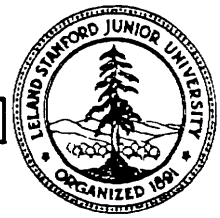
COMPUTER SYSTEMS LABORATORY

DEPARTMENTS OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
STANFORDUNIVERSITY · STANFORD, CA 94305



CONSISTENCY IN INTERPROCESSOR COMMUNICATIONS FOR FAULT-TOLERANT MULTIPROCESSORS

Peter Lincoln Fu

CRC Technical Report No. 81-10

(CSL TR No. 219)

September 1981

CONSISTENCY IN INTERPROCESSOR COMMUNICATIONS FOR FAULT-TOLERANT MULTIPROCESSORS

Peter Lincoln Fu

CRC Technical Report No. 81-10

(CSL TR No. 219)

September 1981

CENTER FOR RELIABLE COMPUTING
COMPUER SYSTEMS LABORATORY
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

This work was supported in part by the Air Force Office of Scientific Reseach under Contract No. F49620-79-C-0069 and in part by the National Science Foundation under Grant. No. MCS-7904864.

CONSISTENCY IN INTERPROCESSOR COMMUNICATIONS FOR FAULT-TOLERANT MULTIPROCESSORS

Peter Lincoln Fu

CRC Technical Report No. 81-10
(CSL TR No. 219)

September 1981

CENTER FOR RELIABLE COMPUTING
COMPUTER SYSTEMS LABORATORY
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California

ABSTRACT

Consistency among processors is vital for fault-tolerant multiprocessors. This report describes modular communication interprocessor interface units which implement distributed consistency schemes such that failures within a single processor module cannot affect the consistency of data transferred among the remaining processors. Furthermore, one scheme provides concurrent and consistent self-diagnostic data on the integrity of the units themselves. Another scheme is tolerant to almost all failures within two processor modules. The theory of the schemes are explained and their implementations in LSI circuits are described in detail. The interprocessor communication structure defined by any of these schemes serves well as a critical element in highly reliable multiprocessor systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I. INTRODUCTION

A. Motivations

As more computers are put to use in critical elements in such applications as aircraft flight controls, spacecrafts, operation of mass transit systems, power plants and others involving large investments or safety of human lives, the reliability and availability of these computers are of utmost importance.

Redundancy is the means to achieve fault tolerance in computers. The use of error detecting and correcting codes in data transfer or storage is an example of redundancy at a low level in a computer system. As hardware cost goes down and processing power per module is increased, the use of replication at the computer module level in a fault-tolerant system becomes very attractive. Examples abound, such as Pluribus [Kas78] for highly available communications processing, Tandem [Kaz78; Bar78] for "Non-Stop" on-line transactions processing applications, SIFT [Wen78] and FTMP [Hop78] for commercial aircraft flight-critical control.

With the use of increasingly higher level modules as the basic units for redundancy as in these multiprocessor systems, new problems arise. The most crucial of these is the problem of maintaining consistency among the processing units in the presence of faults. Areas in which consistency is vital in a multiprocessor system include

the concept of time in each of the processors (affecting synchronization,) their output (affecting reliable performance,) their concept of the integrity of the stored data (affecting consistency of distributed databases) and their concept of the integrity of the whole system (affecting reliable reconfiguration.)

This report presents a general solution to this problem. A communication structure is defined in which consistency algorithms are performed in communication interface units that handle information transfers among the processors. With such a structure, highly reliable multiprocessor systems can be constructed.

B. Related Problems

In this report, the following related problem areas are addressed in order that the solution can be complete and practical.

1) There are many ways in which computer systems may fail. The major concern here is hardware faults causing system failures. Hardware faults may be broadly classified as either permanent or temporary faults. The latter type is more difficult to handle, due to the random nature of their occurrences and the short duration when they can be detected. The algorithms and designs implemented here consider only fault occurrences and not the nature of these faults. Hence they

are equally effective for both kinds of faults, under the same constraints as to the location where they may occur.

2) Traditionally, systems such as TMR (triple modular redundancy) and TMR with stand-by spares (Hybrid) are tightly synchronized systems. The consistent systems described in this report, along with many highly reliable systems are designed to be loosely synchronized. One advantage is the absence of lock-step operations, so that errors in different modules are less likely to be correlated to external interference. This also enables the modules of a system to be physically distributed, increasing the survival probability of the system when physical damage or interference occurs to a part of it. In general, greater flexibility such as variable fault-tolerance for different tasks via different number of replications is possible, because the modules are more independent during the longer intervals between synchronization.

3) In a multiprocessor system designed to be fault-tolerant, the most critical element is the communication structure among the computer modules. This is the element which physically supports massive redundancy, fault isolation and system recovery and reconfiguration. The design of a "consistent multiprocessor system" is centered around providing fault-tolerance to this critical element.

4) In designing for high fault tolerance, extreme care must be taken to ensure that the circuits that implement the fault-tolerant

functions are not susceptible to faults causing system failures. In this report, functions are implemented in large-scale integrated (LSI) circuits. In LSI, circuit constraints are minimal, allowing the use of a larger number of simpler circuits on a single chip for higher reliability and testability. In this context, parallelism is also exploited for efficiency and performance. A distributed control structure is used, resulting in minimal inter-dependency among the various control functions, allowing them to be effectively testable concurrent with normal operation.

C. Overview of the Report

This report describes the theory and design of constructing consistent fault-tolerant multiprocessor systems using specialized units that interface each processor to a set of interprocessor buses.

Although the algorithms employed can be extended to larger systems with more processors, only systems with four processors are described. Figure 1.1 shows the general structure of such a system.

Chapter II demonstrates the use of an LSI circuit to implement the basic consistency algorithm for multiprocessors. The theory of the algorithm is explained. The design of the circuit which employs very large-scale integrated circuit (VLSI) design techniques is fully described. Called a Consistency Unit (CU), it was actually designed

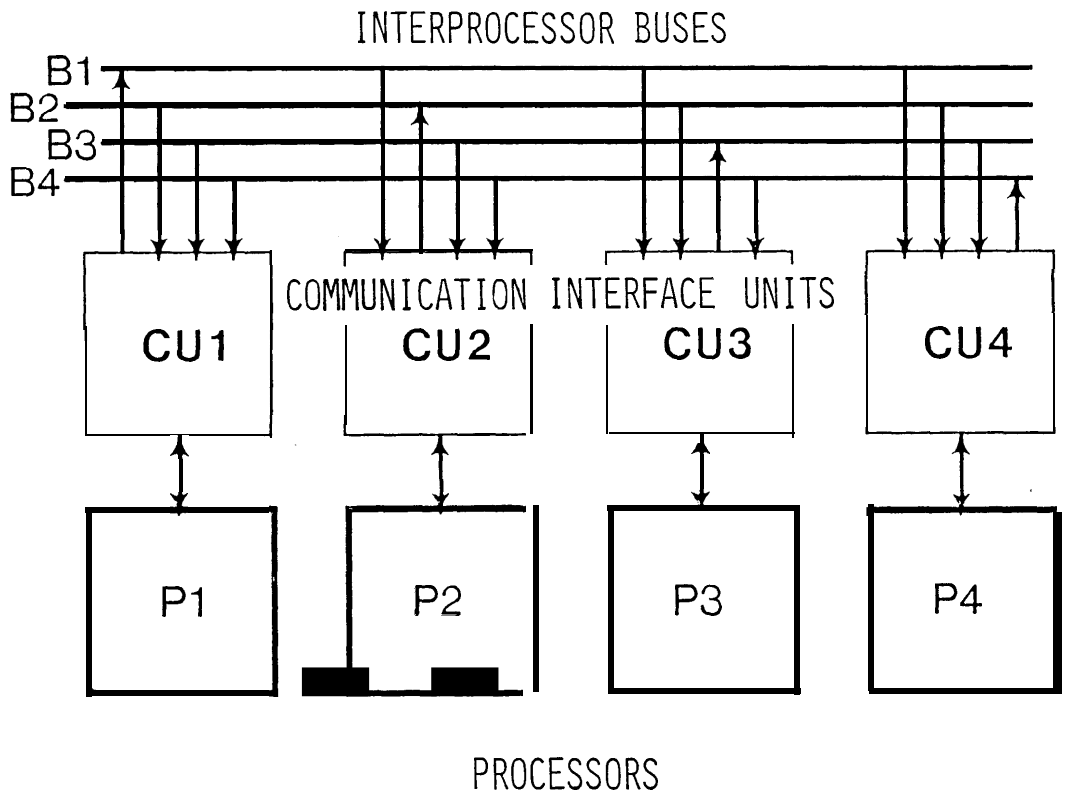


Fig. 1.1 Structure of a Fault-Tolerant Multiprocessor System with Communication Interface Units

and tested as an NMOS circuit. (The work reported in this chapter was presented at the FTCS-10 conference [Fu80].)

Chapter III describes an expanded design of the CU called the fault-tolerant Consistent Interprocessor Communication Interface (CI). The CI is regarded as central in the critical communication structure of highly fault-tolerant multiprocessors. The aspects of fault detection within the CI, double fault tolerance and concurrently testable hardware are addressed. Algorithms are described in detail and a block diagram design for LSI implementation is presented.

Chapter IV concludes the report with opportunities for further research and summarizes some implications of the design of the CU and the CI in the realm of fault-tolerant computing.

CHAPTER II. THE BASIC FRAMEWORK FOR CONSISTENCY --- THE CONSISTENCY UNIT (CU)

A. Achieving Agreement among Processors

1. The Consistency Problem

As computer systems become more modularized, the modules themselves interact at an increasingly higher level. The consistency of common data shared among the modules is then a critical issue.

In massive redundant fault-tolerant systems, processing units are replicated and are run in as independently a manner as possible so that any fault in the system can be properly detected, contained, and its effects eliminated from the output of the overall system.

However, complete independence is not possible in practice. The operations in all the units need to be synchronized, either tightly or loosely, and inputs to them need to be consistent to within a certain tolerance. Without these conditions being met, the disagreement among processors will eventually be large enough to constitute an error in the system.

Take, for example, the problem of clock synchronization across all the units of a system. Simple majority voting is insufficient. Consider three processors A, B and C, which would advance their own

independent clocks when a majority of them sets its own boolean flag to true to indicate that it is time to make the advance. The independent clocks will vary. If C fails in such a way that its boolean flag appears to A as true and to B as false at a time when clock A is ahead of clock B, then processor A and B will drift apart even though only C is failing. This is analogous to the digital clock synchronization problem in [Dal73].

2. The Consistency Algorithm

A solution to the problem of achieving consistency was developed by Pease, Shostak and Lamport [Pea80] from a general treatment of reliable synchronization and matching in redundant systems [Dav78; Dav79]. It involves an algorithm that guarantees data consistency provided that the number of processors (n) is strictly greater than three times the number (m) of faulty processors in the system. That is, for $n > 3m$, by $m+1$ rounds of information exchange among the n processors, and subsequent computations, the consistency conditions discussed below will hold.

Let P1, P2, . . . , Pn be the processors, and let i, j, and k be integers between 1 and n. At some given times, each Pi holds an internal datum that it wishes to communicate to all the other processors through the consistency process. Pi's datum is called its private value, denoted as PV[Pi]. After the exchanges and

computations, details of which are described later, Pj will have derived a vector of n values, the interactive consistency vector (of Pj), denoted as ICVj, with each element ICVj[Pk] corresponding to the private value PV[Pk] of processor Pk.

The consistency conditions are as follows:

CONDITION 1. All nonfaulty processors will compute exactly the same ICV. That is, ICVi = ICVj for all i,j such that Pi and Pj are nonfaulty.

CONDITION 2. A nonfaulty processor Pi will be able to derive the true private value PV[Pj] held by a nonfaulty processor Pj. That is, ICVi[Pj] = PV[Pj] .

In this chapter, only the single fault case (m=1) for four processors (4 = n > 3m) will be discussed. The general procedure for multiple faults (m>1) is a recursive one.

The system thus consists of four isolated processors, of which no more than one is faulty. The processors can communicate by messages, with the sender of the message always identifiable by the receiver.

Two rounds of information exchange are required. In the first round, the processors exchange their private values. In the second round, they exchange the results obtained in the first round. The faulty processor, if any, may "lie", or refuse to send messages. In

the latter case, the receiver chooses a value at random to substitute for the missing value.

Having completed the exchanges, each nonfaulty processor Pi records its private value PV[Pi] for the element of the vector corresponding to Pi itself, ICVi[Pi]. The element corresponding to every other processor Pk is obtained by choosing the majority of the three received reports of Pk's private value (one of the three was received directly from Pk in the first round and the other two were received from the remaining two processors in the second round.) If no majority exists, a default value such as 'NIL' is used.

Figure 2.1 details an example. Each column represents one processor. Each row represents the information contained in each processor at each step. Each processor Pi holds a 4x4 matrix Mi and a 1x4 ICV vector of values. Each matrix element Mi[j,k] represents the value Pi received from Pj that corresponds to PV[Pk]. Elements irrelevant for the algorithm is represented by "-".

The only faulty processor is P4. Initially (row 1), each Pi has its PV[Pi]. After the first round of data exchanges (row 2), each Pi has obtained values directly from the remaining three processors. After the second round (row 3), each Pi has additionally the pertinent values obtained indirectly from the other processors. For example, P1 has obtained from P2 the values of P3 and P4 that P2 has obtained in the first round. Finally (row 4), the ICV's are computed using

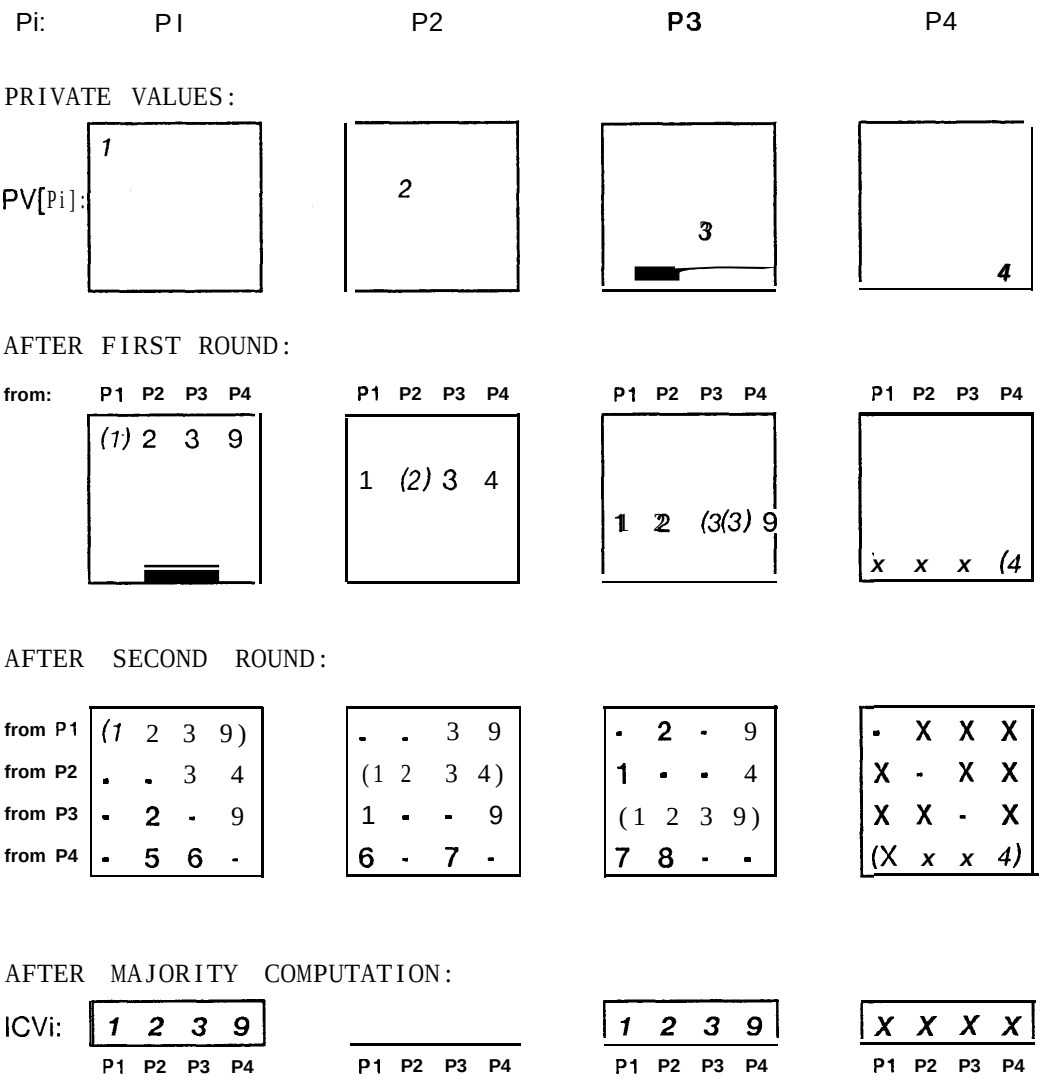


Fig. 2.1 Illustrative Example of the Consistency Algorithm

majority voting. If no majority exists, and such would be the case if P4 has sent three different values to P1, P2 and P3 in the first round, 'NIL' would be used. Comparing ICV1, ICV2 and ICV3 with each other and with PV[P1], PV[P2], PV[P3] and PV[P4], it is clear that conditions 1 and 2 are satisfied.

To see that this procedure does assure consistent vectors to be computed in each nonfaulty processor, consider the computation in nonfaulty processor Pi for ICVi[Pk], the element in ICV corresponding to processor Pk:

1) If Pk is nonfaulty, Pi will receive PV[Pk] both from Pk and from the other nonfaulty processor(s). Pi will therefore determine that there is a majority of values and its ICVi[Pk] should be PV[Pk];

2) otherwise, Pk is the only faulty processor. Two cases follows:

2.1) if all nonfaulty processors find no majority in their received values for Pk, then they will report NIL for their ICV[Pk] and thus they are consistent;

2.2) otherwise, for nonfaulty Pi to conclude that ICVi[Pk] is the value V (non-NIL), it must have received V from at least two other processors. Again, there are two cases:

2.2.1) if these two *are* nonfaulty, then they must receive V from every processor other than Pi (and possibly also from Pi.) Thus all nonfaulty processors compute V to be their ICV[Pk];

2.2.2) otherwise, Pi receives a value VV not equal to V from nonfaulty processor Pj. Then Pj receives V from all processors except Pk. Hence Pj determines that its ICV[Pk] is V. All other nonfaulty processors received V from all processors except Pj, hence they will compute V to be their ICV[Pk].

B. Hardware Implementation

1. Advantages of Using Hardware

Advances in integrated circuit technology have enabled more and more circuitry to be placed in a single IC package. These developments can be put to use advantageously in the design of fault-tolerant systems. The concept of fault-tolerant building blocks [Ren78] is an example of the use of specialized LSI circuits. However the current trend is towards software implementation of fault tolerance as in Pluribus [Kas78], Tandem [Kaz78; Bar78], and SIFT [Wen78].

While there are advantages in performing fault-tolerant functions in software, such as more flexibility and lower cost, many such functions do lend themselves well to hardware implementation with respect to efficiency and testability. Consider the following points:

* Algorithms requiring some computational complexity on a single processor may be simply performed in hardware as concurrent or parallel operations. In fact, the era of VLSI ushers in a new realm of cheap computation where performance is limited mostly by circuit communications[Fos80].

* While it is possible to verify programs performing a given fault-tolerant function, it is considerably more difficult to prove that they will run correctly on a processor that also performs other system functions.

* Since hardware is the basic structure supporting fault tolerance, fault characterization in dedicated circuits at this level is more practical and reliable.

* Hardware provides a real physical level of system hierarchy above which the fault-tolerant function is transparent to levels above. Higher modularity aids fault diagnosis and fault location.

* While software algorithms might integrate well with application programs, hardware algorithms might also integrate well with the system structure.

The design of the Consistency Unit described in the following sections is an attempt to illustrate some of these points.

2. Design of the Circuit

This section shows the design of the Consistency Unit (CU) which acts as a bus interface unit serving one processor in a four-processor system using massive redundancy for fault tolerance. The CU implements the consistency algorithm described in section II.A.2 above.

The processors communicate by means of four dedicated interprocessor buses, each of which is assigned to be output from only one processor through one CU. Each processor together with its associated CU and bus constitute a module. The system is then single-fault-tolerant with respect to any fault in any one module.

The concern here is data consistency, in the sense of the two conditions described in section II.A.2. The detailed timing for transactions on the bus is assumed to be synchronized with a fault-tolerant clocking scheme such as the one proposed in [Dav78].

Figure 2.2 shows the detailed connections for such a system. Note that four CU's are used. These CU's are identical but are connected to the buses in the symmetrical manner shown.

Thus the CU consists of four ports that connect to the interprocessor buses. They are 01, the output port, 12, I3 and 14 the input ports. There is also the PIO bidirectional port that connects

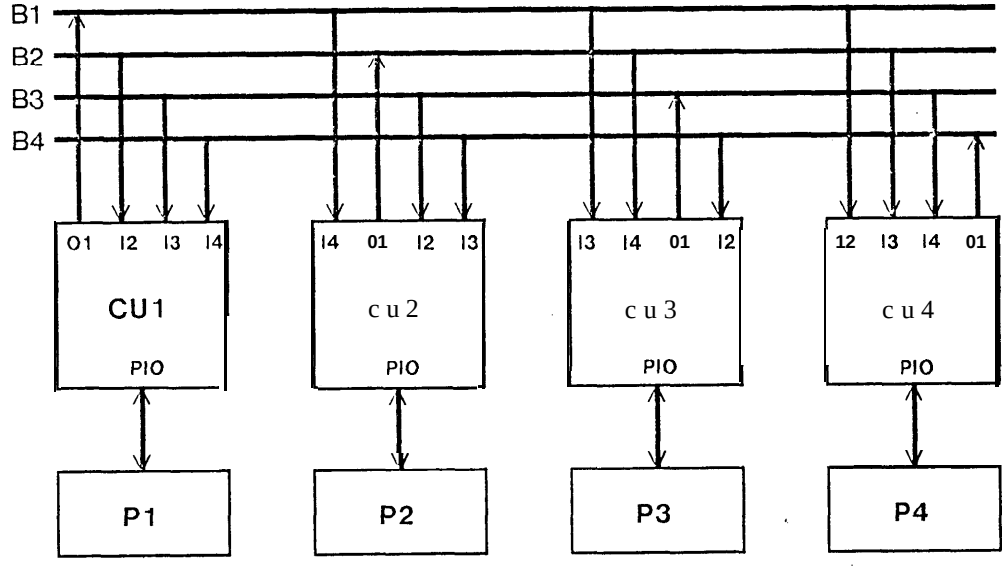


Fig. 2.2 Detailed Bus Connections of Four CU's in a Four-Processor System

to the processor. All these ports and the buses are of the same bit-width. There are 12 internal CU registers; each again has the same bit-width as the ports. Table 2.1 lists all the ports and registers and describes the function of each. The naming convention used is tailored to describe the operation of CU1 conveniently.

Table 2.2 shows that the CU goes through eight clock cycles, PHO through PH7, for each computation of the interactive consistency vector with elements ICV[P1], ICV[P2], ICV[P3] and ICV[P4]. In the first cycle, PHO, the first round of exchanges takes place. Each processor Pk outputs its private value PVk and CUk then broadcasts this value to all the other CU's through the bus Bk. At the same time, all the other CU's broadcast their values obtained from their respective processors over their respective interprocessor buses. Thus, from the input ports 12, I3 and 14, CUk is able to obtain the communicated values of all the other processors directly, that is without going through other modules. CUk then stores these values in R2, R3 and R4 respectively. This constitutes round one of the algorithm.

In round two, there are six values to be communicated in an orderly manner. The six values are values of each of the other three processors received indirectly through the two remaining modules. In this implementation, these values are transferred in three cycles, namely, PHI, PH2 and PH3. To describe these transfers, consider, for example, the transfer that takes place during PH1 on B2 for CU1.

Table 1. CU Ports and Registers

NAME	FUNCTION
PI0	Bidirectional port connecting to processor Pk; CUk receives PVk and sends out the resultant vector, ICVk, through this port.
01	Output port to bus, Bk; broadcast values.
I2,I3,I4	Input ports connecting to the other buses.
R2,R3,R4	Registers to hold results of first round from 12, I3 and 14 respectively; (for CU1, they will correspond to PV2, PV3 and PV4.)
R21,R23,R24 R31,R32,R34 R41,R42,R43	Registers to hold results of second round: Rij is the value PVj obtained from CUj via CUi where i=2,3,4; j=1,2,3,4; and'i not equal j.

Table 2. CU Cycle Functions

CYCLE	FUNCTION
PH0	O1<-PIO; Pk outputs its private value PVk and CUk outputs this from PIO onto Bk for the first round of exchanges.
	R2<-I2; R3<-I3; R4<-I4; } receiving from other CU's for first round.
PHI	O1<-R2; Contents of R2 broadcast onto Bk.
	R23<-I2; R34<-I3; R41<-I4; } first part of second round
PH2	O1<-R3; Contents of R3 broadcast onto Bk.
	R24<-I2; R31<-I3; R42<-I4; } second part of second round
PH2	O1<-R4; Contents of R4 broadcast onto Bk.
	R21<-I2; R32<-I3; R43<-I4; } last part of second round
PH4	PIO <- maj(R2,R32,R42) (*) ; for CU1 this is ICV1[P2].
PH5	PIO <- maj(R23,R3,R43) (*) ; for CU1 this is ICV1[P3].
PH6	PIO <- maj(R24,R34,R4) (*) ; for CU1 this is ICV1[P4].
PH7	PIO <- maj(R21,R31,R41) (*); for CU1 this is ICV1[P1], it is redundant, but can serve as a check for CU1.

(*) where maj(Ri,Rj,Rk) is the majority value of the contents of Ri, Rj and Rk if such exists; otherwise it is 'NIL'.

B2 is always driven by CU2, and, at this moment, CU2 is outputing the contents of its R2. This is the communicated value of PV[P3] from CU3 during PH0, the first round. (This is evident from the way the CU's are connected as shown in Figure 2.2.) Now, for CU1, this value is received on 12 to be stored in CU1's R23. Thus R23 will contain the communicated value of P3 indirected through CU2. (R23 is thus named.)

Besides the six necessarily communicated values, three additional values can be communicated through the interprocessor buses within the three cycles of the second round. For CUk, these are its own PV[Pk] received indirectly through the other three modules. They serve as further checks for the whole system.

During the remaining cycles PH4, PH5, PH6 and PH7, the word-by-word majority voter determines if a majority of identical values exists for the appropriate sets of three registers. For example, in PH4, the values in R2, R32, and R42 are compared. (See Table 2.2.) In CUI, these three values correspond to the value of P2 obtained in the first round, the same value obtained indirectly through CU3 in the second round and the same through CU4 respectively. Hence, the second component of the ICV vector, ICV1[P2] for CU1, can be determined. This is then output on PIO to be received by processor PI.

Figure 2.3 shows the logical block diagram of a CU. The sequence of the above operations is indicated by the labelled cycles (PH0 through PH7) at which the registers are being loaded from the

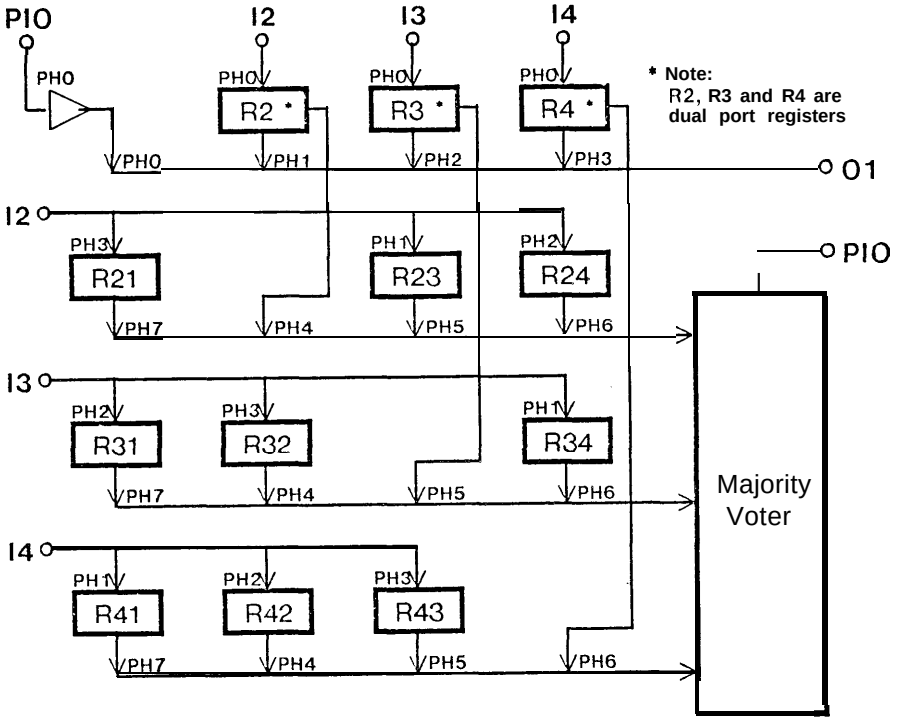


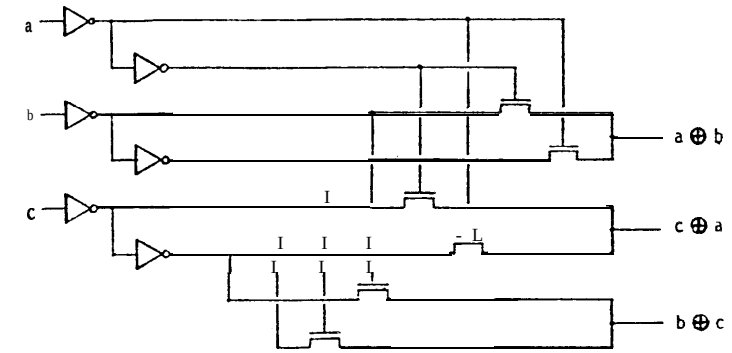
Fig. 2.3 CU Logical Block Diagram

buses or their contents gated onto the buses. The sequencing of the cycles is done by a self-starting ring counter with an enable control.

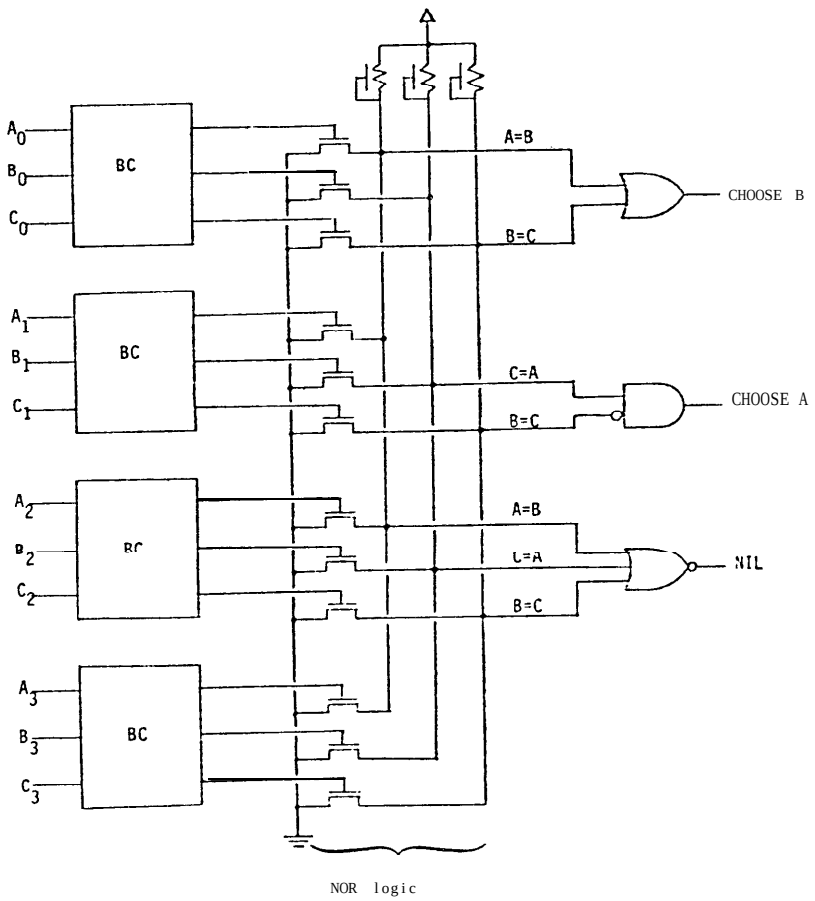
Both the majority voter circuit and the ring counter can be implemented quite compactly using the n-channel MOS (NMOS) technology of current LSI circuits. Figure 2.4 shows the former.

An experimental IC chip design was done implementing a CU with a word width of four bits. The block diagram of this chip is shown in Figure 2.5. The following modifications of the basic design are made. First, the PIO and 01 ports are merged, since 01 is basically used in the first four cycles and PIO in the last four cycles. Secondly, R21, R31 and R41 are eliminated, as they serve only secondary purposes. Lastly, and most importantly for testing purposes, all the ports are made bidirectional, thus making all the internal registers and the majority voter circuit accessible from the pins; this is a necessary consequence of the design of the data flow in the circuit.

A photomicrograph of the chip is shown in Figure 2.6 and details of the layout is shown in Figure 2.7. Using a 5 micron technology, it measures about 100 mil square. It has been fabricated and tested. There were some layout problems at the registers and hence these do not function in the test chip. Otherwise, the control circuit and the majority voter functions as designed.



(a) Bit Comparator (RC)



(b) Word Comparator (WC)

Fig. 2.4 Majority Voter Details

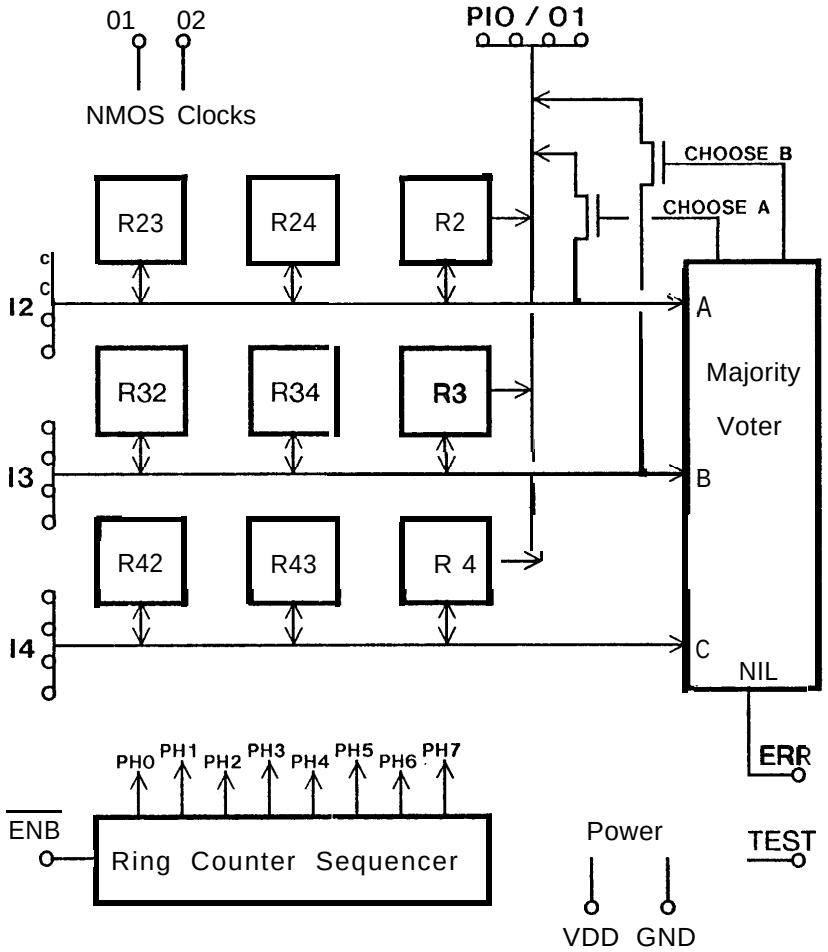


Fig. 2.5 CU Chip Block Diagram

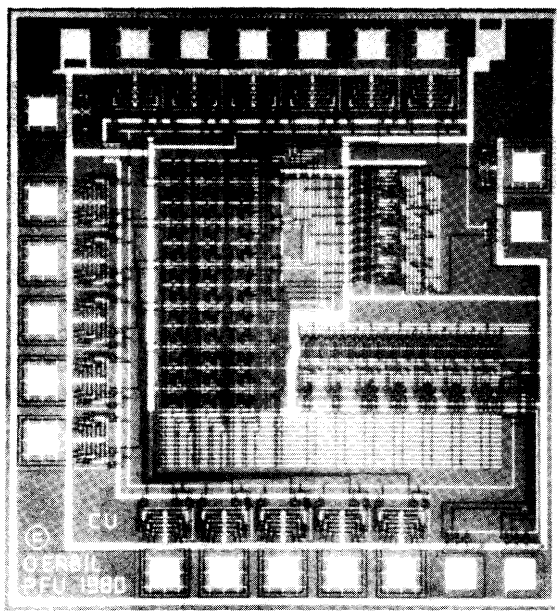


Fig. 2.6 Photomicrograph of CU Chip

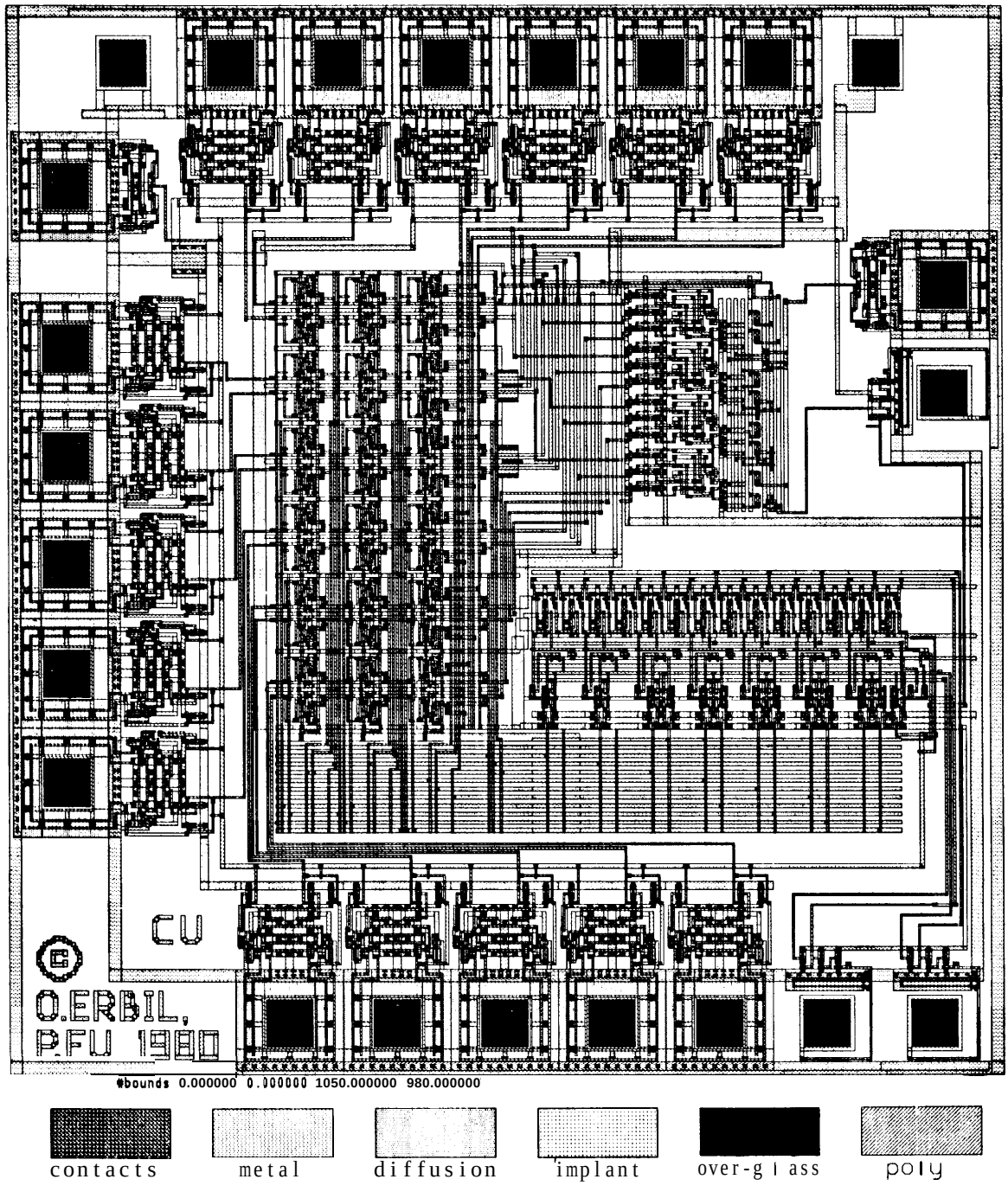


Fig. 2.7 CU Chip Layout

3. Enhancements of the CU

The basic design can be enhanced in two ways. Two CU's can be connected in parallel to increase the performance of the CU function by a factor of two when many private values are to be communicated. An extra register, R1, is needed to store PV[Pk] from Pk at PH7 to be used in PHO of the next group of 8 cycles. Given this, the CU would perform data exchanges on the interprocessor buses only in PHO through PH3 and compute the ICV and communicate with the processor only in PH4 through PH7. Thus, two CU's can interleave their uses of the buses and the bandwidth of the buses can then be fully utilized.

Another enhancement involves bringing out the majority voter -internal signals, A=B, C=A and B=C, to external pins in open-drain condition for NOR-tieing with additional CU chips. (See Figure 2.4(b).) This allows expansion to wider data bit-width than is possible on a single chip due to pin number limitations.

4. Applications of the CU

The major function of the CU is in its ability to arrive at identical data values despite a single faulty module. For replicated input sensors on distinct modules, this provides perfect input synchronization because each nonfaulty processor can compute exactly the same mean value given an identical ICV vector. For timing synchronization, the same advantage applies: given that the processors communicate their concept of time, synchronization is provided on the system level. The operation of any reconfigurable multiprocessor depend on the crucial ability to reliably determine system configuration status and to effect system changes. The CU can also provide this needed function.

CHAPTER III. FULL CONSISTENCY IN COMMUNICATIONS --- THE CONSISTENT INTERPROCESSOR COMMUNICATION INTERFACE (CI)

A. Purposes and Goals

The CU described in Chapter II provides a multiprocessor system with a guarantee of consistency in the data communicated among the processors. With such a communication link available, nonfaulty processors can reliably arrive at agreement with all other nonfaulty processors. However, the constrain in such a system is, of course, that there can only be at most one module (the processor, the CU and the associated bus) that is faulty. During the occurrence of such a fault, the CU gives no good indications as to which module is faulty. And, if a fault remains to be undetected, the CU would no longer guarantee consistency in the event of a second fault. These shortcomings are dealt with in the design of a fully consistent communication scheme made possible by the Consistent Interprocessor Communication Interface (CI) described in this chapter.

Also, section III.B.3 in this chapter describes a double fault-tolerant scheme which can be implemented using the communication structure of the CI.

The Communication Interface (CI) can be considered as an extension of the CU, the functions of the latter being a subset of

those of the CI. The basic assumptions and organization of the overall fault-tolerant multiprocessor system are the same as that of the system using the CU (see sections II.A.2 and II.B.2.) Again, for fault location purposes, the processor, its associated CI and the bus dedicated to it are considered to be one processor module.

The CI has three modes of operation. In the first and elementary mode, it acts simply as a buffer: a message from the local processor is broadcast and the received messages from the other processors are sequentially sent back to the local processor. This mode is provided for system initialization.

In the second mode, the messages received from the other processors are re-broadcast. Simultaneously, other re-broadcast messages from the other processors are received. This is called two rounds of message exchange. In this mode, the CI is functionally similar to the CU.

In the third mode, there is a third round of message exchange, the purpose of which is to reliably derive the integrity of each individual connection in the communication structure.

B. The Algorithms

1. Definitions and Identification of Messages

In order to describe the algorithms involved, the following definitions are convenient. Let P1, P2, P3 and P4 be the computers, and let i, j, and k be integers between 1 and 4. At some given times, each Pi holds an internal data item that it wishes to communicate reliably to all the other processors through the communication process. Pi's data item is called its private value, denoted as PV[Pi].

After the rounds of exchanges and computations, details of which are described later, Pj will have derived four vectors of four values. The first of which is the interactive consistency vector (of Pj), denoted as ICVj, with each element ICVj[Pk] corresponding to the private value PV[Pk] of processor Pk. This ICV is identical to that of the CU. The other three vectors are the consistent connectivity vectors (of Pj), denoted as CCVj,m (where m ranges from 1 to 3), with each element CCVj,m[Pk] corresponding to the private value PV[Pk] of processor Pk as seen by its m-th (circular left-most) neighbor processor. For example, CCV1,3 is the CCV of P1 corresponding to the private values of all processors as seen by P2.

For all four vectors, the consistency conditions given in Chapter II will hold for any single module fault condition. However, the three rounds of message exchanges also support a consistency scheme yielding a fifth vector of four values similar to that of the ICV, and satisfying the consistency conditions with only a high probability, given the possibility of two module faults. The procedure for obtaining this particular vector is given in section III.B.3 below.

The three rounds of exchanges take place as follows. In the first round, the processors exchange their private values. In the second round, they exchange the results obtained in the first round. In the third round, they finally exchange the results obtained in the second round. The faulty processor, if any, may send an erroneous message, or different messages to different processors, or does not send a message at all. In the last case, the receiver processor simply chooses a value at random to substitute for the missing value.

The procedure is symmetric among the four processors assuming a similar symmetric connection of the CU's in Figure 2.2. Thus it is sufficient to describe the operation of the whole system by examining that of CI1, the CI attached to P1. Each message is identified uniquely by its origin and its path through the processors. Thus a string of one to three digits represent a message, with the rightmost digit being its origin, the next digit to the left being its stop during the first round and the leftmost digit its stop during the

second round and an always omitted fourth digit being its final destination (since we are referencing P1, this digit is always 1.) For the sake of uniformity, all message names are made to be three digits by padding 1's on their left if there are less than three. For example, 423 is the message that originated at P3,was at P2 after the first round, at P4 after the second round and is now at Pl after the third and last round. 112 is the message that originated from P2 and after the first round and thereafter is at Pl. 111 is the message originating from Pl.

2. Details of Message Exchanges and Computations

Table 3.1 details the message exchanges during the three rounds. In each round, the messages broadcast on bus B1 is shown on the first line and those received from buses B2, B3 and B4 are shown in the same column in the following three lines. All four CI's operate identically in synchrony and that is the basis for deriving which messages are received from the other modules.

Table 3.1. Details of Three Rounds of exchanges at CI1

ROUND 1

Output to B1:	111
Received from B2:	112
Received from B3:	113
Received from B4:	114

ROUND 2

Output to Bl:	112	113	114
Received from B2:	123	124	121
Received from B3:	134	131	132
Received from B4:	141	142	143

ROUND 3

Output to B1:	123	234	141	124	131	142	121	132	143
Received from B2:	234	241	212	231	242	213	232	243	214
Received from B3:	341	312	323	342	313	324	343	314	321
Received from B4:	412	423	434	413	424	431	414	421	432

The procedure for arriving at the ICV and CCV's from the exchanged values can best be understood if the three rounds are considered as two overlapped levels of two-round exchanges: The first two rounds are then identical to the two rounds in the CU. The last two of the three rounds are simply three two-round exchanges operating on the three received messages received in the very first round.

First examine the results of round 2, which could be rearranged as shown in Table 3.2. Here, messages originating from the same computer Pi are placed on the same row with the appropriate label. The consistency algorithm calls for a majority function to be applied on precisely each of the three rows corresponding to P2, P3 and P4. The majority function is defined as the majority of identical messages, or if there is no majority, a 'NIL' value to stand for a null message. The results are named C2, C3 and C4. Thus ICV1 is simply:

$$ICV1 = (C1\ c2\ c3\ C4)$$

where C1 is the message 111.

Table 3.2. Computations After Round 2

PV's	MESSAGES				COMPUTATIONS
PV[P1]:	141	131	121		
PV[P2]:	112	142	132	->	c2 = maj(ll2, 142, 132)
PV[P3]:	123	113	143	->	c3 = maj(123, 113, 143)
PV[P4]:	134	124	114	->	c4 = maj(134, 124, 114)

Similarly, the results of round 3 can be processed by rearrangement and applying the same majority function, as shown in Table 3.3. The difference here is that three vectors of values are obtained:

$$CCV1,1 = (C41\ C12\ C23\ C34)$$
$$CCV1,2 = (C31\ C42\ C13\ C24)$$
$$CCV1,3 = (C21\ C32\ C43\ C14)$$

In reality, C2, C3 and C4 computed earlier can be used as C12, C13 and C14 respectively, under the assumptions stated for this system. Hence, C12, C13 and C14 need not be computed.

These connectivity vectors provide a consistent view of the messages considered received by each CI from one another in the first round.

Table 3.3. Computations After Round 3

PV's	MESSAGES				COMPUTATIONS
PV[P1]:	341	241	141	->	c41 = maj(341, 241, 141)
PV[P2]:	412	312	212	->	C12 = maj(412, 312, 212)*
PV[P3]:	123	423	323	->	C23 = maj(123, 423, 323)
PV[P4]:	234	134	434	->	C34 = maj(234, 134, 434)
PV[P1]:	231	131	431	->	C31 = maj(231, 131, 431)
PV[P2]:	342	242	142	->	C42 = maj(342, 242, 142)
PV[P3]:	413	313	213	->	C13 = maj(413, 313, 213)*
PV[P4]:	124	424	324	->	C24 = maj(124, 424, 324)
PV[P1]:	121	421	321	->	c21 = maj(l21, 421, 321)
PV[P2]:	232	132	432	->	C32 = maj(232, 132, 432)
PV[P3]:	343	243	143	->	C43 = maj(343, 243, 143)
PV[P4]:	414	314	214	->	C14 = maj(414, 314, 214)*

* --- see text

The results for all three rounds of exchanges and computations can be nicely summarized in the following Consistent Communication Matrix (CCM):

$$CCM = \begin{vmatrix} C1 & C12 & C13 & C14 \\ c21 & c2 & C23 & C24 \\ C31 & C32 & C3 & C34 \\ C41 & C42 & C43 & C4 \end{vmatrix}$$

where C12, C13 and C14 are simply 112, 113 and 114 respectively. They are the messages received during the first round. Each row of the CCM represent the four messages received at each of the four CI's. Each column represent the same message as received by each of the four CI's.

3. Procedure for Double Fault Tolerance

The structure for the three rounds of exchanges also supports a double module fault tolerant scheme that guarantees consistency with a high probability. In the original paper on the consistency algorithms [Pea80], a scheme was discussed where cryptography was used to attach an authenticator to each message issued from any computer module so that any alteration of the messages enroute can be detected. In addition, the origin and path of the message can be verified with arbitrarily high probability. Given the above, consistency can be arrived theoretically with m+l rounds of messages, where m is the number of faulty modules.

A simplified version of this scheme is possible for the CI. Here, the authenticators are simply replaced by a good error-detecting code. This often is not an extra overhead since coding may already be employed in the communication hardware. Verification of the origin and path is partially accomplished by appending a code of the origin into the message and using the fact that a message received from a particular dedicated bus must come only from the associated computer.

The procedure for arriving at a double fault tolerant interactive consistency vector, denoted as <2F>ICV, follows. In Table 3.4, the components of <2F>ICV1 for CI1 are shown. Each <2F>ICV1[Pk] is derived from a set of 5 different messages. Majority function is not applied on this set. Instead, those messages are found to be in

error upon decoding are set to NIL values and the rest are compared for equality. If these are all identical, the message value is taken as the appropriate element in the <2F>ICV. If they are not all identical, or if all messages are in error, the NIL value is used.

Table 3.4. Components for Double Fault Tolerant <2F>ICV

Pk	MESSAGES						<2F>ICV1[Pk]
PI:	111					->	<2F>C1
P2:	112	142	132	342	432	->	<2F>C2
P3:	113	123	143	423	243	->	<2F>C3
P4:	114	134	124	234	324	->	<2F>C4

To see that this procedure does assure consistent vectors to be computed with high probability in each nonfaulty processor, consider the computation in nonfaulty processor Pi for <2F>ICVi[Pk], (the element in <2F>ICV corresponding to process Pk):

1) If Pk is nonfaulty, Pi will receive PV[Pk] from Pk. With high probability, all other values received will either be the same or NIL, by the fact that corrupted messages are detected and set to NIL.

2) otherwise, Pk is faulty, and at most one other processor is also faulty. Suppose Pi received a message with a value V (non-NIL)

from Pk, the following shows that another nonfaulty processor Pj will also receive V from Pk.

2.1) If Pj is in the path of the message, then Pj must have already received V from Pk.

2.2) otherwise, Pj is not in the path. Again, there are two cases:

2.2.1) if the message is received in the third round, two CI's other than Pj is involved. One of them must be nonfaulty, and it would have passed the value V to Pj directly;

2.2.2) otherwise, the message is received in the first or second round. Hence, Pi will be passing this on to Pj directly in the next round.

The above shows that any non-NIL value received by a nonfaulty processor will be identical to the same value received by another nonfaulty processor. Since the NIL values are eliminated if non-NIL values are received, this fulfills consistency condition 1. Also, with point 1) above, consistency condition 2 is also satisfied.

Optionally, this scheme can be superimposed onto the CI as a degraded mode of operation once a single module fault is determined.

C. An Implementation of the CI

In this section, one implementation of a CI chip design is described. The VLSI design principles of exploiting parallelism, regularity of circuit structure and distributed control [Mea80] are utilized. Moreover, an effort was made to render the chip fully testable and, for the control part, concurrently testable as well.

Table 3.5 shows a translation of the algorithmic three rounds of information exchange and the subsequent majority computations into a practical procedure by allotting specific time slots from a sixteen time interval cycle for each operation. The time slots are named T0, T1, ... , to T15. The middle column lists messages to be broadcast onto bus B1 by CI1 at each time slot. The right column lists message values transmitted between computer P1 and CI1.

Mode one operation will comprise of T0, T1, T2 and T3. Mode two operation will comprise of T0 through T7. Mode three operation will comprise of the full sixteen time intervals, T0 through T15.

Table 3.5. Timing of CI Functions

Time Interval	To Port 01	From/To Port PIO
T0	111	111 (from P1)
T1	112	112 (to P1)
T2	113	113 (to P1)
T3	114	114 (to P1)
T4	123	c2 (to P1)
T5	134	C3 (to P1)
T6	141	c4 (to P1)
T7	124	C23 (to P1)
T8	131	c34 (to P1)
T9	142	c41 (to P1)
T10	121	C24 (to P1)
T11	132	C31 (to P1)
T12	143	C42 (to P1)
T13	---	c21 (to P1)
T14	---	C32 (to P1)
T15	---	c43 (to P1)

A block diagram showing the internal structure and timing of the CI is shown in Figure 3.1. Registers are shown with the names of the messages that they will hold. Internal buses are either direct connections to external buses or are buses that connect to the word-by-word majority voter. The voter circuit is identical to that used in the CU as described earlier in Chapter II, where bit-by-bit comparators are used to build word-by-word comparators. The outputs of the word-by-word comparators are then gated to choose the majority message. If there is no majority, a NIL signal is generated.

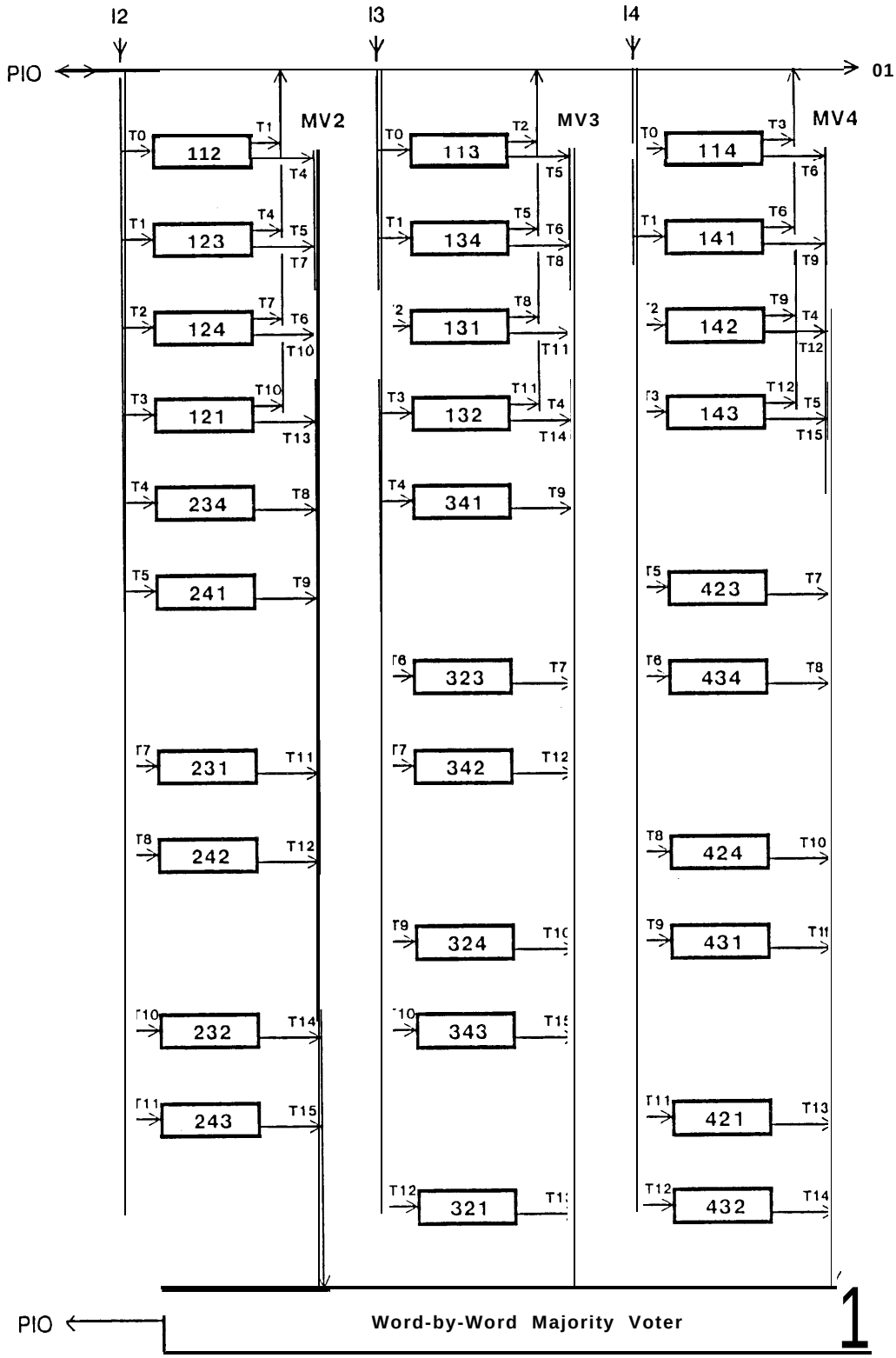


Fig. 3.1 CI Logical Block Diagram

The actual implementation follows very closely to the layout of the block diagram. Sequencing is accomplished by two synchronous ring counters of the type shown in Figure 3.2. NMOS logic is assumed. Once ENABLE'd, a single logic '1' will ripple through the shift registers until the end of the distributed NOR gate as defined by the mode selection signals MODE1 and MODE2. If MODE? is asserted, the rippling '1' will go back to TO after T3. If MODE2 is asserted, it will go back after T7 and for mode 3 operation, it will traverse the whole length. Two such ring counters will be used. By comparing matching signals from the two, the sequencing control can be checked concurrent with normal operation.

In Figure 3.3, the distributed control structure is shown imbedded among the registers. (The data paths are deleted for clarity.) One ring counter lies parallel to the input buses 12, 13 and I4. It controls the loading of the registers from these buses, as indicated by the arrows on the left of all the registers. The second ring counter interleaves the top four rows of registers. These registers have two independent output ports for outputing to the output bus 01 and to the majority voter.

There are three separate buses, MV2, MV3 and MV4, that connect to the majority voter. All registers except the top row are like associative store registers in that they are read by matching their key to keys on the buses. Each register has hard-wired four-bit keys that correspond to the right two digits of their names. Each register

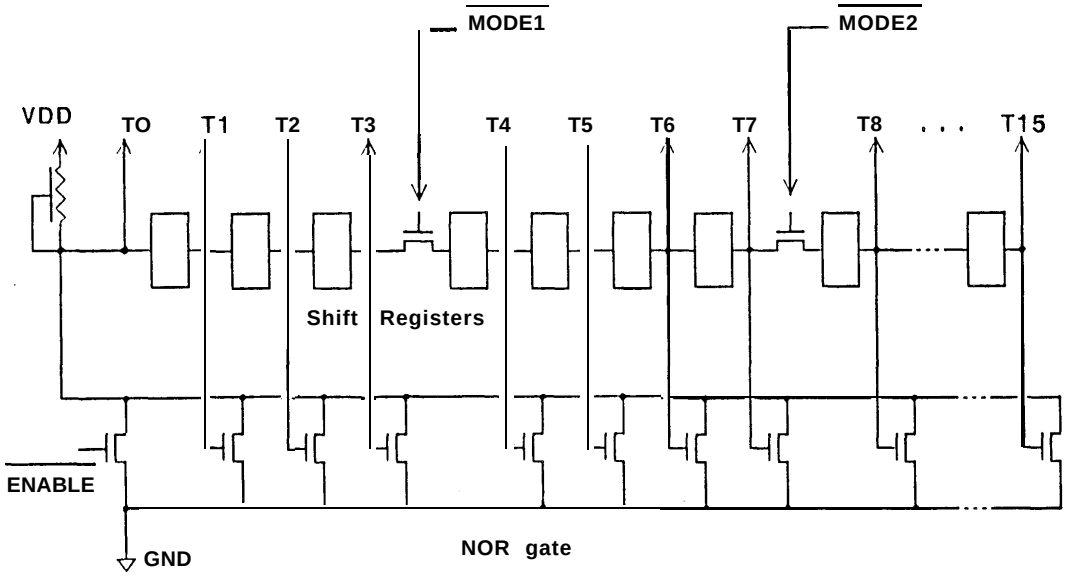


Fig. 3.2 Ring Counter for Sequencing

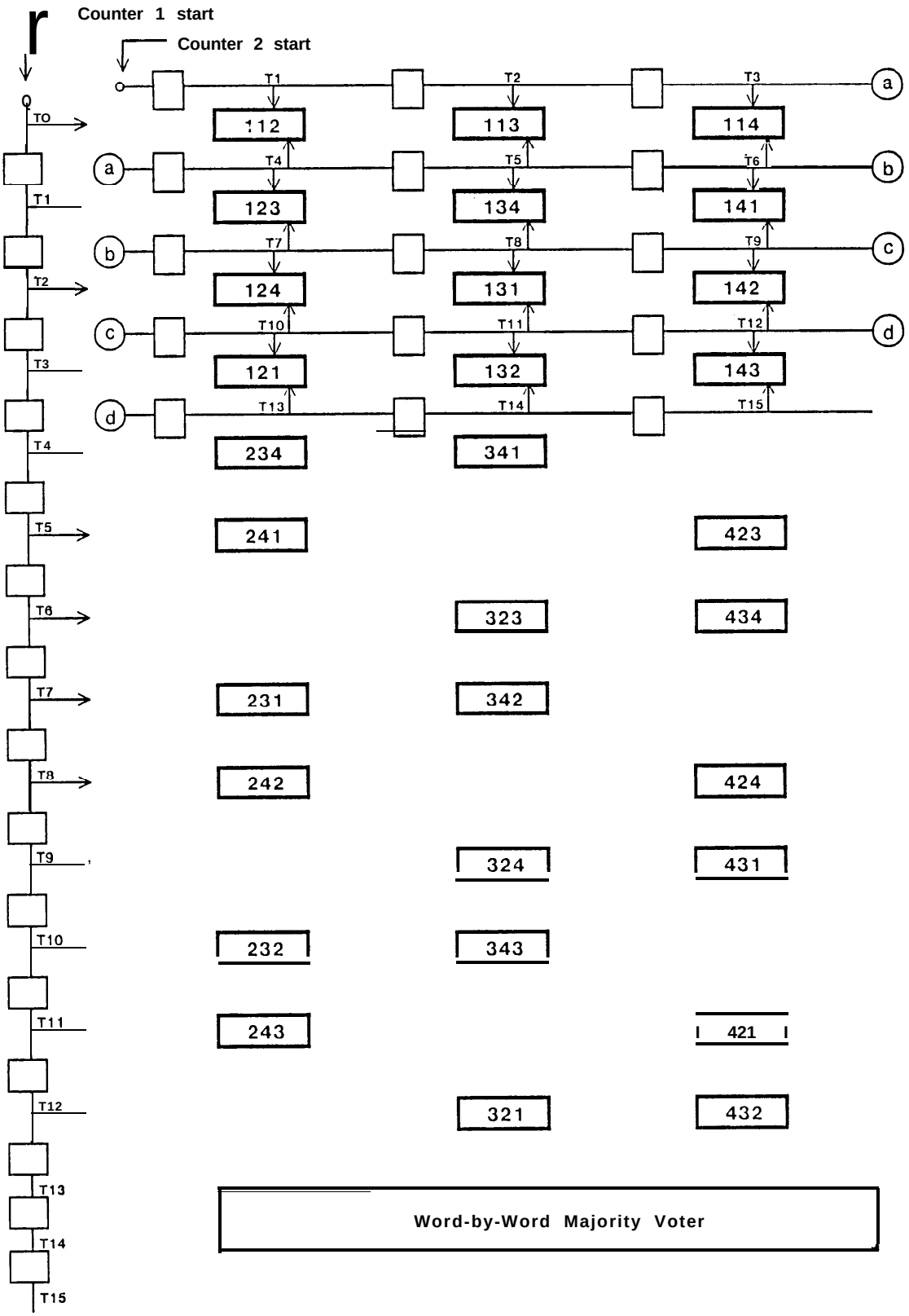


Fig. 3.3 CI Distributed Control Structure

primed by the second ring counter to output to a majority voter bus will provide the four-bit key on all three buses. For example, at T12, register 142 will be primed, and the key is then '42', causing registers 242 and 342 to output. However, the 11x registers (those on the first row), when they are priming other buses, uses the name of the bus as the second digit of the key. For example, at T5, 113 will be primed to output and thus the key '13' will generate keys '23', '33' and '43' on buses MV2, MV3 and MV4 respectively, causing registers 123 and 143 to output.

The necessary hardware to compute the double fault ICV is not illustrated. It can be implemented by adding an accumulator-like -registers that have access to the input buses. It is expected that decoders would screen out corrupted messages. Hence these accumulators need simply make equality checks as uncorrupted and relevant messages arrive one at a time. A fourth mode of operation is needed to transmit the double fault ICV to the computer P1.

D. Applications and Merits of the CI

While there are in existence numerous means and techniques to interconnect multiprocessors, especially in light of the trend towards distributed processing [Luc78; Jen76], most of them are designed to address the multiple goals of achieving resource sharing, higher performance in throughput, smaller response time and higher reliability. Reliability requirements center around the two aspects of high availability and integrity. But most systems can afford a trade-off between these two requirements or there may be a recourse to a larger system [Jon80 (pp. 143, 144)].

When computers are applied to critical tasks, they need to maintain high integrity and availability simultaneously. As mentioned in the introduction, the key element in a fault-tolerant system for such application is the communication structure among the multiple computers. The use of the CI generates one such structure to serve as the backbone for the redundant computers to achieve the system goal of ultra-high reliability.

Most applications for control of highly critical tasks do not have high volumes of data to process. Hence, the multiplicity of the rounds of data exchanges may be tolerated.

The most significant function of the CI is its implementation of consistency for the messages communicated among the computers.

Consistency is also implemented for the diagnostic findings on the integrity of the communication structure. With consistency, the individual computers can operate on the same SET of input values, thereby guaranteeing identical results given that the same computational procedure is used on the inputs for any redundant computations. This provides relief to the problem of output voting, which is often difficult when the results range over a range of values. A more valuable benefit is the possibility of concurrent detection of errors, thus reducing fault latency, which present serious problems to system recovery.

Although mode 3 operation is four times slower than mode 2 operation, the former is the preferred mode for two reasons. Firstly,. concurrent error detection in the communication structure is afforded by the consistent connectivity values supplied by the CI, without waiting until a low activity period when mode 3 is switched on to function as the diagnostic mode. Secondly, studies have shown that there is a correspondence between the amount of system activity and its reliability [But80; Cas80]. Hence, it is prudent to maintain an approximate stable load on the communication hardware and software.

The CI also tackles temporary (or transient) faults, noted as problematic for fault-tolerant systems in the introduction. The error detection capability, besides being concurrent, operates on the actual data being communicated. The effect is the detection of these

faults, which may not be truly determined through diagnostics, which runs either at other times or on different data.

The CI may fail itself. In this case, system integrity is not compromised since this is effectively the same as a processor module failure. It is possible to employ two CI's in parallel, interleaving their functions on the same set of buses or provide an alternate set of buses.

The CI can also be used as a passive unit as a listener on the inter-computer buses. One application is to drive output processors, taking advantage also of consistency. Another application may be a watchdog monitor for the buses, perhaps used for human interface to report the status of the computers for human intervention, if needed.

CHAPTER IV. CONCLUSIONS

A. Summary of the Report

Communication interfaces for use in a highly fault-tolerant multiprocessor environment are described. They implement consistency algorithms that enable different processor modules to compute on precise data even though there is the possibility of faults in a single module. A scheme which is tolerant to faults in two processor modules is also presented. All the processors are otherwise independent of one another, thus fault tolerance is enhanced in this way also.

In the CU, the basic algorithm is implemented in an LSI circuit. The very reliable interprocessor communication scheme is transparent to the processors involved, which only participate in broadcast and receiving messages as if the CU is simply a buffer. Full testability of the CU is integral in its design.

In the CI, the means to provide an ultra-reliable communication structure is based on concurrent checking using the actual data being communicated. The processors are provided with consistent information on the communication structure in order to perform consistent diagnosis and reconfiguration of the system.

For both the CU and the CI, the algorithms used can be extended to multiprocessor systems with *more* than four processors.

B. Further Research

This paper has not addressed itself to managing the ICV and CCV's received from the CI in an efficient manner. In a wider context, the message processing in the CI share some architectural similarities with data flow computers [Den80] in that messages are queued to be operated on by majority or equality operation units or to be re-sent to a bus. Thus, the CI structure may provide a compatible reliable communication system for data flow multiprocessors. On the other hand, a data flow architecture may free the CI's from operating in synchronous mode, which require some synchronization on the bus level among the CI's in order to operate.

C. Implications on Fault-Tolerant Computing

Looking beyond the problems of designing reliable VLSI circuits, the design of the CU and the CI demonstrates that there is much to be exploited in VLSI for the enhancing fault tolerance. This will have impact on the future of massive redundancy systems, which is still the most viable approach for achieving high SYSTEM reliability.

Although the design of the CI addresses itself mainly to providing a highly fault-tolerant communication structure for multiprocessor systems, it actually has wider implications. The use of concurrent checking minimizes error latency in error detection. The ability to consistently reconfigure permits very reliable repair. Overall, the highly reliable communication structure actually generates highly reliable multiprocessor systems.

ACKNOWLEDGMENTS

The author wishes to thank Professor E. J. McCluskey for his guidance throughout the course of this work, Professor R. Mathews, Professor J. Newkirk and 0. Erbil for the help in the VLSI circuit design and layout and the Integrated Circuits Laboratory for fabricating the CU chip. Helpful comments and assistance offered by R. Iyer, D. Lu, S. Bozorgui-Nesbat, A. Kuk, S. Chen and L. Christopher and generally the members of the Center for Reliable Computing are gratefully acknowledged. This work was supported in part by the Air Force Office of Scientific Research under Contract No. F49620-79-C-0069 and the National Science Foundation grant No. MCS-7904864.

REFERENCES

[Bar78] Bartlett, J. F., "A 'NonStop' Operating System," Hawaii International Conference of System Sciences, January 1978, vol. 3, pp. 103-117.

[But80] Butner, S. E. and R. K. Iyer, "A Statistical Study of Reliability and System Load at SLAC," Dig. of Papers, 10th Int. Sym. on Fault-Tolerant Computing, 1980, pp. 207-209.

[Cas80] Castillo, X. and D. P. Siewiorek, "A Performance-Reliability Model for Computing Systems," Dig. of Papers, 10th Int. Sym. on Fault-Tolerant Computing, 1980, pp. 187-192.

[Dal73] Daly, W. M., A. L. Hopkins, Jr. and J. F. McKenna, "A Fault-Tolerant Digital Clocking System," 3rd Int. Sym on Fault-Tolerant Computing, 1973, pp. 17-22.

[Dav78] Davies, D. and J. F. Wakerly, "Synchronization and Matching in Redundant Systems," IEEE Transactions on Computers, vol. C-27, pp. 531-539, June 1978.

[Dav79] Davies, D., "Reliable Synchronization and Matching in Redundant Systems," Computer Systems Laboratory Technical Report No. 169, Stanford University, January 1979.

[Den80] Dennis, J. B., "Data Flow Supercomputers," Computer, vol. 13, No. 11, November 1980, pp. 48-56.

[Fos80] Foster, M. J. and H. T. Kung, "The Design of Special-Purpose VLSI Chips," Computer, vol. 13, No. 1, January 1980, pp. 26-40.

[Fu80] FU, P. L., "Consistency Unit for Fault-Tolerant Multiprocessors," Dig. of Papers, 10th Int. Sym. on Fault-Tolerant Computing, 1980, pp. 363-368.

[Hop78] Hopkins, A. L., Jr., T. B. Smith, III and J. H. LaLa, "FTMP---A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," Proceedings of the IEEE, vol. 66, No. 10, October 1978, pp. 1221-1239.

[Jen76] Jensen, E. D., K. J. Thurber and G. M. Schneider, "A Review of Systematic Methods in Distributed Processor Interconnection," Conf. Rec., Int. Conf. on Communications, June 1976, pp. 717-722.

[Jon80] Jones, A. K. and P. Schwarz, "Experience Using Multiprocessor Systems---A Status Report," Computing Surveys, vol. 12, No. 2, June 1980, pp. 121-165.

[Kas78] Katsuki, D., E. S. Elsam, W. F. Mann, E. S. Roberts, J. G. Robinson, F. S. Skowronski and E. W. Wolf, "Pluribus -- An Operational Fault-Tolerant Multiprocessor," Proceedings of the IEEE, vol. 66, No. 10, October 1978, pp. 1146-1159.

[Kaz78] Katzman, J. A., "A Fault-Tolerant Computing System," Hawaii International Conference of System Sciences, January 1978, vol. 3, pp. 85402.

[Luc78] Luczak, E. C., "Global Bus Computer Communication Techniques," Proc., Computer Networking Symposium, 1978, pp. 58-71.

[Mea80] Mead, C. A. and L. A. Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, Mass., 1980.

[Pea80] Pease, M., R. Shostak and L. Lamport, "Reaching Agreement in the Presence of Faults," Journal of the ACM, vol. 27, No. 2, April 1980, pp.228-234.

[Ren78] Rennels, D. A., "Architectures for Fault-Tolerant Spacecraft Computers," Proceedings of the IEEE, vol. 66, No. 10, October 1978, pp. 1255-1268.

[Wen78] Wensley, J. H., L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," Proceedings of the IEEE, vol. 66, No. 10, October 1978, pp. 1240-1255.