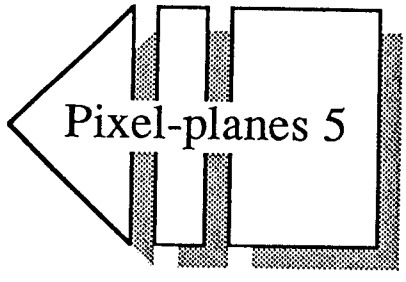
# Pixel-Planes 5 Board and Connector Conventions

tg 7/18/89



# PART III Application Boards

1. Host Interface Board
2. Graphics Processor Board
3. Graphics Accelerator Board
4. Renderer Board
5. Frame Buffer Board



## Part III Chapter 1

# HOST INTERFACE

### 1.1 Introduction

The Host Interface provides a link between the Pixel-planes5 host, a Sun-4, and the devices attached to the ring network, such as renderer boards and graphics processor boards. The Host Interface (HIF) passes data to and from the host via a VME prototype board plugged into the Sun. The HIF and host driver are responsible for booting the ring network. Information concerning the VMEbus may be obtained from The VMEbus Specification.

### 1.2 Implementation Overview

#### 1.2.1 Division of Function

The Host Interface for PXPL5 is implemented as two physically separate circuit boards. The VME prototype board contains buffers for the VMEbus signals and decode logic to identify VMEbus cycles which are intended for the Host Interface. The VME board also contains the data transceivers and the logic to generate certain VMEbus signals when the other board is not operational. The remainder of the Host Interface logic which includes data transmission control, data receipt control, register control, and interrupt control, resides on a second board, referred to as the system board. The system board is connected to the VME prototype board via two ribbon cables and is also connected to a PXPL5 ring network board. The following signals are transferred between the protoboard and the system board. **Figure 1** is a block diagram of the two boards which comprise the Host Interface.

**D[0:31]**

These signals comprise the 32-bit bidirectional data bus between the protoboard and the system board. Bit 0 is least significant and bit 31 is most significant.

**WrPktCycleUL**

This signal is low-active and is generated on the protoboard. When active, the current VME cycle is a "write packet" cycle.

**RdPktCycleUL**

This signal is low-active and is generated on the protoboard. When active, the current VME cycle requests the contents of a receive buffer.

**WrCSRCycleUL**

This signal is low-active and is generated on the protoboard. When active, the current VME cycle is for writing bits 0-2 of the *Control and Status Register.*

**RdCSRCycleUL**

This signal is low-active and is generated on the protoboard. When active, the current VME cycle requests the contents of the *Control and Status Register.*

**EOPCycleUL**

This signal is low-active, is generated on the protoboard and indicates to the Host Interface that the last word of the packet being transmitted to the ring network has been transferred.

**BSysResetL**

This signal is low-active and is the buffered version of the VME system reset. It is subsequently mousetrapped on the system board.

**BDS0L, BDS1L**

These signals are the buffered version of the VME Data Strobes. They are subsequently mousetrapped on the system board.

**SysIACKINL**

This signal is low-active and is generated on the protoboard. It indicates that the host is acknowledging the interrupt request of the Host Interface.

**SysIRQ2L**

This low-active signal is generated on the system board and is buffered to become the VMEbus interrupt request line 2 on the protoboard.

### SysDTACK

This high-active signal is generated by the system board and is used to generate the VMEbus Data Acknowledge on the protoboard.

### ALIVE

This high-active signal is generated on the system board and indicates that the system board is operational.

| Signal Name | Connector J1 Pin Number | Signal Name | ConnectorJ1 Pin Number |
|---|---|---|---|
| D[0] | A1 | D[16] | A17 |
| D[1] | A2 | D[17] | A18 |
| D[2] | A3 | D[18] | A19 |
| D[3] | A4 | D[19] | A20 |
| D[4] | A5 | D[20] | A21 |
| D[5] | A6 | D[21] | A22 |
| D[6] | A7 | D[22] | A23 |
| D[7] | A8 | D[23] | A24 |
| D[8] | A9 | D[24] | A25 |
| D[9] | A10 | D[25] | A26 |
| D[10] | A11 | D[26] | A27 |
| D[12] | A12 | D[27] | A28 |
| D[12] | A13 | D[28] | A29 |
| D[13] | A14 | D[29] | A30 |
| D[14] | A15 | D[30] | A31 |
| D[15] | A16 | D[31] | A32 |

| Signal Name | Connector J2 Pin Number | Signal Name | Connector J2 Pin Number |
|---|---|---|---|
| GND | A1 | RdCSRCycleUL | A17 |
| GND | A2 | GND | A18 |
| GND | A3 | EOPCycleUL | A19 |
| GND | A4 | GND | A20 |
| SysDTACK | A5 | BDS0L | A21 |
| GND | A6 | GND | A22 |
| SysIRQ2L | A7 | BDS1L | A23 |
| GND | A8 | GND | A24 |
| ALIVE | A9 | BSysResetL | A25 |
| GND | A10 | GND | A26 |
| WrPktCycleUL | A11 | SysIACKINL | A27 |
| GND | A12 | GND | A28 |
| RdPktCycleUL | A13 | GND | A29 |
| GND | A14 | GND | A30 |
| WrCSRCycleUL | A15 | GND | A31 |
| GND | A16 | GND | A32 |

Pins B1-B32 are connected to GND on both connectors.

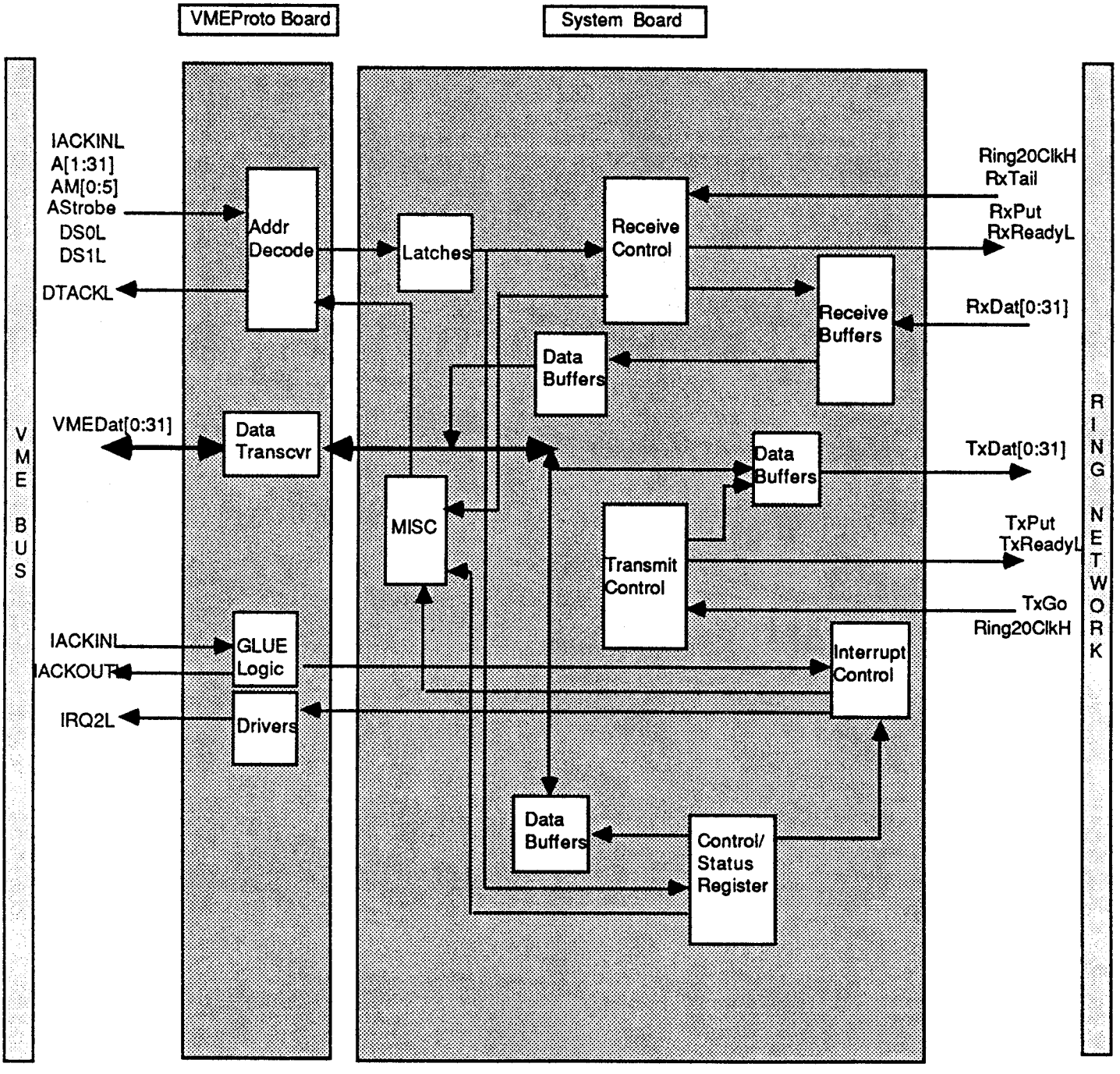

VMEProto Board
System Board
VME BUS
RING NETWORK
IACKINL
A[1:31]
AM[0:5]
AStrobe
DS0L
DS1L
DTACKL
Addr Decode
Latches
Receive Control
Ring20ClkH
RxTail
RxPut
RxReadyL
Receive Buffers
RxDat[0:31]
Data Buffers
VMEDat[0:31]
Data Transcvr
Data Buffers
TxDat[0:31]
MISC
Transmit Control
TxPut
TxReadyL
TxGo
Ring20ClkH
IACKINL
GLUE Logic
IACKOUT
Interrupt Control
IRQ2L
Drivers
Data Buffers
Control/ Status Register

**Figure 1**

### 1.2.2 Functional Description

The Host Interface is memory-mapped into the Sun-4's address space and occupies eight contiguous addresses within that space. Each of the Host Interface's "addresses" corresponds to a specific function to be performed by the HIF. These functions include: write packet data, read packet data, write the *Control and Status Register,* read the *Control and Status Register*, and indicate that the last word of a data packet has been written.

The Host Interface has one *port* to the ring network, implemented as separate *transmit* and *receive interfaces*. The packet transfer scheme of the multi-ring is supported by the host interface. Thus, data is transferred between the ring network and the host in the form of packets comprised of a destination address followed by zero or more data words.

Also contained on the HIF is the circuitry for interrupting the host under certain conditions. These conditions are: receiving an entire packet from the ring, acquiring a channel for data transmission by the ring network, and detecting a receive buffer overflow. *Interrupt Request Line 2* of the VMEbus is used to alert the host to one (or more) of these conditions. The host must read the *Host Interface Control and Status Register* to determine the cause of the interrupt.

The Host Interface contains a *Control and Status Register* (*CSR*) which contains information about the status of data transfers, status of the ring network, and the setting of HIF control parameters (See Section **1.6**). A special register on the prototype board, the *Test Register*, is used exclusively for intitial debug of the Host Interface (See Section **1.7**).

## 1.3 Memory Map

The Host Interface is memory-mapped into the Sun-4 and occupies eight (8) contiguous addresses within a page of memory. The upper (19) nineteen VMEbus address bits define which page is mapped to the HIF and are selected by switches on the HIF VME board. Since address bits 5 through 12 are not specified in the HIF address map, the eight addresses are duplicated throughout the page.

The host accesses the Host Interface by performing memory read or write cycles to an address within the HIF memory map. The HIF monitors the address lines and the address modifier lines of the VMEbus and performs the appropriate function when the HIF is the target of a particular cycle. The functions or commands that correspond to the HIF addresses are as follows:

| A31-A13 | A12-A5 | A4A3A2 | Address Modifier (Hex) | *WriteL** | Function |
|---|---|---|---|---|---|
| HIF Page | X | 000 | 09,0A,0D,0E | 0 | Write Packet Data |
| HIF Page | X | 000 | 09,0A,0D,0E | 1 | Read Packet Data |
| HIF Page | X | 001 | 09,0A,0D,0E | 0 | End of Packet |
| HIF Page | X | 001 | X | 1 | Reserved |
| HIF Page | X | 010 | 09,0A,0D,0E | 0 | Write CSR Data |
| HIF Page | X | 010 | 09,0A,0D,0E | 1 | Read CSR Data |
| HIF Page | X | 011 | 09,0A,0D,0E | 0 | Write Test Register |
| HIF Page | X | 011 | 09,0A,0D,0E | 0 | Read Test Register |
| HIF Page | X | 1XX | X | X | Reserved |

**Table 1**

As indicated by the Address Modifier bits, the HIF only responds to non-block transfer cycles. 'X' = don't care. 'HIF Page' corresponds to the page in memory assigned to the Host Interface. *WriteL** is the VMEbus signal that indicates the direction of data transfer: it is high for transfers from the Host Interface and low for transfers to the Host Interface.

## 1.4 Transmit Control

### 1.4.1 Transferring the Destination Address

When the host executes the "Write Packet Data" command, the VME protoboard activates the signal *WrPktCycleUL*. In response, the system board activates *EnbTxDatL* to enable *D[0:31]* through the system board data buffers to the *TxDat* bus, and sets the **XferPend** bit of the *Control and Status Register*. Two cycles later, the system board latches the VME data so that the VME cycle can be terminated. The system board's transmit logic then asserts *TxReadyL*, indicating to the ring network that the destination address of the packet is on the *TxDat* bus and has been valid for the two previous cycles. Also asserted in this cycle is *TxDTACKL*, which causes the assertion of the VME data transfer acknowledge signal, *DTACKL*, and thus indicates to the host the receipt of the data. Once the mousetrapped data strobe signals *LDS0L* and *LDS1L* are detected high, the transmit control logic de-asserts *TxDTACKL*(causing *DTACKL*to be de-asserted). *TxReadyL* remains active.

### 1.4.2 Acquiring the Receiver and A Channel

Once the host has transferred the destination address of a packet to the Host Interface, it must wait for the ring network to acquire the intended receiver and a channel. When the ring network indicates it is ready for the data transmission by asserting *TxGo*, the HIF system board de-asserts **XferPend** and sets the **RingRdy** bit. The host may become aware of the "ring ready" condition in one of two ways. If **IntEnb1** of the *CSR* is set, the host will be interrupted once *TxGo* is asserted. Otherwise, the host must poll the *CSR* to check the **RingRdy** bit. If the former is the case, the interrupt control logic will assert *SysIRQ2L* (causing *IRQ2L* to be asserted) coincident with setting **RingRdy**. The host will eventually acknowledge the interrupt and will request *status/ID* information from the Host Interface. In response, the HIF transfers a word that contains its device identification and de-asserts the *Interrupt Request Line*. During the interrupt acknowledge sequence, the host interface prevents further interrupts by clearing **IntEnb0**, **IntEnb1**, and **IntEnb2**. The host may then ascertain the cause of the interrupt by reading the *CSR*.

### 1.4.3 Transferring Data Words

Once the **RingRdy** bit has been set, the host may transfer the packet data. When the host executes the "Write Packet Data" command, the VME board activates the signal *WrPktCycleUL*. Upon receipt of the mousetrapped signal *WrPktL*, the transmit control circuitry enables *D[0:31]* onto the *TxDat* bus by activating *EnbTxDatL*. During this cycle the system board logic also asserts *TxPut* to indicate to the ring network that valid data will be placed on the data bus during the next cycle. *TxPut* is de-asserted on the following cycle. Two cycles later, *TxDTACKL* goes low, causing the assertion of the VMEbus data transfer acknowledge signal and thus terminating the VME cycle. Once *LDS0L and LDS1L* go high, *TxDTACKL* (and thus *DTACKL*) and *EnbTxDatL* are de-asserted. The host must execute a "Write Packet Data" command for each word of the packet. The transmit control circuitry holds *TxReadyL* low and asserts *TxPut* prior to each cycle in which a data word is transferred.

### 1.4.4 Indicating the End of the Packet

Once all words of the packet have been transferred to the Host Interface (and thus to the ring network as well), the host executes the "End of Packet" command. The VME board will activate the signal *EOPCycleUL*. Upon receipt of the mousetrapped signal *EOP*, the transmit control circuitry de-asserts *TxReadyL* since all words of the packet have been transferred, resets the **RingRdy** bit, and asserts *TxDTACKL*. Assertion of *TxDTACKL* causes the assertion of the VMEbus data transfer acknowledge signal. When the system board logic detects *LDS0L* and *LDS1L* high, indicating that the host has terminated the VMEbus cycle, *TxDTACKL* is de-asserted (and thus *DTACKL* is also de-asserted).

## 1.5 Receive Control

### 1.5.1 Host Interface Receive Buffers

When the host interface receives data from the ring network, it stores the information in one of its receive buffers. Each of the two Host Interface receive buffers is 32-bits wide, 16K words deep and may contain the data for a single packet. Data from the ring network may be written into one buffer while data is read from the other buffer by the host. A maximum of two packets may be stored in the receive buffers before the host reads the contents of one buffer. The switching of read and write operations between the buffers is transparent to the host.

Associated with each buffer is a 16Kx1 "latch" which is used to keep track of the end of a packet; the ring signal **RxTail** is written into this "latch". Also associated with each receive buffer are address counters which are incremented each time a word is written into or read from the buffer. When an entire packet has been written into a buffer, the packet word counter contains the number of words stored in that buffer, including the destination address of the packet. If the ring network attempts to write more than 16K words into either buffer, the **BufOvrFlA(B)** signal is activated, and **BufOvrFl** is also activated. If the corresponding interrupt bit of the CSR, **IntEnb2**, is set, the interrupt request line to the host is asserted due to the buffer overflow. The buffer overflow bits remain asserted until the ring network is rebooted.

There are three status bits internal to the receive control state machine. Two of these bits (*BufStatA* and *BufStatB*) indicate whether the buffers contain entire, unread packets, and one (*LRW*) indicates which of the buffers contains the least-recently written packet. If both buffers are empty when a packet is received from the ring network, the packet is placed in Buffer A; *BufStatA* is set and the *LRW* bit is cleared once the entire packet has been written into the buffer. *BufStatB* is set when an entire packet has been written into Buffer B. The following chart indicates the possible values of the status bits and the corresponding status of the two receive buffers:

| BufStatA | BufStatB | LWR | Status |
|---|---|---|---|
| 0 | 0 | 0 | Neither buffer contains an entire packet |
| 0 | 0 | 1 | Not valid |
| 0 | 1 | 0 | Buffer B contains an entire packet |
| 0 | 1 | 1 | Not valid |
| 1 | 0 | 0 | Buffer A contains an entire packet |
| 1 | 0 | 1 | Not valid |
| 1 | 1 | 0 | Both buffers contain an entire packet; Buffer A packet was received first |
| 1 | 1 | 1 | Both buffers contain an entire packet; Buffer B packet was received first |

**Table 2**

### 1.5.2 Writing Data into the Receive Buffers

When one (or both) of the the receive buffers is empty, the HIF receive control circuitry asserts the ring network signal *RxReadyL* to indicate that it can receive a packet of length 16K words. The system board logic also asserts the signal *CSAWrL* or *CSBWrL*, depending upon which buffer is empty. If both are empty, the data will be placed in Buffer A, as described above. *CSAWrL* causes the chip-select line to the Buffer to be asserted. *EnbRxDatA(B)L* is asserted during the same cycle to connect the ring network receive data (RxDat) with the I/O pins of the Buffer. Write-enable pulses are generated on clock cycles following the assertion of *RxPut* by the ring network. *RxReadyL* is de-asserted on the cycle following the assertion of *RxPut* and remains low while the packet is written into the Buffer.

The assertion of *RxTail* by the ring network indicates that the ring network has already transmitted the last word of that packet. Two cycles after the assertion of *RxTail*, *EnbRxDatA(B)L* is de-asserted. The high logic level on *EnbRxDatA(B)L* essentially disconnects the RxDat bus from the I/O pins of the Buffer. On the following clock cycle *CSAWrL* (or *CSBWrL*) is de-asserted, *BufStatA(B)* is asserted, and the **PktPend** bit of the *CSR* is set (if the bit is not already set ).

### 1.5.3 Alerting the Host to the Receipt of a Packet

Once the entire packet has been latched into a receive buffer, the host may be interrupted. If the **IntEnb0** bit of the *Control and Status Register* is set, the interrupt control logic asserts *Interrupt Request Line 2* on the VMEbus. The host's interrupt handler will eventually acknowledge the interrupt request by setting the lower three address bits to binary '010' (corresponding to interrupt request line 2) and asserting *IACKINL*. When the interrupt control circuitry detects low levels on *SysIACKINL* and the mousetrapped signal *LDS0L*, it de-asserts the interrupt request. *EnbIntStatL* is asserted during the same cycle to enable the *status/id* word onto the VME data bus. This word identifies the interrupting device to the host and is defined by the switches of Switch Bank 1. The host interface clears **IntEnb0**, **IntEnb1**, and **IntEnb2** to prevent further interrupts. On the following cycle, the interrupt control circuitry responds with *IntDTACKL*, causing *DTACKL*, the VMEbus data transfer acknowledge signal, to be asserted. Eventually the interrupt handler of the host will terminate the interrupt acknowledge sequence by driving *DS0L* high. The Host Interface system board logic de-asserts *EnbIntStatL*when it detects *LDS0L* high. On the following cycle, *IntDTACKL* is de-asserted, causing *DTACKL* to be de-asserted. The host may verify the cause of the interrupt by reading the Host Interface's *Control and Status Register*. If no other interrupt conditions exist, such as a buffer overflow, the host may begin to read the packet from the Buffer.

### 1.5.4 Reading the Word Count from the Receive Buffer

Since there is no indication of packet length inherent in the structure of the packet, the host must read the value stored in the packet word counter. The host executes a "Read Packet Data" cycle and receive control circuitry asserts *EnbBufCntAL* (or *EnbBufCntBL*). Thus, the packet length, or word count, is enabled onto the VMEbus. During the next clock cycle, *RxDTACKL* is asserted, resulting in the assertion of *DTACKL*. Of course, *DTACKL* signals the host that the data requested is currently available on the VME data bus. Eventually the host will terminate the transfer and *LDS0L* and *LDS1L* will go high. The receive logic de-asserts *EnbBufCntAL* in response and on the next cycle de-asserts *RxDTACKL*. De-assertion of *RxDTACKL* causes de-assertion of the VMEbus data transfer acknowledge signal *DTACKL*. *EnbRxAddrAL* (or *EnbRxAddrBL*) is asserted during this cycle as well.

### 1.5.5 Reading the Packet from the Receive Buffer

Once the word count of the packet has been transferred to the host, the actual contents of the packet may be read. The host executes another "Read Packet Data" command, causing the assertion of the *RdPktL* signal. In response to *RdPktL*, the receive control circuit asserts *CSARdL* (or *CSBRdL*) and thus asserts *CSAL*, the Receive Buffer chip select line. The Buffer output enable, *OEAL*, is also asserted during this cycle. Since *EnbRxAddrAL* is still asserted, the data from the Buffer is enabled onto the VME data bus shortly after the high-to-low transition of *CSAL*. Two cycles later, *RxDTACKL*, and ultimately *DTACKL*, is asserted, indicating to the host that the data requested is currently available on the bus. Once the host terminates the data transfer by driving *DS0L* and *DS1L* high, *CSARdL* and *OEAL* transition to a high logic level. On the following cycle *RxDTACKL* is de-asserted, causing the VMEbus data transfer acknowledge signal, *DTACKL*, to be de-asserted. This sequence of events is repeated for each word of the packet. Each time a word is read from the Buffer, the address counter increments the buffer's addresss.

### 1.5.6 Reading the Last Word of the Packet

Each time a word is read from a Buffer, the contents of the associated "tail latch," *TailBitA(B)*, are also read. When the *TailBitA(B)* is detected to be '1', the Buffer's address counters are cleared, *BufStatA(B)* is cleared and **PktPend** is cleared if *BufStatB(A)* is not set.

The host must read the *Control Status Register* after the entire packet has been read to determine whether a packet is stored in the other receive buffer. Interrupts should be enabled if no interrupt conditions exist.

If the host driver attempts to read more words than are contained in a particular packet and a packet is latched into the other Buffer, the HIF will transfer the packet word count and the contents of the other packet erroneously. If the other buffer is empty or only part of a packet has been received, the VMEbus cycle is acknowledged, but no data is transferred to the host.

## 1.6 Host Interface Control and Status Register (CSR)

### 1.6.1 CSR General Description

The *Control and Status Register* is a 32-bit register which contains information about the status of the Host Interface board, the ring, transfers to the ring, and the setting of certain control parameters. All bits except **ResetGenL** are reset during the ring-boot sequence and on power-up. Bit 0 is the least significant bit.

Bit 0:

**IntEnb0** When set to '1', the interrupt circuitry of the Host Interface activates the interrupt request line when an entire data packet has been received. When set to '0', this condition does not cause an interrupt request.

Bit 1:

**IntEnb1** When set to '1', the interrupt circuitry of the Host Interface activates the interrupt request line once the ring network has acquired a channel for data transmission. When reset to '0', this condition does not cause an interrupt request.

Bit 2:

**IntEnb2** When set to '1', the interrupt circuitry of the Host Interface activates the interrupt request line if **BufOvrFl** becomes activated. When reset to '0', the buffer overflow condition does not cause an interrupt.

Bit 3:

**ResetGenL** When this bit is low, the HIF activates ring interface signal *ResetGenL* When set to '1', *Reset GenL* is high.

Bit 4:

**PktPend** The Host Interface sets this bit to '1' when a data packet has been received from the ring network and written into one of the receive buffers. This bit is reset when the contents of both receive buffers have been read by the host.

Bit 5:

**XferPend** When this bit is set, the HIF has requested a transfer to the ring network and is awaiting an acknowledgement from the ring network.

Bit 6:

**RingRdy** When this bit is set, the HIF has acquired a ring network channel for data transmission.

Bit 7:

**PktCntNxt** When set, this bit indicates that if a "Write Packet Data" cycle is executed and at least one of the buffers contains a packet (**PktPend** = 1), the data transferred will be the packet word count.

Bit 8:

**BufOvrFl** This bit is set if the ring network attempts to write more than 16K words into either of the receiver buffers. The bit remains set until the ring network is rebooted.

Bits 9-31:
**Reserved**

### 1.6.2 Reading the Control and Status Register

The host may read the contents of the *CSR* by executing the "Read CSR" command. The protoboard logic will assert *RdCSRCycleUL* in response, and register control logic asserts *EnbCSRDatL* as a result. The *Control And Status Register* data is enabled onto the VME data bus by *EnbCSRDatL* and occupies the lower 9 bits. On the next clock cycle, the register control logic signals the host that the data on the VMEbus is valid by asserting *CSRDTACKL* (the VME data transfer acknowledge signal *DTACKL* is asserted due to *CSRDTACKL*). The host terminates the transfer by driving *DS0L* and *DS1L* high, causing the mousetrapped versions of these signals to go high and ultimately causing the register control logic to de-assert *EnbCSRDatL*. On the next cycle, *CSRDTACKL* is de-asserted, causing *DTACKL* to be de-asserted to terminate the cycle.

### 1.6.3 Writing the Control and Status Register

The host may write bits 0-3 of the *Control and Status Register* by executing a "Write CSR" command. The host must place the values to be written on the three least significant bits of the data bus, with the least significant bit corresponding to the value to be written for **IntEnb0**. The protoboard address decode logic asserts *WrCSRCycleUL* in response to the command. The double-latched version of this signal initiates the write sequence in the control logic. *RegCLK* is asserted on the next clock cycle to latch the lower three bits of the *D[0:31]* data bus. It is de-asserted on the following cycle simultaneous with the assertion of *CSRDTACKL*. Assertion of *CSRDTACKL* causes the VMEbus data acknowledge (*DTACKL*) to be asserted and thus indicates to the host that the bus data has been written. The host eventually terminates the cycle due to *DTACKL* and drives *DS0L* and *DS1L* high. When *LDS0L* and *LDS1L* are both high, *CSRDTACKL* is de-asserted, causing *DTACKL* to be de-asserted.

## 1.7 Test Register

A special register, the *Test Register*, physically resides on the prototype board. The *Test Register* is a 32-bit read-write register that can be used in the initial stages of debug. This register is accessed by executing the "Write Test Register" and "Read Test Register" commands with Jumper 1 set to position 3-1. Jumper 1 also resides on the prototype board and should be set to position 3-5 for normal operation. This register contains no meaningful information and will contain random values upon power-up.

## 1.8 Boot Sequence

The Host Interface board and the host driver are responsible for booting Pixel-planes5 ring network. When the driver is ready for the host interface to boot the multi-ring, it executes the "Write CSR" command with bit 3 of the data bus set to '0'. This command causes the *ResetGenL* signal to the ring network to be asserted; the ring network in turn asserts the network reset signal, *DevReset*. (Details of how the *CSR* is written are contained in Section **1.6.3**.) After at least 12.5 μsecs, the host executes another "Write CSR" command in order to set *ResetGenL* to '1'. Bit 3 of the data bus should be set to '1' during this command. All system board circuitry (except that associated with the booting of the ring network) is reset by *DevReset* from the ring during the boot sequence.

## 1.9 Special Conditions

The Host Interface hardware is designed to minimize the chances of the host crashing due to faulty software and to allow the host to operate with an installed protoboard when the rest of the Pixel-planes5 system is not operational .

If host software attempts to access one of the locations within the Host Interface memory map that is *reserved*, logic on the protoboard will activate *DTACKL*, the VMEbus data transfer acknowledge signal, to terminate the VME cycle. Activating *DTACKL* prevents a "time-out" on the VMEbus. No valid data is transferred during such cycles.

If a protoboard is installed in the host but the remainder of the Pixel-planes5 system is not operational, the protoboard will generate the data transfer acknowledge signal, *DTACKL*, in response to any VME access within the Host Interface page. By acknowledging the cycle, the Host Interface prevents a "time-out" on the VMEbus. Of course, no valid data is transferred during the access. The protoboard also generates *IACKOUTL*, the daisy-chain interrupt acknowledge signal, when the rest of the system is not "alive". In this case, the protoboard merely passes the falling edge of its *IACKINL* line to *IACKOUT L* to complete the daisy-chain.

# Chapter III.2 GRAPHICS PROCESSOR

## III.2.1 Graphics Processor Specification

The Pixel-Planes 5 primary design goal of one million Phong-shaded triangles per second requires approximately 250 MFLOPS in front-end performance. We meet this requirement with a MIMD array of up to 16 (or more) floating-point engines called *Graphics Processors* (GPs).

A Graphics Processor consists of an Intel i860 CPU, 8 MB of DRAM memory, and a ring interface. The ring interface occupies one ring node, and allows devices on other ring nodes to send programs and data to the GP, and to receive data from it. The GP consists of six main modules: the i860 CPU, its DRAM memory, a ring transmit interface, a ring receive interface, a status register, and an EPROM for booting the i860. Figure 1 shows the block diagram of a Graphics Processor.

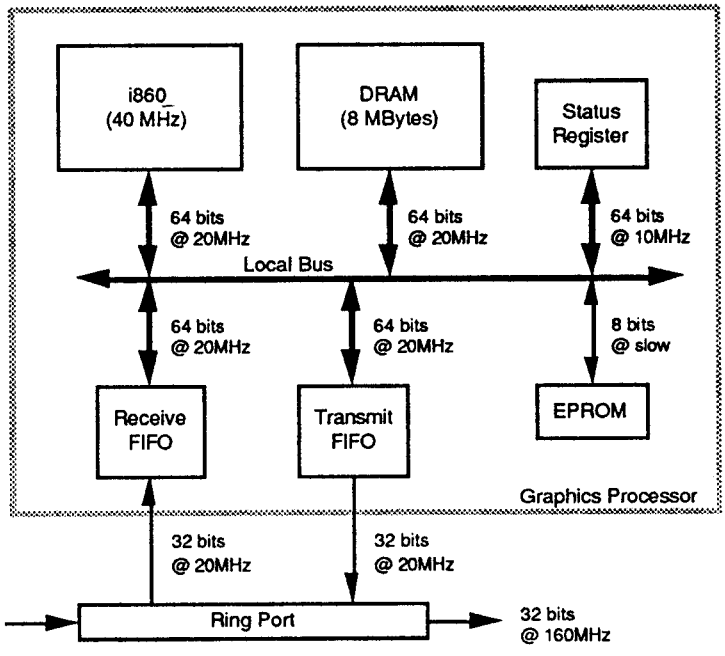



**Figure 1. Graphics Processor block diagram.**

### III.2.1.1 Intel i860 Microprocessor

This subsection gives a brief overview of the Intel i860 Microprocessor, the heart of the GP design. For more information, see the *i860 64-Bit Microprocessor Data Sheet*, the *i860 64-Bit Microprocessor Programmer's Reference Manual*, or the *i860 64-Bit Microprocessor Hardware Reference Manual.*

The i860 is a 64-bit RISC microprocessor with integrated code and data caches. Internally, it

contains a RISC integer processor and a 64-bit floating-point processor. The processor runs at 40 MHz and can perform an integer operation and a floating-point operation (potentially a multiply/accumulate) every 40 MHz clock cycle. This gives it a peak processing speed of 40 integer MIPS and 80 MFLOPS. Figure 2 shows a block diagram of the i860 Microprocessor.

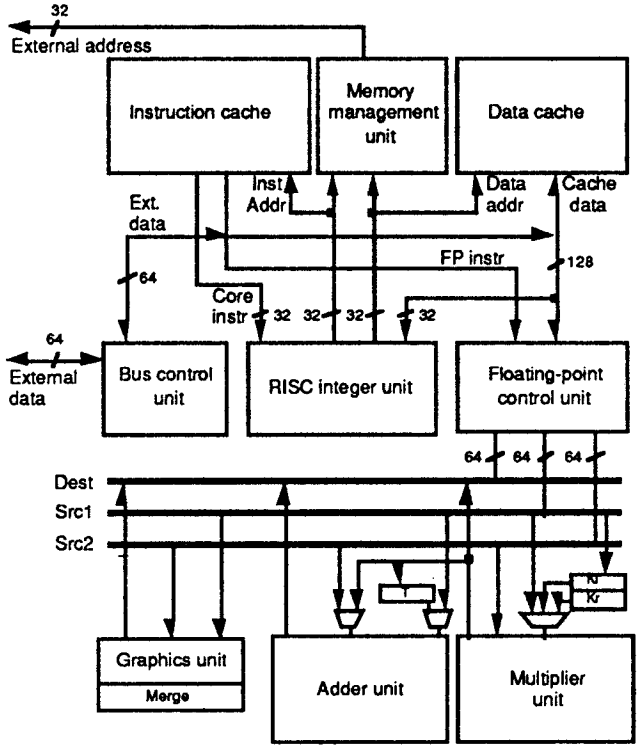



**Figure 2. Block diagram of the i860 Microprocessor.**

The integer processor controls overall operation of the i860. It executes load, store, integer, and control-transfer instructions and fetches instructions for the floating-point processor. It contains a set of 32 32-bit registers. It has a trap mechanism that allows it to rapidly respond to software-initiated and externally-initiated traps and interrupts. Virtual memory is supported with a 64-entry, four-way set-associative memory called the *Translation Lookaside Buffer* (TLB). When paging is enabled, the i860 uses the TLB to translate logical addresses to physical addresses and to check for access violations. Both user and supervisor privilege levels are provided.

The floating-point processor contains a 64-bit floating-point adder, a 64-bit floating-point multiplier, and a 64-bit parallel graphics unit. It has an additional set of dedicated registers, which can be accessed as 32 32-bit registers or 16 x 64-bit registers. In addition, special load and store instructions allow four adjacent 32-bit registers to be loaded from or stored to cache simultaneously. All floating-point and graphics instructions use these registers as their source and destination operands. Graphics instructions make use of a special, parallel integer unit that allows multiple additions and comparisons to be performed in parallel. Floating-point and graphics instructions have a latency of 3–4 cycles, but generate a new result every clock cycle.

The instruction cache is a 4 KB two-way set-associative memory with 32-byte blocks. It transfers up to 64-bits per clock cycle to the instruction unit. The data cache is an 8 KB two-

way set-associative memory, also with 32-byte blocks. It transfers up to 64-bits per clock cycle to the integer unit and up to 128-bits per clock cycle to the floating-point unit. The data cache uses a write-back scheme. Caching can be inhibited by software where necessary.

The processor communicates with external memory and I/O devices via a 64-bit bus called the *local bus*. The local bus can initiate a new cycle every 50 nsec (two clock cycles). It is pipelined, allowing up to three bus cycles to be outstanding at once.

**Address Map.** The i860 local bus has a 32-bit address space. All memory and I/O devices must be mapped to this address space, since no separate provisions are made for I/O. Figure 3 describes the GP's address map. Notice that DRAM memory is mapped redundantly. The reasons for this are described in the following sections. This means, however, that different logical addresses may correspond to the same physical address.

| A31[1] | A30-A27 | A26 | A25 | A24 | Device[2] | Address (Addr[27:0])[3] |
|---|---|---|---|---|---|---|
| X | 1 | 1 | 1 | 1 | EPROM/DRAM[4] | F000000–FFFFFFF |
| X | 1 | 1 | 1 | 0 | DRAM (8 Mbytes) | E000000–EFFFFFF |
| X | 1 | 1 | 0 | 1 | DRAM w/ transmit FIFO | D000000–DFFFFFF |
| X | 1 | 1 | 0 | 0 | Receive FIFO | C000000–CFFFFFF |
| X | 1 | 0 | 1 | 1 | Status/Command Registers | B000000–BFFFFFF |
| X | 1 | 0 | 1 | 0 | Unassigned | A000000–AFFFFFF |
| X | 1 | 0 | 0 | 1 | Unassigned | 9000000–9FFFFFF |
| X | 1 | 0 | 0 | 0 | Unassigned | 8000000–8FFFFFF |
| X | not all 1 | X | X | X | Invalid | 0000000–7FFFFFF |

[1]A31 controls cache enabling and is not used for address decoding.

[2]All devices are assigned 16 Mbyte blocks. They are multiply mapped within these blocks

[3]Address bits 30-27 all must be '1' for any valid address.

[4]After booting the system from the EPROM, the DRAMs are remapped to this location.

**Figure 3. GP Address Map.**

Address bit 31 is not used to decode addresses at all. This bit is used to control whether a memory cycle is to be cached or not. If A31 is 1, any date read during the bus cycle will not be cached. If A31 is 0, the data word is assumed to be cacheable. By setting A31 appropriately, software can control which data are cached, and which are always written to external memory (this is particulary important for I/O).

A reference to an unassigned or invalid address results in a bus error interrupt (*BErrInt*).

### III.2.1.2 DRAM Memory System

A Graphics Processor 8 MBytes of DRAM memory. The memory is 64 bits wide (to match the local bus) and contains four banks of 16 Motorola MCM514256 256K x 4 DRAM plastic ZIPs. The local bus accesses the memory system a maximum of one cycle every 50 nsec using fast page mode. DRAM pages are 512 words long, so as long as memory accesses do not cross page boundaries, the memory system can keep up with the i860's local bus.

DRAM refresh cycles are performed automatically in hardware every 12.8 μsec. They cost nothing if memory is not being accessed, and at worst cost 7 cycles every 249 cycles (2.8%).

### III.2.1.4 Ring Transmit Interface

A GP's transmit port consists of a 1024-entry, 64-bit FIFO built using four 18-bit IDT72225-20 FIFO chips. The FIFO is arranged so that 64-bit data words can be loaded from the i860's local bus and the high and low 32-bit words can be read independently from the FIFOs to interface with the 32-bit ring transmit port.

To transmit a message over the ring, the i860 loads the FIFO with data from the DRAM memory system. The i860 reads from the DRAMs while the FIFOs listen in, latching data as it becomes available. In this fashion, the i860 takes the place of a conventional DMA engine. Transmit FIFO control circuitry automatically acquires a ring channel and loads data from the FIFO to the ring whenever a message is present. A flag in the status register allows the i860 to determine when the transmit FIFO is sufficiently empty that a new transmission can be begun.

**Loading the transmit FIFO.** Rather than use a DMA to transfer data from DRAM to FIFO, the i860 generates DRAM addresses. It does this by reading data from an alternate set of DRAM addresses. When the i860 reads from these addresses, the transmit FIFO "listens in", latching data as it becomes available on the local bus. The transmit FIFO can accept data at the DRAMs' maximum speed (one 64-bit word every 50 nsec peak). Since the ring transmit port can transfer only one 32-bit word every 50 nsec, the FIFOs can be loaded twice as fast as messages are transmitted to the ring.

By performing 32-bit reads, the i860 can transmit either the upper or lower half of a 64-bit word. The FIFO control circuitry samples the i860's byte enable signals to determine which word (or words) is desired. These *valid* bits are latched into the FIFO along with the data. No extra FIFO parts are needed, since only 16 bits of the IDT72225's 18 data bits are needed to store the message. This allows the FIFOs to be loaded with data at any 32-bit word boundary. If desired, data for a single message can be loaded from two or more separate regions of DRAM memory. When the message is transmitted on the ring, *TxPut* is asserted only for words whose valid bits are set.

To send a message, the destination address must first be stored in DRAM. It must reside in the lower 32-bits of a 64-bit word; the upper 32 bits of this word will be ignored. The destination address is loaded into the transmit FIFO by reading from the "DRAM w/ Transmit FIFO" region of address space. The remainder of the message is loaded in the same manner. Since the destination address and message data are loaded directly from DRAM memory, software must ensure that the messages are stored in uncacheable memory (memory that has always been accessed with Addr[31] = 1) or that the data cache is flushed prior to transmitting the message. To terminate a transmission, the i860 writes to the *TxSndTail* command register, which causes a dummy word with a special "tail" bit to be set. If there is sufficient room in the FIFO, a second message can be read into the FIFO immediately. Figure 4 shows how a typical message is stored in the transmit FIFO before transmission over the ring begins.

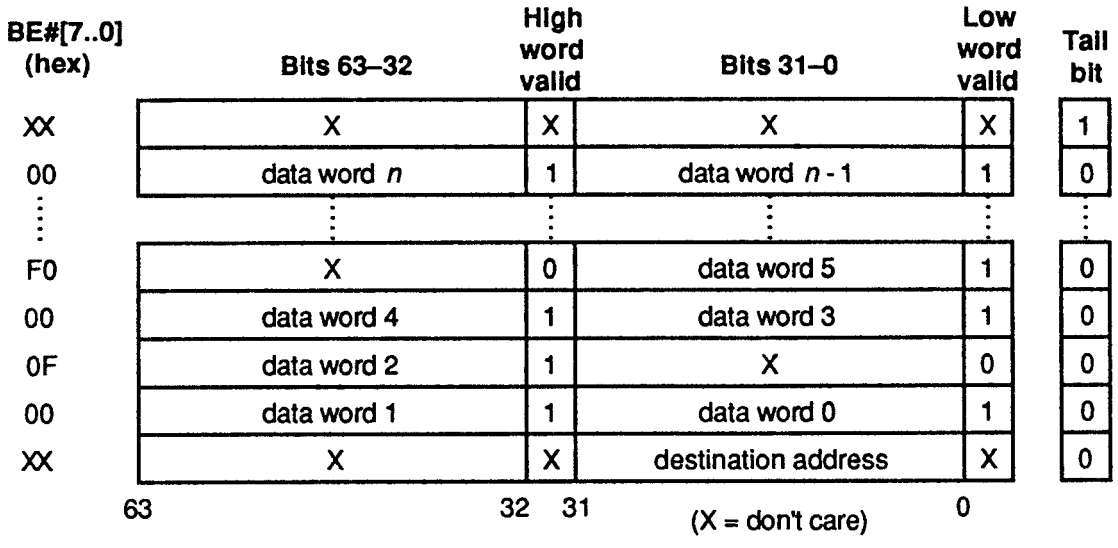



**Fig. 4. Contents of transmit FIFO prior to transmitting a message over the ring. BE#[7..0] are the byte enables derived from the read instruction and read address.**

To transmit messages longer than the FIFO length (2K 32-bit words), the i860 must know how full the FIFO is. Additionally, if the ring port's full bandwidth is to be used, the i860 must know when the FIFO is nearly empty, so it can write more data into the FIFO before it empties completely. The *TxAEInt* bit in the status register serves this purpose. This interrupt bit, which can be separately enabled and disabled, is set every time the transmit FIFO's "almost empty" flag is asserted (this occurs when the FIFO contains fewer than 128 64-bit words). This bit is cleared by writing to the *ClrTxAEInt* command register. If it remains set after clearing it, up to 895 64-bit words may be written to the transmit FIFO (note that a 32-bit word occupies the same space in the FIFO as a 64-bit word).

**Reading data onto the ring.** The transmit port controller automatically requests a ring channel and loads data onto the ring as soon as a destination address and a second word (either message data or tail word) are loaded into the transmit FIFO (this is to save ring bandwidth if there is a pause between loading the destination address and the remainder of the message).

The destination address is recognized by the transmit port controller by being either 1) the first 64-bit word after the last tail word, 2) the first 64-bit word after system initialization, or 3) the first 64-bit word after the transmit port is reset after *TxClrFIFO* command register write. The low 32 bits of the 64-bit destination address word are interpreted as the destination address regardless of how the valid bits are set. The high 32 bits of this word are ignored and are not transmitted When the FIFO is read, the valid bits are used to control *TxPut*, indicating when a valid word is to be sent over the ring.

A transmission may be aborted in mid-stream by writing to the *TxClrFIFO* command register. This truncates any transmission currently in progress and flushes all messages that may be stored in the transmit FIFO. This capability should generally be used only to report fatal errors (such as ring device timeouts, etc.) to the host interface.

Figure 5 shows how this message is actually transmitted over the ring to another device.

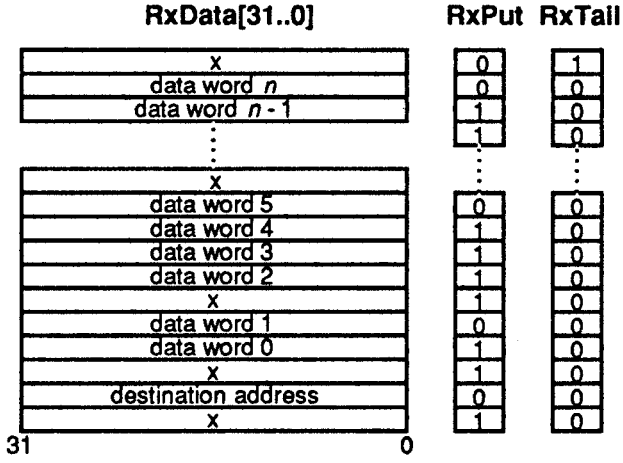



**Figure 5. Clock-by-clock log showing how above message arrives at another device's receive port.**

### III.2.1.4 Ring Receive Interface

A GP's ring receive interface consists of a 1024-entry, 64-bit FIFO arranged in two 32-bit banks. Packets arriving from the ring are buffered in this FIFO so that the i860 need not respond to incoming messages immediately. Each message that arrives on the receive port begins on an even word boundary in the 64-bit FIFO. The destination address is written into the low word. A garbage word is written into the high word. Successive data words are then written into alternate FIFO banks.

The receive FIFO is mapped into the i860's address space and can be read onto the local bus at the i860's full bus bandwidth (one 64-bit word every 50 nsec). The receive FIFO controller does not count the number of data words that have been received. This simplifies the receive controller circuitry and allows reads from the FIFO to occur at the i860's full bus speed, but has two implications for the message protocol:

1) No message should be sent to a GP that is longer than 1750 32-bit words. The GP's receive port will only accept messages when the receive FIFO is 1/8 full or less, thereby guaranteeing that 7/8 * 2048 = 1792 words of unused FIFO space remain whenever the receive port indicates that it is ready to receive a message. The extra 42 words are a safety margin to allow propagation time for the flow control signals.

2) Every message sent to a GP must encode the number of words in the message in the destination address or some other standard place. The receive port controller hardware will not block the i860 from reading an empty FIFO (this does cause an *RxEmptyInt* interrupt, however) or from reading data from a succeeding message. The software message handler running in the i860 must assure that the proper of number of reads is performed for each message.

After the i860 has finished processing a message (ie. read all data words from the FIFO), it writes to the *ClrRxMsg* command register. This informs the receive FIFO controller that the

i860 is ready to receive a new message. If any other messages have been received in the meantime, the *RxMsgInt* will again be asserted. In this manner, one interrupt is received for each incoming message, and each message must be acknowledged by a *ClrRxMsgInt* command. The receive port will accept additional messages from the ring unless the FIFO contains three unread messages or the FIFO is more than 1/8 full.

Figure 6 is a snapshot of the FIFO containing two messages, the first having an odd number of data words, the second having an even number of data words.

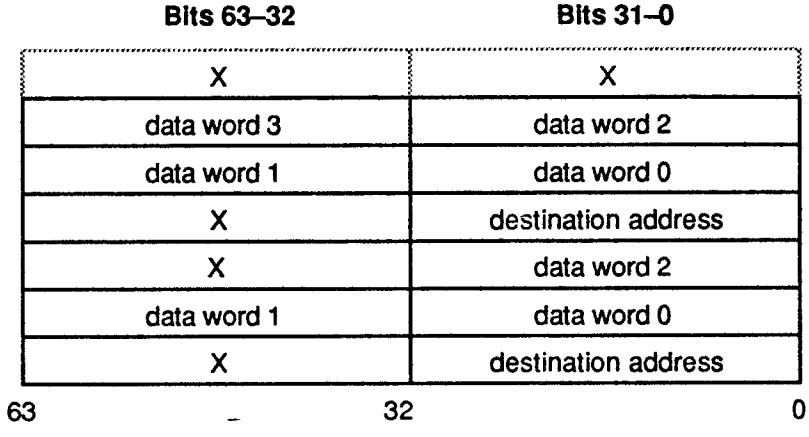



**Figure 6. Snapshot of receive FIFO containing two messages. The first has three data words (odd); the second has four data words (even).**

### III.2.1.5 Status and Command Registers

A 32-bit register containing various status information is available on the i860's local bus. This status register is located at absolute address 0xFB000004 (notice that this is the high 32-bits of a 64-bit word). It contains the 16-bit synchronization timer, an 8-bit board ID, and eight status bits (see Figure 7).

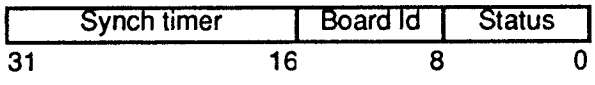



**Figure 7. Status register.**

**Synchronization timer.** Status register bits 16–31 are the *Synchronization Timer*, a 16-bit counter that increments every 50 nsec. This counter is reset to 0x0000 when a system reset occurs and begins counting the first Ring20Clk cycle after reset is deasserted. As a result, this timer is nearly synchronous from board to board. By keeping the high-order bits of the counter in memory (incremented whenever a timer interrupt occurs), a global system timestamp can be maintained.

**Board ID.** Bits 8–15 contain an eight-bit vector that specifies the board's permanent ID or serial number. Each Pxpl5 board has a unique board ID. This allows a GP to uniquely identify itself without relying on information obtained from the ring port.

**Interrupt Status.** Bits 0–7 indicate the status of pending interrupts. Bits 0–6 are sticky bits which are set (active low) when an interrupt conditions occurs and remain set until explicitly cleared by writing to the appropriate command register (described in the following section). All interrupts are always enabled except for the *TXAEInt* interrupt, which may be enabled or disabled by writing to the appropriate command register. Bit 7 indicates whether the transmit FIFO almost empty interrupt is enabled. The *TxAEInt* bit is set whenever the transmit FIFO is in an almost empty condition and remains set until explicitly cleared. The enable/disable bit only controls whether or not the *TxAEInt* bit causes a processor interrupt.

The *Interrupt Mode* (IM) bit in the *Processor Status Register* (PSR) should not be reset until the fifth instruction following a command register write that clears an interrupt bit. This allows time for the interrupt control logic to deassert the i860's interrupt pin, avoiding a spurious interrupt.

| Bit # | Abbreviation | Description |
|---|---|---|
| 0 | BErrInt | Bus error interrupt |
| 1 | TimInt | Timer interrupt (every 50 nsec * 2^16 = 3.2768 msec) |
| 2 | RxMsgInt | Received message interrupt |
| 3 | RxOvFlwInt | Receive FIFO overflow interrupt |
| 4 | RxUnFlwInt | Receive FIFO underflow interrupt |
| 5 | TxOvFlwInt | Transmit FIFO overflow interrupt |
| 6 | TxAEInt | Transmit FIFO almost empty interrupt |
| 7 | TxAEEn | Enable bit for transmit FIFO almost empty interrupt (low when enabled) |

**Command registers.** A number of special write-only addresses are reserved for command registers. These registers reset interrupt bits and implement various board-level commands such signaling the end of an outgoing message or turning signal LEDs on or off. The command register addresses and their functions are the following:

| Address | Abbreviation | Command register description |
|---|---|---|
| 0xFB000000 | LED0Off | LED 0 Off |
| 0xFB010000 | LED0On | LED 0 On |
| 0xFB020000 | LED1Off | LED 1 Off |
| 0xFB030000 | LED1On | LED 1 On |
| 0xFB040000 | LED2Off | LED 2 Off |
| 0xFB050000 | LED2On | LED 2 On |
| 0xFB060000 | LED3Off | LED 3 Off |
| 0xFB070000 | LED3On | LED 3 On |
| 0xFB080000 | LED4Off | LED 4 Off |
| 0xFB090000 | LED4On | LED 4 On |
| 0xFB0A0000 | LED5Off | LED 5 Off |
| 0xFB0B0000 | LED5On | LED 5 On |
| 0xFB0C0000 | LED6Off | LED 6 Off |
| 0xFB0D0000 | LED6On | LED 6 On |
| 0xFB0E0000 | LED7Off | LED 7 Off |
| 0xFB0F0000 | LED7On | LED 7 On |

| | | |
|---|---|---|
| 0xFB100000 | ClrBErrInt | Clear bus error interrupt |
| 0xFB110000 | ClrTimInt | Clear timer interrupt |
| 0xFB120000 | ClrRxMsgInt | Clear receive FIFO message received interrupt |
| 0xFB130000 | ClrRxOvFlwInt | Clear receive FIFO overflow interrupt |
| 0xFB140000 | ClrRxUnFlwInt | Clear receive FIFO underflow interrupt |
| 0xFB150000 | ClrTxOvFlwInt | Clear transmit FIFO full interrrupt |
| 0xFB160000 | ClrTxAEInt | Clear transmit FIFO almost empty interrupt |
| 0xFB170000 | EnTxAEInt | Enable transmit FIFO almost empty interrupt |
| 0xFB180000 | DisTxAEInt | Disable transmit FIFO almost empty interrupt |
| 0xFB190000 | TxSndTail | Clock tail bit into transmit FIFO |
| 0xFB1A0000 | TxClrFIFO | Reset the transmit FIFO (after time-out condition) |
| 0xFB1B0000 | DRAMMap | Map DRAM to EPROM address space |
| 0xFB1C0000 | Reserved | |
| 0xFB1D0000 | Reserved | |
| 0xFB1E0000 | Reserved | |
| 0xFB1F0000 | Reserved | |

Note that data on the data bus is ignored during command-register writes; only the address conveys meaning.

### III.2.1.6 Boot EPROM

A Graphics Processor contains a 64K x 8 bit EPROM for booting the i860 after system reset. This is necessary, since the DRAM memory will be in an undefined state after system reset, and the i860 must load software from the ring. The i860 has a special boot-up mode in which it reads instruction words 8 bits at a time, rather than 64-bits at a time when it is normally operating. Upon system reset, the Boot EPROM is mapped to the upper 16 MByte portion of address space. By writing to the *DRAMMap* command register, the i860 can map DRAM into the memory space previously occupied by the EPROM. This is necessary since the trap vectors are located in the upper region of memory.

## Part III Chapter 5

# FRAME BUFFER

### 5.1 Introduction

The function of the Frame Buffer Board is to store the pixel data generated by the renderer(s) and to generate the appropriate RGB signals for a 1280 x 1024 display. A 64x64 pixel, user-definable hardware cursor is implemented on the Frame Buffer. Other features include three display modes for each of the two Buffers: high-resolution, low-resolution, and high-resolution stereo.

### 5.2 Implementation

The Frame Buffer Board is comprised of several basic subsystems. These subsystems include the ring interface, the video subsystem, the memory subsystem and control circuitry.

#### 5.2.1 Ring Interface

The ring interface provides the communication link between the Frame Buffer Board and the rest of the devices in the system. The Frame Buffer Board has two *ports* to the ring, designated as Port 0 and Port 1. Both the *transmit* and *receive interfaces* of Port 1 are implemented. Port 0's *transmit interface* is unused. Packets containing data to be written into a Frame Buffer register must be sent to Port 0. The Frame Buffer uses Port 1 to transmit the new *Control Register* contents when the register has been updated.

#### 5.2.2 Video Circuitry

The video subsystem is comprised of three RAMDACs and two hardware cursor generator chips. The RAMDACs provide the red, green, and blue color lookup tables and perform the digital-to-analog conversion of the pixel data. Registers within the RAMDACs control parameters such as the IRE pedestal and enabling bit planes to access the color palette RAM.

The hardware cursor chips provide data for addressing up to three locations within the overlay RAM. Chip 0 controls overlay plane 0 and Chip 1 controls overlay plane 1. These chips allow the user to define a 64x64 pixel cursor and (optionally) a cross-hair cursor. The cursor pattern can be changed by merely updating the cursor chips' RAM. The cursor is moved by writing to various registers within the chips. (See **5.4.3**).

### 5.2.3 Memory Organization

The Frame Buffer memory is physically organized as sixteen banks of 512 x 512 x 24 pixels. Logically, the Frame Buffer consists of 80 128x128 pixel regions. There are two identical halves of the frame buffer, each occupying 256 pages of memory. Thus, double-buffered operation is possible; one image can be displayed from the upper 256 pages while a new image is written into the lower 256 pages. The upper 256 pages of memory are referred to as Buffer 0 and the lower 256 as Buffer 1 (See **Figure 1**). Bit 2 in the Frame Buffer *Control Register* indicates which half of the frame buffer is being displayed. Switching buffers takes place only during vertical retrace.

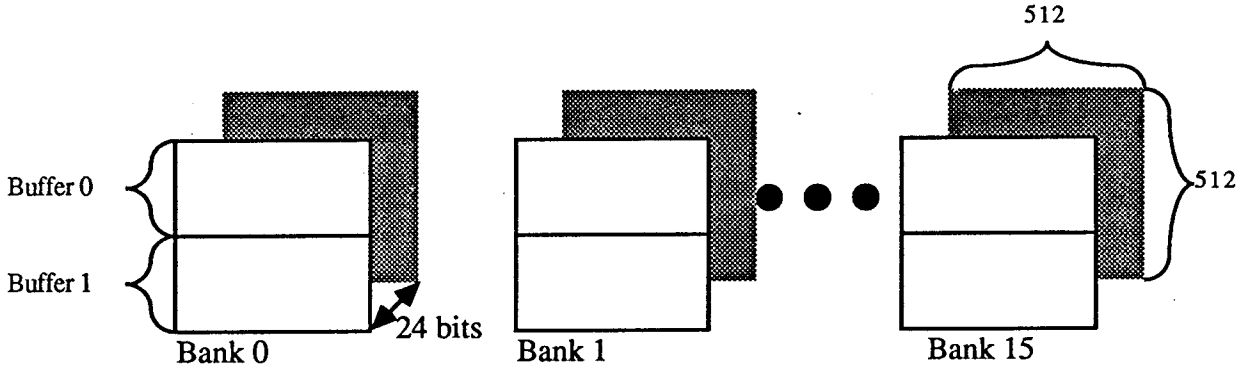



**Figure 1**

Pixel data from the renderers arrives at the Frame Buffer in row-major order such that consecutive pixels are written into the same page (row) of consecutive banks of memory. In this manner, all of the pixels of a scan line are located in one page (across the sixteen banks) of memory as shown in **Figure 2.** In addition, four adjacent scanlines are contained in one page of memory. Scanlines 0-3 are located on Page 0, for example, and scanlines 4-7 are located on Page 1.

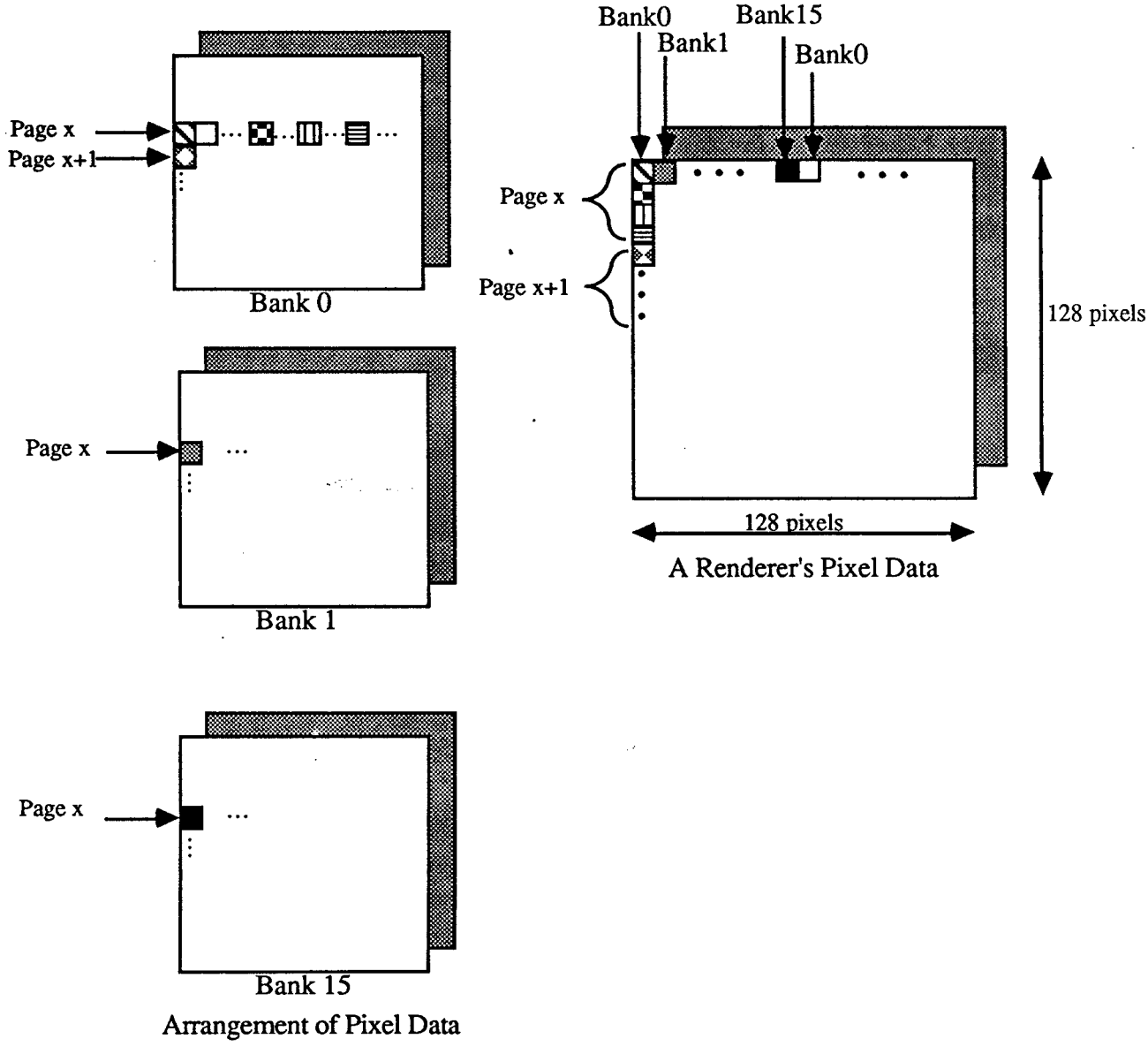

Page x
Page x+1
Bank 0
Page x
Bank 1
Page x
Bank 15
Arrangement of Pixel Data
Bank0
Bank1
Bank15
Bank0
Page x
Page x+1
128 pixels
128 pixels
A Renderer's Pixel Data

**Figure 2**

## 5.3 Display Modes

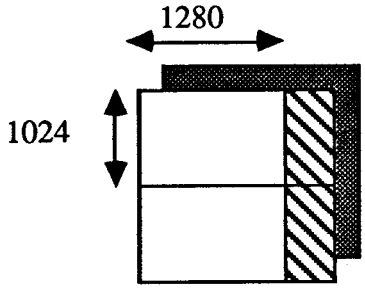
### 5.3.1 Display Modes Overview

There are six valid display modes for the Frame Buffer: high-resolution, upper and lower Buffer; low-resolution, upper and lower Buffer; and high-resolution stereo, upper and lower Buffer. The active display mode is indicated by bits 0, 1 and 2 of the *Control Register*. In high-resolution mode, 1280 x 1024 pixel values from either Buffer are displayed. In low-resolution, or zoom mode, a 640 x 512 "window" on the Frame Buffer memory is blown up to fill the entire display. The location of the two possible "windows" (one from each Buffer) is shown in **Figure 3.**

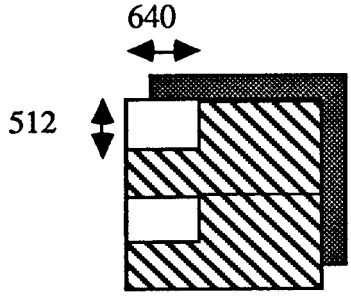
High-resolution stereo mode involves displaying on alternate frames two 1024x1024 regions. One of the two regions contains the right-eye image and the other contains the left-eye image. As with the other display modes, these regions can be located in either Buffer of the Frame Buffer Memory. Since the Frame Buffer holds 2048x1024 pixels per Buffer, only 1024x1024 pixels are available for each eye's image. Therefore, in stereo mode, the image is located on the left side of the screen, with the remaining 128x1024 pixels blanked out.

### 5.3.2 Changing Display Modes

An application board such as the Host Interface may send a packet to the Frame Buffer requesting that the display mode be changed. The packet data is latched, but is not written into the Frame Buffer *Control Register* until the next vertical retrace. At the beginning of the next vertical retrace, the mode data is transferred to the appropriate bits of the *Control Register*. At this time, the Frame Buffer also sends a packet containing the *Control Register* contents to the device which requested the mode change. At the end of vertical retrace, the display is refreshed according to the new mode.

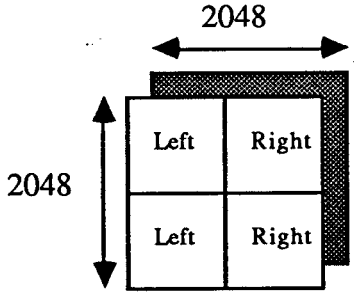



a) High-Resolution Mode




b) Low-Resolution Mode

Memory Contents
Not Displayed




c) High-Resolution
Stereo Mode

**Figure 3**

## 5.4 Frame Buffer Registers

Several registers reside on the Frame Buffer Board. These registers include: *Control Register*, RAMDAC Registers, and Hardware Cursor Registers.

### 5.4.1 Control Register

The Frame Buffer *Control Register* contains information concerning the display mode. All bits are write-only and occupy the 3 least-significant bits of a data packet. The *Control Register* is written by sending the Frame Buffer a packet whose *Command Bits* correspond to "Write Control Register" (See **5.5**). Note that this register is only updated during vertical retrace. The bit assignments are as follows:

Bits 0,1,2:

**Display Mode** These bits indicate which of the six display modes is currently active.

| Mode 2 | Mode1 | Mode0 | |
|---|---|---|---|
| 0 | 0 | 0 | High-Resolution, Buffer 0 Mode |
| 0 | 0 | 1 | Low-Resolution, Buffer 0 Mode |
| 0 | 1 | 0 | High-Resolution Stereo, Buffer 0 Mode |
| 1 | 0 | 0 | High-Resolution, Buffer 1 Mode |
| 1 | 0 | 1 | Low-Resolution, Buffer 1 Mode |
| 1 | 1 | 0 | High-Resolution Stereo, Buffer 1 Mode |

### 5.4.2 Registers Associated with the RAMDACs

Each of the three RAMDACs on the Frame Buffer Board contains several 8-bit registers which are implemented as write-only. Registers pertaining to this system include: address registers, command registers, pixel read mask registers, pixel blink mask registers, overlay read mask register, and overlay blink mask register. All registers must be initialized upon power-up. Command Register 0 should be set for 4:1 multiplexing and the alternate palette should be disabled. Command Register 1 should be set for 0-pixel pan. Command Register 2 should be set so that sync is enabled, the IRE pedestal is 7.5, loading of the palette RAM is normal, PLL select is sync and the test register is disabled. A

detailed description of the function of the RAMDAC registers is found in [1].

### 5.4.3 Registers Associated with the Hardware Cursor

There are several registers within each hardware cursor chip. In this system, these registers are write-only and include: address register 0, address register 1, command register, cursor (x) high register, cursor (x) low register, cursor (y) high register, cursor (y) low register, window (x) low register, window (x) high register, window (y) low register, window (y) high register, window width low register, window width high register, window height low register and window height high register. The command register should be set for 4:1 multiplex control. All registers should be initialized on power-up. A detailed description of the hardware cursor chips' registers may be found in [2].

## 5.5 Commands

The purpose of data received by the Frame Buffer is indicated by Bits 8-23 of the packet's destination address. In essence, bits 8-10 indicate what "command" the Frame Buffer is to perform . The following table indicates the bit settings and the corresponding commands:

| Bit 8 | Bit 9 | Bit 10 | Command |
|---|---|---|---|
| 0 | 0 | 0 | Write data into Frame Buffer Memory |
| 0 | 0 | 1 | Read data from Frame Buffer Memory |
| 0 | 1 | 0 | Write Control Register |
| 1 | 0 | 0 | Write RAMDACs |
| 1 | 0 | 1 | Write Hardware Cursors |

### 5.5.1 Frame Buffer Writes

For writes to Frame Buffer memory, bits 0-3 of the packet's destination address indicate the starting horizontal address of the 128x128 region to be written, and bits 4-7 indicate

the starting vertical address. So, for example if the lower 11 bits of the destination address were :

| COMMAND | VERTICAL | HORIZONTAL |
|---|---|---|
| Bit 10 9 8 | 7 6 5 4 | 3 2 1 0 |
| 0 0 0 | 0 0 0 1 | 0 1 1 1 |

the data would be written into the frame buffer at the memory location corresponding to the seventh renderer region from the left-most region, one region from the top (See Figure 4).

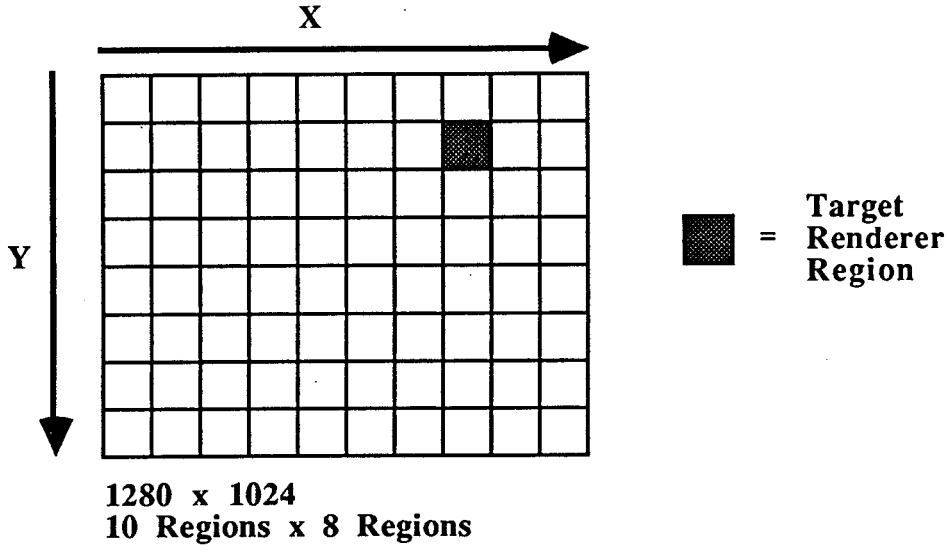



**Figure 4**

### 5.5.2 Frame Buffer Reads

Reads of Frame Buffer memory are allowed only for initial system debug. As with writes to the frame buffer, bits 0-3 of the packet's destination address indicate the starting horizontal address of the 128x128 region to be read, and bits 4-7 indicate the starting vertical address. Requests to read have the lowest priority of all Frame Buffer commands, and thus may take a long time to be serviced.

### 5.5.3 Write Control Register

When a device requests a write to the Frame Buffer *Control Register* , bits 0-2 of the data word are latched into a buffer. At the beginning of the next vertical retrace, these latched values are written into the *Control Register* and the appropriate changes in display mode are implemented in the next frame. A packet containing the new *Control Register* contents is sent to the requesting device at the beginning of vertical retrace.

### 5.5.4 Write RAMDACs

A packet which contains data for the RAMDACs is distinguished by bits 8-10 of its destination addess. The number of data words in the packet is variable, depending upon is what being written (a register or RAM). Each data word of the packet contains one byte for each of the three RAMDACs, as well as two control bits, C1, and C0, which correspond to the Command Control inputs of the RAMDACs (See **Figure 5**). As indicated in [1], the Command Control inputs, along with the internal address register, determine which RAMDAC register, color palette RAM or overlay RAM location will be accessed.

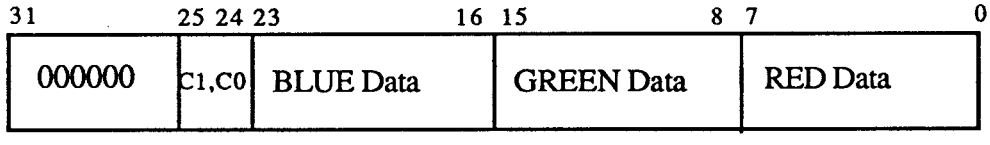



**Figure 5**

Thus, to access one or more of the control registers, a device might send a packet to the Frame Buffer whose data words are constructed as shown in **Figure 6.** The first two data words of the packet contain the address of the register, color palette RAM or overlay RAM location to be accessed. The next data word(s) contains the information to be written into the register or RAM. Note in [1] that the RAMDAC address registers do not increment after write cycles to the control registers. **Table 1** indicates the required address registers' settings and values of CO,C1 for accessing the registers and RAM.

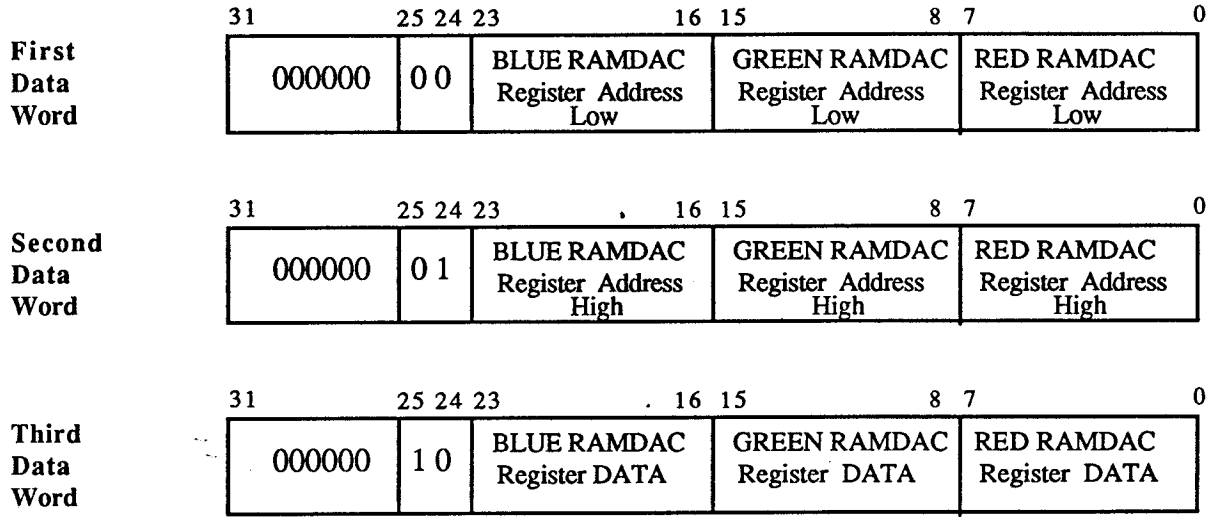



**Figure 6**

| Address Register (hex) | | C1 | C0 | Location Addresssed |
|---|---|---|---|---|
| High | Low | | | |
| x | xx | 0 | 0 | Address Register Low (ADDR0-7) |
| x | xx | 0 | 1 | Address Register Low (ADDR8-11) |
| 1 | 00 | 1 | 0 | Overlay RAM, Color 0 |
| . | .. | . | . | . |
| 1 | 1F | 1 | 0 | Overlay RAM, Color 31 |
| 2 | 01 | 1 | 0 | Command Register 0 |
| 2 | 02 | 1 | 0 | Command Register 1 |
| 2 | 03 | 1 | 0 | Command Register 2 |
| 2 | 04 | 1 | 0 | Pixel Read Mask Register Low |
| 2 | 05 | 1 | 0 | Pixel Read Mask Register High |
| 2 | 06 | 1 | 0 | Pixel Blink Mask Register Low |
| 2 | 07 | 1 | 0 | Pixel Blink Mask Register High |
| 2 | 08 | 1 | 0 | Overlay Read Mask Register |
| 2 | 09 | 1 | 0 | Overlay Blink Mask Register |

**Table 1**

### 5.5.5 Write Hardware Cursor

A device may write to a RAM location or to one of the various registers on the hardware cursor chip by sending a packet to Port 0 with bits 8-10 of the packet's destination address set to 101, respectively. The number of data words in the packet is variable, depending on the register(s) or RAM being accessed. Each data word of the packet contains one byte for each of the hardware cursor chips, as well as two control bits, C1 and C0, which correspond to the Control inputs of the chips (See **Figure 7**). As indicated in [2], the Command inputs, along with the internal address pointer register, determine which cursor register or RAM location will be accessed.

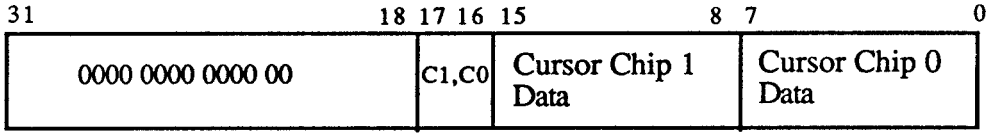



**Figure 7**

To access one or more of the control registers, a device might send a packet to the Frame Buffer whose data words are constructed as shown in **Figure 8**. The first two data words contain the address of the register (or RAM) to be written, along with the proper values of C1 and C0 for writing to the address registers. The next data word(s) of the packet contains the information to be written. **Table 2** shows the required address registers' settings and the values of CO,C1 for accessing the registers and RAM. A detailed description of the registers within the hardware cursor chips is contained in [2].

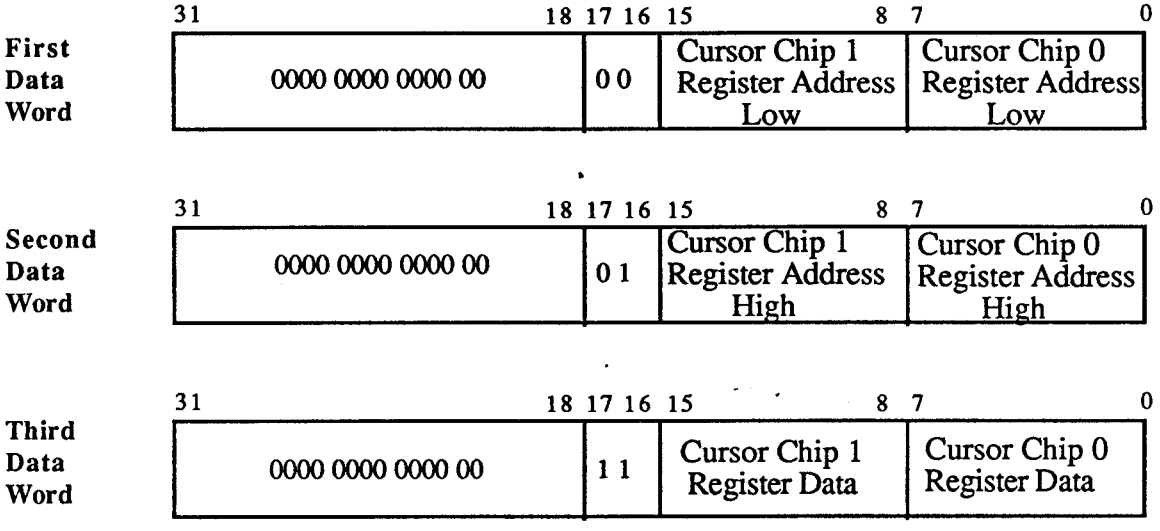

31
18 17 16 15
8 7
0
First Data Word
0000 0000 0000 00
0 0
Cursor Chip 1 Register Address Low
Cursor Chip 0 Register Address Low
Second Data Word
0000 0000 0000 00
0 1
Cursor Chip 1 Register Address High
Cursor Chip 0 Register Address High
Third Data Word
0000 0000 0000 00
1 1
Cursor Chip 1 Register Data
Cursor Chip 0 Register Data

**Figure 8**

| C1 | C0 | Address (hex) Register1 | Register0 | Register/RAM Addressed |
|---|---|---|---|---|
| 0 | 0 | xx | xx | Address Register 0 |
| 0 | 1 | xx | xx | Address Register 1 |
| 1 | 0 | 00 | 00 | Cursor RAM Location 000 |
| . | . | .. | .. | . |
| . | . | .. | .. | . |
| 1 | 0 | 01 | FF | Cursor RAM Location 1FF |
| 1 | 1 | xx | 00 | Command Register |
| 1 | 1 | xx | 01 | Cursor (x) Low Register |
| 1 | 1 | xx | 02 | Cursor (x) High Register |
| 1 | 1 | xx | 03 | Cursor (y) Low Register |
| 1 | 1 | xx | 04 | Cursor (y) High Register |
| 1 | 1 | xx | 05 | Window (x) Low Register |
| 1 | 1 | xx | 06 | Window (x) High Register |
| 1 | 1 | xx | 07 | Window (y) Low Register |
| 1 | 1 | xx | 08 | Window (y) High Register |
| 1 | 1 | xx | 09 | Window Width Low Register |
| 1 | 1 | xx | 0A | Window Width High Register |
| 1 | 1 | xx | 0B | Window Height Low Register |
| 1 | 1 | xx | 0C | Window Height High Register |

**Table 2**

# Part IV, Chapter 1
# General Technology Considerations for Pxpl5 Custom IC's

## 1.1 Technology Considerations for Fabrication of Pxpl5 EMC

### Packaging Constraints

The Pxpl5 EMC will be packaged in a 44-pin ceramic leaded chip carrier. Note that there is a plastic (PLCC) part with identical footprint, providing a path to lower cost packaging for this part. VLSI Technology's "Semiconductor Package Selection Guide" provides information for a ceramic leaded chip carriers, including VTI Bond Form # 23-67510: Pad Size: 0.340" x 0.340" (8.636 mm square); **Max Die Size: 0.300" x 0.300" (7.62 mm square)**. This package is available from Kyocera as part number VAO-44J.

### Project Size

The size of the EMC layout will be driven primarily by the 32,000 (65,000) 6-T SRAM cells in the memory array, whose dimension will be 128 (256) PE's (pairs of memory rows) by 256 bits (128 columns). Initial layouts of scalable 6-T memory cells (MOSIS Rev 6 Rules) suggest that the height (dimension along Word line) for two cells (*i.e.*, the vertical pitch for pixel processing elements) can be no less than 52 lambda. We have designed a "short-fat" cell that is 40 x 52 lambda (2080 lambda$^2$) for a pair of cells placed one above the other, with the vertical dimension limitied hard by m2/m2 and ndiff/ndiff spacings. Similarly, it appears that the width is constrained to be $\geq$ 35 lambda; a "tall-thin" cell, which is 35 x 64 lambda (2240 lambda$^2$) for a vertical pair. has horizontal dimensions limited hard by m1/m1 and ndiff/pdiff spacings. (Note that these two cells are constrained electrically as well, and have somewhat different transistor ratios; it is possible that it will be necessary to give up more area on either design to meet speed and noise margin goals.) J. Eyles estimates that the Tree and ALU strips will be 1200 lambda wide (about 2000 lambda in mcncmos); it will, of course, have the same vertical dimensions as the memory. This dimension, added to the total memory width, gives the overall width of the project internals.

Space is required around the outside of the project for pads and signal routing; we assume that ~500µ will be required on all four sides for this (not a generous budget!). This requirement implies that we have a space for the project internals about 6.5 mm square.

Various options for memory size and number of processing elements that can be mapped onto the available fabrication services:

| Organization (#PI's,#PE's,#Bits/PE) | Mem Cell | Size (lambda) | Feature Size | Project Size (mm) | Project Designation |
|---|---|---|---|---|---|
| 2 x 128 x 160 | T-T | 8000 x 8192 | Mosis 1.6µ | 6.4 x 6.55 | Pxpl5A |
| 2 x 64 x 256 | S-F | 6320 x 6656 | Mosis 2.0µ | 6.32 x 6.66 | Pxpl5B |
| 2 x 128 x 256 | T-T | 11360 x 8192 | Mosis 1.2µ | 6.82 x 4.92 | Pxpl5C |
| 2 x 128 x 256 | mcnc | 17440 x 11776 | MCNC 1.25µ | 7.0 x 4.7 | Pxpl5D |

(Note that lambda has a different interpretation for the MCNC process, where lambda = 0.4µ).

## Yield

From Cesar Pina's talk at the Spring '88 DARPA contractor's meeting, we have the following yield models:

### Poisson:

$$Y = e^{-\lambda A}$$

Assumes a uniform distribution of point defects, *i.e.*, constant defect density. The Poisson model underestimates yield for larger chips, and is therefore applicable only to very small area devices.

### Murphy:

$$Y = \left(\frac{1 - e^{-\lambda A}}{\lambda A}\right)^2$$

Assumes defect density is Gaussian, with lowest value at the center of the wafer. This model is best when the expected yield $\geq 20\%$.

### Seeds:

$$Y = \exp(-\sqrt{\lambda A})$$

Assumes variable defect density; probability of large defect density is low and probability of small defect density is high. This model is best when the expected yield is $< 20\%$.

The expected defect density for a "well-controlled, mature" process is in the range of 1-3 defects/cm$^2$, with an average of 1.5, while for an "well-controlled, advanced" process is in the range of 1.5-3, with an average of 2.3. For "standard" processes, average defect density is 3.

For the EMC, which is 7.62 x 7.62 mm = 0.581 cm$^2$, the yields predicted are:

| $\lambda$ | Murphy | Seeds |
|---|---|---|
| 1.0 | 57% | 47% |
| 1.5 | 44.5% | 39% |
| 2.0 | 35% | 34% |
| 2.5 | 28% | 30% |
| 3.0 | 22% | 27% |

## Cost

# 2. HPNID 1.6μ (HPCMOS40) Fabrication

This section describes the electrical parameters (§IV.1.2.1) for the Hewlett Packard Northwest Integrated Circuits Division 1.6 micron nwell CMOS fabrication service and the pad library (§IV.1.2.2) designed for this fabrication. (This service is called "CMOS40" because its metal pitch is 4 microns.)

## 2.1 Electrical Parameters for HPCMOS40 Fabrication

### 2.1.1 Wiring Layers

The following capacitances and resistances were determined from HPNID's Electrical Design Rule document, assuming worst-case parameters. They have been scaled to "lamda" = 0.4 microns. For computing the maximum allowed currents in metal layers, see "CMOS40 Electrical Design Rules" for details.

| Layer(s) | Ca (af/λ2) | Cp (af/λ) | Rsh (Ω/square) | Imax (ma/λ) |
|---|---|---|---|---|
| Poly | 12.5 | 23 | 50 | |
| Metal-1 | 8.3 | 27 | 0.050 | 0.22 (0.065, many cuts) |
| Metal-2 | 4.5 | 26 | 0.044 | 0.32 (0.094, many vias) |
| NDiff | 36.8 | 100 | 70 | |
| PDiff | 107 | 81 | 150 | |
| NWell | 20 | 90 | 1750 | |
| M1/Poly | 18.4 | 38 | | |
| M2/Poly | 6.4 | 32 | | |
| M2/M1 | 13.6 | 41 | | |
| M1/Active | 18.4 | 38 | | |
| M2/Active | 5.6 | 31 | | |
| NDiff Cut | | | 45Ω/contact | 0.75ma/contact |
| PDiff Cut | | | 50Ω/contact | 0.75ma/contact |
| Poly Cut | | | 15Ω/contact | 0.75ma/contact |
| Via | | | 0.1Ω | 1.5ma/via |
| Channel | 235 | | | |

**Table 2.1.1:** Electrical Parameters for MOSIS Fabrication, Lambda = 0.4μ.

## 2.1.2 Transistors, Lambda = 0.4μ

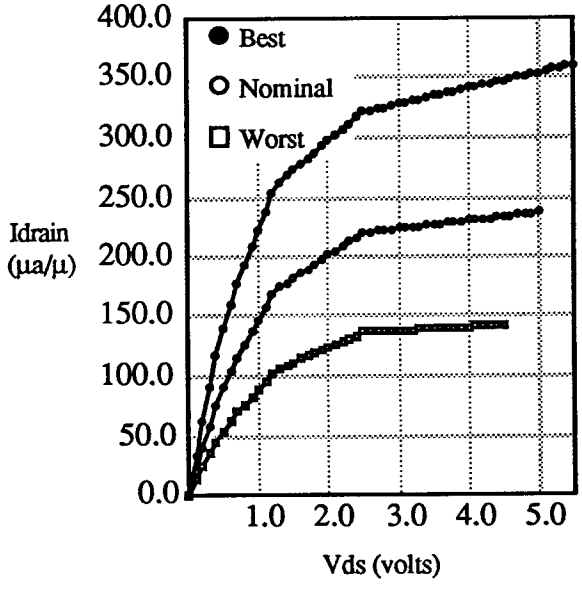
(dirs: ~jp/icfab/hpcmos40/dc)

Drain characteristics for Best, Worst, and Nominal models are shown on the next page. These characteristics were determined using CAzM simulation of 1.6μ/100μ devices under the following conditions:

| Simulation | Model | Vdd | Temp |
|---|---|---|---|
| Best | HP160CB | 5.5 volts | 25C |
| Worst | HP160CW | 4.5 volts | 100C |
| Nominal | HP160CN | 5.0 volts | 50C |

The characteristics are reported in μamps/micron of channel width.

**nfet Drain Characteristics (drain current per micron channel width with Vgate = Vdd).**




**pfet Drain Characteristics (drain current per micron channel width with Vgate = 0).**

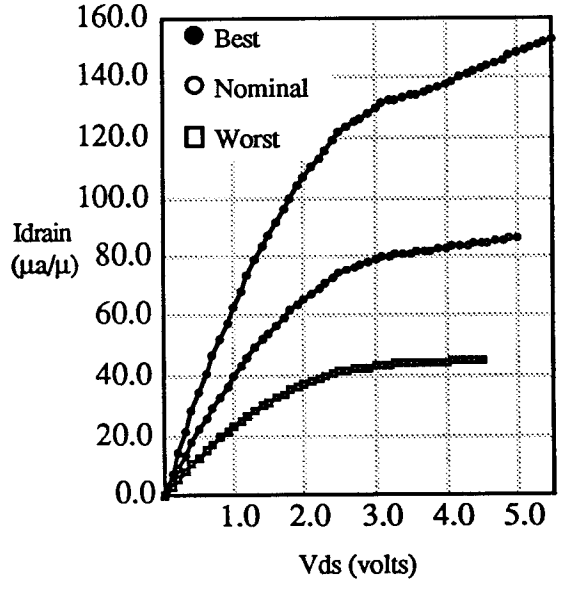



**Figure 2.1.1:** DC Characteristics of 1.6μ Transistors per μ Channel Width.

From these characteristics we can find the equivalent resistance of the channel in the saturation region, the equivalent resistance in the linear region, and the maximum drain current (Vds = Vdd):

| | Worst | Nominal | Best |
|---|---|---|---|
| Rsat (nfet) | 357K | 150K | 76.6K |
| Rlin (nfet) | 11.2K | 6.79K | 4.49K |
| Imax (nfet) | 142μa | 237μa | 361μa |
| Rsat (pfet) | 523K | 198K | 93.7K |
| Rlin (pfet) | 43.3K | 25.1K | 15.8K |
| Imax (pfet) | 45.3μa | 86.3μa | 153μa |

per μ channel width

**Table 2.1.2:** Rsat, Rlin, and Imax per μ channel width for various simulation parameters.

### Noise Modelling Considerations

(dirs: ~jp/icfab/hpcmos40/noise)

The models HP160CNN1 combines the Worst-case nfet with the Best-case pfet, but uses Worst-case values for Tox, Ld, and Wd. HP160CNN2 is Worst nfet/Best pfet, but with Best-case values for Tox, Ld, and Wd. Models HP160CNP1 and HP160CNP2 combine Best-case nfet and Worst-case pfet in a similar way. If we simulate to find the drain current in a wide (100μ) device under each model over the range of supply voltages, then find the ratio of pfet current to nfet current, normalized to the ratio under Nominal (Vdd=5.0v) conditions, we get:

| Model | Vdd=4.5v T=100C | Vdd=5.0v T=50C | Vdd=5.5v T=25C |
|---|---|---|---|
| **Worst** | 0.873 | 0.936 | 0.985 |
| **Nominal** | 0.928 | 1 | 1.065 |
| **Best** | 0.991 | 1.025 | 1.168 |
| **NN1** | 0.800 | 0.843 | 0.877 |
| **NN2** | 1.091 | 1.182 | **1.261** |
| **NP1** | **0.789** | 0.855 | 0.908 |
| **NP2** | 0.859 | 0.947 | 1.024 |

From this exercise, we determine that the extreme cases for noise are

| Model | Vdd | Temperature |
|---|---|---|
| HP160CNN2 | 5.5 | 25C |
| HP160CNP1 | 4.5 | 100C |

## 2.1.3 Transistor Widths

(dirs: ~jp/icfab/hpcmos40/twidth)

For layout of very-high-fanout structures, it is necessary to consider the propagation time down the RC network represented by the width of a very wide transistor. Using CAzM, 1.6μ length devices were modelled as a series of π-networks, each representing 1-unit transistors (1.6μ L x 2.8μ W); Rπ = 116.7 Ω, Cπ = 3293 af. Simulation shows that gate voltage at the far end of a 160μ wide transistor rises to 90% of its full swing in about 1.27 nsec. The effect is quadratic with gate width, so

**Width Rule: No single device shall be drawn wider than 375 lambda (150 μ).**

### 2.1.4 Speed Estimation

(dirs: ~jp/icfab/hpcmos40/tau)

We use the results of IV.1.A to estimate the speed of various circuits. Using HP160CW models, the worst-case parameters are graphed below:

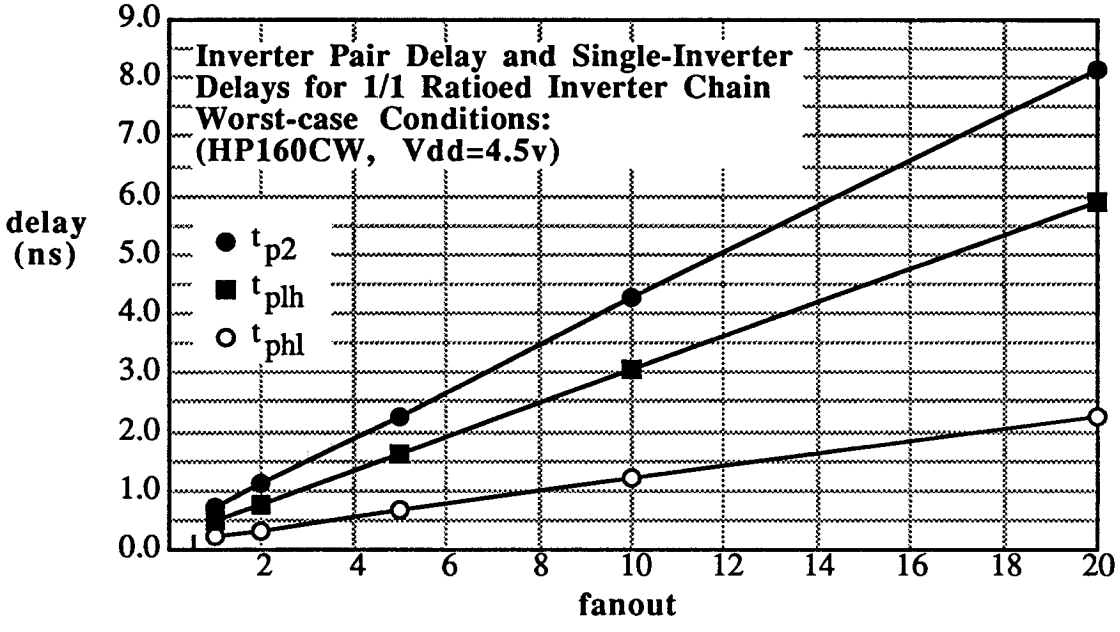



**Figure 2.1.2:** Worst-case Properties for 1/1-Ratioed Inverter Chain.

From the slopes of these plots, we abstract the speed parameters:

| Parameter | From slope of: | Worst | Nominal | Best |
|---|---|---|---|---|
| $\tau$ | $t_{p2}(f)$ | 195ps | 110ps | 67ps |
| $\tau_P$ | $t_{plh}(f)$ | 141ps | 77ps | 46ps |
| $\tau_N$ | $t_{plh}(f)$ | 54ps | 33ps | 21ps |

From the same circuit simulations that produce this data, we can abstract the rise and fall times; for the Worst case models, these are:

$$t_r = 0.3 + 0.61\ f \ \text{(nsec)}$$

$$t_f = 0.1 + 0.128\ f \ \text{(nsec)}$$

### 2.1.5 Optimum P/N Ratio

(dirs: ~jp/icfab/hpcmos40/beta)

To determine the optimum P/N ratio for a chain of inverters, we model a chain with fanout = 3; results for Worst-case conditions are shown below:

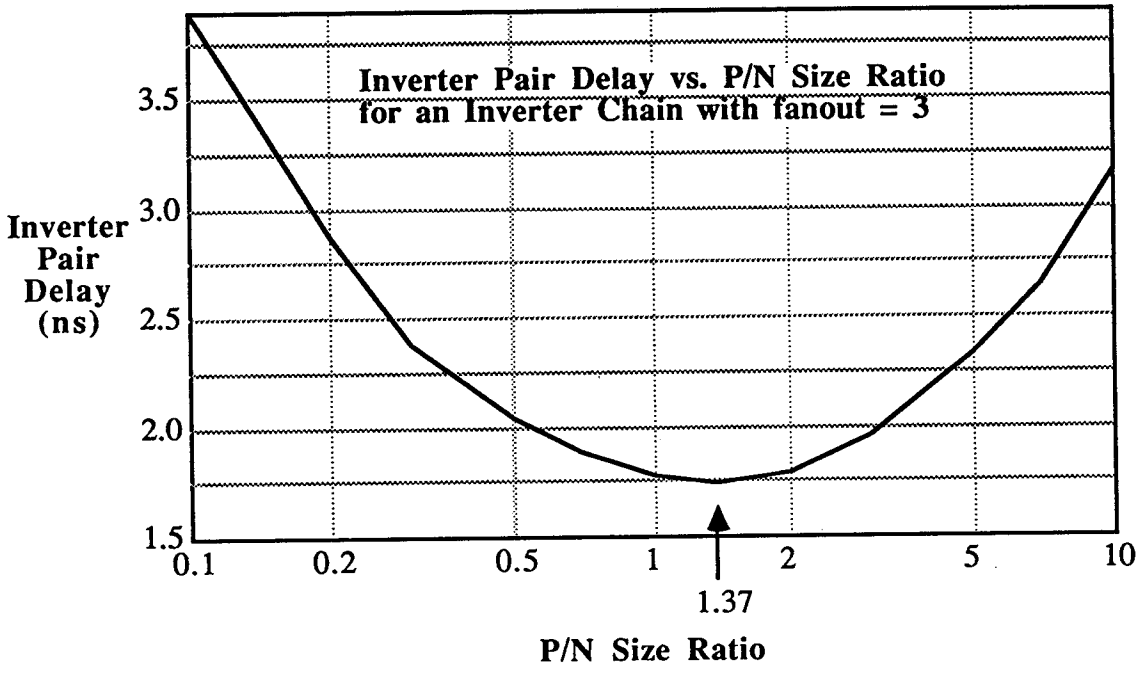



**Figure 2.1.3:** Pair Delay vs P/N Size Ratio for an Inverter Chain with Fanout = 3.

Notes

(1) The horizontal scale is logarithmic; the minimum is very shallow.
(2) The minimum occurs very close to the predicted minimum at $P/N = (\tau_P/\tau_N)^{1/2}$.
(3) Delay at $1 < P/N < 2$ differs from the optimum by only 3%.

### 2.1.6 Drain to Gate Capacitance Ratio

(dirs: ~jp/icfab/hpcmos40/Q)

Appendix A defines Q as the ratio of the parasitic capacitance on an inverter's drains to its input gate capacitance. This factor adds to the fanout of an inverter. It is layout sensitive, of course; values of Q for various layout styles can be used to predict how far the speed of a circuit is diminished by parasitics in given style. The results of circuit extraction for several simple inverter layouts are shown below:

| Layout style | $W_P/W_N$ | $C_{gate}$ | $C_{in}$(para) | $C_{in}$(total) | $C_{drain}$ | Q |
|---|---|---|---|---|---|---|
| 1/1 inverter, exposed drains | 3.2μ/3.2μ | 14.0f | 3.0f | 17.0f | 19f | 1.12 |
| 2/1 inverter, exposed drains | 6.8μ/3.2μ | 22.1f | 3.0f | 25.1f | 29f | 1.16 |
| 2/1 inverter, shared drains | 13.6μ/6.4μ | 44.2f | 7.0f | 51.2f | 31f | 0.61 |
| 1/1 inverter, shared drains | 28μ/28μ | 123.6f | 7.0f | 130.6f | 69f | 0.53 |
| Huge 1/1 inv., shared drains | 160μ/160μ | 707f | 29f | 736f | 382f | 0.52 |
| 1/1 inverter, annular trans's | 24.4μ/24.4μ | 107.8f | 4.0f | 111.8f | 27f | 0.24 |

The table indicates that for the best straight-gate layout style, Q's of about 0.5-0.6 can be achieved. For layout style typical of low-fanout multi-level logic, Q is typically about 1.15. The very best that can be achieved is Q=0.24 for annular transistors. Note that "bent" gates are not very desirable (according to considerable hearsay), so we will not be using such layout strategies in any of the Pxpl5 custom chips.

The critical, high-fanout circuits in our designs (pads, clock generators, control-line drivers, etc.) will be layed out using the shared drain approach with Q ~ 0.55. From Appendix A, Eqn A.4, this gives an optimum number of stages for an inverter chain of

$$n = \frac{1}{1+Q/e} \cdot \ln\left(\frac{S_n}{S_0}\right) = 0.8317 \ln\left(\frac{S_n}{S_0}\right)$$

The optimum fanout per stage is, then,

$$f_{opt} = \left(\frac{S_n}{S_0}\right)^{\frac{1}{n}} = \left(\frac{S_n}{S_0}\right)^{1/[0.832 \cdot \ln(\frac{S_n}{S_0})]} = \exp(1.202) = 3.33$$

### 2.1.7 Wire Sizing to Avoid Electromigration

In Appendix C we develop rules of thumb for determining the effective (RMS) current used to compute miminum wire size needed to avoid early electromigration failures. "CMOS40 Electrical Design Rules" gives maximum allowable currents, and describes de-rating for width and for via/contact placement for the two metal layers. The following table summarizes the maximum capacitance load that can be supplied by various wire sizes for each of the two metal layers with various numbers of contacts; the table applies the rules of thumb of Appendix C and assumes 5.5 volt operation at 50 MHz.

| Width (lambda) | # Cont's/Vias | Metal-1(NRZ) Max Cap | Metal-1(RZ) Max Cap | Metal-2(NRZ) Max Cap | Metal-2(RZ) Max Cap |
|---|---|---|---|---|---|
| 7 | 0 | 1.7 p | 1.0p | 2.0p | 1.2p |
| 8 | 0 | 2.1p | 1.3p | 2.4p | 1.4p |
| 10 | 0 | 2.8p | 1.7p | 3.1p | 1.9p |
| 12 | 0 | 3.5p | 2.1p | 3.8p | 2.3p |
| 14 | 0 | 4.3p | 2.6p | 4.5p | 2.7p |
| 16 | 0 | 5.0p | 3.0p | 5.3p | 3.2p |
| 7 | 1 | 0.27p | 0.16p | 0.55p | 0.33p |
| 8 | 1 | 0.64p | 0.38p | 0.91p | 0.55p |
| 10 | 1 | 1.4p | 0.82p | 1.6p | 1.0p |
| 12 | 1 | 2.1p | 1.3p | 2.4p | 1.4p |
| 14 | 1 | 2.8p | 1.7p | 3.1p | 1.9p |
| 16 | 1 | 3.5p | 2.1p | 3.8p | 2.3p |
| + per lambda | 0 | 0.36p/lambda | 0.22p/lambda | 0.36p/lambda | 0.22p/lambda |

## 2.2 HPCMO40 Pad Library

### 2.2.1 Layout Strategy

### Outside Dimensions

We will likely be fabricating both the IGC and the EMC for Pxpl5 using the MOSIS 1.6μ service, but with non-scalable, hpcmos40, rules. It may be advantageous to use one of the MOSIS standard pad frames for these projects. These pad frames impose some constraints on pad placement and size within an I/O cell. The pad frames document defines pad-to-pad distance and pad-to-edge distance:

| Frame Designation | Frame Size | Pin Count | Pad to Edge | N/S Pad/Pad | N/S Pad/Corner | E/W Pad/Pad | E/W Pad/Corner |
|---|---|---|---|---|---|---|---|
| 64P79X92 | 7900x9200 | 64 | 250 | 400 | 950 | 500 | 850 |
| 84P79X92 | 7900x9200 | 84 | 250 | 300 | 950 | 375 | 850 |
| 40P69X68 | 6900x6800 | 40 | 200 | 500 | 1200 | 500 | 1150 |
| 64P69X68 | 6900x6800 | 64 | 200 | 300 | 1200 | 300 | 1150 |
| 84P69X68 | 6900x6800 | 84 | 200 | 300 | 450 | 300 | 400 |
| 40P46X68 | 4600x6800 | 40 | 150 | 300 | 950 | 500 | 1150 |
| 64P46X68 | 4600x6800 | 64 | 150 | 256 | 380 | 300 | 1150 |

**Table 2.2.1:** MOSIS Standard Pad Frame dimensions.

Of the two projects that will use hpcmos40 technology, the IGC will likely be pad limited and will use a 132-pin PGA package, recently announced by MOSIS. We here calculate the necessary pad spacing for a 132-pin chip in such a package: MOSIS requires that exactly 1/4th of the pins be placed on each edge of the package, and that they be spaced equally on the Pad/Pad spacing; this spacing will, in general, be different for the North/South and East/West sides. It is likely that the 132-pin standard frame, when announced, will specify a 7900x9200 micron frame size. Assuming that an I/O cell has width equal to the minimum Pad-to-Pad spacing, then this distance must satisfy

$$7900 = 33 \cdot (\text{Pad-to-Pad}) + 2 \cdot (\text{Pad-to-Corner})$$

If we allow 850μ for Pad-to-Corner, then Pad-to-Pad comes to ~188μ. There were some rumours at the Spring '88 DARPA Microsystems meeting that there is an industry standard of 200μ (0.008") Pad-to-Pad distance, which would leave 650μ for Pad-to-Corner.

We will specify and design, therefore, a library of pads for pad-limited designs. These will be "tall-thin" designs that will fit the narrowest Pad-to-Pad distance, and will be designated by names with root **Pad160a**. All pads in this library will be designed to fit in all of the larger pad frames, and to accomodate our best guess at the forth-coming 132-pin pad frame. Thus, both Pad-to-Pad and Pad-to-Edge distance will be 200μ (500 lambda), except for the clock generator functions, which require Pad-to-Pad distances of 240μ (and must be mounted on the wide edges of the pad frame).

The EMC is not a pad-limited design, but is instead package-limited. The chip must fit within a 7.5 mm square in order to fit in a 44-pin PLCC package. (It would be extremely handy if the project fit within the MOSIS 6.9x6.8mm standard frame; MOSIS penalizes users who do not use standard project sizes by putting them at the bottom of the fab queue.) For this chip we will

specify and design a separate "short-fat" pad library, designated **Pad160b**. This pad library will have a Pad-to-Edge distance of 200μ (100μ for a power rail, 100μ square pad, and 50μ space between rail and pad). The distance from the lower-left corner to the pad center will also be 100μ for these pads. The width of these pads will be variable, to allow for a variety of functions.

Dimensions for the two pad families are shown in Figure 2.2.1.

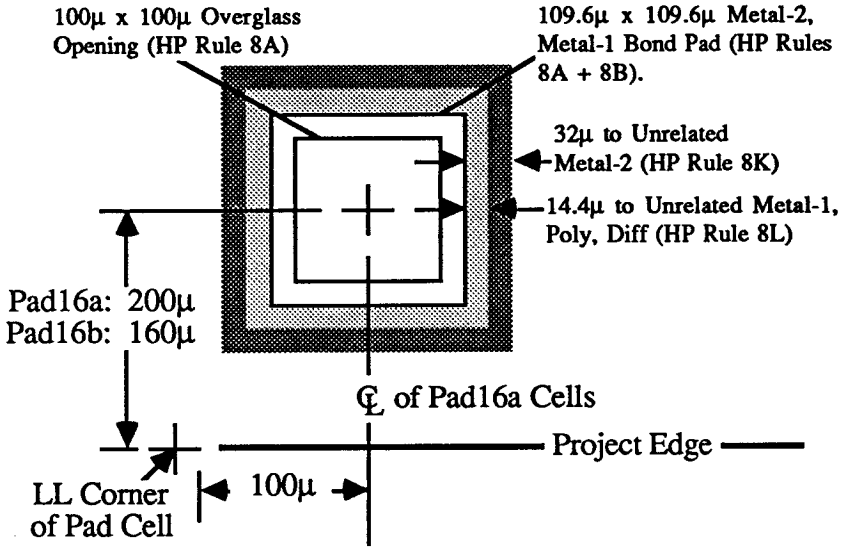



**Figure 2.2.1:** Design-rule driven overall pad dimensons for HPCMOS40 Pad Libraries.

### Power Rail Sizing

It is desirable to support a rule for output pads of the form: "Place one Vdd and one GND pad per every *n* output pads". A reasonable worst-case value for *n* is 4. Consider the situation of four output pads driving probes on the Mega-1 tester at 40 MHz (NRZ); the tester is reputed to present a load of 80pf of shunt capacitance. Under these conditions,

$$I_{switch} = C\,V\,f = 80\times10^{-12} \cdot 5 \cdot 20\times10^{6} = 8 \text{ ma}$$

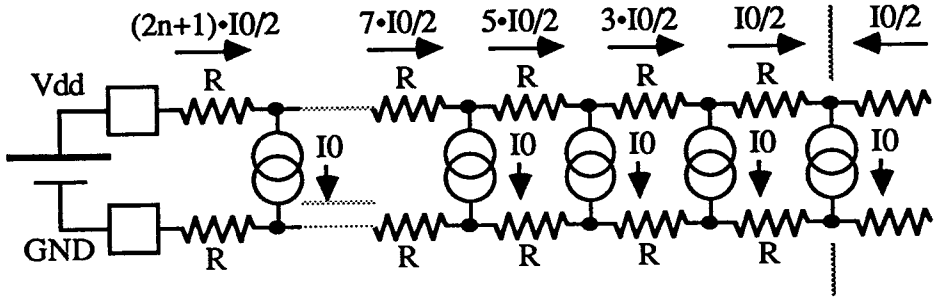
Ignoring static current, the power rails should be sized to handle 32 ma.

We assume that there will be vias in both metal-2 power rails, spaced at the pitch of fet's layed out using the "tuning-fork" approach (28 lambda = 11.2 μ). We need to derate the width of the metal-2 rail by 4 lambda (1.6μ) for each via, so each 28 lambda of width should be current rated as though it were 24 lambda wide. A 28-lambda metal-2 rail with one via can safely carry 24•0.32 = 7.68 ma, so the switching current requirements imply a 120 lambda (48μ) power rail.

Another worrisome situation arises with input-only pads supplied by a minimal number of power pads; we want to make sure that dI/dt noise on these rails can never turn on any devices. Consider the (extreme) situation of 29 input pads on one edge of a 132-pin chip, all supplied from sets of power pads at each corner. Suppose all of the input pins are driving pairs of complementary 10pf loads within the chip, and the the rise and fall time of the signals is 3nsec, then each pad must supply a peak current of

$$I_{peak} = \frac{10\ pf \bullet 5.5\ volts}{3\ nsec} = 18.3\ ma$$

to each of the input pads. As shown below, each pad also consumes about 2.2 ma of static current. Assume the pads are spaced evenly along the rails, then the following situation obtains:




The voltage drop along the rails for n = 14 is

$$V_{drop} = \frac{I_0 R}{2} \sum_{i=1}^{14} (2i - 1) = I_0 R \sum_{i=1}^{14} i - \frac{14 I_0}{2} = I_0 R\ (105 - 7) = 98\ I_0 R$$

We want to keep the voltage drop well below the threshold voltage of either transistor, and small enough to prevent significant forward conduction in any diodes. Suppose we require Vdrop < 0.5 volts, then

$$R < \frac{0.5\ volts}{98 \bullet 20.5\ ma} = 250\ m\Omega$$

Assume the worst case, that the pads are distributed uniformly along the 9200μ side of a standard pad frame, that there is 650μ of Pad-to-Corner space, then each pad has its rails extended across (9200μ - 1300μ)/33 = 240μ. Metal-2 resistance, from Table 1.1.1, is 44mΩ, so each 240μ-long power rail segment can have at most 250mΩ/44mΩ = 5.68 squares of metal-2 in each rail, requiring a rail width of 42.2μ. Using the above via derating of 28/24, this becomes 49μ or 122 lambda. To be on the safe side:

**Power rails in HPCMOS40 pads shall be 150 lambda (60μ) in width or greater.**

## Input Protection

Following the conclusions of Appendix B, we will use the native diodes to protect input gates on input-only pads. These diodes actually form parasitic bipolar transistors, and we can use the emitter-collector current to advantage to shunt ESD events. We do not want to induce latchup, however, when the input diodes are forward biased. Latchup prevention strategy will follow that suggested by Lance Glasser ("Design and Analysis of VLSI Circuits", pages 295-301). For the Pad160a designs, we will also place these circuits, along with their own power rails toward the outside of the project, on the opposite side of the pad site from the active input circuitry. At the same time, we can take advantage of the large nwell around the diodes to build in some supply bypassing; we make the well area as large as possible, and tie it to the Vdd rail with wide metal and many contacts.

## Guard Rails

In addition to the guard rings intrinsic in the ESD structures, we will also require that all active circuitry be completely surrounded by guard rings heavily strapped with metal. Following the reccommendations in MOSIS's old 3μ pwell documentation, we place on the inner edge of the standard pad layout a pair of guard rings, N+/nwell toward the outside, P+ toward the inside of the project. These will have their own, separate metal conductors back to the power pads, so that dI/dt noise induced in the I/O circuitry will change their potentials as little as possible.

Overall layout strategy for the Pad160a cells is the same as that for the 2.0μ library (see Figure 1.2.2). For the Pad160b library, the layout strategy is:

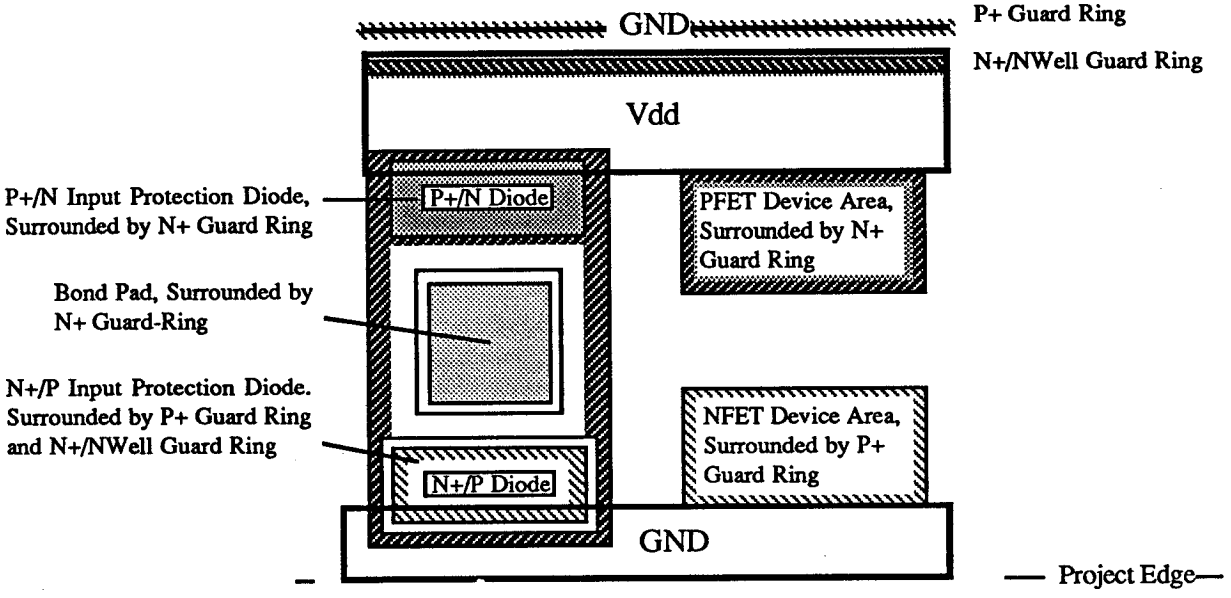
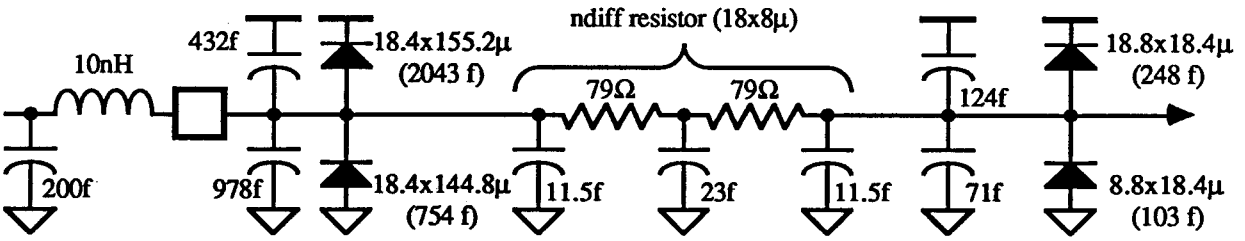



**Figure 2.2.2:** Pad layout strategy for Pad160b Library.

### 2.2.3 Input Protection Network (dirs: ~jp/pads/hpcmos40/SIM/inprot & ~////tsprot)

The initial cut at an input protection network, with "large" diodes connected directly to the pad metal and ~150Ω series resistor to a second set of (smaller) diodes, gives the following extracted circuit:




Scaling the saturation currents as indicated in Appendix B.2, we get the following models for the diodes (with conservative values of Rs, and M and Vj from CMOS40 EDR document):

.model big_n diode Is=10.2e-15 Rs=3 n=1.03 Cjo=754e-15 M=0.5 Vj=0.8
.model big_p diode Is=5.34e-13 Rs=3 n=1.24 Cjo=2043e-15 M=0.5 Vj=0.9
.model small_n diode Is=1.33e-15 Rs=10 n=1.03 Cjo=103e-15 M=0.5 Vj=0.8
.model small_p diode Is=0.647e-13 Rs=10 n=1.24 Cjo=248e-15 M=0.5 Vj=0.9

**ESD Thermal & Electrical Behavior.** Electrical simulation indicates that the voltage at the pad reaches only about 4.8 volts during a 2000-volt ESD event, and the voltage on the protected node reaches only about 1.1 volts; ESD maximum current is about 1.33 amps. For a 5000-volt ESD event, the voltage on the pad reaches about 10.7 volts, the voltage on the protected node reaches 1.50 volts, and the current is about 3.33 amps. Thermal simulation, using the finite element analysis of Appendix B indicates that the peak temperature rise on the pad-shunt diodes is about 13.5C for a 2000-volt event and about 70.8C for a 5000-volt event. This ESD circuit should prove fairly bullet-proof.

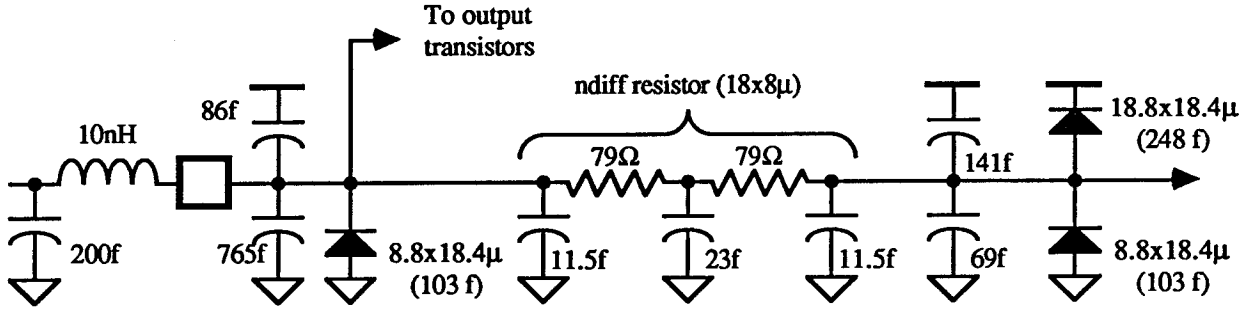
**Performance.** Performance was simulated by driving the circuit above with a voltage source toggling between 0.4 volts and 2.4 volts, and deterimining the propagation delay between input and output passing through 1.4 volts. Loads were symmetrical cmos inverters. The results produce the relationship

$$t_{p(InProt)}(psec) = 65 + 0.41 \cdot W_{channel}\ (microns)$$

where $W_{channel}$ is the *total* width of pfet and nfet channels in the load (assuming minimum-length devices, l = 4 lambda).

Equivalent input capacitance (assuming the zero-bias junction capacitances) for the Pad16a version of this circuit is 4.80 pf; the Pad16b version has slightly different parasitics (but is otherwise closely similar)--it's equivalent capacitance is 4.47 pf.

**Tri-State Input Protection.** For tri-state (bi-directional) pads, the input buffer's gates are protected at the pad node by the large output transistors. We provide a resistor and gate-shunt diodes to protect the input buffer further. The extracted circuit is:




Equivalent input capacitance for this circuit is 1.56 pf.

## 2.2.4 TTL/CMOS Input Buffer

The purpose of this critical circuit is to convert signals from the EMC's environment, generated by FAST-series TTL, bus-oriented parts, into the form needed by the EMC's internal circuitry. The most difficult problem is the signal-level conversion of the TTL logic signal, centered at 1.4 volts, to the Vdd/2-centered CMOS internal form. From the Fairchild FAST Databook:

| Sym. | Parameter | Min | Typ | Max | Units | Conditions |
|---|---|---|---|---|---|---|
| $V_{IH}$ | Input HIGH Voltage | 2.0 | | | V | |
| $V_{IL}$ | Input LOW Voltage | | | 0.8 | V | |
| $V_{OH}$ | Output HIGH Voltage, Std Logic, 3-State | 2.7 | 3.4 | | V | $I_{OH}$=-1ma, Vdd=4.5v |
| $V_{OH}$ | Output HIGH Voltage, 3-St, Line Driver | 2.7 | 3.3 | | V | $I_{OH}$=-3ma, Vdd=4.5v |
| $V_{OH}$ | Output HIGH Voltage, Line Driver | 2.0 | 3.1 | | V | $I_{OH}$=-15ma, Vdd=4.5v |
| $V_{OL}$ | Output LOW Voltage, Standard Logic | | 0.30 | 0.5 | V | $I_{OL}$=20ma, Vdd=4.5v |
| $V_{OL}$ | Output LOW Voltage, 3-State | | 0.35 | 0.5 | V | $I_{OL}$=24ma, Vdd=4.5v |
| $V_{OL}$ | Output LOW Voltage, Line Driver | | 0.42 | 0.55 | V | $I_{OL}$=64ma, Vdd=4.5v |

**Table 1.2.1:** FAST-TTL signal levels

From the min and max of VIH and VIL, respectively, we require that an input pad on the EMC

(1) have a switching threshold of 0.8 + 1/2(2.0 - 0.8) = 1.4 volts
(2) that the circuit transform an input voltage of 2.0 volts or higher as a HIGH over all process, temperature, and Vdd variations, and
(3) that the circuit transform an input voltage of 0.8 or lower as a LOW over all process, temperature, and Vdd variations.

We use an inelegant, but simple approach to designing this circuit: a simple inverter, ratioed appropriately to lower its switching threshold, will serve as the input buffer for our input pads. It will be followed by a second inverter, ratioed to provide approximately equal delays for rising and falling inputs through the pair of inverters, and sized to optimize speed/power for the pair.

### 1.2.4.1 DC Characteristics

(dirs: ~jp/pads/hpcmos40/SIM/ttl_dc)

To determine the optimum ratio Wp/Wn for the input inverter, we found the drain characteristic for 100µ-wide devices (using CAzM with Nominal models) with Vgs = 1.4 volts for the nfet, and Vgs = -3.6 volts for the pfet. We use devices here drawn 2.0µ long, on advice of HPCMOS40 Electrical Design Rules not to use minimum-length devices in input circuits. Taking the ratio of these currents with Vds = 2.5 volts for both devices,

$$\frac{W_P}{W_N} = \frac{0.8636}{3.7578} = 0.230$$

The transfer characteristics of a ratioed inverter with Wp/Wn = 0.230 are shown in Figure 2.2.3.

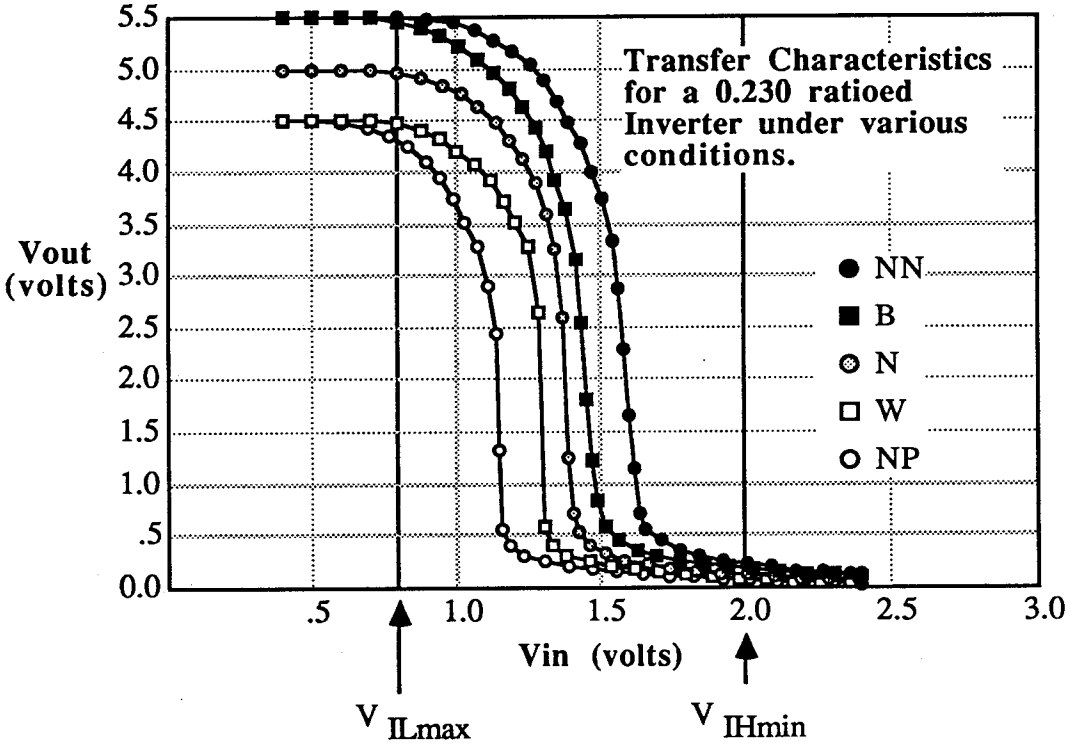
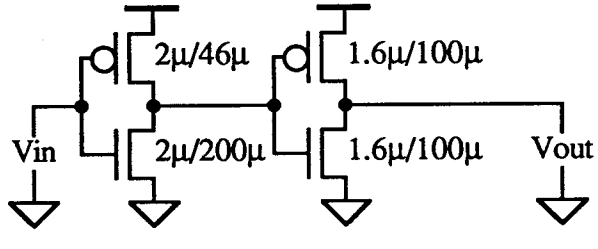



**Figure 2.2.3:** Drain Characteristics for fet's at TTL swithching threshold.

Choosing the second inverter ratioed Wp/Wn = 1, and of the size shown, will satisfy this requirement for a load of up to 2pf.



The transfer characteristics of the complete inverter pair are shown in Figure 2.2.3.

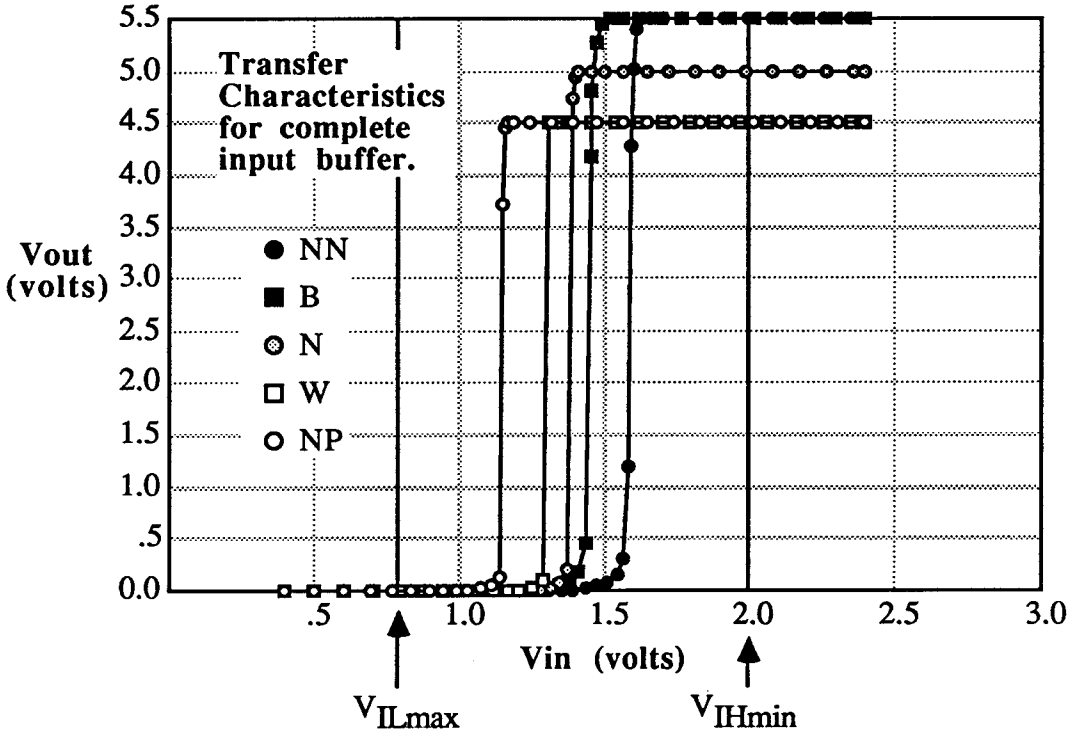



**Figure 2.2.3:** Drain Characteristics for fet's at TTL swithching threshold.

### 2.2.3.2 Transient Characteristics

(dirs: ~jp/pads/hpcmos40/SIM/ttl_ac)

We now want to optimize the size of the second inverter relative to the first. As the first inverter is made larger, the circuit should operate faster up to some threshold, where the speed of the overall circuit is driven mainly by the second inverter. When converting a logic-HI input, the first inverter consumes static current. As the first inverter is made larger, the static current increases, and the load on the input pad increases. We look for the point at which trading current and input drive for speed begins to yield diminishing returns. Simulation suggests that the circuit used to generate Figure 2.2.3 is about optimal, though this is a difficult optimization. A simple tau-based analysis is not useful; because both transistors of the input stage are in saturation for a HI input, the usual tau model doesn't work.

Rather than try to perform an optimization, we will take as the standard input converter the circuit above (it appears to have adequate performance). We present the speed and power consumption (averaged over a cycle) of a circuit with an output inverter ratioed at 1/1 and "reasonable" drain parasitics (Q ~ 0.55).

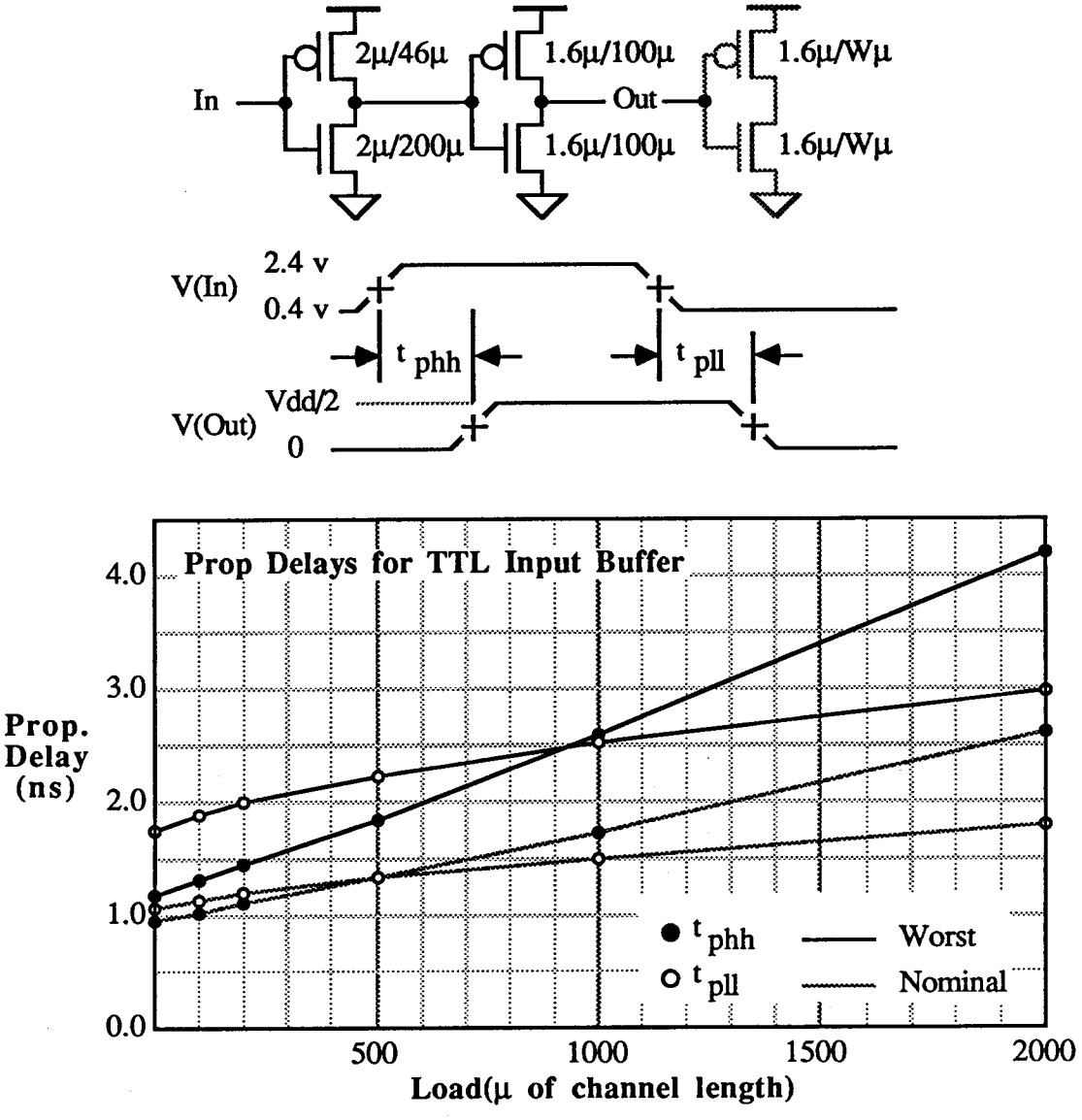



**Figure 2.2.4:** Transient characteristics of TTL input buffer.

From this graph, one can optimize the width-ratio for the output inverter of the buffer. Static current for a HI input, and average power consumption are:

| Model: | Power (mW) | $I_{HI}$ |
|---|---|---|
| Worst | 1.28 | 0.32 |
| Nominal | 2.62 | 0.82 |
| Best | 5.14 | 1.84 |

Power consumption and static current scale roughly linearly with the size of the input inverter. From the graph of Figure 2.2.4, we approximate the Worst-case speed performance by the expression

$$t_{pin}(\text{nsec}) = 1.5 + 0.105\ f$$

where f is the fanout from the second inverter.

### 2.2.4 Clock Generator

(dirs: ~jp/pads/hpcmos40/SIM/clkgen/hand)

A first cut at optimization of the clock generator leads to the following circuit

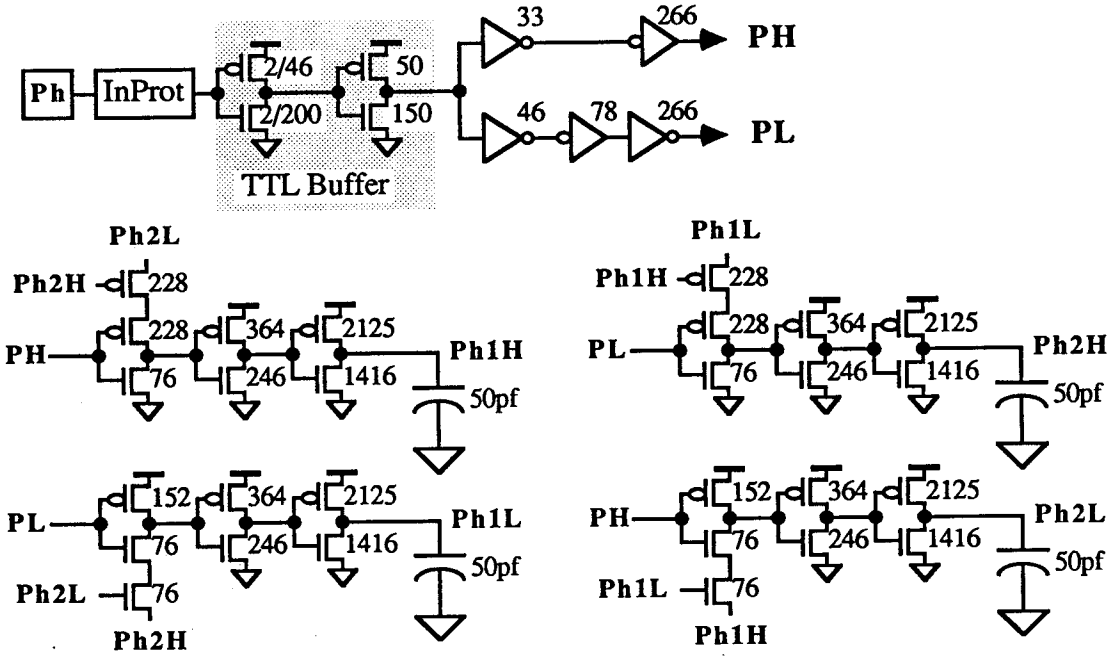



**Figure 2.2.5:** First cut at 1.6μ clock generator.

For "reasonable" sized clock loads (50pf per phase), this circuit seems to be the minimum size that will guarantee operation at 50 MHz clock over all process variations. We note the following observations, gathered during the optimization/simulation process:

(1) The output buffers had to be ratioed about 1.5:1 (pfet/nfet width) to get sufficient output voltage under Worst and NoiseP conditions.

(2) The gating used to ensure non-overlap had to be ratioed somewhat asymmetrically for best rise time and sufficient non-overlap.

(3) It was not necessary (or desirable) to ratio the equal-delay inverter chains.

(4) It was necessary to offset the ratio of the output inverter in the TTL input buffer to make the two clock phases approximately equal in length under Worst and NoiseP conditions.

(5) Relative sizes were "optimized" using a spread sheet and an algebraic approximation to the behavior of each component in the generator. Parasitics for source/drain terminals were approximated assuming a layout style with Q = 0.55.

(6) This circuit will not operate properly at 50MHz with clock input assymetries greater than about 3% or so under Worst-case conditions. It will, however, meet the required specs at 40MHz.

Below are the clock waveforms and defining parameters:

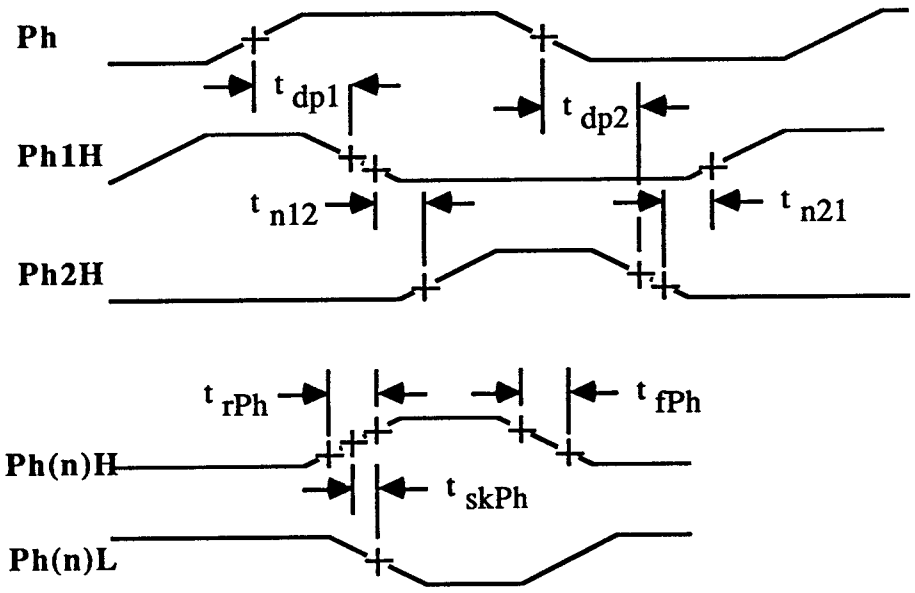



**Figure 2.2.6:** Waveforms and characteristic times for Clock Generator.

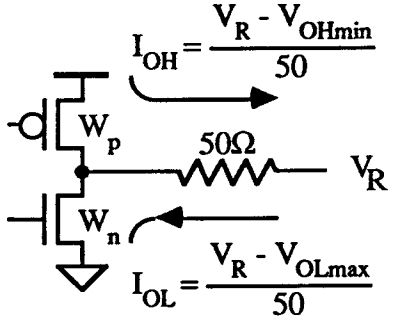
For this circuit, with source/drain parasitics crudely estimated (and no other parasitics included), we get, for Worst-case simulation (Pd and Imax are for f=50MHz):

| Parameter | Worst | Nominal | Best | NoiseN | NoiseP |
|---|---|---|---|---|---|
| $t_{dp1}$ | **6.93** | 4.44 | 2.92 | 3.46 | 6.17 |
| $t_{n12}$ | 1.74 | 0.97 | **0.56** | 0.71 | 1.32 |
| $t_{dp2}$ | **7.34** | 4.31 | 2.63 | 2.70 | 6.93 |
| $t_{n21}$ | 1.84 | 1.03 | **0.60** | 0.73 | 1.42 |
| $t_{rPh}$ | **3.18** | 1.87 | 1.20 | 1.20 | 3.15 |
| $t_{fPh}$ | **1.98** | 1.19 | 0.79 | 0.84 | 1.89 |
| $t_{skPh}$ | 0.41 | 0.20 | 0.12 | 0.16 | **0.62** |
| $P_d$ | 315mW | 402mW | **491mW** | 491mW | 320mW |
| $I_{max}$ | 427ma | 822ma | **1,377ma** | 1,377ma | 425ma |

Where the Worst-case values of the various parameters are shown in bold.

### 2.2.5 Output Pad Considerations

We would like the output pads in this family to be able to meet the TTL specifications for $V_{OH}$ and $V_{OL}$ (Table 3.1.1) when driving a 50Ω load, returned to any convenient termination voltage between 0 and Vdd. For convenience of layout, we choose Wp and Wn equal for the output transistors. What is the minimum W (= Wp = Wn) and the $V_R$ that will meet the objective?




There is a potential problem with using minimum-length devices; it may be physically impossible to squeeze enough contacts and vias along their width to handle the maximum possible drain currents. For this reason, it may be judicious to use somewhat longer transistors in output drivers. Drain currents for l = 2.0μ devices under Worst-case conditions are:

$$I_{Dn} = 34.219\mu a/\mu \text{ when } V_{ds} = 0.4 \text{ volts}, V_{gs} = 4.5 \text{ volts}$$

$$I_{Dp} = 30.984\mu a/\mu \text{ when } V_{ds} = 4.5 - 2.4 \text{ volts}, V_{gs} = -4.5 \text{ volts}$$

from which

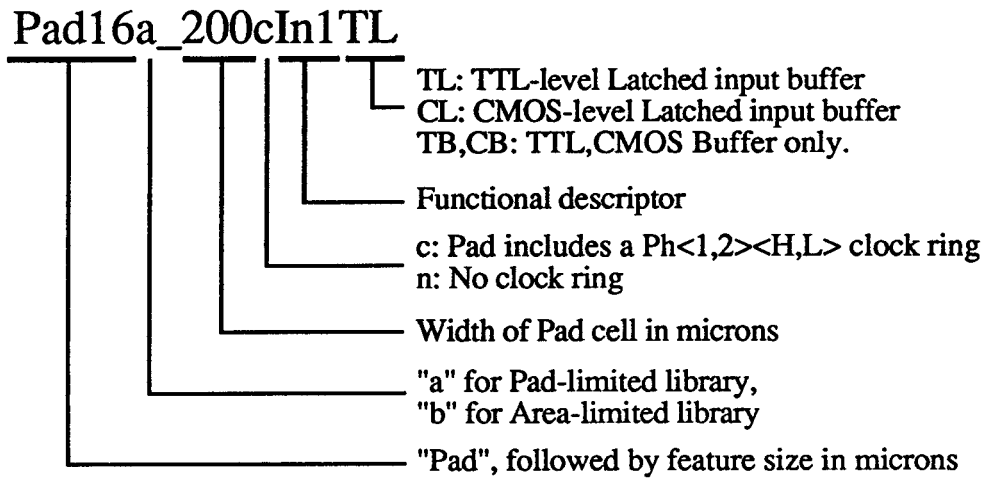
$$I_{OH} = W_P(\mu) \bullet\text{-}30.984\mu a = \frac{V_R - V_{OHmin}}{50}$$

$$I_{OL} = W_n(\mu) \bullet 34.219\mu a = \frac{V_R - V_{OLmax}}{50}$$

Setting $W_n = W_p = W$, and solving for W and $V_R$, we get W = 613μ and $V_R$ = 1.45 volts. An output pad sized with $W_p = W_n$ = 620μ should therefore just satisfy the output drive requirements for TTL-level output into 50Ω. Under Best-case conditions, we can determine the operating point of the pfet with a 50Ω load returned to 1.45 volts, with Vgs = -5.5 volts; the load line intersects the drain curve at $I_{Dp}$ = 46.3 ma. The operating point with nfet turned on (Vgs = 5.5 volts) is $I_{Dn}$ = 25.3 ma. 46.3 ma is the maximum current that can flow in the output circuit; the most conservative practice would require the metal wire, vias, and contacts on the pfet to be sized to handle this current. Metal-1, without vias, would be 46.3/0.75 = 63μ wide (157 lambda); we need 46.3/0.75 = 63 contacts and 46.3/1.5 = 31 vias. If we take the average of the currents, assuming the output spends half its time HI and half LO, then 36ma/0.75ma = 48μ (120 lambda).

## 2.2.6 HPCMOS40 Library Cells (dirs: ~jp/pads/hpcmos40/SIM; ~vlsi/lib/pads/hpcmos40)

### Naming Convention




### Schematic Conventions

All transistors are assumed to have minimum (1.6μ) length, unless shown otherwise, and are labelled with their width in microns. Inverters with equal-sized pfet and nfet are labelled with the width of either device. Input protection circuits for input and bi-directional pads are labelled "InProt" and "TSProt"; their detailed schematics are shown above in Section 2.2.2.

### Electrical Conventions

All cells are expected to operated with Vdd in the range of 4.5 volts ≤ Vdd ≤ 5.5 volts. Power consumption for each cell can be estimated by applying the expression

$$P_D = (C_{pd} + \sum_{loads} C_{load}) V_{dd}^2 f + I_{dd} V_{dd}$$

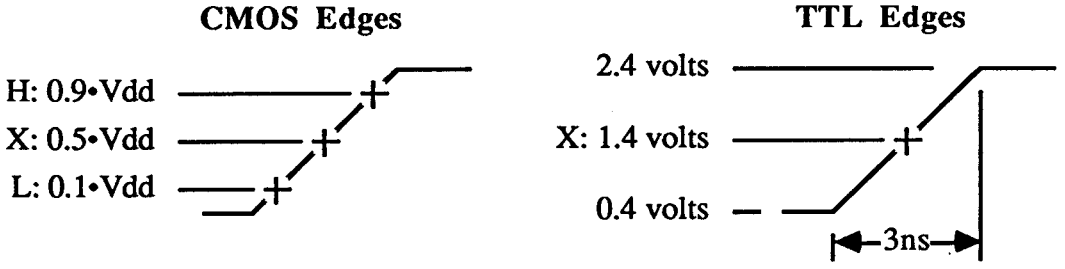
Where $C_{pd}$ and $I_{dd}$ are given in the data sheets.

Transfer characteristics for input buffers are measured by determining the maximum input LO voltage at which the output is LO (for non-inverting buffers and latches) and the minimum input HI voltage at which the output is HI. Output HI and LO are defined by 0.9•Vdd and 0.1•Vdd respectively.

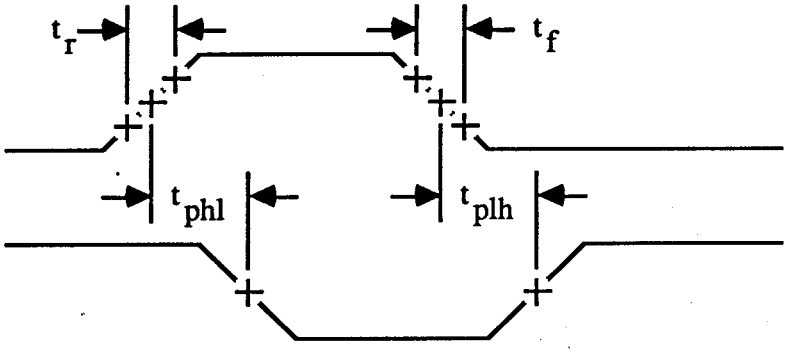
Input/output capacitances are figured by adding the extracted parasitic capacitances to an estimate of the capacitance of gates connected to a node. Gates are computed using 2.352 ff per micron of gate width.

## Waveform and Timing Conventions

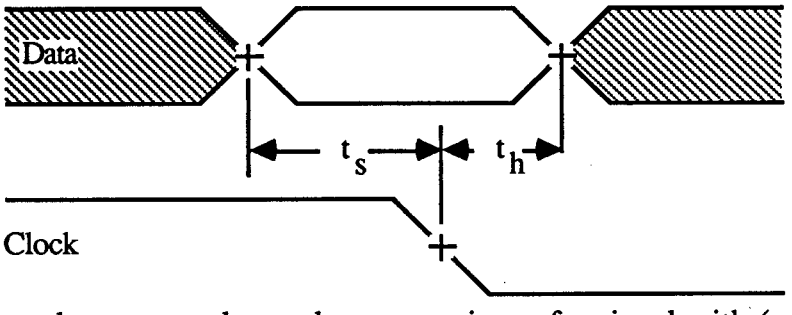
For each timing edge we define a points L,X, and H as follows:



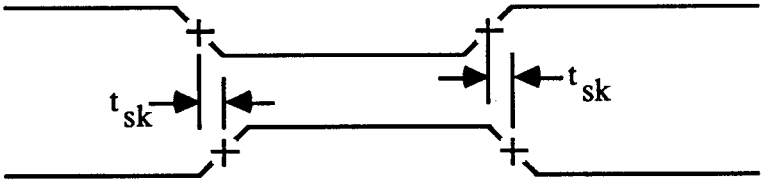

Rise and fall times, and various flavors of propagation times are defined in the usual way:




All latches in this pad library are transparent during Ph2, and latch on the trailing edge of Ph2 to produce an **SP1** output. Setup and hold times for these Latched input pads are defined with respect to the Ph2 clock as follows:

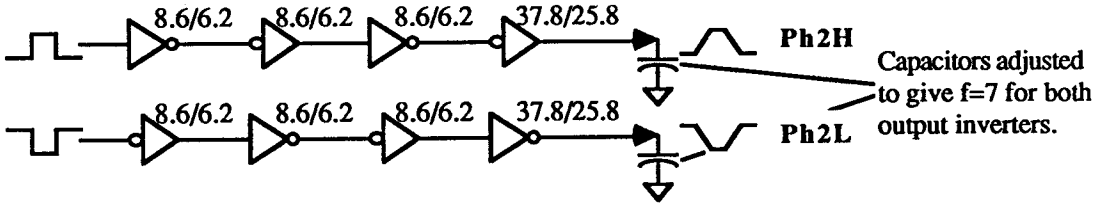



Several pad types produce true and complement versions of a signal with (presumably) equal delay. The skew between these, and other, complementary signals is defined as the maximum skew between any two associated edges:



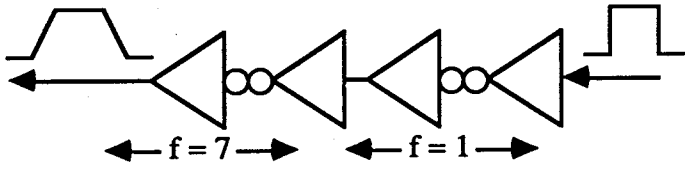
## Clock/Data Inputs; Loads

Clocks for these pads are modelled using the circuit below:

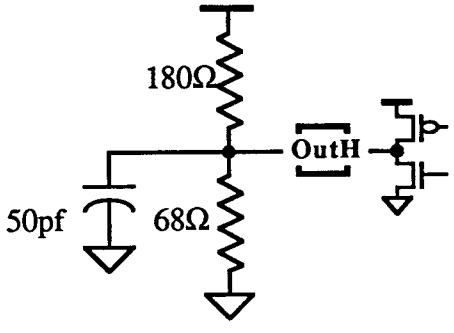



The output inverters are sized so that a load of 1pf is a fanout of 7. For each of the pad circuits, the effective load capacitance for each clock phase is determined; the loading for the clock circuit above is adjusted with additional fixed capacitors to give the correct fanout.

Data inputs to pads are likewise modelled using an inverter chain where the output stages are sized for a fanout of 7:




Output structures are sized to drive a 50Ω load returned to 1.45 volts; using the nearest available standard resistance values, we use the following circuit as a standard load for evaluating a pad's performance driving a "TTL" load:




Pads should never be terminated with an equivalent resistance smaller than 50Ω. DC and transient performance are determined using TTL levels: $V_{OH}$ = 2.0 volts, $V_{OL}$ = 0.8 volts, $V_{switch}$ = 1.4 volts. Note that for Best-case fabrication and Vdd=5.5 volts, output currents may exceed the design rule for metal migration in the Pad16a family's output structure with a 50Ω load. These pads should be unconditionally safe for a 70Ω load.

## Pad16a: HPCMOS40 Pad Library for Pad-limited Designs

| Name | Function |
|---|---|
| **Pad16a_200cBiDirTL** | Bi-directional I/O pad with TTL-level input and full latch |
| **Pad16a_240cClkGND** | Clock generator module with GND pad |
| **Pad16a_240cClkVdd** | Clock generator module with Vdd pad |
| **Pad16a_200cIn1TB** | TTL-level input pad, true/complement outputs |
| **Pad16a_200cIn1TL** | TTL-level latched input pad, true/complement SP1 outputs |
| **Pad16a_200cIn1TLL** | TTL-level double-latched input pad, T/C SP2 outputs |
| → **Pad16a_200cOut1** | Non-inverting output pad with clock ring |

**Pads containing no active circuitry:**

| | |
|---|---|
| **Pad16a_200cBlank** | Blank pad (pad metal unconnected), with clock ring |
| **Pad16a_200cGNDI** | GND Pad for internal circuitry, with clock ring |
| **Pad16a_200cGNDR** | GND Pad for pad-ring circuitry, with clock ring |
| **Pad16a_200cVddI** | Vdd Pad for internal circuitry, with clock ring |
| **Pad16a_200cVddR** | Vdd Pad for pad-ring circuitry, with clock ring |

## Pad16b: HPCMOS40 Pad Library for Area-limited Designs

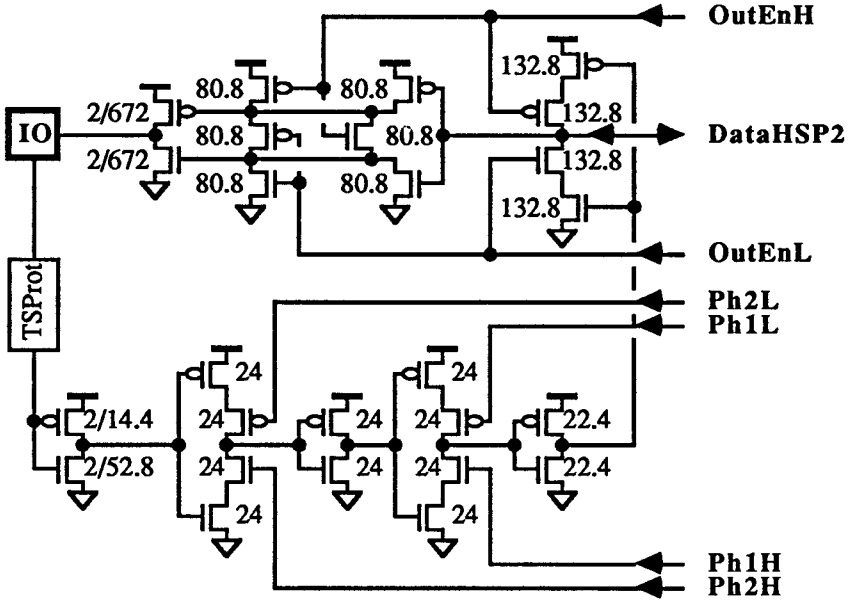
| Name | Function |
|---|---|
| **Pad16b_200cClkIn** | TTL-level input pad with equal-delay true/complement outputs |
| **Pad16b_200cIn1TL** | TTL-level latched input pad, true/complement outputs |
| **Pad16b_200cIn2TL** | TTL-level latched input pad, true/complement outputs |
| **Pad16b_200cOut1** | Non-inverting output pad with clock ring |
| **Pad16b_200cTSTL** | Tri-state I/O pad with TTL-level input latch |

**Pads containing no active circuitry:**

| | |
|---|---|
| **Pad16b_200cGNDI** | GND Pad for internal circuitry, with clock ring |
| **Pad16b_200cGNDR** | GND Pad for pad-ring circuitry, with clock ring |
| **Pad16b_200cVddI** | Vdd Pad for internal circuitry, with clock ring |
| **Pad16b_200cVddR** | Vdd Pad for pad-ring circuitry, with clock ring |

# Pad16a_200cBiDirTL

## Bi-Directional I/O Pad with Full TTL-compatible Input Latch




**Description.** A bi-directional pad with a TTL-level input latch. When **OutEn** is asserted the pad drives signals off-chip on **IO** from the **DataHSP2** internal node. When **OutEn** is not asserted, data on **IO** is latched and driven onto **DataHSP2**. The output circuitry in this pad can drive a 50Ω load at TTL-compatible levels at speeds up to 20MHz (NRZ). The input buffer is suitable for loads up to about 5pf on **DataHSP2** at 25MHz (NRZ) rates.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VOL | LO output voltage | NoiseN, Rload=50Ω (1) | 0.26 | 0.34 | 0.33 | volts |
| VOH | HI output voltage | NoiseP, Rload=50Ω (1) | 2.24 | 3.00 | 5.50 | volts |
| VIL | LO input threshold voltage | NoiseP | 1.16 | 1.39 | | volts |
| VIH | HI input threshold voltage | NoiseN | | 1.39 | 1.60 | volts |
| Idd | Ave. static supply current | No DC load | | 0.12 | 0.15 | ma |
| Cpd | Equiv. power diss. cap. | | | 7.7 | 7.7 | pf |
| CIO | Capacitance on pad IO | | | 5.1 | | pf |
| CData | Capacitance on input DataHSP2 | | | 1076 | | ff |
| CDOEH | Capacitance on inputs OutEnH | | | 944 | | ff |
| CDOEL | Capacitance on inputs OutEnL | | | 898 | | ff |
| CPh1H | Capacitance on inputs Ph1H | | | 168 | | ff |
| CPh1L | Capacitance on inputs Ph1L | | | 343 | | ff |
| CPh2H | Capacitance on inputs Ph2H | | | 193 | | ff |
| CPh2L | Capacitance on inputs Ph2L | | | 236 | | ff |

## Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tpoLL | Data-LO to IO-LO delay | load=50Ω\|\|50pf | 3.6 | | 4.4 | nsec |
| tpoHH | Data HI to IO-HI delay | load=50Ω\|\|50pf | 2.6 | | 4.4 | nsec |
| tro | IO risetime | load=50Ω\|\|50pf | 1.5 | | 4.5 | nsec |
| tfo | IO falltime | load=50Ω\|\|50pf | 1.3 | | 3.2 | nsec |
| tpoLL | DataOut-LO to IO-LO delay | Cap. load, Cload = 50pf | 4.7 | | 7.5 | nsec |
| tpoHH | DataOut HI to IO-HI delay | Cap. load, Cload = 50pf | 5.3 | | 8.4 | nsec |
| tro | IO risetime | Cap. load, Cload = 50pf | 7.4 | | 11.2 | nsec |
| tfo | IO falltime | Cap. load, Cload = 50pf | 3.3 | | 5.4 | nsec |
| tpena | Output Enable time | load=50Ω\|\|50pf | 2.4 | | 4.4 | nsec |
| tpdis | Output Disable time | load=50Ω\|\|50pf | 3.2 | | 4.5 | nsec |
| tpP | Ph1 to DataHSP2 | Cload = 4pf | 5.1 | | 8.3 | nsec |
| tri | DataHSP2 risetime | Cload = 4pf | 5.5 | | 8.8 | nsec |
| tfi | DataHSP2 falltime | Cload = 4pf | 2.3 | | 3.5 | nsec |

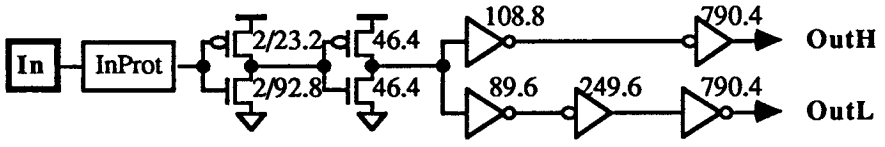
## Operating Requirements

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| ts | Setup time w.r.t. Ph2 trail | Worst | | | 5 | nsec |
| th | Hold time w.r.t. Ph2 trail | Best | 0 | | | nsec |

## Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tpoLL | = | 4.4 + 0.065 Cload (pf) | nsec | W, capacitive load only |
| tpoHH | = | 3.2 + 0.105 Cload (pf) | nsec | W, capacitive load only |
| tro | = | 1.4 + 0.215 Cload (pf) | nsec | W, capacitive load only |
| tfo | = | 1.9 + 0.072 Cload (pf) | nsec | W, capacitive load only |
| tpP | = | 5.4 + 0.73 Cload (pf) | nsec | W |
| tri | = | 2.4 + 1.63 Cload (pf) | nsec | W |
| tfi | = | 1.8 + 0.43 Cload (pf) | nsec | W |

# Pad16a_200cIn1TB
## TTL Input Buffer




**Description.** A 200µ-wide buffered input pad for TTL input signals. This cell produces true and complement outputs with approximately equal delay and can drive internal loads up to about 25 pf at 20 MHz (RZ). It is intended as the input stage for a clock generator, but can be used wherever high-fanout, true/complement, unlatched signals are needed.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VIL | LO input threshold voltage | NoiseP | | | 1.16 | volts |
| VIH | HI input threshold voltage | NoiseN | 1.60 | | | volts |
| VSW | Input switching threshold | Nominal | | 1.39 | | |
| Idd | Ave. static supply current | | | 0.21 | 0.26 | ma |
| Cpd | Equiv. power diss. cap. | | | 13.42 | 13.87 | pf |
| CIn | Capacitance on input In | | | 5.6 | | pf |

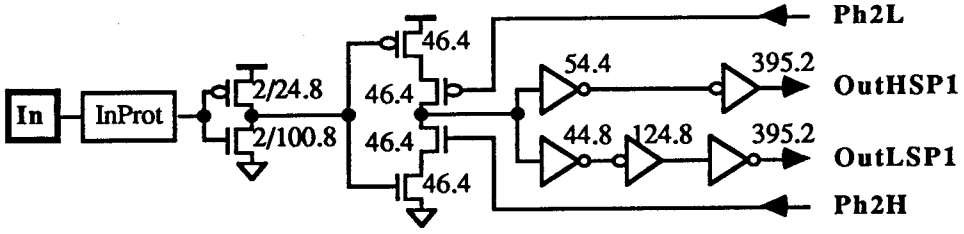
### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tp | In to Out delay | Cload = 10pf, both outputs | 4.2 | | 6.6 | nsec |
| tsk | Skew OutH to OutL | Cload = 10pf, both outputs | 0.05 | | 0.07 | nsec |
| tr | Typical Out<H,L> risetime | Cload = 10pf | 1.3 | | 2.1 | nsec |
| tf | Typical Out<H,L> falltime | Cload = 10pf | 1.1 | | 1.7 | nsec |

### Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tp | = | 6.0 + 0.061 Cload (pf) | nsec | W |
| tr | = | 0.8 + 0.13 Cload (pf) | nsec | W |
| tf | = | 1.2 + 0.049 Cload (pf) | nsec | W |

# Pad16a_200cIn1TL
## Latched TTL Input Buffer




**Description.** A 200μ-wide latched input pad, buffered for TTL input signals. The latch is transparent during **Ph2**, and thus is trailing edge triggered by **Ph2**. This cell produces true and complement outputs of type **SP1** with approximately equal delay and can drive internal loads up to about 10 pf at 50 MHz clock rates.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VIL | LO input threshold voltage | NoiseP for Max | | 1.39 | 1.16 | volts |
| VIH | HI input threshold voltage | NoiseN for Min | 1.60 | 1.39 | | volts |
| Idd | Ave. static supply current | | | 0.22 | 0.27 | ma |
| Cpd | Equiv. power diss. cap. | | | 9.8 | 10.4 | pf |
| CIn | Capacitance on input In | | | 4.97 | | pf |
| CPh2H | Capacitance on input Ph2H | | | 238 | | ff |
| CPh2L | Capacitance on input Ph2L | | | 293 | | ff |

### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tp | Worst delay In to Out<H,L> | Cload = 4pf | 4.3 | | 6.2 | nsec |
| tsk | Worst skew between <H,L> | Cload = 4pf | 0.05 | | 0.01 | nsec |
| tpP | Worst delay Ph2 to Out | Cload = 4pf | 3.0 | | 4.9 | nsec |
| tr | Out<H,L> risetime | Cload = 4pf | 1.2 | | 1.9 | nsec |
| tf | Out<H,L> falltime | Cload = 4pf | 1.1 | | 1.6 | nsec |

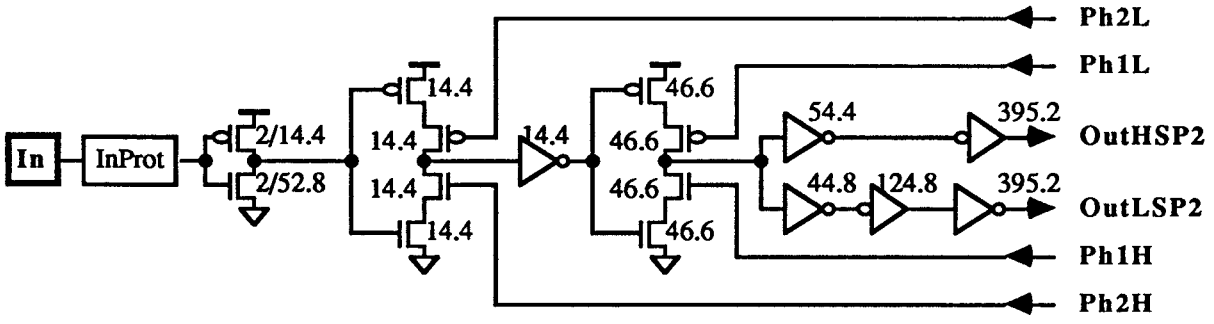
### Operating Requirements

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| ts | Setup time w.r.t. Ph2 trail | Worst, Cload = 10pf (1) | | | 7 | nsec |
| th | Hold time w.r.t. Ph2 trail | Best, Cload = 0pf (1) | 0 | | | nsec |

### Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tp | = | 5.7 + 0.12 Cload (pf) | nsec | W |
| tpP | = | 4.5+ 0.12 Cload (pf) | nsec | W |
| tr | = | 0.8 + 0.27 Cload (pf) | nsec | W |
| tf | = | 1.2 + 0.10 Cload (pf) | nsec | W |

# Pad16a_200cIn1TLL
## Fully-Latched TTL Input Buffer




**Description.** A 200µ-wide fully-latched input pad, buffered for TTL input signals. Input signals are latched on the trailing edge of **Ph2**, and driven out during **Ph1** to provide true/complement **SP2** outputs with approximately equal delays from the clock. This cell produces true and complement outputs of type **SP1** with approximately equal delay and can drive internal loads up to about 10 pf at 40 MHz clock rates. This cell has exactly the same output drive as **Pad2_200cIn1TL**.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VIL | LO input threshold voltage | NoiseP for Max | | 1.38 | 1.15 | volts |
| VIH | HI input threshold voltage | NoiseN for Min | 1.61 | 1.38 | | volts |
| Idd | Ave. static supply current | | | 0.13 | 0.16 | ma |
| Cpd | Equiv. power diss. cap. | | | 12.9 | 13.3 | pf |
| CIn | Capacitance on input In | | | 5.1 | | pf |
| CPh1H | Capacitance on input Ph1H | | | 225 | | ff |
| CPh1L | Capacitance on input Ph1L | | | 300 | | ff |
| CPh2H | Capacitance on input Ph2H | | | 162 | | ff |
| CPh2L | Capacitance on input Ph2L | | | 221 | | ff |

### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tsk | Worst skew between <H,L> | Cload = 4pf | 0.05 | | 0.01 | nsec |
| tpP | Worst delay Ph1 to Out | Cload = 4pf | 3.0 | | 4.9 | nsec |
| tr | Out<H,L> risetime | Cload = 4pf | 1.2 | | 1.9 | nsec |
| tf | Out<H,L> falltime | Cload = 4pf | 1.1 | | 1.6 | nsec |

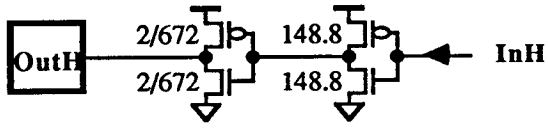
### Operating Requirements

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| ts | Setup time w.r.t. Ph2 trail | Worst | | | 5 | nsec |
| th | Hold time w.r.t. Ph2 trail | Best | 0 | | | nsec |

## Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tpP | = | 4.5 + 0.12 Cload (pf) | nsec | W |
| tr | = | 0.8 + 0.27 Cload (pf) | nsec | W |
| tf | = | 1.3 + 0.08 Cload (pf) | nsec | W |

# Pad16a_200cOut1
## Output Pad




**Description.** A 200µ-wide output (only) pad that can drive a 50Ω load at TTL-compatible levels at speeds up to 20MHz (NRZ).

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VOL | LO output voltage | Rload=50Ω | 0.20 | 0.24 | 0.33 | volts |
| VOH | HI output voltage | Rload=50Ω | 2.24 | 3.02 | 5.50 | volts |
| Idd | Ave. static supply current | No DC load | | 13 | 14 | µa |
| Cpd | Equiv. power diss. cap. | | | 15.9 | 15.9 | pf |
| CIn | Capacitance on input DataOutH | | | 858 | | ff |

### Switching Characteristics

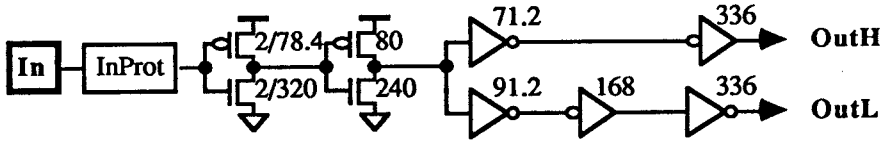
| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tpLL | In to Out delay, Lo-to-Lo | Rload=50Ω\|\|50pf, TTL levels | 1.7 | | 3.1 | nsec |
| tpHH | In to Out delay, Hi-to-Hi | Rload=50Ω\|\|50pf, TTL levels | 1.4 | | 3.5 | nsec |
| tr | OutH risetime | Rload=50Ω\|\|50pf, TTL levels | 0.9 | | 4.2 | nsec |
| tf | OutH falltime | Rload=50Ω\|\|50pf, TTL levels | 0.6 | | 2.2 | nsec |
| tpLL | In to Out delay, Lo-to-Lo | Cap. load, Cload = 50pf | 1.9 | | 5.1 | nsec |
| tpHH | In to Out delay, Hi-to-Hi | Cap. load, Cload = 50pf | 2.8 | | 7.2 | nsec |
| tr | OutH risetime | Cap. load, Cload = 50pf | 4.7 | | 11.8 | nsec |
| tf | OutH falltime | Cap. load, Cload = 50pf | 1.8 | | 4.2 | nsec |

### Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tpLL | = | 3.0 + 0.041 Cload (pf) | nsec | W, capacitive load only |
| tpHH | = | 2.2 + 0.101 Cload (pf) | nsec | W, capacitive load only |
| tr | = | 1.1 + 0.215 Cload (pf) | nsec | W, capacitive load only |
| tf | = | 1.2 + 0.061 Cload (pf) | nsec | W, capacitive load only |

if input threshold [illegible] (2 [illegible]
after ph2 crosses threshold

# Pad16b_400cClkIn
## TTL Input Buffer




**Description.** A 400µ-wide buffered input pad for TTL input signals. This cell produces true and complement outputs with approximately equal delay and can drive internal loads up to about 25 pf at 20 MHz (RZ). It is intended as the input stage for a clock generator, but can be used wherever high-fanout, true/complement, unlatched signals are needed.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VIL | LO input threshold voltage | NoiseP | | | 1.16 | volts |
| VIH | HI input threshold voltage | NoiseN | 1.61 | | | volts |
| VSW | Input switching threshold | Nominal | | 1.39 | | |
| Idd | Ave. static supply current | | | 0.68 | 0.84 | ma |
| Cpd | Equiv. power diss. cap. | | | 6.74 | 7.02 | pf |
| CIn | Capacitance on input In | | | 6.9 | | pf |

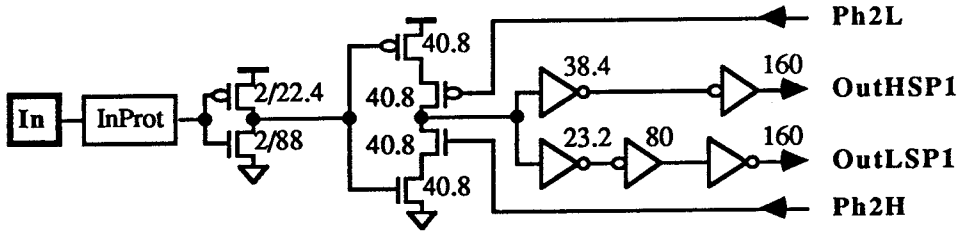
### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tp | In to Out delay | Cload = 10pf, both outputs | 4.1 | | 5.4 | nsec |
| tsk | Skew OutH to OutL | Cload = 10pf, both outputs | 0.4 | | 0.7 | nsec |
| tr | Typical Out<H,L> risetime | Cload = 10pf | 2.4 | | 3.8 | nsec |
| tf | Typical Out<H,L> falltime | Cload = 10pf | 1.2 | | 1.9 | nsec |

### Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tp | = | 3.9 + 0.15 Cload (pf) | nsec | W |
| tr | = | 0.6 + 0.33 Cload (pf) | nsec | W |
| tf | = | 0.9 + 0.10 Cload (pf) | nsec | W |

# Pad16b 400cIn1TL
## Latched TTL Input Buffer




**Description.** A 400µ-wide latched input pad, buffered for TTL input signals. The latch is transparent during **Ph2**, and thus is trailing edge triggered by **Ph2**. This cell produces true and complement outputs of type **SP1** with approximately equal delay and can drive internal loads up to about 5 pf at 50 MHz clock rates.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VIL | LO input threshold voltage | NoiseP for Max | | 1.38 | 1.15 | volts |
| VIH | HI input threshold voltage | NoiseN for Min | 1.60 | 1.38 | | volts |
| Idd | Ave. static supply current | | | 0.19 | 0.23 | ma |
| Cpd | Equiv. power diss. cap. | | | 5.73 | 5.81 | pf |
| CIn | Capacitance on input In | | | 4.84 | | pf |
| CPh2H | Capacitance on input Ph2H | | | 280 | | ff |
| CPh2L | Capacitance on input Ph2L | | | 253 | | ff |

### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tp | Worst delay In to Out<H,L> | Cload = 2pf | 3.8 | | 5.9 | nsec |
| tsk | Worst skew between <H,L> | Cload = 2pf | 0.5 | | 0.87 | nsec |
| tpP | Worst delay Ph2 to Out | Cload = 2pf | 2.6 | | 4.3 | nsec |
| tr | Out<H,L> risetime | Cload = 2pf | 1.2 | | 2.0 | nsec |
| tf | Out<H,L> falltime | Cload = 2pf | 0.9 | | 1.3 | nsec |

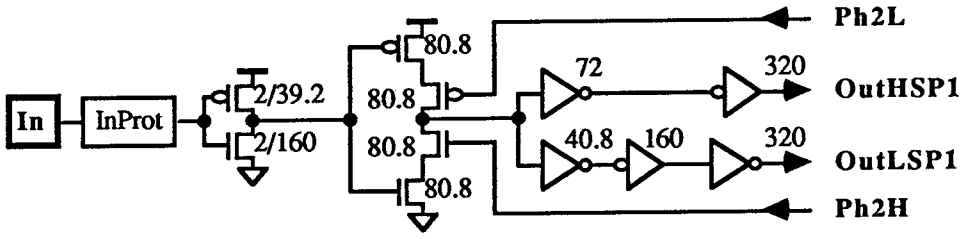
### Operating Requirements

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| ts | Setup time w.r.t. Ph2 trail | Worst, Cload = 5pf (1) | 6 | | | nsec |
| th | Hold time w.r.t. Ph2 trail | Best, Cload = 0pf (1) | 0 | | | nsec |

### Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tp | = | 5.2 + 0.32 Cload (pf) | nsec | W |
| tpP | = | 3.8 + 0.21 Cload (pf) | nsec | W |
| tr | = | 0.7 + 0.65 Cload (pf) | nsec | W |
| tf | = | 0.9 + 0.20 Cload (pf) | nsec | W |

# Pad16b_400cIn2TL
## Latched TTL Input Buffer




**Description.** A 400µ-wide latched input pad, buffered for TTL input signals. The latch is transparent during **Ph2**, and thus is trailing edge triggered by **Ph2**. This cell produces true and complement outputs of type **SP1** with approximately equal delay and can drive internal loads up to about 10 pf at 50 MHz clock rates.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VIL | LO input threshold voltage | NoiseP for Max | | 1.38 | 1.16 | volts |
| VIH | HI input threshold voltage | NoiseN for Min | 1.60 | 1.38 | | volts |
| Idd | Ave. static supply current | | | 0.35 | 0.43 | ma |
| Cpd | Equiv. power diss. cap. | | | 8.16 | 8.65 | pf |
| CIn | Capacitance on input In | | | 5.11 | | pf |
| CPh2H | Capacitance on input Ph2H | | | 382 | | ff |
| CPh2L | Capacitance on input Ph2L | | | 343 | | ff |

### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tp | Worst delay In to Out<H,L> | Cload = 4pf | 3.7 | | 5.7 | nsec |
| tsk | Worst skew between <H,L> | Cload = 4pf | 0.5 | | 0.83 | nsec |
| tpP | Worst delay Ph2 to Out | Cload = 4pf | 2.5 | | 4.0 | nsec |
| tr | Out<H,L> risetime | Cload = 4pf | 1.2 | | 1.9 | nsec |
| tf | Out<H,L> falltime | Cload = 4pf | 0.8 | | 1.3 | nsec |

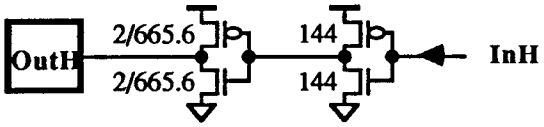
### Operating Requirements

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| ts | Setup time w.r.t. Ph2 trail | Worst, Cload = 10pf (1) | 6.5 | | | nsec |
| th | Hold time w.r.t. Ph2 trail | Best, Cload = 0pf (1) | 0 | | | nsec |

### Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tp | = | 5.0 + 0.16 Cload (pf) | nsec | W |
| tpP | = | 3.6 + 0.10 Cload (pf) | nsec | W |
| tr | = | 0.6 + 0.34 Cload (pf) | nsec | W |
| tf | = | 0.8 + 0.11 Cload (pf) | nsec | W |

# Pad16b_400cOut1
## Output Pad




**Description.** A 400µ-wide output (only) pad that can drive a 50Ω load at TTL-compatible levels at speeds up to 25MHz (NRZ).

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VOL | LO output voltage | Rload=50Ω | 0.20 | 0.24 | 0.32 | volts |
| VOH | HI output voltage | Rload=50Ω | 2.28 | 3.06 | 5.50 | volts |
| Idd | Ave. static supply current | No DC load | | 13 | 15 | µa |
| Cpd | Equiv. power diss. cap. | | | 16.1 | 16.3 | pf |
| CIn | Capacitance on input DataOutH | | | 841 | | ff |

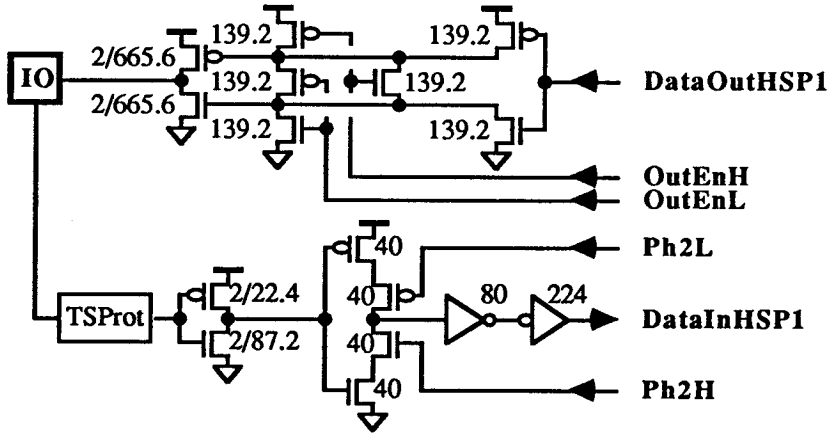
### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tpLL | In to Out delay, Lo-to-Lo | Rload=50Ω\|\|50pf, TTL levels | 1.7 | | 3.2 | nsec |
| tpHH | In to Out delay, Hi-to-Hi | Rload=50Ω\|\|50pf, TTL levels | 1.4 | | 3.5 | nsec |
| tr | OutH risetime | Rload=50Ω\|\|50pf, TTL levels | 0.9 | | 3.9 | nsec |
| tf | OutH falltime | Rload=50Ω\|\|50pf, TTL levels | 0.6 | | 2.1 | nsec |
| tpLL | In to Out delay, Lo-to-Lo | Cap. load, Cload = 50pf | 1.9 | | 5.1 | nsec |
| tpHH | In to Out delay, Hi-to-Hi | Cap. load, Cload = 50pf | 2.7 | | 7.1 | nsec |
| tr | OutH risetime | Cap. load, Cload = 50pf | 4.6 | | 11.3 | nsec |
| tf | OutH falltime | Cap. load, Cload = 50pf | 1.8 | | 4.1 | nsec |

### Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tpLL | = | 3.0 + 0.040 Cload (pf) | nsec | W, capacitive load only |
| tpHH | = | 2.2 + 0.097 Cload (pf) | nsec | W, capacitive load only |
| tr | = | 1.0 + 0.207 Cload (pf) | nsec | W, capacitive load only |
| tf | = | 1.1 + 0.060 Cload (pf) | nsec | W, capacitive load only |

# Pad16b_400cTSTL
## Tri-State I/O Pad with TTL Input Buffer




**Description.** A 400µ-wide bi-directional pad with an input latch, buffered for TTL levels. The output circuitry in this pad can drive a 50Ω load at TTL-compatible levels at speeds up to 20MHz (NRZ). The input buffer is suitable for loads up to about 5pf at 40MHz clock rates.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VOL | LO output voltage | NoiseN, Rload=50Ω (1) | 0.20 | 0.26 | 0.33 | volts |
| VOH | HI output voltage | NoiseP, Rload=50Ω (1) | 2.27 | 3.05 | 3.83 | volts |
| VIL | LO input threshold voltage | NoiseP | | 1.39 | 1.15 | volts |
| VIH | HI input threshold voltage | NoiseN | 1.60 | 1.39 | | volts |
| Idd | Ave. static supply current | No DC load | | 0.19 | 0.42 | ma |
| Cpd | Equiv. power diss. cap. | | | 4.86 | 4.95 | pf |
| CIO | Capacitance on pad IO | | | 3.6 | | pf |
| CDOut | Capacitance on input DataOutH | | | 869 | | ff |
| CDOE | Capacitance on inputs OutEn<H,L> | | | 860 | | ff |
| CPh2H | Capacitance on input Ph2H | | | 350 | | ff |
| CPh2L | Capacitance on input Ph2L | | | 294 | | ff |

### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tpo | Worst DataOut to IO delay | load=50Ω\|\|50pf | 2.1 | | 4.1 | nsec |
| tro | IO risetime | load=50Ω\|\|50pf | 0.9 | | 4.1 | nsec |
| tfo | IO falltime | load=50Ω\|\|50pf (1,2) | 0.6 | | 2.5 | nsec |
| tpo | Worst DataOut to IO delay | Cap. load, Cload = 50pf | 3.0 | | 7.8 | nsec |
| tro | IO risetime | Cap. load, Cload = 50pf | 4.3 | | 11.5 | nsec |
| tfo | IO falltime | Cap. load, Cload = 50pf | 1.7 | | 4.6 | nsec |
| tpena | Output Enable time | load=50Ω\|\|50pf | 1.8 | | 2.6 | nsec |
| tpdis | Output Disable time | load=50Ω\|\|50pf | 1.0 | | 3.3 | nsec |
| tpi | Worst-case IO to DataIn | Cload = 2pf | 2.2 | | 4.9 | nsec |
| tpPi | Worst-case Ph2 to DataIn | Cload = 2pf | 1.2 | | 3.9 | nsec |
| tri | DataIn risetime | Cload = 2pf | 0.6 | | 1.6 | nsec |
| tfi | DataIn falltime | Cload = 2pf | 0.4 | | 0.9 | nsec |

## Operating Requirements

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| ts | Setup time w.r.t. Ph2 trail | Worst, Cload = 5pf | 6 | | | nsec |
| th | Hold time w.r.t. Ph2 trail | Best, Cload = 0pf | 0 | | | nsec |

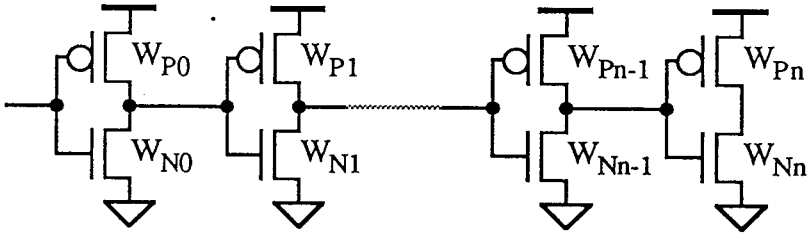
## Loading Characteristics

| Parameter | Equation | Units | Conditions |
|---|---|---|---|
| tpoLL = | 3.8 + 0.048 Cload (pf) | nsec | W, capacitive load only |
| tpoHH = | 2.7 + 0.099 Cload (pf) | nsec | W, capacitive load only |
| tro = | 1.2 + 0.205 Cload (pf) | nsec | W, capacitive load only |
| tfo = | 1.5 + 0.061 Cload (pf) | nsec | W, capacitive load only |
| tpi = | 4.4 + 0.24 Cload (pf) | nsec | W |
| tpPi = | 3.4 + 0.25 Cload (pf) | nsec | W |
| tri = | 0.74 + 0.45 Cload (pf) | nsec | W |
| tfi = | 0.77 + 0.12 Cload (pf) | nsec | W |

# Appendix A — The τ Model

## A.1 Optimum Delay in a Fanout Chain

Following Shoji's "CMOS Digital Circuit Technology", Section 8.8, consider a long chain of inverters needed to amplify a "weak" signal sufficiently to drive a large capacitive load.




Assuming all transistors are the same (minimum) channel length, the times for stage i to pull its load up and down are:

$$t_u = \tau_P \frac{W_{Pi+1} + W_{Ni+1} + Q\,(W_{Pi} + W_{Ni})}{W_{Pi}}$$

$$t_d = \tau_N \frac{W_{Pi+1} + W_{Ni+1} + Q\,(W_{Pi} + W_{Ni})}{W_{Ni}}$$

Eqns A.1

The parameter $\tau_P$ is a time constant given by the product of the gate capacitance of a unit-sized device and the equivalent resistance of a unit-sized pfet, and similarly for $\tau_N$. The parameter Q is the ratio of the parasitic capacitance of the output drains to the gate capacitance of a typical stage. Assume that the pullup and pulldown times are the same, then these equations imply that

$$\frac{\tau_P}{W_{Pi}} = \frac{\tau_N}{W_{Ni}} = \frac{\tau}{S_i} \quad \text{and} \quad \frac{W_{Pi}}{W_{Ni}} = \frac{\tau_P}{\tau_N} = \beta_e$$

where $\tau = \tau_P + \tau_N$, the *size* of an inverter $S_i = W_{Pi} + W_{Ni}$, and $\beta_e$ is the ratio of pfet width to nfet width that gives equal pullup and pulldown delays. From this we have the delay at each stage of the fanout chain is

$$t_i = \tau \left(\frac{S_{i+1}}{S_i} + Q\right) = \tau\,(f_i + Q)$$

where $f_i$ is the fanout of stage i. The total delay through the chain is

$$t_D = \tau \sum_{i=0}^{n-1} f_i + \tau\, n\, Q$$

A standard arithmetic lemma states that the arithmetic average of n positive numbers is always less than the geometric average, so

$$\frac{1}{n}\sum_{i=0}^{n-1} f_i \geq \left(\prod_{i=0}^{n-1} f_i\right)^{\frac{1}{n}} = \left(\frac{S_n}{S_0}\right)^{\frac{1}{n}}$$

The equality holds only when the $f_i$ are all equal, so the optimum delay is attained when the fanout at each stage is the same and

$$t_{d(opt)} = \tau\, n\, (Q + f_{i(opt)}) \quad \text{where} \quad f_{i(opt)} = \left(\frac{S_n}{S_0}\right)^{\frac{1}{n}}$$

The parameter n, the number of stages, is still free. The *minimum* delay is found by diffentiating $t_{d(opt)}$ with respect to n, setting the result equal to zero, then solving for n. With Q=0, this gives

$$\frac{1}{\tau}\frac{\partial t_{d(opt)}}{\partial n} = Q + \left(\frac{S_n}{S_0}\right) - \frac{1}{n}\ln\left(\frac{S_n}{S_0}\right) \cdot \left(\frac{S_n}{S_0}\right)^{\frac{1}{n}} = 0 \Big|_{Q=0} \Rightarrow n = \ln\left(\frac{S_n}{S_0}\right)$$

Eqn A.2

The number of stages, of course, must be an integer; the closest integral value gives the smallest practical delay for a given $S_n$ and $S_0$. The absolute value of the minimum delay (Q=0) is, then

$$t_{d(min)} = \tau \ln\left(\frac{S_n}{S_0}\right) \cdot \left(\frac{S_n}{S_0}\right)^{1/\ln\left(\frac{S_n}{S_0}\right)} = \tau\, e \ln\left(\frac{S_n}{S_0}\right)$$

Eqn A.3

So, the optimal fanout at each stage is e = 2.718, and the optimal number of stages is the logarithm of the ratio of the load capacitance to the source capacitance. With Q=0, we can write the normalized delay for a given fanout as

$$\frac{t_d}{t_{d(min)}} = \frac{\tau\, n\, f}{\tau\, e \ln\left(\frac{S_n}{S_0}\right)}$$

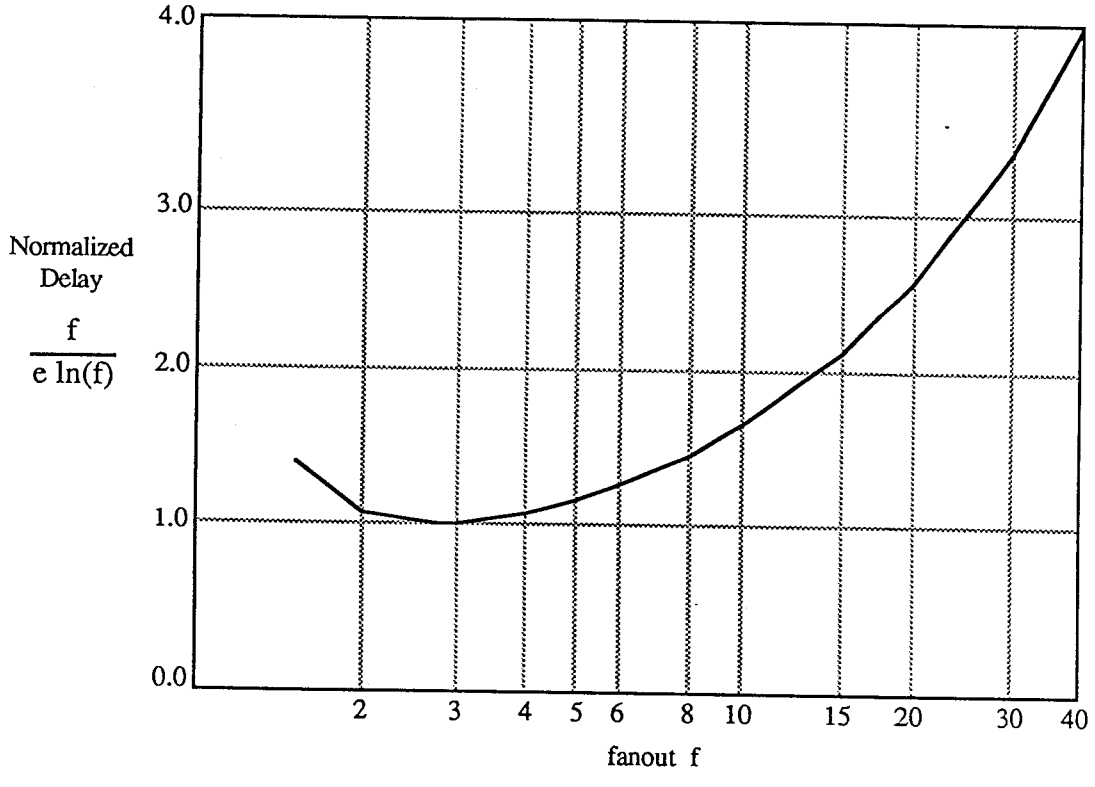
But

$$\ln(f) = \frac{1}{n}\ln\left(\frac{S_n}{S_0}\right)$$

So the normalized delay is

$$\frac{f}{e \ln(f)}$$

It is useful to plot the normalized delay as a function of f, as shown below. The function is quite flat between fanouts of 2 and 4, and there is only a small penalt for relatively large fanouts (10 or so).




For $Q \neq 0$, the optimal number of stages is smaller. For small Q

$$n = \frac{1}{1 + Q/e} \cdot \ln\left(\frac{S_n}{S_0}\right)$$

Eqn A.4

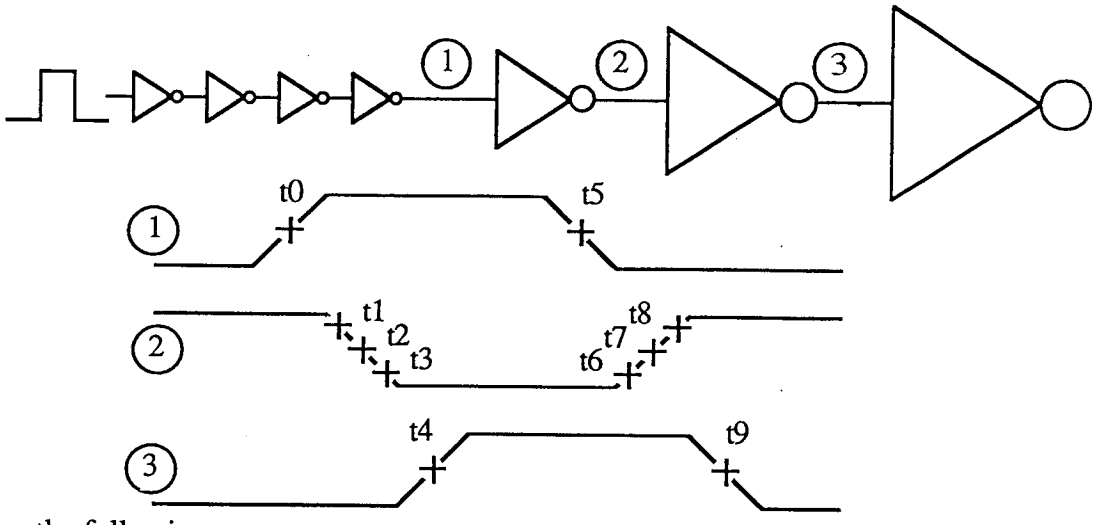
## A.2 Determining τ from Simulation

Consider an inverter chain with equal-width nfet and pfets, then from A.1 (with Q=0), the delay through an adjacent pair of inverters in the chain is

$$t_{p2} = t_u + t_d = \tau_P \frac{f \cdot 2W}{W} + \tau_N \frac{f \cdot 2W}{W} = 2\, f\, \tau$$

Therefore, by simulating the inverter pair delay $t_{p2}$ as a function of fanout f, the fundamental unit of delay, τ, can be determined from the slope.

We use the following circuit to determin τ. The times t0, t2, t5, t7, and t9 refer to crossings at the switching threshold, defined as Vdd/2, where t1, t3, t6, and t8 refer to crossings at 0.1*Vdd and 0.9*Vdd.




We define the following parameters:

| | | |
|---|---|---|
| tr | rise time | t8 - t6 |
| tf | fall time | t3 - t1 |
| trf | rise time + fall time | t8 - t6 + t3 - t1 |
| tphl | prop delay, 1inv, HI-to-LO | [(t2 - t0)+(t9 - t7)]/2 |
| tplh | prop delay, 1inv, LO-to-HI | [(t4 - t2)+(t7 - t5)]/2 |
| tp2 | prop delay, 2 inv's | [(t4 - t0)+(t9-t5)]/2 |

| Parameter | Method for Determination |
|---|---|
| $\tau_N$ | 1/2 of slope of tphl *vs* f |
| $\tau_P$ | 1/2 of slope of tplh *vs* f |
| $\tau$ | 1/2 of slope of tp2 *vs* f |

## A.3 Optimum β Ratio

Suppose that the inverter chain of section A.1 is built with a β-ratio different from $\beta_e$. For this circuit, the pullup and pulldown times for a single stage will be different, but for a *pair* of stages, the delay for rising and falling inputs will be the same (under this simple model). The delay of a pair of stages is

$$t_{p2} = (f + Q)\left[\tau_P + \tau_N + \frac{\tau_P}{\beta} + \beta\,\tau_N\right]$$

The minimum delay is obtained when

$$\beta = \sqrt{\frac{\tau_P}{\tau_N}} = \beta_e^{1/2}$$

Shoji reports that, for typical processes, the delay tp2 as a function of b has a very shallow minimum at the optimal ratio.

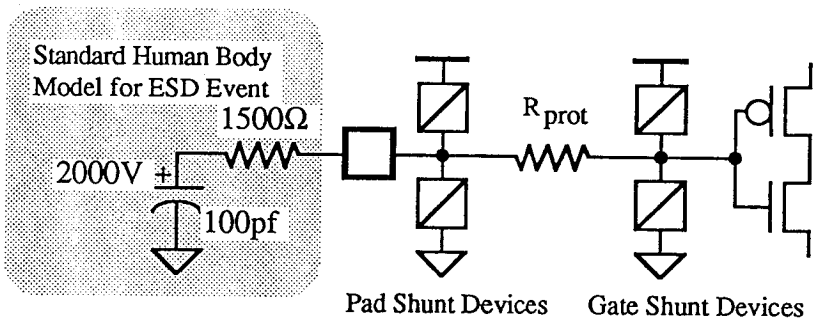
Shoji also remarks: "When a very large fanout ratio is necessary, or when a very complex gate is used, the pfet/nfet ratio should be chosen so that the gate pulls up and pulls down with approximately equal delay. In a long chain of relatively simple gates, the pfet/nfet ratio should be chosen smaller than the ratio for delay symmetry. In this latter case the pullup-pulldown asymmetry of individual gates are compensated by the asymmetries of neighboring gates, and the delay asymmetry does not accumulate."

His treatment of general fet sizing for complex gates indicates that optimization is a complex task, best left to numerical techniques.

# Appendix B — The Chip Input Interface

## B.1 ESD Protection

In this Appendix we will develop simple models for designing input circuitry whose purpose is to protect internal devices from electrical transients. The general form for the input protection circuit, shown with a crude model of the transient it must absorb, is




The purpose of the pad shunt devices is to divert the current from an ESD event to the supply rails, clamping the voltage at the pad to some value low enough that the series $R_{prot}$ and the gate shunts can limit the maximum gate voltage to a safe level. Several kinds of shunt devices are available in MOS technologies:

**Diodes.** CMOS processes offer complementary diodes that can be used to build very effective clamps. The principal design consideration is to make the series resistance of these diodes small. This means drawing the diffusions of the diodes large, covering the entire area with as many metal-1/diffusion contacts as possible, and strapping heavily with metal. A limitation of these devices is that one can *only* clamp to the supply rails; this is not a problem when connecting 5-volt CMOS chips to a 5-volt bipolar logic family, but may present some difficulty when interfacing 3-volt CMOS to a 5-volt bipolar environment.

**Punch-through Devices.** Two diffusions in proximity will exhibit punch-through when the voltage between the regions exceeds some value that depends roughly linearly on the doping density and quadratically with the distance between the structures. In punch-through, the depletion region from the "drain" of the device actually touches the "source". This effect is normally undesirable, but can be used effectively in input structures as a voltage clamp, shunting a positive transient current to the Vss supply rail. For minimum spacing of diffusions, most processes are designed to exhibit punch-through only when the voltage exceeds several times the supply voltage (typicall 10-20 volts).

**Avalanche Devices.** All mosfets exhibit avalanche breakdown above some source-drain voltage. The breakdown voltage is lower for lower gate voltages. The presence of the gate terminal near the diffusion edge of the drain terminal of a transistor produces large electric fields at the substrate surface, lowering the breakdown voltage. Metal gate transistors, with large overlaps of the gate terminal over the drain, exhibit very sharp avalanching behavior. Thus a "field" device, usually considered an undesirable parasitic device, can be used as a fast and effective clamp. Again, the turn-on voltage is typically in the range of 10-20 volts for most small-geometry MOS processes. The main considerations for laying out both avalanche devices and punch-through devices is to provide sufficient area to dissipate the shunted power safely, while keeping the

channel length as small as possible.

**Resistors.** Any of the conducting layers can be used as resistors to limit the current flow through the shunt devices, though usually only polysilicon and various flavors of diffusions and wells are considered useful. Polysilicon is generally a bad choice, as it is thermally isolated from the substrate (which must ultimately carry away the heat) by a thick oxide. For an n-well CMOS process, n-diffusion is a good choice because it is diffused directly into the substrate, providing good thermal contact, and has smaller resistivity (thus larger area for a given resistance) than p-diffusion. The resistance of n-diffusion is usually fairly closely controlled, so that resistance values can be predicted fairly accurately from the drawn shape. The n-well for such a process may also be a candidate, but it requires connections through shorting contacts (n+ in the n-well) and will have to be guard-ringed with p+. Layout considerations are relatively simple: make the resistor only as large as necessary to dissipate its power safely.

## B.2 Electrical Model for Diodes

As we shall see, the native CMOS diodes have outstanding electrical and thermal properties for use as ESD protection devices. We here develop an electrical (simulation) model for these diodes, partly based on empirical measurements. We will then simulate their electrical behavior using CAzM, then use these results togthether with the above discussion on the finite element analysis to predict the thermal behavior during an ESD transient.

Hodges ("Analysis and Design of Digital Integrated Circuits", Chapter 4) develops a simple model for the junction diode in forward bias. The diode current is

$$I_D = I_S \left(e^{\frac{V_D}{nV_T}} - 1\right)$$

We define the thermal voltage

$$V_T = \frac{kT}{q} = 25.86 \text{ mV}$$

Taking the common log of both sides of the diode equation,

$$\log I_D = \log I_S + V_D \left(\frac{0.4343}{n V_T}\right) \quad ; \quad \frac{V_T}{0.4343} = 59.5 \text{ mV/decade}$$

The factor n is the emission coefficient, $I_S$ is the saturation current. The diode current cannot increase exponentially without bound, of course; eventually the diode current is dominated by the series resistance of the diode $R_S$. We can find all three parameters experimentally simply by measuring the forward bias characteristics of the diode.

Fortunately we have available an input protection network fabricated on MOSIS's 3μ bulk PWell process. The diodes for these protection circuits have dimensions 139μ x 6μ for each polarity. They are densely covered by a single row of metal-1/diffusion contacts and strapped directly to the bond pad. Measurements (see Pxpl Lab Notebook, 4/22/88) were made for forward bias currents ranging from 0.5 μamps to 200 mamps, the latter measurements chiefly needed to determine the diode resistance. Results for these diodes were

| | Emission Coeff (n) | Saturation Current (Is) | Series Resistance (Rs) |
|---|---|---|---|
| N+/P | 1.03 | $3.2\times10^{-15}$ amps | 3.2Ω |
| P+/N | 1.24 | $1.56\times10^{-13}$ amps | 3.0Ω |

Hodges develops a simple mathematical model for the saturation current that predicts it is proportional to the diode area. We will take the series resistance as inversely proportional to the area of the diode. For modeling diodes in Pxpl5 designs, we will take the emission coefficients measured above and

$$I_{S(N+/P)} = 3.2\times10^{-15} \cdot \frac{\text{Area } (\mu^2)}{139\mu \cdot 6\mu}$$

$$I_{S(P+/N)} = 1.56\times10^{-13} \cdot \frac{\text{Area } (\mu^2)}{139\mu \cdot 6\mu}$$

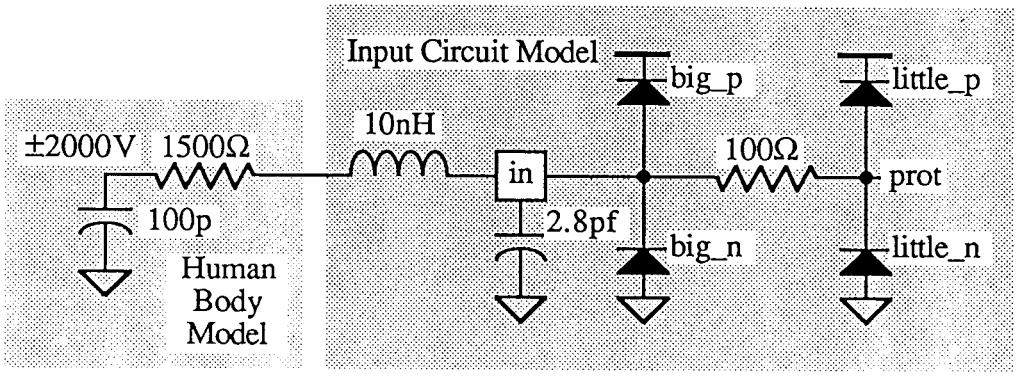
(until fabrication on 2.0μ and 1.6μ nwell processes provides better measurement candidates).

We use the following SPICE/CAzM models for the diodes:

```
.model big_n diode
+    Is=3.237e-15    Rs=5      n=1.03
+    Cjo=550e-15     Vj=0.8
.model big_p diode
+    Is=1.56e-13     Rs=5      n=1.24
+    Cjo=475e-15     Vj=0.8
.model little_n diode
+    Is=0.9315e-13   Rs=17     n=1.03
+    Cjo=158e-15     Vj=0.8
.model little_p diode
+    Is=0.449e-13    Rs=17     n=1.24
+    Cjo=137e-15     Vj=0.8
```

The sidewall and area capacitances were take from the SPICE models for M59A.

Simulation was performed using the following model:



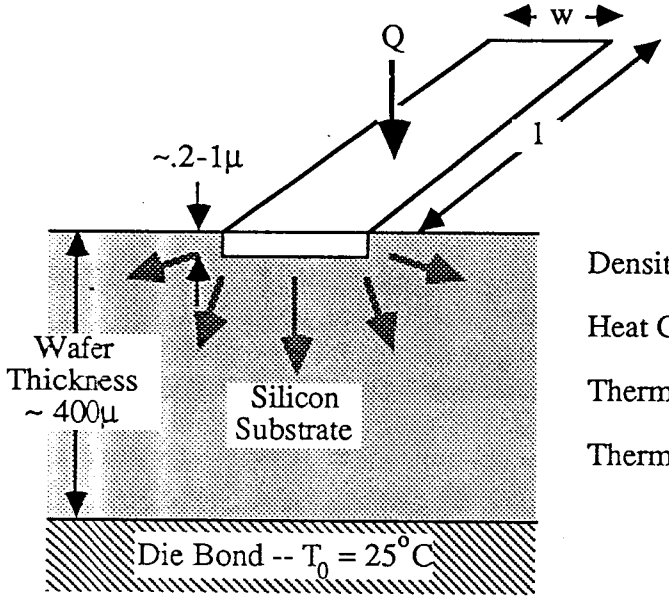

Simulation results shown that the voltage at "in" never exceeds 13.3 volts during a positive-going

(worst-case) ESD event with VHMB = 2000 volts. At node "prot", the voltage never exceeds 4.03 volts. Peak current through "in" is 1.32 amps.

The strategy for determining the "safety" of this circuit is to multiply the current through the protection diode by the voltage across the diode to find the *power* delivered to the diode as a function of time. This power waveform is then used as an input to the finite-element thermal model developed below.

## B.3 Thermal Considerations

How big should the various series and shunt elements of the input protection circuit be to dissipate power from the worst-case transients safely? To answer this question, we construct a simplified thermal model for the devices.




$\text{Density}_{Si} = \rho_{Si} = 2.42\ \text{gm/cm}^3$

$\text{Heat Capacity}_{Si} = s_{Si} = 0.7\ \text{J/gm}^\circ\text{K}$

$\text{Thermal Capacitance}_{Si} = \rho_{Si}\, s_{Si} = 1.69\times10^{-12}\ \text{J/}\mu^3{}^\circ\text{K}$

$\text{Thermal Resistivity}_{Si} = 1/\kappa_{Si} = 7.08\times10^{3}\ \mu^\circ\text{K sec/J}$

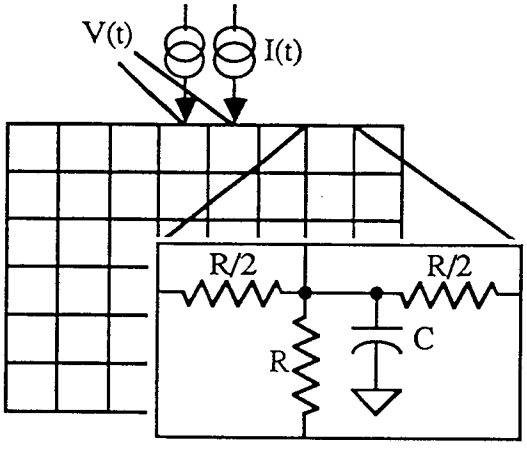
A rectangular heated region is embedded in the surface of a silicon substrate, with thickness very small compared to the thickness of the substrate. The back side of the substrate is held at constant temperature by a low-thermal-resistance die bond to a package. Heat is applied uniformly on the surface of the rectangular region of width w and length l. Heat diffuses away from the heated rectangular region through the substrate. We want to find the maximum temperature reached in the region of the heater during an event.

We first assume l >> w, so that we can treat this as a two-dimensional heat-flow problem. To find an analytical solution, we must solve the homogeneous diffusion equation

$$\nabla^2_{x,y}\, T(x,y,t) = \frac{\partial T}{\partial t}$$

With both inhomogeneous boundary conditions (contant heat input) and homogeneous boundary conditions (contant temperature at the bottom and sides of the substrate). This problem is extremely difficult to solve, and the solution itself will require numerical methods to arrive at the required result. We therefore resort to finite-element analysis to find an approximate result. The

finite element model uses an electrical analog for the heat problem, and we use an electrical circuit simulator (CAzM) to carry out the analysis. The finite element model is




| Thermal | | Electrical | |
|---|---|---|---|
| Quant. | Units | Quant. | Units |
| $\rho_{Si}\, s_{Si}\, L_E^3$ | J/°K | C | Coul/Volt |
| $1/(\kappa_{Si}\, L_E)$ | °Ksec/J | R | Volt sec/Coul |
| T | °K | V | Volts |
| $P\, L_E^2/(l \cdot w)$ | J/sec | I | Coul/sec |

We make the (pessimistic) assumption that no heat flows out of the outermost elements, and we ensure that the number of elements is "large enough" and the size of each element "small enough" by investigating the convergence of the problem by simulation.

### B.3.1 Thermal Properties of Pad Shunt Diodes

For the ESD diodes described above, we have modelled the thermal behavior on a FEA with element size of 1μ. (Covergence was checked by re-simulating with elements sizes of 0.5μ and 0.25μ; 1μ element size appears reasonable, with an array of 20x20 elements.) The power delivered to an element, as shown in the chart above, is the total power delivered to the diode, scaled by the ratio of element area to total diode area,

$$\frac{1\mu^2}{6\mu \cdot 139\mu} = 1.199\times10^{-3}$$

Peak power for the diodes modelled in B.2 is about 17 watts, so the power per element is about 21mW.

Simulation with this finite element analysis indicates a peak temperature rise of 166C. If the die starts at room temperature, the temperature at the diode peaks at 190C. Information from Scott Goodwin-Johansson suggests that the main failure mode for diodes (and other silicon structures) is so-called "spiking", which occurs when the temperature of an aluminum/silicon interface nears 400C. At this point the aluminum and silicon begin to form a solution which will quickly short any nearby junctions. Note that typcial soldering temperatures are about 200C, and chips must withstand repeated exposure to soldering temperatures without signficant damage. Based on these arguments (and having assumed a fairly pessimistic series resistance), it seems clear that the diodes for the test 3μ chip should be adequately protected against HMB events.

### B.3.2 Thermal Behavior of Series Limiting Resistor and Gate Shunt Diodes

For the protection circuit above, the resistor heated by the difference in voltage between nodes "in" and "prot", which difference peaks at about 9.2 volts, so the total peak power is

$$\frac{(9.2 \text{ volts})^2}{100\Omega} = 0.85 \text{ W}$$

If we assume that the power as a function of time is similar to that heating the pad-shunt diode, then for diode areas greater than

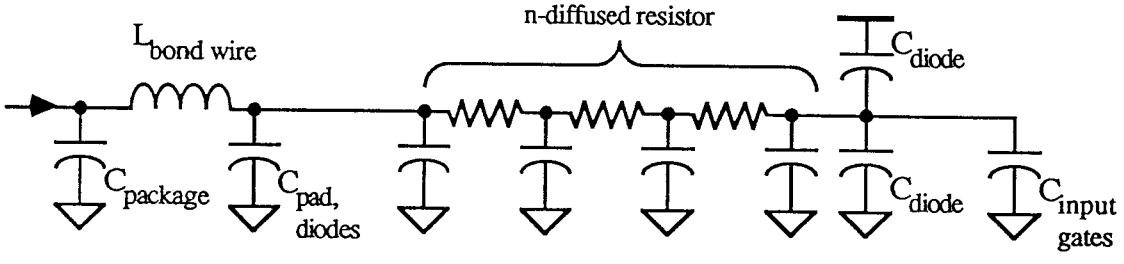
$$\frac{0.85 \text{ W}}{17 \text{ W}} \cdot 139\mu \cdot 6\mu = 42\mu^2$$

the resistor's temperature rise should be less than that of the pad-shunt diode. This means, effectively, that any reasonable layout for the resistor is "safe" thermally. This argument applies to the gate shunt diodes, as well, since they are even less heavily stressed.

## B.4 Impact on Performance

How does the input circuit affect the speed of an IC? During normal operation the shunt elements of the input protection circuitry are never turned on, so no current flows through them from signal sources. However, we have to put up with the capacitance of these structures. The capacitance is associated mainly with the area of the devices, and, as we have seen, the area needs to be large enough to prevent over-heating.

What, then, is the effect of the input network on the speed of the chip? During normal operation, the input clamping structures are turned off, so we may accurately model the diodes or field devices as a capacitance. The input protection resistor is essentially an RC delay line, whose characteristics are those of the layer (n-diffusion) on which the resistor is built, terminated in a capacitance made up of the reverse-biased clamping diodes and the input gates of the first stage of amplification in the input pad. We can approximate the input network as follows:




For small fanin input pads, the most important element in the network is the protection resistor. Our problem is, then, to model the effects of an RC delay line, sourced by a resistive driver (to make things simple) and loaded with a capacitance.

Sakurai has developed a numerical model for delay in interconnect on VLSI chips ("Approximation of Wiring Delay in MOSFET LSI", IEEE JSSC, SC-18, No. 4, pp 418-426) which is very useful. For our purposes, the main result is for a length of interconnect with total capacitance C and total resistance R driven by a step impulse. The time for the output to reach 90% of its final value is

$$t_{0.9}/RC = 1.02 + 2.21(C_T R_T + C_T + R_T)$$

where $R_T$ is the ratio of the driver's source resistance to the wiring resistance $R_S$, and $C_T$ is the ratio of the load capacitance $C_L$ to the wiring capacitance C. For small values of $C_T$ and $R_T$, this

expression is, according to Sakurai, accurate to a few percent in modeling the delay.

We can calculate the product RC for n-diffusion with

$$R = \frac{l}{w}\rho$$

$$C = (l \times w)\, C_a + 2(l+w)\, C_p$$

where $C_a$ and $C_p$ are the area and perimeter capacitances, and $\rho$ is the resistivity in $\Omega$/square. For n-diffusion resistors between a few 10's to a few 1000's of ohms, $l^2 >> l > w$, $C_a \sim C_p$ numerically, when expressed in units of lambda, so, to a good approximation,

$$RC = \frac{l}{w}\rho \times (l \times w)\, C_a = l^2 \rho\, C_a$$

This result may be a bit surprising. Combined with the expression above for delay through the input protection network, we see that for small source resistance and small input-stage capacitance, the delay through the network can be controlled simply by varying the length of the input resistor (which also varies its power handling ability, as shown above), essentially independent of its resistance. The result is that for small fanin input pads (all but the clock input, typically), we can achieve good levels of protection (small input clamp currents) without undue delay penalty.

To get a feeling for the magnitude of the delay due to input protection network, consider a thermally "safe" 100$\Omega$ resistor built from ndiffusion, and layed out with w = 10$\mu$, l = 100/40 • 10 = 25$\mu$. For the MOSIS 2$\mu$ process, for example,

$$\rho = 40\ \Omega\text{/square},\quad C_a = 250\text{af}/\mu^2$$

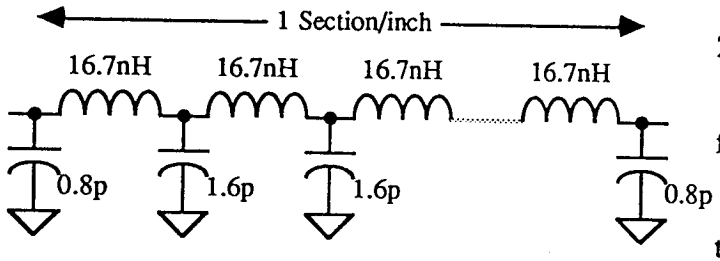
$$RC = (25\mu)^2 \cdot 40 \cdot 250\ \text{af}/\mu^2 = 6.25\ \text{psec}$$

In general, or course, the circuit designer should draw the protection resistor as large as possible while still satisfying the overall delay requirements.

### B.4.1 System-Level Concerns

What is the largest practical total input capacitance for a mosfet input? To answer this, we must consider the system level issue of driving a number of identical inputs from a single driving source (the worst-case problem). The principal effect that is important here is the distributed inductance and capacitance of the printed circuit board traces and the chip input capacitances.

To get a feel for the problem, we model the transmission-line effects in the PCB with a lumped-circuit delay line. National's literature on the "TurboTranceivers" gives typical PCB parasitics as 200nH/ft and 20pf/ft, corresponding to a 100$\Omega$ characteristic impedance. The Motorola "MECL System Design Handbook" (pp 179) indicates that a 0.010" conductor on 0.031" thick PCB will have 100$\Omega$ characteristic impedance with propagation delays of 0.148ns/ft. We use the following model:




$$Z_0 = \sqrt{\frac{L}{C}} = 100\Omega$$

$$f_t = \frac{1}{\sqrt{LC}} = 6\text{GHz}$$

$$t_p = \sqrt{LC} = 0.17\text{ns/inch}$$

Now suppose that we have inputs to custom mosfet chips connected periodically to this structure, with chip-to-chip distance of 1 inch. The characteristic impedance, cutoff frequency, and propagation speed will be reducedd by the additional lumped capacitance. For 74F-series drivers, the lowest possible impedance that can be driven while still maintaining the necessary noise margins is about 30Ω, which corresponds to a chip input capacitance of about 17pf. (This analysis ignores losses in the line, as well as the effects of the chip input's inductance and resistance.) A safer lower limit for lumped PCB line impedance is 50Ω, which limits the total chip input capacitance to about 5 pf, of which about 3pf is required for the pad metal and field plate device. The total capacitance of the input resistor and input gates should therefore not exceed 2pf, according to this simple analysis, for inputs that are connected to a bussed signal source.

### B.4.2 Package Parasitics

For high-speed applications, it is very important to take into account the parasitics of the package into which the IC will be bonded; these parasitics are an essential part of the input network. The package and bond wire contribute a parallel capacitance and a series inductance and resistance. The inductance is typically the most important of these parasitics. Some data on package parasitics was presented in the trade rag *National Anthem* (Jan/Feb 1988, National Semiconductor). The following table summarizes this data for a 40-in dual-inline package, a 44-pin plastic leaded chip carrier (the target package for Pxpl5 EMC), and a 132-pin quad flat-pack:

| | 40-pin DIP | | 44-pin PLCC | | 132-pin QFP | |
|---|---|---|---|---|---|---|
| | Long | Short | Long | Short | Long | Short |
| Lead Length (inches) | 0.99 | 0.13 | 0.20 | 0.15 | 0.38 | 0.26 |
| Resistance (mΩ) | 125 | 123 | 98 | 98 | 102 | 101 |
| Inductance (nH) | 22 | 3.9 | 4.6 | 3.3 | 10 | 7.2 |
| Capacitance (ff) | 680 | 120 | 160 | 120 | 210 | 150 |

**Useful rules of thumb:** a typical bond wire has 1nH of inductance; inductance for the package is ~ 1nH/mm of package trace-conductor length.

# Appendix C — Electromigration

In this Appendix, we develop rules of thumb for sizing metal conductors so as to avoid failures due to electromigration. We consider two cases: first, a conductor carrying uni-directional, but time-varying current (*e.g.*, a power rail); second, a conductor carrying a (roughly symmetric) alternating current (*e.g.*, a clock rail).

## C.1 Electromigration in DC-carrying Conductors

Various sources agree that a reasonable estimate of the mean time to failure for a metal conductor on a chip due to electromigration goes as

$$T_{MTF} \propto A\, J^{-2} \exp[Q/kT]$$

inversely proportional to the square of the current density in the wire. This notion leads to a design rule that specifies the maximum allowed current per unit width of a metal wire.

How, then, to calculate the current, given that currents within chips are not steady, but time varying? Various sources agree that the best estimate for purposes of figuring electromigration current limits is to take the RMS value of the time varying current.

Consider the situation of a mosfet charging a capacitor C. The actual current waveform is complicated by the non-linear nature of the drain characteristics of the mosfet. We consider a simplified model to develop a pessimistic result. Wayne Dettloff has informed us that many in the industry assume an exponential charging current for these calculations. For an exponential charging of a capacitor

$$V = V_0\, e^{-t/RC}$$

and

$$I = \frac{V_0}{R} e^{-t/RC}$$

The mean-square of the current is

$$I_{MS} = \frac{1}{T} (V_0/R)^2 \int_0^T e^{-2t/RC}\, dt$$

Assuming the the period over which the average is to be taken is much greater than a time constant,

$$I_{MS} = \frac{RC}{2T} (V_0/R)^2 = \frac{V_0^2 C^2}{T^2} \cdot \frac{T}{2RC}$$

and the RMS current is

$$I_{RMS} = \frac{V_0 C}{T} \sqrt{\frac{T}{2RC}}$$

For NRZ-style signals, we typically want to design so that under Worst-case conditions, a given load can be charged or discharged fully in roughly half a clock cycle. Under Best-case conditions

(worst-case for power consumption and electromigration), the same circuit will operate substantially faster; in the case of the Hewlett-Packard CMOS40 process, for example, this factor is roughly 3. So we can argue that, for NRZ signals,

$$4 \cdot RC \sim T/6$$

and the RMS current is therefore about

$$I_{RMS} \sim \frac{V_0 C}{T}\sqrt{12} \sim 3.5\, CV_0 f$$

For clock signals, the charging time is roughly half of half of a clock cycle, so the RMS current is roughly

$$I_{RMS} \sim \frac{V_0 C}{T}\sqrt{24} \sim 5\, CV_0 f$$

We therefore propose the rules of thumb:

$$I_{RMS}(NRZ) \leq 3\, CV_0 f$$

$$I_{RMS}(RZ) = 5\, CV_0 f$$

where rounding down is reasonable for NRZ signals, since they tend to have fairly soft leading edges, while clock signals really need to do most of their charging in somewhat less than one-quarter of a cycle to allow for non-overlap. For the MOSIS 2μ fabrication, the ratio between Best and Worst case is closer to 4, so these rules probably should be revised upward somewhat.

Note that the product CV0f can be found from CAzM's "power" command. The command prints a report at the end of transient simulation that contains a line for "Average power consumed" from the Vdd voltage source. Dividing this number by the voltage on Vdd gives the average current consumed, and this can be taken to be the product CV0f.

## C.2 Electromigration in AC-carrying Conductors

There seems to be some disagreement over whether there even exists an electromigration problem for symmetric AC currents flowing in metal wires. The basic notion is that the microscopic process of electromigration is reversible when the current is reversed. The Texas Instruments reference states that this is not quite true because:

> ...(1) the entropy of a thermodynamic system continues to increase with time, i.e., the dislocated Al atoms never quite return to their original positions and (2) gradients (thermal, microstructural, etc.) can turn a seemingly bi-directional diffusion process into a preferred uni-directional process. Therefore, EM failures can be expected due to AC pulsing but whether [MTF estimate based on RMS current values] can be used to accurately predict its effect is unclear. Probably [the RMS model] is too conservative in its EM prediction for AC pulses.

On the other hand, Dillinger says

> The electromigration of metal atoms is reversible in the sense that a reversal of the direction of current flow reverses the momentum direction imparted to the atoms. The reversibility

of this phenomenon is dependent on the extent to which the vacancies have concentrated into a lower potential energy void in the metal; the larger the void size, the less likely that migrated ions will reverse their action. This can be demonstrated by evaluating the resistance of the interconnection under high current density stress over time. The resistance increases as electromigration progresses, but begins to decrease (reversing the process) with a reversal in the current direction. The impact on the chip designer is that the net zero charging transient current in global signal interconnections is not an electromigration concern.

Mike Barbour of HPNID reports that HP's internal engineering staff recommends a maximum RMS alternating current of 6 ma/micron of metal width for the CMOS40 process. This is eight times the 0.75 ma/micron maximum current for DC. One could view this recommendation in an alternate way by computing the effective currents as 1/8th of the RMS currents. The rules of thumb for AC current calculation for EM would then be

$$I_{eff}(NRZ) = 0.4\ CV_0f$$

$$I_{eff}(RZ) = 0.6\ CV_0f$$

where C is the load capacitance on the clock or signal rail, $V_0$ is Vdd, and f is the clock frequency.

## C.3 References

Dillinger, T.E., "VLSI Engineering", (Prentice-Hall, 1988) pages 388-389.

McPherson, J.W., Ghate, P.B., and Perrylou, "Design Implications of Transient-Induced EM Failures" (Texas Instruments internal document).

# Appendix D — Well/Substrate Error Checks

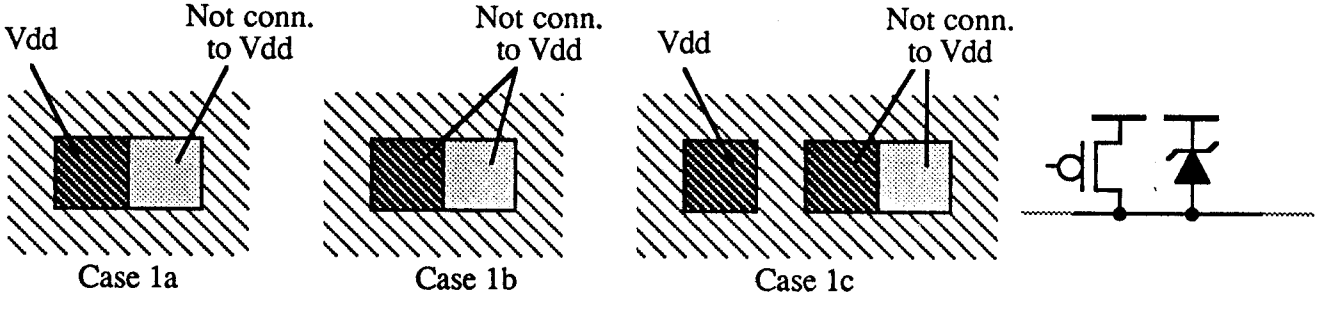
In this Appendix, we discuss the problem of detecting errant well and substrate contacts, and floating nwells. Errors of this kind can be extremely serious, since some combinations can lead to dead silicon. This Appendix develops procedures for detecting the presence of such difficulties.

## D.1 The Problem

For purposes of this discussion, we will use the following stipples to represent various flavors of diffusion:


It is possible to identify seven situations of incorrectly connected well/substrate contacts. We believe these cases are exhaustive. They are:

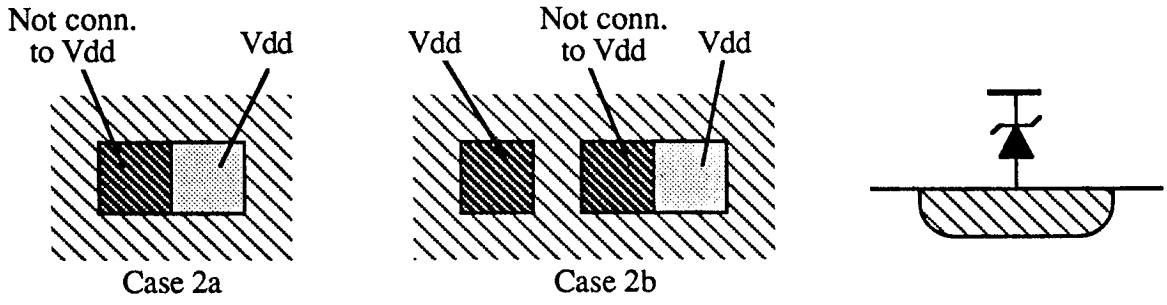
### Case 1 – Zener diode short, signal node to Vdd




One of these situations arises when an n-substrate contact (nsd) is abutted to a piece of p-diffusion (pdiff), typically the drain of a pfet. It is the kind of mistake an inexperienced designer would be likely to make, putting the nwell plug at the "wrong end" of a device. The pdiff node is intended to be a signal node, but the Zener has the unfortunate effect of shorting the node to Vdd so that it is impossible to pull down. This is precisely the situation that was found by the MIPS-X team at Stanford in one of their early designs ("The Design and Testing of MIPS-X", *Proceedins of the Fifth MIT VLSI Conference*, page 105-106).

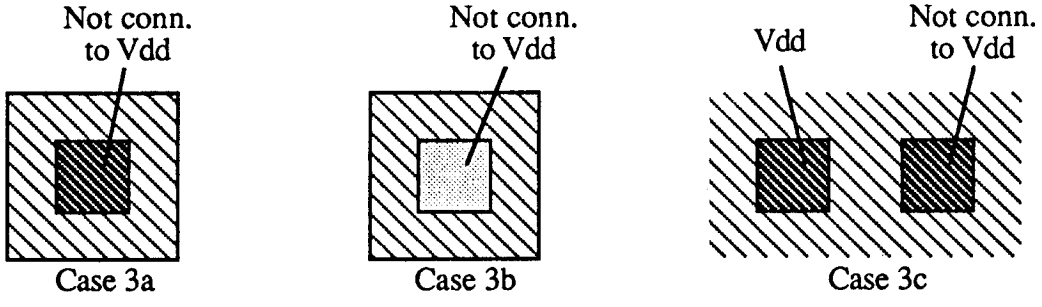
In Cases 1a and 1c, the nwell will be properly connected to Vdd, while in Case 1b, it won't. Case 1b could likely result in a circuit that would easily latch-up. All of the flavors of Case 1 should be considered **FATAL**. (Here's yet another reason not to use the abutting style of well contacts, where nsd and pdiff sit next to each other, unless it is required by density considerations.)

### Case 2 – Zener diode between Vdd and nwell



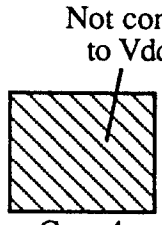

This situation might occur for an abutting-style nwell contact where the pdiff (typicall part of the source of a pfet) is correctly connected to Vdd, but the nsd is not. In Case 2a, the ONLY contact to nwell is the offending nsd area. This situaton is very likely to be **FATAL**. In Case 2b, the nwell is properly connected by a second, unrelated piece of nsd; this situations is not likely to be fatal, but it is rather unesthetic, and should be avoided.

### Case 3 – Isolated, unconnected nsd's and pdiff's




One of these situations is likely to arise when a designer carelessly leaves a piece of nsd or pdiff off in space, far removed from other circuitry. This can happen fairly easily in cells that are large compared to the smallest layout feature; nsd is particularly difficult to see when such a cell is painted on the magic screen. Cases 3a and 3b are likely to be **FATAL**. Case 3c covers the situation where there is a (probably large) piece of nwell, properly connected to Vdd by a correctly installed nsd, but there is a second, unconnected nsd somewhere in the well. This case is not likely to be fatal, but should be avoided.

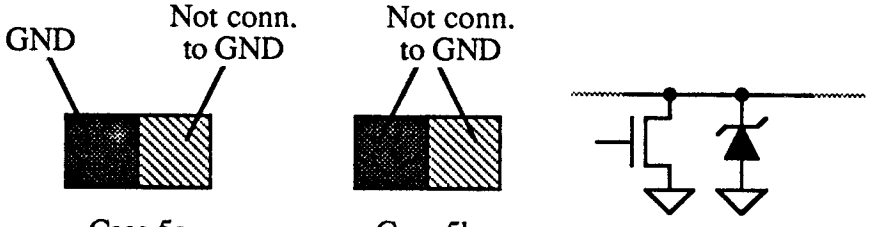
### Case 4 – Floating nwell




Case 4

This situation arises when a designer accidently draws a piece of unconnected nwell that is far enough from properly conected nwell that it is not merged in the CIF generation process. This can happen fairly easily when cropping areas of nwell in large cells, because nwell is difficult to see on the magic screen in these situations. Case 4 should always be considered to be **FATAL.**

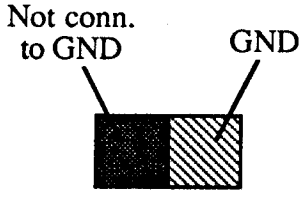
### Case 5 – Zener diode short, signal node to GND




Case 5a Case 5b

These situations are the mirror image of Case 1, but they are somewhat simpler (in an nwell process) since no well is involved. Both Cases should be considered **FATAL.**

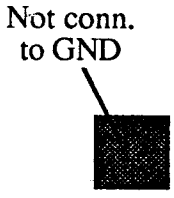
### Case 6 – Zener diode between GND and substrate




Case 6

This is the mirror image of Case 2, in which an abutting substrate contact is not properly strapped to GND by metal. This situation may not be fatal: the substrate is continuous across the chip, rather than broken into islands like nwell, and there is likely to be a properly connected substrate contact somewhere nearby. This construction may cause problems if it is part of a guard structure for a high-current driver. It should clearly be avoided.

### Case 7 – Isolated psd feature




Case 7

This situation arises when the designer accidentally draws a piece of psd that is not connected to anything. It is almost certainly not fatal, but should be avoided.

## D.2 Detecting Well/Substrate Errors

### D.2.1 Zener Diode Detection

The method for detecting Zener diode constructions involves extracting a circuit with a special technology file that causes abutting nsd/pdiff and abutting psd/ndiff to be electrically connected. When this extracted circuit is converted to "sim" format by **ext2sim**, the alias file (.al file) will differ from a normally-extracted .al file **if and only if** there are zener diode errors. Cases 1a, 2a, 2b, 5a, and 6 will cause some internal circuit nodes to be aliased to Vdd or GND, while Cases 1b, 1c, and 5b will cause two internal nodes to be aliased together. For any of these cases, the designer can get a reasonable idea of the location of the bad construction by *grep*ing for the offending node in the .ext file for his chip.

Unfortunately, this simple approach doesn't always work. For some reason, known only to the implementers of ext2sim, the alias files extracted under the two technologies may differ in the *order* of the lines of the file. For the purpose of verifying correctness (ensuring that there are no new aliases introduced), it is probably sufficient to determine that the two .al files are the same *size*. If the sizes are different, there is no obvious, simple way to identify the offending node; the designer will have to employ some cleverness to filter out the differences that result from re-ordering of the lines in the .al files.

### D.2.2 Isolated psd and nsd

We can get double duty from the extraction of D.2.1 by using another trick. In the "extract" secion of this special technology file, we set all layer capacitances to 0 *except* for psd,psc/active. The .sim file that results from the above extraction will therefore have capacitance records (C in first column) *only* for nodes that have psd,psc/active attached to them. A correctly constructed circuit should produce exactly one C-record of the form

```
C Vdd GND [number]
```

If there any lines of the form

```
C [nodename] GND [number]
```

where nodename ≠ Vdd, then an isolated chunk of psd has been detected. Note that this will catch constructs Case 3a and 3c (Case 3b will produce an isolated nwell, which we will consider

separately along with Case 4).

We can use a similar trick to find isolated nsd's (Case 7) at the same time. We assign all layers 0 *resistance* except nsd,nsc/active. A correctly constructed circuit will produce exactly one R-record in the resulting .sim file of the form

```
R Vdd [number]
```

If there are any other lines of the form

```
R [nodename] [number]
```

then an isolated chunk of nsd has been detected.

### D.2.3 Isolated nwell

Detecting isolated nwells is considerably complicated by magic's feature of automatically generating nwells during the CIF output process. To be absolutely sure one has found all such nwells, it is necessary first to generate CIF for a chip, thus producing all of the nwell features that one intends to send to fabricaton. This .cif file must then be read back into magic to produce a new magic layout for the chip; this layout can then be analysed for isolated nwells by circuit extraction.

Extraction for isolated nwells must also be done with a special technology file. Tech files have two parts that deal with the interconnection of layers by contacts. The "connect" section typically has a line of the form

```
nwell,nsc,nsd   nwell,nsc,nsd
```

which is supposed to tell the extractor (and other net-related tools) that anything on the left is connected to anything on the right if they abutt. While some parts of magic seem to understand this connection, the circuit extractor doesn't. Extraction will generate a separate node for each piece of explicitly drawn nwell. We believe this is a bug in the extractor, though that is not entirely clear.

The "contact" section typcially has a line of the form

```
nncont  nndiff  metal1
```

which tells parts of the magic system that nncont (or nsc) connects nndiff (or nsd) to metal1. To get the extractor to connect nwell to nsc, one has to modify the "contact" section of the tech file, replacing the above line, for example, by

```
nncont  nndiff  metal1  nwell
```

which tells all parts of magic that an nncont (nsc) connects all three layers together. This special tech file must also make sure that nwell features generate capacitance, and thus nodes, in the resulting .sim file. To make sure of this, we use the same trick as for isolated nsd and psd, setting the capacitances of all layers to 0 *except* nwell.

When the .cif file from the original layout is read into magic, it will have lost all of its labels. This is ok for our purposes, however, perhaps even an advantage. One must read in the .cif, label each

of the idependent Vdd and GND nets in the chip, then extract the circuit, all using the special tech file for isolated nwell detection. Finally, a .sim file is created, and examined for C-records. A correctly constructed chip will have one such record of the form

```
C Vdd GND [number]
```

If there any lines of the form

```
C [nodename] GND [number]
```

where nodename ≠ Vdd, then an isolated chunk of nwell (Case 4) been detected.

## D.3 Special magic Technology Files

### D.3.1 Zener Diode Detection

For the HPCMOS40 technology, we have created a special tech file for detecting Zener diode constructions, isolated psd, and isolated nsd. This file is located in

/mira/jp/icfab/hpcmos40/tech/Zener

and can be conveniently accessed using the command

/mira/jp/bin/hpzenermag

which fires up magic in the current directory, but with the funky technology "hpzener". This technology file differs from the standard "hpcmos40" in the following ways:

(1) The "extract" section has only one style, "substrate_rc". This style assigns 0 capacitance to all layers except psd,psc/active, and 0 resistance to all layers except nsd,nsc/active.

(2) The "drc" section is empty. Presumably the design has already passed its design rule check, so there is no need to perform further checks.

(3) The "connect" section of technology "hpzener" has two additional lines of the form

```
nsd,nsc/active      pdiff,pdc/active
psd,psc/active      ndiff,ndc/active
```

which explicity connected nsd to ndiff or psd to pdiff in abutting structures.

### D.3.2 Isolated nwell Detection

The tech file "hpnwell" is intended for isolated nwell detection. The file is located in

/mira/jp/icfab/hpcmos40/tech/NWell

and can be conveniently accessed using the command

/mira/jp/bin/hpnwellmag

which fires up magic in the current directory with the funky technology "hpnwell". This

technology differs from standard hpcmos40 in the following ways:

(1) The "extract" section has only a single style "nwell_cap", where all layers have zero capacitance and zero resistance except nwell.

(2) The "drc" section is empty.

(3) The "contact" section contains the line

```
nncont     nndiff     metal1     nwell
```

to connect nwell explicitly to nsd.

### D.4 Proceedure for Well Checks on a Finished Chip

This proceedure assumes you have

(1) finished your chip design,
(2) that it has passed net-list checks contained in *alchk*, *conch*, and *secs*,
(3) that it is free of design rule violations,
(4) that you have generated files "Chip.al" and "Chip.sim" using the standard extraction process and "ext2sim [-L -C -R] Chip", and that
(5) you have generated a cif output file, ready to process by 2mosis.

#### D.4.1 Zener Checking

(1) Go to the directory in which "Chip.mag" is located. Move the file "Chip.al" to "Chip.al.orig" and move "Chip.sim" to "Chip.sim.orig". If disc space is likely to be a problem, you can remove "Chip.sim" or move it to a different disc partition or to another machine, as it will not be needed for these checks.

(2) Invoke magic using

```
/mira/jp/bin/hpzenermag
```

(3) Extract your circuit by entering

```
:load Chip
:extract all
```

(4) When extraction is complete, exit magic **without writing anything out.**

(5) Create "Chip.al" and "Chip.sim" by invoking

```
ext2sim -L -r 1 -c 1 Chip
```

(6) Compare the sizes of the two ".al" files created under the different technologies by

```
lsl Chip.al Chip.al.orig > al.count
```

(7) Examine the file "al.count". The two files should have the same size, exactly. If they don't, you will have to *diff* the files, and then perhaps employ some cleverness to find the

offending node. The diff will likely produce many lines because of ext2sim's annoying habit of re-ordering the lines in the .al file.

(8) Look for dangling psd by doing

```
grep "^C" Chip.sim
```

which should return one, and only one, line of the form

```
C GND GND [number]
```

Any other lines of the form

```
C [node-name] GND [number]
```

imply that [node-name] is connected to a dangling piece of psd.

(9) Look for dangling nsd by doing

```
grep "^R" Chip.sim
```

which should return one, and only one, line of the form

```
R Vdd [number]
```

Any other lines of the form

```
R [node-name] [number]
```

imply that [node-name] is attached to a dangling piece of nsd.

(10) If all is well, get restore the .sim and .al files by

```
mv Chip.sim.orig Chip.sim
mv Chip.al.orig Chip.al
```

### D.4.2 Checking for Dangling nwell

(1) Copy your file "Chip.cif" to a new, empty directory. If disc space is likely to be a problem, you can move to the new directory and make a "soft link" to Chip.cif. In any case, make sure your disc partition has enough room to accept *all* of the .ext files that will be generated when you extract Chip. "cd" to the new directory.

(2) Invoke magic by

```
/mira/jp/bin/hpnwellmag
```

(3) Make sure that cif istyle is set properly by

```
:cif istyle lambda=0.4
```

(4) Read in your cif file by

```
:cif read Chip
```

(5) Magic will instatiate a cell in "(UNNAMED)" called "Chip"; enter

```
:load Chip
```

(6) Label each of the independent Vdd and GND nets.

(7) Extract this layout with

```
:extract all
```

This process will result in a whole bunch of .ext files in the current directory[1].

(8) Exit magic **without saving anything** (or put magic in background). Generate a .sim file (.al file is not needed) by

```
ext2sim -L -r 1 -c 1 -A Chip
```

(9) When ext2sim is finished, look for dangling nwells with

```
grep "^C" Chip.sim
```

You should get one, and only one[2], line of the form

```
C Vdd GND [number]
```

Additional lines of the form

```
C [node-name] GND [number]
```

will alert you that there are dangling nwells.

(10) If you want to save a dated record that you have performed this procedure, then send the output of the "grep" above to a file, such as "Chip.nwell.ok" or some such. If all is well, remove all .ext files, the .sim file, and the .cif file (unless it is a soft-link).

## D.5 Checking an SCMOS Design

If your design was carried out using MOSIS's scmos rules, you can use the same procedure outlined above to check for bad substrate diffusions/contacts. Assuming the design was implemented in an nwell technology, the proceedure is essentially identical. You should generate

[1] It may also take a great deal of time. For a ~400K transistor design, the nwell location process required about a day's computing on a 2-processor VAX3600.

[2] There is a chance that no such line will be generated. The value of the node capacitance on Vdd is likely to be very large. We have found that if the value overflows the maximum integer size, the ext2sim will fail to output any C-record at all. To cover this possibility, the special technology file also sets all resistances on all layers to zero, *except* nwell, which is set to 100 Ω/square. There should be only one R-record, associated with the Vdd node.

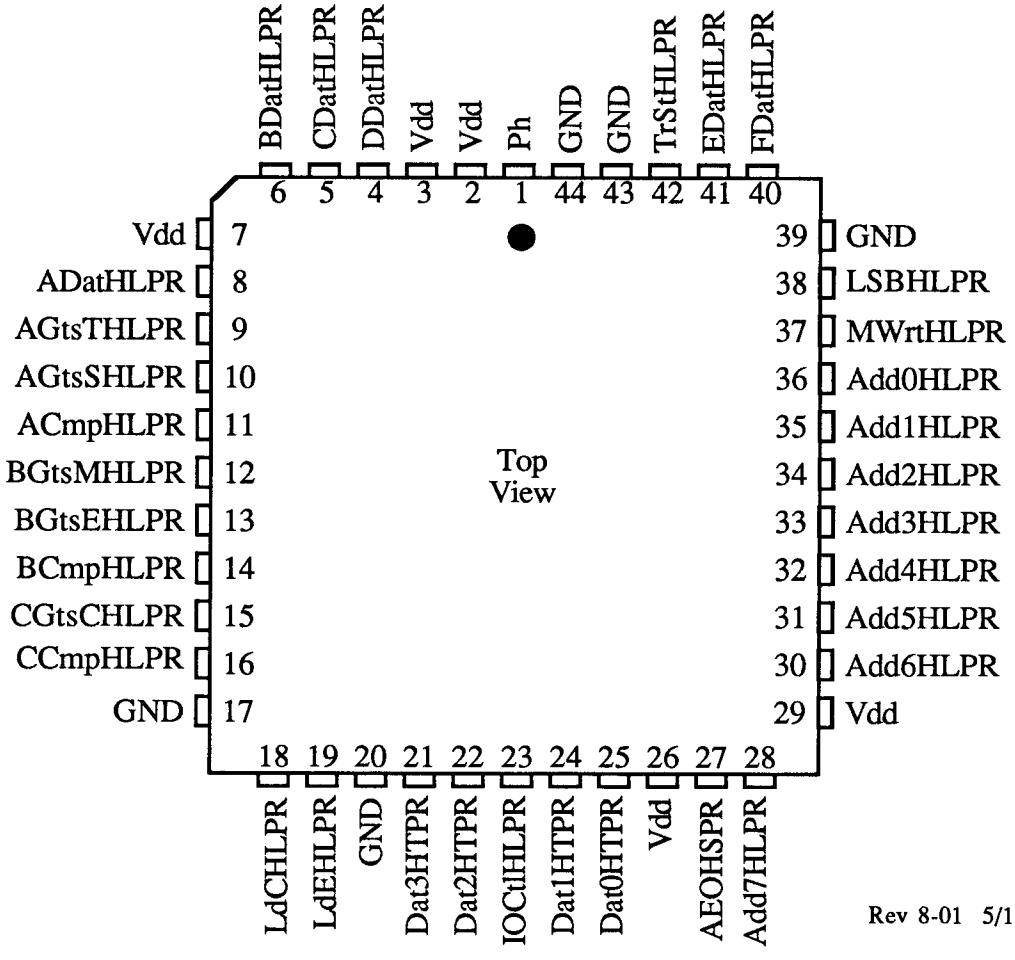
cif with the appropriate "ostyle" for the intended fabricator (*e.g.*, "cif ostyle lambda=1.0(nwell)"). The funky technology files are called "sczener" and "scnwell", and can be invoked using

```
~jp/bin/sczenermag
~jp/bin/scnwellmag
```

For reading in cif, you should set the cif istyle to the appropriate value (*e.g.*, "cif istyle lambda=1.0(nwell)" ).

# Part IV, Chapter 2, Part 1
# Pxpl 5 Enhanced Memory Chip Specifications

The Pxpl5 EMC is a logic-enhanced memory chip, an array of which implement a 'smart' virtual image buffer. The chip contains circuitry for two columns of 128 pixels in a virtual pixel space. Each pixel has 208 bits of random-access memory, addressable bit-wise under control of address inputs **Add<0:7>HLPR**; the memory can be written when input **MWrtHLPR** is asserted. Each pixel has a simple bit-serial ALU, fully horizontally microcode controllable with inputs **AGstTHLPR, AGtsSHLPR, ACmpHLPR, BGtsMHLPR, BGtsEHLPR, BCmpHLPR, CGtsCHLPR, CCmpHLPR, LdCHLPR,** and **LdEHLPR**. Each pixel also sits at a leaf output of a bi-quadratic expression evaluator that supplies bit-serial values of the expression Ax+By+C+Dx2+Exy+Fy2 to the pixel ALU; the inputs **ADatHLPR...FDatHLPR** provide bit-serial inputs to the chip for each coefficient of this expression, **LSBHLPR** is a pipeline control input. A special register in the pixel memory can be read or written on data port **Dat<0:3>HTPR** under control of input **IOCtlHLPR**. All chip internal operations and data port transfers are controlled by the clock **Ph**; **TrStHLPR** steps the operation of the expression evaluator. **Vdd/GND** are power pins; nominally GND = 0 volts (ground), Vdd = 5.0 volts. The Pxpl5 EMC is packaged in a 44-pin CLCC or PLCC with pinout shown below.




Rev 8-01 5/11/89

## 1.1 External Interface Signals

| Signal Name | Pin | Description |
|---|---|---|
| | | **Power/Ground** |
| **Vdd** | 2 | Nominally +5.0 volts, two of these pads power the chip's internal circuitry, while the remainder power the ChipIO circuitry. Input and output pads have their own separate supply pins, 2 of which are shared with the clock generator ClockGen. |
| **Vdd** | 3 | |
| **Vdd** | 7 | |
| **Vdd** | 26 | |
| **Vdd** | 29 | |
| **GND** | 17 | Signal and power ground. Two pads for internal circuitry, 5 for ChipIO (separate GND and Vdd for I/O pads and ClockGen). |
| **GND** | 20 | |
| **GND** | 39 | |
| **GND** | 43 | |
| **GND** | 44 | |
| | | **Data/Control Inputs/Output** |
| **Ph** | 1 | The EMC system clock. All I/O are referred to the rising edge of this clock, whose nominal frequency is 40.0 MHz. Duty factor for this signal should be closely controlled to 50%. |
| **ADatHLPR** | 8 | The six Multiplier data input pins, these correspond to the six coefficients in the bi-quadratic expression $Ax+By+C+Dx^2+Exy+Fy^2$. These coefficients are bit-aligned in the Pxpl5 EMC (unlike its predecessor, Pxpl4). They are applied LSB-first. |
| **BDatHLPR** | 6 | |
| **CDatHLPR** | 5 | |
| **DDatHLPR** | 4 | |
| **EDatHLPR** | 41 | |
| **FDatHLPR** | 40 | |
| **LSBHLPR** | 38 | The pipeline control pin. An LSB 'token' is injected by asserting this signal, nominally for one clock cycle. Successive tokens should be at least 12 cycles apart (*i.e.*, they should be separated by 11 cycles). See Section 2.3 and documentation on the IGC (§IV.3) for further details on Multiplier timing. |
| **TrStHLPR** | 42 | This signal (Tree Step), when de-asserted, causes the Multiplier to halt, savings its current state dynamically. |
| **AGtsTHLPR** | 9 | The control signals for the ALU. Their function is summarized in the tables below: |
| **AGtsSHLPR** | 10 | |
| **ACmpHLPR** | 11 | |
| **BGtsMHLPR** | 12 | |
| **BGtsEHLPR** | 13 | |
| **BCmpHLPR** | 14 | |
| **CGtsCHLPR** | 15 | |
| **CCmpHLPR** | 16 | |

**AField**

| AGtsT | AGtsS | ACmp | Function |
|---|---|---|---|
| 0 | 0 | 0 | A = 1 |
| 0 | 0 | 1 | A = 0 |
| 0 | 1 | 0 | A = Sum |
| 0 | 1 | 1 | A = !Sum |
| 1 | 0 | 0 | A = Tree |
| 1 | 0 | 1 | A = !Tree |
| 1 | 1 | 0 | A = Sum•Tree |
| 1 | 1 | 1 | A = !(Sum•Tree) |

**BField**

| BGtsM | BGtsE | BCmp | Function |
|---|---|---|---|
| 0 | 0 | 0 | B = 1 |
| 0 | 0 | 1 | B = 0 |
| 0 | 1 | 0 | B = Enab |
| 0 | 1 | 1 | B = !Enab |
| 1 | 0 | 0 | B = Mem |
| 1 | 0 | 1 | B = !Mem |
| 1 | 1 | 0 | B = Mem• Enab |
| 1 | 1 | 1 | B = !(Mem• Enab) |

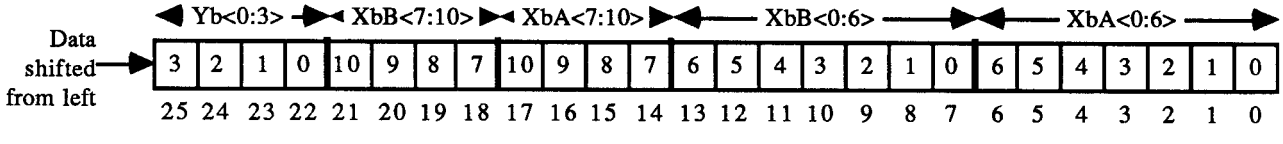
**CField**

| CGtsC | CCmp | Function |
|---|---|---|
| 0 | 0 | C = 1 |
| 0 | 1 | C = 0 |
| 1 | 0 | C = Cry |
| 1 | 1 | C = !Cry |

**Note well:** one particular "redundant" set of ALU codes is reserved and decoded to allow the IGC to select and load the Configuration Register (CfReg). This code is:

AGtsT • AGtsS • !BGtsM • !BGtsE • !BCmp • !CGtsC • CCmp

(In other words, the function selection for B=1,C=0 is not permitted; this function is "redundant" in the sense that B=0,C=1 produces the same results.) CfReg contains the x,y address of each of the EMC's two Planes of pixels. It is selected whenever the above redundant code is input to the EMC *AND* the pixel memory address **Add<0,7>HLPR** has the value **Add** = 0. On clock cycles where this condition obtains, the bit value on the input **ACmpHLPR** is shifted into CfReg. CfReg is organized as follows:



Xb{A,B} are the "base" X-addresses of the A and B Planes of pixels; each Plane represents a vertical column of pixels on the screen coordinates. Likewise Yb is the "base" Y-address of *both* Planes' columns of pixels; in the conventional coordinate system of Pxpl5, this is the *top*-most pixel in a column.

**LdCHLPR** 18
**LdEHLPR** 19

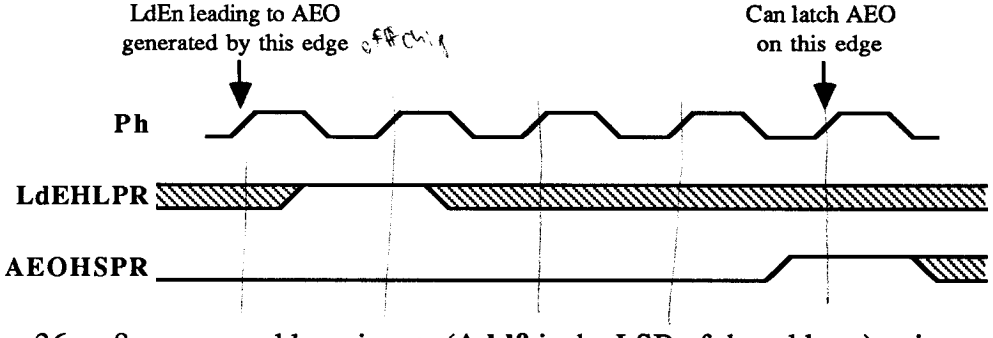
Register load signals for the ALU; their function is summarize below:

**LField**

| LdC | LdE | Function |
|---|---|---|
| 0 | 0 | Cry = Cry(old) ; Enab = Enab(old) |
| 0 | 1 | Enab = Cry(new) |
| 1 | 0 | Cry = Cry(new) |
| 1 | 1 | Cry,Enab saved; Write forced on **MWrt** asserted |

**AEOHSPR** 27

This signal, "All Enables are Off", is asserted when Enable registers in all 128 ALU's of both planes are cleared. Timing is such that there is a 4-cycle delay between issuing the instruction leading to assertion of **AEOHSPR** and latching the value resulting from that instruction:




**Add0HLPR** 36
**Add1HLPR** 35
**Add2HLPR** 34
**Add3HLPR** 33
**Add4HLPR** 32
**Add5HLPR** 31
**Add6HLPR** 30
**Add7HLPR** 28

8 memory address inputs (**Add0** is the LSB of the address) point to one of up to 208 bits to be operated upon on a given micro-cycle. Note that the low 32 bits comprise the Communications Register (CR). During an IO operation, these bits cannot be accessed from the ALU. If the programmer causes such accesses to occur, they will, however, neither corrupt memory contents nor disturb the IO operation.

**MWrtHLPR** 37

Causes pixel ALU data to be written to memory when asserted.

## I/O Interface Signals

**IOCtlHLPR** 23

This signal controls all operations of the data port, **Dat<0:3>HTPR**. When held HI for two or more cycles, it disables the port and sets the internal data pointer into the CR to nybble #0 of Pixel #0. Port operation is initiated by bringing **IOCtl** LO for one cycle to get the

attention of the port. During the next cycle, **IOCtl** = LO puts the port into the Read mode, **IOCtl** = HI puts the port into Write mode. Data transfers are then enabled by a *data request*, defined by bringing **IOCtl** HI for one cycle. (Note that the first request can follow immediately after an initializing sequence = 01.) Data transfers can be halted for any required length of time by holding **IOCtl** LO (any length of time up to the latency of the internal dynamic storage registers). See section 2.1.3 for timing details.

| | | |
|---|---|---|
| **Dat0HTPR** | 25 | Data I/O pins. During a Read data transfer, these pins can drive a nybble of CR data off-chip with timing such that data can be latched on the rising edge of **Ph**. During a Write transfer, data can be input on this port, and should be of type LPR. |
| **Dat1HTPR** | 24 | |
| **Dat2HTPR** | 22 | |
| **Dat3HTPR** | 21 | |

## 1.2 External Interface Specifications

**Operating Conditions**

| Parameter | | Min | Nom | Max | Units |
|---|---|---|---|---|---|
| Vdd | Power supply voltage | 4.5 | 5.0 | 5.5 | Volts |
| Vss | Power supply return | | 0 | | Volts |
| VIH | Logic-HI input voltage | 2.0 | | 5.5 | Volts |
| VIL | Logic-LO Input voltage | -1.2 | | 0.8 | Volts |
| TA | Ambient air temperature | 0 | 25 | 40 | C |
| TJ | Junction temperature | 25 | 50 | 70 | C |

**DC Electrical Characteristics** (Vdd = 5.0 volts)

| Parameter | | Conditions | Min | Nom | Max | Units |
|---|---|---|---|---|---|---|
| $V_{OH}$ | HI output voltage, TPR,SPR outputs | IOH = -20ma | 2.4 | | | Volts |
| $V_{OL}$ | LO output voltage, TPR,SPR outputs | IOL = 20ma | | | 0.36 | Volts |
| $I_{ISPR}$ | Input leakage current, LPR inputs | 0 ≤ Vin ≤ Vdd | | | 10 | µA |
| $I_{BH}$ | Input clamp current, I/O's HI | Vin = Vdd + 1 volt | | | 70 | mA |
| $I_{BL}$ | Input clamp current, I/O's LO | Vin = - 1 volt | | | -70 | mA |
| PD | Power dissipation, fPh = 40.0 MHz | | | 1.160[2] | | W |

**Capacitance**[4]

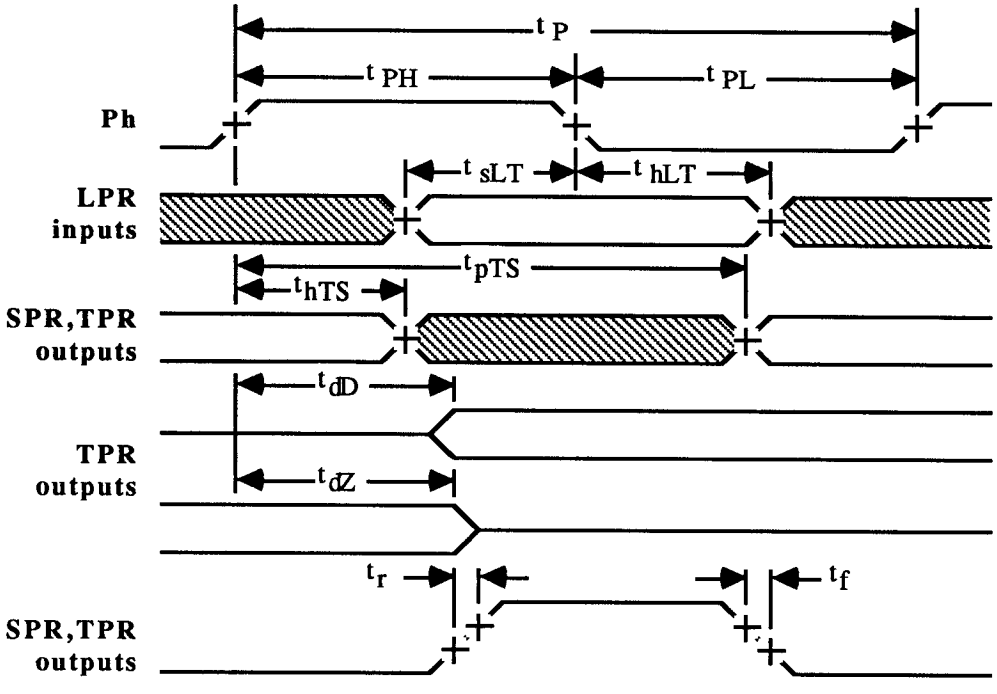
| Parameter | | Min | Nom | Max | Units |
|---|---|---|---|---|---|
| CLPR | Input capacitance, LPR inputs | | 3.7 | | pF |
| CTPR | Capacitance, TPR I/O pins | | 3.7 | | pF |
| CPh | Input capacitance, Ph clock input | | 5.4 | | pF |

**Timing Requirements** (Timing measurements are referred to Vin = 1.40 volts.)

| Parameter | | Min | Nom | Max | Units |
|---|---|---|---|---|---|
| tP | Clock period | | 25 | | ns |
| tPH | Clock HI period | | 12.5 | 14.0[2] | ns |
| tPL | Clock LO period | 7.0[2] | 12.5 | | ns |
| tsLT | Setup time, LPR,TPR inputs | –3.0[3] | | | ns1 |
| thLT | Hold time, LPR,TPR inputs | 9.0 | | | ns1 |

**Timing Characteristics**

| Parameter | | Conditions | Min | Nom | Max | Units |
|---|---|---|---|---|---|---|
| tdD | Delay, clock edge to data drive | CL=50Ω‖50pf | 5.4 | | 22 | ns1 |
| tdZ | Delay, clock edge to data Hi-Z | CL=50Ω‖50pf | 5.4 | | 23 | ns1 |
| trTS | Rise time, TPR, SPR outputs | CL=50Ω‖50pf | | | 5.6 | ns1 |
| tfTS | Fall time, TPR, SPR outputs | CL=50Ω‖50pf | | | 4.1 | ns1 |
| thT | Guaranteed hold time, TPR outputs | No Load | 9 | | | ns1 |
| tpT | Worst-case propagation time, TPR outputs | CL=50Ω‖50pf | 25 | | | ns1 |
| thS | Guaranteed hold time, SPR output | No Load | 9 | | | ns1 |
| tpS | Worst-case propagation time, SPR output | CL=50Ω‖50pf | 21 | | | ns1 |




**Notes:**

1 Based on CAzM simulation. Max. times: Worst conditions; Min. times: Best conditions.

2 Value obtained from measurement. Vdd = 5.0 volts, $T_A$ = 25°C, typical part from run M94V.

3 Setup times for data/control inputs are negative w.r.t. the falling clock edge (input can settle after edge).

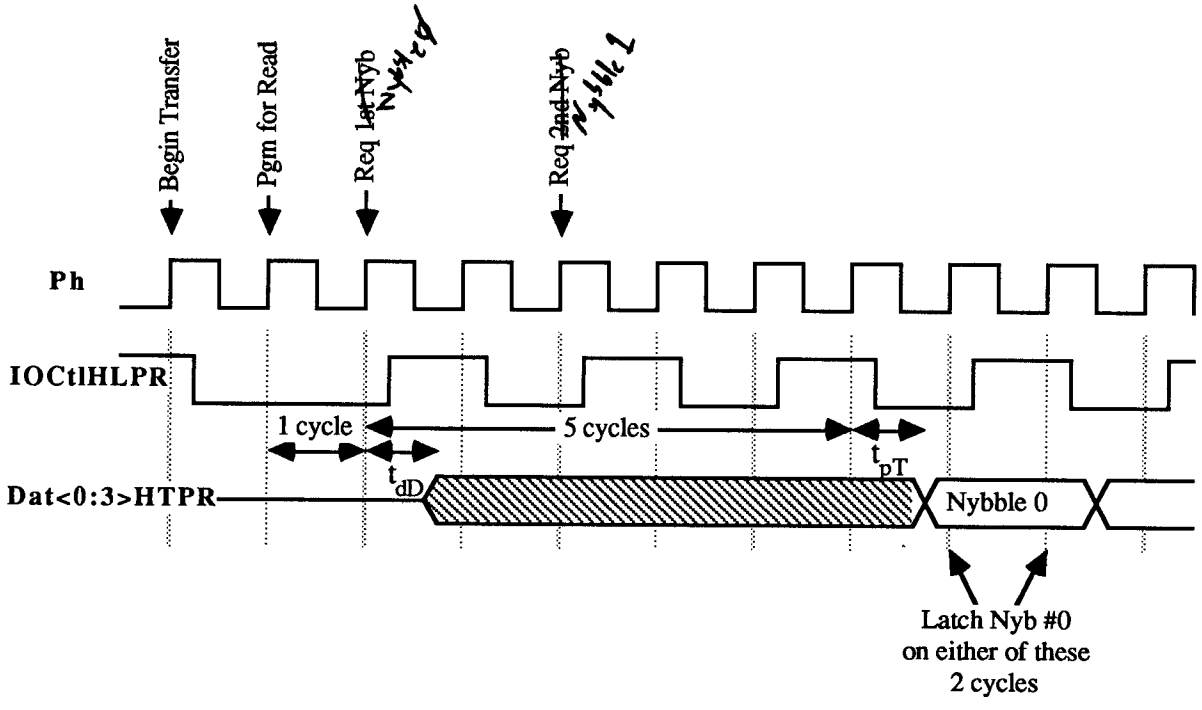
4 Measured using HP4195A, $T_A$=25°C, Vdd=5.0v, Vin=1.40v, typical part from run M94V.

## 1.3 IO Port Timing

This section summarizes the sequencing of the **IOCtlHLPR** signal to effect data transfers on the data port **Dat<0,3>HTPR**.

**IO Reset.** The communications register (CR) is a set of 32-bit registers, each of which comprises the lower 32 bits of a pixel's memory. Altogether, the CR contains 2,048 nybbles (256 pixels x 8 nybbles/pixel). An internal data pointer selects nybbles in the CR for reading or writing on the data port. This pointer may be reset to Nybble #0 of Pixel #0 on Plane A by holding **IOCtlHLPR** HI for four or more cycles.

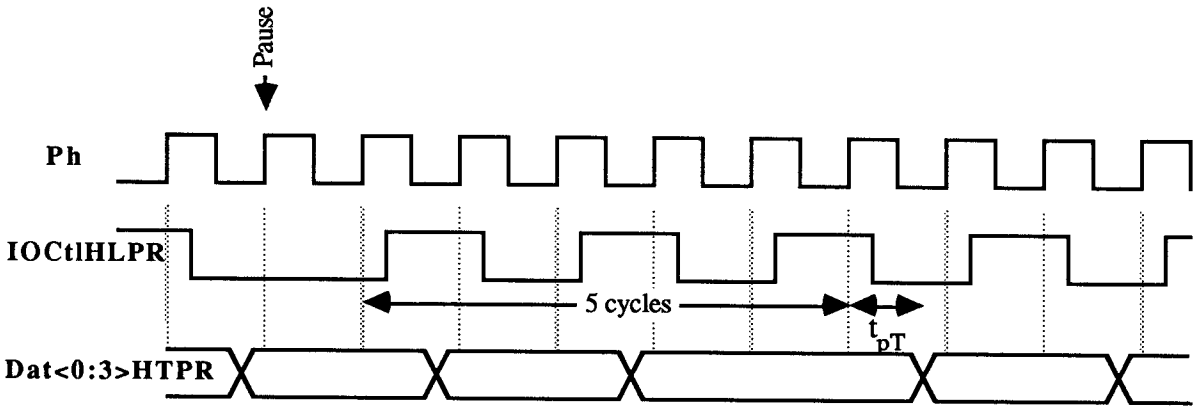
**Initializing Sequence, Reads. IOCtl** is brought LO for one cycle to get the port's attention. During the next cycle, **IOCtl** is left LO to program the port for a Read operation. **IOCtl** can then stay LO for as long as desired before the first data request.



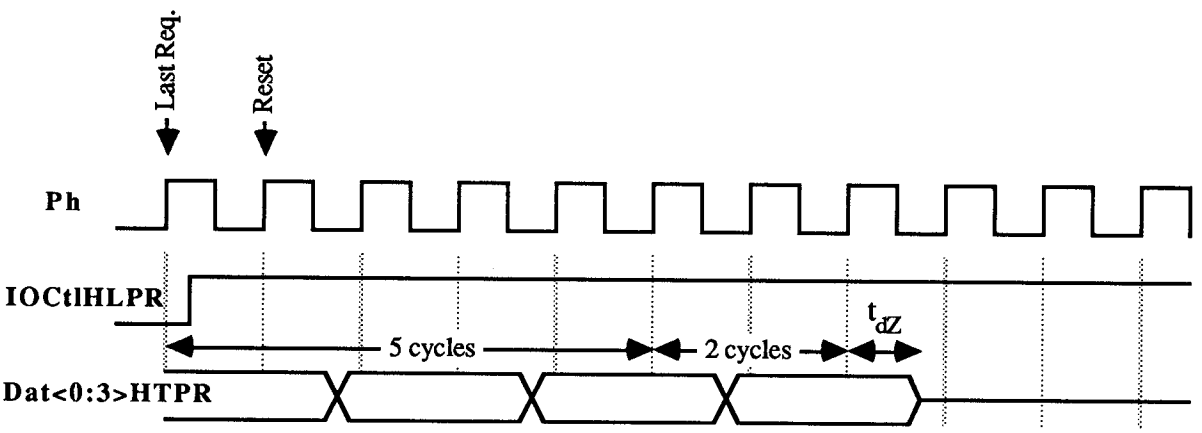

**IO Read Transfers.** A nybble is read from the CR by initiating an IO Request, defined by holding **IOCtl** HI for one cycle. Successive requests must be separated by one or more cycles with **IOCtl**=LO. During Read operations, the port **Dat<0,3>HTPR** becomes an active driver two cycles after the Read-programming cycle of IO initialization. The data output is undefined until 6 cycles after the first Request, at which point Nybble #0 of Pixel #0 on Plane A is output and stable during two successive cycles. Note particularly that the data output is undefined during the first 5 cycles during which the data port is actively driving data out onto the **Dat** pins. Reads from the CR fetch data from within the chip in pixel-major, nybble-minor order, alternating between Planes on every pixel. The order is as follows (Plane,Pixel,Nybble):

A,0,0 A,0,1 ... A,0,7 B,0,0 ... B,0,7 A,1,0 ... A,1,7 B,1,0 ... B,1,7 A,2,0 .... B,127,7

Successive Reads cause successive nybbles to appear 6 cycles after each Request. IO Reads may be paused for short periods (less than the latency of the internal dynamic latches in the data port, typically several hundred μsecs) by holding **IOCtl** LO. As shown below, a Pause leaves the last-requested data on the port for an extended period.

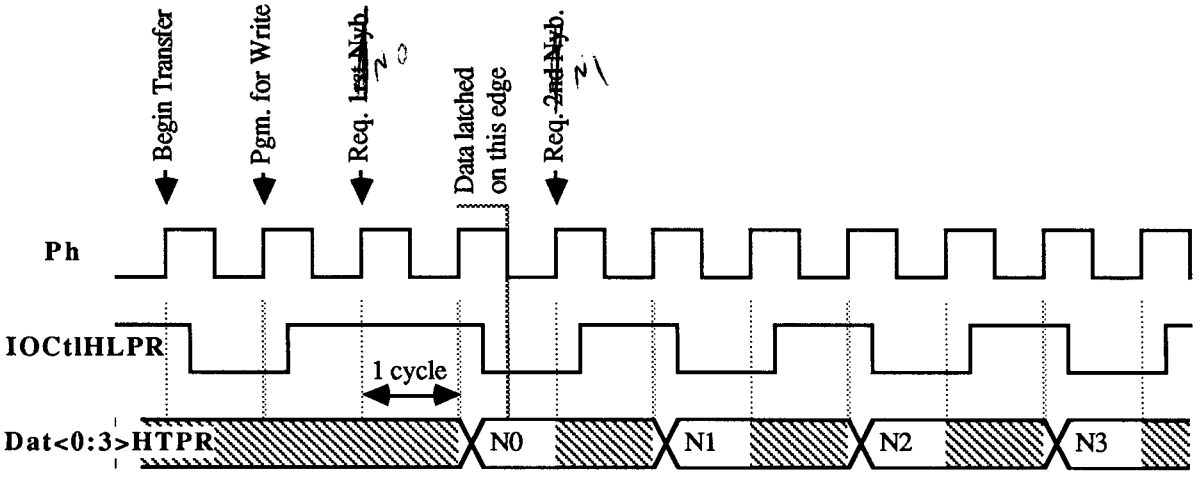



**CR Read Termination.** A Read sequence is terminated by holding **IOCtl** HI for two or more cycles. The port continues to drive the data pins until 8 cycles after the last Request preceding a Reset.

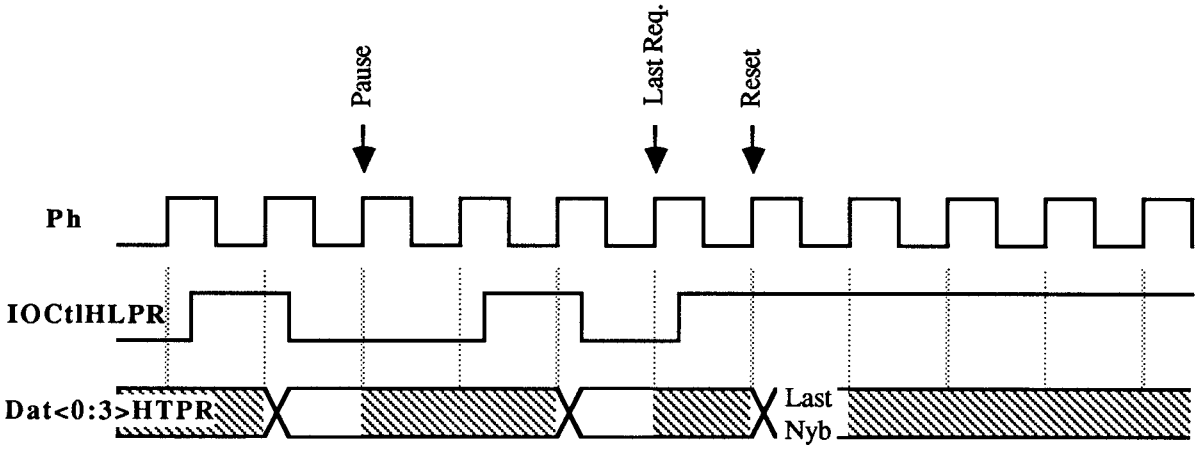



Normally CR IO Reads are expected to contain exactly 2048 Requests. On the 2049th Request, the internal Read circuitry is disabled and the data outputs become undefined after the 2048th nybble is output. The port continues to drive the output pins, however, until 8 cycles after the last Request preceding a Reset. For this reason, at least 8 cycles, during which **IOCtl** is HI, should be inserted between the last Read Request and the first LO-going transition for a Write operation.

**Initializing Sequence, Writes.** **IOCtl** is brought LO for one cycle to get the port's attention. During the next cycle, **IOCtl** is brought HI to program the port for a Write operation.




**CR Write Transfers.** A nybble is written to the CR by initiating an IO Request, defined by holding **IOCtl** HI for one cycle. Successive requests must be separated by one or more cycles with **IOCtl**=LO. During Write operations, the port **Dat<0:3>HTPR** should be driven with the data to be written; the data should be stable on the clock cycle *following* the request (and, of course, may be stable during the cycle of the request). Data is written into the CR in the same order as that for Reads (see above). CR Writes, like Reads, may be paused for short periods by holding **IOCtl** LO. As shown below, a request following a Pause requires the same timing relationship between valid data and its Request as a normal Request.




**CR Write Termination.** As with Reads, a Write sequence is terminated by holding **IOCtl** HI for two or more cycles. If more than 2048 Write requests are made, then the 2049th and succeeding Write Requests have no effect on the CR.

# Part IV, Chapter 2, Part 2
# Pxpl 5 Enhanced Memory Chip Logic Design

This section describes the logic design for the Pixel-planes 5 Enhanced Memory Chip. See §IV.2.1 for a description of the external interface to the EMC.

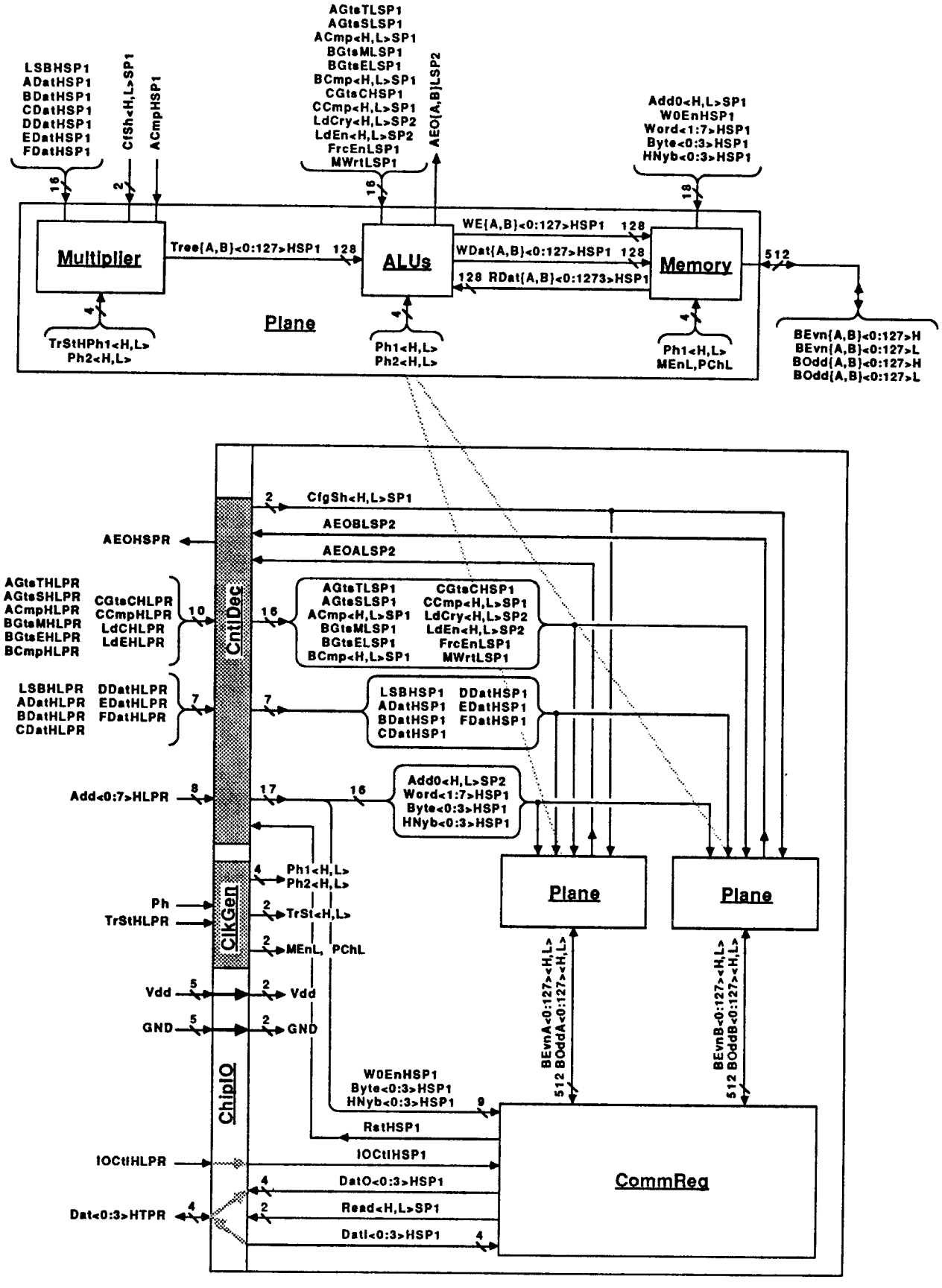



**Figure 2.2.1 Block diagram of Pxpl5 Enhanced Memory Chip.**

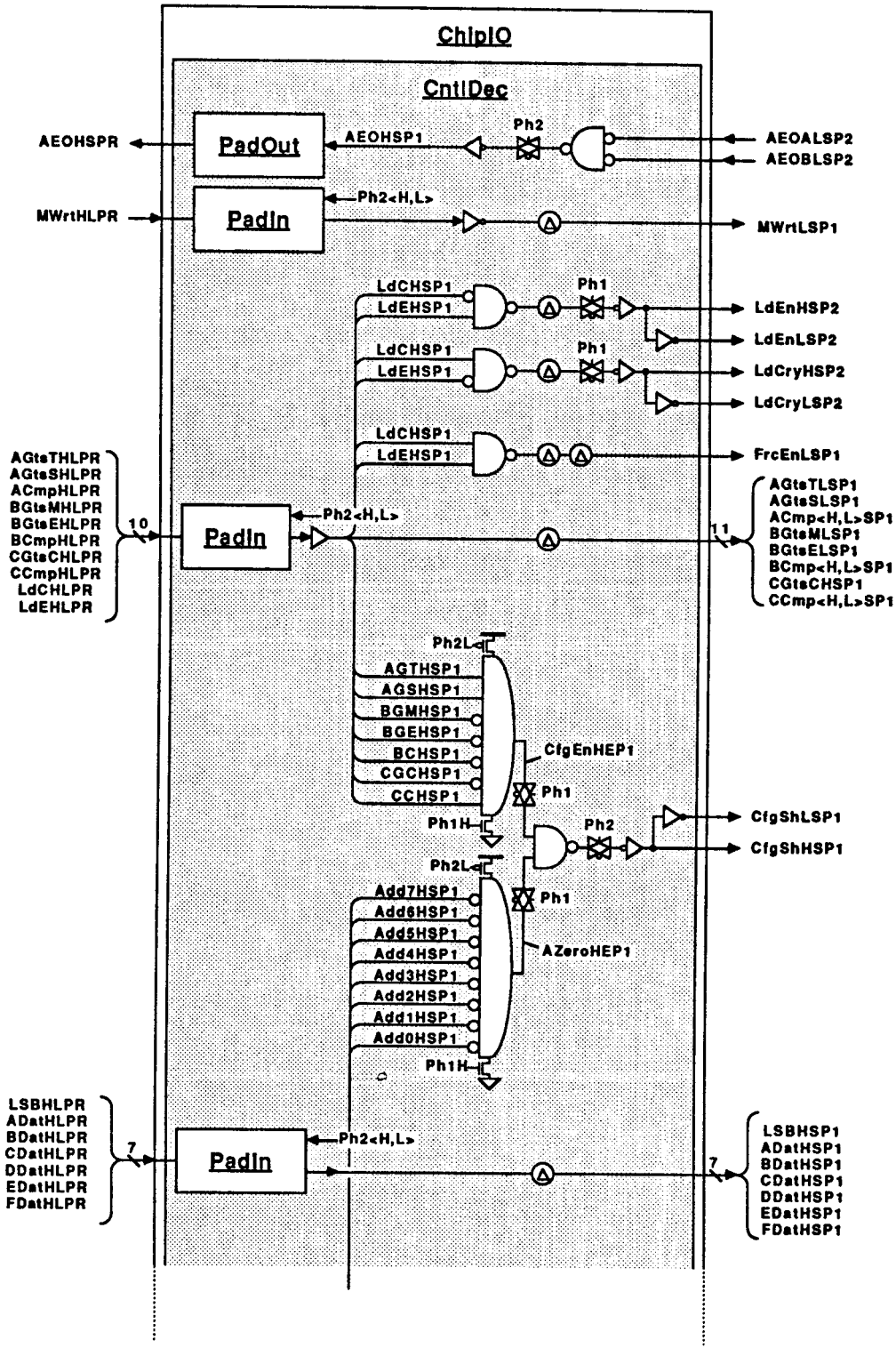



**Figure 2.2.2 Block diagram detailing upper part of CntlDec sub-module of ChipIO module.**

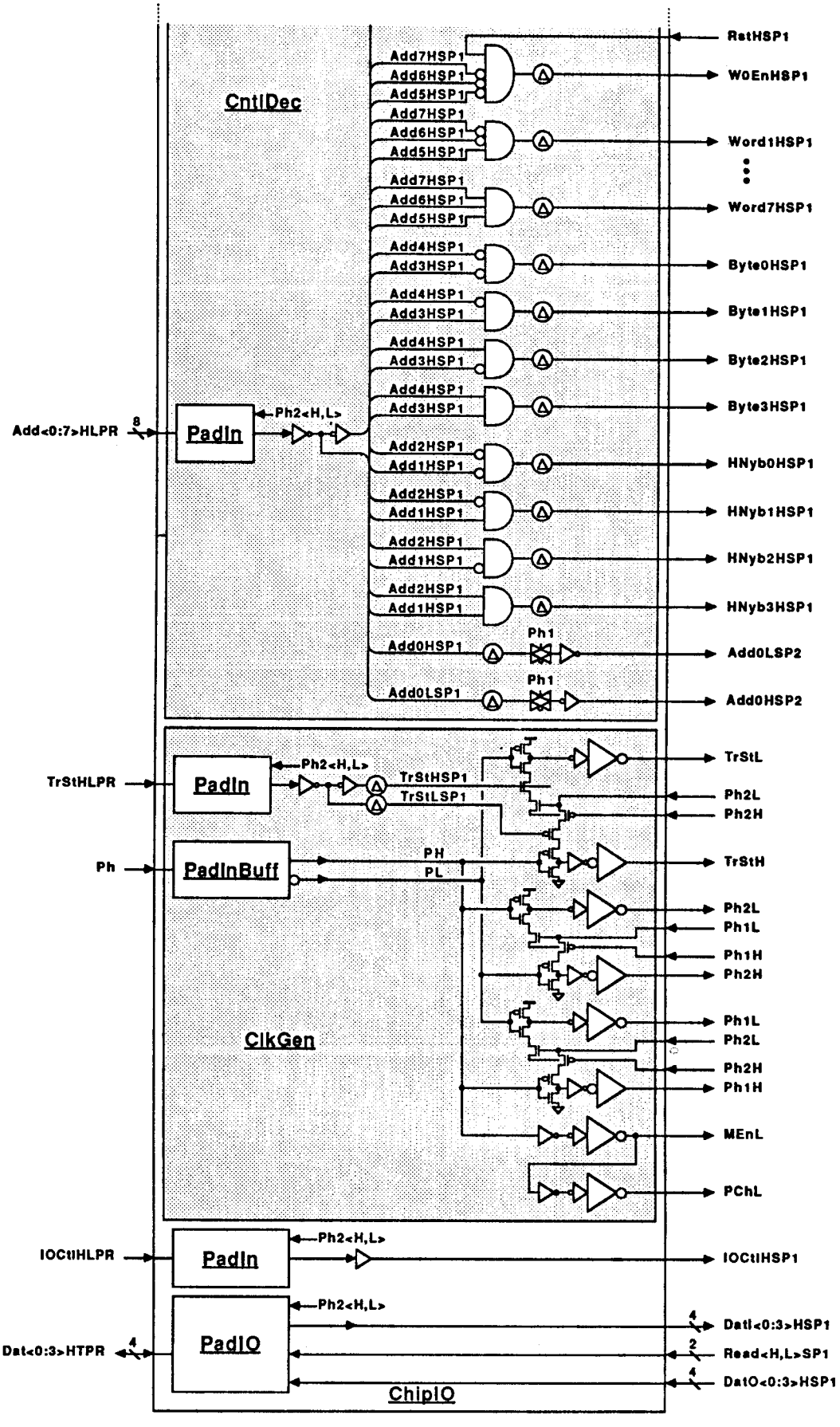
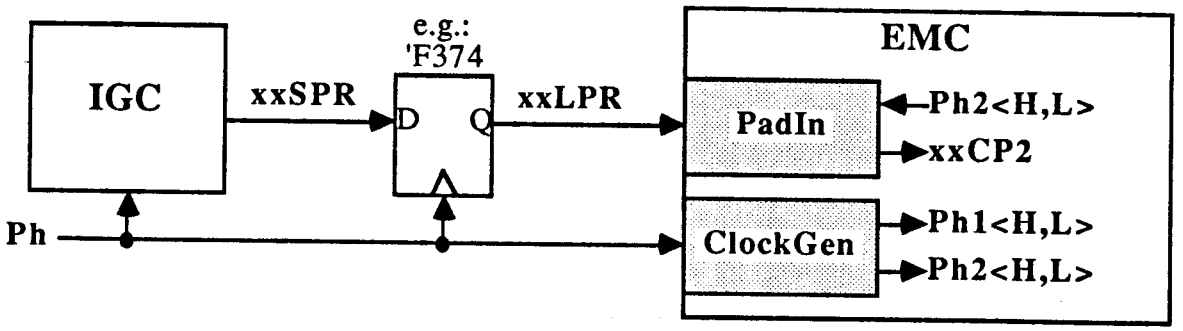



**Figure 2.2.3 Block diagram of lower part of ChipIO module detailing lower part of CntlDec and ClkGen sub-modules.**

$t_{sTL} = t_s - t_{p2}$ ; $t_{hTL} = t_h + t_{p2}$

The expected means of generating LPR signals (from the IGC) is:

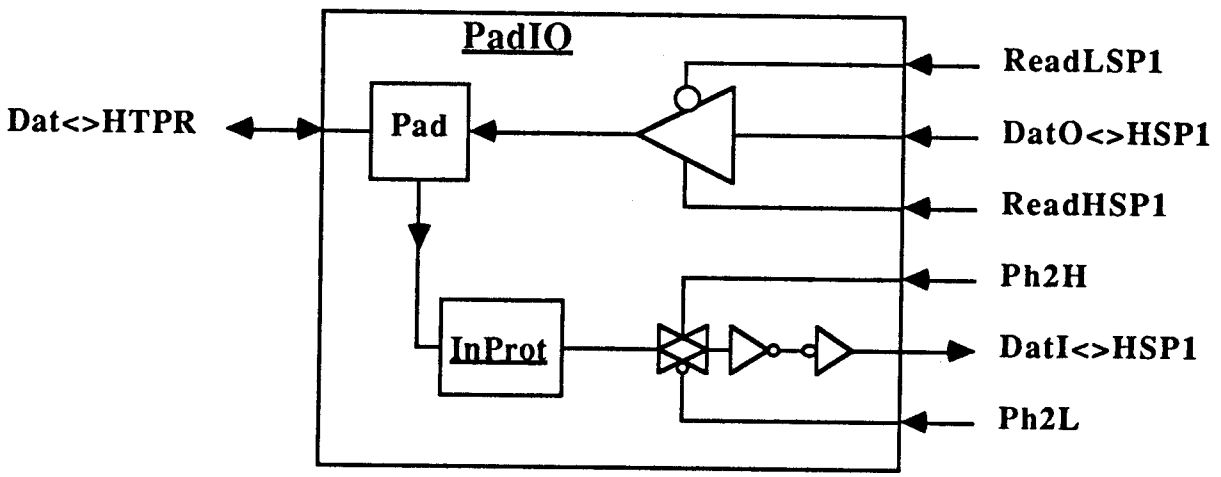



From the system designer's point of view, the timing constraint for setup of the LPR signal type is

$$MAX(t_{pHL}, t_{pLH} \text{ for 'F374}) < t_{pH} - t_{sTL}$$

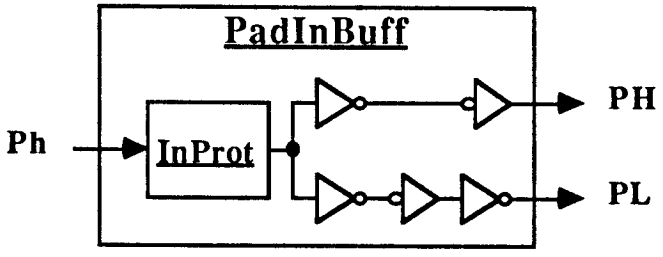
The hold-time constraint will typically be met trivially.

**PadIO** is a full bi-directional pad that includes input circuitry similar to PadIn. The output buffer is of the tri-state variety, which, when disabled, allows the Pad node to float to Hi-Z.




**PadOut** is a simple high-fanout buffer.

**PadInBuff**, which is used to bring the external clock **Ph** into the ClockGen sub-module, is an input buffer producing true and complement signals with equal delays.



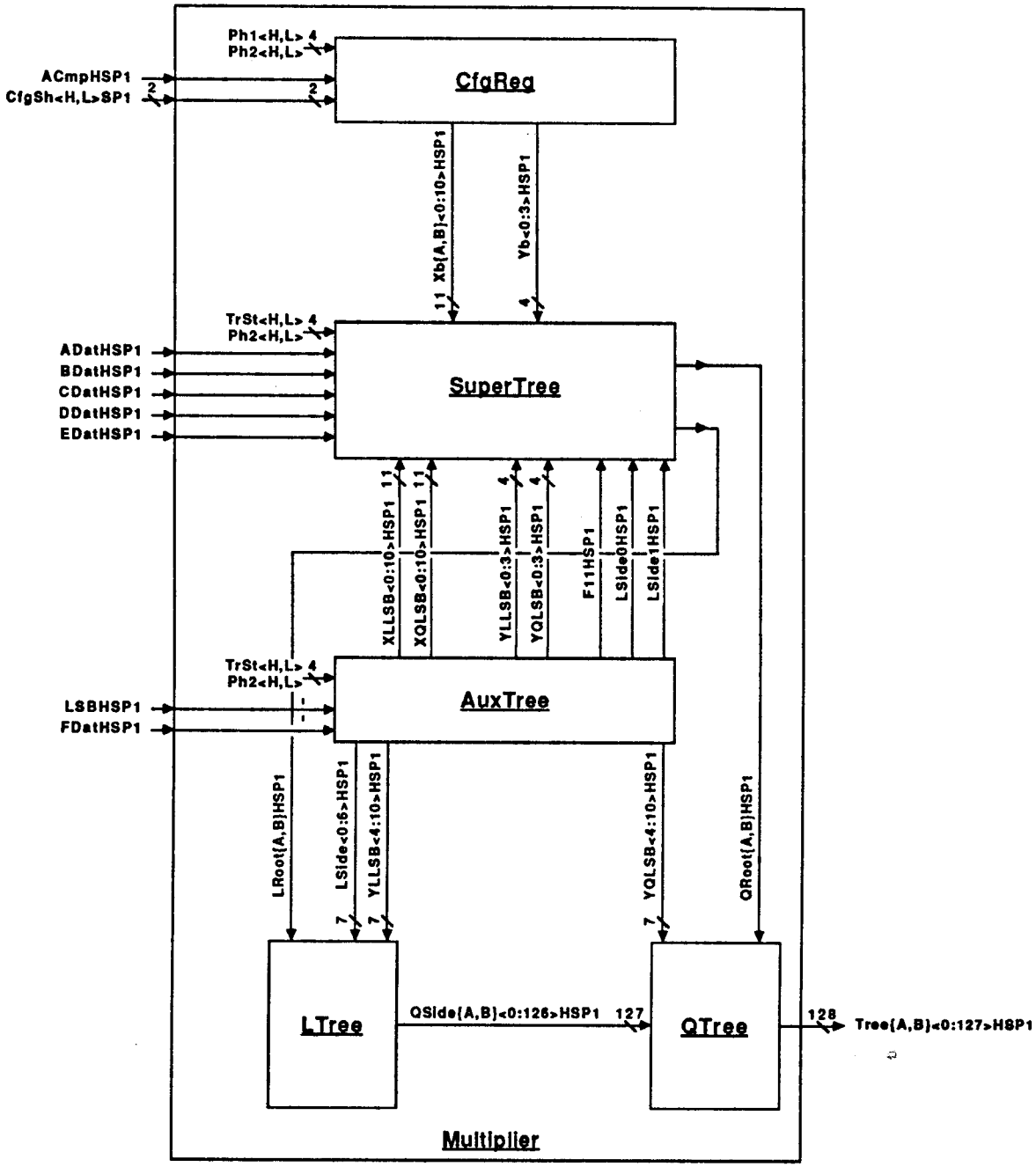



**Figure 2.2.4 Block diagram of Multiplier sub-module of Plane.**

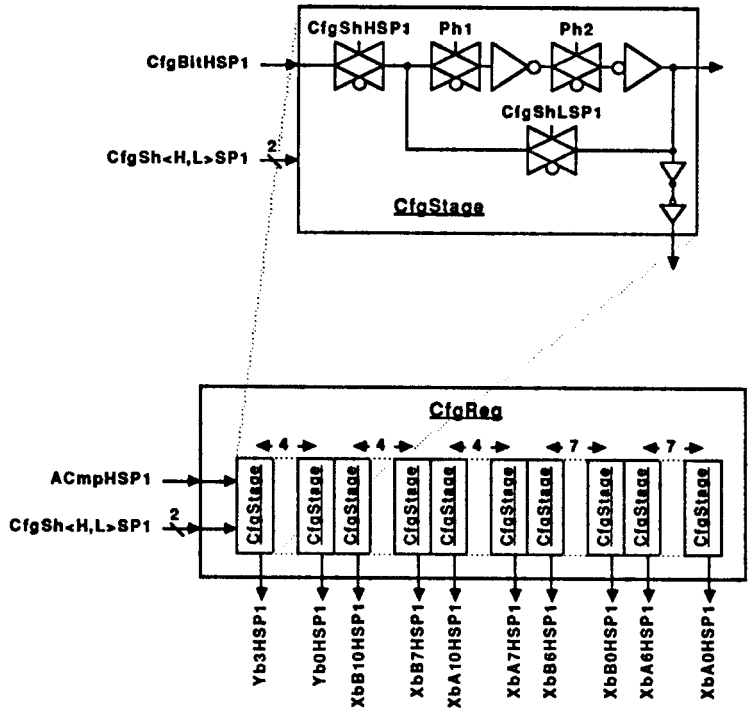



**Figure 2.2.5 Block diagram of CfgReg sub-module of Multiplier.**

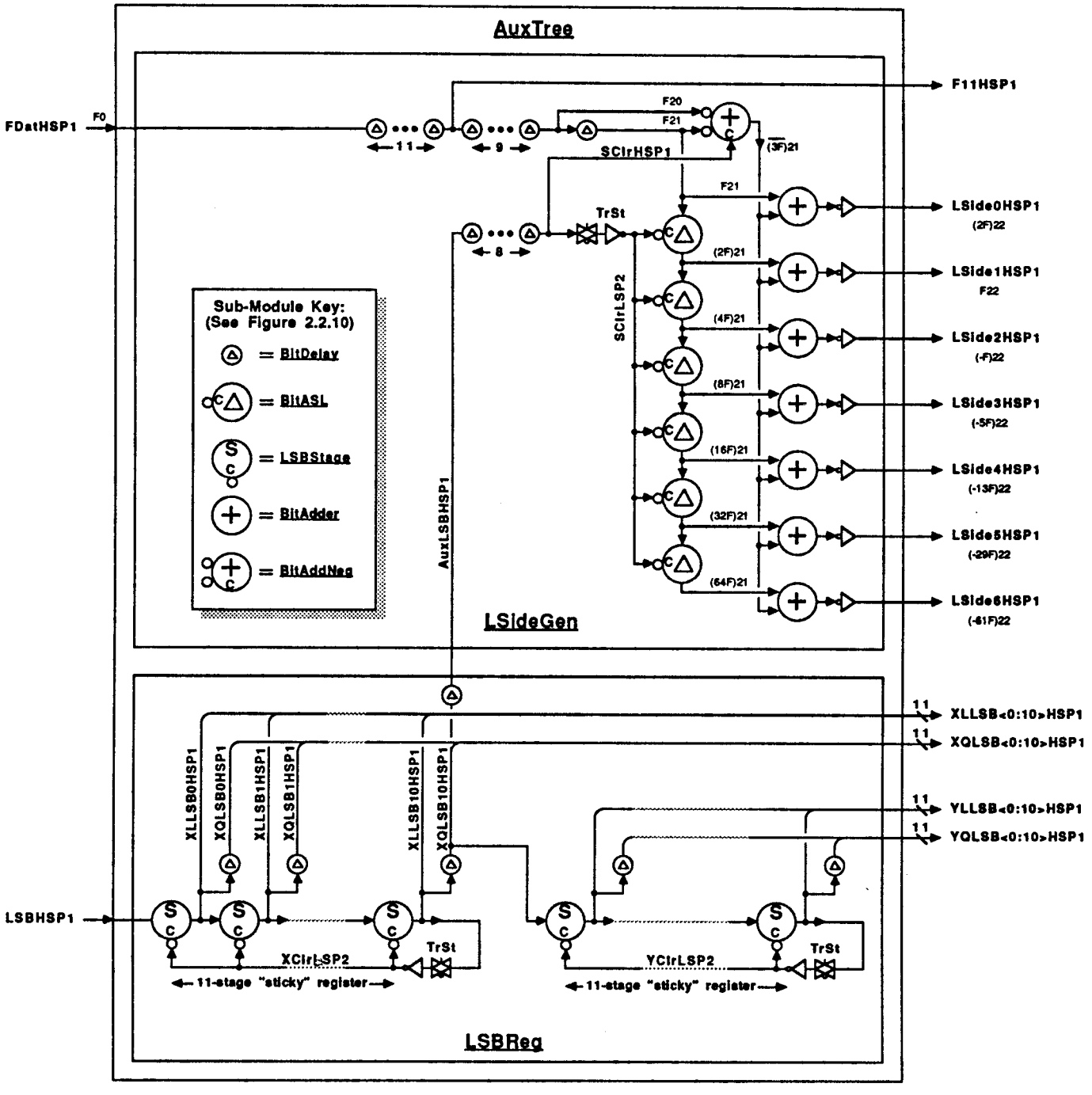



**Figure 2.2.6 Block diagram of AuxTree submodule of Multiplier, showing details of its LSideGen and LSBReg sub-modules.**

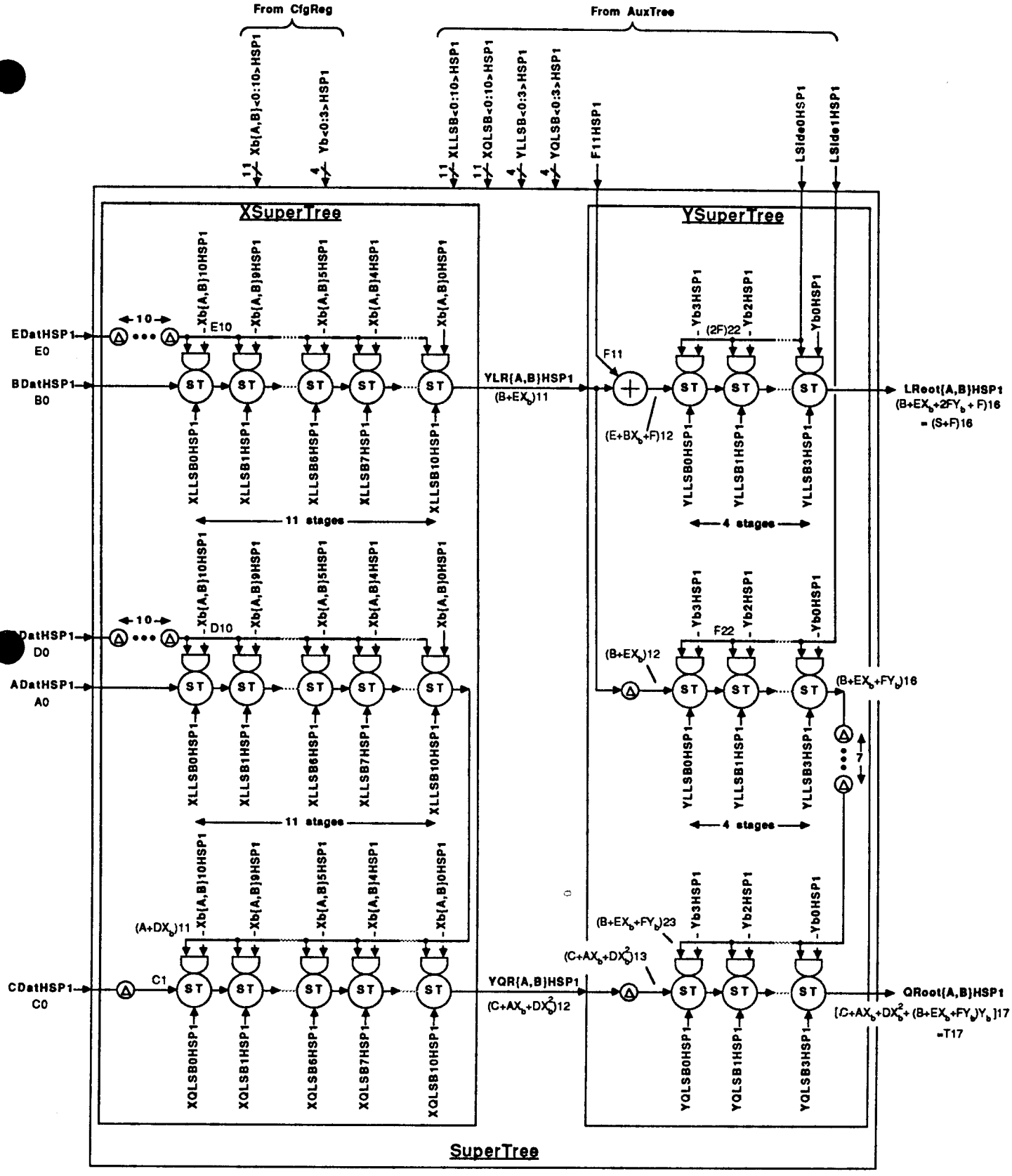



**Figure 2.2.7 Block diagram of SuperTree sub-module of Multiplier showing details of XSuperTree and YSuperTree.**

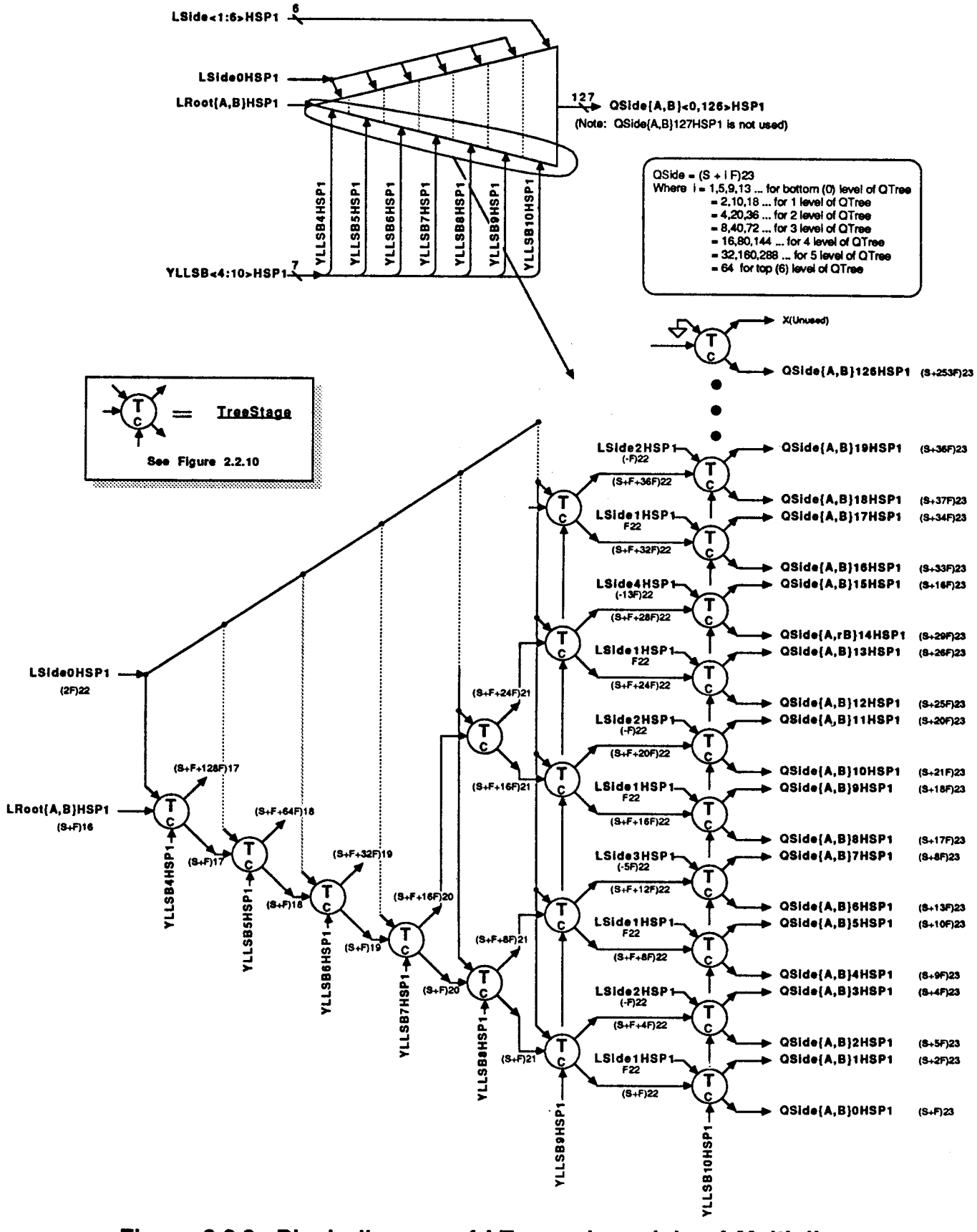



**Figure 2.2.8 Block diagram of LTree sub-module of Multiplier.**

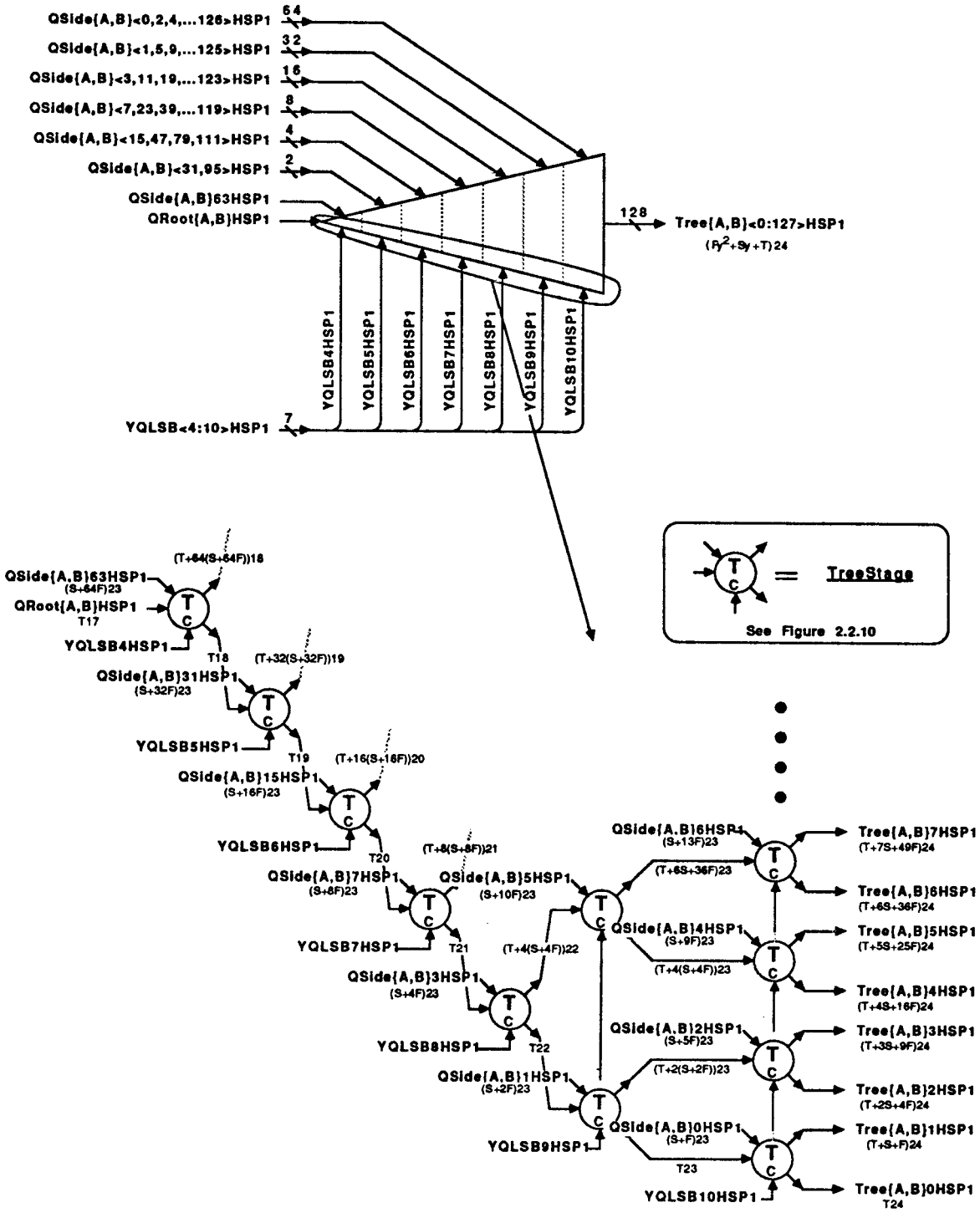



**Figure 2.2.9 Block diagram of QTree sub-module of Multiplier.**

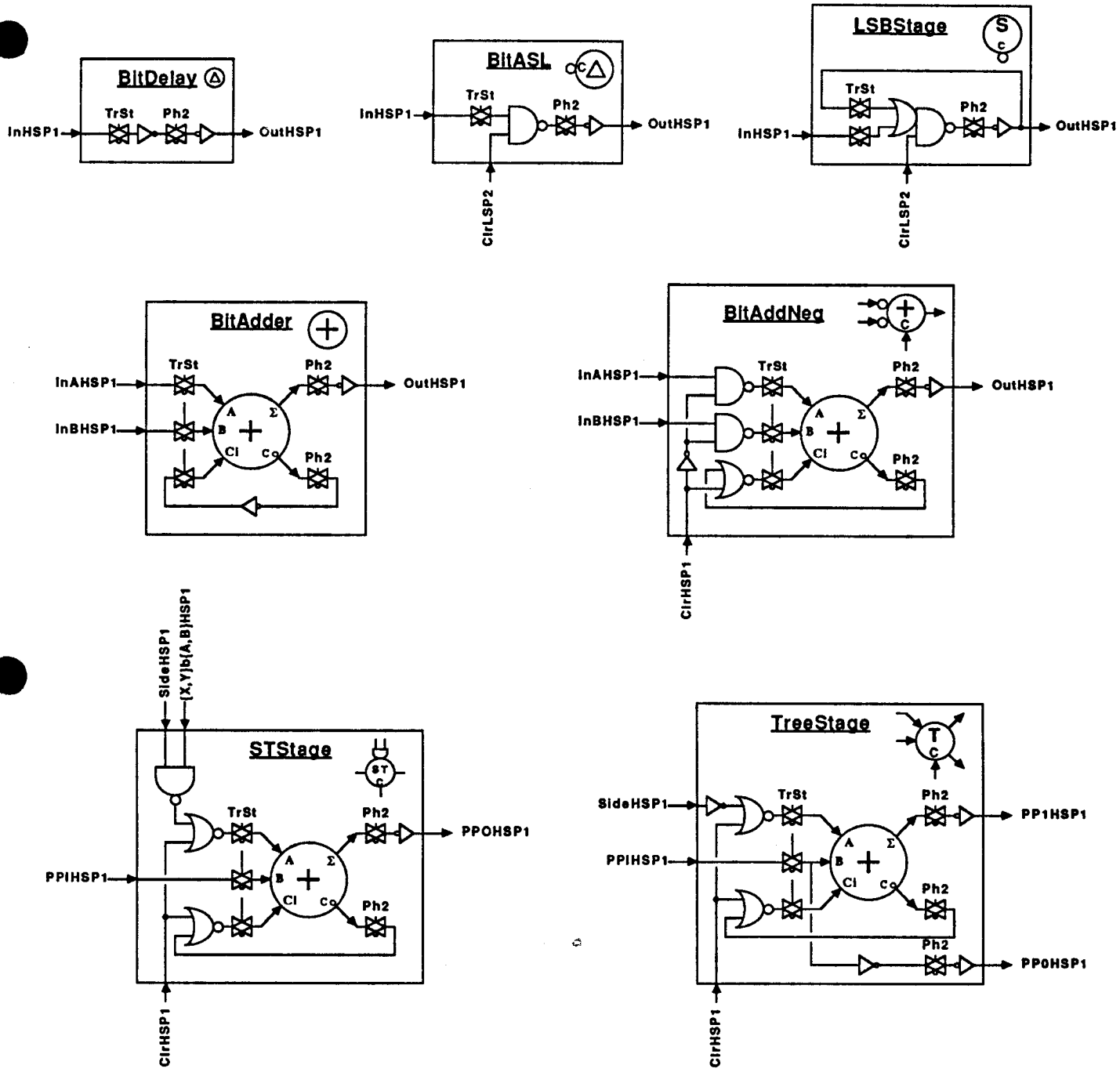



**Figure 2.2.10 Logic for Multiplier's leaf cells.**

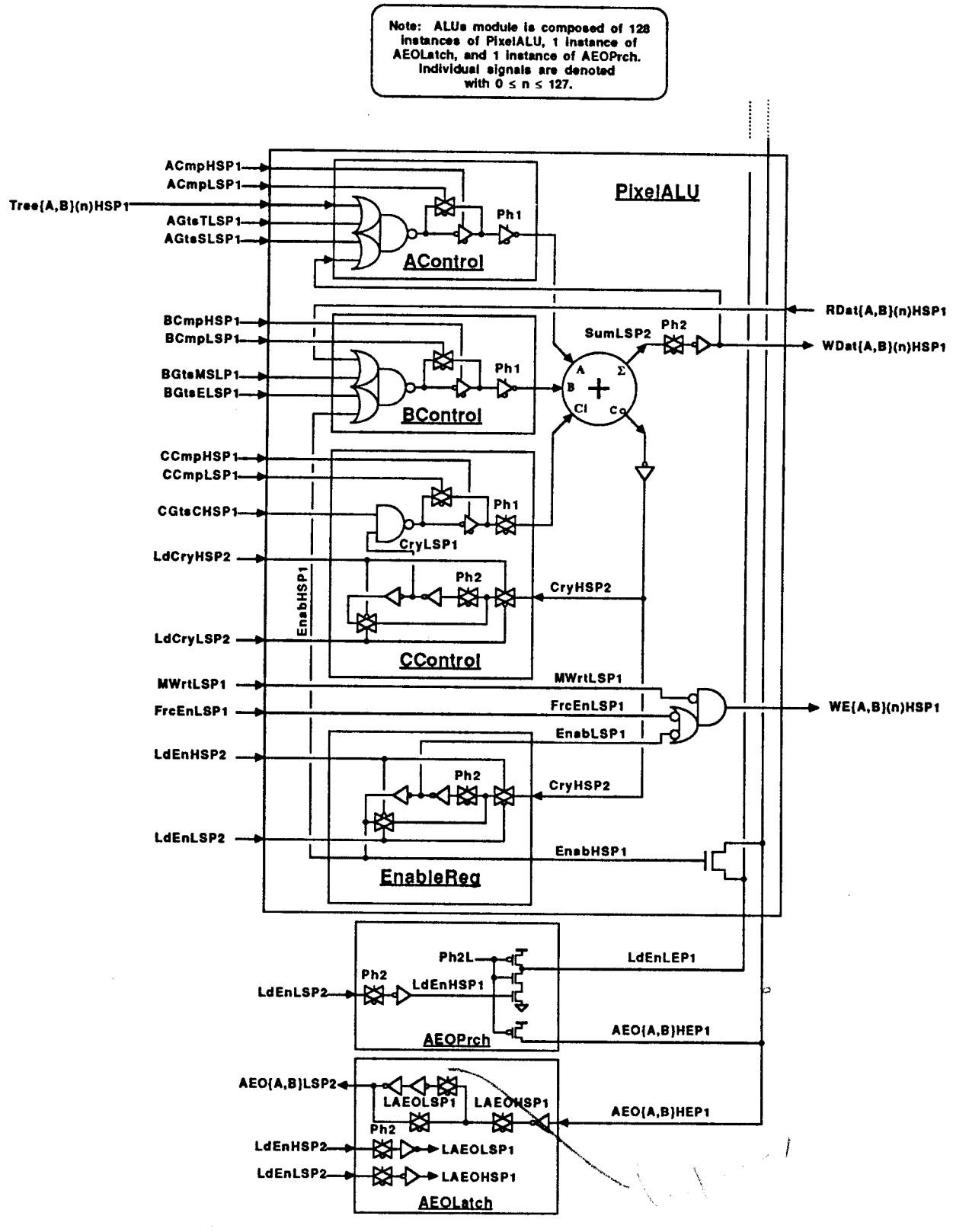



**Figure 2.2.11 Block diagram of PixelALU and AEOLatch sub-modules of ALUs.**

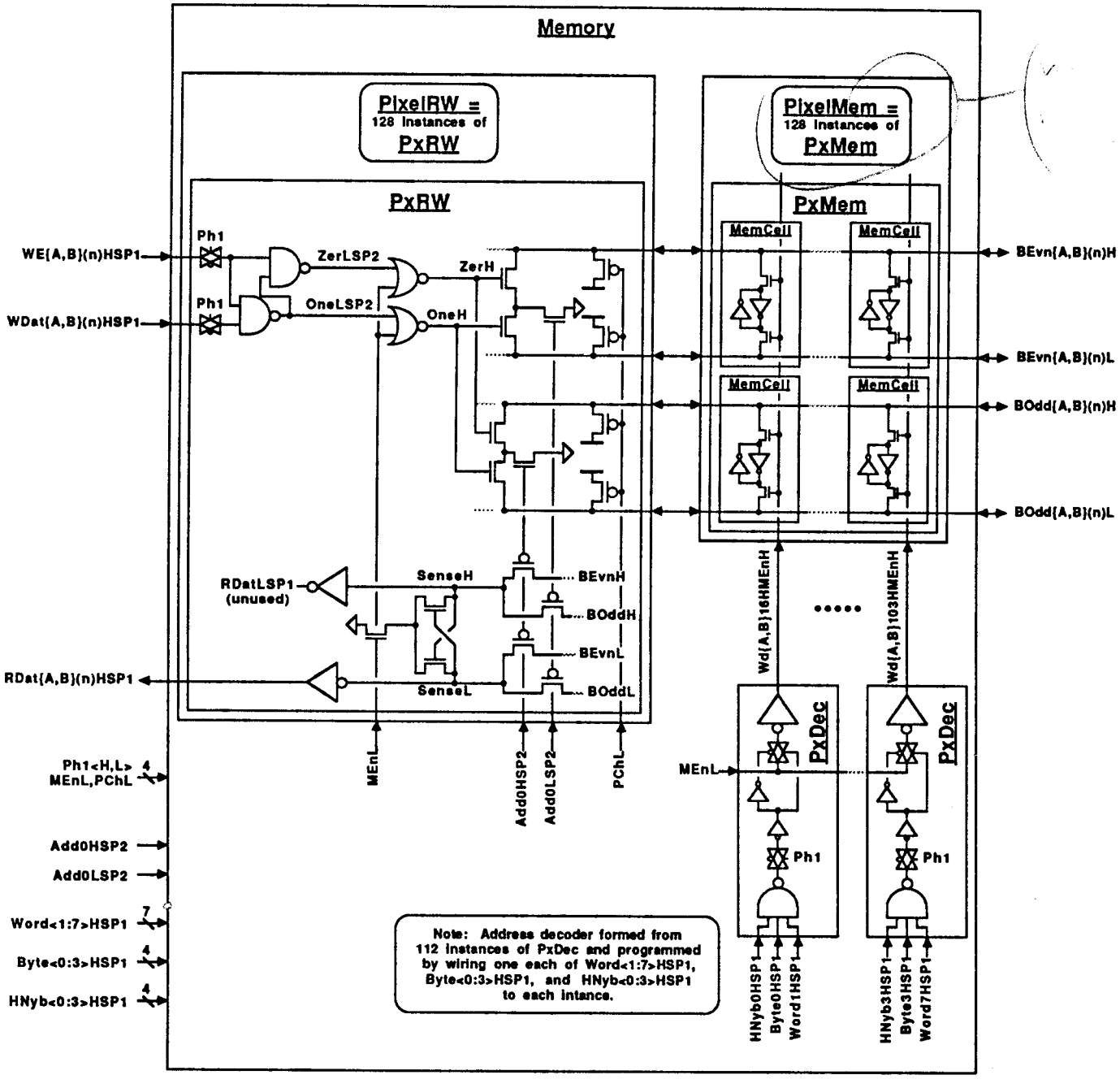
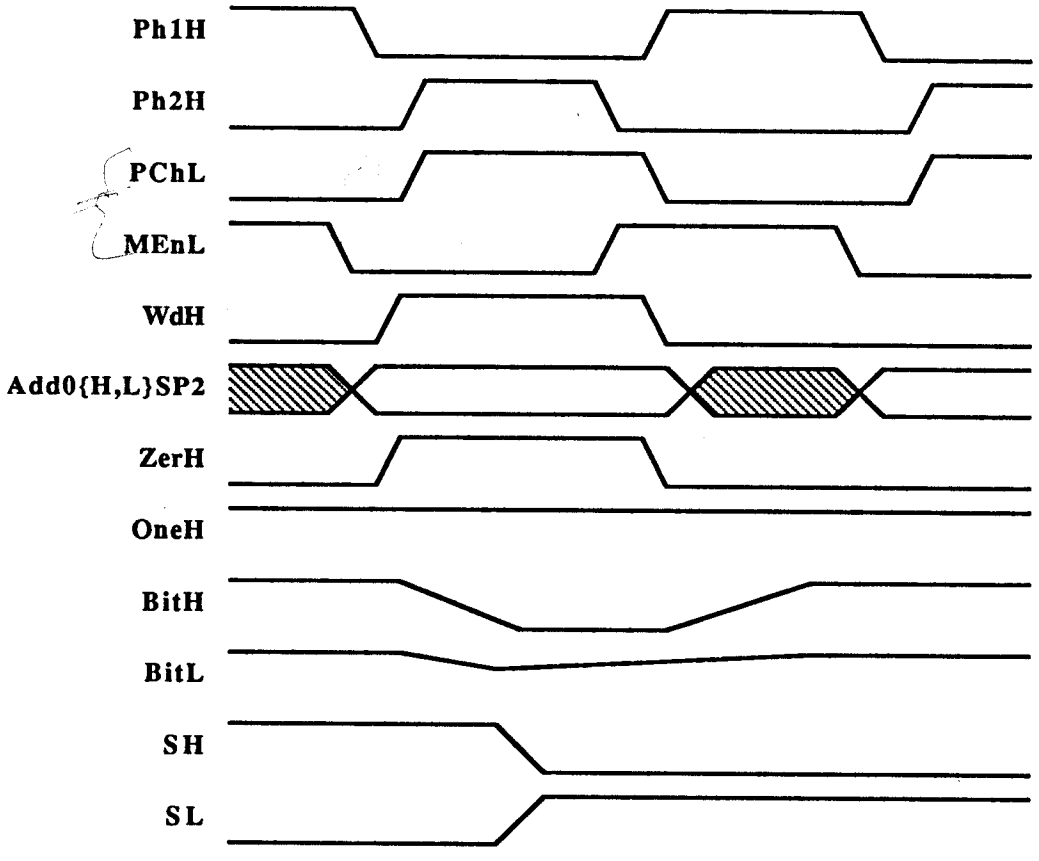



**Figure 2.2.12 Block diagram of Memory showing details of PixelRW (128 instances of PxRW, 0≤n≤127), PixelMem (128 instances of PxMem), and PxDec sub-modules of Memory.**

### 2.3.3.4 Memory Write Timing

The timing for memory Writes is straightforward:




This example assumes that a "0" was latched on **WDatHSP1** during the **Ph1**, and that **WEHSP1** was HI during that phase. **MEnL** going LO leads to **ZerH** being asserted, which pulls down one of **BEvnH** or **BOddH**, as selected by **Add0{H,L}SP2**. **WdH** is asserted simultaneously, selecting one of the memory cells. In this example, the cell was previously storing a "1", and it begins to pull down **BitL**, but this read is swamped by the rapidly falling **BitH**, and the cell's contents are flipped. Note that **RDatHSP1** will produce the value written during the next **Ph1**.

```
#include "emcs.h"
#include "lgcfuns.h"
#include "CommReg.h"

/*****************************************************************************/

CommRegtype (p)
struct CommRegtype *p;
{
    int i;

    /* connect input to IOPLA sub-module */
    p->IOPLA.IOCtlHSP1 = p->IOCtlHSP1;

    /* exercise IOPLA sub-module (must be first sub-module exercised) */
    IOPLAtype (&p->IOPLA);

    /* connect inputs to PPointer */
    p->PPointer.RstHSP1    = p->IOPLA.RstHSP1;
    p->PPointer.CREnHSP1   = p->IOPLA.CREnHSP1;
    p->PPointer.PShftLSP1 = p->CRWord.PShftLSP1;

    /* exercise PPointer */
    PPointertype (&p->PPointer);

    /* connect inputs to CRWord sub-module (CRWordA and CRWordB) */
    p->CRWord.RstHSP1    = p->IOPLA.RstHSP1;
    p->CRWord.CREnHSP1   = p->IOPLA.CREnHSP1;
    p->CRWord.W0EnHSP1   = p->W0EnHSP1;
    for (i=0; i<4; i++) p->CRWord.ByteHSP1[i] = p->ByteHSP1[i];
    for (i=0; i<4; i++) p->CRWord.HNybHSP1[i] = p->HNybHSP1[i];

    /* exercise CRWord sub-module (must be called before CRMem) */
    CRWordtype (&p->CRWord);

    /* connect inputs to CRMem sub-module */
    p->CRMem.ReadHSP1   = p->IOPLA.ReadHSP1;
    p->CRMem.BJnHSP2    = p->CRWord.BJnHSP2;
    p->CRMem.CREnHSP1   = p->IOPLA.CREnHSP1;
    p->CRMem.DIEvnHSP1 = p->IOMux.DIEvnHSP1;
    p->CRMem.DIOddHSP1 = p->IOMux.DIOddHSP1;
    for (i=0; i<16; i++) p->CRMem.WdAH[i] = p->CRWord.WdAH[i];
    for (i=0; i<16; i++) p->CRMem.WdBH[i] = p->CRWord.WdBH[i];
    for (i=0; i<TSIZE; i++) p->CRMem.PSHSP2[i] = p->PPointer.PSHSP2[i];

    /* connect bit-lines to CRMem sub-module */
    for (i=0; i<TSIZE; i++) p->CRMem.BEvnAHEP2[i] = p->BEvnAHEP2[i];
    for (i=0; i<TSIZE; i++) p->CRMem.BEvnALEP2[i] = p->BEvnALEP2[i];
    for (i=0; i<TSIZE; i++) p->CRMem.BOddAHEP2[i] = p->BOddAHEP2[i];
    for (i=0; i<TSIZE; i++) p->CRMem.BOddALEP2[i] = p->BOddALEP2[i];
    for (i=0; i<TSIZE; i++) p->CRMem.BEvnBHEP2[i] = p->BEvnBHEP2[i];
    for (i=0; i<TSIZE; i++) p->CRMem.BEvnBLEP2[i] = p->BEvnBLEP2[i];
    for (i=0; i<TSIZE; i++) p->CRMem.BOddBHEP2[i] = p->BOddBHEP2[i];
    for (i=0; i<TSIZE; i++) p->CRMem.BOddBLEP2[i] = p->BOddBLEP2[i];

    CRMemtype (&p->CRMem);

    /* connect bit-lines from CRMem sub-module to CommReg */
    for (i=0; i<TSIZE; i++) p->BEvnAHEP2[i] = p->CRMem.BEvnAHEP2[i];
    for (i=0; i<TSIZE; i++) p->BEvnALEP2[i] = p->CRMem.BEvnALEP2[i];
    for (i=0; i<TSIZE; i++) p->BOddAHEP2[i] = p->CRMem.BOddAHEP2[i];
    for (i=0; i<TSIZE; i++) p->BOddALEP2[i] = p->CRMem.BOddALEP2[i];
    for (i=0; i<TSIZE; i++) p->BEvnBHEP2[i] = p->CRMem.BEvnBHEP2[i];
    for (i=0; i<TSIZE; i++) p->BEvnBLEP2[i] = p->CRMem.BEvnBLEP2[i];
```

```
    /* connect inputs to IOMux sub-module */
    p->IOMux.IOReqHSP1 = p->IOPLA.IOReqHSP1;
    p->IOMux.DOEvnHSP1 = p->CRMem.DOEvnHSP1;
    p->IOMux.DOOddHSP1 = p->CRMem.DOOddHSP1;
    for (i=0; i<4; i++) p->IOMux.DatIHSP1[i] = p->DatIHSP1[i];

    /* exercise IOMux sub-module */
    IOMuxtype (&p->IOMux);

    /* connect outputs (to tri-state pads in ChipIO module) */
    p->RstHSP1  = p->IOPLA.RstHSP1;
    p->ReadHSP1 = p->IOPLA.ReadHSP1;
    for (i=0; i<4; i++) p->DatOHSP1[i] = p->IOMux.DatOHSP1[i];


}

/*****************************************************************************/

IOPLAtype (p)
struct IOPLAtype *p;
{
    int i;

    if (Ph1) {

        p->ICLSP2 = INV (p->IOCtlHSP1);
        p->ICHSP2 = INV (p->ICLSP2);
        p->S0LSP2 = INV (p->S0HSP1);
        p->S0HSP2 = INV (p->S0LSP2);
        p->S1LSP2 = INV (p->S1HSP1);
        p->S1HSP2 = INV (p->S1LSP2);
        p->S2LSP2 = INV (p->S2HSP1);
        p->S2HSP2 = INV (p->S2LSP2);
        p->S3LSP2 = INV (p->S3HSP1);
        p->S3HSP2 = INV (p->S3LSP2);

        p->RdDelLSP2[0] = INV (p->S2HSP1);
        for (i=1; i<4; i++) p->RdDelLSP2[i] = INV (p->RdDelHSP1[i-1]);

        p->CREnLSP2 = INV (p->CREHSP1);

        p->RstLSP2 = INV (INV (p->S3HSP1));

    }

    if (Ph2) {

        p->S0HSP1 =
            p->S3HSP2 && p->S0LSP2 && p->ICHSP2 ||
            p->S3HSP2 && p->S1LSP2 && p->ICHSP2 ||
            p->S3LSP2 && p->S2LSP2 && p->S1LSP2 && p->S0LSP2 && p->ICLSP2
        ;

        p->S1HSP1 =
           p->S3HSP2 && p->S1LSP2 && p->S0HSP2
        ;

        p->S2HSP1 =
            p->S3LSP2 && p->S2LSP2 && p->S1LSP2 && p->S0HSP2 && p->ICLSP2 ||
            p->S3HSP2 && p->S2HSP2 && p->S1LSP2                           ||
            p->S3HSP2 && p->S2HSP2 && p->S0LSP2
        ;
```

```
        p->S2LSP2 && p->S1LSP2 && p->S0HSP2 ||
        p->S3HSP2 && p->S0LSP2              ||
        p->S3HSP2 && p->S1LSP2
    ;

    p->IOReqHSP1=
        p->S3HSP2 && p->S0LSP2 && p->ICHSP2
    ;

    p->CREHSP1=
        p->S3HSP2 && p->S0LSP2 && p->ICHSP2 ||
        p->S3HSP2 && p->S1LSP2 && p->S0HSP2
    ;

    p->RstHSP1 = INV (p->RstLSP2);

    for (i=0; i(4; i++) p->RdDelHSP1[i] = INV (p->RdDelLSP2[i]);
    p->ReadHSP1 = OR (p->S2HSP1, p->RdDelHSP1[3]);
    p->ReadLSP1 = INV (p->ReadHSP1);

    p->CREnHSP1 = INV (p->CREnLSP2);
  }
}

/*****************************************************************************/

IOMuxtype (p)
struct IOMuxtype *p;
{
    int i;

    p->ILHSP1 = p->IOReqHSP1;
    p->OLHSP1 = p->IOReqHSP1;

    if (Ph1) {
        p->IL01HSP2 = p->ILHSP1;
        p->IL23HSP2 = INV (p->ILDelLSP1);
        p->ODelLSP2 = INV (p->OLHSP1);
        p->OL01LSP2 = INV (p->ODelHSP1);
        p->OL23LSP2 = INV (p->OL01HSP1);
        p->OLHSP2 = p->OL23HSP1;

        /* compute -Ph1 qualified clocks */
        if (p->ILHSP1)   p->ILHPh1   = 1;
        if (p->OL01HSP1) p->OL01HPh1 = 1;
        if (p->OL23HSP1) p->OL23HPh1 = 1;

        /* bring -Ph2 qualified clcoks low */
        p->IL01HPh2 = p->IL23HPh2 = p->OLHPh2 = 0;
    }


    if (Ph2) {
        p->ILDelLSP1 = INV (p->IL01HSP2);
        p->ODelHSP1  = INV (p->ODelLSP2);
        p->OL01HSP1  = INV (p->OL01LSP2);
        p->OL23HSP1  = INV (p->OL23LSP2);

        /* compute -Ph2 qualified clocks */
        if (p->OLHSP2)   p->OLHPh2   = 1;
        if (p->IL01HSP2) p->IL01HPh2 = 1;
        if (p->IL23HSP2) p->IL23HPh2 = 1;

        /* bring -Ph1 qualified clcoks low */
```

```
    /* multiplex 4 input wires into half nibble data */
    if (p->ILHPh1)
        for (i=0; i<4; i++) p->DatILSP2[i] = INV (p->DatIHSP1[i]);
    if (p->IL01HPh2) {
        p->DIEvnHSP1 = INV (p->DatILSP2[0]);
        p->DIOddHSP1 = INV (p->DatILSP2[1]);
    }
    if (p->IL23HPh2) {
        p->DIEvnHSP1 = INV (p->DatILSP2[2]);
        p->DIOddHSP1 = INV (p->DatILSP2[3]);
    }

    /* de-multiplex half-nibble data into 4 output wires */
    if (p->OL01HPh1) {
        p->DatOLSP2[0] = INV (p->DOEvnHSP1);
        p->DatOLSP2[1] = INV (p->DOOddHSP1);
    }
    if (p->OL23HPh1) {
        p->DatOLSP2[2] = INV (p->DOEvnHSP1);
        p->DatOLSP2[3] = INV (p->DOOddHSP1);
    }
    if (p->OLHPh2)
        for (i=0; i<4; i++) p->DatOHSP1[i] = INV (p->DatOLSP2[i]);

}

/*****************************************************************************/

CRWordtype (p)
struct CRWordtype *p;
{
    int i;

    p->NSavHSP1 = NOR (p->RstHSP1, p->CREnHSP1);

    /* check to make sure shifter controls are reasonable */
    if (!ONE_OF(p->RstHSP1,p->CREnHSP1,p->NSavHSP1)) {
        fprintf (stderr, "Controls for NSelect are messed up, exiting\n");
        exit (-1);
    }

    if (Ph1) {

        p->BJnHSP2  = p->RstHSP1;

        /* pull word-lines down, since Ph2 (MEnL) is de-asserted */
        for (i=0; i<16; i++) p->WdAH[i] = p->WdBH[i] = 0;

        /* compute SP2 versions of word lines */
        for (i=0; i<16; i++) p->WdALSP2[i] = NOR (
            AND3 (p->W0EnHSP1, p->ByteHSP1[(i)>>2)&3], p->HNybHSP1[i&3]),
            AND (p->CREnHSP1, INV(p->NSALSP1[i])));
        for (i=0; i<16; i++) p->WdBLSP2[i] = NOR (
            AND3 (p->W0EnHSP1, p->ByteHSP1[(i)>>2)&3], p->HNybHSP1[i&3]),
            AND (p->CREnHSP1, INV(p->NSBLSP1[i])));

        /* exercise N0Stage in NSelectA */
        if (p->RstHSP1)  p->NSAHSP2[0] = INV (0);
        if (p->CREnHSP1) p->NSAHSP2[0] = INV (p->NSBLSP1[15]);
        if (p->NSavHSP1) p->NSAHSP2[0] = INV (p->NSALSP1[0]);

        /* exercise NStage's for NSelectA */
        for (i=1; i<16; i++) {
```

```
        if (p->NSavHSP1) p->NSAHSP2[i] = INV (p->NSALSP1[i]);
    }

    /* exercise NØStage in NSelectB */
    if (p->RstHSP1)  p->NSBHSP2[Ø] = INV (1);
    if (p->CREnHSP1) p->NSBHSP2[Ø] = INV (p->NSALSP1[15]);
    if (p->NSavHSP1) p->NSBHSP2[Ø] = INV (p->NSBLSP1[Ø]);

    /* exercise NStage's for NSelectB */
    for (i=1; i<16; i++) {
        if (p->RstHSP1)  p->NSBHSP2[i] = INV (1);
        if (p->CREnHSP1) p->NSBHSP2[i] = INV (p->NSBLSP1[i-1]);
        if (p->NSavHSP1) p->NSBHSP2[i] = INV (p->NSBLSP1[i]);
    }

    /* exercise delay for NSB15LSP1 */
    if (p->RstHSP1)  p->PShftHSP2 = INV (1);
    if (p->CREnHSP1) p->PShftHSP2 = INV (p->NSBLSP1[15]);
    if (p->NSavHSP1) p->PShftHSP2 = INV (p->PShftLSP1);
}

if (Ph2) {

    /* exercise NSelect stages */
    for (i=Ø; i<16; i++) p->NSALSP1[i] = INV (p->NSAHSP2[i]);
    for (i=Ø; i<16; i++) p->NSBLSP1[i] = INV (p->NSBHSP2[i]);

    /* evaluate word lines */
    for (i=Ø; i<16; i++) {
        p->WdAH[i] = INV (p->WdALSP2[i]);
        p->WdBH[i] = INV (p->WdBLSP2[i]);
    }

    /* exercise delay for NSB15LSP1 */
    p->PShftLSP1 = INV (p->PShftHSP2);

}
}

/*************************************************************************/

CRMemtype (p)
struct CRMemtype *p;
{
    int i, j;                   /* gruff but lovable loop counters */

    if (Ph1) {

        /* precharge the secondary bit-lines */
        for (i=Ø; i<TSIZE; i++) {
            p->CEvnAHEP2[i] = p->CEvnALEP2[i] = 1;
            p->COddAHEP2[i] = p->COddALEP2[i] = 1;
            p->CEvnBHEP2[i] = p->CEvnBLEP2[i] = 1;
            p->COddBHEP2[i] = p->COddBLEP2[i] = 1;
        }

        /* precharge the transfer lines */
        p->DEvnHEP2 = 1;
        p->DOddHEP2 = 1;
        p->DEvnLEP2 = 1;
        p->DOddLEP2 = 1;

        /* latch input data lines */
        p->WEnHCP1 = !p->ReadHSP1 && p->CREnHSP1;
```

```
    p->DIOddLSP2 = NAND (p->WEnHCP1,p->DIOddHSP1);
    p->DIOddHSP2 = NAND (p->WEnHCP1,p->DIOddLSP2);

    p->CREnHSP2 = p->CREnHSP1;        /* used only for test below */
}


if (Ph2) {

    if (p->BJnHSP2 && p->CREnHSP2) {
        fprintf (stderr, "Hep me, CR op attempted on unified bit-lines\n");
        exit (-1);
    }

    if (! p->DIEvnHSP2) p->DEvnHEP2 = 0;
    if (! p->DIEvnLSP2) p->DEvnLEP2 = 0;
    if (! p->DIOddHSP2) p->DOddHEP2 = 0;
    if (! p->DIOddLSP2) p->DOddLEP2 = 0;

    if (p->BJnHSP2) {                 /* the bit-lines are unified ! */

        for (i=0; i<TSIZE; i++) {

            if (p->PSHSP2[i]) {
                fprintf (stderr, "Gracious, PS asserted when unified\n");
                exit (-1);
            }

            for (j=0; j<16; j++) {
                if (p->WdAH[j]) {
                    if (! p->BEvnAHEP2[i] || ! p->BEvnALEP2[i])
                        p->EvnBitA[i][j] = p->BEvnAHEP2[i];
                    if (! p->BOddAHEP2[i] || ! p->BOddALEP2[i])
                        p->OddBitA[i][j] = p->BOddAHEP2[i];
                    p->BEvnAHEP2[i] =   p->EvnBitA[i][j];
                    p->BEvnALEP2[i] = ! p->EvnBitA[i][j];
                    p->BOddAHEP2[i] =   p->OddBitA[i][j];
                    p->BOddALEP2[i] = ! p->OddBitA[i][j];
                }
                if (p->WdBH[j]) {
                    if (! p->BEvnBHEP2[i] || ! p->BEvnBLEP2[i])
                        p->EvnBitB[i][j] = p->BEvnBHEP2[i];
                    if (! p->BOddBHEP2[i] || ! p->BOddBLEP2[i])
                        p->OddBitB[i][j] = p->BOddBHEP2[i];
                    p->BEvnBHEP2[i] =   p->EvnBitB[i][j];
                    p->BEvnBLEP2[i] = ! p->EvnBitB[i][j];
                    p->BOddBHEP2[i] =   p->OddBitB[i][j];
                    p->BOddBLEP2[i] = ! p->OddBitB[i][j];
                }
            }
        }
    }


    else {                            /* we are doing backing store op */

        for (i=0; i<TSIZE; i++) {

            if (p->PSHSP2[i] == 0) continue;

            p->CEvnAHEP2[i] = p->CEvnBHEP2[i] = p->DEvnHEP2;
            p->CEvnALEP2[i] = p->CEvnBLEP2[i] = p->DEvnLEP2;
            p->COddAHEP2[i] = p->COddBHEP2[i] = p->DOddHEP2;
            p->COddALEP2[i] = p->COddBLEP2[i] = p->DOddLEP2;
```

```
                    if (p->WdAH[j]) {
                        if (! p->CEvnAHEP2[i] || ! p->CEvnALEP2[i]) {
                            p->EvnBitA[i][j] = p->CEvnAHEP2[i];
                        }
                        if (! p->COddAHEP2[i] || ! p->COddALEP2[i]) {
                            p->OddBitA[i][j] = p->COddAHEP2[i];
                        }
                        p->CEvnAHEP2[i] =   p->EvnBitA[i][j];
                        p->CEvnALEP2[i] = ! p->EvnBitA[i][j];
                        p->COddAHEP2[i] =   p->OddBitA[i][j];
                        p->COddALEP2[i] = ! p->OddBitA[i][j];
                    }
                    if (p->WdBH[j]) {
                        if (! p->CEvnBHEP2[i] || ! p->CEvnBLEP2[i]) {
                            p->EvnBitB[i][j] = p->CEvnBHEP2[i];
                        }
                        if (! p->COddBHEP2[i] || ! p->COddBLEP2[i]) {
                            p->OddBitB[i][j] = p->COddBHEP2[i];
                        }
                        p->CEvnBHEP2[i] =   p->EvnBitB[i][j];
                        p->CEvnBLEP2[i] = ! p->EvnBitB[i][j];
                        p->COddBHEP2[i] =   p->OddBitB[i][j];
                        p->COddBLEP2[i] = ! p->OddBitB[i][j];
                    }
                }

                if (!p->CEvnAHEP2[i] || !p->CEvnBHEP2[i]) p->DEvnHEP2 = 0;
                if (!p->CEvnALEP2[i] || !p->CEvnBLEP2[i]) p->DEvnLEP2 = 0;
                if (!p->COddAHEP2[i] || !p->COddBHEP2[i]) p->DOddHEP2 = 0;
                if (!p->COddALEP2[i] || !p->COddBLEP2[i]) p->DOddLEP2 = 0;

            }
        }


        /* connect outputs */
        p->DOEvnHSP1 = INV (p->DEvnLEP2);
        p->DOOddHSP1 = INV (p->DOddLEP2);

    }
}

/*****************************************************************************/

PPointertype (p)
struct PPointertype *p;
{
    int i;

    /* compute controls for pixel-pointer shift register, and check them */
    p->PIncHSP1  = AND (p->CREnHSP1, ! p->PShftLSP1);
    p->PSavHSP1  = NOR (p->PIncHSP1, p->RstHSP1);
    p->PRstHSP1  = p->RstHSP1;
    if (!ONE_OF(p->PIncHSP1,p->PSavHSP1,p->PRstHSP1)) {
        fprintf (stderr, "Controls for PPtr are messed up, exiting\n");
        exit (-1);
    }

    if (Phi) {

        /* exercise P0Stage */
        if (p->PRstHSP1) p->PSelLSP2[0] = INV (1);
        if (p->PIncHSP1) p->PSelLSP2[0] = INV (0);
        if (p->PSavHSP1) p->PSelLSP2[0] = INV (p->PSelHSP1[0]);
```

```
        for (i=1; i<TSIZE; i++) {
            if (p->PRstHSP1) p->PSelLSP2[i] = INV (0);
            if (p->PIncHSP1) p->PSelLSP2[i] = INV (p->PSelHSP1[i-1]);
            if (p->PSavHSP1) p->PSelLSP2[i] = INV (p->PSelHSP1[i]);
        }

        /* compute outputs */
        p->CREnLSP2 = INV (p->CREnHSP1);
        for (i=0; i<TSIZE; i++)
            p->PSHSP2[i] = NOR (p->CREnLSP2, p->PSelLSP2[i]);
    }


    if (Ph2) {
        for (i=0; i<TSIZE; i++) p->PSelHSP1[i] = INV (p->PSelLSP2[i]);
    }
}

/*****************************************************************************/
```

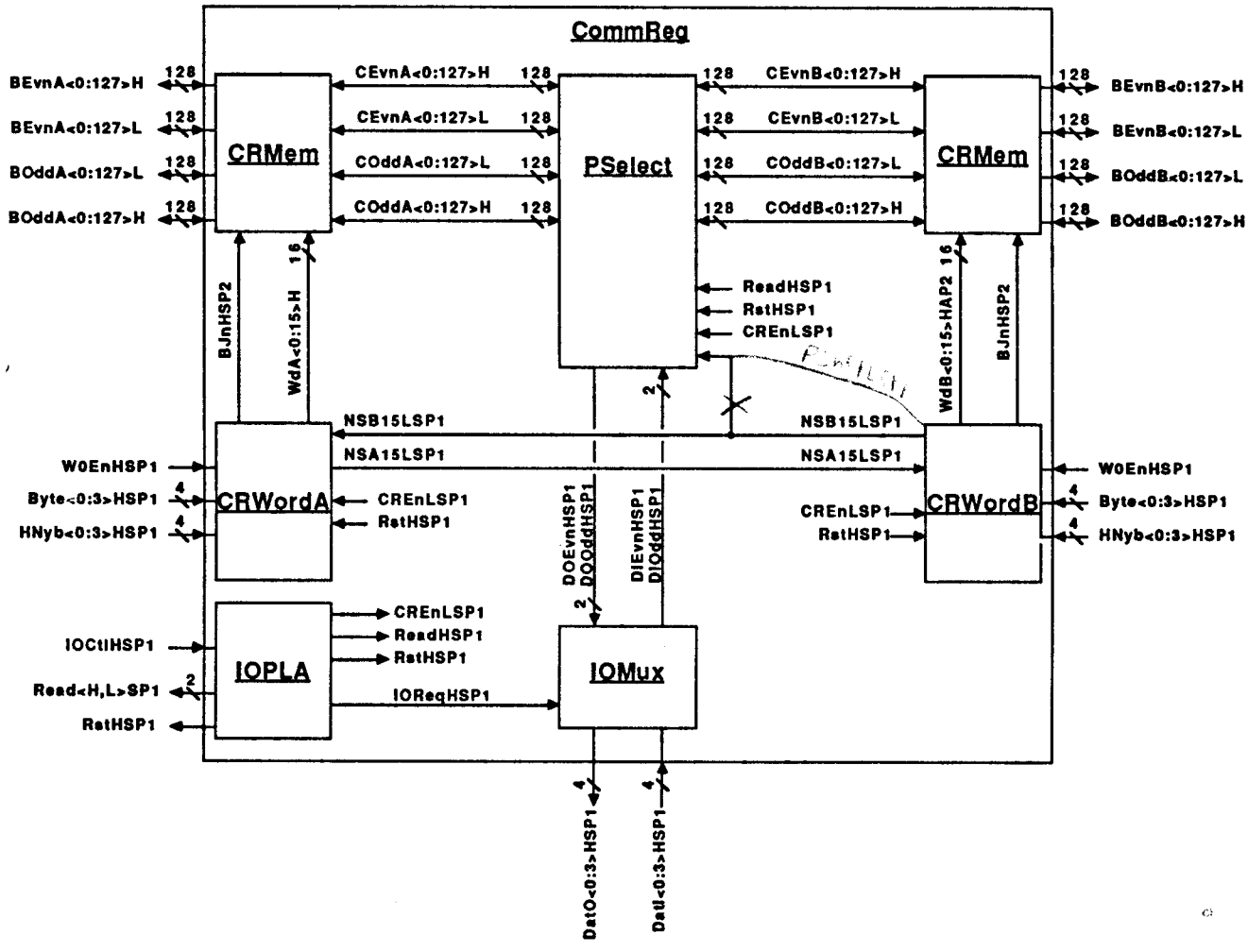



**Figure 2.2.13 Block diagram of CommReg module.**

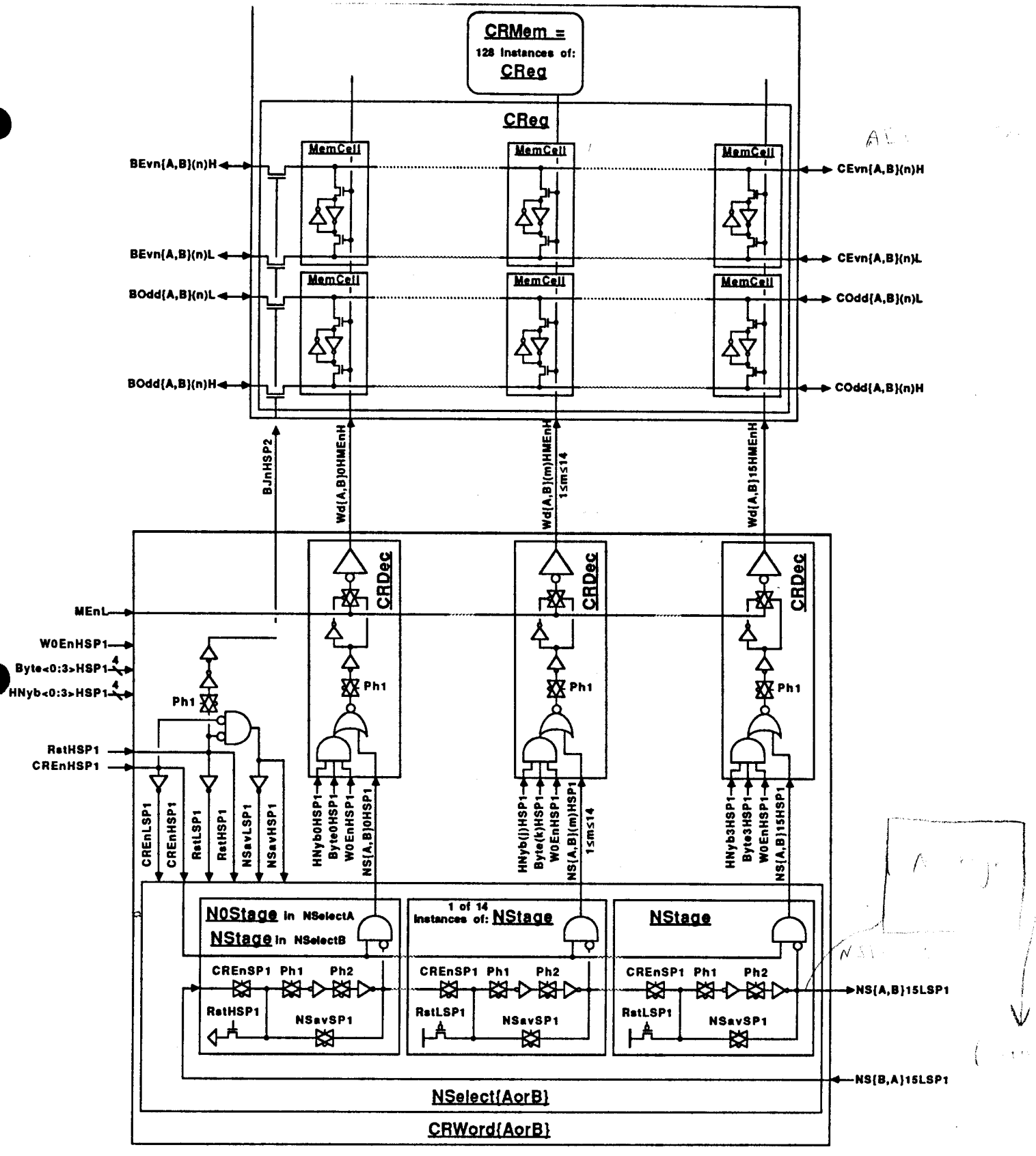



**Figure 2.2.14 Details of CRMem, CRWordA, and CRWordB sub-modules of CommReg, showing internal sub-modules of each. CRMem is composed of 128 instaces of CReg, 0≤n≤127. CRWord{A,B} differ only in NSelect{A,B}, which differ only in their first stages.**

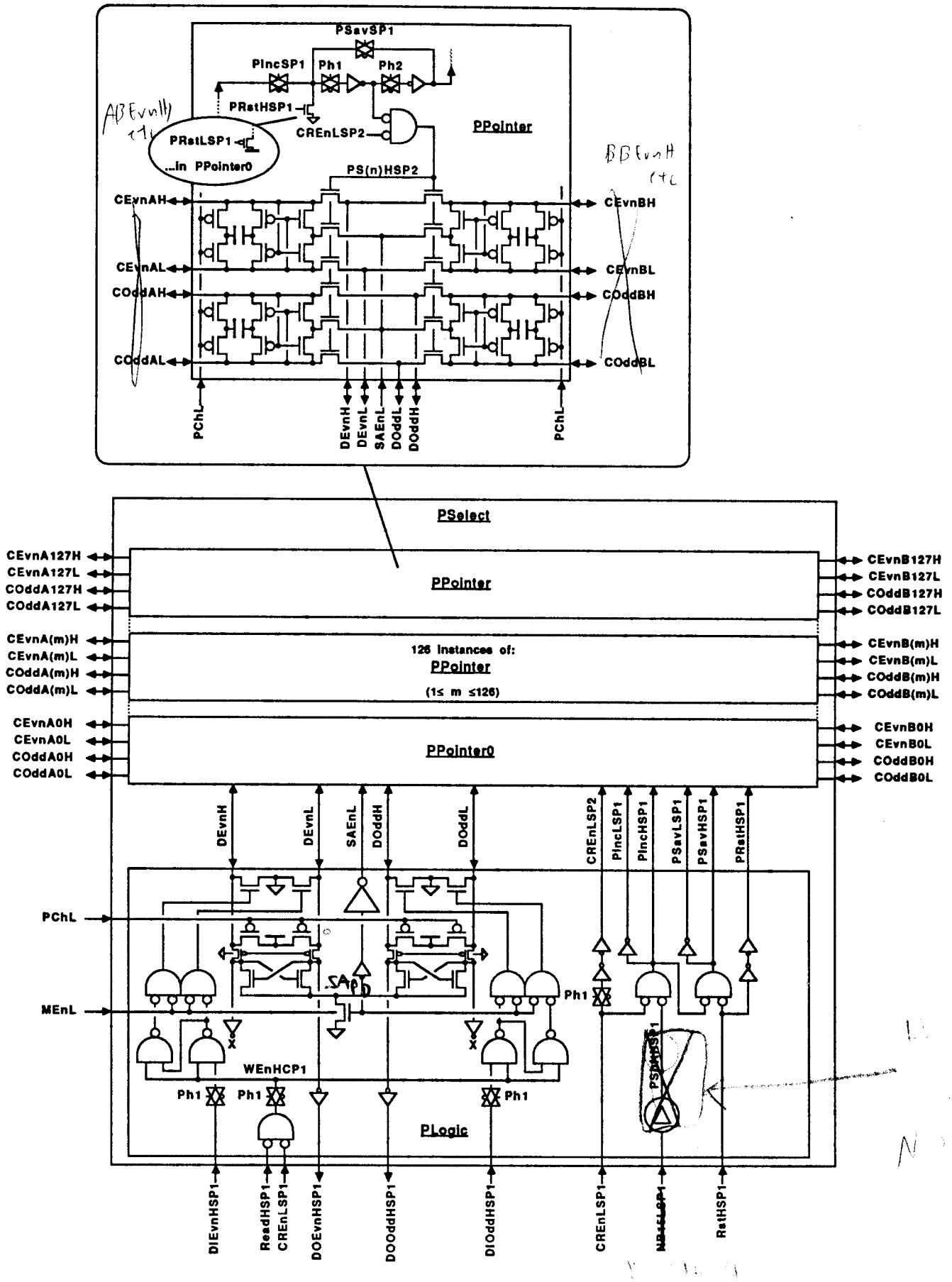



**Figure 2.2.15 Block diagram of PSelect sub-module of CommReg .**

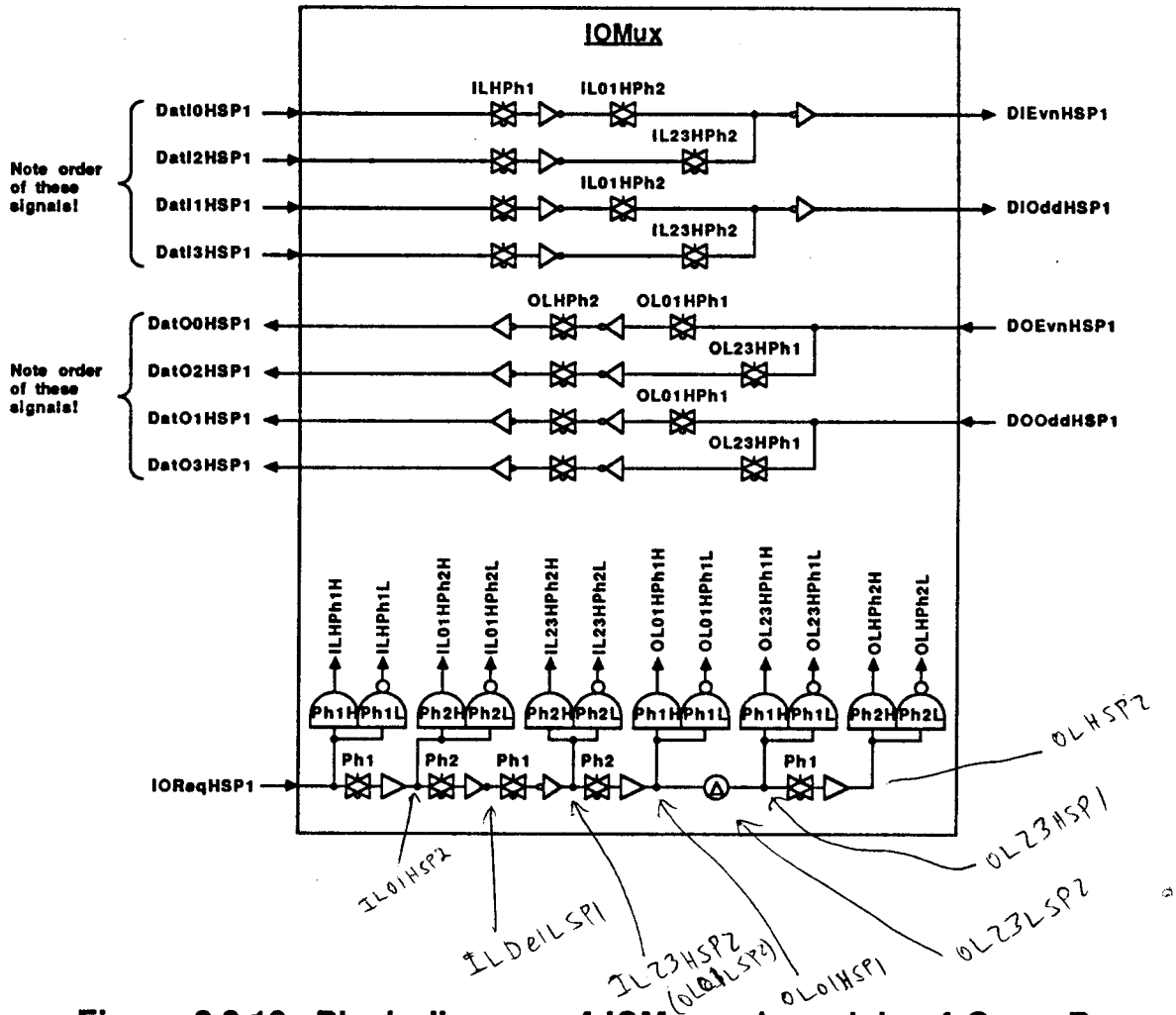



**Figure 2.2.16 Block diagram of IOMux sub-module of CommReg.**

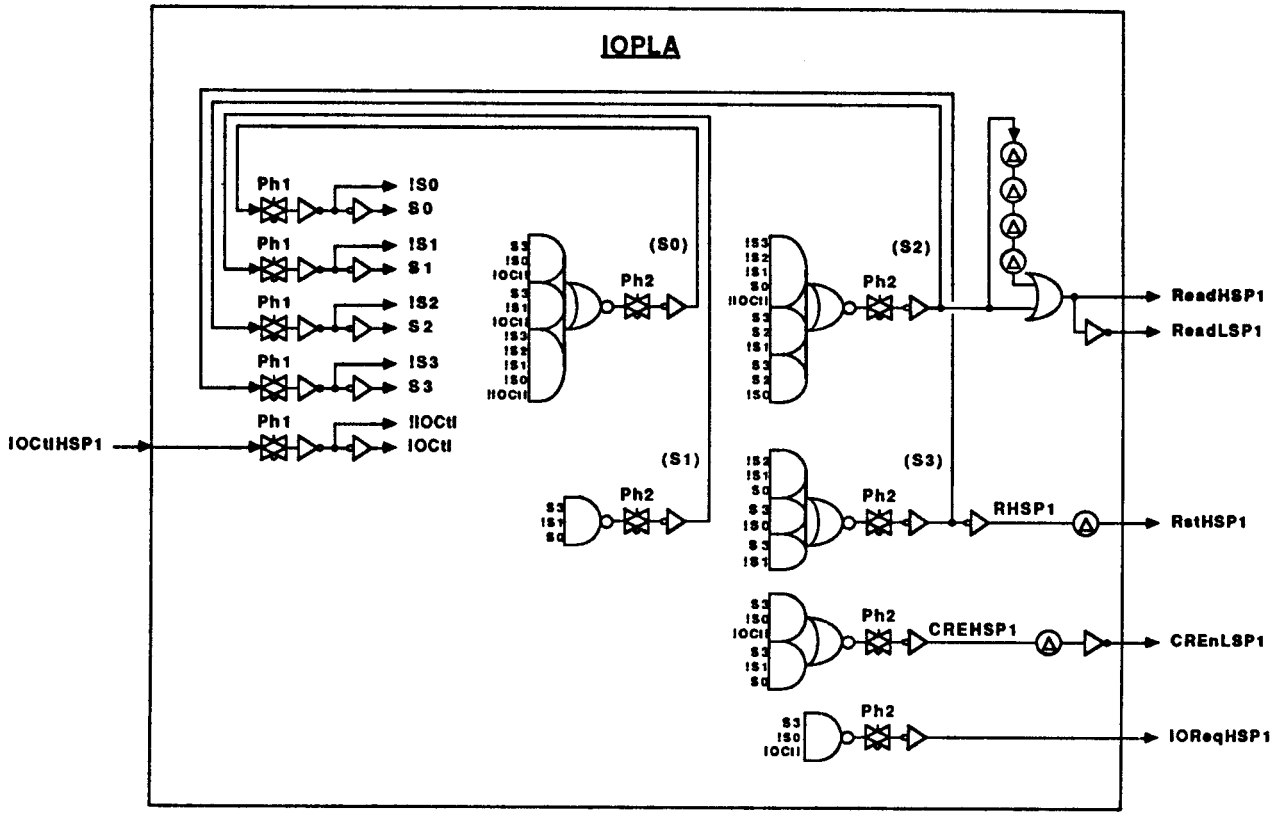
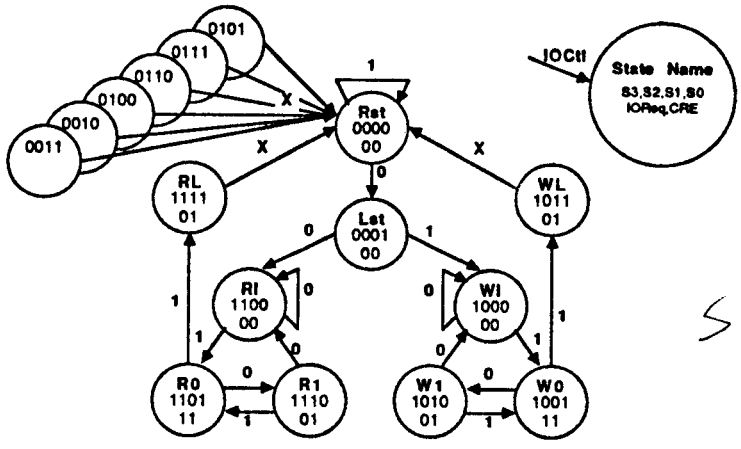



**Figure 2.2.17 Diagram of IOPLA sub-module of CommReg.**




## IOPLA State Tables and Logic Equations

next S3 (=!Rst)

| S3,S2,S1 \ S0, IOCtl | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 000 | | | 1 | 1 |
| 001 | | | | |
| 011 | | | | |
| 010 | | | | |
| 110 | 1 | 1 | 1 | 1 |
| 100 | 1 | 1 | 1 | 1 |
| 101 | 1 | 1 | | |
| 111 | 1 | 1 | | |

next S3 = !S2·!S1·S0
+ S3·!S0
+ S3·!S1

next S2 (=Rd)

| S3,S2,S1 \ S0, IOCtl | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 000 | | | | 1 |
| 001 | | | | |
| 011 | | | | |
| 010 | | | | |
| 110 | 1 | 1 | 1 | 1 |
| 100 | | | | |
| 101 | | | | |
| 111 | 1 | 1 | | |

next S2 = !S3·!S2·!S1·S0·!IOCtl
+ S3·S2·!S1
+ S3·S2·!S0

next S1

| S3,S2,S1 \ S0, IOCtl | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 000 | | | | |
| 001 | | | | |
| 011 | | | | |
| 010 | | | | |
| 110 | | | 1 | 1 |
| 100 | | | 1 | 1 |
| 101 | | | | |
| 111 | | | | |

next S1 = S3·!S1·S0

next S0

| S3,S2,S1 \ S0, IOCtl | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 000 | 1 | | | |
| 001 | | | | |
| 011 | | | | |
| 010 | | | | |
| 110 | | 1 | 1 | |
| 100 | | 1 | 1 | |
| 101 | | 1 | | |
| 111 | | 1 | | |

next S0 = S3·!S0·IOCtl
+ S3·!S1·IOCtl
+ !S3·!S2·!S1·!S0·!IOCtl

next IOReq

| S3,S2,S1 \ S0, IOCtl | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 000 | | | | |
| 001 | | | | |
| 011 | | | | |
| 010 | | | | |
| 110 | | 1 | | |
| 100 | | 1 | | |
| 101 | | 1 | | |
| 111 | | 1 | | |

next IOReq = S3·!S0·IOCtl

next CRE

| S3,S2,S1 \ S0, IOCtl | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 000 | | | | |
| 001 | | | | |
| 011 | | | | |
| 010 | | | | |
| 110 | | 1 | 1 | 1 |
| 100 | | 1 | 1 | 1 |
| 101 | | 1 | | |
| 111 | | 1 | | |

next CRE = S3·!S0·IOCtl
+ S3·!S1·S0

**Figure 2.2.18 IOPLA State Diagram and Logic Details.**

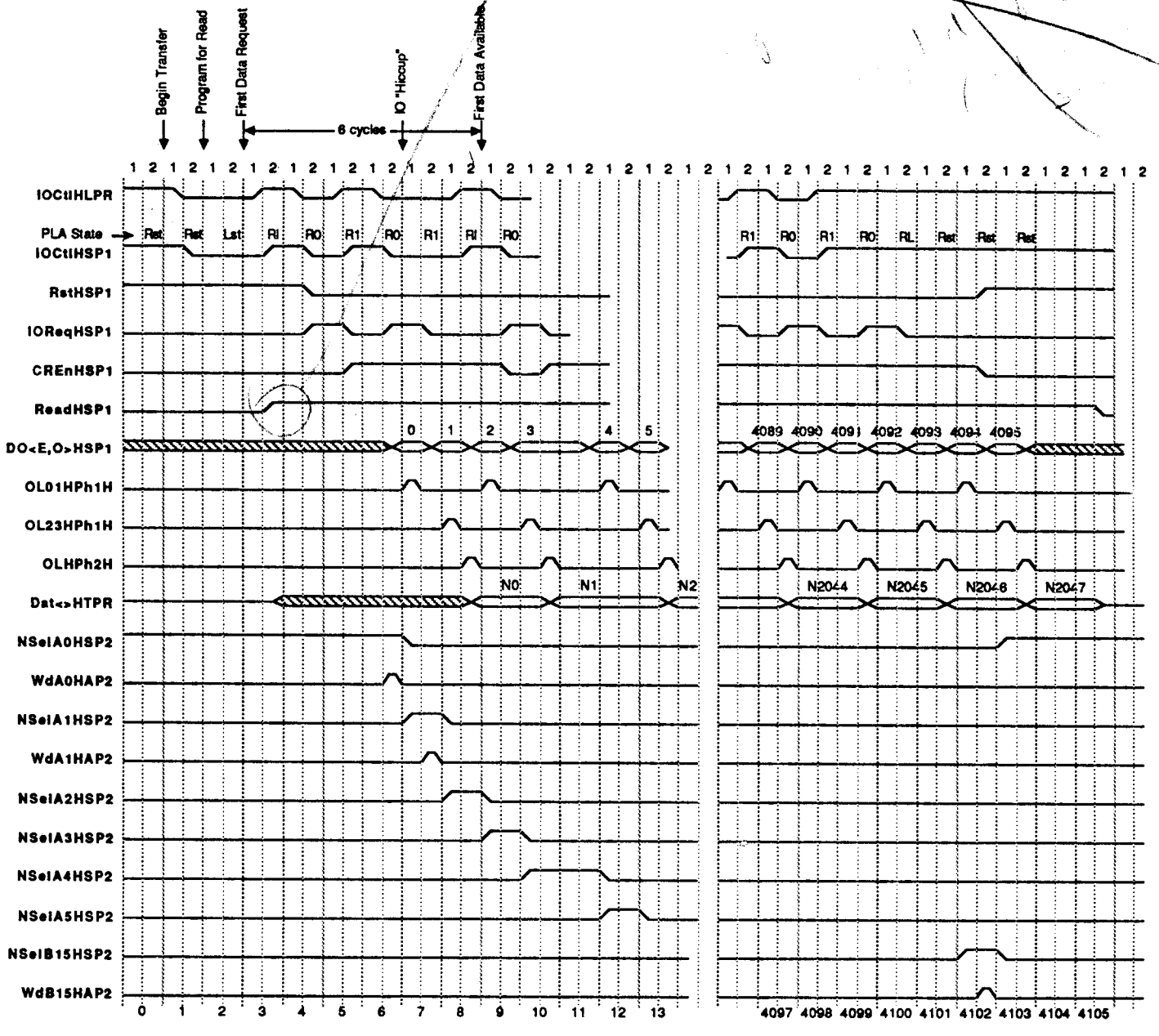



**Figure 2.2.19 CR Read Timing Details**

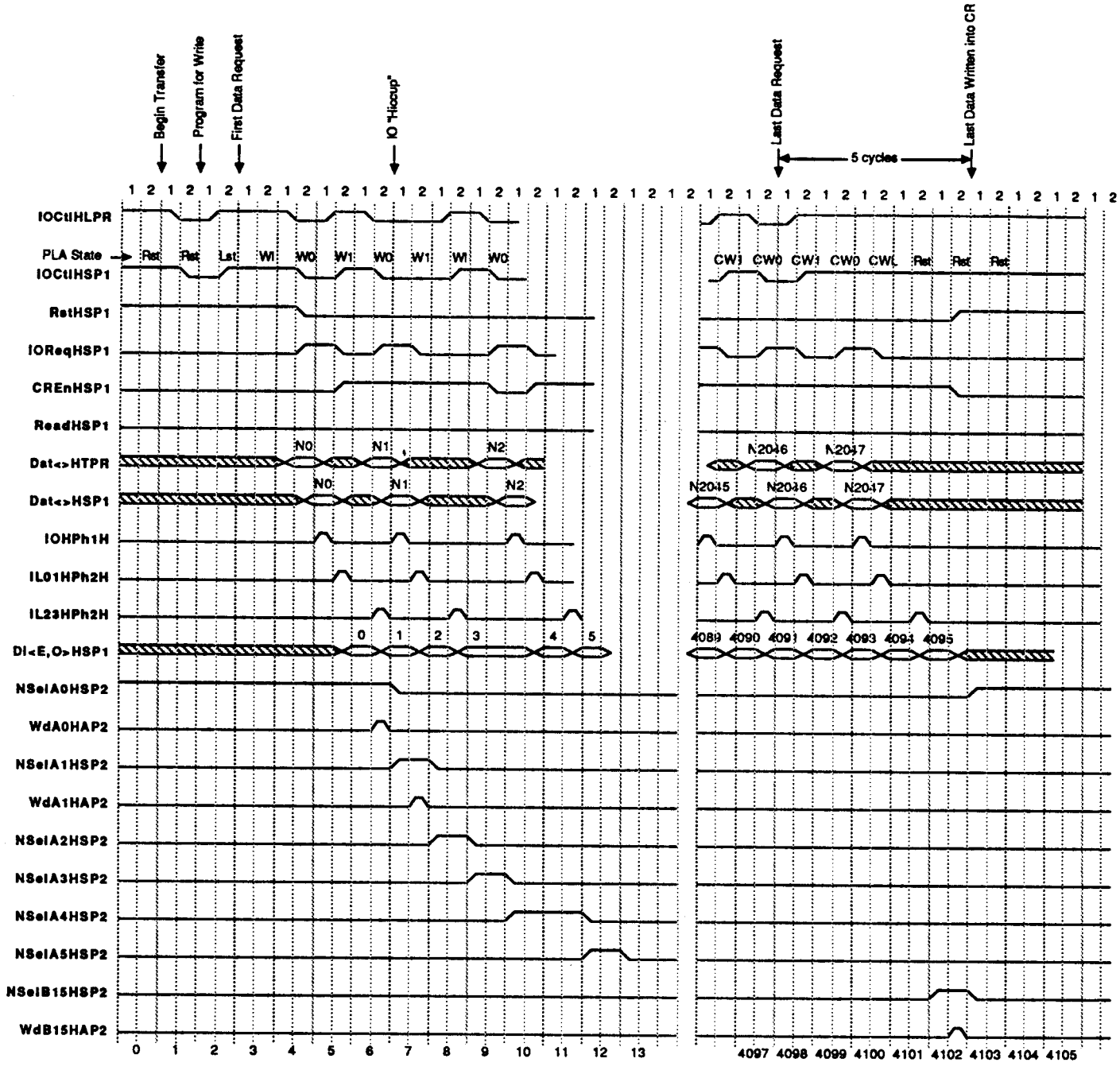



**Figure 2.2.20 CR Write Timing Details**

# IV.4 CORNER TURNER

This section describes the custom CMOS chip that transfers pixel data between the communication ring and the video ram backing store on renderer boards. This chip, the 'Corner Turner', converts word-parallel data (the format required by the ring and the rest of the Pxpl5 system) to and from nibble-serial format (the format required by the EMCs). Although the Corner Turner was designed with the needs of the Pxpl5 renderer board in mind, it is a general-purpose bidirectional nibble-serial-to-word-parallel conversion chip, potentially useful in a number of applications.

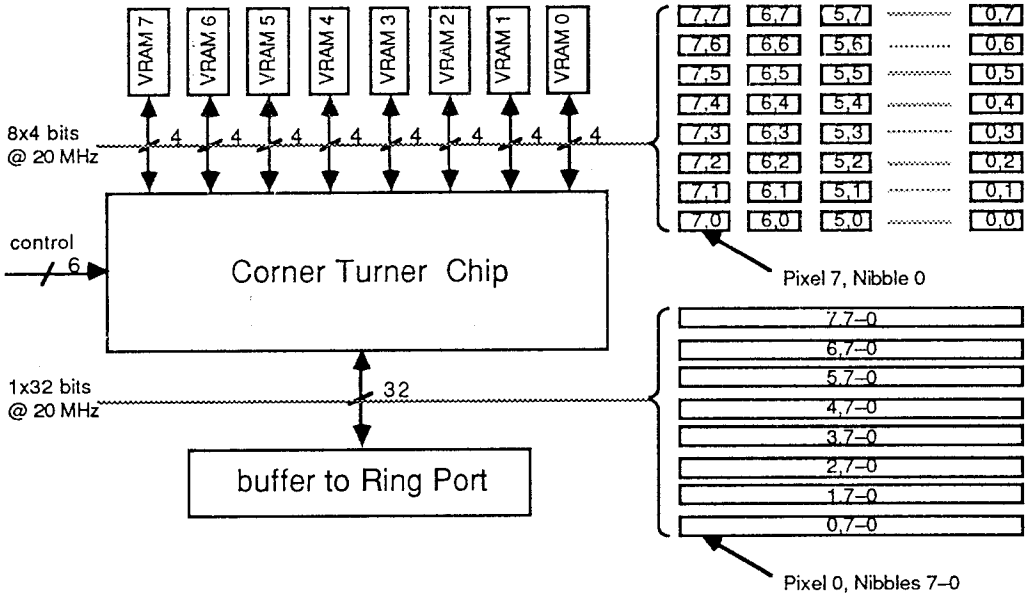



**Figure IV.4 Snapshot of Corner Turner in Operation**

The Corner Turner can operate in either of two modes: *load* (ring port to VRAMs) and *store* (VRAMs to ring port). The normal operation is to transfer pixels from the VRAMs to the ring port for transfer to the ring and frame buffer. The ring port to VRAM backing store path is used to load pre-computed images into the EMC's (or for mystical algorithms, such as volume rendering, ray tracing in the pixels, etc.).

## IV.4.1 Interface Requirements

The Corner Turner is merely a data transfer chip, so it must meet the bandwidth and data organization requirements at each port. Its fundamental operation is to transfer eight 32-bit words in nibble-serial format at one port to eight 32-bit words in parallel format at the other port (and vice-versa). The Corner Turner is not responsible for sequencing transfers; this is done by an external controller.

Bandwidth Requirements

The chip must be able to transfer pixel data at a sustained rate of 20 Mega-words per second when operating in either direction. The chip may actually run at 10 MHz, but the design goal is 20Mhz to allow flexibility in the renderer board design.

VRAM Interface

Each corner turner chip interfaces to 32 VRAMs (4 groups of 8 VRAMs, 1 group active at any given time). Each VRAM has a 4-bit serial port transferring one nibble (4 bits) of data from a single pixel every 50 nanoseconds. Since eight nibbles form one 32-bit word, one pixel of data will be transferred to/from a single VRAM every eight cycles. When transferring data from VRAMs to the ring port, the least significant nibble will be sent first, followed by the next most significant nibble and so on. When writing data to the VRAMs, nibbles are loaded in the same order.

Ring Port Interface

Each corner turner interfaces to the parallel ring interface port. One 32-bit word will be transferred to/from the ring interface every 50 nanoseconds.

Control Signals

Control signals are needed to tell the Corner Turner chip which direction to operate, when to shift data, and when to transfer data between left/right and up/down quads. These signals are generated external to the chip and must observe setup and hold times described in Section IV.4.7.

| | |
|---|---|
| HClkH | Single-phase clock for horizontal latches. |
| VClkH | Single-phase clock for vertical latches. |
| HLoadH | Load horizontal latches from corresponding vertical latches. |
| VLoadH | Load vertical latches from corresponding horizontal latches. |
| HOutEnabH | Output enable for horizontal pads. |
| VOutEnabH | Output enable for vertical pads. |

## IV.4.2 High-Level Design Issues

The Corner Turner is implemented as a two-dimensional array of edge-triggered D-flipflops. Since the unit of information is the nibble (data enters nibble-parallel), a unit cell is an array of four flipflops, or *quad*. Because the chip must shift serial data in at the same time that it shifts parallel data out, each element of the array actually contains two quads, which are distinguished as *horizontal* and *vertical* (based on the direction data flows from quad to quad).

Figure IV.4.2 shows the corner turner operating in its normal mode (transferring data from VRAM backing store to the ring). Columns of horizontal quads contain nibbles being read from the VRAMs. After eight clock cycles, all eight columns are loaded. Each horizontal quad then shifts its data into the corresponding vertical quad. Each row of vertical quads now contains an entire pixel (32 bits) of data. A pixel is clocked off the chip each cycle for the next eight cycles. At this time, the vertical quads have emptied their data, but another set of eight pixels is ready to be loaded from the horizontal quads. The chip operates continuously in this

manner. For transmitting data from the ring port to VRAMs, the entire process occurs in reverse.

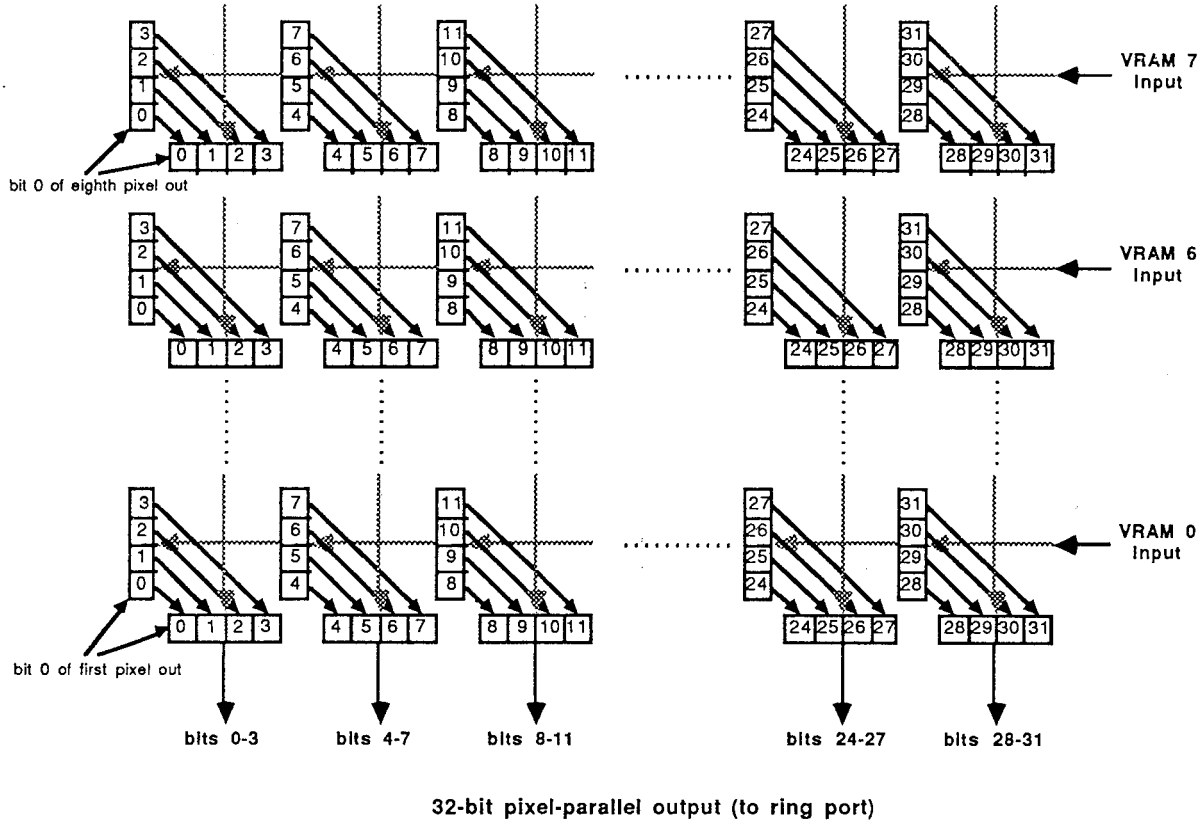



**Figure IV.4.2.1 Pixel Transfer from VRAMs to the ring port**

## IV.4.2.2 Design Alternatives

Since the corner turner chip is essentially a two-dimensional array of quads, the logic design of a single quad has a large impact on the layout of the entire chip. Conversely, the global layout constrains the design of a single quad. We considered a number of possible layout schemes. Each has its own advantages and disadvantages:

• Make everything bidirectional. D-flipflops would have inputs and outputs available at both ends. Multiplexers would be needed to switch between four possible inputs at each flipflop. We have made trial layouts for this design. Although simplest conceptually, it requires a great deal of area for the various interconnects and multiplexers. It also would require redesign of the EMC serial interface, since serial pixels would be read and written in different orders.

• Unidirectional array with rows and columns circularly connected. Each row of upper quads would have its ends connected, making it circular. The same would be true for each column of lower quads. The net effect would be to allow the chip to function in reverse, yet the elements in the array would be unidirectional. The problem with this approach is the length of the

connecting wires running across the chip; "adjacent" quads on the ends of a row would be connected by a very long wire, making fast operation more difficult. This can be mitigated by driving the long runs from I/O pads, rather than from quads inside the chip.

• Unidirectional array folded over twice. Topologically, a circularly-connected array is a torus. This suggests that the quads in each direction could be distributed evenly around the "circle" and laid on top of each other. In terms of the chip's layout, one can imagine grabbing the right edge of the flipflop array and folding it back to the left until it meets the left edge of the array. The same is done in the vertical direction. The same number of wires cross the chip as in the previous design, but instead of having seven short wires and one long one for each row or column, there are eight wires, each approximately twice the length of the previous short ones. This seems ideal for speed, but makes the cell design quite intricate and requires interleaved data wires.

All of these schemes share the disadvantage of having all of the wires emerge at two neighboring sides of the trip, rather than being uniformly distributed around its perimeter. We have not found a clean solution to this problem. The the timing effects of differing length wires can be ameliorated somewhat by latching inputs and outputs at the padframe, so setup and hold times do not vary.

We have attempted layouts for each of the above alternatives. We found that the first alternative is very inefficient in terms of area. The second and third are both feasible, but the second lends itself to a simple chip with an aspect ratio near 1. This is the layout strategy we adopted. It is shown in Figure IV.4.2.2.

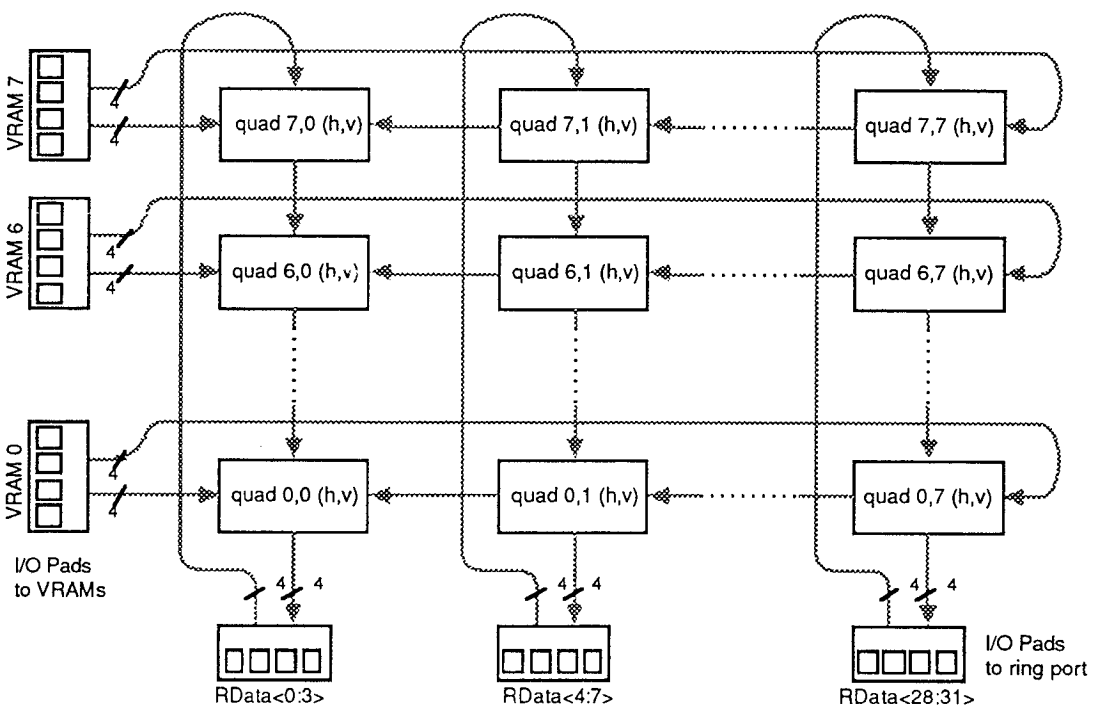



**Figure IV.4.2.2 Corner Turner Chip Global Layout**

## IV.4.3 Low-Level Design Issues

The edge-triggered D-flipflop is the basic unit of design within the corner turner chip. It has the advantage of being easy to lay out in complementary logic and requires a single clock. We adapted the logic design from more complicated D-flipflops in Weste and Eshraghian (p. 217, Figure 5.51c). Figure IV.4.3.1 shows the logic design for a single flipflop composed of complex gates. The layout for a cell was inspired by the layout in Weste and Eshraghian (Plate 10–note that this plate contains an error).

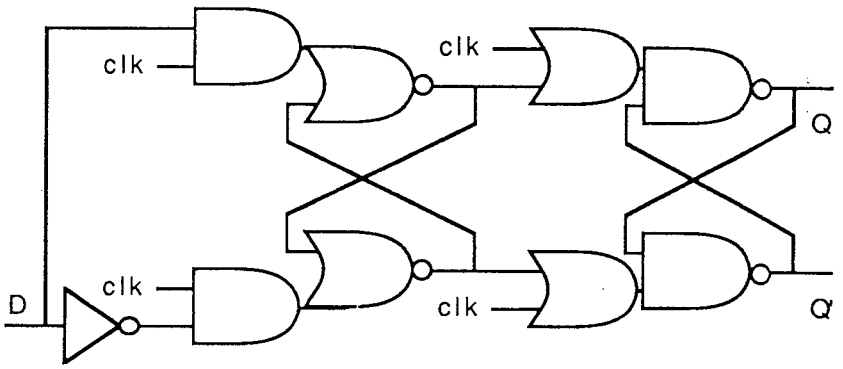



**Figure IV.4.3.1 Logic Design of Edge-Triggered FlipFlop**

One quad is composed of eight separate D-flipflops: four horizontal and four vertical. The arrangement of these flipflops within a quad determines the aspect ratio of the entire chip, since the chip is simply an 8x8 array of quads. Figure IV.4.3.2 shows an arrangement that does a reasonable job of minimizing area, yet preserves an aspect ratio near unity. The actual layout of a quad cell follows this diagram very closely, however, double rail signals are used instead of the single rail signals shown.

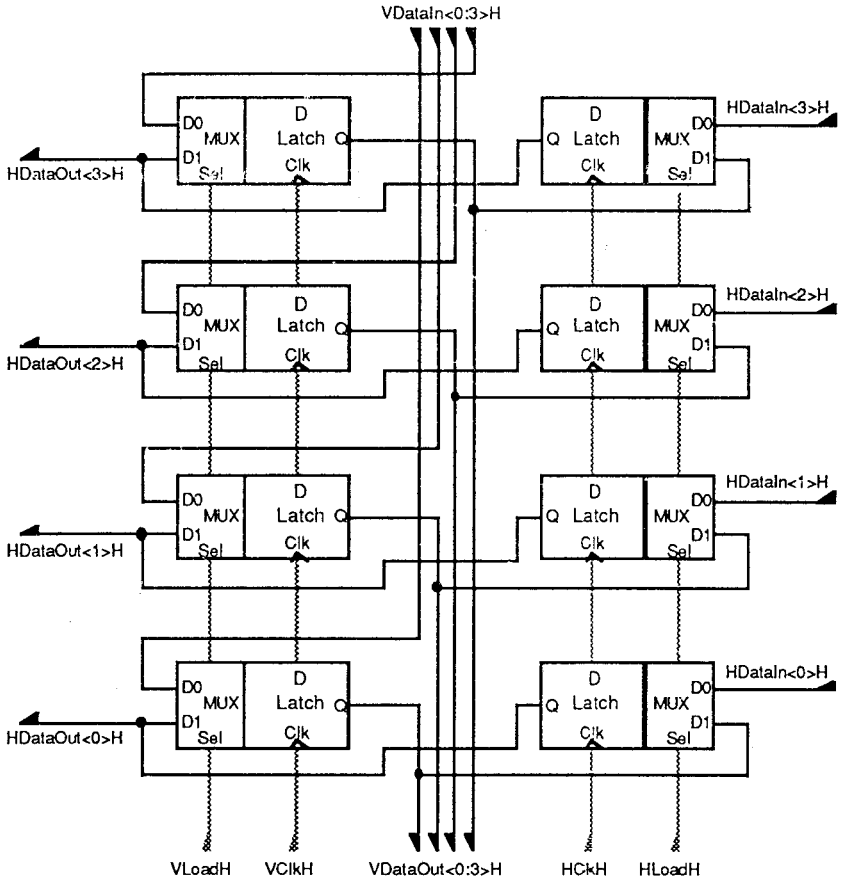



**Figure IV.4.3.2 Schematic Layout of a Single Quad**

## IV.4.4 Power Distribution

Power is distributed within the chip by interleaved metal1 Vdd and GND rails. A number of separate power and ground pins are used to distribute power evenly around the chip. We must calculate the power requirements of various parts of the chip for two reasons: to prevent faults due to metal migration and to assure that power and ground noise levels stay safely below transistor switching thresholds. These effects can be mitigated by providing extra power and ground pins and by increasing the widths of Vdd and GND rails within the chip.

### IV.4.4.1 Internal power distribution

To compute the widths for power and ground rails, we compute the minimum width needed to prevent metal migration (Wmm), and the minimum width needed for noise protection (Wn). The actual width chosen (Wpg) is the larger of these two values.

Simulation with CAzM provides a worst-case average power estimate of 0.779mW for each master/slave flip-flop (using best-case models and conditions). The worst-case average current for a master/slave flip-flop ($I_{ff}$) is, therefore, 0.779mW / 5.5 volts = .142 mA. Power is provided by horizontal, interleaved power and ground rails. To calculate the width for these

rails, we must find the average current carried by a single rail:

$$I_{rail} = I_{ff} * (4 \text{ M/S flip-flops / column}) * (8 \text{ columns}) = .142 * 4 * 8 = 4.53 \text{ mA}$$

Using the rules derived in §IV.1.Appendix C for computing RMS currents, we conservatively estimate Irms for a single power rail to be 5 * Irail = 22.7 mA. Since Imax for metal1 is 1mA/μ (table §IV.1.1.1), Wmm = 22.7μ.

Simulation using CAzM provides a worst-case peak power consumption of 11.0mW for a single M/S flip-flop, and a corresponding peak current (Iff) of 11.0mW / 5.5 V = 2.0 mA. Irail under these conditions is given by:

$$I_{rail} = I_{ff} * (4 \text{ M/S flip-flops / column}) * (8 \text{ columns}) = 2.0 * 4 * 8 = 64.0 \text{ mA}$$

The voltage drop Vrail is given by:

$$Vrail = Irail * (\# \text{ squares} / 2) * Rsh$$

The factor of 2 appears because current is consumed uniformly along the length of the power and ground rails. A tolerable voltage drop is 250mV. Plugging this into the above equation:

$$\# \text{ squares} = 2 * Vrail / (Irail * Rsh) = 2 * 250mV / ( 64.0mA * 0.06\Omega/\text{square}) = 130$$

Since each power and ground rail is approximately 4000μ long, Wn = 4000/130 = 30.7μ. Since resistances were ignored in the above calculation, this number will be somewhat larger than necessary. Since Wmm and Wn are each in the neighborhood of 25μ, we adopted a power rail size of 28μ (allowing a couple of μ for contact cuts).

### IV.4.4.2 I/O Pad power distribution

The I/O pads must drive large capacitances on and off chip. Power distribution is, therefore, an important requirement. A total of eight power and ground pins are available for the pad frame. A power and ground pad was placed at each corner of the chip. This was done to equitably distribute power around the padframe, and because the power/ground pins were the only pins that could be made to fit in the corners.

## IV.4.5 Pad Frame and Chip Floorplan

The corner turner chip requires 64 data pins plus 6 control pins, a total of 70 pins (plus power and ground). An 84-pin PGA, therefore, is an ideal mounting choice for this chip, allowing 6 power and 6 ground (plus a required substrate contact). Mosis provides 84-pin PGA's in two cavity sizes: 7.9mm x 9.2mm and 6.9mm x 6.8mm. Since the corner turner chip is pin-limited, the smaller die cavity is appropriate. Standard pad frames are available for these dia sizes, which simplify the positioning of pads. We used John Eyles' program *magframe* to generate the chip layout from Mosis' pad frame and the pads we designed. Figure IV.4.5.1 shows a rough sketch of the chip's floorplan.

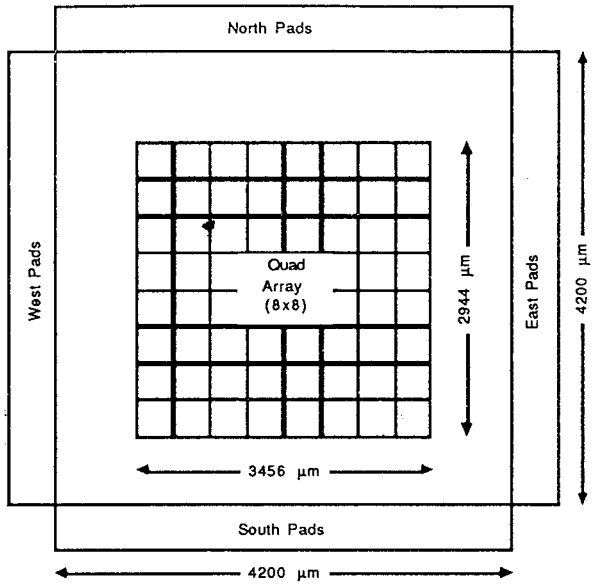



**Figure IV.4.5.1 Corner Turner Floorplan**

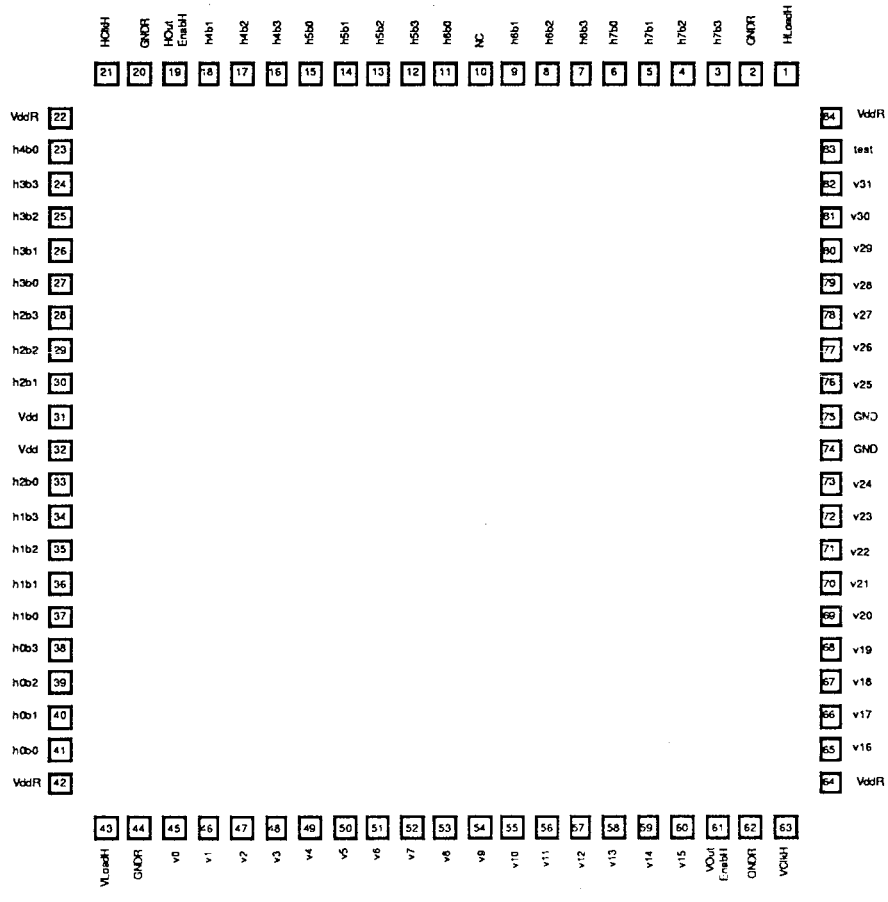
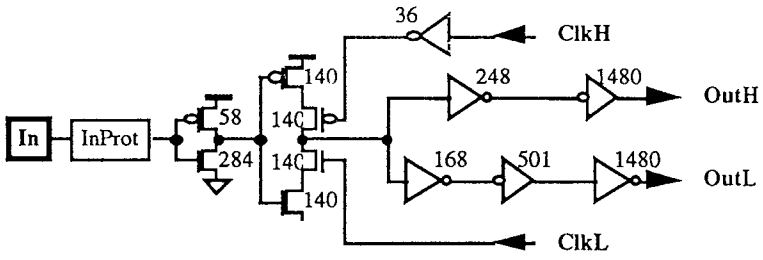



**Figure IV.4.5.2 Corner Turner pinout**

## IV.4.6 Pad Library

After an initial version of the chip was laid out, it was determined that changes were needed to the standard Mosis 2.0μ pad library. In particular, to minimize variations in setup, hold and propagation times between pins, we decided to latch inputs and outputs at the pads. Also, the input buffers in the standard pad library were not large enough to drive the high capacitances on the global clock and control signals. To solve these concerns, we designed a new family of pads, wider and taller than standard size. The largest input buffers were designed to make use of wasted area in the corner. The remainder of this section contains data sheets for the Corner Turner Pad library.

# Pad2_CORIIn2TL
## Latched TTL Input Buffer




**Description.** A latched corner input pad, buffered for TTL input signals. The latch is transparent when **ClkL** is high and latches when **ClkL** goes low. This cell produces true and complement outputs with approximately equal delay and can drive internal loads up to about 35 pf at 20 MHz clock rates. Because of the large drive capacity, this cell is combined with a Vdd and GND pad into a cell that fits in one of the four corners of the Mosis 84p69x68 pad frame.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VIL | LO input threshold voltage | NoiseP for Max | | 1.41 | 1.08 | volts |
| VIH | HI input threshold voltage | NoiseN for Min | 1.62 | 1.41 | | volts |
| Idd | Ave. static supply current | | | 0.53 | 1.59 | ma |
| Cpd | Equiv. power diss. cap. | | | | 28.5 | pf |
| CIn | Capacitance on input In | | | 4.0 | | pf |
| CClkL | Capacitance on input ClkL | | | 876 | | ff |

### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tp | Delay In to Out<H,L> | Cload = 30pf | 2.3 | | 7.8 | nsec |
| tsk | Skew between <H,L> | Cload = 30pf | | | 0.7 | nsec |
| tpP | Delay ClkL to Out | Cload = 30pf (1) | 1.4 | | 6.9 | nsec |
| tr | Out<H,L> risetime | Cload = 30pf | 1.3 | | 4.0 | nsec |
| tf | Out<H,L> falltime | Cload = 30pf | 0.8 | | 2.4 | nsec |

### AC Operating Requirements

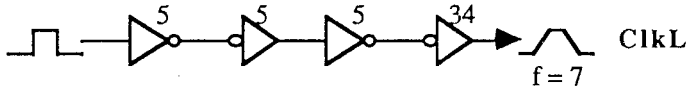
| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| ts | Setup time w.r.t. ClkL trail | Worst (1) | 4.9 | | | nsec |
| th | Hold time w.r.t. ClkL trail | Best (1) | 0 | | | nsec |

## Loading Characteristics

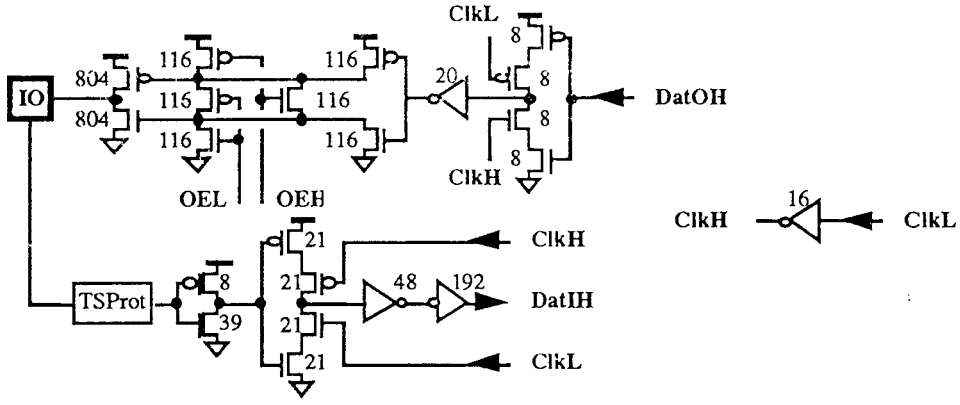
| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tp | = | 6.48 + 0.044 Cload (pf) | nsec | W |
| tpP | = | 5.64 + 0.041 Cload (pf) | nsec | W |
| tr | = | 0.89 + 0.102 Cload (pf) | nsec | W |
| tf | = | 1.16 + 0.040 Cload (pf) | nsec | W |

Notes

(1) Clock modeled by:



# Pad2_240ITSTL
## Tri-State Latched I/O Pad with TTL Input Buffer




**Description.** A 240µ-wide bi-directional pad with input and output latches, buffered for TTL levels. The output circuitry in this pad can drive a 50Ω load at TTL-compatible levels at speeds up to 20MHz (NRZ). (it is equivalent to that of Pad2_240cTSTL). The input buffer is suitable for loads up to about 5pf at 40MHz clock rates. The input latch is transparent when **ClkL** is high and latches when **ClkL** goes low. The output latch is transparent when **ClkL** is low and latches when **ClkL** goes high.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VIL | LO input threshold voltage | NoiseP | | 1.40 | 1.07 | volts |
| VIH | HI input threshold voltage | NoiseN | 1.62 | 1.40 | | volts |
| Idd | Ave. static supply current | No DC load | | 0.07 | 0.21 | ma |
| Cpd | Equiv. power diss. cap. | | | | 5.4 | pf |
| CIO | Capacitance on pad IO | | | 4.1 | | pf |
| CDOut | Capacitance on input DatOH | | | 180 | | ff |
| CDOE | Capacitance on inputs OE<H,L> | | | 599 | | ff |
| CPh2H | Capacitance on input ClkL | | | 372 | | ff |

### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tpo | DatOH to IO delay | Rload=50Ω, Cload = 50pf (1) | 3.0 | | 11.2 | nsec |
| tpPo | ClkL to IO delay | Rload=50Ω, Cload = 50pf (1,2) | 3.4 | | 11.9 | nsec |
| tro | IO risetime | Rload=50Ω, Cload = 50pf (1) | 0.77 | | 4.5 | nsec |
| tfo | IO falltime | Rload=50Ω, Cload = 50pf (1) | 0.62 | | 3.6 | nsec |
| tpena | Output Enable time | Rload=50Ω, Cload = 50pf (1,3) | 1.0 | | 3.2 | nsec |
| tpdis | Output Disable time | Rload=50Ω, Cload = 50pf (1,3) | 1.0 | | 4.2 | nsec |
| tpo | DatOH to IO delay | Cap. load, Cload = 50pf | 3.1 | | 15.6 | nsec |
| tpPo | ClkL to IO delay | Cap. load, Cload = 50pf (2) | 3.6 | | 15.4 | nsec |
| tro | IO risetime | Cap. load, Cload = 50pf | 0.71 | | 2.8 | nsec |
| tfo | IO falltime | Cap. load, Cload = 50pf | 0.52 | | 1.8 | nsec |
| tpi | IO to DataIn delay | Cload = 3pf | 2.1 | | 8.1 | nsec |
| tpPi | ClkL to DataIn | Cload = 3pf (2) | 1.3 | | 7.4 | nsec |
| tri | DataIn risetime | Cload = 3pf | 1.1 | | 3.6 | nsec |

| | | | | | |
|---|---|---|---|---|---|
| tfi | DataIn falltime | Cload = 3pf | 0.7 | 2.1 | nsec |

## AC Operating Requirements

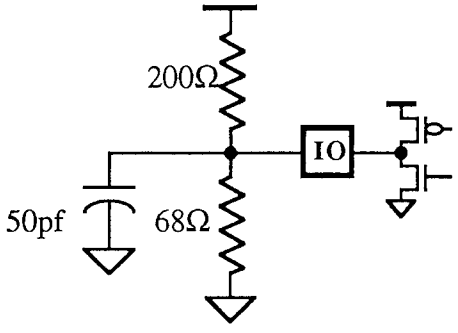
| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tsi | Input setup time w.r.t. ClkL trail | Worst (2) | 6.9 | | | nsec |
| thi | Input hold time w.r.t. ClkL trail | Best (2) | 0 | | | nsec |
| tso | Output setup time w.r.t. ClkL lead | Worst (2) | 6.5 | | | nsec |
| tho | Output hold time w.r.t. ClkL lead | Best (2) | 0.5 | | | nsec |

## Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tpi | = | 7.0 + 0.36 Cload (pf) | nsec | W |
| tpPi | = | 6.1 + 0.41 Cload (pf) | nsec | W (2) |
| tri | = | 1.2 + 0.79 Cload (pf) | nsec | W |
| tfi | = | 1.2 + 0.28 Cload (pf) | nsec | W |

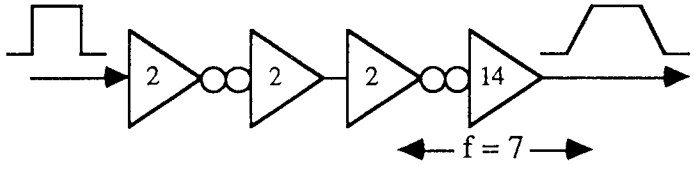
Notes:

(1) Load for these tests is:



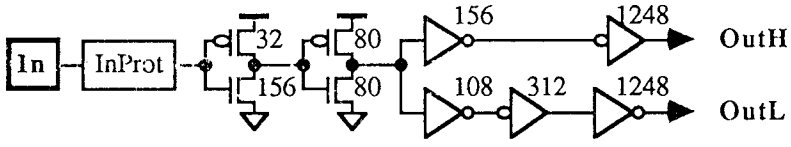

This is the recommended 50Ω termination for this pad; the pad should never be terminated with an equivalent resistance smaller than 50Ω. DC and transient performance determined using TTL levels: $V_{OH}$ = 2.0 volts, $V_{OL}$ = 0.8 volts, $V_{switch}$ = 1.4 volts.

(2) Following circuit used to generate **ClkL** clock:




(3) Output enable and disable times are measured from the transition of OEH and OEL to the time when the output current reaches 90% of its maximum value. Rise times and fall times should be added to the enable/disable times to determine when the output becomes valid.

# Pad2_300IIn2TB
## TTL Input Buffer




**Description.** A 300μ-wide buffered input pad for TTL input signals. This cell produces true and complement outputs with approximately equal delay and can drive internal loads up to about 25 pf at 20 MHz (RZ). It consists of circuitry identical to that in **Pad2_240nIn1TB**, but with wider devices, and the wider power and ground rails, and clock rail of the 'T' series pads.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VIL | LO input threshold voltage | NoiseP | | | 1.07 | volts |
| VIH | HI input threshold voltage | NoiseN | 1.61 | | | volts |
| Idd | Ave. static supply current | | | 0.28 | 0.87 | ma |
| Cpd | Equiv. power diss. cap. | | | 20.1 | 26.0 | pf |
| CIn | Capacitance on input In | | | 3.9 | | pf |

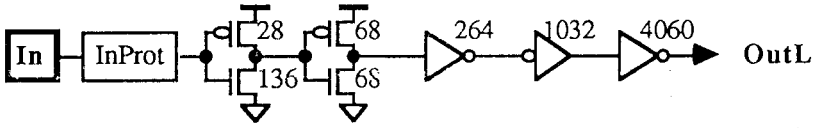
### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tpLL | In-LO to OutH-LO delay | Cload = 20pf, both outputs | 2.2 | | 6.9 | nsec |
| tpLH | In-LO to OutL-HI delay | Cload = 20pf, both outputs | 2.4 | | 7.6 | nsec |
| tpHH | In-HI to OutH-HI delay | Cload = 20pf, both outputs | 2.1 | | 6.2 | nsec |
| tpHL | In-HI to OutL-LO delay | Cload = 20pf, both outputs | 2.0 | | 6.1 | nsec |
| tr | Typical Out<H,L> risetime | Cload = 20pf | 1.3 | | 4.1 | nsec |
| tf | Typical Out<H,L> falltime | Cload = 20pf | 0.8 | | 2.4 | nsec |

### Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tpLL | = | 5.73 + 0.057 Cload (pf) | nsec | W |
| tpLH | = | 5.98 + 0.080 Cload (pf) | nsec | W |
| tpHH | = | 4.61 + 0.081 Cload (pf) | nsec | W |
| tpHL | = | 5.16 + 0.045 Cload (pf) | nsec | W |
| tr | = | 0.87 + 0.161 Cload (pf) | nsec | W |
| tf | = | 1.13 + 0.065 Cload (pf) | nsec | W |

# Pad2_CORIIn4TB
## TTL Input Buffer




**Description.** A corner buffered input pad for TTL input signals. This cell produces a complement output only and can drive internal loads up to about 120 pf at 20 MHz. It can be used, for example, as a single-phase clock buffer for a medium project.

### Electrical Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| VIL | LO input threshold voltage | NoiseP | | | 1.08 | volts |
| VIH | HI input threshold voltage | NoiseN | 1.62 | | | volts |
| Idd | Ave. static supply current | | | 0.27 | 0.79 | ma |
| Cpd | Equiv. power diss. cap. | | | 41.6 | 52.6 | pf |
| CIn | Capacitance on input In | | | 3.7 | | pf |

### Switching Characteristics

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tpLH | In-LO to OutH-HI delay | Cload = 100pf | 2.2 | | 7.1 | nsec |
| tpHL | In-HI to OutH-LO delay | Cload = 100pf | 2.5 | | 8.6 | nsec |
| tr | Typical Out<H,L> risetime | Cload = 100pf | 0.9 | | 4.6 | nsec |
| tf | Typical Out<H,L> falltime | Cload = 100pf | 0.5 | | 2.3 | nsec |

### Loading Characteristics

| Parameter | | Equation | Units | Conditions |
|---|---|---|---|---|
| tpLH | = | 5.95 + 0.011 Cload (pf) | nsec | W |
| tpHL | = | 6.75 + 0.018 Cload (pf) | nsec | W |
| tr | = | 0.70 + 0.038 Cload (pf) | nsec | W |
| tf | = | 0.82 + 0.014 Cload (pf) | nsec | W |

## IV.4.7 Simulation and Circuit Verification

Two levels of simulation were applied to The Corner Turner chip. Logic simulation was used to verify circuit correctness and circuit simulation was used to analyze timing and logic levels.

Logic simulation was performed using COSMOS (Carnegie-Mellon Univ.). During the initial design phase of the chip, individual and groups of subcells were simulated for correctness. After the entire chip was layed out and wired, COSMOS was used to verify its behavior by applying test vectors and verifying results. A program was used to generate random test vectors for COSMOS, then to clock them through the chip in both directions, checking the outputs against predicted results. Logic simulation on the corner turner is very fast (approximately two clock cycles per second on our VAX 3200).

Two circuit simulators were available: SPICE and CAzM. After verifying that the two simulators produce comparable results given the same circuit data, we shifted all of our simulation to CAzM (much easier and faster). We simulated several aspects of the circuit. We have already mentioned the simulation done to estimate gate delays in Section IV.4.4. The following sections detail each other type of simulation we performed.

There are three pertinent timing specifications needed when designing a circuit with edge-triggered latches: the setup and hold times for inputs, and the clock to output propagation delay for outputs. These delays are shown schematically in Figure IV.4.6.1.

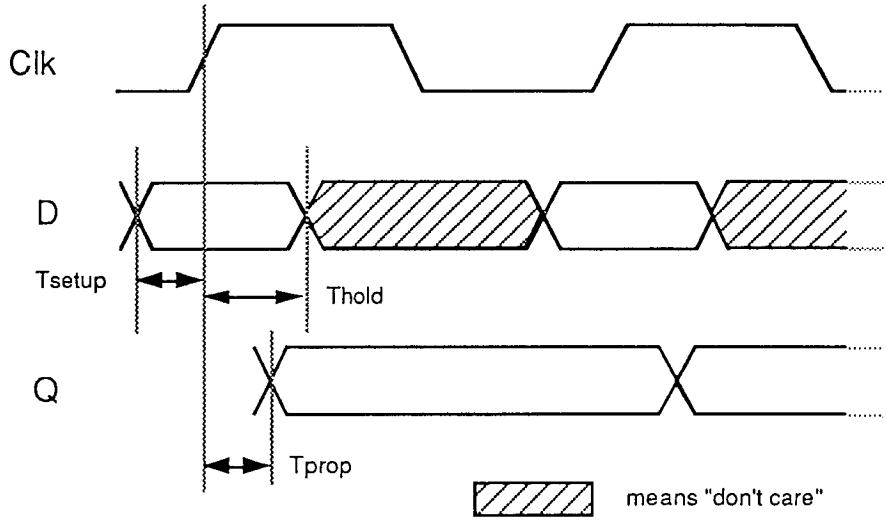



**Figure IV.4.7.1 Timing for leading edge triggered latch**

The setup and hold times can be measured by providing the circuit with clock and data inputs of slightly different frequencies. This causes the signals to slowly shift phase with respect to each other. After each cycle, the output of the flip-flop is examined for validity. The point at which the output goes from valid to invalid (and vice versa) determines the setup and hold times.

### IV.4.7.1 Switching characteristics for a single quad

Transistors of the master/slave flip-flops must be sized so that the delay from the rising edge of the chip's internal clock satisfies the setup and hold times of the succeeding quad. CAzM was used to calculate the worst-case delay through a quad. The worst-case delay occurs for quad vertical flip-flops, which drive wires to adjacent quads that are approximately 500μ long.

Simulation using CAzM produced the following results:

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tpPo | ClkL to QH, QL delay | C(QH,QL) = 150f for min | 0.6 | | 13.2 | nsec |
| tro | QH, QL risetime | times, 500f for max times | 2.5 | | 17.2 | nsec |
| tfo | QH, QL falltime | | 1.2 | | 6.9 | nsec |
| ts | Setup time w.r.t. ClkL | C(QH,QL) = 0f | 15.0 | | | nsec |
| th | Hold time w.r.t. ClkL | C(QH,QL) = 0f | 0.0 | | | nsec |

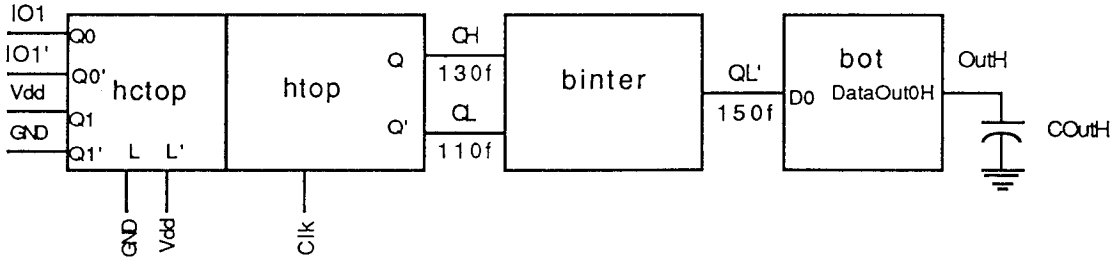
### IV.4.7.2 Delay from quad to output pad (vertical quads)

Border quads must drive cross-chip wires to the pads. The parasitic capacitance of these wires is too much to drive without buffering. Horizontal and vertical border quads have different capacitive loads and different buffering requirements. For vertical quads, three layers of buffering were used: a 21/21 μ inverter near the slave flip-flop in cells **binter**[0-3] (to drive the parasitic capacitances in the vertical wiring channels—up to 432ff), a 15/15 μ inverter in cell **bot**, followed by a 96/96 μ inverter in cell **bot** to drive the cross-chip wires (between 773 and 1996ff parasitic capacitance). Worst case device models and longest wire lengths were used to calculate maximum parameters below; best case models and shortest wire lengths were used to calculate minimum parameters below. Simulation using CAzM produced the following:

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tpPo | ClkL to VDataOutH | COutH = 400f for min, 1700f for max (1) | 3.4 | 8.3 | 13.8 | nsec |

Notes:

(1) Circuit for this test is:




### IV.4.7.3 Delay from quad to output pad (horizontal quads)

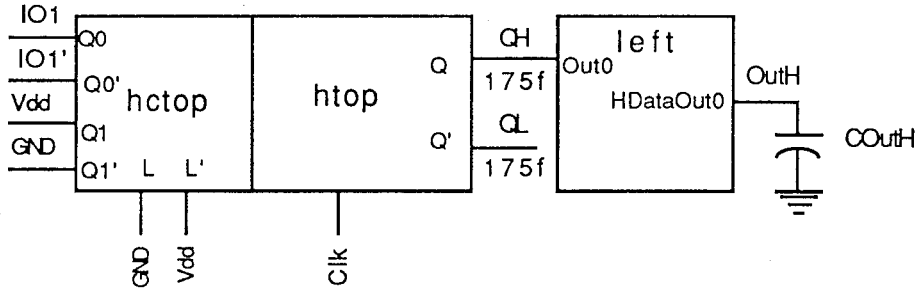
For horizontal quads, only two layers of buffering were used: a 28/28 μ inverter fanout stage,

followed by a 126/126 μ inverter to drive the cross-chip wires (between 991 and 2066ff parasitic capacitance). Both inverters are contained in cell left. Worst case device models and longest wire lengths were used to calculate maximum parameters below; best case models and shortest wire lengths were used to calculate minimum parameters below. Simulation using CAzM produced the following:

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tpPo | ClkL to HDataOutH | COutH = 500f for min, 1700f for max | 1.8 | 7.8 | 12.8 | nsec |

Notes:

(1) Circuit for this test is:



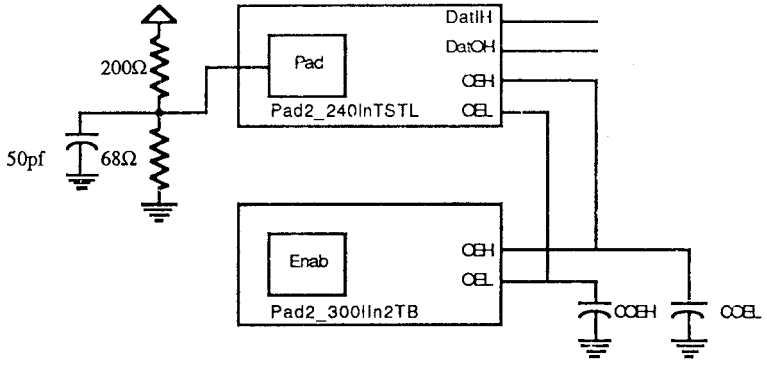

### IV.4.7.4 Tri-state pad output enable/disable times

Enable/disable times depend on the characteristics of the tri-state pads, the enable input driver pad, and the total capacitive load in the OEH/OEL wires. To measure the enable/disable times, we constructed a test circuit containing the tristate pad Pad2_240ITSTL, a true/complement input pad, Pad2_300IIn2TB, and the necessary load capacitors. Simulation using CAzM produced the following results:

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| Ten | Output enable time | COEH, COEL = 23pf (1) | 2.1 | | 8.2 | nsec |
| Tdis | Output disable time | COEH, COEL = 23pf (1) | 2.5 | | 10.0 | nsec |

Notes:

(1) Circuit for this test is:



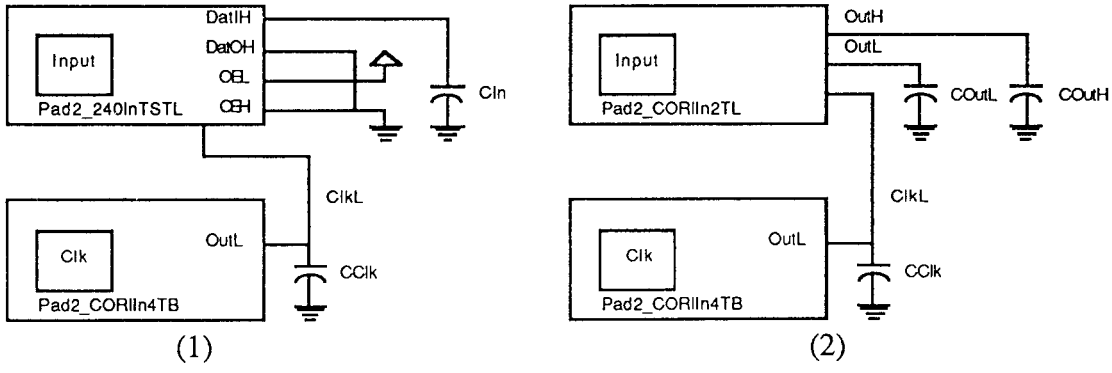

### IV.4.7.5 Setup and Hold times for latched inputs

To measure the setup and hold times for latched inputs, the delay through the clock driver must be taken into account. Rather than simulating the entire chip, we constructed a test circuit which contains just the clock driver (with its load capacitance) and latched input pad (with its associated loads). In the corner turner chip, two latched pad types are needed: **Pad2_240ITSTL** for data I/O pins, and **Pad2_CORIIn2TL** for load inputs. Setup and hold times were calculated by simulating under best and worst-case conditions and choosing the most conservative value. For setup times, a small clock capacitance (90pf) and a large input load capacitance were used (see "Conditions"). For hold times, a large clock capacitance (110pf) and small input load capacitance were used. CAzM was used to obtain the following operating requirements:

| Parameter | | Conditions | Min. | Units |
|---|---|---|---|---|
| tsd | Setup time for data input | Worst, CClk=90p, CIn=3.0p (1) | 1.0 | nsec |
| thd | Hold time for data input | Worst, CClk=110p, CIn=0.5p (1) | 8.0 | nsec |
| tsl | Setup time for load input | Worst, CClk=90p, CLoad(H,L) =35p (2) | 1.0 | nsec |
| thl | Hold time for load input | Worst, CClk=110p, CLoad(H,L)=25p (2) | 8.0 | nsec |
| ts | Setup time for any input | | 1.0 | nsec |
| th | Hold time for any input | | 8.0 | nsec |

Notes:

(1,2) Circuits for these tests are:



### IV.4.7.6 Propagation delay and overall timing check

Although each stage in the chip's pipelined operation has been simulated above, it would still be comforting to verify that the chip's overall timing is consistent. We can do that by building a somewhat more complicated test circuit, containing at least one copy of each cell needed in the overall design. The circuit we used is diagrammed in Figure IV.4.6.6 below.

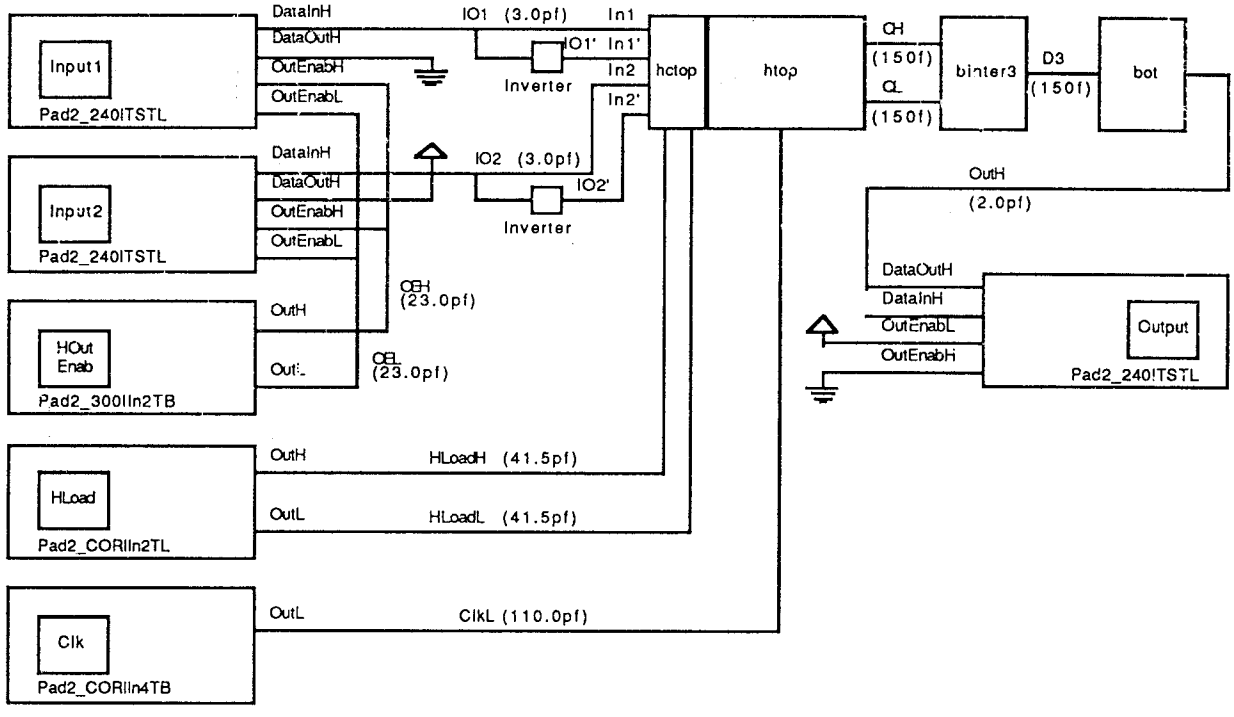



**Figure IV.4.7.6 Test circuit used to model delays**

The capacitances in the figure are extra capacitors used to model actual circuit parasitics (calculated using getcap on the whole chip). It was simulated under worst and best-case conditions at 20MHz (the design clock speed for the chip). From this composite circuit, we were able to compute the output propagation delay for the entire chip:

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tPttl | Prop. delay w.r.t rising edge | Worst case, ttl load (1) | 5.6 | | 19.3 | nsec |
| tPcmos | Prop. delay w.r.t. rising edge | Worst case, 50pf load | 6.4 | | 22.8 | nsec |

Notes:

(1) TTL output load modeled by following circuit:

## IV.4.8 Data Sheet

The Corner Turner is a general-purpose, bidirectional data transfer chip that converts eight nibble-serial data streams into a single 32-bit word-parallel data streams. Its two ports are independent, each having its own clock. The only interdependence is the cycle in which data are loaded from one side to the other, which places restrictions on the clock skew between H and V clocks. Each port runs at a maximum speed of 20 MHz. Section IV.4.1 contains more information about the Corner Turner's logic

The Corner Turner is implemented using MOSIS' 2μ CMOS technology in an 84-pin PGA. Figure IV.4.8.1 shows the assignment of logical signals to PGA pins. Figure IV.4.8.2 depicts the chip's operation when transferring data from the H side to V side (nibble-serial to word-parallel). Figure IV.4.8.3 shows the chip transferring data from the V to H side (word-parallel to nibble-serial). Figure IV.4.8.4 shows timing requirements.

| Location | Signal | Location | Signal | Location | Signal | Location | Signal |
|---|---|---|---|---|---|---|---|
| A1 | Vdd | C1 | h7b2 | F9 | v8 | K1 | HOutEn |
| A2 | v31 | C2 | GND | F10 | v7 | K2 | Vdd |
| A3 | v30 | C5 | GND | F11 | v12 | K3 | h4b0 |
| A4 | v28 | C6 | GND | G1 | h5b2 | K4 | h3b1 |
| A5 | v25 | C7 | v21 | G2 | h5b1 | K5 | h2b2 |
| A6 | v27 | C10 | GND | G3 | h5b0 | K6 | h2b3 |
| A7 | v23 | C11 | v15 | G9 | v4 | K7 | h1b2 |
| A8 | v20 | D1 | h7b0 | G10 | v5 | K8 | h0b3 |
| A9 | v18 | D2 | h7b1 | G11 | v6 | K9 | h0b0 |
| A10 | v17 | D10 | v14 | H1 | h4b3 | K10 | VLoad |
| A11 | VClk | D11 | v13 | H2 | h4b2 | K11 | v0 |
| B1 | h7b3 | E1 | h6b1 | H10 | v2 | L1 | HClk |
| B2 | HLoad | E2 | h6b2 | H11 | v3 | L2 | h3b3 |
| B3 | test | E3 | NC* | J1 | h4b1 | L3 | h3b2 |
| B4 | v29 | E9 | v9 | J2 | GND | L4 | h3b0 |
| B5 | v26 | E10 | v11 | J5 | Vdd | L5 | h2b1 |
| B6 | v24 | E11 | v10 | J6 | Vdd | L6 | h2b0 |
| B7 | v22 | F1 | h5b3 | J7 | h1b1 | L7 | h1b3 |
| B8 | v19 | F2 | h6b3 | J10 | GND | L8 | h1b0 |
| B9 | v16 | F3 | h6b0 | J11 | v1 | L9 | h0b2 |
| B10 | Vdd | | | | | L10 | h0b1 |
| B11 | VOutEn | | | | | L11 | Vdd |

**Figure IV.4.8.1 Pin cross-reference by number**

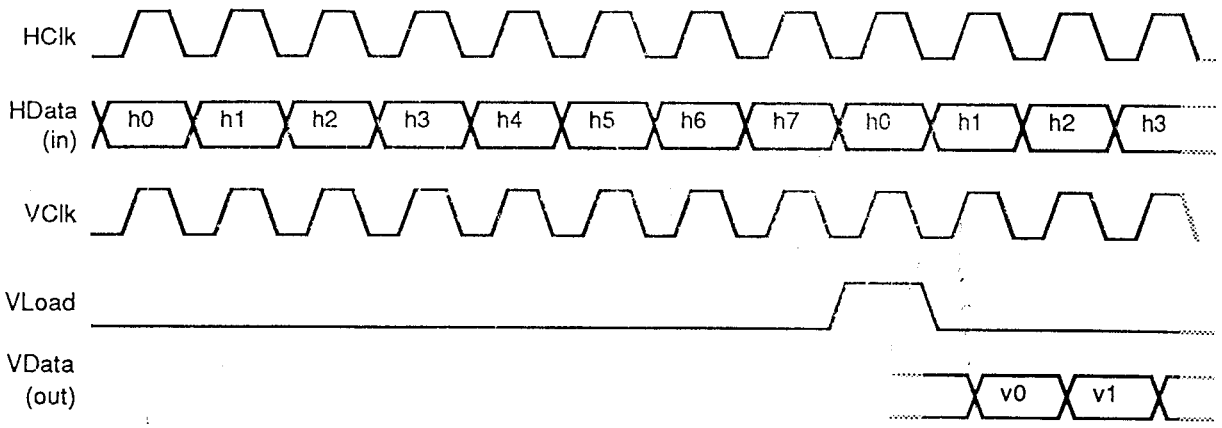



**Figure IV.4.8.2 Input and output waveforms for transferring data from H to V port**

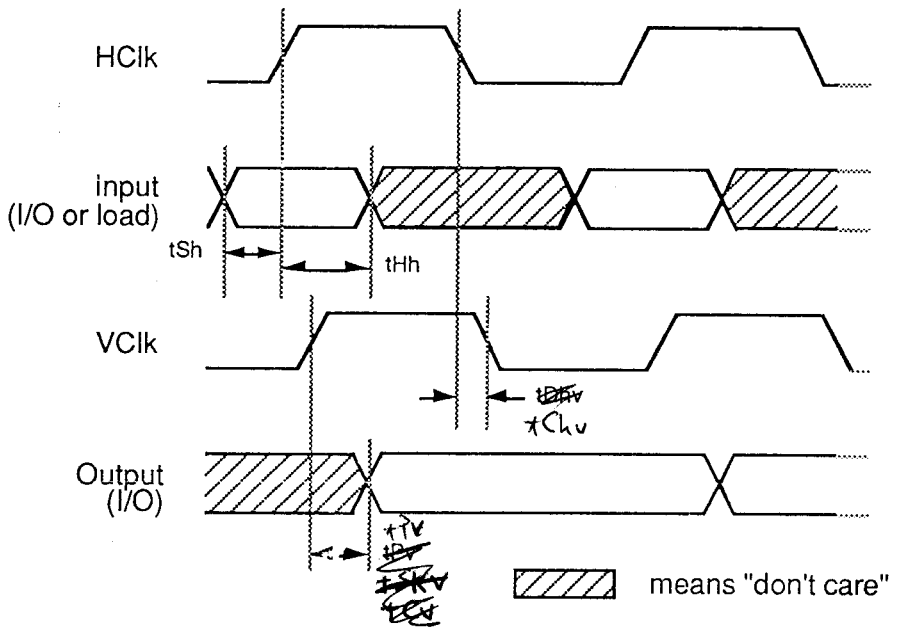



**Figure IV.4.8.3 Definition of timing parameters (H to V direction shown only)**

## Switching Characteristics

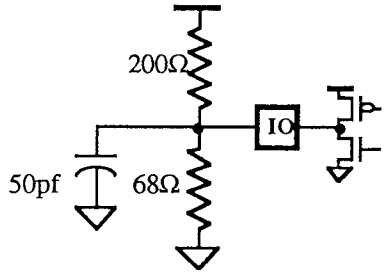
| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tEnh, tEnv | Output enable time | | 2.1 | | 8.2 | nsec |
| tDish, tDisv | Output disable time | | 2.5 | | 10.0 | nsec |
| tPh, tPv | Clock to output delay | ttl load (1) | 5.6 | | 19.3 | nsec |
| tPh, tPv | Clock to output delay | 50 pf load | 6.4 | | 22.8 | nsec |

## AC Operating Requirements

| Parameter | | Conditions | Min. | Nom. | Max. | Units |
|---|---|---|---|---|---|---|
| tSh, tSv | Input setup time | | 2.0 | | | nsec |
| tHh, tHv | Input hold time | | 8.0 | | | nsec |
| tSkhv | Clock skew from HClk to VClk | | | | 8.0 | nsec |
| tSkvh | Clock skew from VClk to HClk | | | | 8.0 | nsec |

Notes:

(1) TTL output load modeled by following circuit:



Lower lurker

proceeds counterclockwise around the die.) The columns of the grid are labelled alphabetically and the rows numerically. Pin #10 of this package is internally wired to the bottom of the cavity, and therefore to the silicon substrate.

Results

| | L | K | J | H | G | F | E | D | C | B | A | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 21 | 19 | 18 | 16 | 13 | 12 | 9 | 6 | 4 | 3 | 84 | 1 |
| 2 | 24 | 22 | 20 | 17 | 14 | 7 | 8 | 5 | 2 | 1 | 82 | 2 |
| 3 | 25 | 23 | | | 15 | 11 | 10 | | | 83 | 81 | 3 |
| 4 | 27 | 26 | | | | | * | | | 80 | 79 | 4 |
| 5 | 30 | 29 | 31 | | | | | | 75 | 77 | 76 | 5 |
| 6 | 33 | 28 | 32 | | | | | | 74 | 73 | 78 | 6 |
| 7 | 34 | 35 | 36 | | | | | | 70 | 71 | 72 | 7 |
| 8 | 37 | 38 | | | | | | | | 68 | 69 | 8 |
| 9 | 39 | 41 | | | 49 | 53 | 54 | | | 65 | 67 | 9 |
| 10 | 40 | 43 | 44 | 47 | 50 | 52 | 56 | 59 | 62 | 64 | 66 | 10 |
| 11 | 42 | 45 | 46 | 48 | 51 | 57 | 55 | 58 | 60 | 61 | 63 | 11 |
| | L | K | J | H | G | F | E | D | C | B | A | |

Pad to Pin Layout -- TOP View

Caution: the pad-to-pin layout varies for different 84 pin products, even those from a single vendor. For our purposes, equivalent parts have the same pin grid, the same pad to pin layout, and the same pad to cavity (substrate) connection. Be sure to use this diagram for MOSIS-supplied parts unless or until instructed otherwise.