A Self-Timed Chip Set for Multiprocessor Communication

Douglas L. Whiting

Computer Science Department
California Institute of Technology

5000:TR:82

CALIFORNIA INSTITUTE OF TECHNOLOGY

Computer Science Department

Technical Report 5000

A Self-Timed Chip Set For Multiprocessor Communication

by

Douglas L. Whiting

Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Science

February 24, 1982

Table of Contents

List of Figures

ABSTRACT

This paper describes a family of chips used to link multiple processors together on a speed-independent communication bus. Sendership arbitration is included as an integral part of the signalling scheme, incurring very little overhead and providing a measure of fairness. The protocol allows for one-to-many communication in which the sender must wait for all receivers to respond to each datum transmitted. The width of the data bus is arbitrary, and only three control wires are necessary for normal transmission cycles. In order to alleviate congestion, the global bus may be divided into several local buses by a method which is entirely transparent to the processor software. Thus the bus topology may be reconfigured for each processing network using these chips as building blocks.

Functional verification of speed-independent circuits is also discussed. The problem is seen to be very complex, but some conclusions are drawn about the type of tools which will be helpful in implementing self-timed systems.

1. INTRODUCTION

1.1 Motivation

An initial aim of this project was to determine what type of self-timed building blocks might be useful for the VLSI age. One of the first conclusions reached was that since small synchronous processing systems can be built cheaply and reliably in today's technology, any immediate contributions from the self-timed discipline would have to come at a higher level than that of a small system such as a microprocessor. It became clear that to achieve the goal of having self-timed VLSI building blocks, the issue of communication must be squarely addressed.

Self-timed signalling schemes have many advantages over synchronous designs in connecting multiple processing units, since the composition of self-timed components can be specified entirely by the interconnection topology, without regard to the electrical parameters that determine worst-case timing. Thus it is much simpler to construct large computing systems if each subsystem has a speed-independent interface.

Historically, the most significant attempt to define a speed-independent processing framework was the macromodules [3]. The internal logic of the modules was implemented in standard ECL, mostly SSI, and each macromodule was a small box which could be inserted into a rack. Typical module functions were arithmetic operations, Boolean operations, memory, registers, and control flow (forks, joins, etc.). The overall system function was determined wholly by the topology of interconnection, some of which was implicit in the placement within the rack and the rest of which was explicitly determined by data and control cables. One nice feature was the ease with which concurrent

processing could be implemented. No knowledge about timing constraints was needed, and the only rule was, "if it fits, it works." Unfortunately, the concept of macromodules does not map well into the era of VLSI, where many module functions can be placed on a single chip, and random interconnection by cable is impossible.

More recently, various speed-independent bus schemes have been proposed to allow independent processors to communicate with one another. An example is the TRIMOSBUS [7], which allows for any sender on the bus to send a message to one or more receivers, and wait for the last of them to respond.

1.2 Results

One product of this research is a bus communication technique which is quite flexible and (for the most part) transparent to the processors. A typical application is for message communication between microprocessors. Each sender specifies a destination for its message, and, since receivers may have shared aliases, there is the capability of one-to-many communication.

The bus is speed-independent. On each data transmission cycle of the message, the sender interface waits until all participants have signalled receipt of the data before initiating the next cycle. By employing such a speed-independent signalling scheme, any number of ports may be put on the bus without loading problems.

In addition to the goal of finding useful self-timed building blocks, another purpose of this research was to produce an exemplary self-timed design in MOS. The design was done in a top-down fashion. In the following chapters, the bus specifications are presented in the order in which they were

devised: the global bus structure, the signalling conventions, and the implementation in MOS. Finally, it was hoped that this experience in self-timed design would provide insight into the type of tools needed for functional verification of such designs, and chapter five presents some of the conclusions reached in this area.

2. BUS STRUCTURE

2.1 The IP Chip

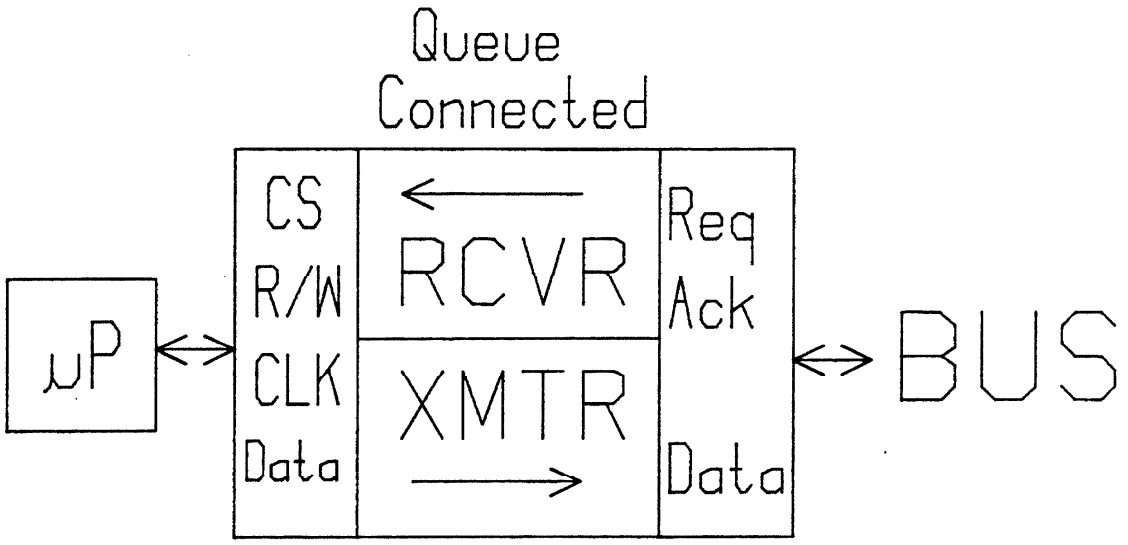

Figure 2-1: The IP Chip

The fundamental unit of the bus is the interface-to-processor (IP) chip (fig.2-1). On one side, it looks just like an ordinary microprocessor peripheral, with clock, chip select, read/write, address, and data pins. On the other side it is connected to the bus. Actually the chip consists of two almost independent halves, the receiver and the transmitter, which are queue-connected between the bus side and the processor side. Outgoing messages are written into the transmitter queue by the microprocessor and are subsequently sent asynchronously on the bus. When a message comes into the receiver queue, a processor interrupt is generated, and the message may then be read.

Multiple IP chips are connected together on the bus side to form a communication pathway. This bus may be thought of as a wheel, in which

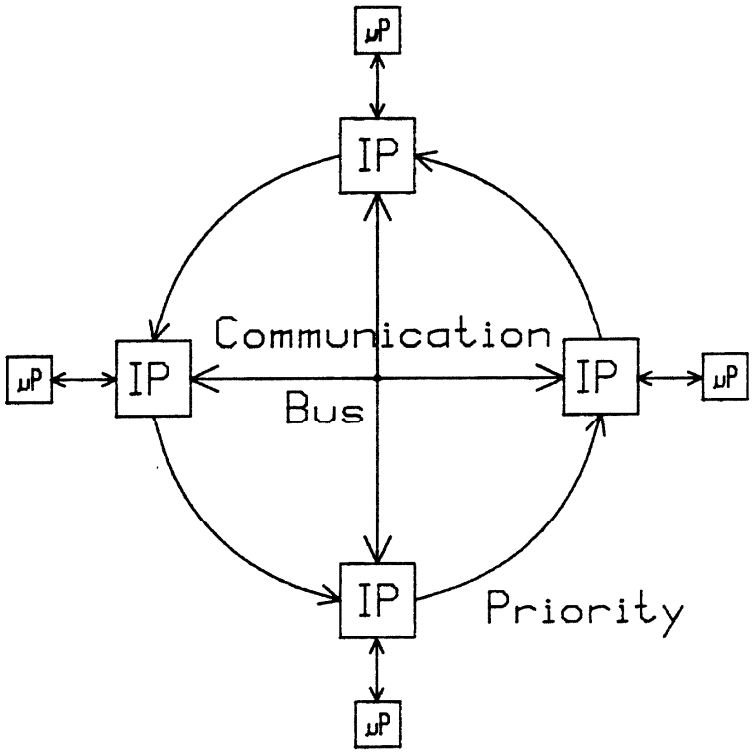

Figure 2-2: A Local IP Bus

sendership priority is passed serially around the rim, and communication takes place in parallel through the hub (fig.2-2).

Upon obtaining sendership, control logic in the IP chip determines if there is a message in the transmitter queue. An arbiter is used to make the decision, so the necessary synchronization between the processor side and the bus side will never exhibit synchronization failure[5]. If the queue is empty, sendership is immediately relinquished to the next IP chip in the ring; otherwise, one message is sent, and then priority is passed. This ring arbitration scheme inherently provides a measure of fairness: since at most one message will be sent each time an IP obtains sendership, every bus

participant is assured that it will receive sendership within a finite time.

2.2 Message Protocol

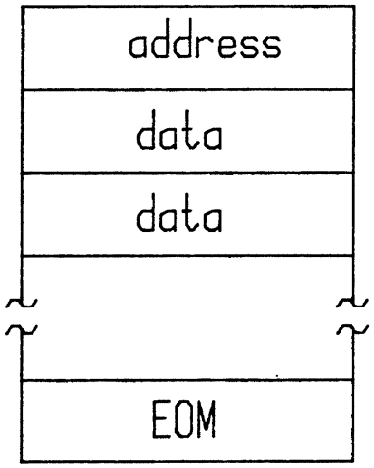

Figure 2-3: Message Format

Each message consists of a destination address, followed by the data, and concluded by a special end-of-message (EOM) character (fig.2-3). Note that the width of the communication path is arbitrary, as the signalling scheme is independent of bus width. For example, there could be eight data wires, plus one wire for codes such as EOM. This implementation would enable byte-parallel communication and in addition would allow for up to 256 special codes. There would not be a need for so many special codes, so they could be rather loosely encoded.

Each processor gives to its IP(s) an identity, which consists of an address and an address mask. This initialization defines a set of addresses to which each IP will respond. Since each IP may have several aliases, it is possible to share these addresses with other IPs, thus allowing multiple receivers for a single message. In particular, if there were eight (or less) IPs on the bus with eight data wires and each IP masked all but one (unique) bit of the

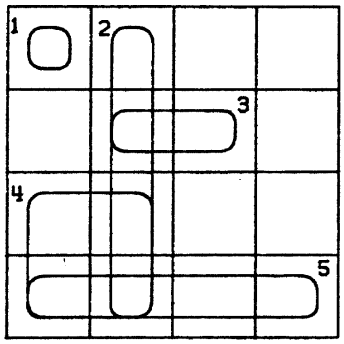

Figure 2-4: Alias Sharing

address, then a message could be directed to any subset of the IPs. In general, however, alias sharing should be optimized for the communication flow expected in the network. The alias overlap can be seen by plotting each IP's identity on a Karnaugh map. An example is shown in figure 2-4 for a data bus width of four. In this example there are five IPs on the bus, and messages can be directed to any of the following subsets: {1;2;3;4;5;2&3;2&4;2&4&5; 4&5}

At the beginning of each message, the receiver examines the destination address and decides if it is one of the intended receivers for the message. If so, the ensuing data will be put into the queue; if not, the receiver waits for the next message to begin.

2.3 The F-Box

As IPs are added on the ring, message traffic on the bus may become congested. To alleviate this problem, the F-box (F=filter or forward), another member of the chip family (fig.2-5), may be inserted to split the global bus into smaller local buses. Thus it filters the bus activity on one side from the other by forwarding only inter-bus messages. If the processors

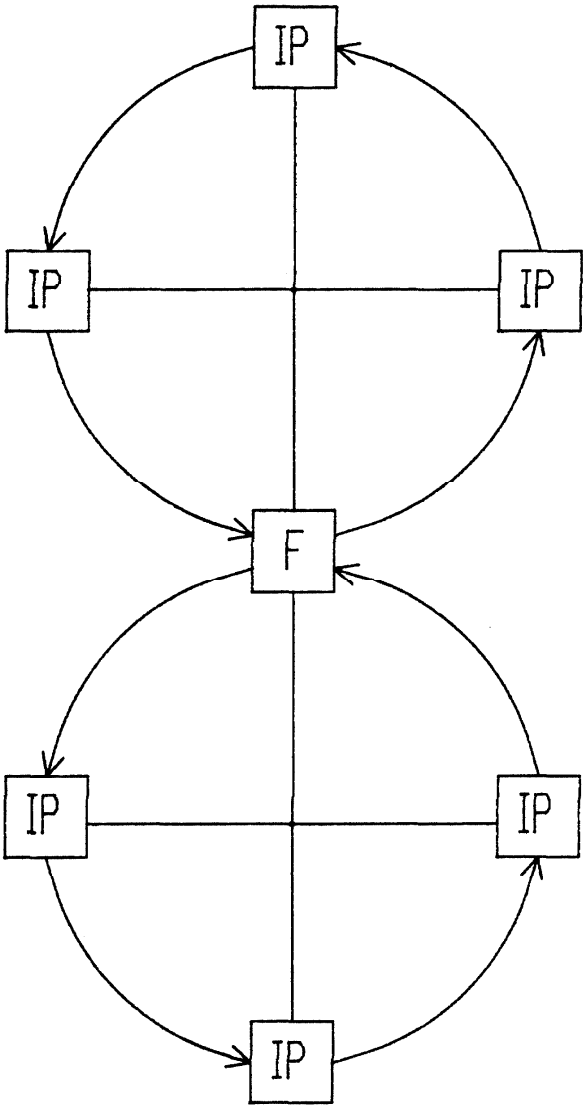

Figure 2-5: A Global Bus Using the F-Box

can be split into groups with dense communication within the group and sparse communication outside the group, the congestion is relieved by use of the F-box.

The F-box may be thought of as two bus halves of an IP chip stuck back to back. The priority ring is broken on the global bus and reconnected to form independent rings, one for each local bus. Each F-box or IP on the local bus is then termed a port. Note that the F-box actually has two ports, one on each local bus to which it is connected.

2.4 Message Routing

The routing of inter-bus messages was one of the first issues addressed in the bus specifications. The first alternative was to provide explicit routing information in the message itself. In this case there could be multiple routes available, and the processors would have to choose the desired route. Although such a scheme imposes no constraints on the bus topology, it would require each processor to have a knowledge of the topology.

The other choice was to insure that there is a unique route for each message, thus relieving the IPs and the processors of the necessity of knowing how the global bus is structured. This latter alternative was chosen because of its 'transparency'. A few of the consequences of this decision are examined below.

First, unique routing imposes a constraint on the topology of the bus network. If each local bus is viewed as a vertex and each F-box as an edge of an undirected graph, there must be no closed cycles on the resulting graph (fig.2-6). Otherwise, it would be possible for a message to be forwarded

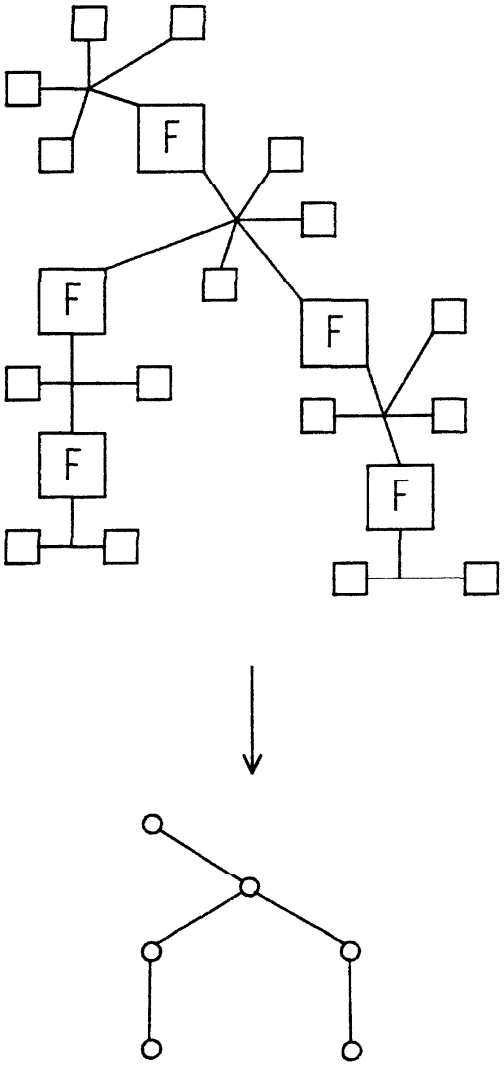

Figure 2-6: Global Bus Topology

indefinitely by the F-boxes around a cycle. In fact, the only connection between the receiver and transmitter halves of the F-box is due to the necessity of inhibiting the receiver when the transmitter is active, so that this problem can be avoided.

Also, since routing is entirely left up to the F-boxes, they must learn the location of each receiver. This information is distributed by a very simple mechanism. After a system reset, each IP chip must send out a message consisting of a list of its aliases. This message is distinguished by a special code used as the destination address. These messages are ignored by the IP chips but are forwarded by the F-boxes, which also take note of 'who is where' and store the information on-chip. After the power up sequence is complete, the F-boxes know from the address whether to forward a message across buses.

Perhaps the most important result of the unique route scheme is that the F-box is entirely transparent to the processors! That is, since the processor is unaware of how messages are routed, no processor code changes are required if an F-box is inserted in a bus. This allows great flexibility in design, since radical changes in bus topology may be made without affecting the software.

2.5 Deadlock Avoidance

One system issue which must be handled carefully is that of deadlock. Suppose for example that one of the recipients of a message has no room in its queue. This receiver cannot acknowledge without losing the data, and thus the entire local bus must wait until there is room in the queue. If the queue will eventually be emptied, there is no danger of deadlock, although bus

throughput will suffer. However, suppose the receiver is an F-box, attempting to forward the message to another local bus. If both sides of the F-box are waiting for their queues to empty before giving an acknowledge, the two local buses are deadlocked.

A straightforward solution is for each intended receiver to determine at the beginning of the message if there is room in its queue to hold the entire packet, and if there is not, the attempt to send the message must be refused until the next time the sender is given priority. The simplest way to implement such a scheme is to legislate a maximum message length. If a receiver determines that there is not room in its queue for a maximum length message, it intiates a negative acknowledge cycle, which informs the sender (and all other receivers) that this message transmission must be deferred. Other mechanisms were considered to circumvent the problem, but all were much more complicated and showed little promise of higher performance.

2.6 Observations

The modularity and flexibility offered by this communication structure are truly impressive. Note that the scheme is independent of the signalling convention involved and indeed could work effectively even for a synchronous protocol. Also, no mention has been made of what type of processor is involved. Thus, for example, the bus could be used to connect multiple processors to a shared resource such as memory (fig.2-7a) by a suitable modification to the processor side of the IP chip. The shared aliases may be dynamically reconfigured, and it is possible to statically reconfigure the bus topology to accomodate changes in number of processors and communication bandwidth without affecting the software.

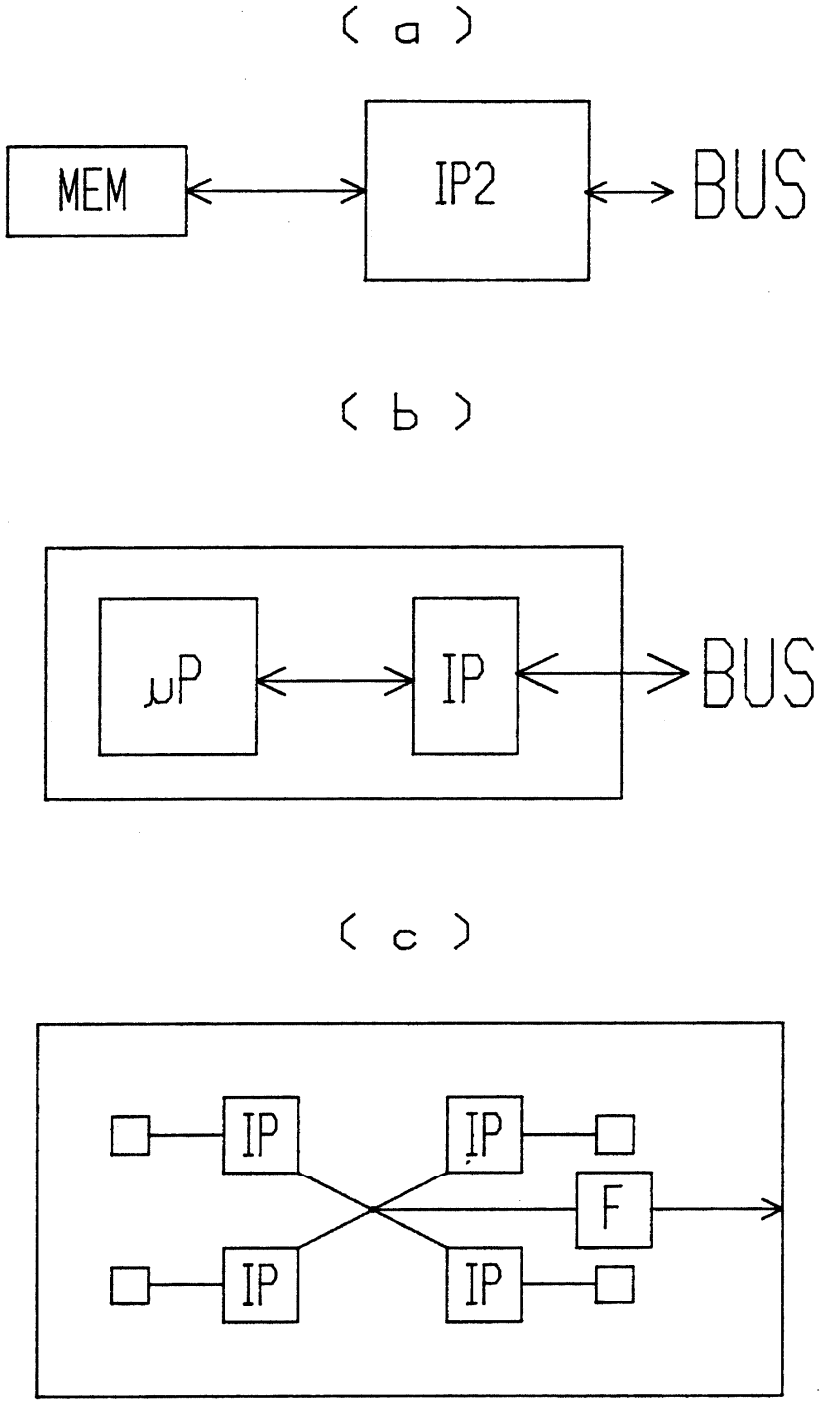

Figure 2-7: Possible Extensions of the Bus Concept

Furthermore, this bus concept should remain viable as VLSI enables more and more function to be put on one chip. For example, if the processor were implemented as a single chip, it would be very attractive to include the bus interface on-chip also (fig.2-7b). In fact, if several such processors could be placed on a single chip, the bus scheme could be used quite effectively inside the chip, and an F-box connection brought out to connect multiple chips (fig.2-7c). Such a scheme would take advantage of the inherent disparity in speed between on-chip and off-chip communication. Many alternatives are conceivable which fit well into the general framework.

3. SIGNALLING CONVENTION

3.1 Wires

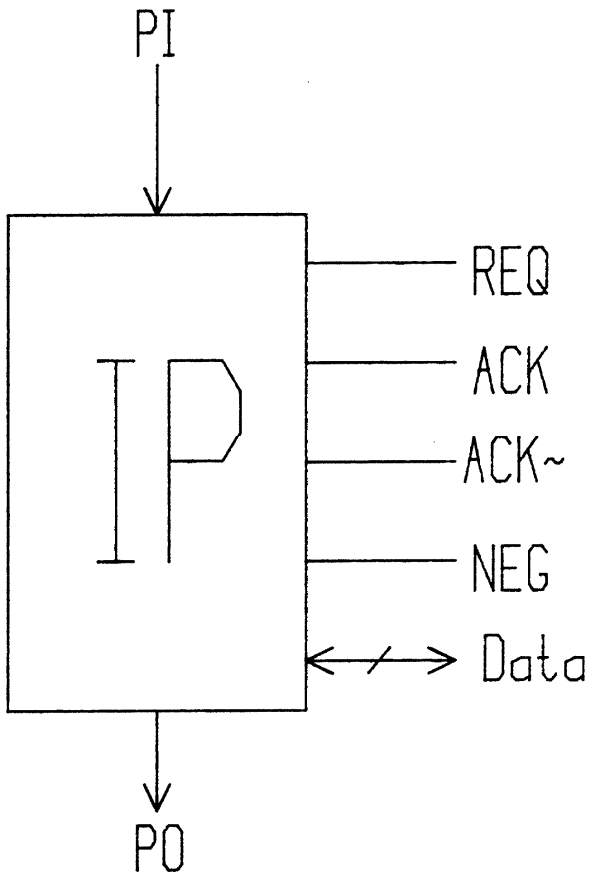

Figure 3-1: IP Signal Wires

The several signal wires used in this self-timed protocol can be grouped into data, priority, and control wires (fig.3-1). The number of data wires is arbitrary, and they require no special termination. Each port on the local bus has one priority in (PI) and one priority out (PO) wire, which are connected in a ring as described below. The first control wire is the request line (REQ), used by the sender to notify the system that there is valid data available. This is a tri-state line and must be terminated with a negative

resistance so that it will retain its value even when no sender is driving it. There are two acknowledge lines (ACK and ACK~), which are wire-anded, and are used to return a system acknowledge to the sender that the data has been read and a new value may be placed on the bus. Note that the basic request/acknowledge scheme requires three wires, just as in the TRIMOSBUS, and indeed the signalling here is somewhat similar in flavor, although easier to implement. Finally, the wire-anded negative acknowledge line (NEG) is used to refuse a message if an intended receiver does not have room in its queue.

Although the signalling is speed-independent, in this implementation each local bus is assumed to be an equipotential region (fig.3-2). Thus when the transmitter senses that the data are valid, it can assume that the receivers see the same values on the wires. Actually, the condition is slightly weaker: as long as the transmission delay on the data wires is less than or equal to the delay on the request wire, the assumption holds. Note that, if needed, the F-boxes can be used as 'repeaters' to extend the effective range of the equipotential region. If this solution were not acceptable, the protocol would have to be encoded into the data, for example by a double-rail scheme [6], which would add the requirement of a negative resistance on the data lines. In the equipotential case, the only wire requiring negative resistance is the Request line.

3.2 Arbitration

The sendership arbitration mechanism described in the previous chapter requires only two pins per port, regardless of the number of ports on the bus. A transition on the priority input pin notifies the port that it is now the sender and may transmit a message if one is available. When the message

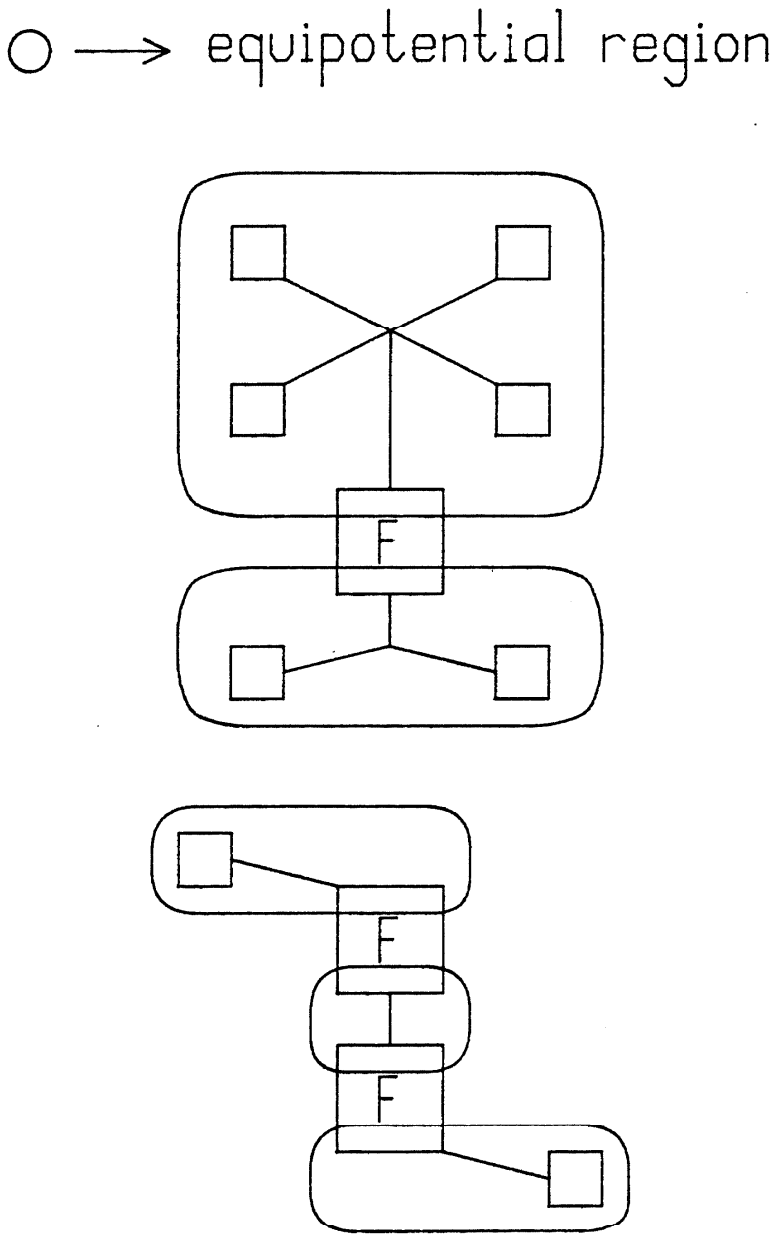

Figure 3-2: Local Equipotential Assumption

transmission is complete, or if there is no message to be sent, the port causes a transition on the priority output pin, passing sendership to the next port on the ring. Note that 'receive-only' ports may be excluded from the ring, thus removing any unnecessary overhead.

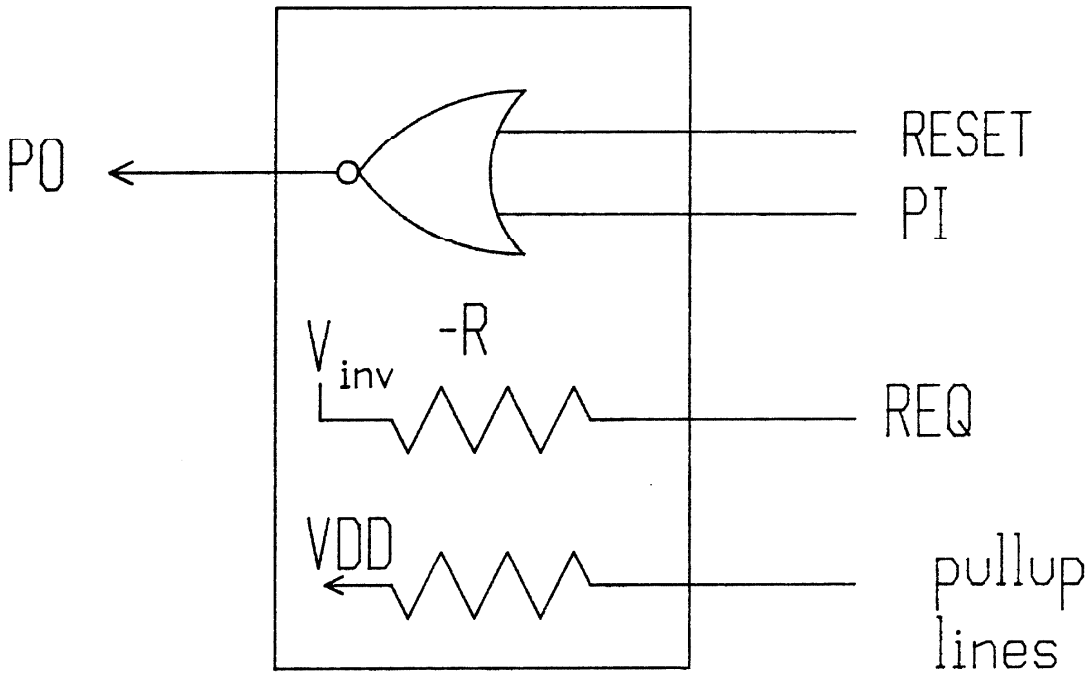

Figure 3-3: The Bus Controller Chip

Somewhere in the ring there must be an inverter to insure that sendership is rotated indefinitely. This function will be performed by an extremely simple member of the chip family, the bus controller chip (fig.3-3). On system reset, to allow the priority ring to stabilize, this SSI chip will source a logical zero. Once the reset is complete, the nor output switches high, and the priority ring essentially becomes a ring oscillator. This chip will also provide negative resistance on the request line and pullup resistors for the wire-anded acknowledge lines.

3.3 Request/Acknowlege Cycles

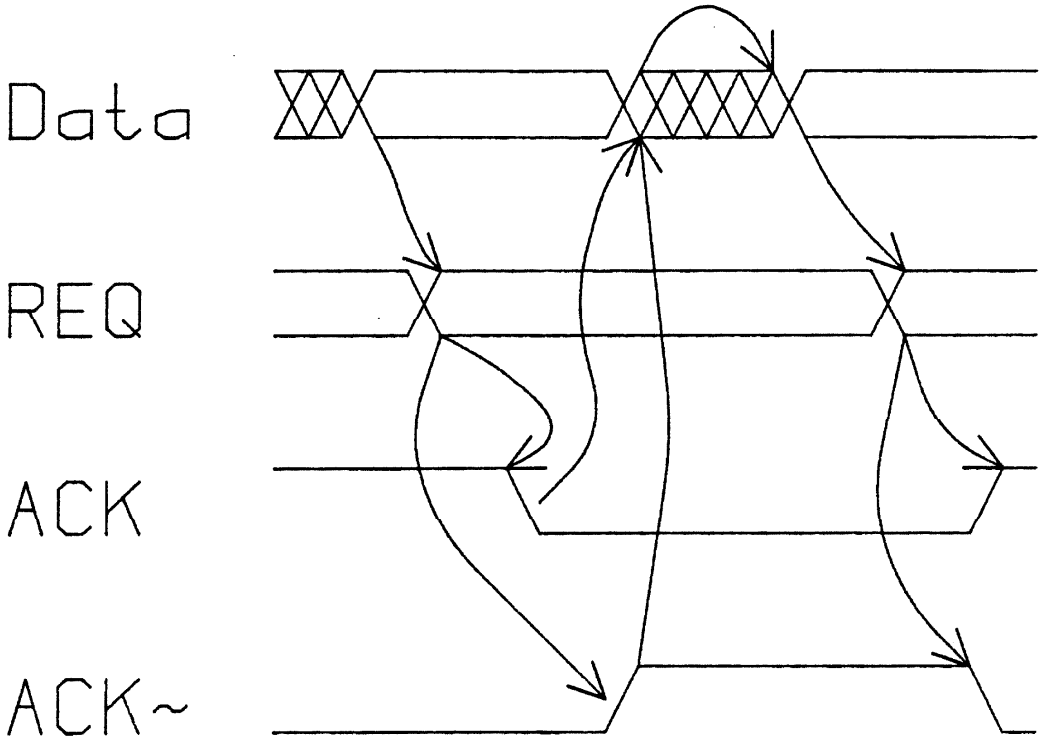

Figure 3-4: Normal Data Transmission Cycle

Once a port has obtained sendership, each word (address, data, and EOM) is sent in the following manner (fig.3-4). The sender first drives the data pins until they are seen to be valid (this is where the equipotential assumption is important). Some hysteresis is added to the comparator to help insure that the data lines are well past the threshold voltage of all devices on the bus. The sender continues to drive the data lines until the system acknowledge is received, thus providing an additional margin of safety. Next, a transition is put on REQ to notify all bus participants that there is valid data on the bus.

Each receiver drives ACK with its two-cycle acknowledge, and uses the complement to drive ACK~; thus the system acknowledge occurs only after both

acknowledge lines have changed state. This scheme effectively computes the Muller C function of all the receiver acknowledges. Note that a port must participate in the acknowledge even if it is ignoring the present message. Internally, receivers and transmitters use a four-cycle scheme, but this is converted to transition signalling at the pins to enhance performance. After the EOM has been sent, the sender relinquishes priority to the next port on the ring.

At the receiver end, the most interesting activity takes place on the address cycle of each message. The previous EOM (or system reset) sets a flag which is used to detect the address cycle. Each receiver compares the address with its own identity to determine if it is to be a recipient of the ensuing message. If there is not a match, the receiver ignores the packet, passively acknowledging each datum until an EOM is seen. Otherwise, two possibilities arise. When there is room in the receiver queue for a maximum length message, a normal acknowledge sequence is followed, and the receiver then puts the message into its queue.

However, if there is not room in the queue, a negative acknowledge cycle (fig.3-5) is initiated in order to avoid deadlock, as was explained in the previous chapter. After pulling NEG low, the receiver waits until it senses that the line has gone low, and then gives a normal acknowledge. Each time a system acknowledge is received, the sender samples NEG. If there is a negative acknowledge, the message is immediately terminated with a special EOM cycle (special in the sense that the EOM word is not put into any receiver queue), and sendership is relinquished to the next port on the ring. After this EOM cycle, the receiver releases the NEG line, and all ports must delay

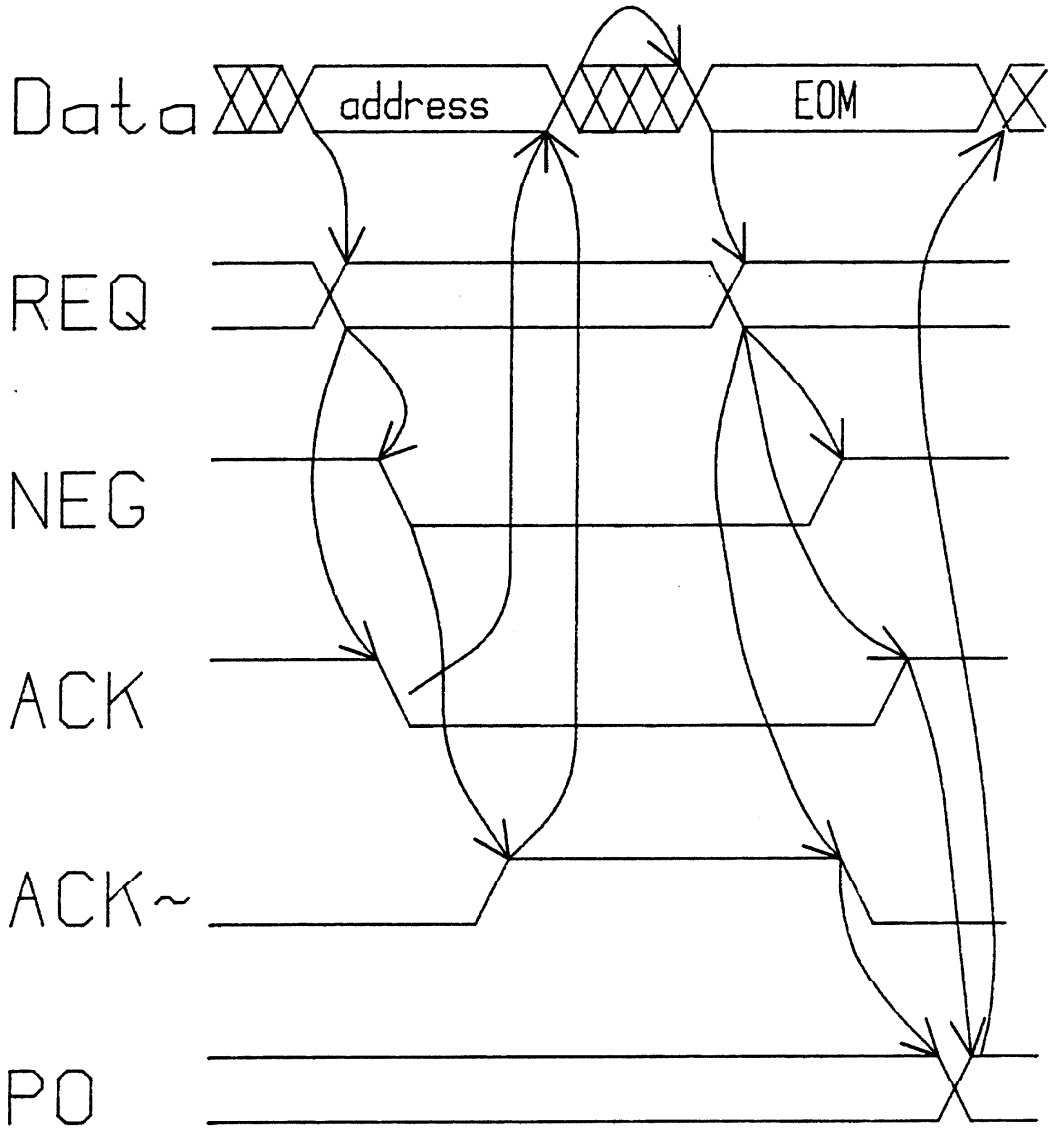

Figure 3-5. Negative Acknowledge Cycle

their acknowledge of the following address cycle until they have seen NEG return to its inactive state. Another attempt will be made to send the message the next time the port obtains sendership. Note that the assumption of an equipotential region is invoked here, because the system must see the transition on NEG before the system acknowledge is received. However, adapting this scheme to a non-equipotential system would only require that the system give an acknowledgement that it had seen the NEG transition before the receiver could give its acknowledge. Obviously the equipotential case is much more economic and thus was used in this implementation.

3.4 Observations

The above protocol is very efficient in its use of wires and transitions. For normal communication cycles, only three control wires are used, and only three transition times are required: one to drive valid data onto the bus, one to put a transition on REQ, and one to respond on ACK and ACK~ (changes take place concurrently on these two lines). Unlike the TRIMOSBUS, the control wires have the same function on each cycle, which greatly simplifies design of the control logic. Debugging and testing are accomodated by the fact that the bus can be single-stepped with pulldowns on the acknowledge lines; such a facility could very easily be incorporated into the bus control chip. Sendership arbitration is automatically included and incurs very little overhead. Despite the simplicity and parsimony of this scheme, it provides a fairly complex communication mechanism which will proceed at the maximum speed allowed by the prevailing load and operating conditions.

4. DESIGN OF THE IP CHIP

4.1 Design Methods

A gate and MOS transistor level design of the IP chip has been completed, and efforts are underway to have the chip laid out and fabricated in MOS. In order to insure the speed-independence of the signalling scheme, it was decided to approach the chip as a self-timed circuit from the bottom up. Only in a few instances were concessions made to chip area by invoking a knowledge of the actual timing involved. Very limited use was made of formal asynchronous logic design techniques, since they tend to be rather combinatorial and do not take advantage of the pass structures available in MOS. An intuitive approach was found to be more effective for design, but formal techniques were used for analysis and verification.

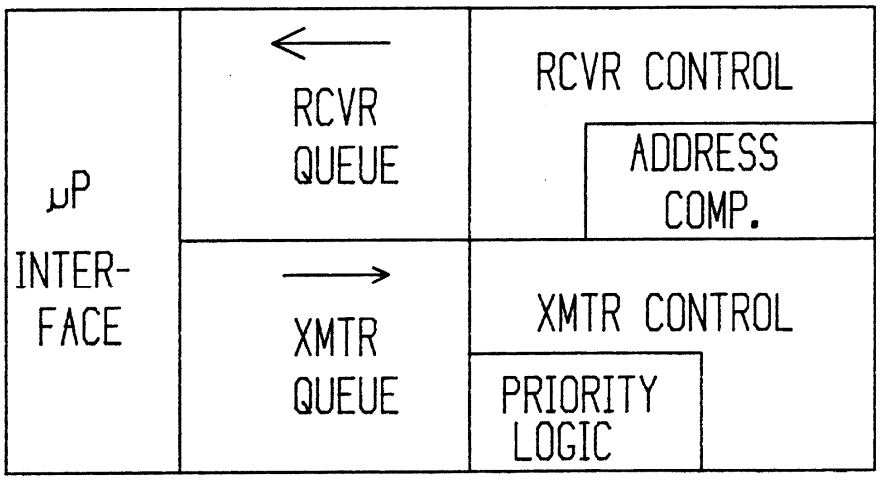

Figure 4-1: IP Chip Floor Plan

The design split rather naturally into small modules with a minimal number of interconnections (fig.4-1). These sections are: processor interface logic, queues, address comparator, receiver control, priority logic, and transmitter control. Several advantages result from this modularity. First, because the

number of interconnections is small, each module may be fabricated as a separate chip, which greatly aids in testing the design, even though the final system will be put on a single chip. Also, each module is simple enough to be drawn easily on a single sheet of paper; thus the control flow may be understood intuitively for each section.

This modularity allows the use of abstraction, one of the most powerful tools available in the design of self-timed systems. After functional verification of a module, its internal characteristics can be abstracted away, leaving only the behavior at its external interface. The function of the composition of several such modules can then be verified using only the abstracted behavior of each module. For example, an arbiter is a fairly complex analog circuit, but once it is certified to work, one may confidently use arbiters in a design without concern about the internal workings of the circuit.

4.2 Parts

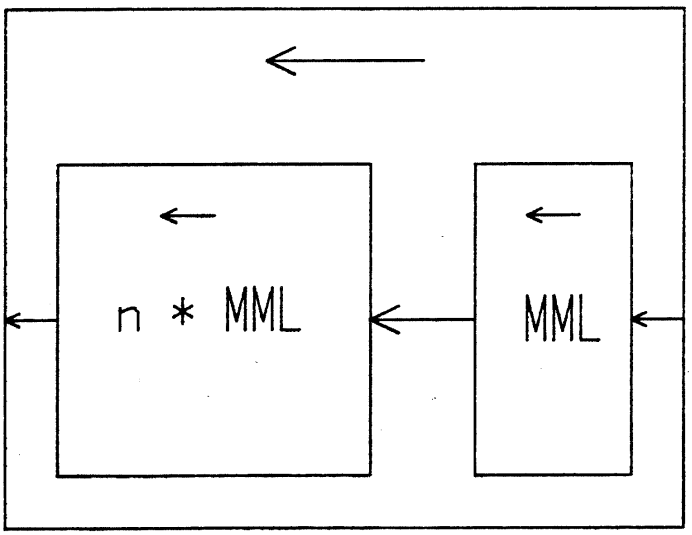

Figure 4-2: Emptiness Detection in the Receiver Queue

The queues are implemented as RAM-based FIFOs with speed-independent control logic. To maximize circuit density, the RAM cells are dynamic and only the control logic is static. Design of the FIFO reduces to creating one slice which can then be cascaded to form queues of arbitrary length. The queue length is left unspecified at present, although this parameter will have a major impact on the maximum message length (MML). In order to detect easily whether there is room for a maximum length message, the receiver FIFO will be built in two stages (fig.4-2), the first of which will be of length MML and thus will only need to be tested for emptiness. This first stage feeds into a second queue, which provides additional buffering and will probably be several times longer.

The processor interface is very elementary, consisting of a few latches, a decoder, and some random logic. This interface is assumed to be synchronous, and it will be the responsibility of the processor to insure that setup and hold times are met. Fortunately, the logic is simple enough that this should present no limitation for conventional processors. This section is one of the rare pieces of conventional design on the chip.

The address comparator is used in the receiver during the address cycle of each message, and is a good example of abstraction. Note that, in the receiver design below, the comparator is shown as a box with only its external connections made explicit.

By far the most complicated parts of the chip were the receiver and transmitter control modules, which required several design iterations before their correctness was verified. Use of four-cycle signalling on chip greatly simplified the design, and the two-cycle interface at the pins was easily

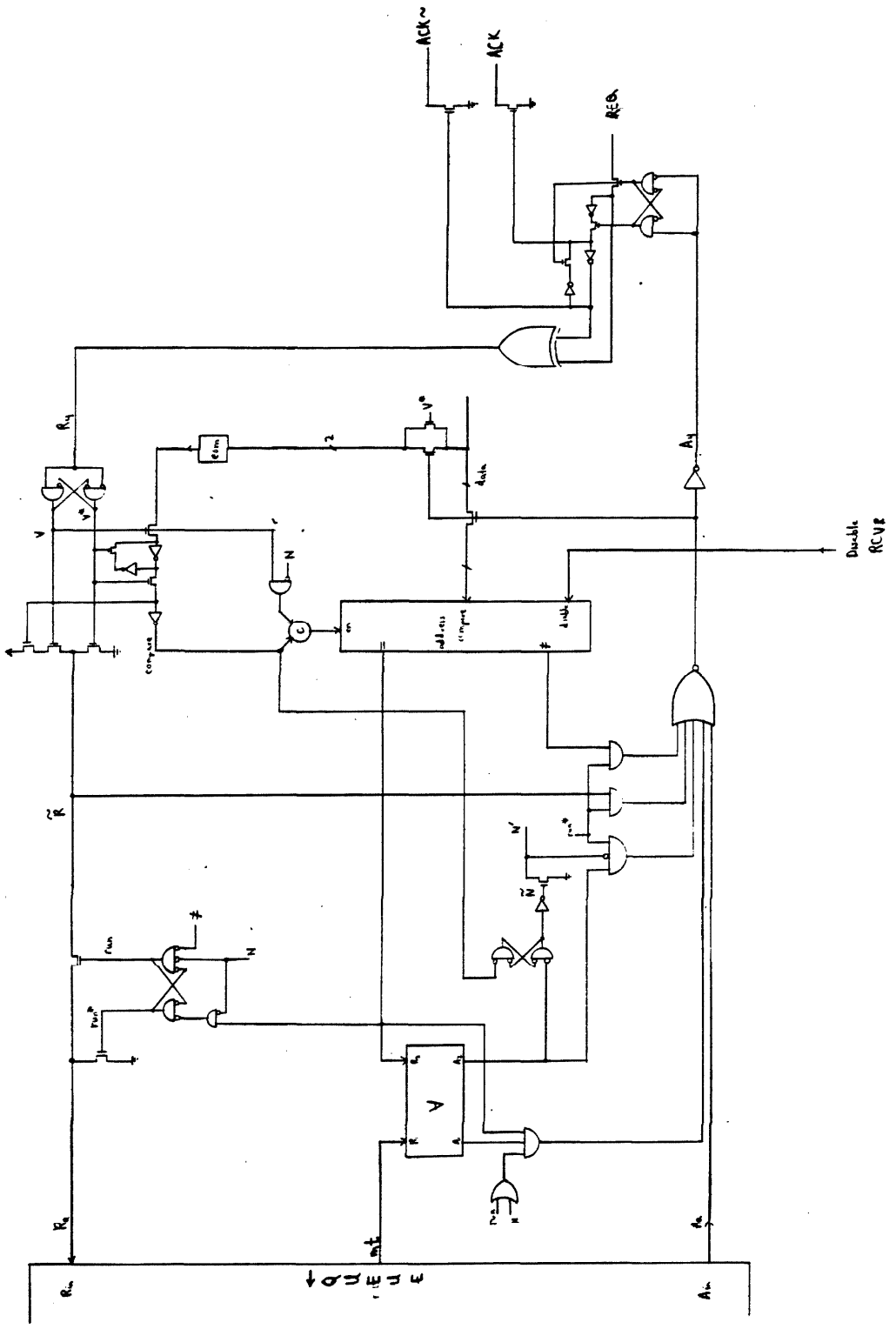

Figure 4-3: Receiver Logic Design

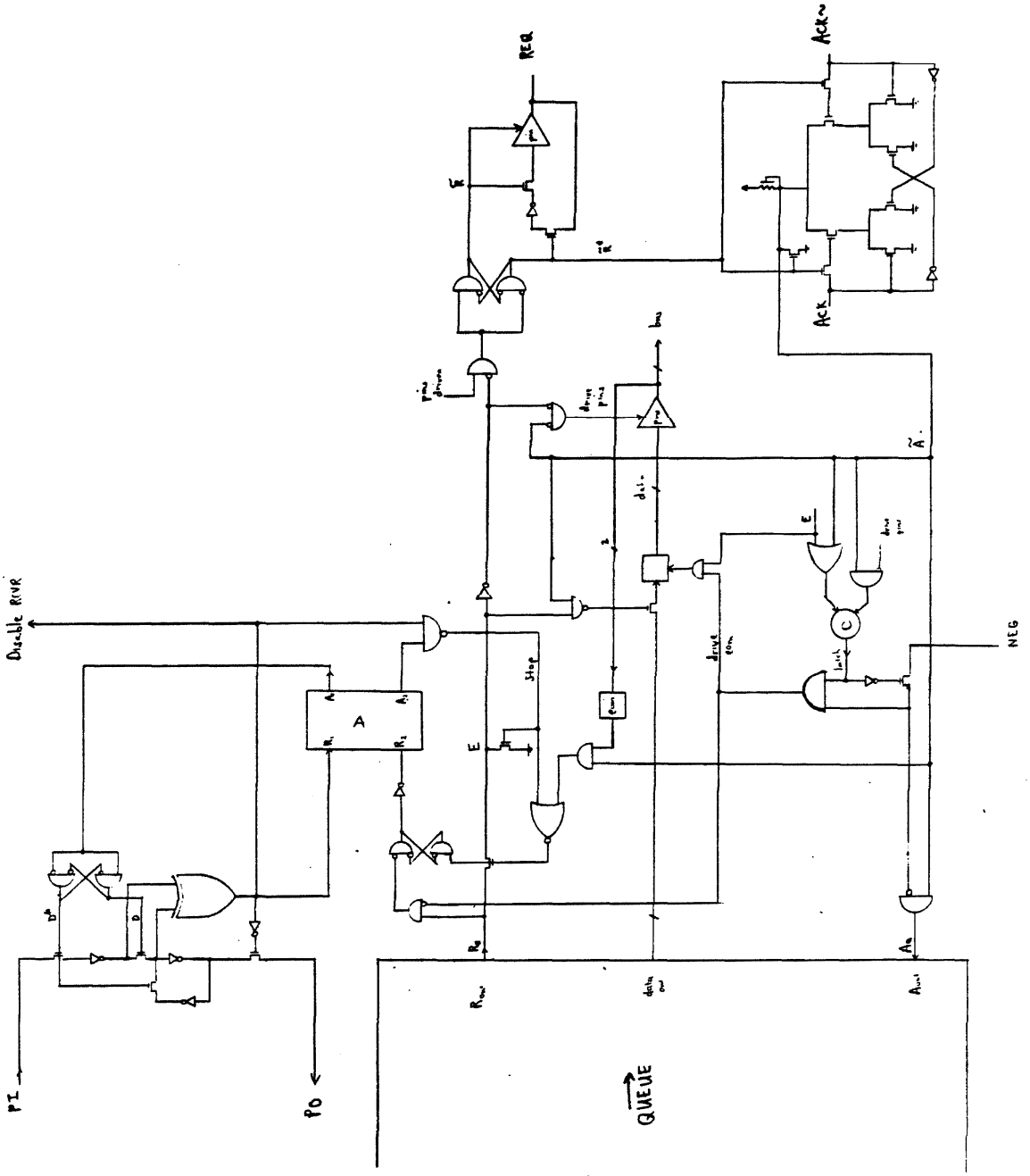

Figure 4-4: Transmitter Logic Design

implemented with the aid of MOS dynamic storage nodes. No attempt is made here to explain the internal workings of these sections, as they are rather involved. However, the reader may find it valuable to study these designs (figs. 4-3, 4-4) to get a feel for how self-timed logic maps into MOS.

The priority logic section (upper left portion of fig.4-4) is small enough that the entire functional verification is presented concisely in the following chapter, giving an excellent indication of how the verification was done for the more complex modules.

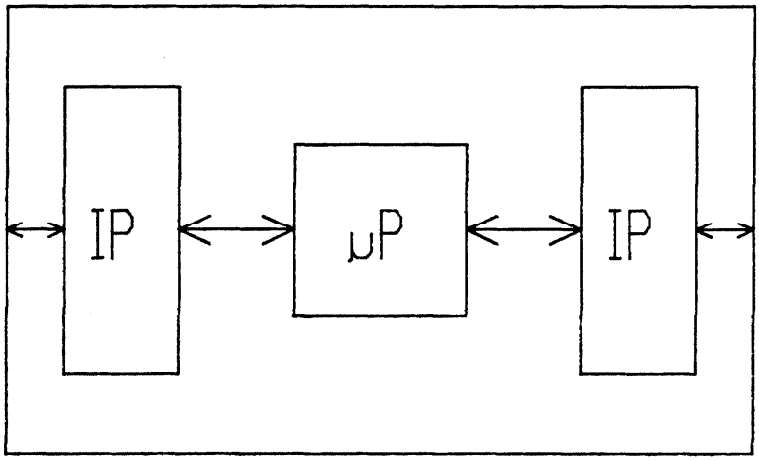

Figure 4-5: An F-Box Constructed from IP Chips

The extensive use of abstraction should be apparent in these designs. At every interface between different sections, only the external behavior of the modules is important. It is also important to note that almost all of these modules can be used unchanged in the design of the F-box. In fact, the only modification necessary is that, in the F-box, the address comparator becomes a RAM which is used to store the topology information for the bus. Conceptually this is only a minor change, although the implementation will require substantial effort. Note that an F-box (albeit a slow one) can be implemented

as two IP chips with a processor in between to accomplish the forwarding (fig.4-5). Thus the IP chip alone can be used to test the validity of this bus concept.

5. VERIFICATION TECHNIQUES

5.1 An Example

Once an integrated circuit is fabricated, internal nodes are virtually inaccessible, and it becomes exceedingly difficult to locate the cause of functional errors, or even to determine whether the problem is in the design or in the layout. Because mistakes must be found and eliminated before the chip is actually fabricated, design and verification must be addressed as a single issue. Generally, the functional verification of an integrated circuit is accomplished by an analog circuit simulator or a logic simulator, using information either supplied by the designer or extracted from the layout itself. Unfortunately, since such tools implicitly model a circuit in the time domain, neither technique is useful for speed-independent circuits.

Before describing some possible methods of automating the verification process, perhaps it would be helpful to explain the manual technique used to verify the design of the IP chip. The basic algorithm is to make a sequence diagram (not a timing diagram) of the levels of each wire in the circuit, starting from some initialized state. Arrows show the precedence relation between events, and transitions which cannot be ordered by the arrows are concurrent. This method is illustrated for the priority logic section of the chip in figure 5-1.

Note that in the sequence sense, the acknowledge from the arbiter follows the request input, although there may be activity on the other half of the arbiter which takes an indefinite period of time. Concurrent activity is represented by transitions occurring in the same vertical 'slice' of the diagram. No transitions, other than those shown, are possible; thus the

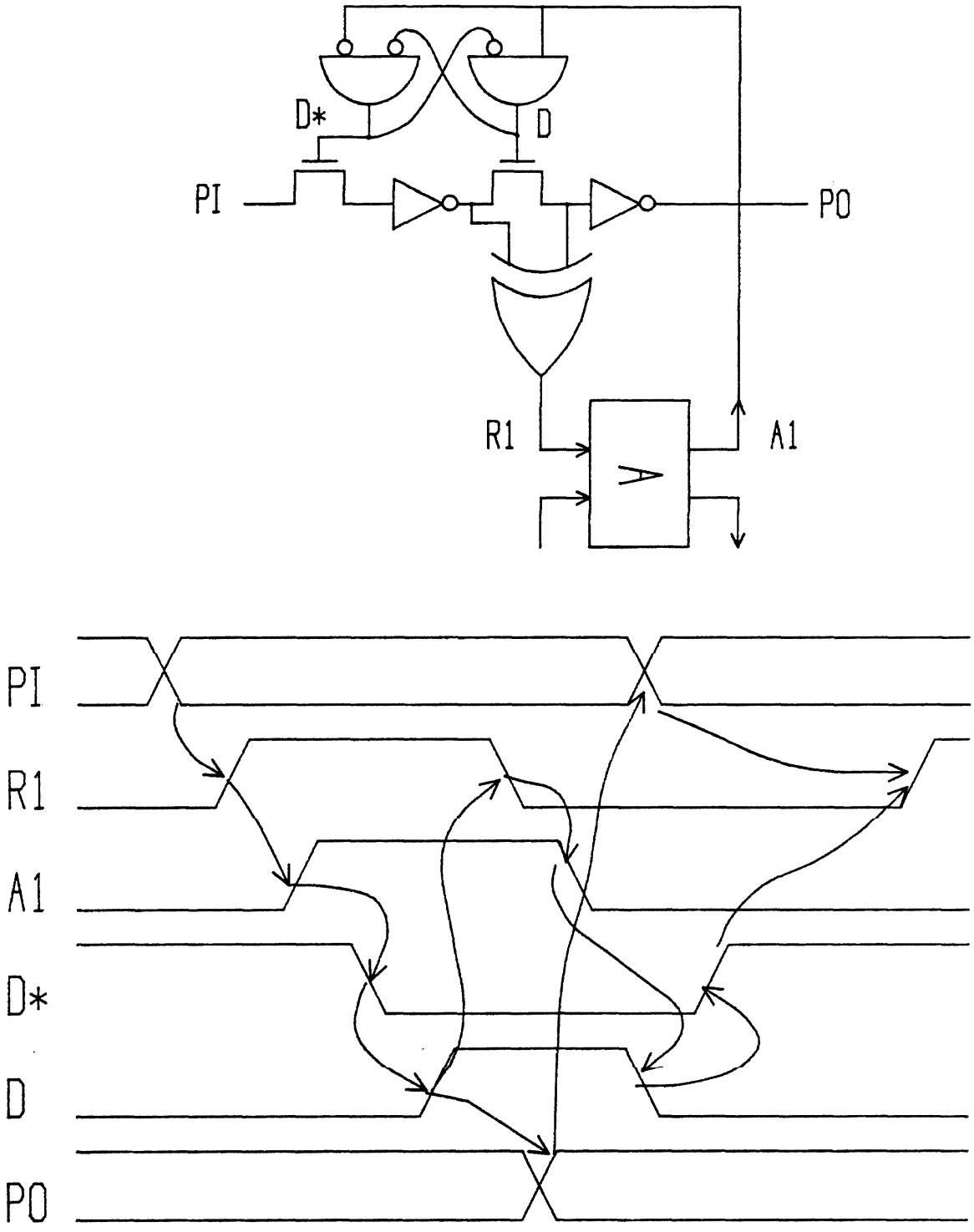

Figure 5-1: Verification of the Priority Logic

entire circuit behavior has been demonstrated. Unfortunately, this method requires a large amount of intuition to see which transitions are enabled, and also to realize that the circuit has returned to its original state, implying further simulation is unnecessary. However, the designer can see from such a diagram that the circuit is performing as intended, and, by excluding all internal wires and invoking the transitivity of the 'precedes' relation, the external behavior of the circuit may be abstracted for use at a higher level.

5.2 Obstacles

Obviously there is a need for a simulator for speed-independent circuits, but it is unclear what algorithm could be used to implement such a simulator. There are several approaches, each involving an attempt to characterize all possible sequence behaviors of the circuit, assuming arbitrary delays in the gates and/or wires. The problem can easily become combinatorial for all but the simplest circuits, such as those in which there is only one possible sequence of events (e.g.- a suitably initialized ring oscillator). Another difficulty that arises in this characterization is the question of when to terminate the simulation. For example, a ring oscillator has a infinite sequence behavior; however, this infinite behavior can easily be described as repetition of a finite sequence. Therefore some means of pattern recognition must be included to stop the simulation reliably.

Concurrent activity is the root of the problem. If a circuit has no possible concurrent transitions, there is only one sequence behavior. One way to prohibit concurrency is to disallow the forking of wires in a circuit, so that one transition can cause at most one other transition, and then initialize the circuit to a state in which there is only one transition

enabled. However, such a limitation is hardly satisfactory, and concurrency is very desirable for high performance in integrated systems.

To get an idea of the magnitude of the problem, suppose a certain transition enables n other independent chains of events, each chain consisting of k ordered transitions; after all nk events there is a 'join', and the activity again proceeds serially. The possible number of sequences is then $(nk)!/(k!)^n$, which grows faster than $(n!)^k$. As an example, for n=2, k=3, there are twenty possible behaviors. Further, if several such fork-joins occur in series, the total number of sequences is the product of the individual numbers. Obviously this rate of growth is totally unacceptable, so no successful algorithm can handle the problem in a brute force manner.

A subtlety should be noted at this point. The analysis of the priority circuit above was done for the case of arbitrary gate delay, not arbitrary wire delay. It is interesting to realize that the priority logic circuit can misbehave in the sequence sense if the latter is assumed. From a practical standpoint, the model should only assume arbitrary gate delay, treating the elements as if they all resided in an equipotential region. Where the equipotential assumption breaks down, it is the responsibility of the designer to insert 'ghost' buffer elements, effectively informing the simulator that a wire delay is to be modelled. Thus, although the question of wire delay versus gate delay is a fundamental one, it can be handled very well with a little insight on the part of the designer.

There is a much more difficult problem to account for in a simulator, however. It was noted in the previous chapter that assumptions about the sequence behavior can occasionally be deduced from a knowledge of circuit

timing parameters. For example, in the receiver section of the IP chip there is a two-input NOR gate which decodes the EOM character. It is implicitly assumed that the output of this gate will be valid before the system acknowledge is received and the data on the bus is changed (otherwise the NOR gate requires longer to change state than two off-chip transition times). Obviously this is a safe assumption from a practical standpoint. In making a sequence diagram, the designer can show such an assumption by an 'orange arc', an arrow drawn with a different color to indicate that the precedence is enforced only in the timing domain, not in the sequence sense. However, the simulator does not 'know' that such an assumption was made and will show the consequences of this assumption being false as a possible misbehavior of the circuit. Specifying an orange arc to the simulator is very complicated, since it involves placing sequence constraints on independent elements, which violates the principle of arbitrary delay. No satisfactory method of incorporating orange arcs into an automatic verification tool has yet been devised.

Even ignoring orange arcs, the problem is still substantial. A major difficulty arises in specifying exactly what characterizes the correct behavior. The designer certainly knows what he intends the circuit to do, but how can the simulator intuit that the observed behavior matches the desired behavior, particularly when the intended sequence is usually specified only at the external interface?

5.3 Possible Approaches

One method is to list any incorrect behaviors and have the simulator search the possible sequences to insure that these behaviors are never observed.

This approach amounts to proving theorems about the circuit, and logic programming languages allow such concepts to be expressed quite elegantly. Unfortunately, the idea has a few major drawbacks. First, circuit specifications are generally not given in terms of what it doesn't do, and the designer would have to be certain to list all incorrect behaviors for the simulator to verify that a design functions properly. Also, such a technique requires an exhaustive search through all possible behaviors, which was shown above to be intolerable for most circuits of interest.

Ternary logic simulation has also been suggested as a solution by Bryant [1]. The algorithm is linear in circuit size and is very effective at locating critical races in synchronous systems. Unfortunately, the underlying model is so conservative that it flags any unclocked feedback path as a critical race, and it is not clear whether this difficulty can be overcome.

Perhaps the most promising approach is to generate the possible behaviors in a way which efficiently exhibits the concurrency (thus avoiding the combinatorial explosion), allowing the designer to decide if the circuit functions correctly. One method is the use of Petri nets, which provide a very concise medium for describing concurrent activity. The gate level description of a circuit may be mapped onto a Petri net, and simulation consists of firing the enabled transitions [4]. Also, an algebraic method of dealing with this type of simulation has been developed by Young-Il Choo [2]. The main advantage of these methods is that they incorporate the 'shuffle' of events as a single sequence. For example, the two distinct sequences (ab,ba) are considered as one sequence (a||b). One can see that these methods avoid the combinatorial explosion encountered in the brute force enumeration technique.

Speed-independent circuit implementation requires much iteration between design and verification. For the IP chip, the iteration time was quite long because verification was done manually. For larger designs this delay would become intolerable, although abstraction could relieve some of the burden. An automated self-timed circuit simulator would be invaluable, enabling the effect of any modification to be seen very quickly, and drastically cutting design time. Also, abstraction could be effectively utilized by verifying each module and then extracting its external behavior for use by the simulator at a higher level. It seems likely that the simulator will be fairly interactive, utilizing the designer's intuition to interpret and guide the progress of the simulation. Research in this area will hopefully prove fruitful in the future.

6. CONCLUSIONS

The major goals of this research were to discover self-timed building blocks that would be useful in VLSI, to produce an exemplary self-timed design in MOS technology, and to gain insight into the design and verification of self-timed systems.

In the first area, a family of chips has been proposed to implement a speed-independent communication bus. Because the signalling is speed-independent, a bus may be composed from these chips without concern for electrical problems. By considering both system aspects and the signalling scheme, the bus specifications allow for a flexible yet powerful interconnection of multiple processing units (or other resources) with the capability of one-to-many comunication. Great care must be taken at the system level in choosing a topology which maximizes the communication bandwidth, but it is clear that the chips do form a useful set of building blocks for multiprocessing systems.

The design of the IP chip has been presented, along with an overview of how the design was verified. The modularity of this design allows fabrication of the various parts to facilitate testing, and most of the F-box design can be copied directly from these modules. The IP will soon be fabricated as part of a multi-project chip.

The question of verification methods remains a very difficult one. Unfortunately, conventional techniques of circuit simulation are not applicable to self-timed systems, and since formal asynchronous design techniques prove unwieldy for large systems, intuition and abstraction are often the only tools available for both design and verificaiton. There is a

great need for a simulator which can concisely present all possible sequence behaviors of a circuit. The combinatorics of the problem dictate that exhaustive enumeration of all behaviors is unfeasible, so some method must be developed which takes concurrent activity into account in the sequence domain, not in the time domain. Some progress has been made in this area, but no workable algorithm has yet been discovered.

ACKNOWLEDGEMENTS

Special thanks are due to my advisor, Chuck Seitz, for his continued guidance (and zealous proofreading) in this research. Most of the ideas presented here were conceived during discussions held in his office, and his enthusiasm for this project was truly infectious. In particular, the idea of a two-port device (which became the F-box) was his inspiration, and this concept was the key to developing such a flexible bus structure.

I also wish to thank Randy Bryant and Young-il Choo, with whom I spent many hours discussing the problem of verification of self-timed systems. Many thanks to everyone in the computer science department, for their willingness to help and to be friendly.

I gratefully acknowledge the support of my wife and daughter, who have always encouraged my research efforts. They also make home a nice place to be.

The research described in this thesis was sponsored in part by the Defense Advanced Research Projects Agency, ARPA Order number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597.

REFERENCES

[1] R. E. Bryant. A Switch-Level Simulation Model for Integrated Logic Circuits. PhD thesis, MIT, 1981.

[2] Y. Choo. Concurrency Algebra: Towards an Algebraic Semantics of Petri Nets. Display File #4085, Caltech, December, 1980.

[3] W. A. Clark. Macromodular Computer Systems. AFIPS Conference Proceedings 30:335-6, Spring, 1967.

[4] C. L. Seitz. Asynchronous Machines Exhibiting Concurrency. In Proceedings of the Project MAC Conference on Concurrent Systems and Parallel Computation. ACM Conference Record, Woods Hole, Massachusetts, June, 1970.

[5] C. L. Seitz. System Timing. In Introduction to VLSI Systems by Carver A. Mead and Lynn A. Conway, Chapter 7, pages 236-242. Addison-Wesley, 1980.

[6] C. L. Seitz. System Timing. In Introduction to VLSI Systems by Carver A. Mead and Lynn A. Conway, Chapter 7, page 248. Addison-Wesley, 1980.

[7] I. E. Sutherland, C. E. Molnar, R. F. Sproull, and J. C. Mudge. T h e Trimosbus. In Proceedings of Caltech Conference on VLSI, pages 395-427. Caltech, January, 1979.