Report Number: WUCS-88-32

1988-10-01

Design of a VLSI Packet Buffer

Neil Barrett

This paper describes the design of the Packet Buffer Chip. Packet Buffers are FIFO queues used for buffering packets and synchronizing a Switch Fabric (SF) and its associated Fiber Optic Links (FOLS) in the Broadcast Packet Switching Network. This chip will be fabricated in 2.0 UM CMOS technology.

DESIGN OF A VLSI PACKET BUFFER

Neil Barrett

WUCS-88-32

October 1988

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Abstract

This paper describes the design of a Packet Buffer Chip. Packet Buffers are FIFO queues used for buffering packets and synchronizing a Switch Fabric (SF) and its associated Fiber Optic Links (FOLS) in a Broadcast Packet Switching Network. This chip will be fabricated in 2.0 μm CMOS technology.

This work supported by Bell Communications Research, Bell Northern Research, Italtel SIT, and National Science Foundation grant DCI-8600947.

Contents

List of Figures

DESIGN OF A VLSI PACKET BUFFER

Neil Barrett
neil@wuccrc.WUSTL.EDU

1. Introduction

This paper describes the design of the Packet Buffer Chip for use within a broadcast packet switching network. This network is described in [Tu85] and [Tu86].

1.1. Packet Switch

The overall structure of the prototype packet switch is shown in Figure 1. The switch terminates 7 fiber optic links. A production version would be much larger (probably terminating 63 fiber optic links). The packet switch is divided into five major components. These are the connection processor (CP), the Packet Processors (PPs), the Copy Network, the Broadcast Translators and the Routing Network. These are shown in Figure 1. The Copy Network, Broadcast Translators, and Routing Network are collectively known as the Switch Fabric (SF), and are constructed from banyan-interconnected Switch Elements. The CP is responsible for establishing connections. It exchanges control information with CPs in neighboring Switch Modules and controls the actions of the PPs

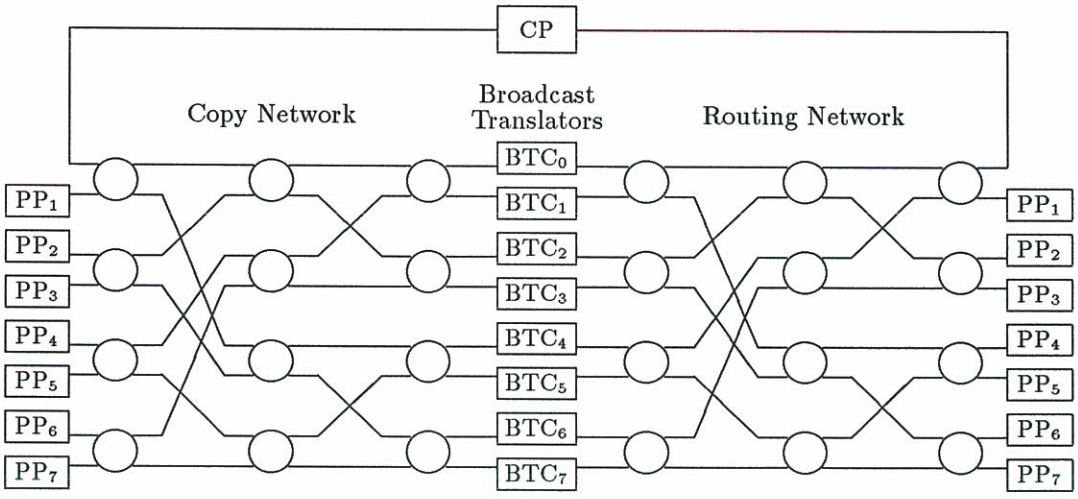

Figure 1: Switch Module

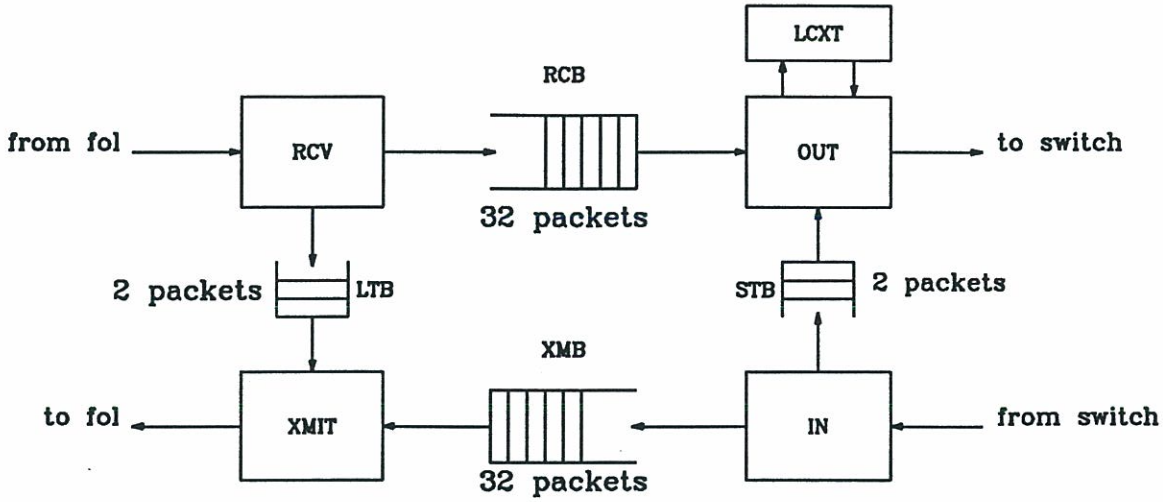

Figure 2: Packet Processor

and the SF. The PPs serve as buffers between the transmission links and also perform the address translation required by routing. The Copy Network's function is to make copies of broadcast packets as they pass through it. Point-to-point packets pass through the Copy Network without change. When copies of a broadcast packet emerge from the Copy Network, their final destinations are yet to be determined. The Broadcast Translators assign a new routing field to each copy of a broadcast packet in such a way that a copy is received by every PP that is supposed to receive one. The Routing Network then routes the packets to the appropriate output link based on information encoded in the routing field of the packet.

1.2. Packet Processor

The PPs perform the link level protocol functions in the packet switch, including the determination of how each packet is to be routed. The components of the PP are shown in Figure 2.

There are two primary packet formats used in the switch: external and internal. Packets are carried in external format on the fiber optic links connecting switches, and in internal format within each switch. The PP translates between these two formats. On entering the switch module a packet is reformatted by the addition of a Routing field, a Control field and a Source field. The Control field identifies the various kinds of control packets, while the Source field identifies the origin of the packet. The Routing field contains information required to route the packet through the switch fabric. For the purposes of this document the details of all the fields are of no direct concern and have therefore not been included.

There are 9 major components in the PP. A quick description of each one follows:

- *Receive Module* (RCV). This module interfaces the PP to the incomming byte stream from the Receive Link Interface chip. It discards frames containing errors, and routes test packets to the *Link Test Buffer* and other packets to the *Receive Buffer*.

- *Receive Buffer* (RCB). A FIFO packet buffer which can hold 32 external format packets. It buffers the packets coming from the fiber optic link which are destined for the switch fabric.
- *Link Test Buffer* (LTB). A FIFO buffer which can hold 2 external format packets. This buffer is used to test the operation of the FOL and is mainly used in the diagnostic mode.
- *Output Circuit* (OUT). Selects packets from the RCB and STB with the STB given higher priority. The external packet from the RCB is converted to internal format, by the addition of a header to the packet. The header is determined by a lookup into a *Logical Channel Translation Table*. The packet is then transmitted to the SF. Control packets and updates to the LCXT memory come from the STB and are processed appropriately.
- *Switch Test Buffer* (STB). A FIFO buffer which can hold 2 internal packets. This buffer is used to test the operation of the SF and is mainly used in the diagnostic mode. In addition it passes control packets to OUT.
- *Logical Channel Translation Table* (LCXT). This is a lookup table that holds the routing information. The PP assigns new packet headers depending on the LCXT entries.
- *Input Circuit* (IN). This module routes data packets to the *Transmit Buffer*, removing the internal packet header in the process. It also routes control packets and switch test packets to the STB.
- *Transmit Buffer* (XMB). A FIFO buffer which can hold 32 external packets. It buffers the packets coming from the switch fabric that are destined for the fiber optic link.
- *Transmit Module* (XMIT). Selects packets from the XMB and LTB with the LTB given higher priority. This module computes and adds the checksum field to packets going out on the FOL.

1.3. Packet Buffer

This paper concentrates on the design and implementation of the four packet buffers used in the PP. A packet buffer is a first-in-first-out buffer for storing packets. During a given packet cycle or epoch, one packet can be written to the buffer and one read out. There is approximately a two fold increase in speed for packets going from the FOL to the SF. The packet buffers then serve as a means of synchronizing the FOL with the SF.

The motivation for a VLSI implementation of the packet buffer as opposed to using off the shelf components is two fold. Firstly, the interface of commercially available FIFOs [TI86] [TI87] is not easily conformable to the other VLSI chips of the PP. Additional components would have to be added to make them functionally equivalent to a packet buffer. Also most high speed FIFOs have limited memory space (on the order of 1024 words) and are restricted to an input data rate of around $22MHz$ word serial [TI87]. Secondly, if we had implemented a packet buffer using discrete components such as an EPLD, a general purpose RAM, and other logic, the chip count for an individual packet buffer would range from somewhere between 10 to 20 chips. For the prototype packet switch which contains 7 PPs with 4 buffers/PP (Fig. 1, Fig. 2), this would bring the total chip count due to buffers to between 280 and 560 chips. If on the other hand, we use a custom VLSI approach, the two test buffers could be integrated along with the other components of the PP (for example the LTB with RCV, XMIT, and the STB with IN, OUT) thus leaving only the two larger buffers as separate chips. With this custom VLSI approach the total chip count due to buffers alone is 14.

In order to reduce the effort involved in the implementation of four packet buffers, a tool for the automatic generation of the circuits was developed. The tool, implemented as a C program takes as input parameters the buffer size, packet size, input and output data-path width, target

clock speeds, and some timing information as a function of the input parameters. The output of the program is the mask level description of the circuit. The advantage of this kind of tool is that it allows the designer to experiment with various circuit architectures, and gives some estimate of various performance issues, since the time consuming chore of mask layout is done by the computer.

For the prototype SM we are building the parameters in Table 1 are used for the packet buffers.

	Clock Speed			Buffer Size	Packet Size	Data Path	Write
	In	Out	Internal	Packets	Words	In, Out (bits)	
RCB	FOL	SF	SF	32	75	9, 9	ASYNC.
XMB	SF	SF/2	SF	32	75	9, 9	SYNC.
LTB	FOL	SF/2	SF	2	75	9, 9	ASYNC.
STB	SF	SF	SF	2	80	9, 9	SYNC.

Table 1: Buffer Parameters

In the column labelled Clock Speed, FOL speed refers to the byte serial data rate on the receiving end of the PP and is roughly around $12.5MHz$. The SF byte serial data speed is estimated at approximately $25MHz$. The packet buffer, however, has been designed to operate with link speeds of up to $25MHz$ or $200Mbps$ and SF data speeds of up to $50MHz$.

The packet buffer supports different clock speeds on the input and the output sides, as well as, asynchronous input. This is accomplished by having three distinct clocks. An internal clock determines the basic operating cycle of the buffer. During one internal cycle the buffer may be read once and written to once. The output clock may operate at the same rate as the input clock, or some integral divisor of the internal rate (in our case half). Note that while the output clock may operate at a different rate from the internal clock, the operation of the buffer on the output side is fully synchronous. On the input side however, asynchronous operation is allowed. A separate clock is allowed to shift in a packet at the input rate. The only constraint placed on the input side when operated asynchronously, is that the input clock rate be no more than about half the internal clock. The input side may be operated synchronously with the internal clock, in which case it may operate at the same rate as the internal clock.

For the rest of this document we concentrate mostly on the RCB as the the other buffers are quite similar. Analysis of the RCB in comparison to the XMB, STB is more interesting since the buffer write is asynchronous, and in comparison to the LTB most of the arguments apply directly. The sections of this paper are organized as follows:

- Section 2 presents the chip specification.
- Section 3 gives a high level description.
- Section 4 contains a detailed circuit level description. Here we examine the choice of circuits and their performance.
- Section 5 describes the tools used to make the design effort easier.
- Section 6 discusses the built in test mechanisms, as well as some thoughts on testing the first implementation of the packet buffer.
- Appendices A and B provide pin assignments and a brief description of each pin's function.

2. Chip Specification

2.1. Operation

The packet buffer, which consists of slots where packets may be stored for future use, serves as a storage and synchronizing element between the FOL and the SF. Packets may only be accessed in a first-in-first-out (FIFO) manner. The proper size of the buffers and the frequency of writing to and reading from the buffers are dependent on the traffic pattern in the system [Tu86].

A packet may be written to a buffer when the buffer is not full. Writing a packet to an already full buffer is not permitted. To write a packet into a buffer, it must first be shifted into a serial register within the buffer byte by byte at the input clock rate. Once the packet has been shifted in, the input side must generate a SYNC or write pulse. This write signal is asynchronous with respect to the internal clock and the output side of the buffer. The asynchronous write signal is used to latch the packet into an internal register and signal its presence to the packet buffer control circuit. A new packet may then be shifted in.

The packet buffer being a FIFO, always tries to output the oldest packet it contains. Packets are shifted out at the output clock rate. A packet cycle (a packet cycle is the time necessary to process a packet) is initiated by synchronizing the internal clock with the down stream clock. The synchronizing signal comes from the modules downstream. At the appropriate time during the packet cycle the oldest packet in the buffer is shifted out byte serially. The packet buffer then waits for 16 clock cycles after shifting out the first byte to receive an acknowledgement from the module downstream indicating that the packet is being read. The 16 clock cycle wait time corresponds to the time necessary for a module downstream to process the header of a packet. For example, the OUT circuit of the PP can receive packets from either of 2 buffers – the RCB and the STB (Fig. 2). Since the STB has higher priority, an acknowledgement will be sent to the STB in the event of a tie and the RCB will have to resend its packet on the next cycle. If the buffer is empty, an empty signal is asserted and zeros (a NULL packet) are shifted out. A global reset causes all stored packets to be discarded and the control circuitry is restored to the starting state.

There are three clocks associated with a packet buffer. They are the input clock, $u\phi_1, u\phi_2$, upstream phase 1 and 2; the internal clock, $i\phi_1, i\phi_2$, internal phase 1 and 2; and the output clock, $d\phi_1, d\phi_2$, downstream phase 1 and 2. Column 2 of Table 1 in Section 1 shows the clock relationships. There are two restrictions on the relationship between the three clocks. The first restriction is that the internal clock rate must be either greater than twice the input clock rate ($(i\phi_1, i\phi_2) \geq 2 \times (u\phi_1, u\phi_2)$) or the input must be identical to the internal clock. The second restriction is that the internal clock rate must be an integer multiple of the output clock rate ($(i\phi_1, i\phi_2) = \alpha \times (u\phi_1, u\phi_2); \alpha \in I$). The first restriction is imposed to reduce the probability of metastable behavior due to asynchronous writing to an infinitesimal level. The second restriction was imposed merely to simplify implementation of the circuit.

2.2. Packet Buffer Interface

The external interface signals of the packet buffer are shown in Fig. 3 and are described briefly below.

- *Upstream input data* ($ud_0..ud_9$). Incomming data from upstream modules; consists of 8 data bits and 1 parity bit at the $u\phi_1$, $u\phi_2$ clock rate (stable $u\phi_1$).
- *Asynchronous write* (SYNC). An asynchronous write signal that causes the buffer to latch in the packet (stable $u\phi_1$).

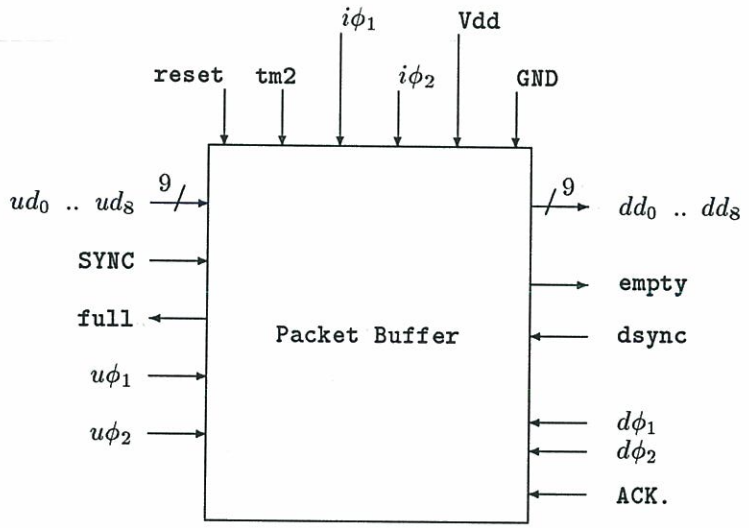

Figure 3: Packet Buffer Interface

- *Buffer full* (`full`). Buffer full alert to the module upstream.
- *Input Clock* ($u\phi_1, u\phi_2$). Upstream 2 phase clock.
- *Global Reset* (`reset`). Global reset signal; causes all stored packets to be discarded and internal circuitry to be initialized.
- *Start of Packet Cycle* (`tm2`). A signal to indicate the start of the internal packet cycle.
- *Internal Clock* ($i\phi_1, i\phi_2$). Internal 2 phase clock.
- *Power and Ground* (`Vdd, GND`).
- *Downstream output data* ($dd_0..dd_9$). Outgoing data to downstream modules at the $d\phi_1$, $d\phi_2$ clock rate (stable $d\phi_1$).
- *Buffer empty* (`empty`). Buffer empty alert to the module downstream (stable $i\phi_2$).
- *Downstream Synchronization* (`dsync`). Signifies which downstream packet cycle to begin shifting out a packet. For example, when there is a factor of two difference between the operating speeds of the SF and the FOL, `dsync` will be asserted during every other packet cycle.
- *Downstream Clock* ($d\phi_1, d\phi_2$). Downstream 2 phase clock.
- *Downstream acknowledgement* (`ACK.`) Acknowledgement from downstream module indicating that a packet has been read out of the buffer.

Fig. 4 shows the timing relationship between the clocks and the data. Notice that data must be stable during ϕ_1 and may change during ϕ_2.

Fig. 5 shows the relationship between the internal and the downstream clocks for $i\phi = 2 \times d\phi$. Note that the duty cycle remains the same, and that only the period of $d\phi$ is increased. The pulse width of ϕ_1 was chosen to give a 33% duty cycle since ϕ_1 is usually qualified with other control

signals and used to latch data. The choice of pulse width for ϕ_2 was limited once the pulse width of ϕ_1 was decided on. For a target clock speed of 42 MHz, one clock period lasts 24 ns. With ϕ_1 being 8 ns, this leaves 16 ns for the interphase gaps (ϕ_{12} and ϕ_{21}) and for ϕ_2. However, a minimum ϕ_{12} of 6 ns is needed at that speed in order to discharge nodes. This translates to ϕ_2 being 4 ns or a 17% duty cycle.

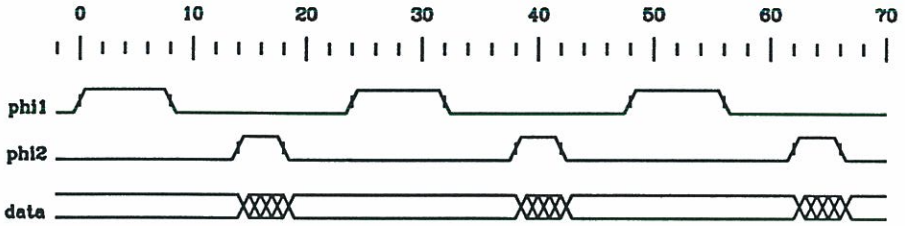

Figure 4: Input and Output Timing

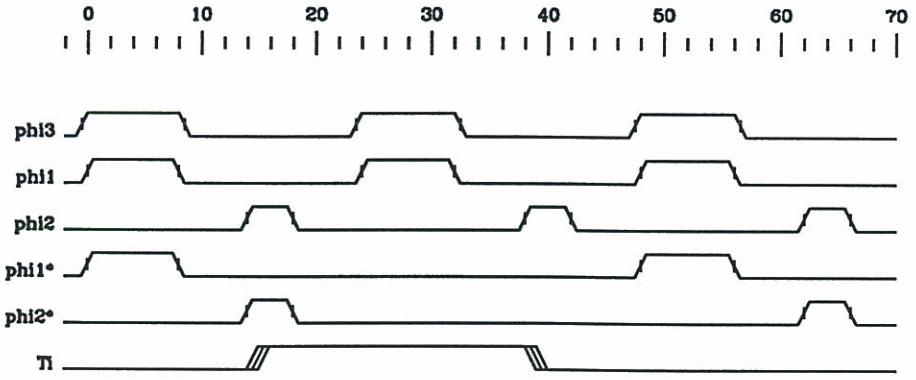

Figure 5: Internal and Downstream Clock relationship

3. High Level Design

The packet buffer may be divided into three basic sections: the memory, the pointers, and the associated control circuitry. A block diagram is shown in Fig. 6.

3.1. Memory

The memory is used to store packets within the buffer. Each bit of the input byte stream ($ud_0..ud_8$) has an associated memory plane. Each plane consists of:

1. A serial to parallel input shift register (SIPO) clocked by $u\phi_1$, $u\phi_2$.
2. A second stage of buffering (MEM LATCH) is used to support asynchronous writing of packets into the buffer.
3. Following the MEM LATCH is the actual memory array where the bits of the packet are stored.
4. A parallel to serial (PISO) output shift register clocked at $d\phi_1$, $d\phi_2$ rate is used to output the packet on the $dd_0..dd_8$ leads.

3.2. Pointers

The pointer circuitry directs which slot to read a packet out of and which slot to write a packet into. The pointers consist of a read pointer, a write pointer and a current capacity pointer. The read pointer, which points to the current slot from where a packet will be output, is incremented every time an `ACK` signal is received. The write pointer, which points to the next available slot, is incremented every time the asynchronous write is asserted. A third pointer, termed current capacity, contains the capacity of the buffer at any particular time. When the buffer is full, a signal is sent to upstream modules indicating the buffer status and thus avoiding buffer overflow. An attempt to write a packet to an already full buffer results in the discarding of that packet. Similarly, when the buffer is empty, an appropriate signal is sent downstream. The current capacity pointer is incremented every time a packet is written to the buffer and it is decremented every time one is read out.

3.3. Control

The control circuit generates the appropriate timing signals for the coordination of the pointers and the memory planes. A packet cycle starts two internal clock ticks after `tm2` is asserted. The packet buffer operates on a fixed packet cycle with control signals asserted at specific points within the packet cycle. The control circuit keeps track of clock ticks within a packet cycle and asserts signals, required by the memory and the pointers, depending on the number of elapsed ticks since the start of the cycle and the status of the buffer.

A second part of the control circuit shown in Fig. 6 is the asynchronous write capture circuit. It consists of the S/R flip flop and the D latch to the left of the main control block. When an asynchronous write occurs it sets the S/R flip flop; the main control circuit samples the output of the S/R flip flop via the D-latch at a fixed time within the packet cycle. Note, a potential synchronizer problem exists here, since the sampling of the D-latch and the asynchronous write may arrive at the same time causing metastability. The time between the sampling of the signal and its actual use within the circuit is much greater than the minimum resolving time. The following

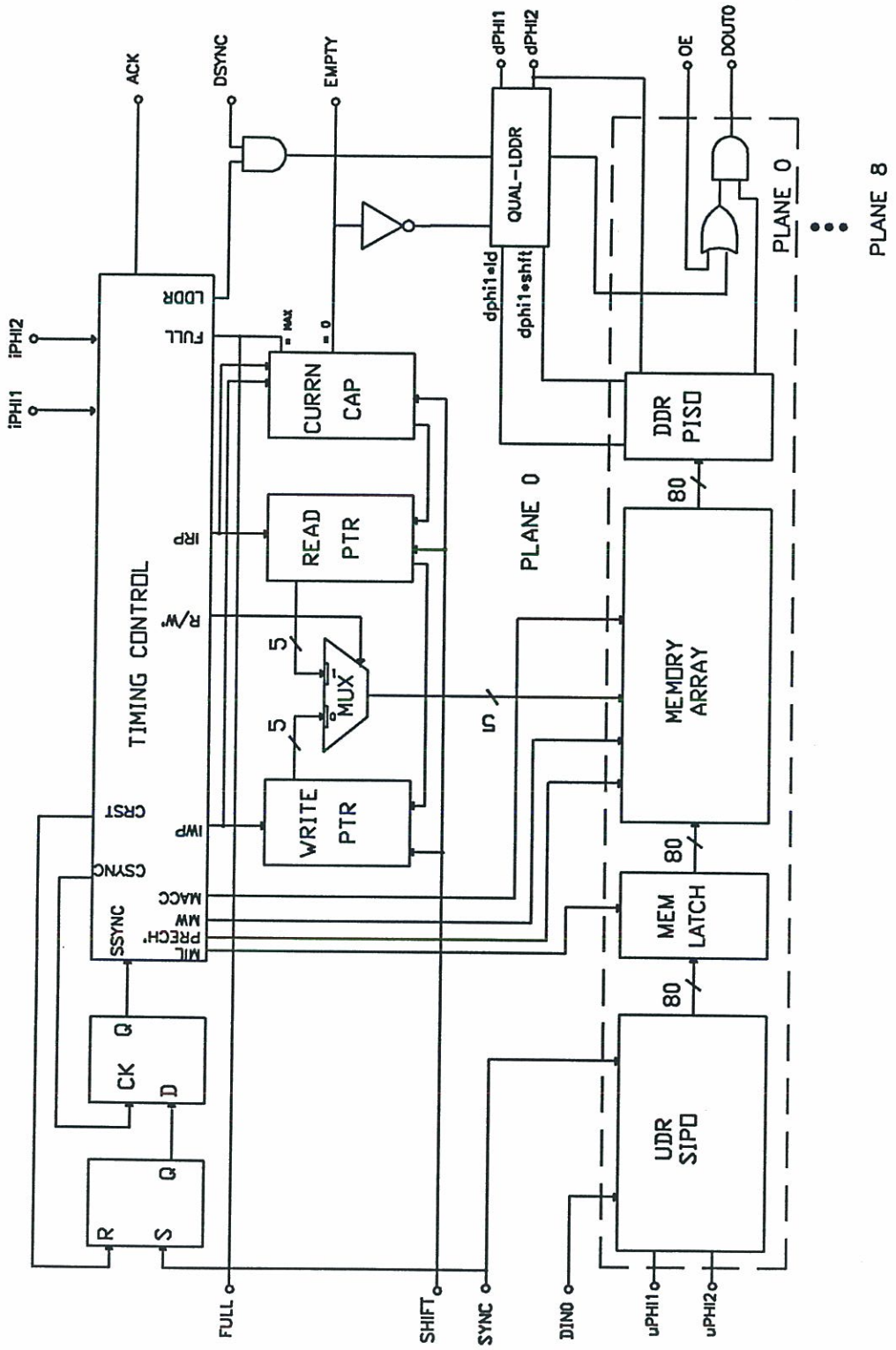

Figure 6: Block diagram of the Packet Buffer

analysis, adapted from [RoMo86], shows an estimate of the expected failure rate. $P_u(\tau_x)$, the probability that a signal is unresolved at time τ_x is :

$$P_u(\tau_x) = \textit{Failure Rate/Event Rate.}$$

Also,

$$P_u(\tau_x) = \frac{2\tau_r e^{-\tau_x/\tau_R}}{5T_c},$$

where τ_r is the maximum rise/fall time of the event, T_c is the system clock rate, τ_R is the recovery time. So we can design our system with a specific failure rate (say only once per year) and a given EVENT/write rate (one every $2.54\mu S$). For a failure rate of 1 per year and an event rate of 1 every $2540nS$,

$$P_u(\tau_x) = \frac{1/yr}{0.394 \times 10^6\ /sec} = 8.05{\times}10^{-14},$$

solving for τ_x in the above equation, gives

$$\tau_x = -\tau_R\ ln(P_u(\tau_x)T_c/\tau_r)$$

Using, $\tau_r = 2nS$, $T_c = 20nS$, $\tau_R = 1.5nS$, gives a τ_x of $42nS$. Note, this τ_x is for a failure rate of once per year. In our design we made $\tau_x \sim 300nS$, thus making the mean time between failures almost insignificant.

A third component of the control circuit is the block labelled QUAL-LDDR. This module qualifies the loading of the PISO output register. Furthermore, if the buffer is empty it causes a NULL packet to be shifted out.

3.4. Global Timing

The major control signals shown in the block diagram of Fig. 6 are described below. The timing relationships between these signals is shown in Fig. 7. The single packet cycle shown is divided up into 86 clock ticks, where 1 clock tick represents a $(i\phi_1, i\phi_2)$ cycle; 75 of the 86 clock ticks consist of the time necessary to shift in a packet while the remaining 11 represent the minimum dead time.

The major control signals and their function are described below:

- *Precharge* (`prech`). Precharge is used to precharge dynamic circuitry in address decoders of the memory and in the case of the read cycle to prepare internal memory lines.
- *Memory Access* (`macc`). `Macc` enables the decoding of the address lines. During the interval when `macc` is asserted the contents of the memory elements are accessed.
- *Load downstream data register* (`lddr`). This signal is asserted in order to load the output PISO register from the memory. It is qualified by the `dsync` lead and by the block QUAL-LDDR. QUAL-LDDR latches the `lddr` signal, waits till the next $d\phi_1$ and then gates it with this latched signal before loading the output PISO. The qualified load signal is also used to clock a D-latch which in turn qualifies the output data. `Lddr` occurs towards the end of the read cycle, thus allowing the maximum settling time for the internal data lines of the memory before being accessed.
- *Memory In Latch* (`mil`). This signal causes the memory latch to latch in data from the input SIPO. The signal is asserted at the beginning of the write cycle to allow maximum stability of the input data on the internal memory lines. Note, `mil` will only be asserted if (i) a `SYNC` was observed earlier within the packet cycle and (ii) the buffer is not full.

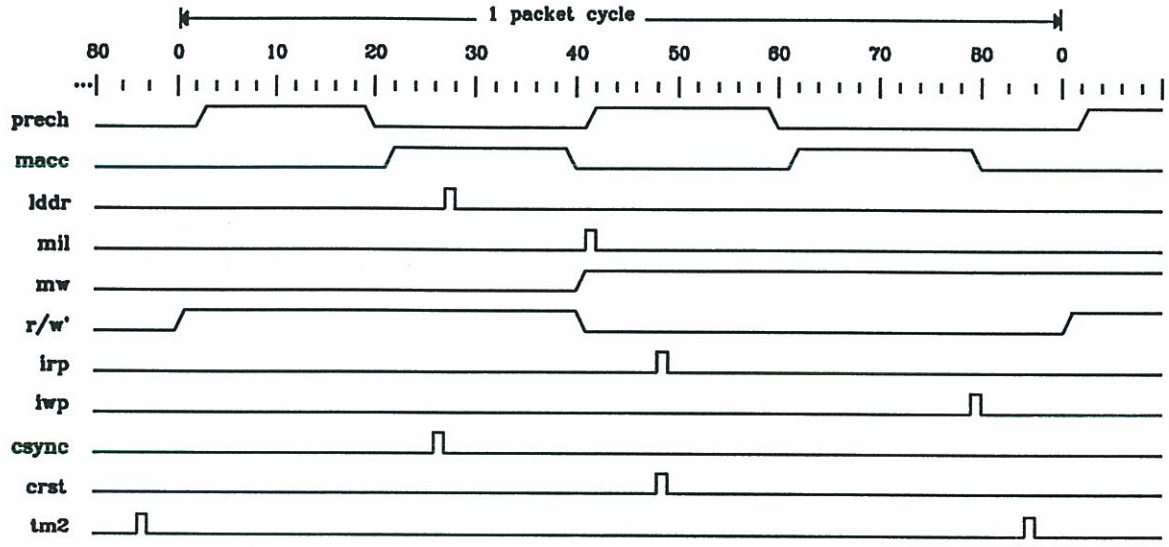

Figure 7: Packet Buffer Timing

- *Memory Write* (`mw`). Memory write is asserted only if the sampled `SYNC` line has been asserted. If `mw` is high then the data latched in from the memory latch is written to an empty slot in memory.
- *Read/Write* (`r`/$\overline{\texttt{w}}$). When (`r`/$\overline{\texttt{w}}$) is high the packet buffer is in the read half of the packet cycle and when low in the write half.
- *Increment read pointer* (`irp`). This signal increments the read pointer. The read pointer is incremented only if a read `ACK` is received within 16 clock cycles after the first byte of the packet is shifted out. If an `ACK` is not received within the allotted time, the same packet is output on the following packet cycle.
- *Increment write pointer* (`iwp`). `Iwp` is used to increment the write pointer. The write pointer is incremented toward the end of the write cycle provided a `SYNC` occurred earlier on within the packet cycle and provided the buffer is not full.
- *Check Sync.* (`csync`). Check Sync is used to check if the asynchronous write occurred.
- *Reset Sync.* (`crst`). This signal is used to reset the `SYNC` capture circuit, provided a `SYNC` was seen and processed.

Sec. 5.3 contains a list of the control signals, when they are asserted/negated in a given packet cycle and what signals qualify them.

4. Medium Level Design

4.1. Integrated circuit design

The circuits for the packet buffer were designed using a set of CAD tools from the University of California at Berkley described in [ScMa86]. The circuit layout tool used was **MAGIC**, which is a graphical layout editor supporting a hierarchical design style. Technology based design rules are checked by **MAGIC**, and are based on a Mead-Conway design style with $\lambda = 1.0\mu m$ [MeCo80]. This results in a minimum line width of $2\mu m$. Chip fabrication is a bulk n-well, 2 layer metal CMOS process, using facilities provided by **MOSIS** [USCa].

Circuit operation was verified using both logic level and circuit level simulators. Logic level simulation was executed using **ESIM** [ScMa86], and more detailed circuit level simulation was performed using both **FACTS** [MCNC] and **SPICE3** [VlZh]. To aid in testing, *PBUFVEC*, a C program that automatically generates test vectors was written. In order to verify correct chip operation the packet buffer must be simulated over several packet cycles. Each packet cycle consists of 86 clock ticks where each clock tick requires 8 simulation steps. For a moderate size simulation run the required test vectors can become extremely long and human entry of these vectors is tedious and error prone. *PBUFVEC* automatically generates the input test vectors, as well as, the expected results. A simple file comparison may then be used to verify circuit operation. Additionally, in order to download test vectors to our chip testing station, pre and post-processors to **ESIM** were written. The chip testing station is a Tektronix DAS9100. Since the DAS9100 does not generate two phase clocks, a clock from the DAS is used to externally derive a 2 phase clock (using delay lines). The external clocks are then fed to the chip being tested.

4.2. Memory

A 6-transistor static RAM cell (SRAM) having an area of $1198\lambda^2$ was used as the basic memory element. For a Manhattan based design style, we found the area occupied by a 3-transistor Dynamic RAM cell (DRAM) to be roughly 1/2 the area of the SRAM. Although this is a significant decrease in total area, we felt that the additional circuitry in the form of refresh controllers and complexity in the form of refresh timing involved with DRAMS did not warrant their use [Re83]. The memory architecture adopted is shown in Fig. 8. The memory arrays are on either side of the row decoders. This layout scheme minimizes the delay associated with long polysilicon word lines. Long runs of polysilicon cause delays and thus make the memory slower. Another feature of this memory is that there is no column decoder. This is because in a particular write cycle we write an entire packet into a row of memory. There is never any need to split up a packet to be able to write it to memory and so distinguishing between individual columns is unnecessary. Dynamic circuits, which inherently occupy less area than static circuits, have been used to implement the row-decoder and the various latches. A more detailed version of the architecture, which shows the actual circuits within each of the blocks, is presented in Fig. 9. The operation of each of the blocks will now be discussed.

4.2.1. Implementation and Architecture.

- Serial-In Parallel-Out input shift register.

  The SIPO was implemented as a dynamic shift register which consists of a chain of clocked inverters. A single stage of the chain contains 2 clocked inverters with the first inverter clocked on $u\phi_1$ and the second on $u\phi_2$. During the interval between the two phases ($u\phi_{12}$) data is stored on the capacitance of the internal node.

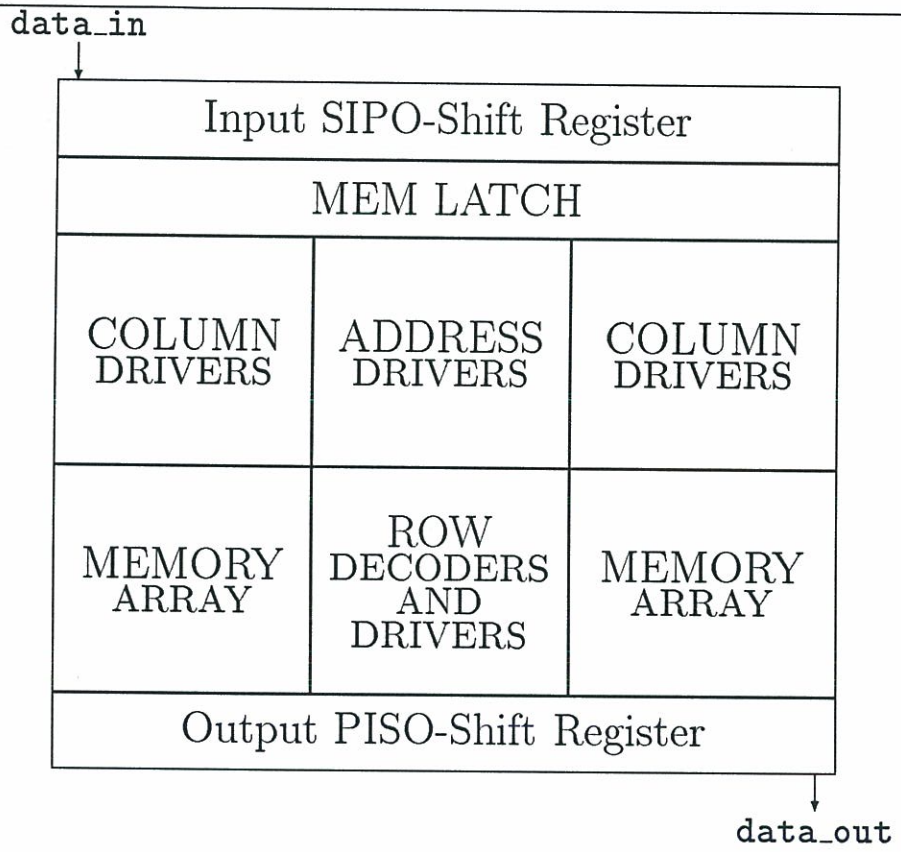

Figure 8: Floor plan of 1 bit plane of memory

The upper limit of operation is dependent on the delay through a clocked inverter and is on the order of $3nS$ per clocked inverter. This delay together with the time necessary to charge the internal node restricts the maximum operation frequency to $\simeq 125MHz$.

Because we are using a dynamic shift register, the minimum operating frequency is also important. For a maximum leakage current of $1nA$ (typical leakage currents are $0.1nA$)[WeEs85], and a capacitance of $0.01pF$ on the internal node, this corresponds to $215\mu S$ before a stored high will degrade to a voltage less than the turn on voltage of the second inverter. This sets the lower operating frequency, due to the SIPO alone, to $\simeq 4.6KHz$.

- Memory Latches.

  Once the `SYNC` signal is asserted, data from the SIPO is transferred to another dynamic latch. This second stage is clocked by `mil`.

  A worst case analysis for minimum and maximum speed of operation is presented pictorially in Fig. 10 and the argument is as follows:

  - Assume a `SYNC` occurs right after `csync`, the sampling signal, in some packet cycle.
  - Data will be then be transferred to the input of the second stage of latches, but not to the column drivers.
  - However, `mil` will not be asserted since a valid `SYNC` was not detected within the current packet cycle.

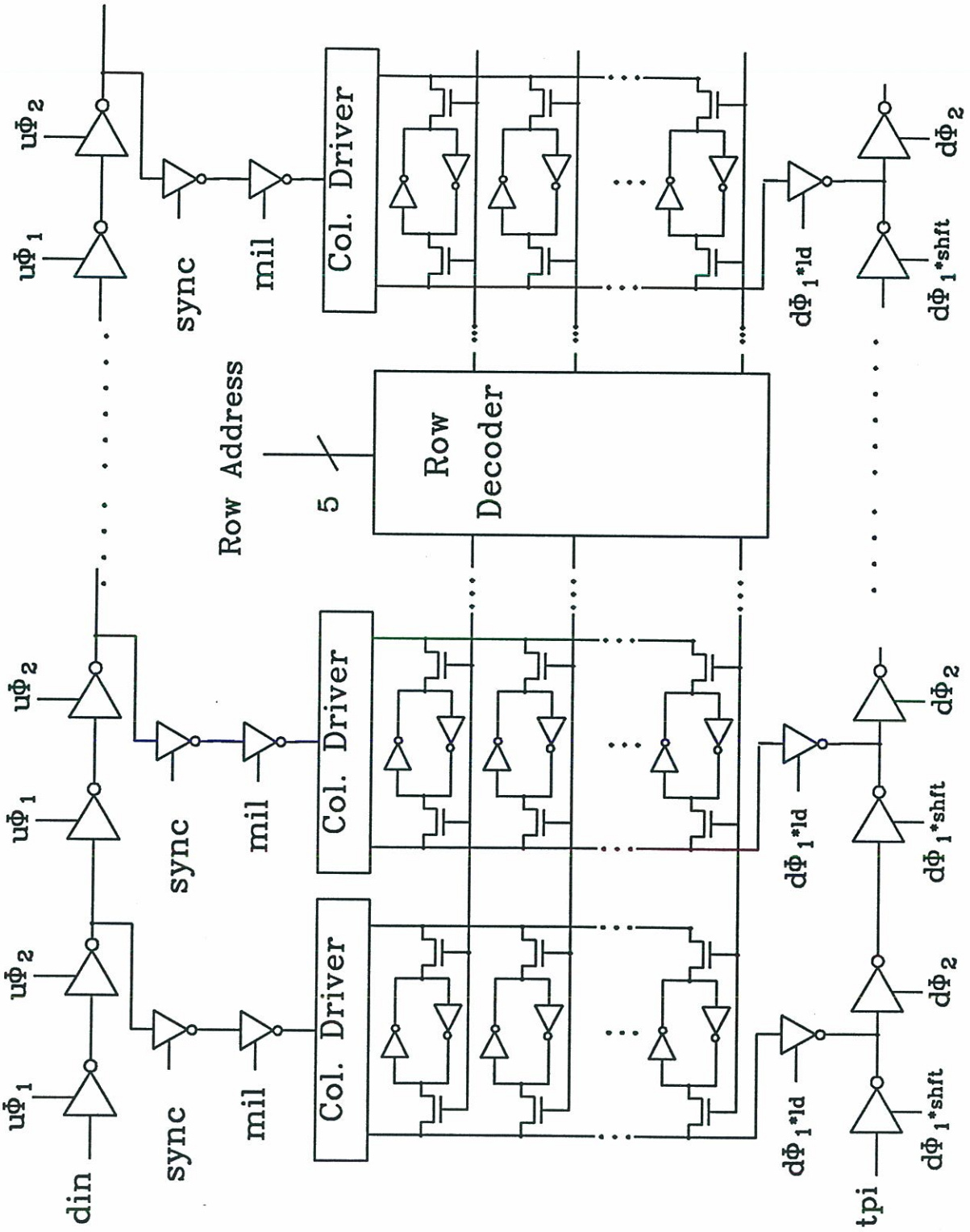

Figure 9: Expanded view of circuits used in a memory plane

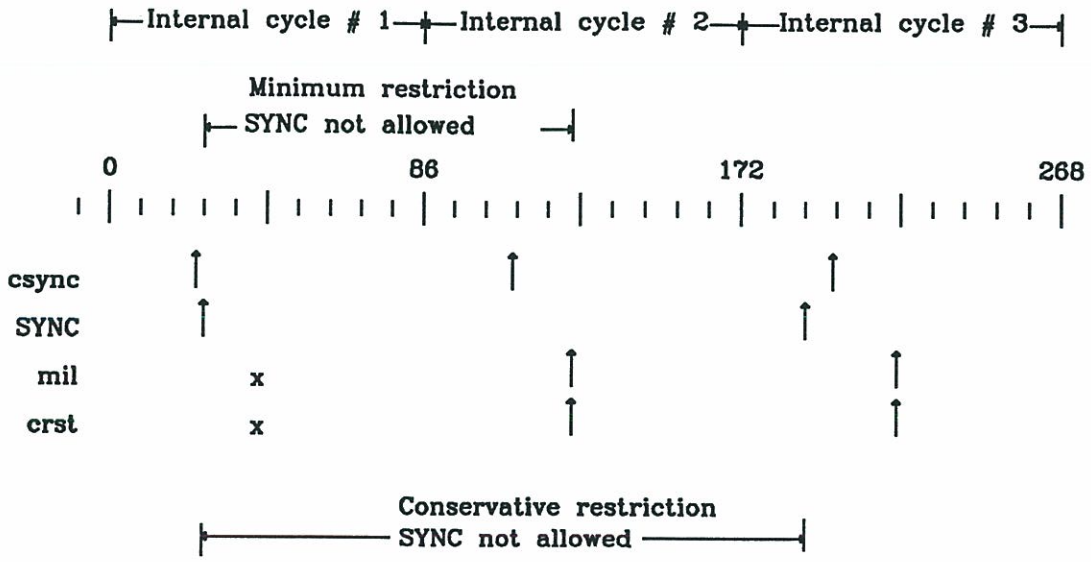

Figure 10: Restriction on write/SYNC pulses

- Since the SYNC was missed on the current packet cycle, the buffer will have to wait an entire packet cycle before writing in the original packet.
- Therefore, we must guarantee that we are finished processing the first SYNC before we accept another packet, i.e. only after the sync-capture circuit has been reset (crst).
- Data will then have to be held in the first set of latches for

$$1\ packet\ cycle\ \times\ 75\ ticks/cycle \times\ 40\ nS/tick\ \sim\ 3.00\mu S.$$

The minimum speed the dynamic latch can be run at must satisfy the following equation,

$$75\ ticks/cycle\ \times\ P\ nS/tick\ \ \leq\ 215\mu S,$$

where $215\mu S$ is the time required for the signal to degrade to an unusable level. Thus, the minimum speed of operation is restricted to $P = 2.86\mu S/tick$, implying a frequency of $350KHz$. The maximum operating frequency is determined by the delay characteristics of the dynamic latch and is restricted to roughly $125MHz$.

The analysis above also explains why we impose the restriction that the internal clock be twice as fast as the input clock. Assuming the above scenario (Fig. 10), and an internal clock speed of $50MHz$, we can say that SYNC pulses must arrive at least more that 127 ticks of the internal clock apart. This corresponds to $2540ns$, and in turn represents $75 + 6$ ticks of the upstream clock. The additional 6 ticks are due to a non-zero inter packet gap. Thus, the upstream clock rate is forced to be no more than $32MHz$. This corresponds to a $1.5\times$ speed difference; as a safety margin we decided to restrict it to 2. If the internal clock and the input clock are synchronous then we know exactly when a SYNC will arrive and so the factor of 2 speed difference is not necessary.

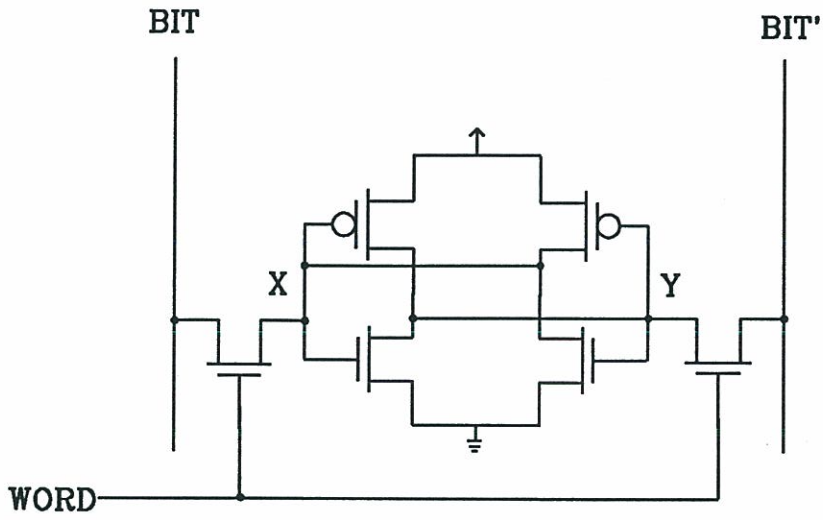

Figure 11: A 6 Transistor Static RAM cell

- Memory Cell.

  Fig. 11 illustrates the 6-transistor Static RAM cell which consists of a pair of cross coupled inverters connected by pass transistors to a BIT and $\overline{BIT}$ line. The memory cell contents are stored at points X and Y in Fig. 11. To write data to the cell, $DATA$ and $\overline{DATA}$ are driven onto BIT and $\overline{BIT}$ respectively. Next the $WORD$ line is asserted. The stronger transistors driving $DATA$ and $\overline{DATA}$ overcome the weak transistors of the memory cell and cause it to change state. To read a data value out of the memory cell, both BIT and $\overline{BIT}$ are precharged high. The word line is then asserted causing the internal nodes of the memory cell to be directly connected to the bit lines via a pass transistor. The pull-down transistors of the memory cell then discharge one of the bit lines while leaving the other one high. The word line is then turned off and the data on the bit lines can be latched out.

  To make the memory reliable while keeping the cell small, special attention was paid to transistor sizing. The sizing criterion is as follows: the p-transistor was chosen to be minimum size, as it only has to offset the effects of leakage. The resistance of the pass transistor and the pull down transistor in series determines the pull down speed of the BIT line, and also affects the write operation. Besides speed, the relative sizes of the pass transistor and the pull down transistor have to be ratioed so that the cell contents are not destroyed while doing a read. If for example the pass transistors were very large, then both points X and Y in Fig. 11 would be driven to $V_{dd} - V_{thn}$ thus erasing the memory cell contents. The pull down effective DC resistance and the pass transistor effective DC resistance must be scaled so that the pull down stores a low that is below the inverter switching point. The dynamic operation of the cell is further complicated by charge sharing of the BIT line capacitance and the internal memory cell capacitance when the word line is first asserted. Hand calculation using discrete element models and several experiments using `SPICE3` were conducted to determine the optimum transistor sizes. The results are presented in Sec. 4.2.2: Timing and Simulation.

  The beauty of the packet buffer design is that the memory need not be very fast at all. `SPICE3` results, which agree with hand calculations, indicate that the discharge time of the bit lines in a read cycle is on the order of $25nS$. The minimum access time, defined here as the time

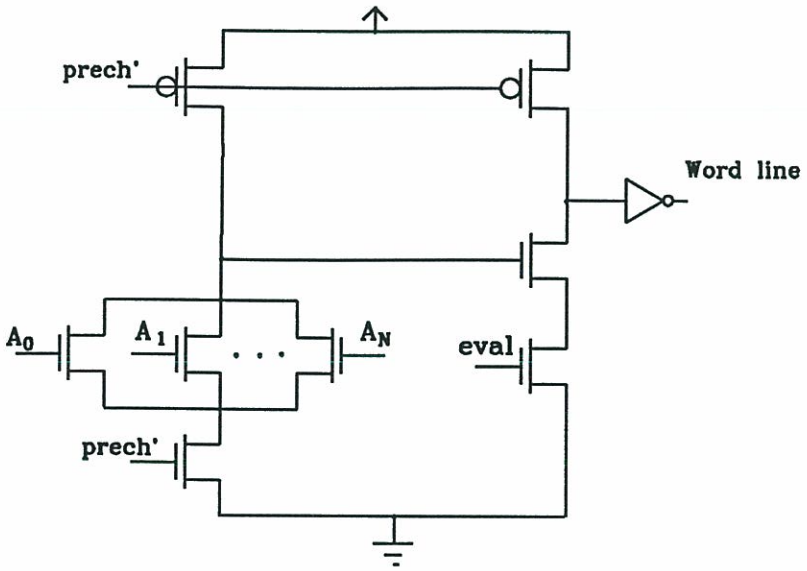

Figure 12: Dynamic NOR decoder for Row_2 $N-1$

between when a read is initiated and when data is accessible, depends on (t_i – means delay due to i)

$$t_{precharge\ of\ bit\ line} + t_{discharge\ of\ bit\ line} + t_{latch\ out\ data}$$

and is on the order of $40ns$, for $t_{precharge} = 10nS$, and $t_{latch\ out\ data} = 5nS$. In our design the total time allotted to a read is $800nS$. During the first $400nS$, the bit lines are precharged, and during the remaining time, the memory cell discharges them, thus leaving an order of magnitude in time as the safety margin.

- Column drivers and Address drivers

  The column drivers are used to drive the bit lines in the write mode and to precharge the bit lines in the read mode. The column driver consists of a $2:12$, L/W ratio driver used to drive the bit lines of memory. For bit lines having $0.5pF$ capacitance, FACTS results indicate that the rise and fall times are on the order of $5nS$ to $7nS$. For the read cycle, the column driver is equipped with a $2:12$, L/W pull up p-transistor. The pull up time, according to simulation, is on the order of $10nS$.

  The address drivers are buffers used to drive addresses from either the read pointer or the write pointer to the address decoder. The drive capability of the address drivers depends on the number of rows of memory. Each row of memory has a part of the dynamic row decoder associated with it. By adding more rows of memory, you add more segments of decoding and thus add more load on the address drivers. For the 32 packet buffer, a total load on an address line is roughly $0.45pF$. Drivers having a $2:12$, L/W ratio were used to drive a bit of the address and its complement, and the simulated rise/fall times are roughly $4.5nS$ and $3.5nS$ respectively.

- Row Decoder.

  Dynamic row decoders are small, fast and relatively safe to design [WeEs85]. We employed the dynamic NOR decoder shown in Fig. 12. Each row of memory has 1 of these decoders to drive its WORD line. In this circuit a domino NOR gate does the decoding. Precharge of the NOR is done on `prech = L` and the evaluation of the NOR gate is during `prech = H`. The

second clocked stage is a dynamic NAND gate, which is also precharged during `prech = L`. However, the NAND gate is evaluated on `eval = H`. The output of the NAND gate feeds a buffered inverter which drives the WORD line. Thus, after the NOR gate has evaluated, only 1 of 2^n NOR decoders will remain precharged. Hence, only 1 of 2^n NAND gates will discharge and only 1 WORD line will be asserted through the inverter.

The NOR transistors were made minimum size as they do not have to drive any significant capacitances. The precharge transistors were made stronger in order to minimize the pull up time. The final driving inverter has a $2:10$ L/W ratio and the simulated rise/fall times for the WORD line (0.25pF) were on the order of $3nS$ to $4nS$.

- Parallel-In Serial-Out output shift register.

  The implementation of the PISO is similar to that of the SIPO. However, the PISO serves two purposes: (1) as a parallel to serial data converter and (2) as a sense amplifier for the memory array.

  The fall time associated with the discharge of the bit lines by the memory cell pull down transistors, in most applications, is usually too long. Sense amplifiers are then used to speed up the discharge of the bit lines. Three types of sense amplifiers were considered, each with its own advantages and problems. The first was a simple inverter; all it has to do is detect a low going signal. The second was a ratioed inverter; it relies on shifting the threshold voltage of the FETS in order to speed up the fall time of a bit line. The third was a differential amplifier; the differential amplifier senses small differences between levels on the bit lines and then pulls the falling bit line to V_{ss} and drives the other bit line to V_{dd}. The ratioed inverter sense amplifier was not used because it is highly susceptible to noise and because the technique relies on close technology control. The differential amplifier, although the fastest requires 7 transistors. We chose to use the simple inverter since the speed gained by using a differential amplifier did not justify the area increase. Also, the memory does not need to perform very fast. The memory cell pulldown transistors can discharge the bit lines in approximately $25nS$: this is already $10\times$ faster than necessary.

  The PISO can be in either of two states: (1) being loaded or (2) being shifted. The load and shift are qualified on $d\phi_1$. The operating frequency range, $4.6KHz$ to $125MHz$, is identical to the SIPO since the implementation is quite similar.

The lower limit on the input speed of the memory is restricted by the Memory Latch to $350KHz$. The upper limit of the memory operation is not restricted by the dynamic circuitry ($125MHz$) but probably by the memory cell and the address decoder. In fact, as will be shown later, the main speed limiting element in the packet buffer is not the memory circuits but the Control/Timing circuit.

4.2.2. Timing and Simulation. Several experiments to determine the optimum sizes of the pass transistor and the pull down transistor in the memory cell were conducted. The basic test set up consisted of a column driver connected to the bit lines of a single memory cell with a capacitive load on the bit lines that was representative of a 32 row memory plane. Both read and write operations were observed. `SPICE3`, with model cards provided by `MOSIS`, was used to determine transient behavior. Note, the timing used in these experiments is approximately 8 to 10 more restricted than what is necessary. This was done to leave a significant margin of safety.

The equivalent circuit in the read and write mode is shown in Fig. 13. For the case of the write cycle, note that the internal node of the memory cell can be driven to a maximum voltage of $V_{dd} - V_{thn}$, although V_{thn} in this case is slightly increased. This slight increase in V_{thn} is due to the body effect (the channel of the pass transistor is not grounded). Thus, the maximum voltage the internal node can be driven to is $\sim 3.5V$. In a write cycle our goal would be to drive

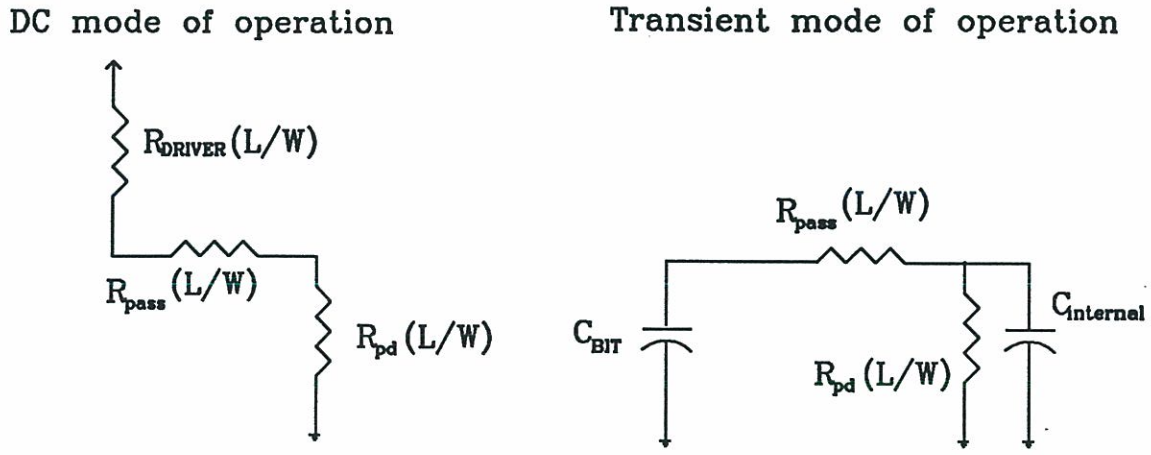

Figure 13: Equivalent circuit models for the memory cell during READ and WRITE

the internal node to at least $V_{inv} = 2.5V$ and come as close as possible to the 3.5 V upper limit. Once the inverter switching point is passed the cell will flip states due to positive feedback. Using DC resistances ($7K\Omega$ (L/W) for n-transistors and $17K\Omega$ (L/W) for the p-transistors) in the above equivalent circuit, the equation for the voltage at the internal node must satisfy :

$$\frac{R_{pd}(L/W)}{R_{pd}(L/W) + R_{pass}(L/W) + R_{driver}(L/W)} V_{dd} \geq V_{dd} - V_{thn}.$$

For the case of the read cycle, the pull down speed, using timing resistances ($20K\Omega$ (L/W) for n-transistors and $30K\Omega$ (L/W) for p-transistors) is :

$$C_{BIT}(R_{pass} + R_{pd}).$$

However, during a read cycle there is charge sharing between C_{BIT} and $C_{internal}$ when the word line is first asserted. This causes a voltage spike at the internal node (since it is initially discharged). We would like the maximum spike voltage to stay below V_{thn} and to be pulled to ground as quickly as possible (it could cause the cell to flip states if it were around for long enough).

Table 2 summarizes the results of the different sizing experiments. We felt that the last entry in the table was the safest sizing since it met the original design criterion of (1) Max. Write Voltage $\geq V_{inv}$, and (2) the spike height was kept below V_{thn}. The second entry in the table is probably a

(L/W) Pass Transistor	(L/W) Pulldown X-Tor Transistor	Max. Write Voltage	Spike Height	Read delay
1:2	2:1	3.8 V	2.25 V	$25nS$
1:2	1:1	3.0 V	1.25 V	$15nS$
1:2	1:2	2.65 V	0.75 V	$10nS$

Table 2: Summary of results of Transistor Sizing Experiments

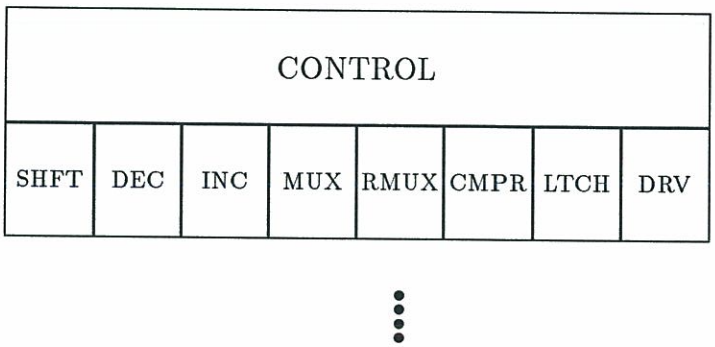

Figure 14: Block diagram of a bit slice of a register

little too ambitious, since variation in processing could change V_{thn} (typically 1 volt) and thus cause problems during a read. The first entry is fine for writing but would cause severe problems for a read.

4.3. Pointers

A two phase pseudostatic latch connected to the output of a 4 × 1 MUX was used as the core element of a static register. The register operates in two states: (1) it either holds its current contents or (2) it gets assigned a new value. A new value can be assigned by using the select lines of the MUX to choose between the output of an increment, a decrement, a shift or a parallel load circuit. If none of the options are selected a hold is assumed and the register holds on to its contents. A block diagram of 1 bit slice of the register is shown in Fig. 14. The register is reset externally by loading '0's into the register; this is done automatically when an external reset occurs. The contents of a register may be fed to a comparator which can then be used to either reset the register or drive an external line. The register tri-state outputs can provide high drive.

Two up-counters with tri-state outputs were used as the read and write pointers. The outputs of the read pointer and write pointer were connected to form the address lines for the memory. Since the write pointer and the read pointer will never be enabled simultaneously, there is no danger of conflict. During the read half of a packet cycle, the read pointer outputs are enabled while the write pointer outputs are disabled. Thus, the read pointer drives the address lines to the memory. The opposite situation occurs during the write half of a packet cycle. The registers were implemented with comparators that would reset when the count reached 31. The read pointer is incremented whenever a read `ACK` from downstream is received, while the write pointer is incremented whenever a `SYNC` is detected within a packet cycle.

The current capacity pointer was implemented as an up-down counter, with the `irp` signal of the read pointer connected to the decrement select of the MUX and the `iwp` signal of the write pointer connected to the increment select of the MUX. Two comparators are used to detect (i) count = 0 and (ii) count = 31. The former signifies buffer empty while the latter indicates that the buffer is full. High drive was put onto the output bits of this register in order to drive test pins.

The pointers in the packet buffer need not be very fast since they increment/decrement only once every packet cycle ($1.72\mu S$). The maximum operating frequency is an increment/decrement every $100nS$. In terms of packet buffer operation this corresponds to a packet cycle of $100nS$ and an

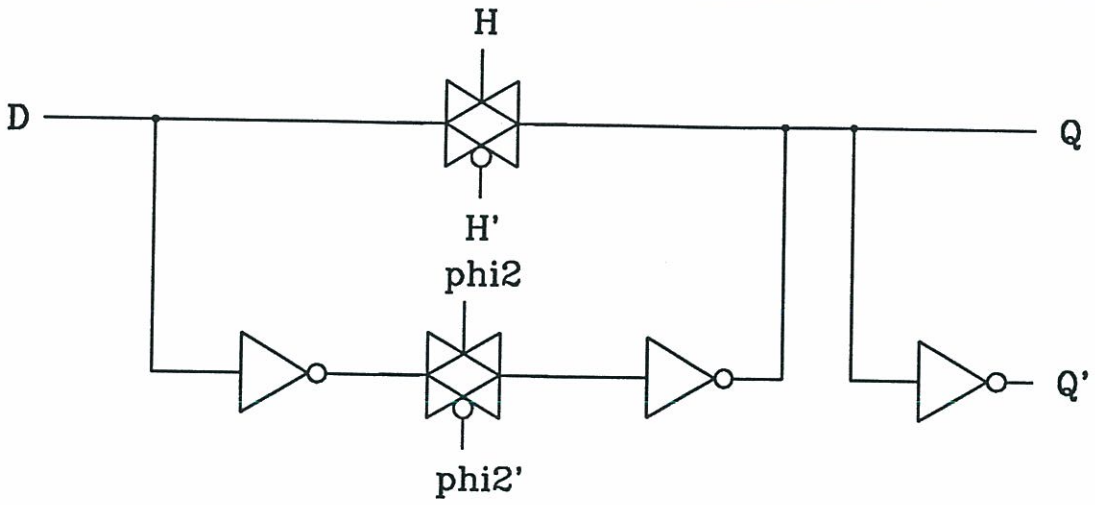

Figure 15: A 2 Phase Pseudo-Static Latch

input rate of $800MHz$. Since we already know that the maximum operation speed of the memory is $125MHz$ we did not bother to try and make the registers any faster. The blocks described in Fig. 14 will now be discussed briefly.

- 2 Phase Pseudo-Static Latch.

  Fig. 15 shows the 2 phase pseudo static latch. It is pseudo static because the latch looses its contents if the clocks are turned off. The operation of the latch is fairly simple. The latch is loaded on phase 2, and a hold is qualified with phase 1. During a hold, a feedback path via a transmission gate enables signal restoration. The maximum speed that the latch can be run at is dependent on the following delays

  $$2 \times t_{TX-gate} + 2 \times t_{inv.} + t_{charge\ of\ nodes},$$

  and is about $166MHz$. For a capacitance of $0.09pF$ on Q, and a $1nA$ leakage current, the stored high takes on the order of $1.7mS$ before becoming unusable. This latch sets the lower operating frequency limit to $588Hz$. The latch provides outputs of both polarity.

- Multiplexor.

  A 4×1 MUX was used to select between incrementing, decrementing or shifting in a new value into the pointers. The MUX was implemented as 4 transmission gates with the enable on each of the 4 transmission gates being complementary signals. The enable signals were qualified with $i\phi 1$.

- Increment circuit and Decrement circuit.

  The increment circuit is a ripple carry counter implemented using transmission gates. Transmission gates were used in order to minimize the delay associated with propagating the carry. The output of the increment circuit is fed to an input of the MUX. The least significant carry bit of the counter is tied high so that the output of the increment circuit is always one more than the contents of the pointer.

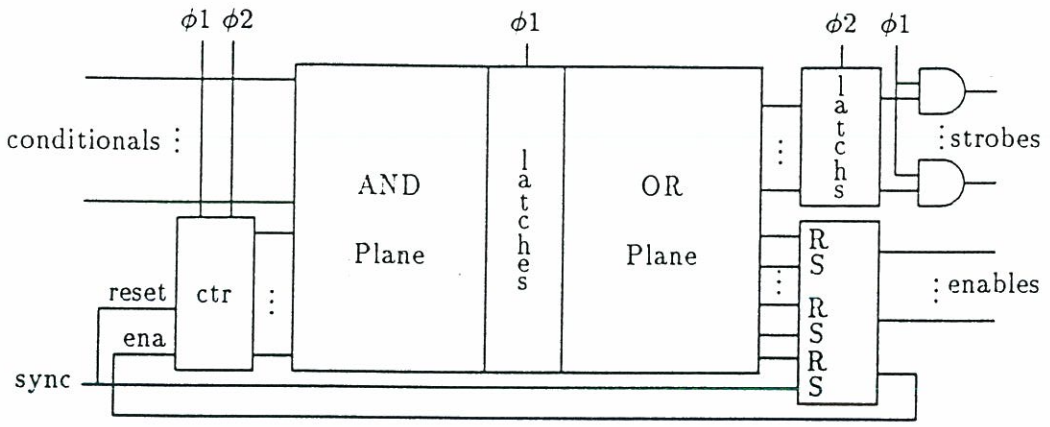

Figure 16: Block diagram of the Timing/Control Module

The decrement circuit, implemented using transmission gates, is very similar to the increment circuit. The least significant bit of the decrement circuit is tied high, thus making the output of the decrement circuit one less than the contents of the pointer.

Simulation with FACTS revealed that a 5 bit counter (32 packet buffer needs a max. of 5 bits) using the above two circuits could increment or decrement once every 110nS. But since these up/down counters would need to increment/decrement only once every $1.6\mu S$, we did not pursue faster circuits.

- Comparators

  The comparators are decoders and were built along the same lines as the memory address decoder. The only difference here is that a static NOR gate was used instead of dynamic circuitry. The output of the NOR is used to (1) either reset the register when a particular count is reached or (2) to signify a particular count (example: count=full or count=0).

- Serial shift in.

  The serial shift in circuit merely connects the output of the 0^{th} bit to the input of the 1^{st} bit and the output of the 1^{st} bit to the input of the 2^{nd} and so on. The input of the least significant bit is driven from the external world while the output of the most significant bit serves to monitor the contents of the register. Using this technique, a register may be loaded externally; this method also forms the basis for the Level Sensitive Scan Design (LSSD) check.

- Tri-state and High Drive output.

  The tri-state feature consists of a transmission gate that isolates its input from its output when not enabled, while the high drive circuit consists of $2 : 12, L/W$ ratioed inverters to provide high current drive.

The design of the pointer registers was fairly straightforward. All that was necessary was to design safe and reliable registers, since the speed and area constraints involved in the case of the memory plane don't hold for the register.

4.4. Timing/Control Module

The Timing/Control Module generates all of the control signals for the packet buffer. It also keeps track of local time within the packet buffer, i.e. time relative to the current packet cycle. The control

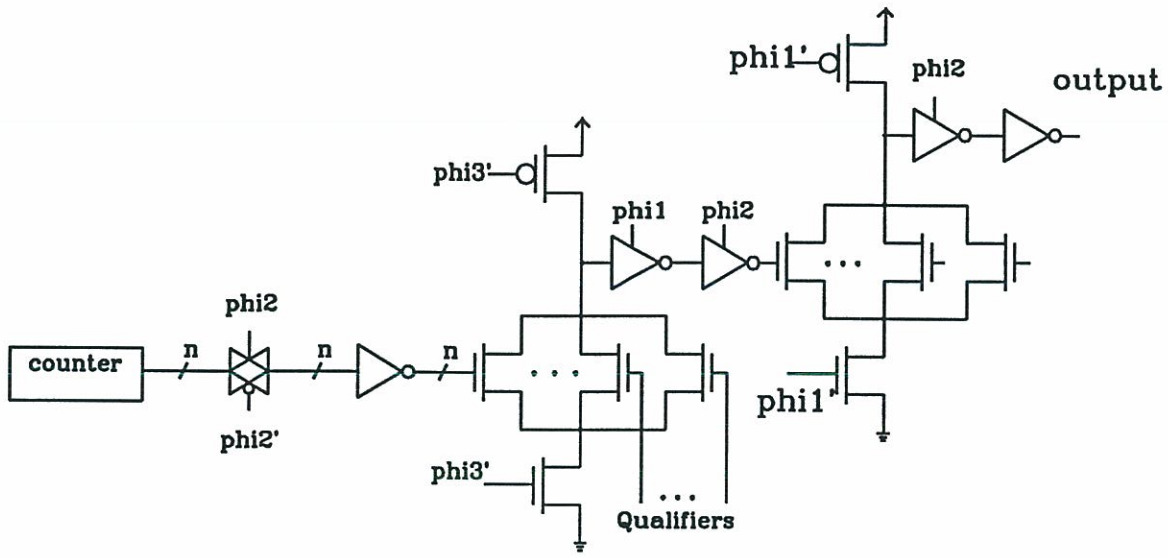

Figure 17: Expanded view of Timing/Control Module

signals it generates are enable signals which are asserted during a given clock cycle and negated at some later time. These signals change state during the second clock phase (stable $i\phi_1$) and may thus be used for enabling outputs which will be latched on the first clock phase. Enable signals may be asserted and negated multiple times within a single epoch. The enable signals may be qualified by some combination of the conditional inputs. For example, an `iwp` is issued only if *time=41* and a `SYNC` was seen earlier within the current packet cycle, and the buffer is not already full. A block diagram of the Timing/Control Module appears in Fig. 16. An expanded view of the block diagram is shown in Fig. 17.

The Timing/Control module was implemented using a counter to count the clock ticks in a packet cycle, and a dynamic PLA to decode the various times and guard conditions. The outputs of the counter are fed into the decoding PLA. The PLA outputs indicate the various clock times of significance. The product terms are all of the form

$$(time = t) \ \wedge \ cond_a \ \wedge \ ... \ \wedge \ cond_n.$$

The PLA outputs are formed by ORing together some of these product terms. These outputs are then connected to a set of R-S flip flops used to generate the enable lines. The enable lines are the buffered outputs of these flip-flops.

A dynamic PLA was used instead of a PLA generated by the existing PLA generator, `mpla`. The PLAs generated by `mpla` can operate up to a maximum frequency of $\sim 30MHz$ for a PLA with 10 product terms. We also considered using an external control chip (Electrically Programmable Logic Device (EPLD)), however, the maximum operation frequency of these devices is $35MHz$. Furthermore, the delay involved with getting an off chip control signal to an internal module is significantly greater than the delay compared to the case where the signal is generated on chip and merely routed to the internal module.

The maximum operating frequency of the Timing/Control module was simulated at $50MHz$ to $55MHz$. The counter and the AND plane of the PLA are the major bottlenecks. Currently, the

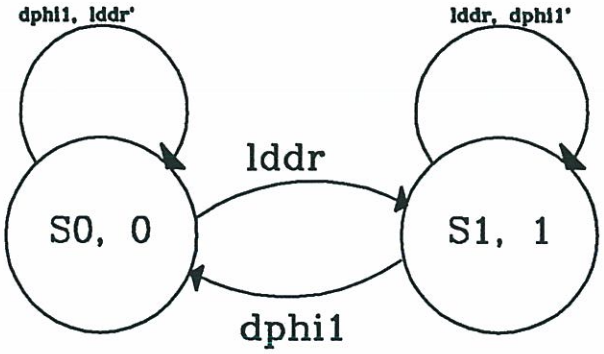

Figure 18: State diagram for the qualifying of the output PISO register

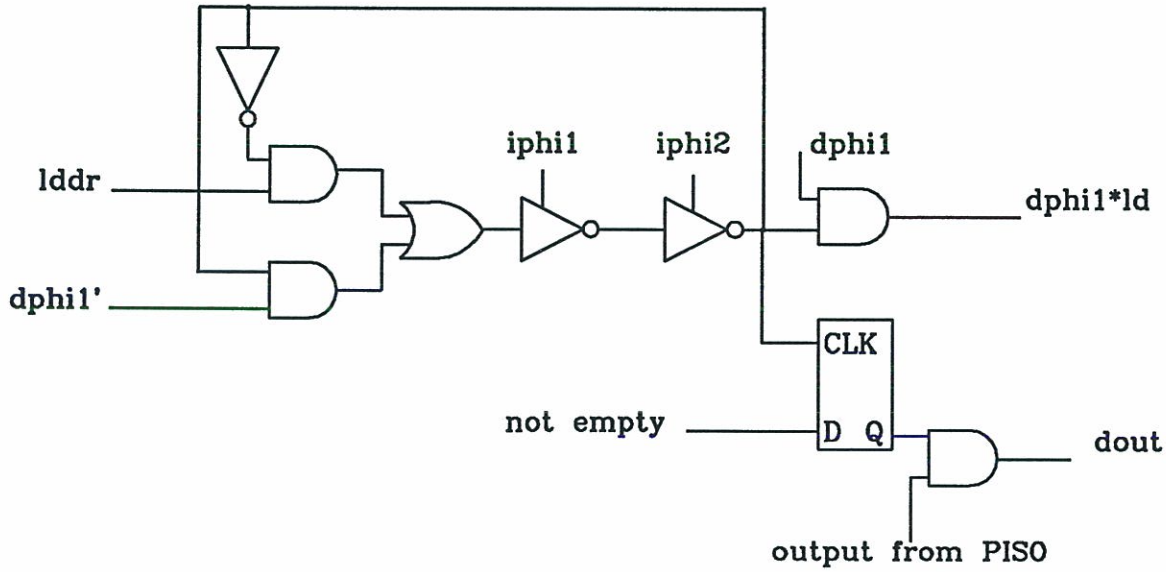

Figure 19: Circuit to qualify loading of the output PISO register

counter has been implemented as a simple ripple carry, and for a 7 bit counter of this type the maximum operating frequency is around $50MHz$. However, even if a faster carry propagate circuit using a look-ahead technique was implemented, the speed would eventually be restricted by the AND plane of the PLA. The AND plane of the PLA which is implemented as a dynamic NOR circuit, with approximately 10 inputs, can operate up to a maximum frequency of $55MHz$. This limit is due to the time required to pull up the NOR in the precharge phase. There are 10 minimum size transistors connected to the pull up line, and as the clock frequency is increased less time is given to pull up the NOR.

The design of the other two components of the control circuitry was fairly straightforward. The SYNC capture circuit, which consists of an R-S flip flop, was built using cross-coupled NORs. A high drive static D-latch was used to sample the output of the R-S flip flop. The output of this D-latch (*ssync*) is then fed to the main control and timing generator.

The circuit to qualify loading of the output shift register is based on the state diagram in Fig. 18

Section	# of trans.	% total trans.	% total area
SIPO	8100	5.0	4.0
Memory Latch	2700	1.6	2.3
Column Driver	8100	5.0	3.9
Memory Cell	129,600	78.5	34.0
Row Decoder	3168	2.0	2.6
PISO	8100	5.0	4.0
Memory	159,768	97.1	50.8
Pointers	1750	1	1.5
Control	800	0.5	1.1
Drivers, pads, etc	$\sim$ 2500	$\sim$ 1.5	$\sim$ 9.2
Routing	-	-	37.4
Total	$\sim$ 165,000	$\sim$ 100	$\sim$ 100

Table 3: Transistor/Area usage of the major components in the Packet Buffer

and the corresponding implementation is shown in Fig. 19. State 0 indicates that the circuit is waiting for a load signal from the control and timing generator. State 1 represents the wait for a $d\phi_1$. The circuit, initially in State 0, changes state once the `lddr` signal from the timing generator is received. It then waits till the next occurence of $d\phi_1$ before going back to state zero and waiting for the next `lddr`. The dynamic shift register clocked by $i\phi_1$, $i\phi_2$ is merely a way of clocking the circuit and ensuring the break in the feedback path that could otherwise affect circuit operation. Another function of QUAL-LDDR is to qualify the output data. If for instance the buffer is empty then a NULL packet must be shifted out. This is accomplished by the D-latch in Fig. 19.

4.5. Whole Chip

The area of 1 bit plane of memory is on the order of $2000 \times 2500\lambda^2$, and the entire 9 planes of memory, along with the pointers, control and clock driver circuitry fit within a $7.2mm \times 9.2mm$ frame. The pads which include a bi-directional, a V_{ss}, a V_{dd}, and a substrate pad were provided by `MOSIS`. A 108 pin grid array (Kyocera KD-P87938-A) was used as the package. Four V_{dd} and four V_{ss} pins have been allotted for uniform power and ground distribution. Clocks and control signals to the memory planes are driven using clock drivers having a $5nS$ delay (pipe-lined). These clock drivers are modified PAD circuits used in other VLSI chips in the prototype effort.

Appendix A describes the pin assignment while Appendix B shows the physical placement on the PGA. A VLSI photograph of the entire chip is presented in Fig. 20. There are approximately 165,000 transistors on the chip The memory cells use up a third of the available area while using 80% of the transistors. The other area dominating factor is the routing which takes up almost 40% of the available space. A detailed breakdown of the transistor/area use with respect to the major sections is presented in Table 3.

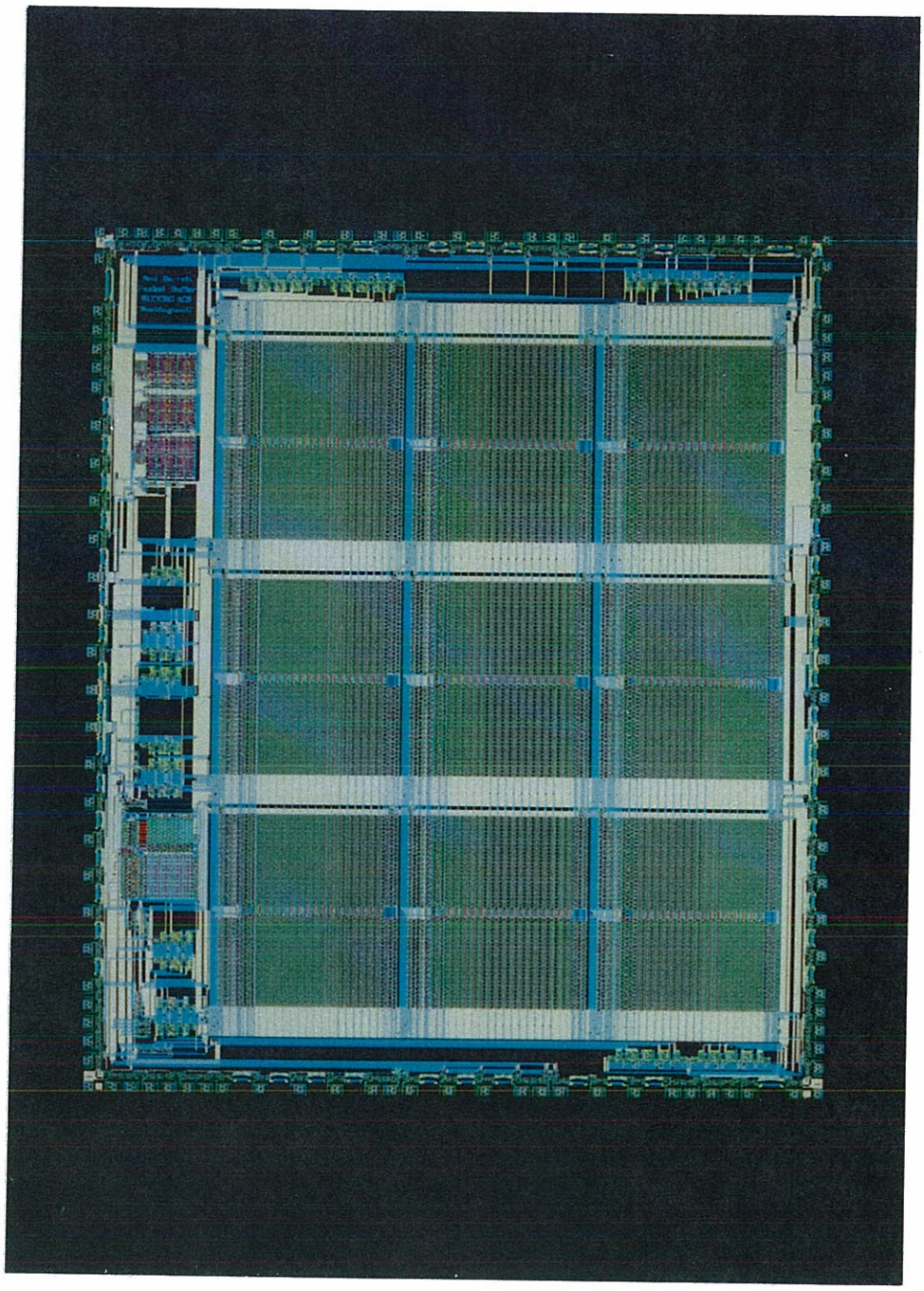

Figure 20: The packet buffer chip

5. Packet Buffer Generator

As mentioned in Section 1, most of the circuit layouts were program generated. Two separate tools were created to aid in design of the Packet Buffer: (i) `mkmem` – to make a bit-plane of memory, (ii) `mkreg` – to construct static registers to serve as pointers. An existing tool, `peg_ctlt` – a timing and control module generator, was used to generate the circuit to control the operation of reading and writing out of the buffer memory [Ro88].

The tools are intelligent cell placing programs, similar to something on the order of **mpla** from the Berkley CAD tools. All three programs rely on a set of library cells/templates from which to construct the final `MAGIC` file. Namely the designer must initially provide a one time effort of hand layout of a set of basic cells. Next, by program control, various combinations of these basic building blocks can be put together (depending on the user's input) to form more complex circuits. It is important to realize that drastic changes such as substitution of new circuits or technology changes don't affect the program since all that matters to it is the pitch of the cells and not its contents.

Our initial goal was to use MQUILT, which come along with the Berkley CAD Tools, as the cell placement program. However, this program does not offer enough user flexibility. Firstly, MQUILT does not allow for any sort of processing on the tiles to assemble together. For example, a conditional case where one set of cells were to be used for one outcome and another set for a different outcome would require external processing before the `MAGIC` file could be created. In other words, mixing of code and `MAGIC` file generation is not possible. Secondly, MQUILT deals only with a single instantiation of `MAGIC` cells. So for a memory plane containing 2400 memory bits (1 bit plane of the 32 packet buffer), this would require 2400 instantiations of memory cells. The output `MAGIC` file when loaded up in `MAGIC` would be tremendously slow and be very difficult to work with. For these two reasons an easier route was chosen. Subroutines written by George Robbert for his Silicon Compiler were used [Ro88]. These routines, allow the intermingling of processing and `MAGIC` file creation and also deal with array structures in `MAGIC`. Several other features such as the ability to include variable line lengths for routing; placing of the user's labels in a `MAGIC` layer directly; making and connecting lines to busses, are a few of the luxuries offered by these routines.

A description of the three tools follows.

5.1. MKMEM – Packet Buffer Memory Generator

`Mkmem` is quite application specific, in that, the circuits generated are tailored toward the memory architecture described in Fig. 8 of Sec. 4. The program would require some work before it could be used effectively as a general purpose RAM generator. `Mkmem` takes as input the row and column size of the memory plane, where row and column size correspond to # of packets and packet length respectively and generates a `MAGIC` file containing a mask level description of the memory plane with all the necessary routing. The row decoder/driver is built on the "fly" depending on the number of rows input. The number of decoding bits is based on the $log_2(number\ of\ rows)$. The number of columns determines the number of stages to be used in the SIPO input shift register, the memory buffer register, and the PISO output shift register. It also determines the number of column drivers necessary. Fig. 21 shows the cell level `MAGIC` file containing a 4 row × 5 column plane of memory (for brevity a 32 × 75 plane was avoided here). The cells prefixed with `rt` are router cells. They exist on the east, west and north side of the main memory plane. The router cells contain busses that are shared among the various bit-planes. In the upper corners and at the lower corners of the memory plane are void spaces, here an individual cell must be placed in order to connect the right input data bit to the memory plane, also test lines and the output bit are routed from the memory plane via cells in the lower right corners.

A manual page for `Mkmem` is included on the following page.

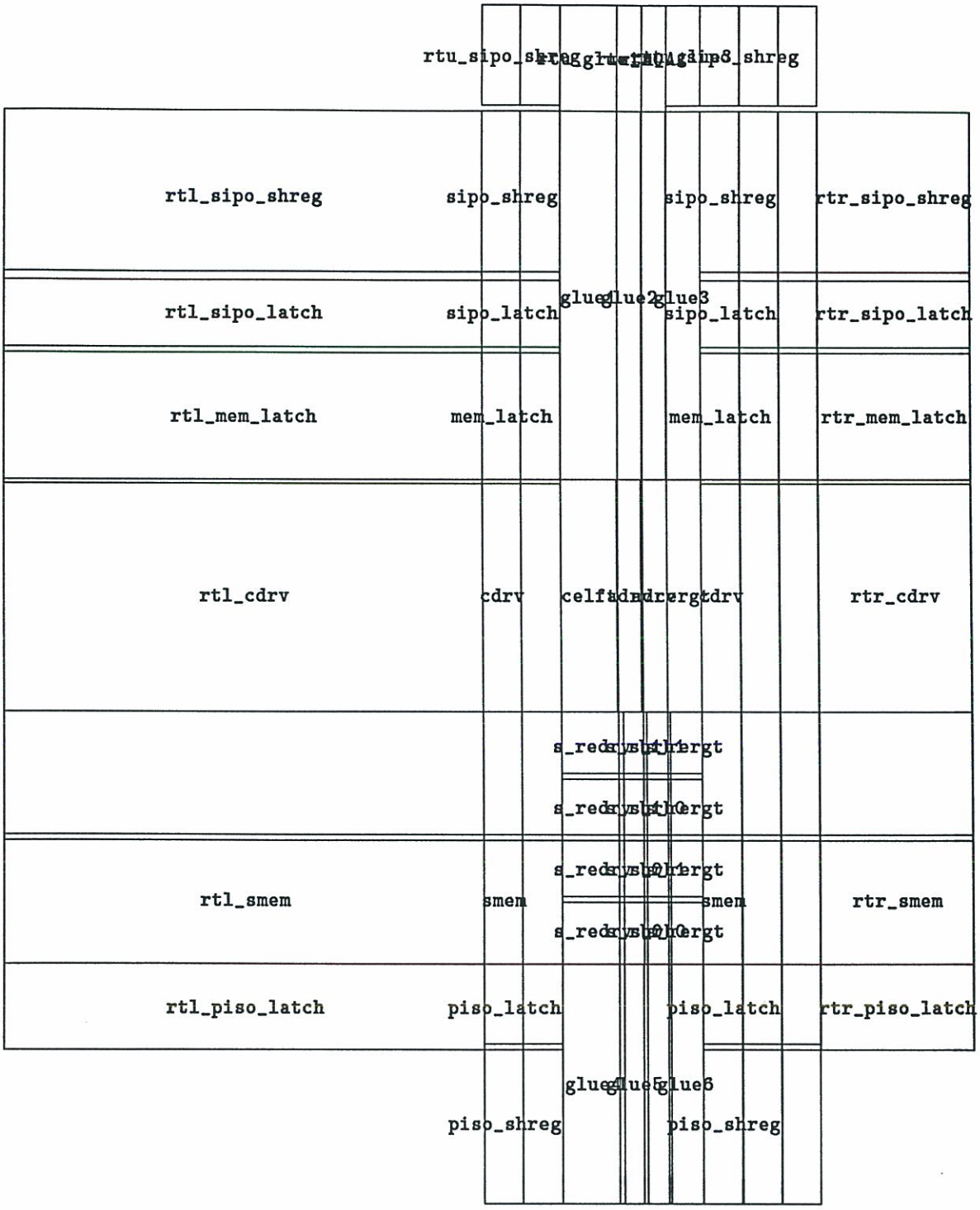

Figure 21: A MAGIC file showing the cell level of a 4 row × 5 column memory plane

NAME

`mkmem` – make a bit-plane of memory

SYNOPSIS

```
mkmem -r# -c# -o outfile.mag -l libname
```

DESCRIPTION

Mkmem is a circuit generator for a bit-plane of memory used in packet buffers. It is a command line driven program that takes as input the number of rows and the number of columns in the memory plane. Note the number of rows signifies the number of packets that the `fifo` will hold, and the number of columns signifies the length in bytes of a packet. The number of rows must be a positive, even integer less than 64. The number of columns must be a postive integer.

`-o filename`. Filename is the where mkmem puts the mask level description of the circuit generated. The default is set to 'memgen/memlib/memory.mag'.

`-l libname`. Libname is the library where the cells to be used to make bit-plane are located. The default is set to 'memgen/memlib'.

`-r rownum`. The number of packets that the buffer is to hold; should be an even number less than 64.

`-c colnum`. The number of bytes in a packet to be stored in the buffer.

EXAMPLE

```
mkmem -r32 -c80 -o memgen/memlib/bitplane.mag -l memgen/memlib
```

The above command will generate the mask layout of a bit-plane for a 32 packet buffer having 80 bytes per packet in the file memgen/memlib/bitplane.

BUGS

An odd number of rows causes wierd tessalation and is recommended that one refrain from using a buffer with an odd number of packets to store.

5.2. MKREG – Packet Buffer Register Generator

Mkreg, unlike **mkmem**, is not application specific. It was initially designed in conjunction with Einir Valdimarsson. The registers created are general purpose static registers. The core element of **Mkreg** is a multipelxor and a static latch, with options to allow the following: (i) an up-counter, (ii) a down-counter, (iii) a parallel load and a serial shift in, (iv) high drive on the output (v) tri-state on the output and (vi) external or internal reset (on a comparator output). Once the user specifies the various options, circuits to perform these functions are connected to the MUX inputs. Fig. 22 shows the cell level **MAGIC** file containing the current capacity pointer. The register consists of a control block which sits on top of 5 bit slices of data path. There is a one to one correspondence between the control for a block and its associated data-path.

A manual page for **Mkreg** is included on the following page.

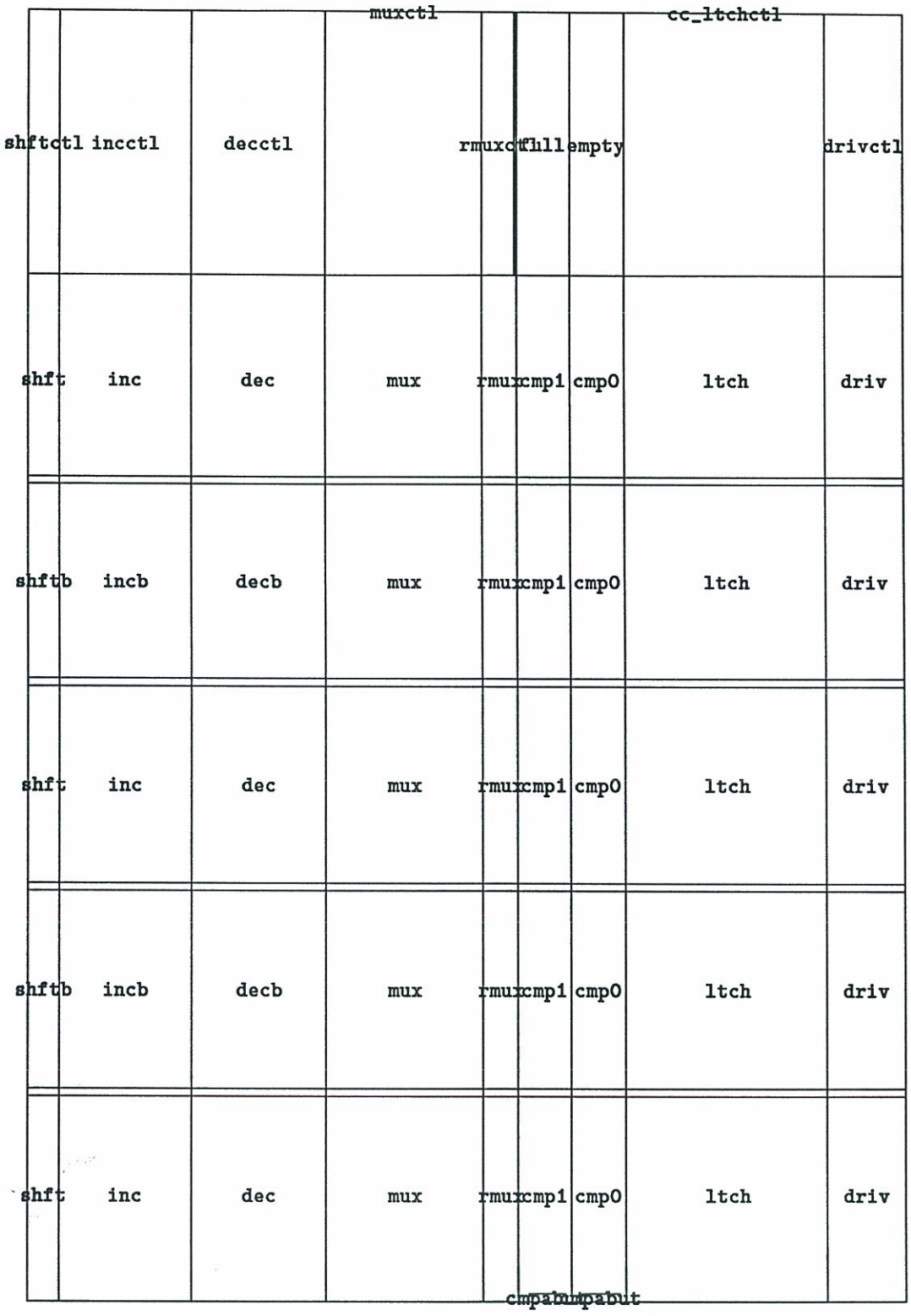

Figure 22: The MAGIC file showing the cell level of the current capacity pointer

NAME

`mkreg` – make a static register

SYNOPSIS

`mkreg -SFIMRLDE -n# -r# -o` *outfile.mag* `-l` *libname*

DESCRIPTION

Mkreg is a command line driven program that generates a static register with several options. Some of these options include the ability to (i) increment the contents of the register, (ii) to decrement the contents of the register, (iii) to load the register in parallel, (iv) to serially shift in data via the least significant bit (LSSD designs), (v) to provide a tri-stateable output, (vi) to provide high-drive feature to be able to drive busses, (vii) an external as well as an internal reset feature, (viii) built in comparators that indicate when the count of the register is equal to the programmed value.

`-S` Serial shift in.

`-F` Decrement by 1.

`-I` Increment by 1.

`-M` A multiplexor to select between serial shift in, increment, decrement, and parallel load.

`-R` External reset and internal reset feature; note the internal feature is not useful without specifying the value to reset on.

`-L` A static latch which holds the contents of the register.

`-D` High drive for busses.

`-E` Enable output for tri-state.

`-n` # Number of bits that the register is to have; i.e. a 5 bit register.

`-r` # A number that the internal comparator is to reset the register on.

`-o` *outfile* Outfile is where mkreg puts the mask level description. The default is set to 'REGLIB/reg.mag'.

`-l` *libname*. Libname is the library where the cells to be used to make bit-plane are located. The default is set to 'memgen/memlib'.

EXAMPLE

```
mkreg -SFIMRLDE -n5 -r31 -o REGLIB/reg5.mag -l REGLIB
```

The above command will generate the mask layout of a 5 bit register, with an up/down count feature, external reset, parallel load, serial LSSD load, high drive and tri-stateable output, additionally the register resets when count = 31. The mask level description is put in 'REGLIB/reg5.mag'.

BUGS

No special error checking is done for the unwise who seek to crash the program. At the very least a nice core will be dumped for you.

5.3. PEG_CTLT – Processing Element: Control and Timing Generator

The program used to generate the mask level layout of the timing/control circuit is PEG_CTLT (Processing Element – Timing and Control). PEG_CTLT was developed by George Robbert as part of a Silicon Compiler for his Master's Thesis [Ro88]. The compiler generates a restricted class of circuits called Synchronous Streams Processors.

The program is invoked as follows: `peg_ctlt -o fifo_ctl -l ctllib fifo.in`; where the input file fifo.in is as follows:

```
condinps ssync ack notfull
muxouts mil csync crst prech mw macc iwp rw irp lddr
cntsize 7
cntmax 90
cntena --- 86 <MAX>
sigset 1-1 41 mil
sigrst --- 42 mil
sigset --- 26 csync
sigrst --- 27 csync
sigset 1-- 41 crst
sigrst --- 42 crst
sigset --- 2 prech
sigrst --- 19 prech
sigset --- 41 prech
sigrst --- 59 prech
sigset 1-1 40 mw
sigrst --- 80 mw
sigset --- 21 macc
sigrst --- 39 macc
sigset --- 61 macc
sigrst --- 79 macc
sigset 1-1 78 iwp
sigrst --- 79 iwp
sigset --- 40 rw
sigrst --- 80 rw
sigset -1- 48 irp
sigrst --- 49 irp
sigset --- 27 lddr
sigrst --- 31 lddr
```

Note the -o option specifies the output `MAGIC` filename and the -l option specifies the library containing the cells to be used in the `MAGIC` file.

The syntax of the program is straight forward. It contains the names of the qualifier inputs, the output signals, and the conditions as well as the time that an ouput signal is to be asserted. Specifically, the first line declares `ssync`, `ack` and `notfull` as the qualifier signals. The second line declares `mil`, `csync`, `crst`, `prech`, `mw`, `macc`, `iwp`, `rw`, `irp` and `lddr` as the output lines of the control/timing circuit. The next three lines declare the number of bits required to count the ticks in a particular epoch, the maximum countsize and when the count should 'stick'. Following these basic definitions, is a description of when signals are asserted and deasserted. For example, take `mil`: the declaration `sigset 1-1 41 mil`, says that `mil` is to be asserted at $time = 41$ provided, `ssync` $= 1$ and `notfull` $= 1$. The next line `sigrst --- 42 mil`, says that `mil` is to be negated at

$time = 42$.

Fig. 23 shows the cell level **MAGIC** file containing the timing and control circuit for the packet buffer. The qualifier signals are brought in on the left; the two phase system clock along with the global reset are brought in on the right to the cell andorcnt. The output of the circuit comes out of the S-R flip flops on the upper right corner of the circuit.

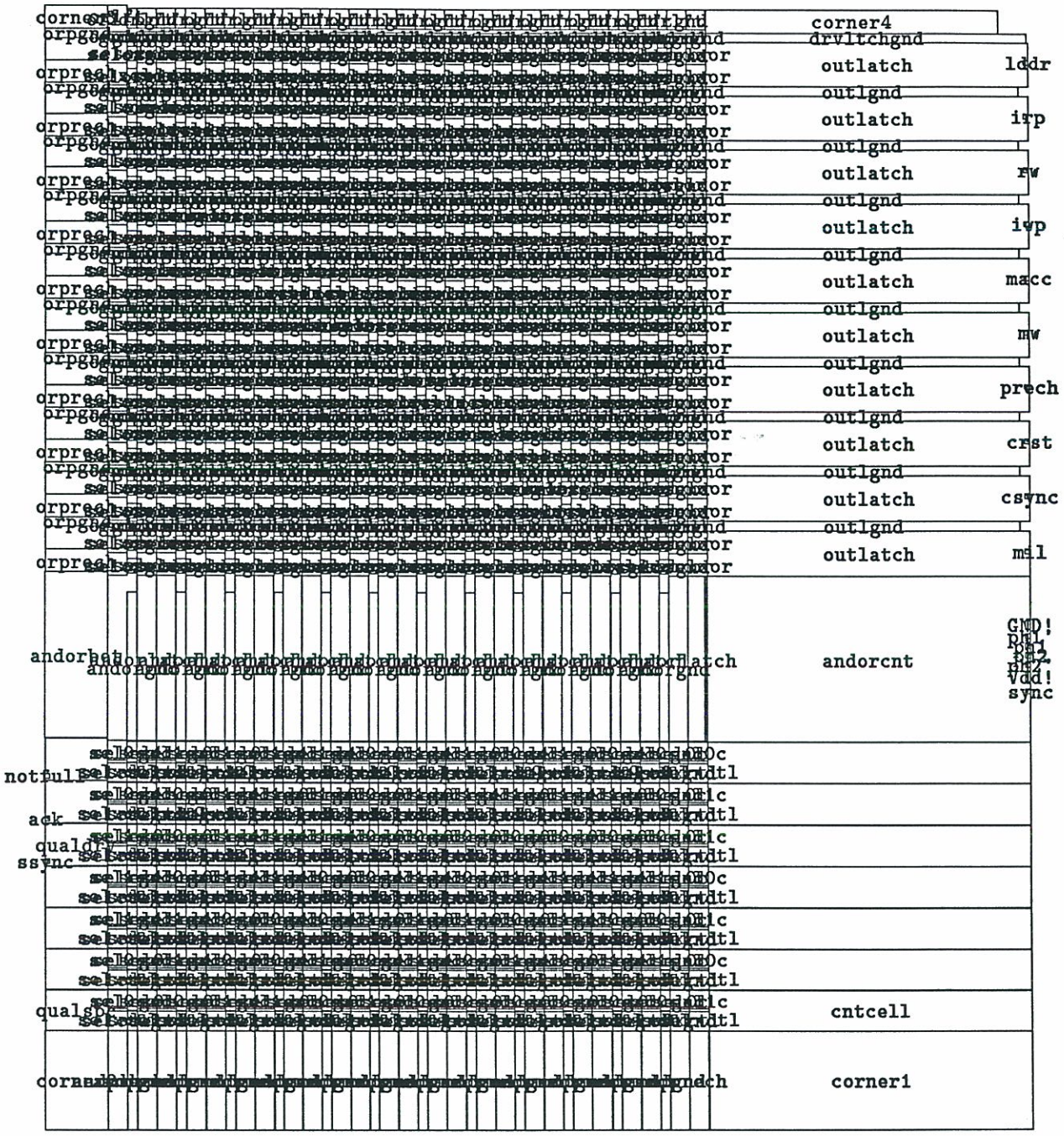

Figure 23: The MAGIC file showing the cell level of the Timing/Control circuit

6. Testing

This section describes some of the testing schemes we implemented in the Packet Buffer. They are divided into two categories: (1) built in testing schemes and the monitoring of major data, address and control busses, and (2) subdivision of the entire chip into smaller chips to be fabricated individually.

6.1. Built in Testing and Monitoring of Busses

Our major goal in testing the packet buffer chip, was to be able to isolate regions of interest and take control of them externally. This scheme involved the use of bi-directional control lines, that under normal operation would simply monitor an internal signal and in test mode would permit the outside world to drive the line. The concept is illustrated in Figure 24. When `Xctl` (external

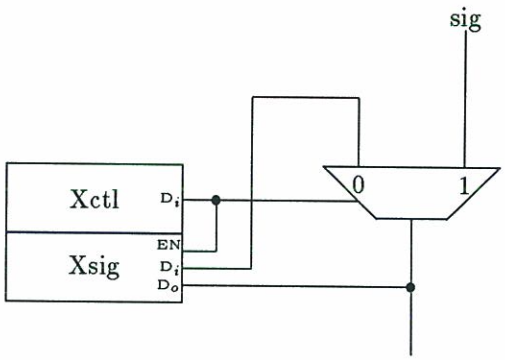

Figure 24: Generic Testing Scheme

control) is asserted, it selects data from "1" of the multiplexor. At the same time, the bi-directional pad `Xsig` is enabled, causing data from the output of the MUX to be driven to D_o of the MUX and out to the external pin. When `Xctl` is not asserted, then line "0" of the MUX. is selected and D_i appears at the input of the MUX. The overall effect of this is that data from `Xsig` is driven to the rest of the circuit.

There are three major sections that need individual isolation: the memory planes, the pointers and the control. By isolating each block we can concentrate on the signals generated by the block. Furthermore, in the case of the control circuit or the pointer circuit, we can provide the signals generated by them for the other circuits externally. For example, let's say we wanted to write a packet to slot 17 of the buffer. By isolating the control circuit and the pointer circuit, we would have external control of the memory. By externally driving the pointer address to 17 and by providing the appropriate control signals to the memory, we could write a packet to slot 17 of the buffer.

6.1.1. Memory Planes. In order to test only the memory planes and isolate them from the other circuits, `XMEM` and `OE` are asserted. This blocks off the following signals coming from the control

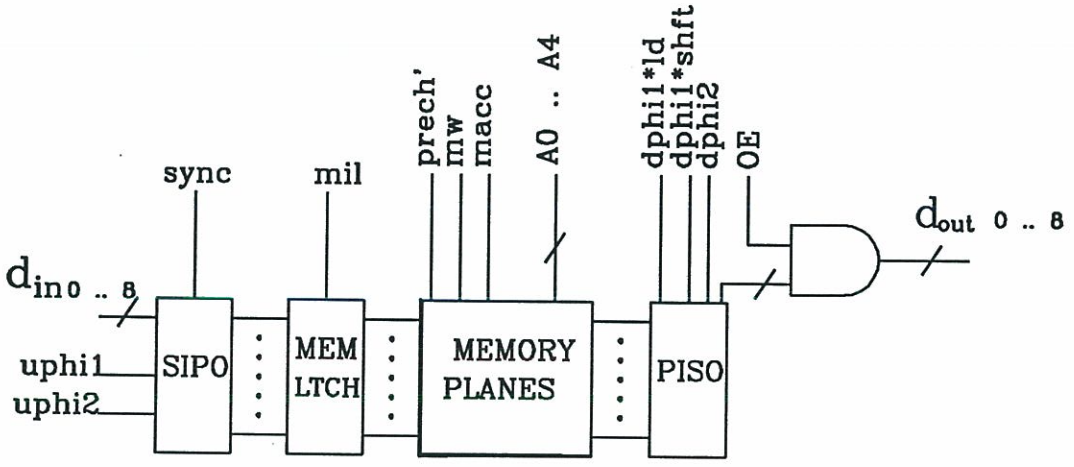

Figure 25: Memory Planes isolated from the rest of the chip

Direction	Name	Function
<>	Xprech	External precharge; when Xmem=L, prech appears on Xprech.
<>	Xmw	External write signal; when Xmem=L, mw appears on Xmw.
<>	Xmil	External memory latch; when Xmem=L, mil appears on Xmil.
<>	Xlddr	External load down stream register; when Xmem=L, lddr appears on Xlldr.
<	OE	Output Enable; pulls up line from Current Capacity register indicating buffer status.
>	XMEM	External control of memory; when asserted signals must be provided from the external world, when not asserted it serves to monitor various control and address lines.
<	tso0 .. tso9	Serial output bit of the SIPO of bit planes 0 through 9.
>	tpi0 .. tpi9	Serial input bit of the PISO of bit planes 0 through 9.

Table 4: Pins associated with testing the memory bit planes

circuit: precharge, memory-in-latch, load PISO, and memory write. Additionally the address lines from the pointers are also blocked off. The user now has to provide these signals externally. Notice that the user can now slow down the various clock speeds and test the circuits for logical operation before cranking up the speed.

Fig. 25 which is extracted from Fig. 6, shows the memory planes isolated from the rest of the circuit. In addition to taking external control of the memory, the contents of the SIPO input shift register are monitored. Once data has been accessed from the SIPO, and while the new packet is being shifted in, the old packet is being shifted out. Thus, the original packet under scrutiny will appear at the output of the SIPO 75 clock ticks (1 packet size) after it was shifted in. Table 4 lists the pins dedicated to testing the memory bit planes (this is in addition to the normal pin out). The notation >, <, <> stands for input pin, output pin and bi-directional pin respectively.

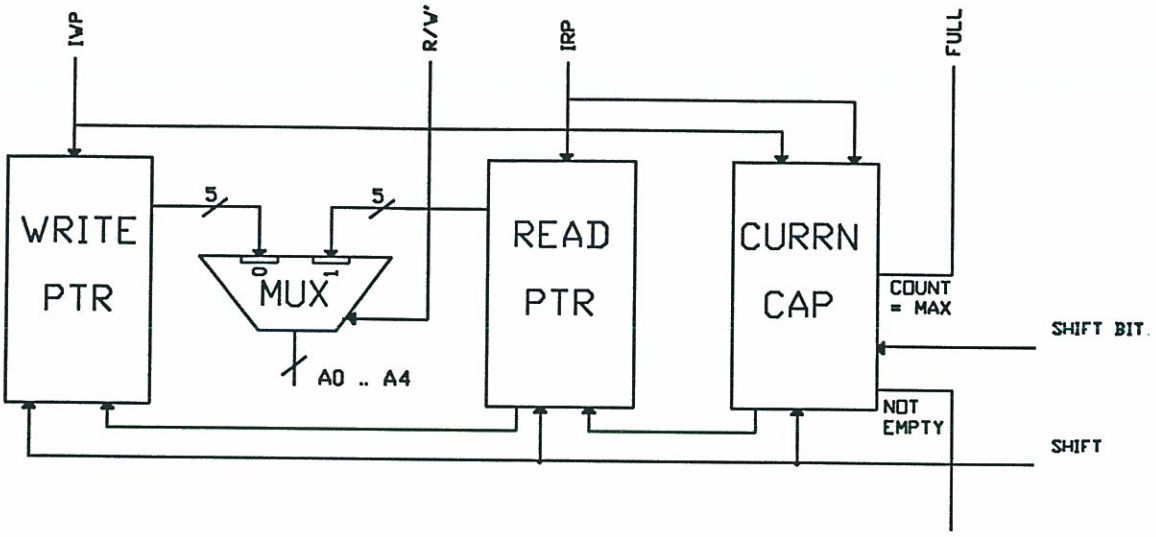

Figure 26: Pointers isolated from the rest of the circuit

Direction	Name	Function
<>	Xiwp	External increment write pointer; when XREG=L, iwp appears on Xiwp.
<>	Xirp	External increment read pointer; when XREG=L, irp appears on Xirp.
<>	Xr/w'	External read/write to select read or write address; when XREG=L r/w' appears on Xr/w'.
<>	XA0 .. XA4	External address lines used to drive memory planes. Note these lines must be used in conjunction with isolation of the memory planes.
>	XREG	External control of registers; when asserted signals must be be provided from the external world, when not asserted it serves to monitor control and address lines.
>	tshift_bit	External data to shift through CC→RP→WP.
>	tshift	External control to shift data through the registers.

Table 5: Pins associated with testing the pointer ciruitry

6.1.2. Pointers. Isolation of the pointer circuitry which comprises of the read pointer, the write pointer and the current capacity pointer, is done in a manner similar to that of the memory planes. For this case, all control lines from the Timing and Control module are driven from the external world giving the user the ability to test at the functional level. Fig. 26 shows the three registers isolated from the rest of the Packet Buffer circuitry. Table 5 lists the pins dedicated to testing the three pointers.

We implemented the three registers using a LSSD technique. This feature enables us to serially load the contents of a register and have it trickle through the registers. Specifically, the Current Capacity register may be serially loaded on the $i\phi_1, i\phi_2$ clock and by the appropriate number of shifts, the contents of the Current Capacity register can be shifted into the Read Pointer and then finally into the Write Pointer. The bits from the registers are brought out to pins and can then be

watched for correct functional, as well as, timing level operation.

6.1.3. Timing/Control Module. During normal operation, the control signals generated by the timing and control module work their way to the outside world. It is only during a memory test mode or a pointer test mode that the signals are brought in from the outside. All the control signals generated by this module are routed to bi-directional pins. In addition to the control signals listed in Table 4 and Table 5, the signals in Table 6 are monitored as well.

Of the available 108 pins around 50 pins are dedicated to monitoring/driving of internal signals.

6.2. TINY CHIPS

Although it is possible to monitor various points on an integrated circuit using a wafer probe, we felt this alternative would be fruitless since access to a probe would be expensive and next to impossible. The layout of a 32 packet buffer encompassed the largest die size that our fabrication facility provided – $9.2mm \times 7.9mm$. Furthermore, since the design area used up by a circuit is approximately proportional to the square of the cost of fabrication [Pr87] we felt it necessary to confirm pieces of design using smaller and less expensive die sizes. Specifically, the cost of fabricating a chip having a design area of $9.2mm \times 7.9mm$ is on the order of $5,900.00, while the cost of fabricating a $1.1mm \times 2.3mm$ (a TINY CHIP) is $400.00 [USCa].

Three tiny chips having a die size of $1.1mm \times 2.3mm$ are being designed. The first tiny chip contains (i) the 3 pointers, (ii) the ASYNC. write capture circuit and (iii) a set of CLOCK drivers (Fig. 27). The second chip contains 1 bit-plane of the buffer memory (Fig. 28). The third chip is being shared with another project and contains the control and timing circuit (lower PLA shaped circuit in Fig. 29). The identical circuits are being used in the larger chip. Although different fabrication runs will provide varying information regarding power dissipation, speed and circuit limitation, we are hoping to gather "ball-park" figures, and confirm functionality.

Direction	Name	Function
<>	Xcrst	External signal to reset the flop that captures the ASYNC write.
<>	Xcsync	External signal to sample the flip-flop output.
<>	Xssync	External signal to indicate that a `SYNC` was seen.
<>	Xfull	External signal to indicate that buffer is full.
<>	Xemt'	External signal to indicate that buffer is not-empty.
>	XCTL	External control of control/timing circuit; when asserted signals must be be provided from the external world, when not asserted it serves to monitor control lines.

Table 6: Addition pins associated with the testing of the control/timing circuitry

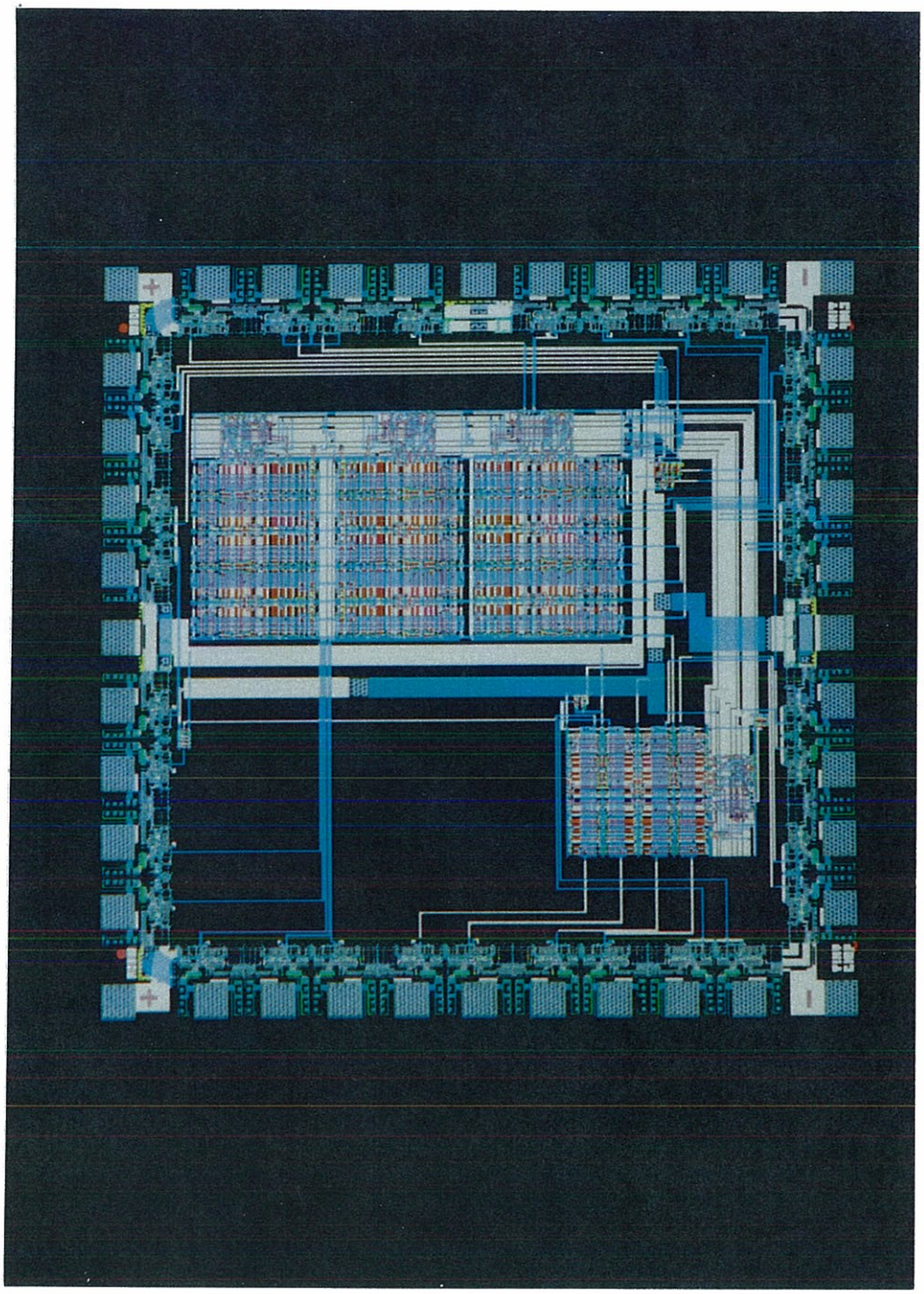

Figure 27: Tiny chip containing the 3 pointers

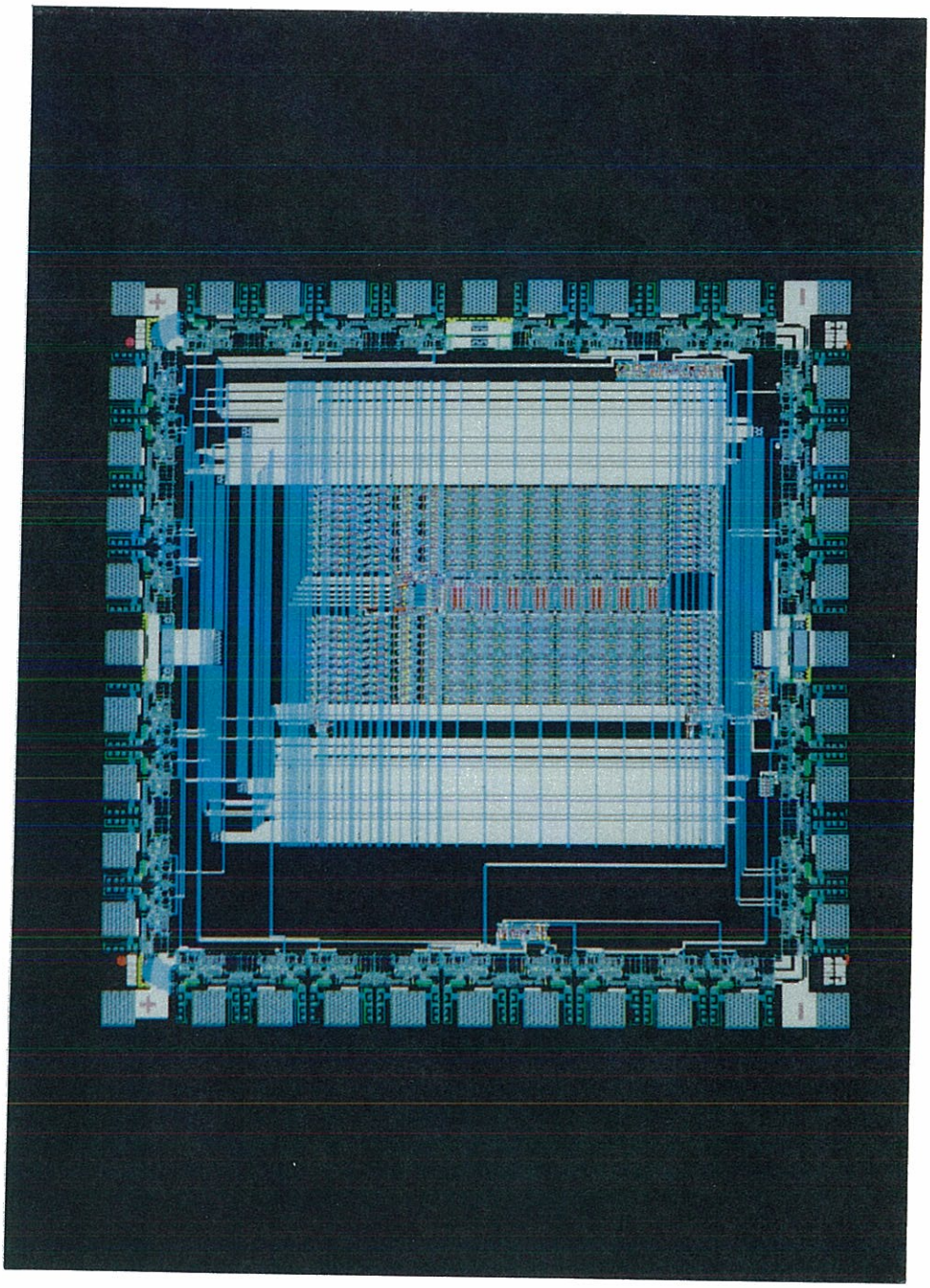

Figure 28: Tiny chip containing 1 bit-plane of memory

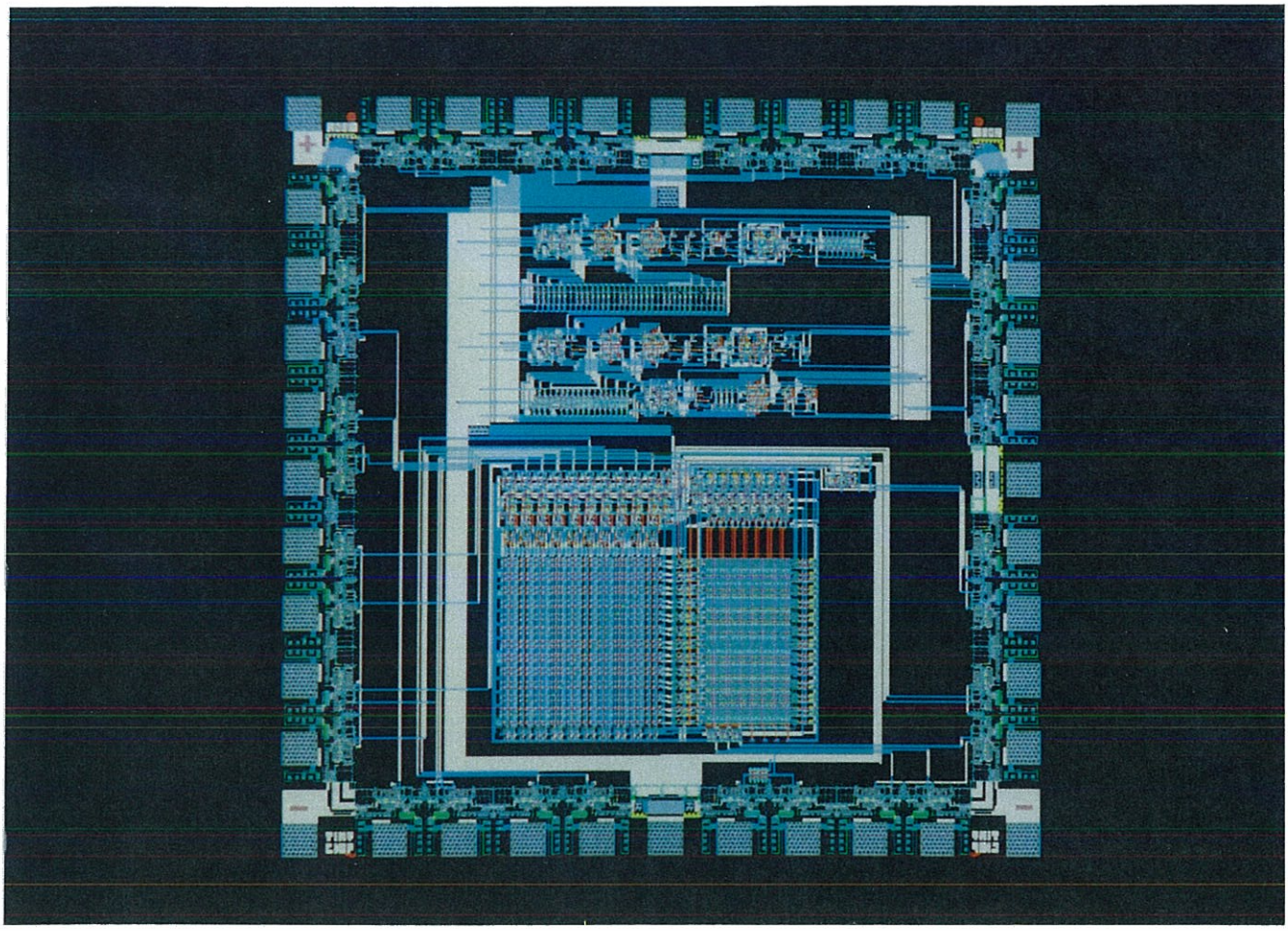

Figure 29: Tiny chip containing the control and timing circuit

References

[MCNC] *VIVID System FACTS Fast Circuit Simulator Designer Tutorial Version 1.3*, and, *VIVID System FACTS Fast Circuit Simulator Designer Reference, Version 1.3*, Microelectronics Center of North Carolina, Research Triangle Park, N.C., 1985.

[MeCo80] Mead, Carver, and Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass., 1980.

[Mu86] Mukherjee, Amar, *Introduction to nMOS and CMOS VLSI Systems Design*, Prentice-Hall, Engelwood Cliffs, N.J., 1986.

[Pr87] Privett Roy F., "The Economics Of Custom Telecommunications VLSI," *IEEE Communications Magazine*, Vol. 25, No. 7, July 1987, pp. 55-61.

[Re83] Reddy, Al, "Dynamic Memory Refresh Considerations," *Application Note AN-887*, Motorola Semiconductor Products Inc., 1983.

[Ro88] Robbert, George, *A Circuit Generator for Synchronized Streams Processors*, Master's Thesis, Washington University Computer Science Department, St. Louis, May 1988.

[RoMo86] Rosenberger Fredrick, Morley Robert, Class Notes from : Digital Integrated Circuits (EE463).

[ScMa86] Scott, Walter S., Robert N. Mayo, Gordon Hamachi, and John K. Ousterhout, ed., *1986 VLSI Tools*, University of California at Berkeley, Computer Science Division (EECS), Report No. UCB/CSD 86/272, Berkeley, Calif., Dec. 1986

[Sh88] Shoji, Masakazu, *CMOS Digital Circuit Technology*, Prentice-Hall, Engelwood Cliffs, N.J., 1988.

[TI86] Texas Instruments, Product Review SN54ALS236, SN74ALS236, 64×4 Asynchronous FIRST-IN FIRST-OUT MEMORY using Advance Low Power Schottky IMPACT technology, October 1986.

[TI87] Texas Instruments, Product Review TACT7202M, TACT7202, 1024×9 Asynchronous FIRST-IN FIRST-OUT MEMORY using Advance CMOS technology, April 1987.

[Tu85] Turner, Jonathan S., *Design of a Broadcast Packet Switching Network*, Washington University Computer Science Department, WUCS-85-4, St. Louis, March 1985.

[Tu86] Turner, Jonathan S., "Design of a Broadcast Packet Network," *Proceedings of IEEE Infocom'86*, IEEE Computer Society, Los Angeles, April 1986, pp. 667-675.

[USCa] *MOSIS User's Manual*, University of Southern California Information Sciences Institute, Marina Del Rey, Calif., (no date)

[VlZh] Vladimirescu, A., Kaihe Zhang, A.R. Newton, D.O. Pederson, and A. Sangiovanni-Vincentelli, *SPICE User's Guide*, University of California at Berkeley, Department of Electrical Engineering and Computer Sciences, Berkeley, Calif., Oct. 1983

[WeEs85] Weste, Neil H. E., and Kamran Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, Reading, Mass., 1985.

A. Pin Assignment and Signal Names

This section documents the pin assignment and signal names in detail. The pins were classified into the following categories : *data in, data out, control in, control out, test* and *power/ground.* This classification is made for all custom VLSI chips in this prototype effort and forms the basis for pin assignment.

Pin #	SIGNAL NAME	PAD	TYPE	FUNCTION
1	sub	blank	—	Substrate
2	uphi1	input	control in	Up stream phase 1
3	tm2	input	control in	Start internal cycle
4	uphi2	input	control in	Up stream phase 2
5	SYNC	input	control in	Asynchronous write
6	tshft	input	test	LSSD shift
7	tsbit	input	test	LSSD data bit
8	Xssync	input/output	test	Internal control signal
9	Xcrst	input/output	test	Internal control signal
10	Xcsync	input/output	test	Internal control signal
11	Xirp	input/output	test	Internal control signal
12	Xr/w'	input/output	test	Internal control signal
13	Xiwp	input/output	test	Internal control signal
14	XCTL	input/output	test	Isolate control circuit
15	Xfull	input/output	test	Internal CC signal
16	Xemt'	input/output	test	Internal CC signal
17	XREG	input	test	Isolate pointers
18	XA0	input/output	test	Address lines
19	XA1	input/output	test	Address lines
20	XA2	input/output	test	Address lines
21	XA3	input/output	test	Address lines
22	XA4	input/output	test	Address lines
23	CC0	output	test	CC register output
24	tpi0	input	test	Test PISO output shift register
25	CC1	output	test	CC register output
26	CC2	output	test	CC register output
27	CC3	output	test	CC register output
28	Vdd	power/ground	power supply	
29	CC4	output	test	CC register output
30	dphi2	input	control in	Down steam phase 2
31	tpi1	input	test	Test PISO output shift register
32	tpi2	input	test	Test PISO output shift register
33	tpi3	input	test	Test PISO output shift register
34	tpi4	input	test	Test PISO output shift register
35	tpi5	input	test	Test PISO output shift register
36	tpi6	input	test	Test PISO output shift register

Pin #	SIGNAL NAME	PAD	TYPE	FUNCTION
37	tpi7	input	test	Test PISO output shift register
38	tpi8	input	test	Test PISO output shift register
39	Vdd	power/ground	power supply	
40	GND	power/ground	ground	
41	empty	output	control out	Buffer empty
42	GND	power/ground	ground	
43	Vdd	power/ground	power supply	
44	ACK	input	control in	Buffer read
45	dd8	output	data out	Data out
46	dd7	output	data out	Data out
47	dd6	output	data out	Data out
48	dd5	output	data out	Data out
49	dd4	output	data out	Data out
50	dd3	output	data out	Data out
51	dd2	output	data out	Data out
52	dd1	output	data out	Data out
53	dd0	output	data out	Data out
54	dphi1	input	control in	Down stream phase 1
55	GND	power/ground	ground	
56	dsync	input	control in	Downstream synchronization
57	OE	input	test	By pass internal output enable
58	qdld	output	test	Qualified downstream load
59	qdsh	output	test	Qualified downstream shift
60		blank		Unused
61		blank		Unused
62		blank		Unused
63		blank		Unused
64		blank		Unused
65	padin	input	test	Direct connection to output pad
66	padout	output	test	Direct connection to output pad
67	invout	output	test	Output of an inverter chain(5)
68	XMEM	input	test	Isolate memory
69	Xprech	input/output	test	Internal control signal
70	Xwr	input/output	test	Internal control signal
71	Xmacc	input/output	test	Internal control signal
72	Xmil	input/output	test	Internal control signal

Pin #	SIGNAL NAME	PAD	TYPE	FUNCTION
73	Xlddr	input/output	test	Internal control signal
74		blank		Unused
75		blank		Unused
76		blank		Unused
77		blank		Unused
78	ud0	input	data in	Data in
79	iphi3	input	control in	Internal phase 3
80		blank		Unused
81		blank		Unused
82	GND	power/ground	ground	
83	XRST	input	control in	Chip reset
84	iphi1	input	control in	Internal phase 1
85	ud1	input	data in	Data in
86	ud2	input	data in	Data in
87	ud3	input	data in	Data in
88	ud4	input	data in	Data in
89	ud5	input	data in	Data in
90	ud6	input	data in	Data in
91	ud7	input	data in	Data in
92	ud8	input	data in	Data in
93	Vdd	power/ground	power supply	
94	GND	power/ground	ground	
95	full	output	control out	Buffer full
96	GND	power/ground	ground	
97	Vdd	power/ground	power supply	
98	wdt	output	control out	Watch Dog timer
99	tso8	input	test	Test SIPO input shift regiser
100	tso7	input	test	Test SIPO input shift regiser
101	tso6	input	test	Test SIPO input shift regiser
102	tso5	input	test	Test SIPO input shift regiser
103	tso4	input	test	Test SIPO input shift regiser
104	tso3	input	test	Test SIPO input shift regiser
105	tso2	input	test	Test SIPO input shift regiser
106	tso1	input	test	Test SIPO input shift regiser
107	tso0	input	test	Test SIPO input shift regiser
108	iphi2	input	control in	Internal phase 2

B. PGA packaging

The Kyocera KD-P87938-A pin grid array package has a 0.450" cavity(topside). The package is square with each side having length 1.200" and pins arranged on a 12 × 12 pin grid at 0.1" centers. There are three full rows of pins around the outside. The following layout shows the view from the top of the package, with the index at the upper right corner.

```
                                                                    X
                                                                   /
     M       L       K       J       H       G       F       E       D       C       B       A
   +-------------------------------------------------------------------------------------------+
   |                                                                                           |
 1 | GND                                     Xmacc   XMEM    padin           qdsh    OE      GND     |
   | (82)    (81)    (80)    (77)    (74)    (71)    (68)    (65)    (62)    (59)    (57)    (55)    |
   |                                                                                           |
 2 | iphi1   XRST    iphi3           Xlddr   Xwr     invout                  qdld    dsync   dphi1   |
   | (84)    (83)    (79)    (76)    (73)    (70)    (67)    (64)    (61)    (58)    (56)    (54)    |
   |                                                                                           |
 3 | ud2     ud1     ud0             Xmil    Xprech  padout                  dd2     dd1     dd0     |
   | (86)    (85)    (78)    (75)    (72)    (69)    (66)    (63)    (60)    (51)    (52)    (53)    |
   |                                                                                           |
 4 | ud5     ud4     ud3                                                     dd5     dd4     dd3     |
   | (89)    (88)    (87)                                                    (48)    (49)    (50)    |
   |                                                                                           |
 5 | ud8     ud7     ud6                                                     dd8     dd7     dd6     |
   | (92)    (91)    (90)                                                    (45)    (46)    (47)    |
   |                                                                                           |
 6 | full    GND     Vdd                                                     GND     Vdd     ACK     |
   | (95)    (94)    (93)                                                    (42)    (43)    (44)    |
   |                                                                                           |
 7 | wdt     Vdd     GND                                                     Vdd     GND     empty   |
   | (98)    (97)    (96)                                                    (39)    (40)    (41)    |
   |                                                                                           |
 8 |tso6     tso7    tso8                                                    tpi6    tpi7    tpi8    |
   |(101)    (100)   (99)                                                    (36)    (37)    (38)    |
   |                                                                                           |
 9 |tso3     tso4    tso5                                                    tpi3    tpi4    tpi5    |
   |(104)    (103)   (102)                                                   (33)    (34)    (35)    |
   |                                                                                           |
10 |tso0     tso1    tso2    tshft   Xcrst   Xr/w'   Xfull'  XA0     XA3     tpi0    tpi1    tpi2    |
   |(107)    (106)   (105)   (6)     (9)     (12)    (15)    (18)    (21)    (24)    (31)    (32)    |
   |                                                                                           |
11 |iphi2    uphi1   uphi2   tsbit   Xcsync  Xiwp    Xemt'   XA1     XA4     CC1     CC4     dphi2   |
   |(108)    (2)     (4)     (7)     (10)    (13)    (16)    (19)    (22)    (25)    (29)    (30)    |
   |                                                                                           |
12 | sub     tm2     SYNC    Xssync  Xirp    XCTL    XREG    XA2     CC0     CC2     CC3     Vdd     |
   | (1)     (3)     (5)     (8)     (11)    (14)    (17)    (20)    (23)    (26)    (27)    (28)    |
   |                                                                                           |
   +-------------------------------------------------------------------------------------------+
     M       L       K       J       H       G       F       E       D       C       B       A
```