THE DESIGN OF PERIODICALLY SELF RESTORING REDUNDANT SYSTEMS

by

Adit D. Singh

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

APPROVED:

F. G. Gray, Chairman

J. R. Armstrong

J. G. Tront

C. W. Bostian

V. Chachra

December 1982
Blacksburg, Virginia

ACKNOWLEDGEMENTS

The author wishes to express his sincere gratitude to Professors F. G. Gray, C. W. Bostian, J. R. Armstrong, J. G. Tront, and V. Chachra for serving on his doctoral committee. Special recognition is due the committee chairman, Dr. F. G. Gray for his expert guidance, encouragement and advice during all phases of this research effort.

This research was supported by the U. S. Army Research Office under grant DAAG29-82-K-0102. Additional financial support from the Department of Electrical Engineering at the Virginia Polytechnic Institute and State University is also gratefully acknowledged.

The author is deeply indebted to his parents for their constant encouragement, understanding and support during his years of graduate study.

TABLE OF CONTENTS

Chapter I

INTRODUCTION

1.1 THE NEED FOR FAULT TOLERANT SYSTEMS

Reliability in computer systems has been a problem since the days of the very first computers built out of relays and vacuum tube devices. While the reliability of electronic components has improved dramatically since that time, overall system reliability remains a problem because of the greatly increased complexity of today's computers. This problem is particularly serious in applications where system failure is unacceptable because it could cause catastrophic loss and/or endanger human life. Examples of operations that involve such applications are space missions, the automatic landing of aircraft, air traffic control, the control of nuclear reactors, and life support systems for medical applications. Since such failure-critical applications of computers will no doubt increase very substantially in the next few decades, it has become imperative to design and build computing systems to stringent reliability specifications. This dissertation is an attempt to address this problem.

The reliability of a system can be increased to some extent by building it from more reliable components, but this approach is often not cost effective and can usually yield only a limited improvement in reliability. However, for a computing system, "correct operation" only requires the correct execution of a set of programs and not necessarily the correct functioning of all components. Thus fault tolerance can be

introduced into systems to increase reliability. Such systems are deliberately designed with built in hardware redundancy to allow continued correct program execution even in the presence of component failures, as long as the number and types of failures are within the fault tolerance limits of the system. As a result, fault tolerant systems are capable of highly reliable operation.

1.2 PRESENT APPROACHES TO FAULT TOLERANT DESIGN

Several schemes have been proposed for employing hardware redundancy to realize fault tolerance in digital systems. These range from the classical component replication schemes first introduced by von Neumann [1], to software oriented approaches involving redundant execution of code with software checks and voting [2]. Redundancy schemes attempt to contain the effects of hardware malfunction within a subsystem of the overall system, thereby preventing errors from propagating from one subsystem to the next, and to the system outputs. The subsystem level at which redundancy is applied determines the level at which errors are contained. Techniques exist for applying redundancy at any desired level, ranging from individual components to complete systems.

Hardware redundancy techniques are classified as static, dynamic or hybrid. Triple modular redundancy (TMR) [3], shown schematically in Figure 1.1, is a static redundancy scheme that has been successfully applied in the space program [4]. A redundant subsystem employing

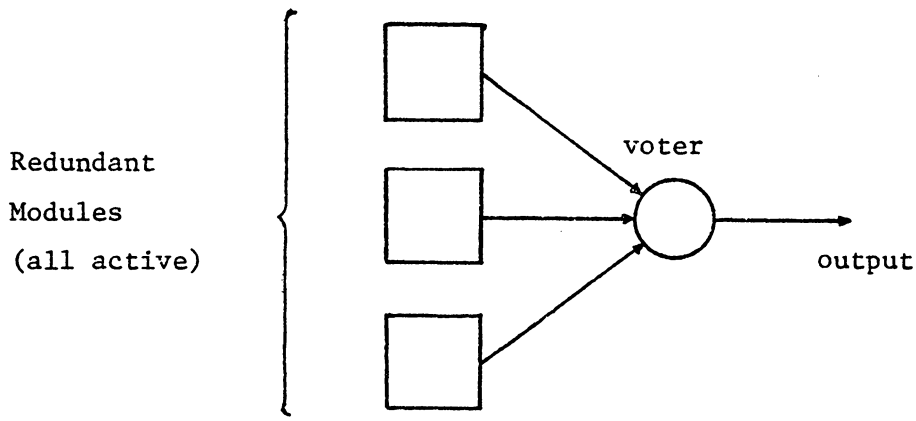

FIGURE 1.1: Triple Modular Redundancy (TMR).

the more general NMR (for N modular redundancy) [5] scheme consists of N identical modules (three for TMR) whose outputs are connected to a voter circuit. The voter masks failures in individual modules so that the subsystem stays operational as long as a majority of the modules continue to operate correctly. Thus, if the voter is free of failures, an NMR subsystem requires $\lfloor N/2 \rfloor + 1$ operational modules to be error free.

The redundant representation of information using error correcting codes is another important static redundancy approach that has been widely applied to improve reliability in digital systems [6]. Such codes can be designed to correct errors in one or more bits of a coded word. Error correcting codes are particularly effective in enhancing the reliability of memory units organized such that each bit in a word is stored in a different module or IC chip. They are less effective in protecting systems implemented on a single chip where bit failures in a word are often not independent.

In contrast to the passive error masking employed by static redundancy schemes, systems employing dynamic redundancy attempt to automatically detect faults when they occur, isolate them at the subsystem level and then replace the faulty modules with fault free spares. Figure 1.2 illustrates the dynamic redundancy standby sparing scheme [7]. A possible way to implement fault detection in such a scheme is to make the modules self checking [8]. A failure in an active module would then result in an invalid code word at the output. This can be detected and used to switch out the failed module and switch in a spare.

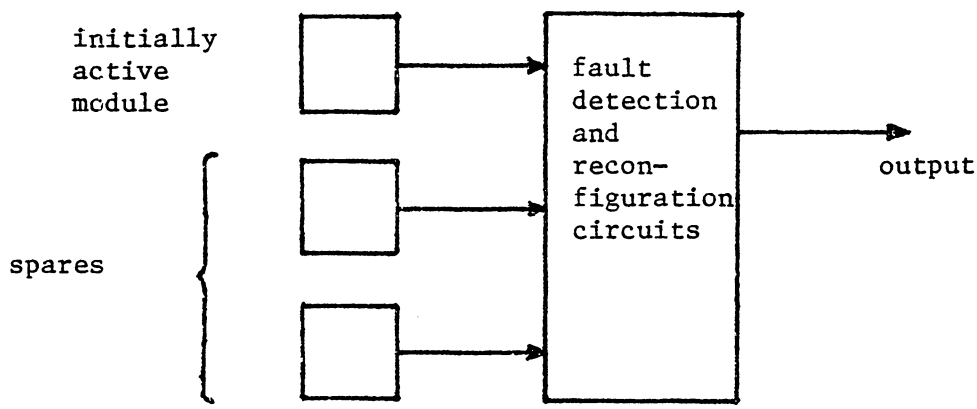

FIGURE 1.2: Standby Sparing Redundancy.

Ideally, a subsystem employing dynamic redundancy should continue to operate correctly, with possible interruptions during reconfigurations, as long as a single module stays failure free. This would make it superior to NMR which requires $\lfloor N/2 \rfloor + 1$ failure free modules to stay operational. However, in practice, the complex hardware required to implement fault detection and module switching (reconfiguration) in dynamic redundancy schemes is itself vulnerable to failures, and as a result such systems can fail before all the spares are exhausted. In the reliability modeling of dynamic redundancy in fault tolerant systems, this possibility has been accounted for through the concept of coverage, defined to be the conditional probability of system recovery following a failure [9]. It is well known [9] that system reliability can be very substantially affected by even a small deviation from unity coverage. While the coverage can be improved somewhat by designing the fault detection and reconfiguration circuitry to be itself fault tolerant, it is impossible to insure system recovery from every possible failure. Among the more significant factors that degrade coverage is fault latency. Recent studies [10] have concluded that often hardware faults do not immediately cause detectable computational errors. During this fault latency period, the arrival of a second fault can defeat the fault tolerance mechanism of systems designed to handle only single faults. Theoretically systems can be designed to tolerate two or more co-existing faults, but Stiffler [11] has shown that this can be self-defeating because the considerable additional hardware required for multiple fault

detection and isolation significantly increases the system complexity and the likelihood of additional failures.

A comparison between NMR systems and standby sparing systems with equal number of modules, first made by von Neumann [1], shows that NMR is almost unbeatable for short missions but that, even with relatively poor coverage, sparing is superior for longer missions. This is because it is unlikely that a majority of modules in a subsystem will fail over a short period; the failure of a subsystem is much more likely to be caused by failures in the voting or fault detection and reconfiguration circuitry. NMR is therefore more reliable for short missions because the relatively simple voting circuitry that it uses is less likely to fail than the complex detection and reconfiguration circuitry of standby sparing systems. On the other hand, over longer missions, subsystems are expected to fail due to a build up of failed modules. For such missions sparing scheme is superior because it can stay operational down to the last correctly operating module.

Other dynamic redundancy schemes that have been proposed include dual redundancy [12] with concurrent comparison for fault detection. On the detection of a fault both modules execute self test programs. The module that successfully completes the test continues executing the user program. In the pair/spare scheme, the spare is invoked upon detection of a disagreement between the pair.

A third class of redundancy schemes, hybrid redundancy, combines the advantages of the static and dynamic approaches by substituting

spares for failed modules in a static system. Such a scheme was first proposed by Mathur [13] in the form shown in Figure 1.3. The switch is used to gate the outputs of three of the modules to the voter, thereby setting up a TMR configuration. If at any time one of the three active modules disagrees with the others, it is switched out and a spare substituted. (It is assumed that two active modules will not fail simultaneously.) Several other hybrid redundancy schemes have since been proposed [14,15] in an attempt to reduce the switch complexity and hence improve system reliability even further.

In illustrating the three classes of redundancy schemes, for the sake of simplicity, we have selected examples where the voting and/or fault detection and reconfiguration is carried out exclusively by hardware. These functions can also be performed by software, or by some combination of hardware and software, thereby increasing even further the diversity of options available to the designer of a fault tolerant system. Unfortunately no systematic approach to fault tolerant design has as yet been developed. While early ultra-reliable designs, such as the Saturn V launch vehicle processor [16], employed the simple TMR scheme, more ambitious present day designs, such as the STAR [17], use a combination of redundancy techniques in an attempt to best match the redundancy method used to the subsystem to be protected. This makes it extremely difficult to estimate, a priori, the reliability of these systems. Reliability models have been constructed to account for the fault handling capabilities of such systems through the concept of cov-

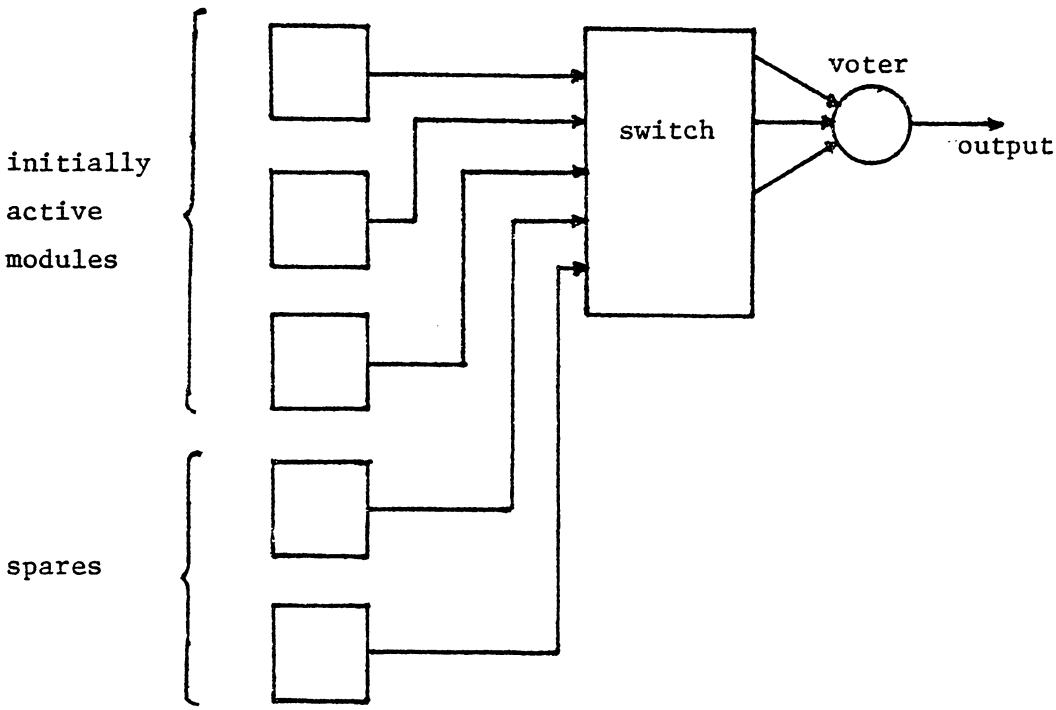

FIGURE 1.3: Hybrid Redundancy.

erage (defined earlier to be the conditional probability of system recovery following a failure). But coverage depends on many diverse factors like fault latency and intermittency, and on how well the types of failures that actually occur are covered by the failures anticipated and protected against by the architecture. Despite much work on identifying and modeling factors that affect coverage [18], it remains virtually impossible to estimate its value with enough accuracy to get meaningful reliablity estimates for ultra-reliable systems, where system reliablity is an extremely sensitive function of coverage. Reliability values for most present day fault tolerant systems can be obtained only after they have been designed and tested. This is a serious drawback because it implies that the design of such systems must be an expensive iterative process involving design modification and testing until the desired reliability is obtained. And this may be altogether impractical for proposed ultra-reliable systems [2,19] with failure probabilities so small (10^{-9} for a 10 hour mission) that their testing and validation with respect to the reliability specification remains a complex and unsolved problem.

An additional problem with the dynamic redundancy schemes being widely used in ultra-reliable designs lies in their handling of module failures due to transient or intermittent faults. Because of the advances in LSI technology, redundancy in present day fault tolerant designs is being applied at the level of processors, memories, I/O controllers, etc. The redundant modules in a subsystem are generally complex sequential circuits containing memory. A transient fault in a sequential

circuit can alter the state of the circuit, resulting in continued erroneous operation until the circuit is resynchronized. A module can, therefore, be disabled by a transient fault. Such a disabled module has the same effect on the subsystem as a module with a permanent failure, even though the former has the hardware capability to operate correctly. Thus a build up of disabled subsystems beyond the fault tolerance capability of the system can cause system failure, even though the hardware resources exist in the system for continued correct operation. This is a significant drawback, particularly in view of recent studies [20,21] which indicate that failures in computer systems due to transient faults are perhaps 10 to 50 times more frequent than those due to permanent faults. The problem can be overcome by building into the system the capability of automatically diagnosing and separating module failures due to transient and permanent faults, and then reassigning the former to the pool of available spares. However, the implementation of such a strategy in either software or hardware is extremely complex and is likely to have an adverse effect on coverage because of the possibility of a module with a permanent fault being incorrectly diagnosed and added to the pool of fault free spares.

In view of these serious problems associated with the current approach to ultra-reliable design, it is clearly very desirable to develop a systematic way of designing fault tolerant systems to desired reliability specification. This was the primary motivation for the research described in this dissertation.

To facilitate the systematic design of highly reliable systems, we have developed a new hardware redundancy scheme called Periodically Self Restoring Redundancy (PSRR). In this dissertation we show how a computer system can be systematically designed to meet desired reliability specifications using the PSRR scheme.

1.3 THE PROPOSED PSRR SCHEME

The proposed periodically self restoring redundancy is a fault tolerance approach that has been developed to protect against system failure caused by the failure of hardware components. Like most other present day redundancy schemes, it does not attempt to protect against failures resulting from improper design or software errors. While it is recognized that a truly fault tolerant system must also address these issues (for a discussion, see [22]), it is assumed here that the system has been properly designed and that the software is error free. PSRR is effective at preventing system failure caused by both permanent and transient (intermittent) faults.

To realize fault tolerance, the PSRR scheme employs N computing units (CU's) operating redundantly in tight synchronization. Each CU has all the computational capabilities of the desired fault tolerant system and, in general, is made up of processors, memories and I/O units. System input is simultaneously provided to all N CU's. System output is taken to be the consensus of the N outputs available from the CU's, decided according to decision rules to be defined in later chapters. If

the system is operational, the consensus output is assured of being error free.

The state of the CU in the system is defined by the binary logical state of all the memory elements in the CU. These memory elements include all the R/W memory in the CU, as well as the registers in the processor and other subsystems that constitute the CU.

Definition 1.1: Two Cu's are said to be in the same state if and only if for each memory element in the CU, the corresponding memory elements in the other CU has the same binary logic value.

Definition 1.2: At any point in the operation of a PSRR system, the operational CU state is the state that the CU's would be in if the operation of the system had been completely free of failures. A CU is said to be operational if it is in the operational CU state.

Theorem 1.1: The input of an operational CU is error free.

Proof: The theorem follows directly from Definition 1.2 and the assumption that the system is free of design faults and is programmed with error free software.

The failure of a hardware component may be either permanent or transient (intermittent) in nature. In the proposed redundant system of N CU's operating in synchronization, a CU may fail and fall out of step due to errors caused by both transient and permanent component failures. This is because a transient failure can alter the state of the CU and possibly result in continued erroneous computations until the system is restored. A CU that has failed due to a transient will be

considered to have temporarily failed. A CU with a permanent failure in a hardware component is said to have permanently failed. While permanently failed CU's obviously cannot be reliably resynchronized with the rest of the system, if the CU failure is temporary (due to a transient), restoring it to an operational state will insure that it stays in synchronization with the rest unless it experiences another failure. In the proposed system this is done by the N CU's communicating with each other periodically to restore failed CU's. To facilitate this communication, each CU is directly linked to all N-1 other CU's. The restoration program is initiated by an interrupt from an external fault tolerant clocking circuit (such as the one described in [23]) and is executed by each CU out of read only memory (ROM). This insures that a transient failure that may have corrupted the memory in a CU does not prevent it from participating in the restoration process. During the restoration interval the N CU's compare their states and restore themselves to a mutually agreed consensus state. The exact rules defining the consensus are described in later chapters. Thus, if over the restoration interval, enough CU's remain operational to make the operational CU state the consensus state, the temporarily failed CU's will be restored provided they do not experience additional failures during the restoration interval.

Definition 1.3: At any point in its operation, a PSRR system is said to be operational if and only if the consensus CU state is the operational CU state.

We shall, in later chapters, define decision rules for deciding the system output from among the N outputs available from the CU's, such that if the system is operational, the system output is assured of being error free.

Operation of the redundant system is broken up into computing intervals, when the system is performing useful computation, and restoration intervals, when temporarily failed processors are being restored. The length of the restoration interval is determined by the time required to execute the restoration program and is a function of the number of CU's in the system and memory size of each CU. The length of the computation interval is a design option and determines the computation-restoration (C-R) cycle time. Shorter C-R cycle times imply more frequent system restoration. This increases system reliability by reducing the probability of system failure due to an accumulation of temporarily failed processors.

It should be clear from the above description that the PSRR scheme with short C-R cycles is very effective in protecting the system against transient failures. This is a major advantage because transients are known to cause the large majority of failures in computer systems. Because a PSRR system can stay operational with fewer than N correctly operating CU's, it can also handle a certain number of permanent faults. However, a quantitative measure of fault tolerance can only be obtained after defining the rules to decide the consensus output and CU state for such systems. This is done when the reliability models for PSRR systems are developed in later chapters.

The proposed PSRR scheme offers other significant advantages over presently used redundancy schemes. Because it does not employ fault detection and reconfiguration to implement fault tolerance, its reliability is not degraded by complex fault diagnosis and reconfiguration circuitry, and the associated coverage factor. Also, fault latency is not a problem. PSRR systems based on the proposed scheme employ high level redundancy and can be largely implemented from proven off-the-shelf hardware (such as processors and memories), with only minimal need for specialized hardware design. Finally, as we show in this dissertation, such systems can be accurately modeled for reliability calculations in terms of the failure rates of their building blocks. This allows the a priori estimation of reliability during the initial design phase, thereby making it possible to systematically design PSRR systems to meet desired reliablity specifications.

In Chapter Two we present and analyze a triple redundant PSRR system that uses a simple majority vote to decide the consensus CU state and system output. Making some realistic simplifying assumptions, a reliability model for such a system is developed and closed form expressions for its reliability and mean time to failure (MTTF) obtained. The possibility of CU failures before the start of the mission is also incorporated into the reliability model. Chapter Three defines a weighted plurality votin scheme for optimally deciding the consensus CU state in general N-redundant PSRR systems. Based on this scheme, a comprehensive reliability model for N-redundant systems is developed. The

model relaxes all the simplifying assumptions made in Chapter Two. Chapter Four describes a restoration algorithm to implement the plurality vote defined in Chapter Three. The execution complexity of this restoration algorithm with respect to redundancy N is also analyzed. Chapter Five studies various tradeoffs in the design of PSRR systems and arrives at a design procedure for implementing such systems to meet desired performance and reliability specifications. The research is summarized in Chapter Six.

Chapter II

TRIPLE REDUNDANT PSRR SYSTEMS

2.1 INTRODUCTION

In this chapter we analyze triple redundant PSRR systems that employ a simple majority vote among the three CU's to decide the system output, and the consensus CU state during restoration.

Definition 2.1: A simple majority opinion in a triple redundant PSRR system is the opinion of the agreeing CU's if two or more CU's agree; it is undefined for the case when all three CU's disagree with one another.

Note that by Definition 1.3, a triple redundant PSRR system will be operational as long as two or more of the three CU's are operational. Also, if the system stays operational over an entire restoration interval, then any temporarily failed CU is assured of being restored to the operational state, provided it does not experience additional failures during this restoration interval.

Theorem 2.1: The output of a triple redundant PSRR system, obtained by a simple majority vote on the output words of the three CU's, is error free if the system is operational.

Proof: If the system is operational, at least two CU's must be in operation. By Theorem 1.1 the outputs of these two CU's must be error free. Therefore, the output obtained by a word by word majority vote on the three outputs available from the CU's must also be error free.

The restoration process in PSRR systems employing simple majority voting is easily implemented. In each CU the non-maskable hardware interrupt initiating the restoration interval causes control to be transferred to a restoration program stored in ROM. The restoration program first saves the CU state in memory by storing the contents of all registers in the processor, and the other subsystems in the CU, at predetermined locations in the CU memory. Then the restoration programs in the three CU's interact to vote, word by word, on their entire (read/write) memory, replacing disagreeing words with the majority opinion. The voting is carried out over the dedicated links interconnecting the CU's, with the CU's operating in tight synchronization. At the end of the vote, the registers in each CU are reloaded from its memory, thereby restoring it to the consensus state.

In the next section we develop a Markov model to estimate the reliability of such triple redundant PSRR systems.

2.2 RELIABILITY MODEL

Definition 2.2: The reliability R(t) of a fault tolerant system to time t = T is the probability that the system stays operational over the inverval $0 \leq t \leq T$, given that it was completely free of failures at the initial time t=0.

For the purpose of this analysis, we shall assume that the restoration interval is negligibly short and that no failure takes place during the restoration process. This assumption implies that at the instant

just after restoration, if the system is operational, it does not contain a temporarily failed CU. At any such instant, therefore, the system must be in one of the following three states.

State 1: All CU's operational.

State 2: Exactly two CU's operational, the third having permanently failed.

State 3: Failed system due to more than one failed CU. (This may result from any combination of temporary and permanent CU failures.)

Using these states we can model the operation of the proposed redundant system as a three state Markov chain [24]. The state transition probabilities over one C-R cycle time form the one step matrix of transition probabilities,

$$T = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \qquad (2.2\text{-}1)$$

It can be easily seen that some of the elements of T are trivial. $p_{21} = 0$ because once the system has a CU with a permanent failure (state 2), it can never go to a state with all operational CU's (state 1). Also $p_{31} = 0$, $p_{32} = 0$ and $p_{33} = 1$, because if the system fails (state 3) it can never recover to an operational state.

Making these entries in T we find that the matrix of transition probabilities is upper triangular.

$$T = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ 0 & p_{22} & p_{23} \\ 0 & 0 & 1 \end{bmatrix} \qquad (2.2\text{-}2)$$

The five remaining unknown elements in T depend on the CU failure probabilities and the C-R cycle time for the system. From these parameters, the remaining elements can be evaluated as explained below.

Let p_t be the probability that a transient component failure capable of causing temporary CU failure, occurs during a C-R cycle. Let $q_t = (1 - p_t)$ be the probability that such a transient does not occur. We shall assume that p_t is the same whether or not the CU is operational, and that the transient always causes a temporary failure in an operational CU. Similarly let p_p be the probability that a permanent CU failure occurs over a C-R cycle, $q_p = (1 - p_p)$ is the probability that such a failure does not occur. It is assumed that transient and permanent failures are mutually independent.

In the matrix of transition probabilities T, p_{11} is the probability that a failure free system will still be failure free after one C-R cycle. Clearly this requires that no permanent failure and at most one temporary failure occurs over this interval. (Since we always consider the

state of the system at the instant following restoration, a single temporary failure will always be restored.) Therefore

$$p_{11} = q_p^3 \times \{ q_t^3 + 3q_t^2 p_t \} \qquad (2.2\text{-}3)$$

q_p^3	q_t^3	$3q_t^2 p_t$
probability of no permanent failure	probability of no temporary failures	probability of exactly one temporary failure

It can be similarly seen that

$$p_{12} = 3q_p^2 p_p \times \{ q_t^3 + q_t^2 p_t \} \qquad (2.2\text{-}4)$$

$3q_p^2 p_p$	q_t^3	$q_t^2 p_t$
probability of exactly one permanent failure	probability of no temporary failures	probability of temporary failure in the same CU that experiences permanent failure

Also since $p_{11} + p_{12} + p_{13} = 1$

$$p_{13} = 1 - p_{11} - p_{12} \qquad (2.2\text{-}5)$$

p_{22} is the probability that a state with one permanently failed CU is retained over a C-R cycle. This requires that no additional failures take place in the two CU's. Therefore,

$$p_{22} = q_p^2 \times q_t^2 \qquad (2.2\text{-}6)$$

Again since $p_{22} + p_{23} = 1$

$$p_{23} = 1 - p_{22} \qquad (2.2\text{-}7)$$

Once the one step transistion probability matrix T is obtained, by the theory of Markov Chains, the transition probabilities over n C-R cycles can be obtained by evaluating $(T)^n$.

The reliability of the system R(t) to n C-R cycle times is the probability that a failure free system stays operational over n C-R cycles. Since state 1 represents a failure free system and state 3 a failed system, the (1,3) entry in $(T)^n$, $p_{13}^{(n)}$, gives the probability that a system that was initially failure free, fails over n C-R Cycles. Therefore

$$R(nt_o) = 1 - p_{13}^{(n)} \tag{2.2-8}$$

We next obtain $p_{13}^{(n)}$ and hence $R(nt_o)$, in closed form in terms of the one step state transition probabilities. This is done by using the fact that since T is upper triangular, $(T)^n$ is also upper triangular.

$$(T)^n = (T)^{n-1} \times T \tag{2.2-9}$$

Writing $p_{12}^{(k)}$ as $(1 - p_{11}^{(k)} - p_{13}^{(k)})$, (2.2-9) can be written as

$$\begin{bmatrix} p_{11}^{(n)} & (1 - p_{11}^{(n)} - p_{13}^{(n)}) & p_{13}^{(n)} \\ 0 & p_{22}^{(n)} & p_{23}^{(n)} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2-10}$$

$$= \begin{bmatrix} p_{11}^{(n-1)} & (1 - p_{11}^{(n-1)} - p_{13}^{(n-1)}) & p_{13}^{(n-1)} \\ 0 & p_{22}^{(n-1)} & p_{23}^{(n-1)} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} p_{11}(1-p_{11}-p_{13})p_{13} \\ 0 \quad p_{22} \quad p_{23} \\ 0 \quad 0 \quad 1 \end{bmatrix}$$

Obtaining expressions for $p_{12}^{(n)}$ from both sides of this identity we get

$$1 - p_{11}^{(n)} - p_{13}^{(n)} = p_{11}^{(n-1)}(1 - p_{11} - p_{13}) + p_{22}(1 - p_{11}^{(n-1)} - p_{13}^{(n-1)}) \tag{2.2-11}$$

Noting that for a triangular matrix $p_{11}^{(k)} = (p_{11})^k$ and $p_{22}^{(k)} = (p_{22})^k$ we get

$$1 - (p_{11})^n - p_{13}^{(n)} = (p_{11})^{n-1}(1 - p_{12} - p_{13}) + p_{22}\{1 - (p_{11})^{n-1} - p_{13}^{(n-1)}\} \tag{2.2-12}$$

Rearranging terms gives the recurrence relation

$$p_{13}^{(n)} - p_{22}p_{13}^{(n-1)} = 1 - p_{22} - (p_{11})^{n-1}(1 - p_{13} - p_{22}) \tag{2.2-13}$$

This equation can be solved using known methods such as the one described in Liu [25]. The general solution to the recurrence relation is

$$p_{13}^{(n)} = \frac{-p_{12}}{p_{22} - p_{11}}(p_{22})^n + \frac{(p_{23} - p_{13})}{p_{22} - p_{11}}(p_{11})^n + 1 \tag{2.2-14}$$

Thus the reliability of the periodically self restoring system can be obtained by substituting for $p_{13}^{(n)}$ in (2.2-8) and is given by

$$R(nt_o) = \frac{p_{12}}{p_{22}-p_{11}}(p_{22})^n - \frac{(p_{23}-p_{13})}{p_{22}-p_{11}}(p_{11})^n \tag{2.2-15}$$

The above derivation establishes Theorem 2.2

Theorem 2.2: The reliability of a triple redundant PSRR system with simple majority voting is given by

$$R(nt_o) = \frac{p_{12}}{p_{22}-p_{11}}(p_{22})^n - \frac{(p_{23}-p_{13})}{p_{22}-p_{11}}(p_{11})^n$$

Theorem 2.2 gives system reliability over n C-R cycles in terms of the one step state transition probabilities. Equations (2.2-3) through (2.2-7) can be used to obtain the required transition probabilities if the probability values for temporary and permanent CU failures over a C-R cycle are known.

For electronic subsystems with no built in redundancy, it is often reasonable to assume constant failure rates over the life of the system. If a constant temporary failure rate λ_t is assumed, then the probability of a CU experiencing temporary failure over a C-R cycle is given by $p_t = [1-\exp(-\lambda_t t_o)]$. Similarly, for a constant permanent failure rate λp, the probability of a CU experiencing permanent failure over a C-R cycle is $p_p = [1 - \exp(-\lambda_p t_o)]$. Typical failure rates were assumed to obtain the reliability plots in Figures 2.1 and 2.2. Siewiorek et al. [20] have shown that permanent faults cause only a small fraction of all detected failures. For this reason the transient failure rate λ_t was taken to be 0.01 per hour (an average of one failure every 100 hours) and the permanent failure rate λ_p, to be 0.001 per hour.

Figure 2.1 shows plots of reliability versus time for different C-R cycle times, t_o. Figure 2.2 shows the same plots for short mission times important in ultra-reliable design. As expected the plots indicate that reliability improves with more frequent restoration i.e. as the C-R cycle time is decreased. Also included in the figures are reliability plots for a conventional TMR design [9] ($t_o = \infty$) and a hypothetical system that recovers instantaneously from temporary failures in any one CU as long as the other two CU's are operational ($t_o = 0$). The reliability of this hypothetical system is derived in Section 2.4. These two plots provide bounds on the reliability of the proposed scheme. If the C-R cycle time is made very large, in the limit infinite, restoration does not take place in the proposed scheme before the system fails, and its reliability is reduced to that of a conventional TMR design, which has no provision for restoring failed CU's. On the other hand as the C-R cycle time is reduced and approaches zero, recovery from temporary failures is almost instantaneous and the proposed scheme approaches the hypothetical system described above. In practice this upper limit on reliability can never be reached since it requires that both the computational interval and the restoration interval be of zero duration.

Notice in Figures 2.1 and 2.2 the substantial improvement in reliability offered by PSRR systems as compared to TMR. This is particularly true of PSRR systems with short C-R cycle times. For example, over a 16-hour mission time, Figure 2.2 indicates that the TMR system is eight times more likely to fail than a PSRR system with $t_o = 0.5$

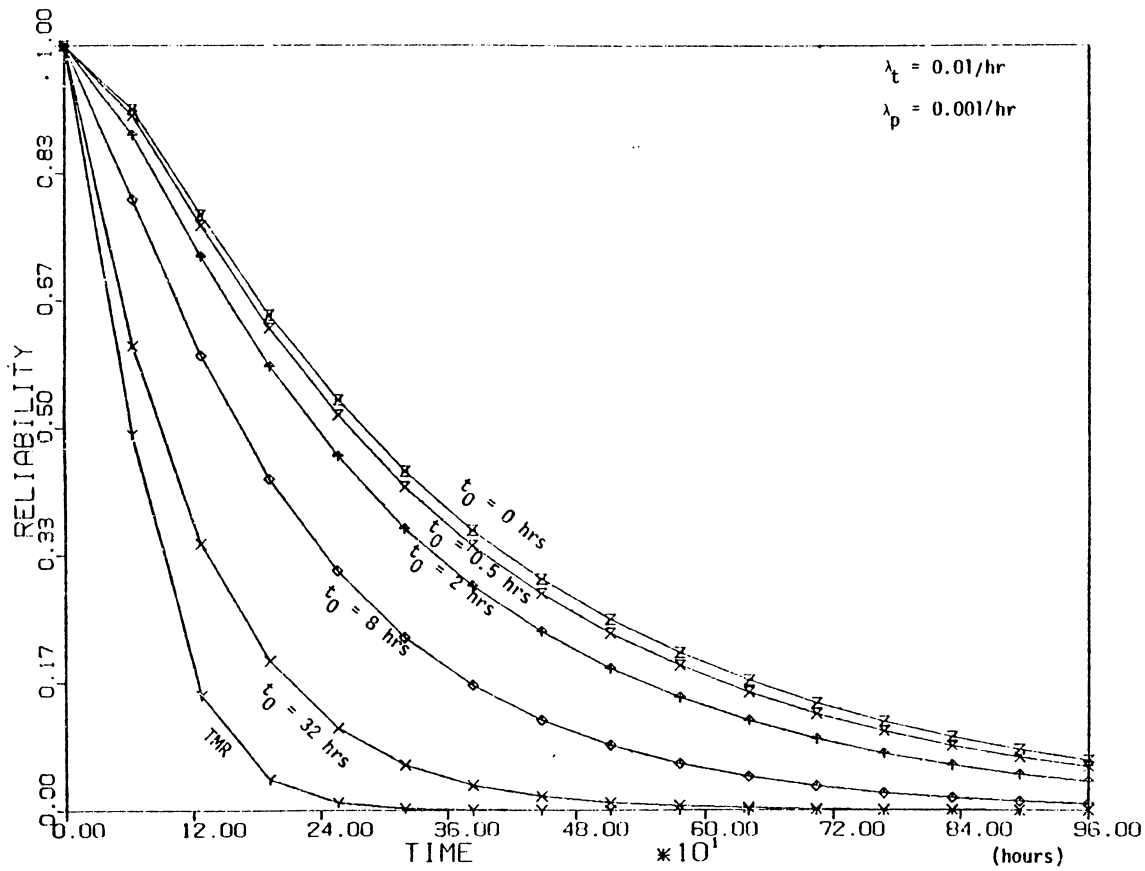

FIGURE 2.1: Reliability Plots for the N = 3 PSRR System.

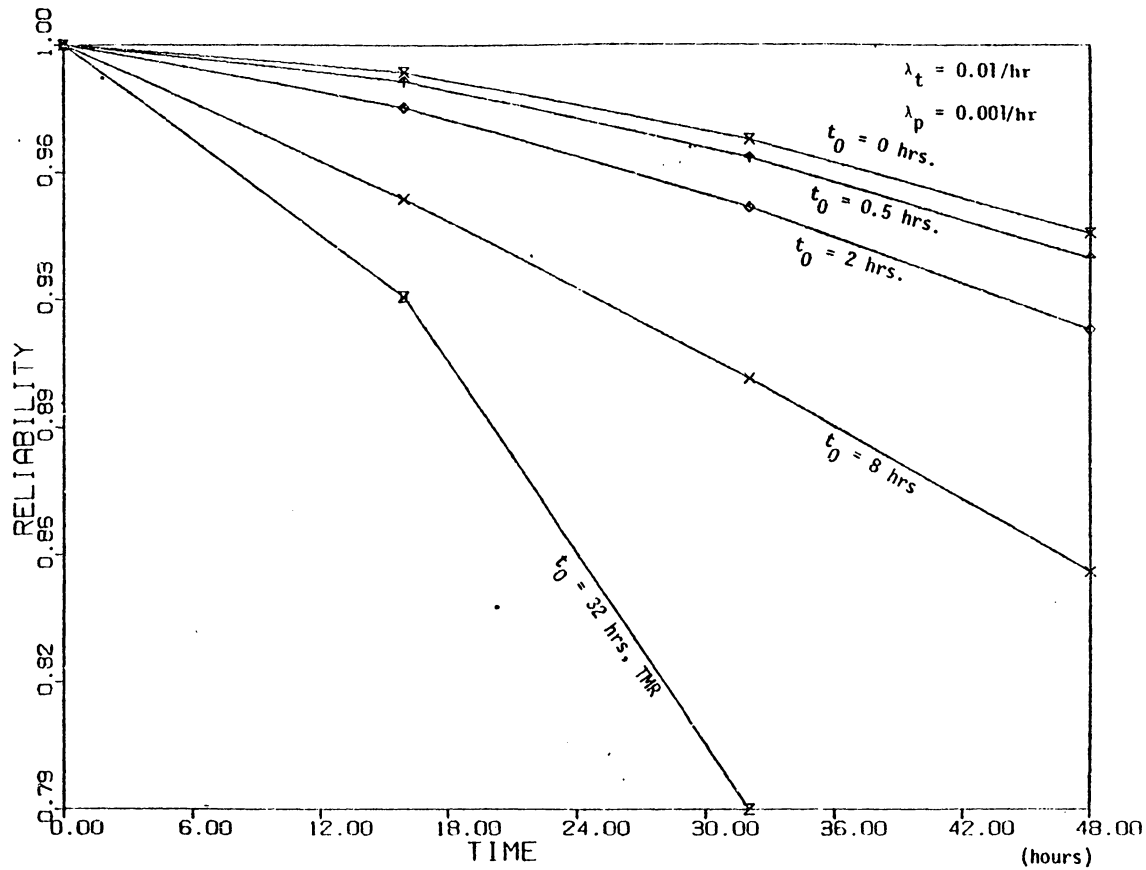

FIGURE 2.2: Reliability Plots for the N = 3 PSRR System for Short Missions.

hours. Recall that the probability of system failure is (1-reliability). Such small values of C-R cycle time t_o appear to be quite practical for PSRR systems. With processor instruction times in microseconds, it should be possible to vote on tens of thousands, and perhaps hundreds of thousands, of memory words a second. Therefore the restoration interval in typical systems will be a few seconds long. This will allow C-R cycle times to be of the order of minutes, without substantially reducing of computing power due to lost computation time during the restoration process.

2.3 MEAN TIME TO FAILURE CALCULATION

Another parameter of interest in fault tolerant systems is the mean time to failure (MTTF). In this section we derive the MTTF for triple redundant PSRR systems.

Let μ_{ij} be the average number of C-R cycles taken by a system starting in state i to go to state j for the first time. Then for the three state triplicated PSRR system under discussion, it takes, on the average μ_{13} C-R cycles for a system with all CU's operational to fail. The system MTTF is, therefore, given by $\mu_{13} \times t_o$.

To evaluate μ_{13}, consider a system one C-R cycle after starting in state 1. At this point in time the system may be in any one of the three system states 1, 2 or 3, with probability p_{11}, p_{12} and p_{13} respectively. Therefore, from this instant the average number of C-R cycles to system failure is given by $p_{11}\mu_{13} + p_{12}\mu_{23} + p_{13}\mu_{33}$. Since

the system was in state 1 one C-R cycle before the instant under consideration,

$$\mu_{13} = 1 + p_{11}\mu_{13} + p_{12}\mu_{23} + p_{13}\mu_{33} \tag{2.3-1}$$

It can be similarly seen that

$$\mu_{23} = 1 + p_{21}\mu_{13} + p_{22}\mu_{23} + p_{23}\mu_{33}. \tag{2.3-2}$$

Noting that $\mu_{33} = 0$ (the average number of C-R cycles to go from state 3 to itself for the first time is 0) and $p_{21} = 0$, the above two equations reduce to

$$\mu_{13} = 1 + p_{11}\mu_{13} + p_{12}\mu_{23} \tag{2.3-3}$$

$$\mu_{23} = 1 + p_{22}\mu_{23} \tag{2.3-4}$$

Solving these simultaneously we get

$$\mu_{23} = \frac{1}{1-p_{22}} \tag{2.3-5}$$

$$\mu_{13} = \frac{p_{12} + p_{23}}{(1-p_{11})(1-p_{22})} \tag{2.3-6}$$

Therefore, the system MTTF, which is $\mu_{13}t_o$, is given by

$$\text{MTTF} = \frac{(p_{12} + p_{23})t_o}{(1-p_{11})(1-p_{22})} \tag{2.3-7}$$

It should be noted that the above expression for MTTF is valid only for $\mu_{13} >> 1$. This is because in the derivation of μ_{13}, we have impliicitly assumed that it always takes more than one C-R cycle for the system to fail, an assumption that is not valid unless $t_o << 1/\lambda_t, 1/\lambda_p$.

Theorem 2.3: The mean time to failure for a triple redundant PSRR system with simple majority voting and $t_o << 1/\lambda_t, 1/\lambda_p$ is given by

$$MTTF = \frac{(p_{12} + p_{23})t_o}{(1-p_{11})(1-p_{22})}$$

The inequality $t_o << 1/\lambda_t, 1/\lambda_p$ is satisfied for the range of values of t_o in Figure 2.3, which displays a plot of MTTF versus t_o for $\lambda_t = 0.01$ per hour and $\lambda_p = 0.001$ per hour. We have seen in Figure 2.1 that the reliability of a PSRR system always improves with more frequent restoration. Therefore, the system MTTF must also increase with reduced t_o. While this is not immediately apparent from the expression for MTTF in Theorem 2.3 because the state transition probabilities also depend on t_o, it is confirmed by Figure 2.3. Again as expected, the longest MTTF is obtained when t_o approaches zero. As t_o grows large, the system MTTF asymtotically approaches that for a conventional TMR system with no provisions for restoring temporarily failed CU's. This is well known [9] to be $5/6(\lambda_t + \lambda_p)$.

Notice in Figure 2.3 the very substantial improvement in MTTF achieved by PSRR systems as compared with TMR. For short C-R cycle times of up to 2 hours expected for such systems, the MTTF for PSRR systems is over four times that for TMR systems.

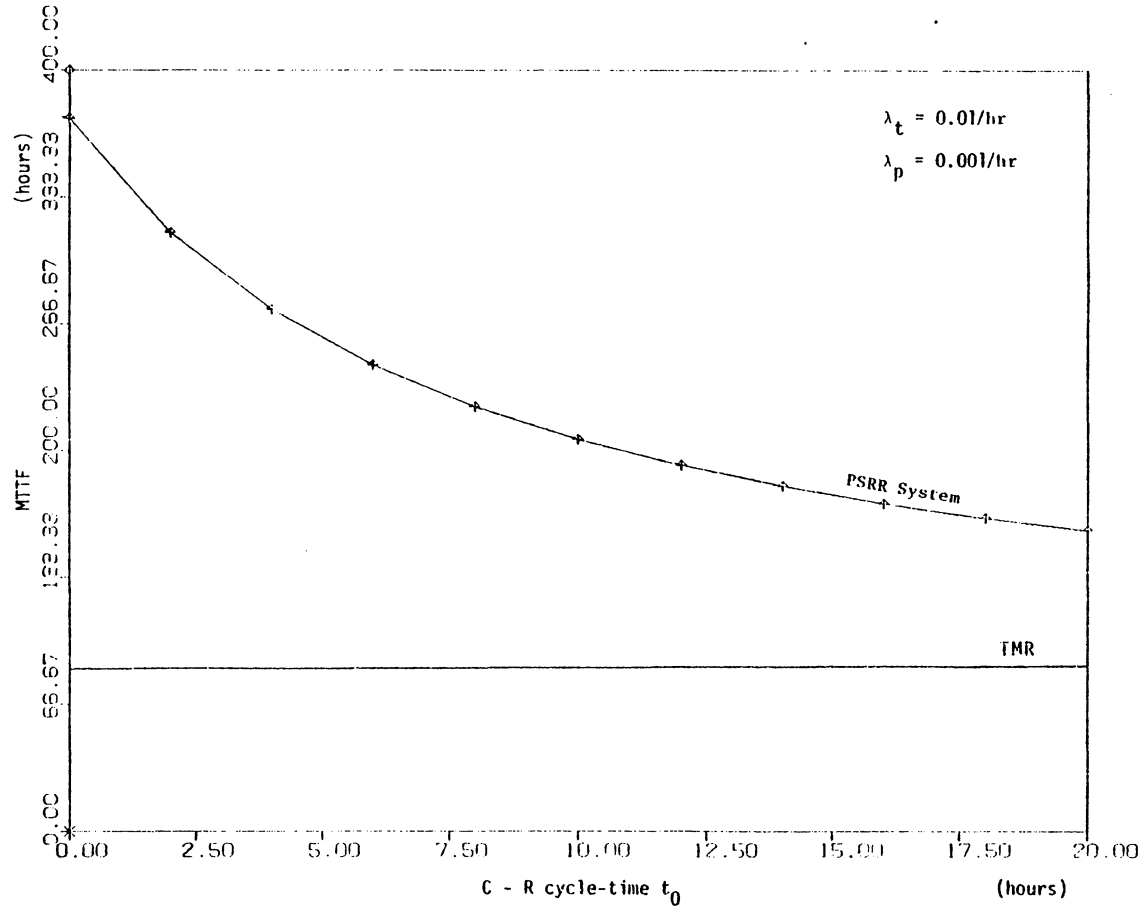

FIGURE 2.3: MTTF versus C-R Cycle Time t_0 for the N = 3 PSRR System.

2.4 INSTANTANEOUSLY SELF RESTORING SYSTEMS

Consider a hypothetical triple redundant PSRR system with $t_o = 0$. In such a system a temporarily failed CU recovers instantly as long as the other two CU's are operational. It should be apparent that if in an actual system the C-R cycle time is small as compared to the mean time between temporary CU failures, its reliability will approach that of the hypothetical instantaneously self restoring system with $t_o = 0$. This can be confirmed by reviewing the plots in Figure 2.1. We have argued earlier in Section 1.2 that in typical PSRR systems, C-R cycle times are expected to be of the order of minutes. This would be very small as compared with the mean time between temporary CU failures for most systems. It is therefore desirable to obtain, if possible, a simple reliability expression for the hypothetical ($t_o = 0$) system since it would be a good approximation for the reliability of practical triple redundant PSRR systems. In this section we derive such an expression for systems with constant failure rates.

Theorem 2.2 gives the reliability of a triple redundant PSRR system over a time period nt_o, corresponding to n C-R cycles. It obviously cannot be used to obtain the reliability of the instantaneously self restoring system with $t_o = 0$. We shall, therefore, use a different approach.

Let $R_t[T]$ be the reliability of a CU over a time interval T, with respect to temporary failures. For a constant temporary failure rate λ_t, $R[T] = \exp(-\lambda_t T)$. Similarly, let the reliability of a CU with re-

spect to permanent failures be $R_p[T] = \exp(-\lambda_p T)$, where λ_p is the constant permanent failure rate.

A CU in the instantaneously self restoring system recovers instantly from a temporary failure as long as the other two CU's are operational. Therefore, the system will stay operational over the interval $0 \leq t \leq T$ in the following two cases.

1. No permanent CU failures occur over the interval $0 \leq t \leq T$. The probability of this event is $R_p[T]^3$.

2. Exactly one permanent CU failure occurs over the interval $0 \leq t \leq T$, at $t = \tau$. Following this permanent failure, no temporary failures occur in the two operational CU's over the interval $\tau \leq t \leq T$. The probability of this event can be seen to be

$$3R_p[T]^2 \int_0^T \lambda_p \exp(-\lambda_p \tau) R_t[T - \tau]^2 d\tau$$

The reliability of the instantaneously self restoring system is, therefore, given by

$$\begin{aligned} R_0[T] &= \exp(-3\lambda_p T) \\ &\quad + 3\exp(-2\lambda_p T) \int_0^T \lambda_p \exp(-\lambda_p \tau) \exp(-2\lambda_t (T-\tau)) d\tau \end{aligned} \tag{2.4-1}$$

On integration and simplification, this yields

$$\begin{aligned} R_0[T] &= \frac{3\lambda_p}{\lambda_p - 2\lambda_t} \exp(-2(\lambda_t + \lambda_p)T) \\ &\quad - \frac{2(\lambda_p - \lambda_t)}{\lambda_p - 2\lambda_t} \exp(-3\lambda_p T) \end{aligned} \tag{2.4-2}$$

Theorem 2.4: The reliability of an instantly self restoring triple redundant system with simple majority voting is given by

$$R_o(T) = \frac{3\lambda p}{\lambda p - 2\lambda t} e^{-2(\lambda t + \lambda p)T} - \frac{2(\lambda p - \lambda t)}{\lambda p - 2\lambda t} e^{-3\lambda p T}$$

Reliability versus mission time plots for such an instantaneously self restoring system are displayed in Figures 2.1 and 2.2 and have already been discussed in Section 2.2. A review of the plots establishes the validity of the above expression as an approximation to the reliability of practical PSRR systems. It should be clear that it is computationally much easier to use this expression for reliability estimation than the one in Theorem 2.2.

The reliability expression in Theorem 2.4 can be used to derive the mean time to failure for the system under discussion. For this derivation we invoke the well known reliability theory result [27] that for systems with constant failure rates,

$$MTTF = \int_0^\infty R\,[T]dt \qquad (2.4\text{-}3)$$

Substituting for R[T] from (2.4-2) and integrating we get the MTTF of the instantaneously self restoring system to be

$$MTTF_o = \frac{5\lambda_p + 2\lambda_t}{6\lambda_p^2 + 6\lambda_p\lambda_t}. \qquad (2.4\text{-}4)$$

Theorem 2.5: The mean time to failure of a instantaneously self restoring triple redundant system with simple majority voting is given by

$$MTTF_o = \frac{5\lambda_p + 2\lambda_t}{6\lambda_p^2 + 6\lambda_p\lambda_t}$$

Theorem 2.5 is useful in estimating the MTTF for triple redundant PSRR systems with short C-R cycles. For λ_t = 0.01 per hour λ_p = 0.001 per hour, it gives a MTTF of 378, which is consistant with the plot in Figure 2.3.

2.5 INITIAL FAILURE PROBABILITIES

The reliability of most current fault tolerant systems is defined and evaluated assuming that it is known with certainty that all redundant subsystems are fault free at the start of the mission. For practical systems, this may not be a valid assumption. Due to the difficulty of completely testing complex systems, testing procedures usually establish to a high probability (fault coverage) that the system is fault free. Unfortunately even a small possibility of a faulty subsystem at the start of the mission can very significantly affect system reliability in highly reliable systems. The PSRR system, therefore, has an important advantage in that the possibility of one or more CU failures before the start of the mission can be incorporated into the reliability model, as shown in this section.

Let p_{st} be the probability that a CU in the system has failed temporarily before the start of the mission. Let p_{sp} be the probability that

a CU in the system has failed permanently before the start of the mission. Then the system must be in one of the following four states at the beginning of the mission:

a) All CU's operational

b) Two CU's operational, one temporarily failed

c) Two CU's operational, one permanently failed

d) Failed system with two or more failed CU's.

If we assume that system operation always begins with a restoration interval (which as before is assumed to be negligibly short in duration so that there is no possibility of failures during this interval), then this instantaneous restoration will always take a system initially in state (b) to state (a). Therefore, for all practical purposes, states (a) and (b) can be grouped together to form a single state corresponding to state 1 in the Markov model of the previous sections. State (c) corresponds to state 2 and the failed system state (d) to state 3.

The probability that the system is in state 1 at the start of the mission is given by

$$p_{S1} = (1 - p_{sp})^3 \times \{ (1 - p_{st})^3 + 3p_{st}(1 - p_{st})^2 \}. \qquad (2.5\text{-}1)$$

probability of no initial permanent failure

probability of at most one initial temporary failure

The probability that the system is initially in state 2 is

$$p_{S2} = 3p_{sp}(1-p_{sp})^2 \times (1-p_{st})^2. \qquad (2.5\text{-}2)$$

probability of exactly one initial permanent failure

probability of no temporary failures in remaining two CU's

And the probability that the system has failed before the start of the mission (i.e. that it is in state 3) is given by

$$p_{S3} = 1 - p_{S1} - p_{S2} \tag{2.5-3}$$

The probability that a system with these initial state probabilities fails before the completion of n C-R cycles is $p_{S1}\, p_{13}^{(n)} + p_{S2}\, p_{23}^{(n)} + p_{S3}$. Therefore the system reliability to n C-R cycles, with initial failure probabilities considered, is given by

$$R'(nt_o) = 1 - p_{S1}p_{13}^{(n)} - p_{S2}p_{23}^{(n)} - p_{S3} \tag{2.5-4}$$

In equation (2.5-4), $p_{13}^{(n)}$ can be obtained from (2.2-14).

Also, because T is 3x3 and upper triangular,

$$p_{12}^{(n)} = 1 - p_{22}^{(n)} = 1 - (p_{22})^n. \tag{2.5-5}$$

Substituting for $p_{13}^{(n)}$ and $p_{23}^{(n)}$ and simplifying we get

$$R'(nt_o) = 1 - p_{S1} - p_{S2} - p_{S3} - \frac{(p_{23} - p_{13})}{(p_{22} - p_{11})}(p_{11})^n p_{S1} \tag{2.5-6}$$

$$+ \frac{p_{12}}{p_{22} - p_{11}}(p_{22})^n p_{S1} + (p_{22})^n p_{S2}$$

Theorem 2.6: The reliability of a triple redundant PSRR system over n C-R cycles, with initial failure probabilities considered is given by

$$R'(nt_o) = 1 - p_{S1} - p_{S2} - p_{S3}\frac{(p_{23} - p_{13})}{(p_{22} - p_{11})}(p_{11})^n p_{S1}$$

$$+ \frac{p_{12}}{(p_{22} - p_{11})}(p_{22})^n p_{S1} + (p_{22})^n p_{S2}$$

The effect of possible temporary and permanent initial CU failures on system reliability is illustrated by the plots in Figure 2.4, which were obtained using the above expression. To isolate the contributions of each of the two types of initial failures and to also observe their combined effect, four reliability plots are displayed. These are for a system with $\lambda_t = 0.01$ per hour, $\lambda_p = 0.001$ per hour and $t_o = 0.1$ hours and (i) no possibility of initial CU failures: $p_{st} = 0$, $p_{sp} = 0$, (ii) the possibility of only temporary intial CU failures: $p_{st} = 0.05$, $p_{sp} = 0$, (iii) the possibility of only permanent initial CU failures; $p_{st} = 0$ $p_{sp} = 0.05$, and (iv) the possibility of both temporary and permanent initial CU failures: $p_{st} = 0.05$, $p_{sp} = 0.05$.

The plots in Figure 2.4 show that both temporary and permanent initial CU failures reduce system reliability at the start of a mission. However, the possibility of a temporary initial CU failure does not have

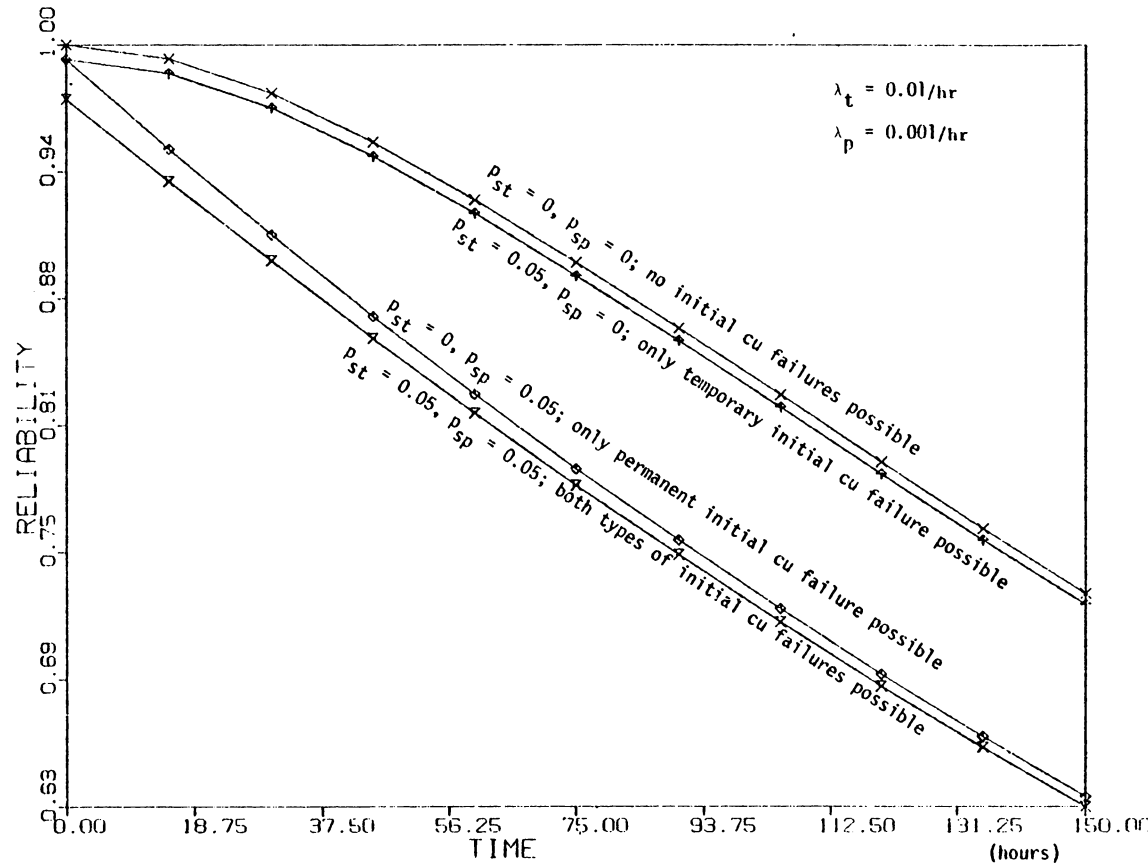

FIGURE 2.4: Reliability Plots Displaying the Effect of Initial CU Failure Probabilities.

any further impact on system reliability over the rest of the mission. This is because any such failure would be restored during the first restoration interval. On the other hand a permanent initial CU failure will exist in the system for all time and continue to degrade system reliability.

Notice from the plots that for equally probable temporary and permanent initial CU failures, the permanent failures degrade the system more severely. Note also that the possibility of initial CU failures has a more significant impact on systems that call for highly reliable operations over short periods. For example, let us compare the increase in system failure probability due to a 0.05 probability of both temporary and permanent initial CU failures, over that for an initial failure free system. For a mission time of 96 hours, the probability of system failure is not significantly increased if only temporary initial CU failures are allowed, and is increased by a factor of 1.5 if only permanent initial CU failures are allowed. On the other hand, for a shorter 16 hour mission time, the increase in system failure probability is much more significant. The 0.05 probability of temporary initial CU failures alone doubles the probability of system failure, while the possibility of permanent initial CU failures increases system failure probability by a factor of 7. The initial CU failure probabilities impact even more significantly on more reliable operation with shorter mission times.

System MTTF can also be derived to take into account the possibility of initial CU failures. Recall that μ_{k3} is the average number of C-R

cycles that it takes for a system in state k to go to state 3 for the first time. At the start of the mission the system is in state 1, 2 or 3 with probability p_{S1}, p_{S2} and p_{S3} respectively. Therefore, the average number of C-R cycles required for the system to go to state 3 is given by

$$N'_{steps} = p_{S1}\mu_{13} + p_{S2}\mu_{23} + p_{S3}\mu_{33} \qquad (2.5\text{-}7)$$

Noting that $\mu_{33} = 0$, and obtaining μ_{23} and μ_{13} from equations (2.3-5) and (2.3-6) respectively, we get

$$N'_{steps} = \frac{p_{S1}(p_{12} + p_{23})}{(1-p_{11})(1-p_{22})} + \frac{p_{S2}}{(1-p_{22})} \qquad (2.5\text{-}8)$$

The system MTTF with initial failure probabilities considered is MTTF' = $N'_{steps} \times t_o$

which is therefore given by

$$MTTF' = \frac{p_{S1}(p_{12} + p_{23})t_o}{(1 - p_{11})(1-p_{22})} + \frac{p_{S2}t_o}{(1-p_{22})} \qquad (2.5\text{-}9)$$

Theorem 2.7: The mean time to failure for triple redundant PSRR systems, with initial failure probabilities considered, is given by

$$MTTF' = \frac{p_{S1}(p_{12} + p_{23})t_o}{(1-p_{11})(1-p_{22})} + \frac{p_{S2}t_o}{(1-p_{22})}$$

Figure 2.5 shows a plot of MTTF' versus t_o for a system with λ_t = 0.01 per hour and λ_p = 0.001 per hour and the same four sets of initial CU failure probabilities plotted in Figure 2.4. Note that for equally probable temporary and permanent initial CU failures, the permanent failures more significantly affect system MTTF. For the range of values of t_o plotted, a 0.05 probability of temporary initial CU failure results in only about a 1% reduction in system MTTF, while the same probability of permanent initial CU failure results in about a 12% reduction in system MTTF. This is to be expected based on the preceding discussion which concluded that the possibility of each permanent initial CU failures has a more significant impact on system reliability.

For the plots in Figures 2.4 and 2.5 the probabilities for temporary and permanent initial CU failures were both taken to be 0.05. While this value is quite pessimistic for practical fault tolerant systems, it was chosen to highlight the effects of the possibility of failed CU's at the start of a mission. The observed trends also hold for more realistic (smaller) initial CU failure probabilities. We therefore conclude that the possibility of initial CU failures, particularly permanent initial CU failures, can significantly degrade the reliability of PSRR systems designed for highly reliable operation over short mission times. Their effect on system MTTF is less significant.

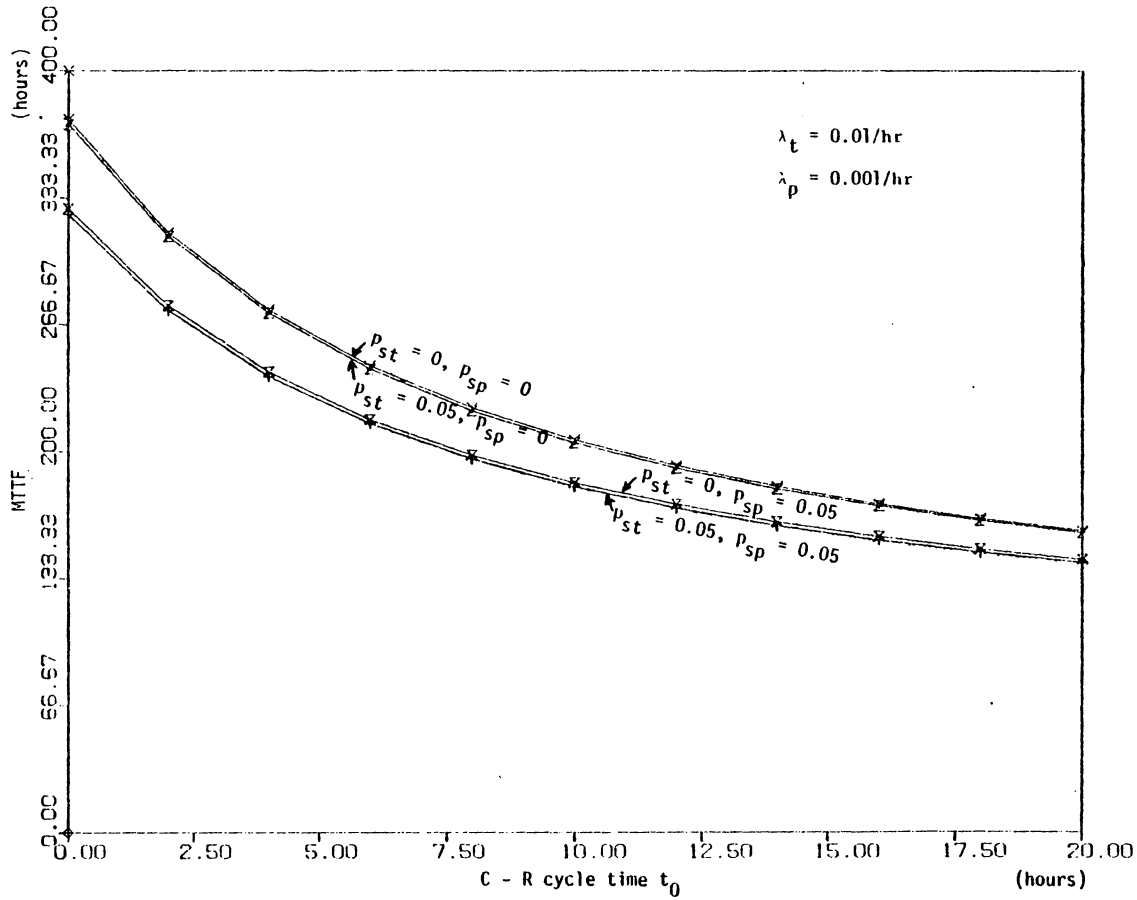

FIGURE 2.5: MTTF Plots Displaying the Effect of Initial CU Failure Probablities.

Chapter III

N REDUNDANT PSRR SYSTEMS

3.1 INTRODUCTION

In Chapter 2 triple redundant PSRR systems that employ a simple majority vote to decide the consensus CU state and the system output were presented and analyzed. While such systems offer a significant improvement in reliability over the simplex system, there may be applications that call for even greater reliability. This can be achieved by going to a higher level of redundancy. In this chapter, general N redundant PSRR systems are analyzed. To ensure that the system stays operational down to the last two correctly operating CU's, such systems employ a weighted plurality vote, rather than a majority vote, to decide the consensus CU state and the system output.

Definition 3.1: A weighted plurality of the N CU's is defined as follows. If two or more of the CU's are in agreement, then the result of the weighted plurality vote is the opinion of the largest number of agreeing CU's. For reasons discussed in the next paragraph, we do not consider the possibility of a tie between equal numbers of agreeing CU's. If all N CU's disagree, then the result is consistently taken to be the opinion of a particular CU called the default CU. This default CU is aribtrarily chosen at the time of system design to take advantage of the small possibility that is is still operating correctly when all N CU's disagree.

The weighted plurality vote is conducted on the CU states and outputs. In this study we neglect the possibility of two or more failed CU's being in the same state, i.e. having identical logic values for all corresponding memory elements. This is reasonable since a typical CU will have hundreds of thousands of memory elements; the possibility that two or more CU's fail in exactly the same way is certainly extremely small and can be neglected. The assumption that two or more CU's cannot be in the same state eliminates the possibility of the tie between equal numbers of agreeing CU's during restoration. We also neglect the possibility of obtaining identical outputs from two or more failed CU's. This latter probability clearly depends on the size of the output blocks being compared. For example, if individual bits are looked at, there is a significant chance (one half if the output of a failed CU is always completely random) that two failed CU's will have the same output at the same time. But if complete output words are compared, this probability is much smaller. In fact, the probability that two failed CU's have identical outputs can be made arbitrarily small by increasing the number of words in the output blocks being compared. We assume here that the weighted plurality vote is carried out on large enough blocks of output words so that there is no possibility of two or more failed CU's having the same erroneous output. This ensures that if the system is operational, the weighted plurality of the N CU outputs is the error free system output.

The weighted plurality voting defined above has very significant advantages over majority voting in general N redundant systems. It ensures that the system stays operational as long as there are two operational CU's in the system. Further, if the default CU is operational, it allows error free operation with a single good CU. Although the proability that the default CU is operational when all CU's disagree is small, it can still significantly impact the reliability of systems with relatively small amounts of redundancy (N small).

Restoration in general N redundant PSRR systems is somewhat more difficult to implement than the restoration in triple redundant systems, discussed in Chapter 2. We shall address this problem in Chapter 4, where we present an algorithm to implement a weighted plurality vote among the CU states. In the rest of this chapter we analyze the reliability of general N redundant PSRR systems assuming that the periodic restoration of the CU's to the weighted plurality state can be implemented as defined.

3.2 RELIABLITY MODEL

In this section we develop a reliability model for general N redundant PSRR systems. The approach that we follow is again based on the theory of Markov Chains and is broadly similar to the method used for analyzing triple redundant systems in Chapter Two. However, this time we relax the simplifying assumptions made in Chapter Two and consider finite restoration intervals with the possibility of failures occuring dur-

ing the restoration process. Note that this implies that a temporarily failed CU is ensured of recovery to an operational state only if it experiences no failures during the entire restoration interval and the default CU, or at least two other CU's, stay operational over the entire restoration interval.

To evaluate the reliability of PSRR systems, we first define a set of system states. Then we obtain transition probabilities for these system states over the computing and restoration intervals. These probabilities allow us to arrive at the state transition probability matrix for a complete C-R cycle. This matrix can be used, as in Chapter 2, to obtain system reliability over any number of C-R cycles.

Let the ordered set $\{CU_0, CU_1, \ldots, CU_{N-1}\}$ represent the N CU's in the redundant system, with CU_0 being the default CU.

Definition 3.2: The condition of a CU in the system is defined as follows:

Condition (CU_i) = 0 if CU_i is operational

= 1 if CU_i has temporarily failed

= 2 if CU_i has permanently failed

The status of the complete system is reflected in its primary state, q_p.

Definition 3.3: The primary state of the system at any time is given by

$$q_P = \sum_{i=0}^{N-1} [\text{Condition } (CU_i) \times 3^i]$$

Note that the primary state of the system is just the decimal equivalent of the base three number formed by concatenating in order the respective conditions of the N CU's, with CU_0 forming the least significant position. Since an N position base three number can represent 3^N values, there are 3^N primary states in an N redundant system.

We shall next obtain transition probabilities for these primary states over the computing and restoration invervals.

3.2.1 Computing Interval Probabilities

The computing interval primary state transition probability $p_{CP}(i,j)$ is the probability of a transition from primary state i to primary state j during the computing interval. Towards evaluating $p_{CP}(i,j)$, we first obtain the probabilities for transitions from one condition to another in individual CU's. Since the CU's do not interact during the computation interval, these transitions are mutually independent. The primary state transitions are a result of these mutually independent conditional transitions in individual CU's.

Let p_{TC} be the probability that a transient fault capable of causing a temporary failure in a CU occurs during the computing interval. Let $q_{TC} = (1-p_{TC})$ be the probability that such a transient fault does not occur. We shall assume that this probability is the same whether or not

the CU is operational, and that the transient fault always causes a temporary failure in an operational CU. Similarly, let p_{PC} be the probability of a permanent fault occuring in a CU during the computing interval. $q_{PC} = (1-p_{PC})$ is the probability that such a fault does not occur. It is also assumed that transient and permanent faults are mutually independent.

The probability of a transition of a CU during the computing interval from condition m to condition n, $p_{CI}(m,n)$ is given in Table 3.1. The table entries can be easily verified. For example, $p_{CI}(0,0)$ is the probability that an operational CU stays operational over the computing interval. Clearly this requires that no faults (temporary or permanent) occur, giving the probability $q_{TC} \times q_{PC}$. Similarly $p_{CI}(1,2)$ is the probability that a temporarily failed processor experiences permanent failure. This is just the probability of a permanent fault p_{PC}, because a permanently failed CU is unaffected by whether or not additional transient faults occur. Note that some transitions, such as $p_{CI}(1,0)$, are impossible because a CU cannot recover from any failure during the computing interval.

The computing interval primary state transition probability $p_{CP}(i,j)$ can now be evaluated by obtaining the product of the transition probabilities for individual CU's from their condition in state i to their condition in state j. This is possible because $p_{CP}(i,j)$ is just the joint probability of all these mutually independent transitions. The procedure is formalized in Algorithm 3.1.

Table 3.1: CU transitiion probabilites.

m	n	$P_{CI}(m,n)$	$P_{R11}(m,n)$	$P_{RI2}(m,n)$	$P_{RI31}(m,n)$	$P_{RI33}(m,n)$
0	0	$q_{TC} \times q_{PC}$	$q_{TR} \times q_{PR}$	0	undefined	0
0	1	$p_{TC} \times q_{PC}$	$p_{TR} \times q_{PR}$	q_{PR}	$p_{TR} \times q_{PR}$	undefined
0	2	p_{PC}	p_{PR}	p_{PR}	p_{PR}	p_{PR}
1	0	0	$q_{TR} \times q_{PR}$	0	0	0
1	1	q_{PC}	$p_{TR} \times q_{PR}$	q_{PR}	q_{PR}	q_{PR}
1	2	p_{PC}	p_{PR}	p_{PR}	p_{PR}	p_{PR}
2	0	0	0	0	0	0
2	1	0	0	0	0	0
2	2	1	1	1	1	1

Algorithm 3.1.

i) Set $p_{CP}(i,j) = 1$.

ii) Set $k = 0$.

iii) Let m = Condition (CU_k) in state i, n = Condition (CU_K) in state j. From Table 3.1, obtain $p_{CI}(m,n)$. Let $p_{CP}(i,j) = p_{CP}(i,j) \times p_{CI}(m,n), k=k+1$.

iv) If $k < N$, go to (iii).

v) Stop.

Using Algorithm 3.1 to evaluate individual state transition probabilities, the computing interval primary state transition matrix M_{CP} can be obtained.

We next consider the restoration interval probabilities.

3.2.2 Restoration Interval Probabilities

The restoration interval primary state transition probability $p_{PR}(i,j)$, is the probability of a transition from primary state i to primary state j during the restoration interval.

Let p_{TR} be the probability that a transient fault capable of causing a temporary failure in a CU occurs during the restoration interval. $q_{TR} = (1-p_{TR})$ is the probability that such a transient fault does not occur. Let p_{PR} be the probability that a permanent fault occurs in a CU during the restoration period. $q_{PR} = (1-p_{PR})$ is the probability that a permanent fault does not occur.

In evaluating $p_{PR}(i,j)$ we have to consider three separate cases.

Case 1: The system is operation throughout the restoration interval.

This requires that the default CU or at least two other CU's stay operational throughout the restoration interval (i.e. are operational in both primary states i and j). In this case temporarily failed processors will be restored, provided they do not experience additional failures during the restoration interval. The transition probabilities from condition m to condition n for individual CU's, $p_{RI1}(m,n)$ are easily arrived at, and are also listed in Table 3.1. Again because the system is operational throughout the restoration interval, transitions in individual CU's are mutually independent. Therefore, the primary state transition probability $p_{PR}(i,j)$ is the product of the individual CU transition probabilities from their condition in state i to their condition in state j, and can be evaluated by using Table 3.1 and Algorithm 3.2.

Algorithm 3.2.

i) Set $p_{RP}(i,j) = 1$.

ii) Set $k = 0$.

iii) Let m = Condition (CU_k) in state i, n = Condition (CU_k) in state j. From Table 3.1, obtain $p_{RI1}(m,n)$. Set $p_{RP}(i,j) = p_{RP}(i,j) \times p_{RI1}(m,n)$. Set $k = k+1$.

iv) If $k < N$, go to (iii).

v) Stop.

Case 2: The system has failed before the start of the restoration interval. This requires that the initial state i correspond to a failed system (i.e. a failed default CU and no more than one operating CU). In this case not only will temporarily failed CU's not be restored, but any correctly operating CU will also experience temporary failure because the result of the weighted plurality vote will be erroneous. The conditional transition probabilities $p_{RI2}(m,n)$ for this case are given in Table 3.1, and can be easily verified. Once again the primary state transition probability $p_{RP}(i,j)$ is the product of the individual CU transition probability and can be evaluated using table 1 and a procedure identical to Algorithm 3.2.

Case 3: The system fails during the restoration interval. This requires that state i correspond to an operational system and state j to a failed system. Transition probabilities for CU's from an operational condition to a temporarily failed condition are no longer mutually independent in this case. This is because failures in the CU's may build up in many different ways to the point where the system fails. Any CU that is still operational will then be forced to a temporarily failed state due to erroneous results from the weighted plurality vote. To obtain the primary state transition probabilities for this case, we partition the CU's in the system into three disjoint sets such that transitions in the condition of CU's in any one set are independent of transitions in other sets.

Set 1 contains only the default CU. Note that the conditional transition probabilities for the default CU are independent of the condition of the other CU's. This is because the default CU can only fail because of a fault. There is no possibility of it being forced to a temporarily failed state by an erroneous result of the weighted plurality vote, because the result of the vote (by definition) will always be correct as long as the default CU is operational. The conditional transition probabilities for the default CU in Case 3 (from condition m to n) are given in table 1 and can be verified.

Set 2 contains all CU's that go from an operational condition in state i to a temporarily failed condition in state j. Let NEWTEMP be the number of CU's in this set. Since the result of the plurality vote is error free (and the system operational) as long as two or more CU's operate correctly, only one CU at most can be forced to temporary failure during restoration by an erroneous result of the weighted plurality vote. Thus at least (NEWTEMP-1) and possibly all NEWTEMP CU's experience transient faults during the restoration interval. In addition, the NEWTEMP CU's in set 2 do not experience any permanent faults. The probability of NEWTEMP transitions from an operational to a temporarily failed condition in this set is therefore given by

$$P_{NEWTEMP} = [(p_{TR})^{NEWTEMP} + (NEWTEMP) \times (p_{TR})^{(NEWTEMP-1)} \times q_{TR}] \times q_{PR}$$

Set 3 contains all the remaining system CU's. Since this set does not contain CU's that experience temporary failure over the restoration in-

terval, conditional transition in CU's in this set are mutually independent. Transition probabilities for individual CU's are listed in Table 3.1 under $p_{RI33}(m,n)$. The joint transition probability for the CU's in set 3 is just the product of the individual transition probabilities and can be obtained with the help of Table 3.1.

To obtain the restoration interval transition probability $p_{RP}(i,j)$ for primary states i and j that satisfy case 3 requirements, (i.e. i corresponds to an operational system and j to a failed system) we partition the CU's into three sets and obtain the joint conditional transition probability for each set as explained. Since the probabilities for these three sets are mutually independent, their product gives $p_{RP}(i,j)$. This procedure is formalized in Algorithm 3.3.

Algorithm 3.3

i) Let m = Condition (CU_0) in state i, n = Condition (CU_0) in state j.

ii) From Table 3.1 obtain $p_{RP31}(m,n)$. Let $p_{RP}(i,j) = p_{RP31}(m,n)$.

iii) Set NEWTEMP = 0, and k = 1.

iv) let m = Condition (CU_k) in state i, n = Condition (CU_k) in state j.

v) If m = 0 and n = 1, then let NEWTEMP = NEWTEMP + 1 and go to (vii).

vi) From Table 3.1 obtain $p_{RP33}(m,n)$. Let $p_{RP}(i,j) = p_{RP}(i,j) \times p_{RP13}(m,n)$.

vii) Let $k = k + 1$

viii) If $k < N$ goto (iv).

ix) Set $p_{NEWTEMP} = [(p_{TR})^{NEWTEMP} + NEWTEMP \times (p_{TR})^{(NEWTEMP-1)} \times q_{TR}] \times q_{PR}$

x) Stop.

Using Algorithm 3.2 or 3.3, as appropriate, the restoration interval primary state transition probabilities can be evaluated and the transition matrix M_{RP} formed.

By the theory of Markov Chains, the product matrix

$$T_p = M_{Cp} \times M_{RP} \tag{3.2-1}$$

is also a transition probability matrix and gives the primary state tarnsition probabilities for a complete C-R cycle. Transition probabilities over n C-R cycle times can be obtained by evaluating $(T_p)^n$. The reliability of the system over a mission time of n C-R cycles can be evaluated by summing up the transition probabilies to all operational system states from primary state 0 (all CU's operational) in $(T_p)^n$.

However, there are computational difficulties in obtaining reliability this way because of the size of the transition matrices. Recall that an N-redundant system has 3^N primary states. The transition probability matrices are therefore $3^N \times 3^N$. For N = 5 there are already almost 60,000 elements in T_p and the number increases exponentially with N. Evaluating $(T_p)^n$ is impractical.

To get around this problem, we reduce the number of system states by defining some equivalence classes. This yields considerably smaller state transition matrices and makes it possible to compute the system reliability.

3.2.3 State Reduction

The set of primary system states is first partitioned into operational primary states in which the system is operational and failed primary states in which the system has failed. Recall that the system is operational if the default CU is operational or at least two other CU's are operational.

Table 3.2 shows the 27 primary states and the corresponding CU conditions for a triple redundant system (N=3). CU_0 is the default CU. Notice that the set of operational primary states is

$$\{0,1,2,3,6,9,12,15,18,21,24\}$$

The set of operational primary states can be further partitioned by the following equivalence relation: Two operational primary states are said to be equivalent if the default CU in the two states has the same condition and the two states have equal numbers of CU's in condition 0, 1 and 2 respectively. The subsets of primary states formed by this partition along with the subset containing failed primary states define the secondary state set of the system.

The system is said to be in secondary state q_S if the primary state of the system is an element of the subset of primary states defining secondary state q_S.

Table 3.2: Primary States in a Triple Redundant System.

Primary State	Condition (CU_2)	Condition (CU_1)	Condition (CU_0)
0	0	0	0
1	0	0	1
2	0	0	2
3	0	1	0
4	0	1	1
5	0	1	2
6	0	2	0
7	0	2	1
8	0	2	2
9	1	0	0
10	1	0	1
11	1	0	2
12	1	1	0
13	1	1	1
14	1	1	2
15	1	2	0
16	1	2	1
17	1	2	2
18	2	0	0
19	2	0	1
20	2	0	2
21	2	1	0
22	2	1	1
23	2	1	2
24	2	2	0
25	2	2	1
26	2	2	2

Primary state q_P is said to be in secondary state q_S if q_P is an element of the subset of primary states defining secondary state q_S.

The secondary states can be assigned consecutive identifying integers 1 through $(3N^2/2 - 5N/2 + 3)$ based on the defining partition of primary states as explained below.

Consider the set of CU's $\{CU_1, CU_2, \ldots, CU_{N-1}\}$, i.e. the set of system CU's not including the default CU, CU_0. Let NTEMP the number of CU's with Condition = 1 and NPERM the number of CU's with Condition =2 in this set. Let q_S be the secondary state of the system.

Definition 3.4: If the system is in a failed primary state,

$$q_S = 3N^2/2 - 5N/2 + 3$$

if condition of default CU = 0, then

$$q_S = \frac{N(N+1)}{2} - \frac{(N-NPERM)(N-NPERM+1)}{2} + NTEMP+1,$$

if Condition of default CU = 1, then

$$q_S = \frac{N(N+1)}{2} + \frac{(N-1)(N-2)}{2} - \frac{(N-NPERM-1)(N-NPERM-2)}{2} + NTEMP+1,$$

finally if Condition of default CU = 2, then

$$q_S = \frac{N(N+1)}{2} + (N-1)(N-2) - \frac{(N-NPERM-1)(N-NPERM-2)}{2} + NTEMP+1.$$

Note that a failure free system is assigned secondary state 1 and a failed system state $3N^2/2 - 5N/2 + 3$. The secondary state numbering scheme can be verified in Table 3.3 which shows the 9 secondary states for a triple redundant system along with the defining subsets of primary states.

In general, the number of secondary states in an N redundant system is $3N^2/2 - 5N/2 + 3$. This expression yields 17 for N=4 and 28 for N=5. The second order polynomial growth of this function with N is much slower than the exponential growth of the 3^N primary states, resulting in considerably smaller state transition matrices. This makes it possible to evaluate system reliability, once the secondary state transition probabilities are known.

Transition probabilities for the secondary states can be obtained from the primary state transition probabilities. Clearly the transition probability from the secondary state representing a failed system to all other secondary states except itself will be 0. The probability that the system remains in the failed state is 1. When the system is in an operational secondary state i, its primary state belongs to the set of primary states defining secondary state i. Since all primary states in an operational secondary state are equivalent, the transition probabilitiy from secondary state i to secondary state j can be evaluated by taking an arbitrary primary state p from the set of primary states in secondary state i and summing up all the transition probabilities from p to primary states in secondary state j.

Table 3.3: Reduced States in a Triple Redundant System.

Secondary State q_S	Defining Partition			Defining Primary State Set
	Condition of Default CU	# Failures PERM	TEMP	
1	0	0	0	{0}
2	0	0	1	{3,9}
3	0	0	2	{12}
4	0	1	0	{6,18}
5	0	1	1	{15,21}
6	0	2	0	{24}
7	1	0	0	{1}
8	2	0	0	{2}
9	Failed System			{4,5,7,8,10,11,13, 14,16,17,19,20, 22,23,25,26}

Formally let $S_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_n}\}$ and $S_j = \{s_{j_1}, s_{j_2}, \ldots, s_{j_m}\}$ be the sets of primary states defining the secondary states i and j respectively. Let s_{i_A} be any arbitrary element of S_i. Based on the above discussion we state the following theorems.

Theorem 3.1: The computing interval secondary state transition probability $p_{CS}(i,j)$ is given by

$$p_{CS}(i,j) = \sum_{k=1}^{m} p_{CP}(s_{i_A}, s_{j_k}) \quad \text{for } i < 3N^2/2 - 5N/2 + 3$$

$$= 1 \qquad \text{for } i = j = 3N^2/2 - 5N/2 + 3$$

$$= 0 \qquad \text{otherwise.}$$

Theorem 3.2: The restoration interval secondary state transition probability $p_{RS}(i,j)$ is given by

$$p_{RS}(i,j) = \sum_{k=1}^{m} p_{RP}(s_{i_A}, s_{j_k}) \quad \text{for } i < 3N^2/2 - 5N/2 + 3$$

$$= 1 \qquad \text{for } i = j = 3N^2/2 - 5N/2 + 3$$

$$= 0 \qquad \text{otherwise.}$$

The computing interval secondary state transition matrix M_{CS} and the restoration interval secondary state transition matrix M_{RS} can now be constructed. By the theory of Markov Chains,

$$T_S = M_{CS} \times M_{RS} \qquad (3.2\text{-}2)$$

is also a transition matrix which gives the secondary state transition matrix probabilities over a complete C-R cycle. Transition probabilities over n C-R cycles can be obtained by evaluating $(T_S)^n$.

The reliability of the system to n C-R cycle times is the probability that a failure free system is still operational at the end of the n C-R cycles. Since secondary state 1 represents a failure free system and secondary state $(3N^2/2 - 5N/2 + 3)$ a failed system, the $\{1,(3N^2/2 - 5N/2 + 3)\}$ entry in $(T_S)^n$ gives the probability, p_f of transition from a failure free system to a failed system over n C-R cycles. $(1-p_f)$ is the probability that this transition does not take place i.e. the system is still operational. Thus, $(1-p_f)$ is the reliability of the system to n C-R cycle times.

3.2.4 Discussion

The reliability model for general N redundant PSRR systems with weighted plurality voting, developed in this section, has been programmed on an IBM system 370/158. As input data, the model requires probability values for temporary and permanent faults occurring during the computation and restoration intervals. For the purpose of exercising the reliability model we again assume constant failure rates over both the computing and restoration intervals. As in Chapter 2, the

transient failure rate λ_t is taken to be 0.01 per hour and the permanent failure rate λ_p is taken to be 0.001 per hour. Applying these failure rates to the classical exponential distribution for reliability $R(t) = e^{-\lambda t}$ (where λ is the failure rate) results in the following failure probabilities for each CU:

$$p_{TC} = (1 - e^{-0.01t_C}) \tag{3.2-3}$$

$$p_{PC} = (1 - e^{-0.001t_C}) \tag{3.2-4}$$

$$p_{TR} = (1 - e^{-0.01t_R}) \tag{3.2-5}$$

$$p_{RP} = (1 - e^{-0.001t_R}) \tag{3.2-6}$$

In the above expressions, t_C is the length of the computing interval in hours t_R the length of the restoration interval in hours. t_R depends on the time required to carry out restoration and was taken to be 0.005 hours (18 seconds). The C-R cycle time t_o is the sum of the t_C and t_R. t_C is taken to be 0.995 so as to make t_o = 1 hour.

Figure 3.1 shows reliability plots for PSRR systems with weighted plurality voting and different levels of redundancy N. Figure 3.2 shows the same plots for short missions. The plots show a very substantial improvement in reliability as the number of CU's in the system is increased. Note in Figure 3.1 that in PSRR systems reliability improves with increased redundancy for all values of mission time. Therefore, the mean time to failure (MTTF) for the system also

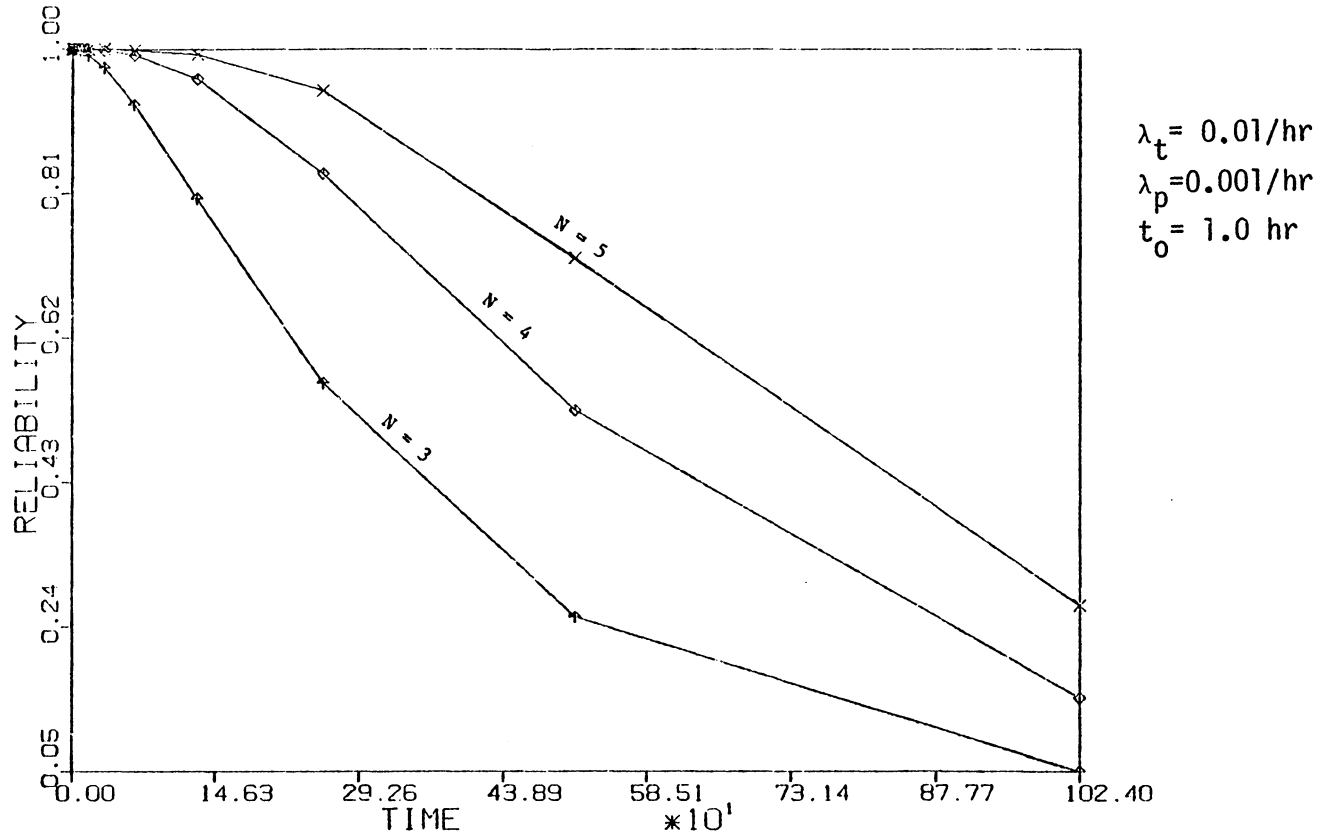

FIGURE 3.1: Reliability Plots for Varying Redundancy Level N.

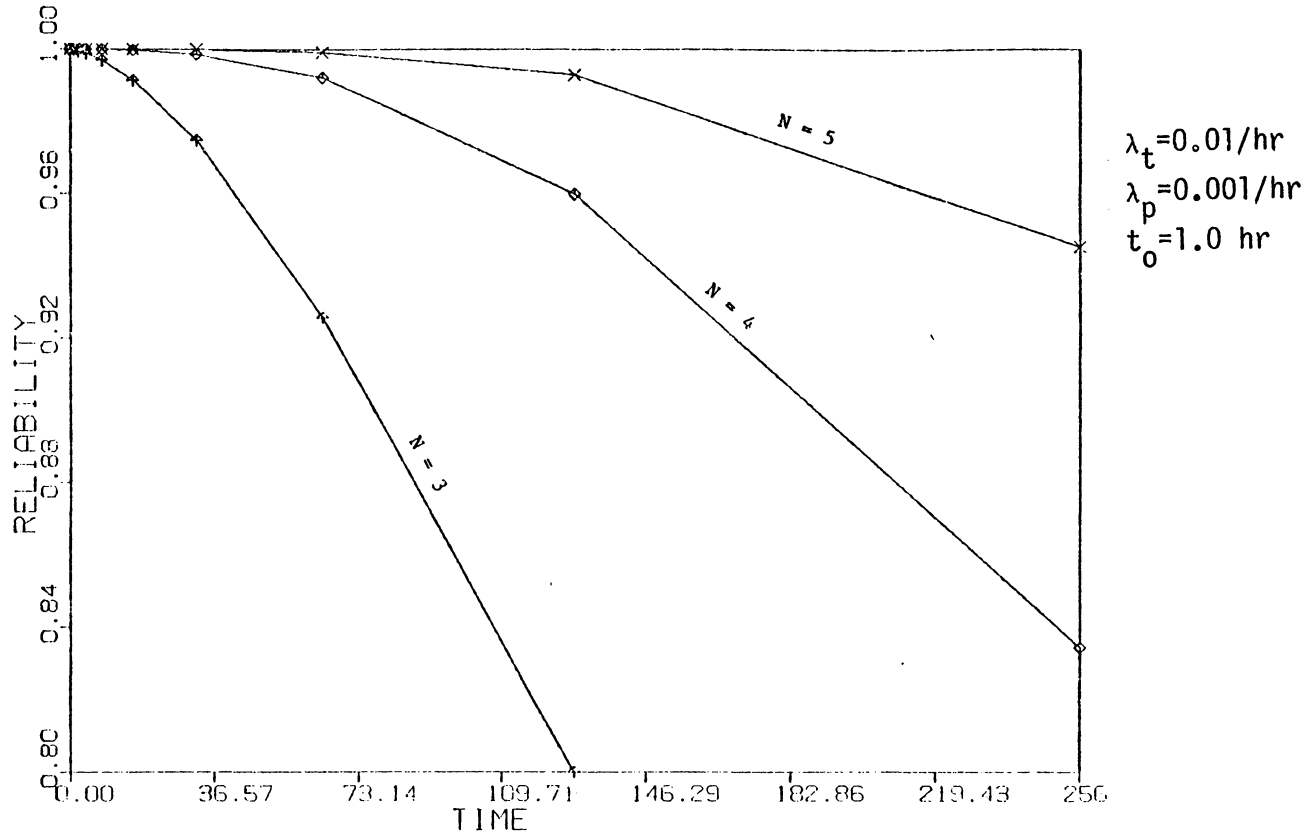

FIGURE 3.2: Reliability Plots for Varying Redundancy Level N and Short Mission Times.

increases with redundancy. This is a marked improvement on the TMR and NMR schemes where increased redundancy improves reliability for short missions, but increases the probability of system failure over longer periods, thereby resulting in reduced MTTF with increasing levels of redundancy. We analyze the MTTF of general N redundant PSRR systems in the next section.

Also displayed in Figures 3.1 and 3.2 are reliability plots for the triple redundant system with simple majority voting described in Chapter 2. The difference between this system and the N=3 system with plurality voting is that no default CU is defined in the former. As a result, the system with simple majority voting always fails when two CU's fail, while there is a possibility that the weighted plurality voting system continues to operate correctly with only one good CU. Notice in the figures that defining a default CU results in a significant improvement in system reliability for triple redundant PSRR systems.

It should be noted that the plots in Figures 3.1 and 3.2 were obtained based on CU failure probabilities to be given by equations (3.2-3) through (3.2-6) and a fixed restoration interval $t_R = 0.005$ hours. These values did not change with the level of redundancy N. However, in the practical PSRR systems, it is expected that both the CU failure probabilities, as well as the time required by the restoration process will increase with increased N. The improvement in system reliability with redundancy will, therefore, not be quite as substantial as indicated by Figures 3.1 and 3.2. These issues are addressed in detail in Chapter 5.

3.3 INITIAL FAILURE PROBABILITIES

The possibility of CU failures before the start of the mission can also be incorporated into the reliability model that has been developed for general N redundant PSRR systems. Let p_{st} be the probability that a CU has temporarily failed before the start of the mission, and p_{sp} the probability that it has permanently failed before the start of the mission. Further, let w represent the number of secondary states in the system, i.e. $w = 3N^2/2 - 5N/2 + 3$. We shall first obtain the starting state probability vector for the system

$$\pi = [p_{S1}\ p_{S2}, \ldots, p_{Sw}]. \tag{3.3-1}$$

Note that π is a w component row vector and the ith entry in π gives the probability that the system is in secondary state i at the start of the mission. We can evaluate the w components of π by using Algorithm 3.4.

Algorithm 3.4

i) Set NPERM = -1

ii) Let NPERM = NPERM + 1. Set NTEMP = -1

iii) Let NTEMP = NTEMP + 1

iv) Let NFAIL = NTEMP + NPERM,

$$q_S = \frac{N(N+1)}{2} - \frac{(N-NPERM)(N-NPERM+1)}{2} + NTEMP + 1,$$

and

$$p_{Sq_S} = (p_{st})^{NTEMP} \times (p_{sp})^{NPERM} \times \{(1-p_{st})\times(1-p_{sp})\}^{(N-NFAIL)}$$

v) If NFAIL <N-1 go to (iii).

vi) If NPERM <N-1 go to (ii).

vii) Set NPERM = -1.

viii) Let NPERM = NPERM+1. Set NTEMP = -1.

ix) Let NTEMP = NTEMP+1.

x) Let NFAIL = NTEMP + NPERM,

$$q_S = \frac{N(N+1)}{2} + \frac{(N-1)(N-2)}{2} - \frac{(N-NPERM-1)(N-NPERM-2)}{2} + NTEMP+1$$

$$p_{Sq_S} = (p_{st})^{(NTEMP+1)} \times (p_{sp})^{NPERM}$$

$$\times \{(1-p_{st})\times(1-p_{sp})\}^{(N-1-NFAIL)}$$

xi) If NFAIL <N-3 go to (ix).

xii) If NPERM <N-3 go to (viii).

xiii) Set NPERM = -1.

xiv) Let NPERM = NPERM + 1. Set NTEMP = -1.

xv) Let NTEMP = NTEMP + 1.

xvi) Let NFAIL = NTEMP + NPERM,

$$q_S = \frac{N(N+1)}{2} + (N-1)(N-2) - \frac{(N-NPERM-1)(N-NPERM-2)}{2} + NTEMP+1,$$

$$p_{Sq_S} = (p_{st})^{NTEMP} \times (p_{sp})^{(NPERM+1)}$$

$$\times \{(1-p_{st})(1-p_{sp})\}^{(N-1-NFAIL)}.$$

xvii) If NFAIL <N-3 go to (xv).

xviii) If NPERM <N-3 go to (xiv).

xix) Set SUM = 0, k=1

xx) Let SUM = SUM + p_{Sk}.

xxi) Let k = k+1. If k < w go to (xx).

xxii) Let p_{Sw} = 1 - SUM.

xxiii) Stop.

Theorem 3.3: Algorithm 3.4 correctly generates all w components of the starting state probability vector π

Proof: Notice that Algorithm 3.4 is set up to generate the components of the starting state probability vector in the same order as the states were defined by Definition 3.4. The first 6 steps of the algorithm generate starting probabilities for secondary states that have operational default CU's. Each such state has different numbers of the remaining N-1 CU's in the operational, temporarily failed, and permanently failed states. The starting probabilities for all these states are obtained by repeating step (iv) for all combinations of conditions in the N-1 CU's. Step (iv) finds the secondary state for any such combination using Definition 3.4, and then evaluates the probability of the system starting in that state.

Steps (vii) through (xii) follow a similar approach to obtain starting probabilities for operational secondary states containing a temporarily failed default CU, i.e. Condition (CU_o) = 1. Note that the default CU is not included in the count kept by the variables NTEMP and NFAIL; its probability of failure is accounted for independently in the starting state probability expression in step (x). Since the temporarily failed

default CU is not included in NFAIL and at least 2 CU's must stay operational to ensure an operational system, we must have NFAIL ≤ N-3. Therefore, to obtain all operational secondary states with Condition $(CU_o) = 1$, we repeat step (x) for all combinations of CU failures such at NFAIL ≤ N-3.

A very similar procedure is used in steps (xiii) through (xviii) to obtain starting probabilities for secondary states with a permanently failed default CU. Finally, the starting probability for the failed system state is obtained by summing up all the other starting probabilities and subtracting the sum from unit.

It is clear from this discussion that Algorithm 3.4, correctly generates all w components of the starting state probability vector π.

Once the starting state proability vector is obtained, system reliablity over n C-R cycles can be evaluated, initial failure probabilities taken into account. Recall that $(T_S)^n$ is a wxw matrix such that each element in the matrix, $p_{ij}^{(n)}$, gives the transition probability from secondary state i to secondary state j over n C-R cycles. Therefore,

$$\pi \times (T_S)^n = p_{S1}p_{1w}^{(n)} + p_{S2}p_{2w}^{(n)} + \ldots p_{Sw}p_{ww}^{(n)} \qquad (3.3\text{-}2)$$

Note that each term on the right hand side of equation (3.3-2) is just the probability that the system starts in some state and then goes from that state to the failed system state over n C-R cycles. Therefore, equation (3.3-2) gives the probability that a system with starting state probability vector π fails before nt_o. Thus,

$$R(nt_o) = 1 - \pi(T_S)^n \qquad (3.3\text{-}3)$$

Theorem 4.4: The reliability of a PSRR system over n C-R cycles is given by $R(nt_o) = 1 - \pi(T_S)^n$, where π is the starting state probability vector.

Note that Theorem 4.4 gives the correct system reliablity even when there is no possibility of initial CU failures. In such a case

$$\pi = [1\ 0\ 0\ \ldots\ 0] \qquad (3.3\text{-}4)$$

since the system is always initially in state 1.

The effect of the possibility of temporary and permanent failures on system reliability was discussed at length for triple redundant PSRR systems in chapter 2. Since the same considerations hold for general N redundant systems, we summarize the discussion here by restating the conclusion that the probability of initial permanent CU failures has a much more significant impact on system reliability than the probability of initial temporary failures.

3.4 MEAN TIME TO FAILURE CALCULATION

The MTTF for general N redundant systems can also be calculated by using the secondary state transition matrix T_S. We again assume that T_S is wxw with entry p_{ij} giving the transition probability from state i to state j.

Let μ_{ij} be the average number of C-R cycles taken by a system starting in state i to go to state j. We first obtain an expression for μ_{1w} by considering a system one C-R cycle after starting in state 1.

At this point in time, the system is in state 1, 2 or (in general) k with probability p_{11}, p_{xii} and p_{1k} respectively. Therefore, from this instant, the average number of C-R cycles to the failed system state w is given by $p_{11}\mu_{1w} + p_{12}\mu_{2w} \cdots + p_{1k}\mu_{kw} \cdots + p_{1w}\mu_{ww}$. Since the system was in state 1 one C-R cycle before the instant under consideration

$$\mu_{1w} = 1 + p_{11}\mu_{1w} + p_{12}\mu_{2w} \cdots + p_{1w}\mu_{ww}. \qquad (3.4\text{-}1)$$

It can be similarly seen that

$$\mu_{2w} = 1 + p_{21}\mu_{1w} + p_{22}\mu_{2w} \cdots + p_{2w}\mu_{ww} \qquad (3.4\text{-}2)$$

and, for an arbitrary k

$$\mu_{kw} = 1 + p_{k1}\mu_{1w} + p_{k2}\mu_{2w} \cdots + p_{kw}\mu_{ww}. \qquad (3.4\text{-}3)$$

Noting that $\mu_{ww} = 0$, setting $w - 1 = v$, and rearranging terms in equations (3.4-1) through (3.4-3) we get

$$(1 - p_{11})\mu_{1w} \quad -p_{12}\mu_{2w} \cdots \quad - p_{1v}\mu_{vw} = 1 \qquad (3.4\text{-}4)$$

$$- p_{21}\mu_{1w} + (1 - p_{22})\mu_{2w} \cdots \quad - p_{2v}\mu_{vw} = 1 \qquad (3.4\text{-}5)$$

$$- p_{k1}\mu_{1w} \quad -p_{k2}\mu_{2w} \cdots + (1 - p_{kk})\mu_{kw} \cdots - p_{kv}\mu_{vw} = 1 \qquad (3.4\text{-}5)$$

Using equation (3.4-6) and k ranging from 1 to v, we can obtain a set of v equations in the v unknowns μ_{1w} through μ_{vw}. This system of equations can be represented in matrix form as

$$(I\text{-}Q)M = \xi \qquad (3.4\text{-}7)$$

where

$$Q = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1v} \\ p_{21} & p_{22} & \cdots & p_{2v} \\ \vdots & & & \\ p_{v1} & \cdots & & p_{vv} \end{bmatrix}, \tag{3.4-8}$$

$$M = \begin{bmatrix} \mu_{1w} \\ \mu_{2w} \\ \vdots \\ \mu_{vw} \end{bmatrix} \tag{3.4-9}$$

I is the vxv identity matrix and ξ is a v component column vector, having all components equal to 1, i.e.

$$\xi = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}. \tag{3.4-10}$$

Equation (3.4-7) can be solved to obtain the v component vector M, whose components give the average number of C-R cycles that it takes for the system to fail from any starting state. Notice that Q is the matrix T_S with the row and column associated with the failed system state removed. It has been shown, in Markov Chain theory (See Kemeny and Snell [24], Theorem 3.2.1) that for such a matrix, $(I-Q)^{-1}$ exists. Therefore the solution to equation (3.4-7) is

$$M = (I - Q)^{-1} \xi \qquad (3.4\text{-}11)$$

The first component of M, μ_{1w}, is the number of C-R cycles that it takes, on the average, for a failure free system to fail. The MTTF of an initially failure-free system is, therefore, given by $\mu_{1w}t_o$. More generally, if π is the starting state probability vector for the system and π' consists of the first v components of π, then it can be seen that

$$MTTF = \pi' M t_o \qquad (3.4\text{-}12)$$

Note again that it was implicitly assumed in the derivation of equations (3.4-1) through (3.4-3) that any operational state in the system takes more than one C-R cycle to fail. Therefore, equation (3.4-7) is only valid for C-R cycle times that are very short as compared to the time to failure for the CU's. This clearly requires $t_o << 1/\lambda t, 1/\lambda p$.

Theorem 4.5: The MTTF for general N redundant PSRR systems with $t_o << 1/\lambda t, 1/\lambda p$ is given by $\pi' M t_o$.

Figure 3.3 shows MTTF for PSRR systems for varying levels of redundancy and C-R cycle times. The CU's in the systems are assumed to have constant failure rates, $\lambda_t = 0.01$ per hour and $\lambda_p = 0.001$ per hour. The restoration interval t_R is taken to be 0.005 hours in all the plots. As expected from observing the reliability plots in Figure 3.1, system MTTF is seen to improve very significantly with both increased level of redundancy and reduced C-R cycle time. It should again be noted, however, that in Figure 3.3 it is assumed that the restoration

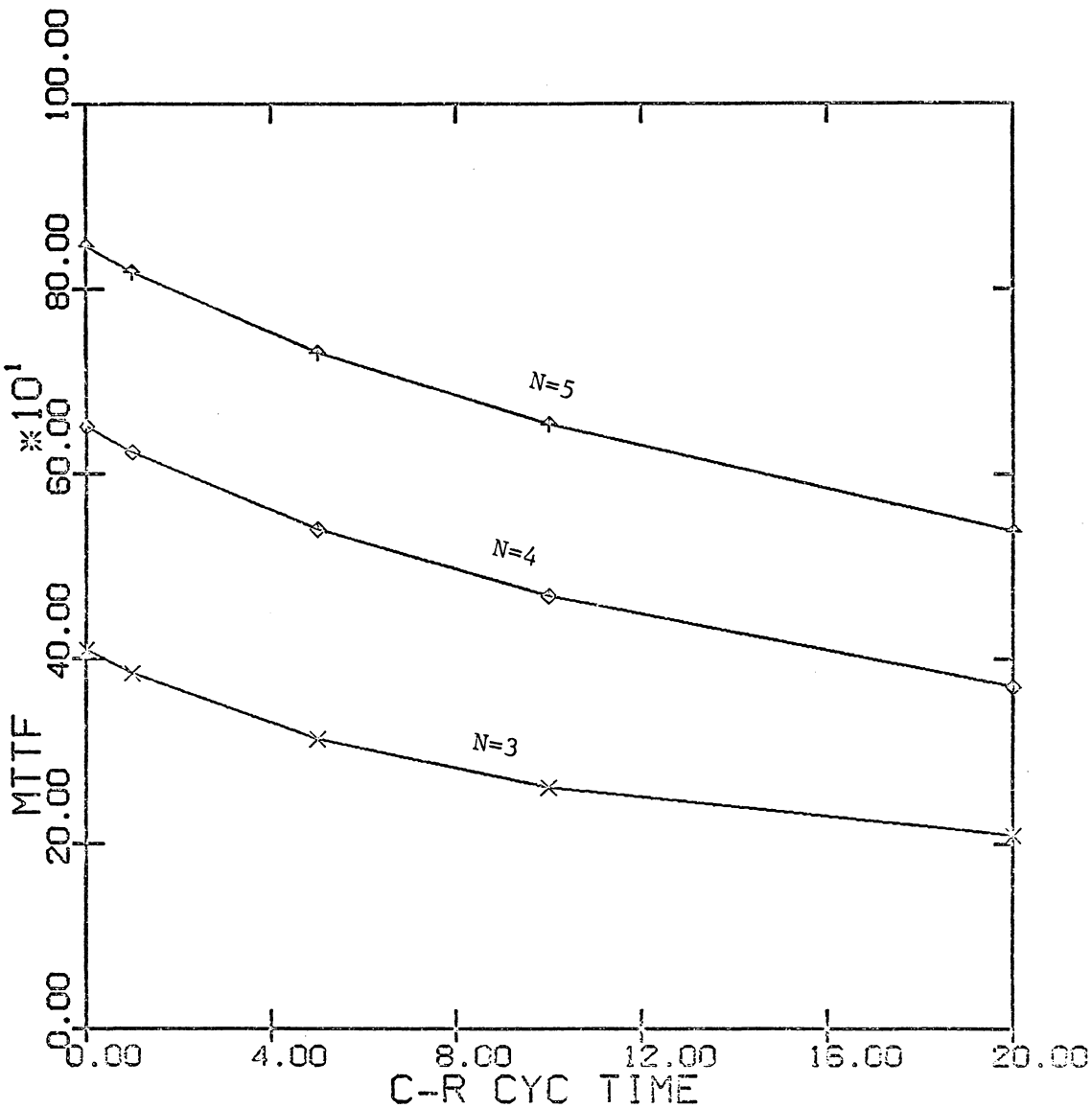

FIGURE 3.3: MTTF versus C-R Cycle Time t_0.

interval and CU failure rates do not increase with increased redundancy. We shall see in Chapter 5 that this may not be a valid assumption for practical systems and that, in practice, the improvement in system MTTF with redundancy may not be as large as indicated by Figure 3.3.

Chapter IV

THE RESTORATION PROCESS

4.1 INTRODUCTION

In Chapter Three we analyzed the reliability of general N redundant PSRR systems. It was shown that such systems are capable of highly reliable operation, provided that it is possible to periodically restore the CU's in the system to a mutual consensus state. This consensus is defined by a weighted plurality vote on the CU states. In this chapter we discuss implementation of the restoration process in PSRR systems.

As described in Chapter One, the periodic restoration process is initiated simultaneously in all the system CU's by a non-maskable hardware interrupt from a fault tolerant clocking circuit. The interrupt causes control to be transfered to the restoration program stored in ROM. The restoration program first saves the CU state (at the time of the interrupt) by storing the contents of all registers in the processor, and any other subsystems in the CU, at predetermined locations in the memory. It then conducts a weighted plurality vote among the R/W (read/write) memories of the CU's, thereby achieving a weighted plurality vote on the CU states at the time of the interrupt. The registers in each CU are finally reloaded from its memory, and the restoration process is complete with control being transfered back to the interrupted program. Thus, the restoration process achieves a weighted plurality vote on the CU states at the time of the interrupt.

In the remainder of this chapter we limit our attention to that part of the restoration program which carries out the weighted plurality vote on the CU memories. It is clear that implementation of the rest of the restoration process is quite straightforward.

The reliability model of Chapter Three requires that the consensus CU state be the operational CU state as long as the default CU or two other CU's in the system stay operational. This cannot be ensured by obtaining the consensus state by a weighted plurality vote conducted independently on each word in the CU memories. In such a vote, two or more failed CU's that have identical erroneous words in any memory location have the potential to out vote an operational default CU, or even two operational CU's. While the probability that two CU's will acquire identical erroneous words directly due to random failures in the memory is small, such a situation can result from the inability on the part of failed CU's to update words in their memories, thereby retaining identical but erroneous contents in one or more locations from some earlier time. A weighted plurality vote conducted with the entire CU memory taken as a unit ensures restoration to the desired state unless failed CU's have identical words in all corresponding memory locations. With thousands of memory words expected in typical CU's, such failures are extremely unlikely.

4.2 A RESTORATION ALGORITHM

An algorithm to implement the weighted plurality vote on the CU states cannot be defined in complete detail without specifying some characteristics of the communication links connecting the CU's, and also the communication protocal employed. Therefore, in this section we present a restoration algorithm in general terms, without specifying details about its implementation. Later in this chapter, we propose a viable communication structure to support the restoration process in PSRR systems, and then present more detailed algorithms for implementing restoration in such systems.

To conduct the weighted plurality vote on the CU states, the processor in each CU must compare the contents of each memory location in itself and all the other CU's. Further, since the vote is conducted on the entire CU state taken as a unit, no decisions on the word to be stored in a memory location as a result of the vote can be made until the entire memory is compared. The restoration process must, therefore, involve two complete passes over the CU memories.

The proposed algorithm is designed to be executed simultaneously in each of the system CU's during the restoration interval. Any CU that executes it correctly is assured of proper restoration. By the end of the first pass, each CU generates a partition of the system CU's. This partition is based on a comparision of memory words such that any two CU's in the same block of the partition agree on the contents of all memory locations, while CU's in different blocks disagree on the contents

of at least one memory location. During the second pass, the comparision of memory words with possible further partitioning of the CU blocks is again carried out, so as to handle failures since the first pass. But this time the memory word being compared is updated, based on the new partition, to that of the largest block of agreeing CU's. If all blocks in the CU partition contain only one element, the memory word is replaced by that from the default CU.

Algorithm 4.1

i) Define an initial partition of system CU's such that all CU's are in the same block of the partition. Define a set of definitely failed CU's and set it to be initially empty. Set ADDRESS=0.

ii) Using the communication links, obtain the word stored in location ADDRESS in all CU's. If a CU does not co-operate in this communication, add it to the set of definitely failed CU's.

iii) Compare each pair of words obtained in the previous step. If two words disagree, ensure that the CU's that they belong to are in different blocks in the partition of system CU's. This may require a new partition with more blocks. Also ensure that each CU in the set of definitely failed CU's is in a block by itself in the partition.

iv) If ADDRESS is the last address in the CU memory, go to step (v) (pass 2). Otherwise increment ADDRESS by 1 and go back to step (ii).

v) Set ADDRESS=0.

vi) Repeat step (ii).

vii) Repeat step (iii).

viii) Find the largest block in the partition of system CU's. Obtain the word in memory location ADDRESS from any CU in this block. If all blocks in the partition contain only a single CU, obtain the contents of memory location ADDRESS in the default CU. Store this word in location ADDRESS.

ix) If ADDRESS is less than the last address in the CU memory, increment ADDRESS by 1 and go back to step (vi).

x) Stop.

The above algorithm is self explanatory and will not be discussed futher.

In the next section we define a communication structure to support the restoration process in PSRR systems, and give a detailed algorithm for its implementataion. The proposed architecture thus provides a viable means of implementing PSRR systems.

4.3 IMPLEMENTATION OF THE RESTORATION ALGORITHM

Figure 4.1 shows a schematic of an N=5 PSRR system and a communication structure designed to facilitate the restoration process. The CU's in the system are completely connected by dedicated bidirectional links. These links are hardwired signal lines directly connecting I/O ports in every pair of CU's in the system. However, the fanning out of

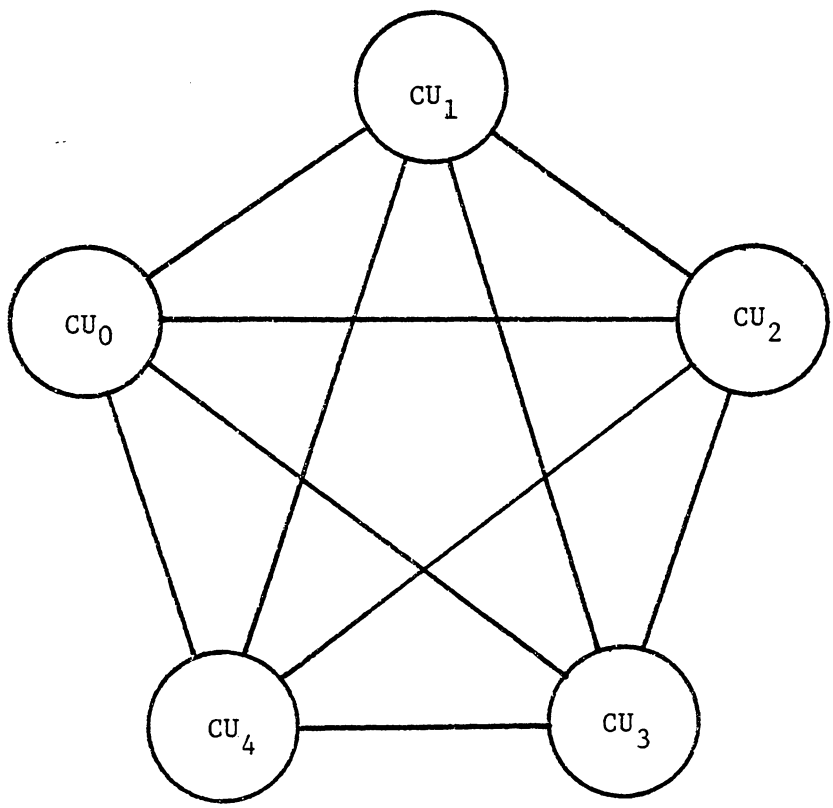

FIGURE 4.1: N=5 PSRR System.

signal lines from a single I/O port in each CU is not permissable because a single failure cannot be allowed to threaten the entire system. Therefore, each CU has N-1 I/O ports to connect it to the other CU's. We use a numbering scheme to identify the N CU'S in general N redundant PSRR systems. This is also illustrated by Figure 4.1. The CU's are arranged in a circle and labelled CU_0 through CU_{N-1} in order, moving clockwise. In keeping with our earlier convention, CU_0 is again the default CU. The N-1 links connecting each CU are labelled $LINK_1$ through $LINK_{N-1}$ such that $LINK_i$ of CU_j connects it to CU_k, where $k = (i+j) \bmod N$. Figure 4.2 shows the links connecting CU_3 in a N=5 PSRR system.

A CU sends a word to another CU by putting it out at the appropriate I/O port, where it is latched on to the link. The receiving CU then reads in the word at the next time step. For the transfer to be successful, the CU's must operate in perfect synchronization. No "hand shaking" is involved. However, a sending CU signals the transmission of a memory word during restoration by sending a BEGIN and an END control word before and after the memory word. A receiving CU that does not read the BEGIN and END control words at the appropriate times assumes that the sending CU has failed. This allows it to refrain from reading from the link when no word is sent. Such protection is essential because the signal levels on the links at idle times are not

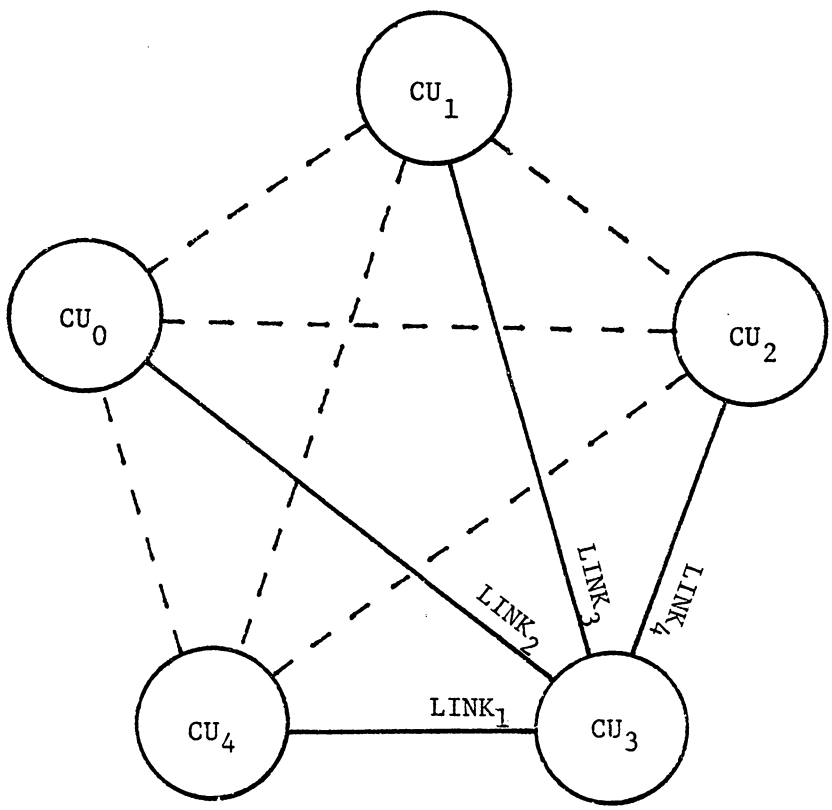

FIGURE 4.2: Links in a N=5 PSRR System.

likely to be completely random. Erroneous words read from the links can, therefore, cause incorrect restoration in ways not accounted for in the reliability model. This may very significantly degrade system reliability.

The BEGIN and END control words can be arbitrarily chosen, but should be distinct. To minimize the effects of unidirectional errors, they should contain about equal numbers of 1's and 0's.

To study the timing involved in transfering a word between the CU's, consider Figure 4.2 again and assume that the CU's are executing identical programs in perfect synchronization. If CU_k outputs a word on $LINK_i$ at any time step, it should read from $LINK_{N-i}$ at the next time step to receive the corresponding word sent by another CU in the system. Since $LINK_{N-i}$ connects CU_k with CU_m, where m=(k-i)mod N, the sending CU is CU_m.

We now present an algorithm which, if executed in synchronization by the CU's in the system, implements step (ii) of Algorithm 4.1. For any memory location, ADDRESS, it obtains for each CU, the contents of ADDRESS in all the other CU's that co-operate in the restoration process. The words obtained are stored in N temporary registers in each CU such that $TEMP_i$ contains the word stored in memory location ADDRESS in CU_i. CU's that do not co-operate in the restoration process are added to the set of definitely failed CU's, DFAIL. The algorithm assumes the existence of scratch registers CONTROLB and CONTROLE to read in the control words.

Algorithm 4.2

i) Let CU_k be the label on the CU executing the algorithm. Set i=1.

ii) Output control word BEGIN on $LINK_i$.

iii) Let j=N-i. Read from $LINK_j$ and store in CONTROLB.

iv) Output the word stored in memory location ADDRESS on $LINK_i$.

v) Read from $LINK_j$ and store the word in $TEMP_m$, where m=(j+k)modN.

vi) Output control word END on $LINK_i$.

vii) Read from $LINK_j$ and store in CONTROLE.

viii) If the contents of CONTROLB disagree with the BEGIN control word, or if the contents of CONTROLE disagree with the END control word, let DFAIL=DFAIL$\cup\{CU_m\}$. Else introduce appropriate delays to ensure that step (viii) always takes exactly the same length of execution time.

ix) Set i=i+1.

x) If $i \leq N$, go to step (ii).

xi) Stop.

Notice the delay introduced in step (viii) to ensure that the CU's stay in synchronization. This is required because during any given iteration (value of i) some CU's may read from failed CU's while others

read from operational ones. They may therefore execute different branches of step (viii). Introducing an appropriate delay to make both branches of exactly equal length guarantees that the CU's executing the restoration program stay in synchronization for all iterations.

We next present Algorithm 4.3 which compares, in each CU, the memory words obtained from the other CU's by Algorithm 4.2. The algorithm ensures that if two CU's disagree in their memory words, they are in different blocks of a partition of system CU's. It also ensures that each element of the set of definitly failed CU's, DFAIL, is in a block of the partition by itself. Thus, Algorithm 4.3 implements step (iii) of Algorithm 4.1.

Let $BLOCK_i$, $0 \leq i \leq N-1$, be the set of CU's that have agreed with CU_i in all memory locations compared so far. It is assumed that CU_i is in $BLOCK_i$. Let CUSET be the set of all CU's in the system, i.e. $CUSET=\{CU_0, CU_1, \ldots CU_{N-1}\}$.

Algorithm 4.3

i) Set i=0, j=0.

ii) If the contents of register $TEMP_i$ and $TEMP_j$ are not the same, set $BLOCK_i = BLOCK_i - \{CU_j\}$. Else introduce an appropriate delay to ensure a fixed execution time for this step.

iii) Set j=j+1.

iv) If j<N then go to (ii).

v) If $CU_i \varepsilon$ DFAIL then set $BLOCK_i = \{CU_i\}$, else introduce an appropriate delay.

vi) Set i=i+1, j=0.

vii) If i<N go to (ii).

viii) Stop.

We can now define a complete restoration algorithm for general N redundant PSRR systems that employ the communication structure described in this chapter. Algorithm 4.4 follows the outline of Algorithm 4.1.

Algorithm 4.4

i) Set $BLOCK_i$ = CUSET for all i, $0 \leq i \leq N-1$. Set DFAIL = {}, and ADDRESS = 0.

ii) Execute Algorithm 4.2.

iii) Execute Algorithm 4.3.

iv) If ADDRESS is less than the last (largest) valid memory address, then let ADDRESS = ADDRESS+1 and go to (ii).

v) Set ADDRESS = 0.

vi) Execute Algorithm 4.2.

vii) Execute Algorithm 4.3.

viii) Find a $BLOCK_j$ such that no other $BLOCK_k$, $0<k<N-1$, contains more elements than $BLOCK_j$. If $BLOCK_j = \{CU_j\}$ then store the contents of $TEMP_0$ at memory location ADDRESS. Else store the contents of $TEMP_j$ at location ADDRESS. (Ensure that both branches of this step take equal execution time).

ix) If ADDRESS is less than the last valid memory address, let ADDRESS = ADDRESS+1 and go to (vi).

x) Stop.

4.4 DISCUSSION

For a PSRR system to achieve the reliability estimates of Chapter Three, it is required that at the end of the restoration interval, temporarily failed CU's that correctly execute the restoration program be restored to the consensus state. This is the state of the largest block of agreeing CU's or, if all CU's disagree with one another, the state of the default CU. We now establish that Algorithm 4.4 correctly carries out this weighted plurality vote on the system states, even if failures take place during the restoration process.

Note that the algorithm does not make any decisions on the word to be stored in a memory location until the first pass over the entire memory in the CU's is complete and a partition based on agreement among the system CU's is obtained. Two CU's are in the same block of the partition if and only if they agree on the contents of every memory location during the first pass. Clearly, if no failures take place during the restoration process, the use of this partition to restore the memory during the second pass ensures correct restoration.

Now consider the possibility of CU failures during the restoration process. If the CU executing the algorithm itself fails, it is not expected to recover during the restoration interval under consideration.

However, failures in other CU's should not result in the incorrect restoration of a CU as long as it correctly executes the restoration program. Algorithm 4.4 ensures this by comparing the memory words from the different CU's again during the second pass and further partitioning the set of agreeing CU's if required. The word entered into memory (during this second pass) is decided by this new partition and is already available in the CU. A correctly operating CU will therefore be incorrectly restored only if two other CU's that have always agreed in the past communicate the same erroneous word to it at the same time during the second pass. Since the communication of a word between the CU's requires the sending of control words before and after the memory word, the two CU'smust be in sychronization with the rest of the system when this occurs. Thus, the error can only be caused by identical failures at the same memory location in the two CU's since the first pass during that restoration interval, or by identical failures at the same time in some other circuitry in the two CU'S. Either of these events is extremely unlikely, and the possibility that they occur can clearly be ignored.

On the basis of the above discussion, it can be concluded that Algorithm 4.4 can be used to implement the weighted plurality vote in general N redundant PSRR systems that employ the communication structure described in this chapter.

In concluding the discussion on the restoration process in PSRR systems, we make some observations on the relation between execution

time for the proposed algorithm and the level of redundancy, N. Note that Algorithm 4.2 is (O)N, while Algortithm 4.3 is $(O)N^2$. Since Algorithm 4.4 executes each of these twice for each memory word, the restoration time t_R for PSRR systems will be of the form

$$t_R = k_1N^2 + k_2N + k_3 \qquad (4.4\text{-}1)$$

where k_1, k_2, and k_3 are constants that depend on the system. For relatively high levels of redundancy, the restoration time is expected to be proportional to N^2.

We shall use this relation between the restoration time and the level of redundancy in the next chapter, where we study trade offs in the design of PSRR systems.

Chapter V

TRADE-OFFS IN THE DESIGN OF PSRR SYSTEMS

5.1 INTRODUCTION

In the preceding chapters, we have established the feasibility of PSRR systems. We now discuss the trade offs available to the designer of such systems. These trade offs are between performance, reliability, and the level of redundancy (which is a measure of cost). The discussion leads to a design procedure for implementing PSRR systems to desired reliability and performance specifications.

5.2 THE PERFORMANCE-RELIABILITY TRADE OFF

A PSRR system is unable to work on user programs during the restoration interval. As a result, while enhancing reliability, the periodic restoration process degrades the computational performance of the system from a user perspective. Let

$$P_C = \frac{t_c}{t_c + t_R} \tag{5.2-1}$$

$$= \frac{t_c}{t_o} \tag{5.2-2}$$

be a measure of this computational performance. Note that P_C is the fraction of the total time that the system is available to the user.

The restoration interval t_R in equation (5.2-1) is determined by the time required to execute the restoration program. While this depends on

the level of redundancy and the memory size of each CU, it is fixed for a given system. The computing interval t_C is a design option and determines the C-R cycle time t_o. We have seen in Chapters 2 and 3 that shorter C-R cycles (smaller values of t_o) increase system reliability by reducing the possibility of system failure due to the accumulation of temporarily failed processors. But from equation (5.2-2) it can be seen that because t_R is fixed, a smaller t_o reduces the computational performance of the system. We thus have a trade off between performance and reliability in PSRR systems.

Figure 5.1 illustrates this trade off for a typical system with N=3, λ_t=0.01 per hour, λ_p=0.001 per hour, and t_R=0.005 hours. Reliability plots for P_C=0.95, 0.8, 0.6, and 0.2 are displayed. Notice in the plots that an equal reduction in P_C has a more significant effect on system reliability for shorter missions. For example, reducing P_C from 0.95 to 0.8 reduces the probability of system failure by 50% over 36 hours of operation, but only reduces it by 20% for a 256 hour mission. Note also that the improvement in system reliability (as measured by the failure probability) for a given reduction in P_C decreases very substantially as P_C becomes smaller. This is explained by the fact that system reliability depends on t_o, which is a measure of how frequently the system is restored. It can be seen from equations (5.2-1) and (5.2-2) that

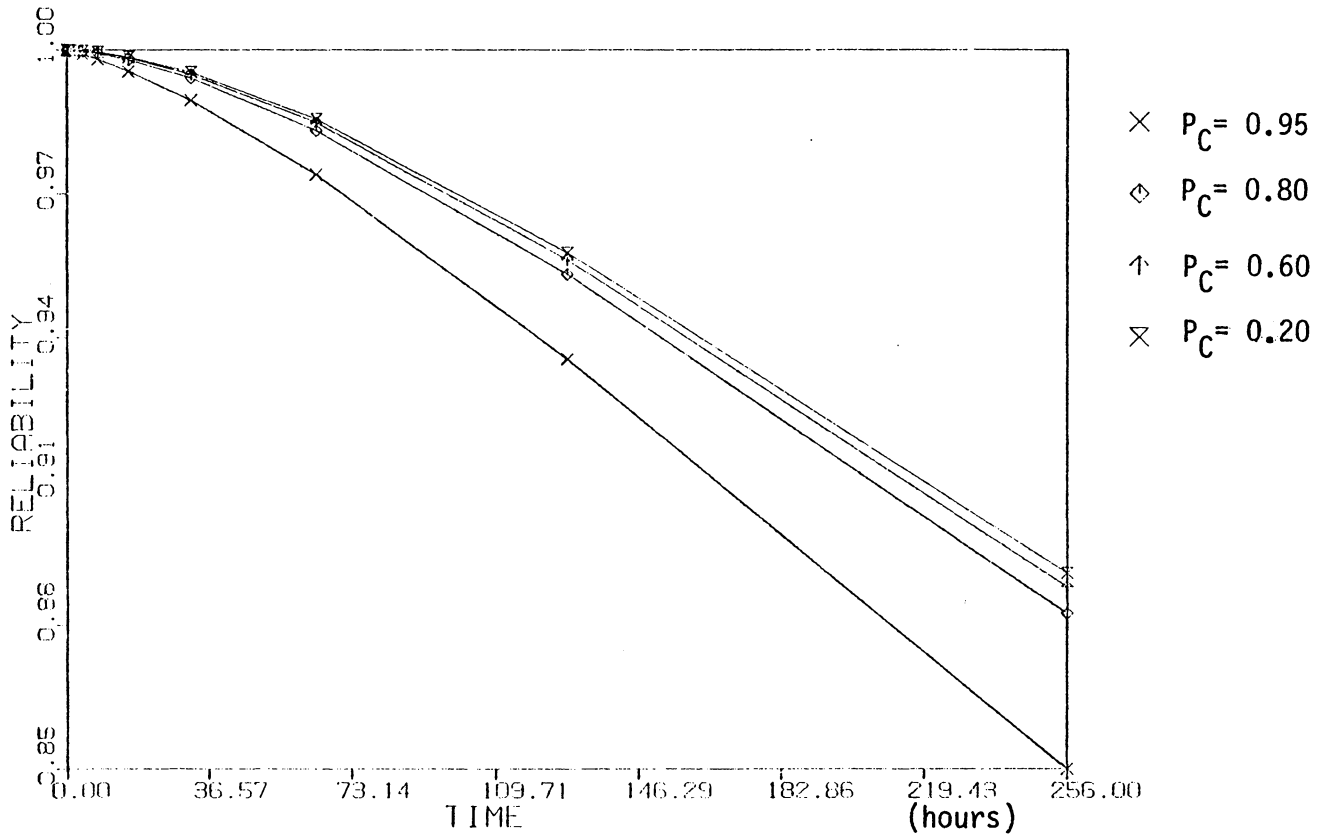

FIGURE 5.1: The Reliability - Performance Trade Off.

$$t_o = \frac{t_R}{1-P_C} \qquad (5.2\text{-}3)$$

Thus, the fractional reduction in t_o for a given reduction in P_C decreases substantially as P_C becomes smaller. For example, reducing P_C from 0.99 to 0.9 reduces t_o by a factor of 10, while reducing P_C from 0.9 to 0.8 only reduces t_o by a factor of 2. Since for most systems, a reduction in the probability of failure by 10 or 20 per cent is not of practical significance, it appears that reducing P_C below 0.8 is not likely to be worthwhile for most systems.

5.3 THE REDUNDANCY-RELIABILITY TRADE OFF

The reliability plots in Chapter 3, Figures 3.1 and 3.2, indicate a very significant improvement in system reliability with increasing levels of redundancy, N. However, the plots were obtained assuming that the C-R cycle time and CU failure rates remain constant for all N. This assumption is not valid for practical systems. We found in Chapter 4 that the restoration interval t_R is $(O)N^2$, and therefore increases quite significantly with N. Since a performance-reliability trade off is available in PSRR systems, a study of the increase in reliability with increasing redundancy is only meaningful if the computational performance factor P_C is kept constant. It can be seen from equation (5.2-1) that this

involves increasing t_C with N to offset the increase in t_R. The resulting increase in t_o neutralizes some of the gain in system reliability from increased redundancy. Further, an increase in N causes an increase in the complexity of the CU's in the system, because of the additional I/O ports required (as explained in Section 4.3). This results in increased CU failure probabilities, which further degrade system reliability.

Figure 5.2 shows reliability plots for PSRR systems with different levels of redundancy. All plots in the figure are for P_C=0.9, with t_C=9t_R. It is assumed that CU failure rates are proportional to their complexity. The CU's in the N=3 system are assigned unit complexity (C=1) and have λ_t=0.01 per hour, λ_p=0.001 per hour, and t_R=0.0045 hours. Because, in Chapter 4, the restoration interval was found to increase proportional to N^2 it is taken to be 0.008 hours for the N=4 system, and 0.0125 for the N=5 system. The plots for the N=4 system are for C=1.0, 1.25, and 1.5, while those for the N=5 system are for C=1.0, 1.5, and 2.0.

The figure clearly shows that despite the degrading affects of increased t_o and failure rates, very substantial improvements in reliability can be obtained by increasing the level of redundancy in PSRR systems. Consider, for example, a mission time of 64 hours and assume that the failure rate for a CU increases by 25% of the N=3 CU failure rate for each increase in the level of redundancy. It is observed in Figure 5.2 that the system failure probability still decreases by a factor

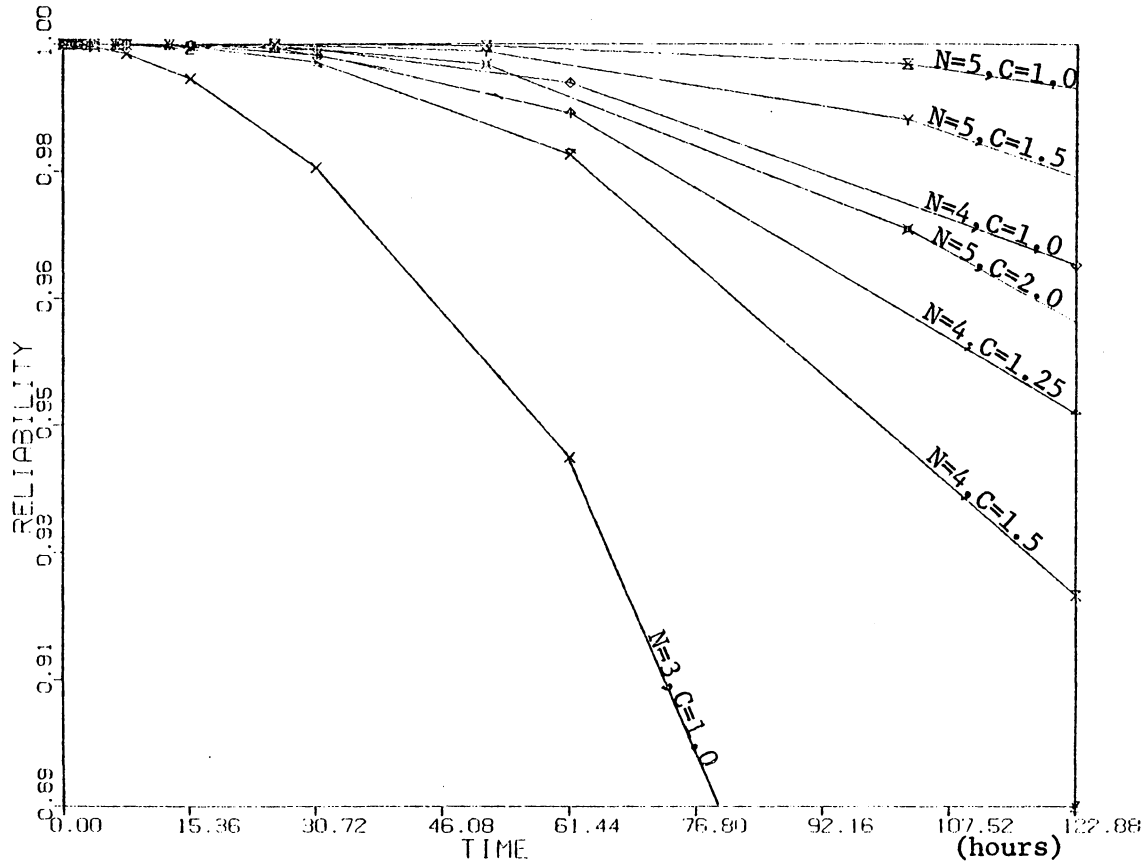

FIGURE 5.2: The Reliability - Redundancy Trade Off.

of more than 5 for each increase in the level of redundancy. This substantial increase in system reliability with redundancy indicates that PSRR systems can be designed to be ultra-reliable.

5.4 A DESIGN PROCEDURE

We now consider the problem of designing PSRR systems to desired performance and reliability specifications. It should be apparent from the trade off discussions earlier in this chapter, that there may be many different PSRR systems that meet such a specification. These systems will have varying levels of redundancy and employ CU's with differing computational capabilities and failure rates. The preferred design must be decided from among these many possibilities based on criteria such as cost and the margin of safety in reliability and/or performance. Therefore, we now present a procedure that evaluates the level of redundancy N required by a PSRR system to meet a minimum reliability specification, given the computational capability and failure probabilities of the basic CU used to implement the system. The procedure also provides the reliability actually obtained, which in general, may be greater than the minimum specification since the choice of N is quantized. This procedure can be carried out for all basic CU's that are likely candidates for the desired system. The best choice based on the above mentioned criteria can then be used to actually implement the system.

Procedure 5.1

i) Let the basic N=3 CU be such that with a computational performance factor P_C (P_C<1) it satisfies the computational requirement of the system specification. This N=3 CU must have two I/O ports to implement the restoration process and a ROM to store the restoration program. It must also have a fault tolerant clocking module that can be linked to similar modules in other CU's to generate the sychronized system clock and also provide the periodic restoration interrupt. For this CU, write a restoration program to implement Algorithm 4.4 presented in Chapter 4. Estimate the execution time t_R for this restoration program for N=3 PSRR system. From equation (5.2-1), we have $t_C = P_C t_R/(1-P_C)$. Therefore, t_C and $t_o=t_C+t_R$ can be calculated for the system.

ii) Estimate the temporary and permanent failure probabilities for the CU's over t_C and t_R. Using these probabilities and the reliability model of Chapter 3, evaluate the reliability of the N redundant system for the desired mission time.

iii) If the reliability is greater than specification, record the level of redundancy N and the reliability obtained. Stop.

iv) Set N=N+1, thereby increasing the level of redundancy by 1. The CU now requires an additional I/O port and a modified clock module capable of generating signals to an additional CU. This increased complexity should be considered when

evaluating failure probabilities at the next step. Estimate the execution time t_R for the restoration program at this higher level of redundancy. Again set $t_C = P_C t_R/(1-P_C)$ and go to step (ii).

5.5 DISCUSSION

Note that Procedure 5.1 is completely general and does not assume that the CU's have constant failure rates. This allows a system may be important because some recent studies[27,28] indicate that transient failures in digital systems do not occur at a constant rate. The failure rates appear to vary with the time since the last transient and the system workload. The reliability model developed in this dissertation, and Procedure 5.1, can still be used if these studies are confirmed and found applicable to CU's in PSRR systems. It should be noted, however that the majority of the reported failure statistics are for systems that employ some redundancy. Futher studies are required before abandoning the assumption that transient faults occur at constant rates.

The generality of the reliability model and Procedure 5.1 with respect to the failure statistics also allows a system designer to consider components that have built in redundancy and therefore cannot be modeled assuming constant failure rates. But again, in practice, most VLSI circuits do not employ large scale internal redundancy. This is because the large majority of the failures in such components are associated with the I/O connections on the chip. Redundancy in the circui-

try only offers very limited improvements in reliability. and often this advantage can be more than offset by the additional difficulty in testing such circuits. It is therefore expected that failures in most CU's can be reasonably modelled by assuming constant failure rates.

Note that individual CU's in PSRR systems cannot be provided with any software protection against transients. The execution of such a routine would put the CU out of synchronization with the other CU's in the system, thereby making any recovery worthless.

Several approaches can be used to obtain the failure probabilities required by step (ii) of Procedure 5.1. The most direct method is by the statistical failure testing of the system CU's, but this will usually not be practical. Since the CU's require only minimal specialized hardware, it may often be possible to implement them using proven off the shelf components. For such systems the failure history of the building blocks can be used to estimate the failure probabilities for the CU's. For systems built from specially designed components, the failure rates must be estimated based on factors such as the logic technology, the complexity of the circuitry, the type of packaging, etc. Some models for predicting LSI microcircuit failure rates based on these parameters have already been developed [29].

To allow a quick estimation of the reliability of PSRR systems with N between 3 and 5 and constant CU failure rates, a set of plots is included at the end of this Chapter. Time is scaled in the plots in terms of the mean time between temporary CU failures, $1/\lambda_t$, so that they can be

used for arbitrary temporary failure rates. The plots display system reliability and MTTF for $\lambda_p=\lambda_t/10$ and $\lambda_t/50$, and t_o ranging between $10^{-1}\lambda_t$ and $10^{-6}\lambda_t$. Since highly reliable operation is of interest to the designer of PSRR systems, the plots only show relatively short mission times that give system reliability of 0.5 or better.

As an example, consider a quad redundant (N=4) PSRR system built out of CU's with $\lambda_t=0.005$ per hour, $\lambda_p=0.0004$ per hour, and $t_R=0.01$ hours (36 seconds). Let us estimate the highest reliability obtainable over a 50 hour mission if the system is to be available to the user at 95% of the time, i.e. $P_C=0.95$.

Note that by equation (5.2-3) $t_o=0.2$ hours. Scaling t_o and the desired mission time T so as to express them in terms of λ_t, we get $t_o=10^{-3}/\lambda_t$ and $T=0.25/\lambda_t$. Since $\lambda_t/\lambda_p=12.5$, which is quite close to 10, we can estimate the system reliability from Figure 5.9 to be 0.9992. Observing, informally, the sensitivity of this estimate to variations in t_o and λ_t/λ_p (from Figure 5.10), we expect the corresponding failure probability to be at least within an order of magnitude of the actual value.

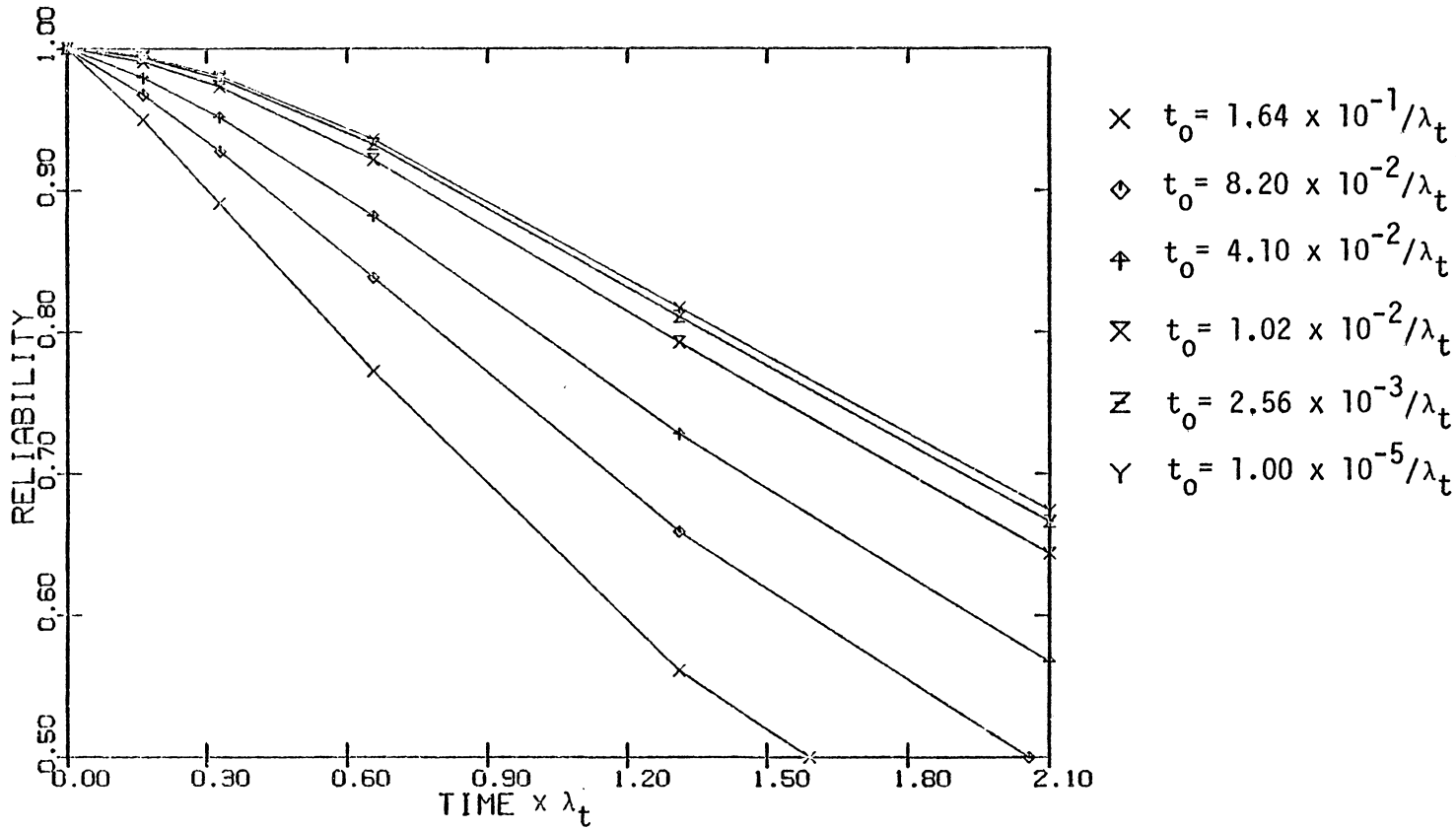

FIGURE 5.3: Reliability Plots for N = 3, $\lambda_p = 0.1\lambda_t$.

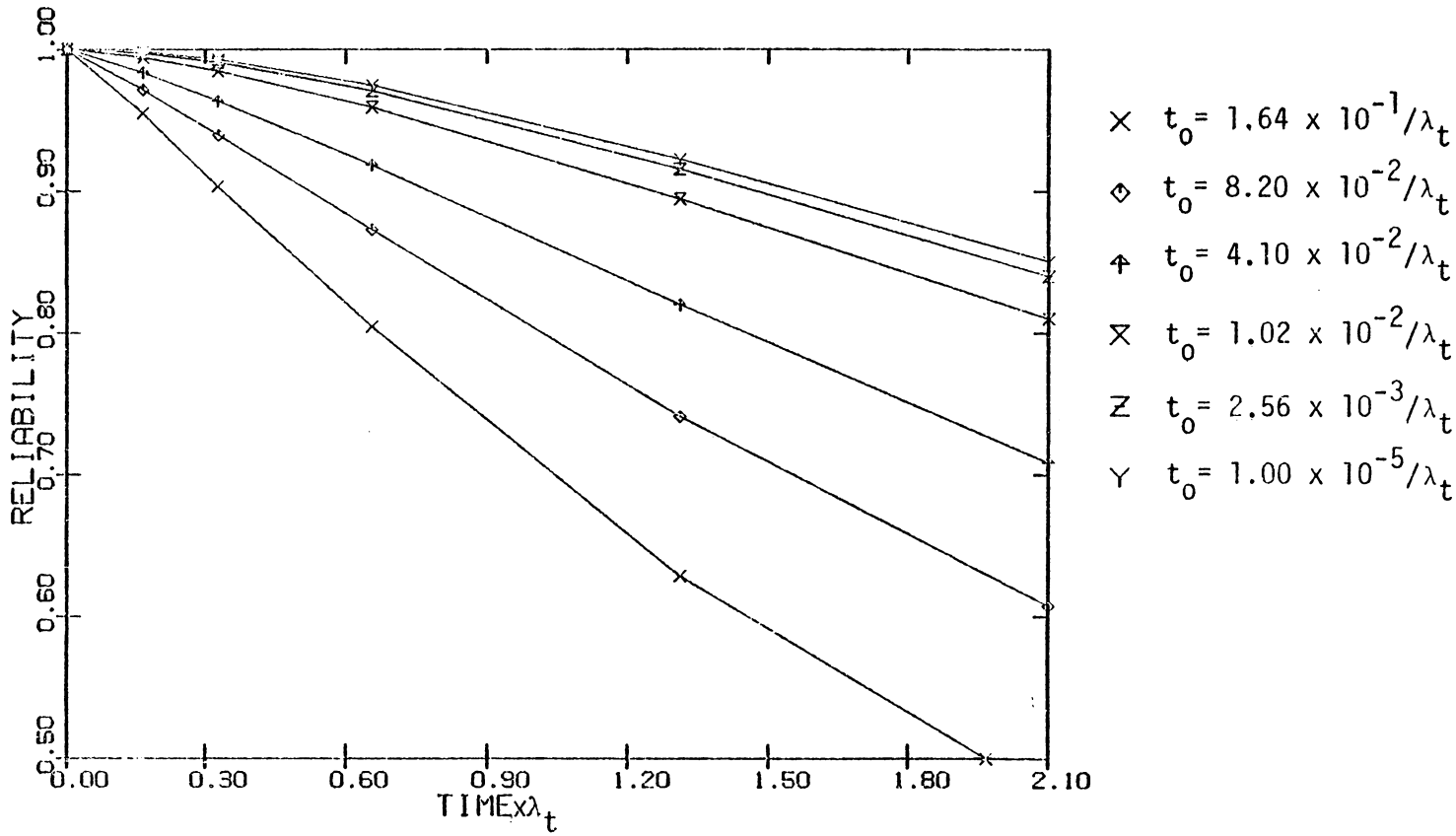

FIGURE 5.4: Reliability Plots for N = 3, $\lambda_p = 0.04\lambda_t$.

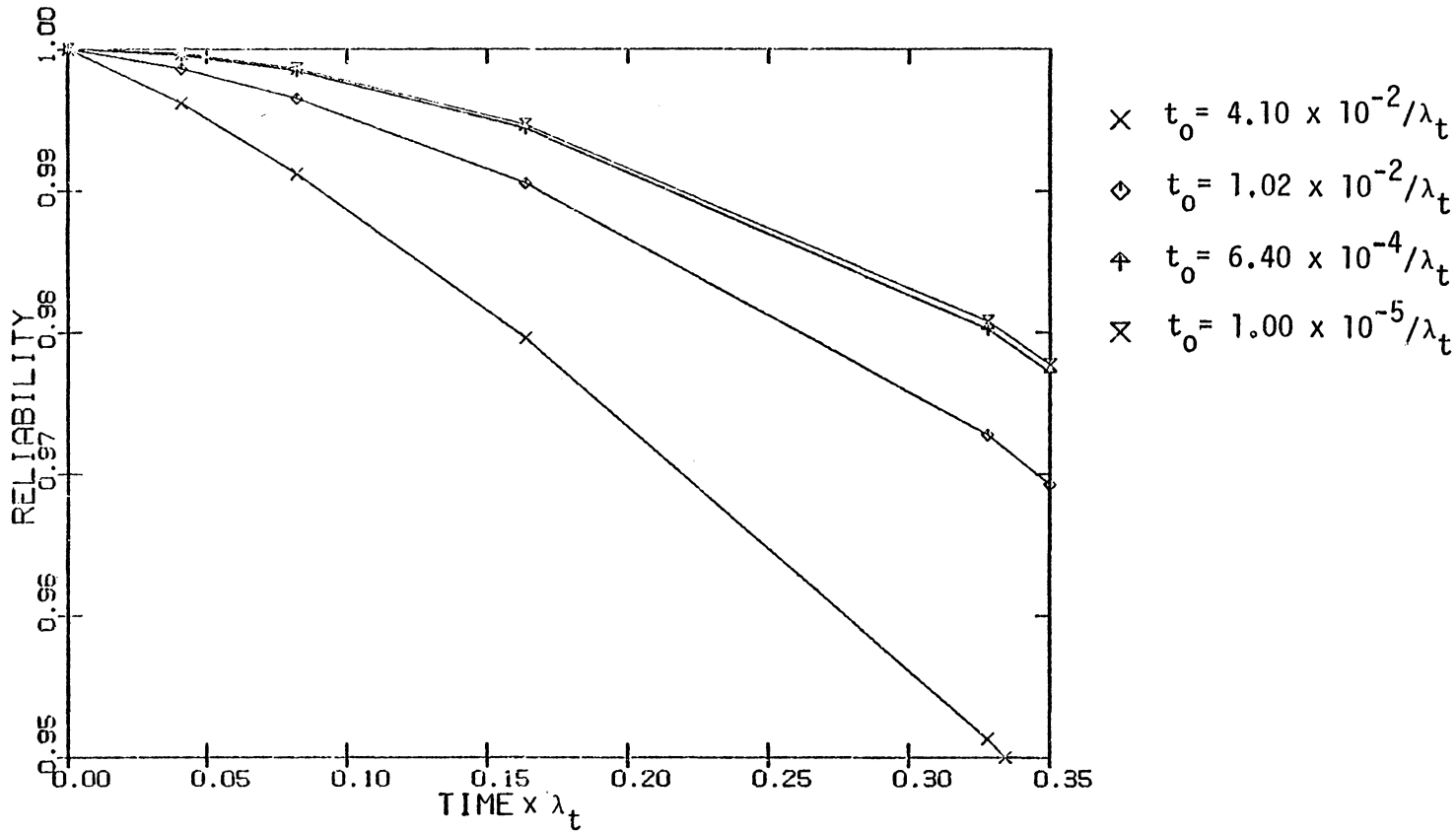

FIGURE 5.5: Reliability Plots for N = 3, $\lambda_p = 0.1\lambda_t$.

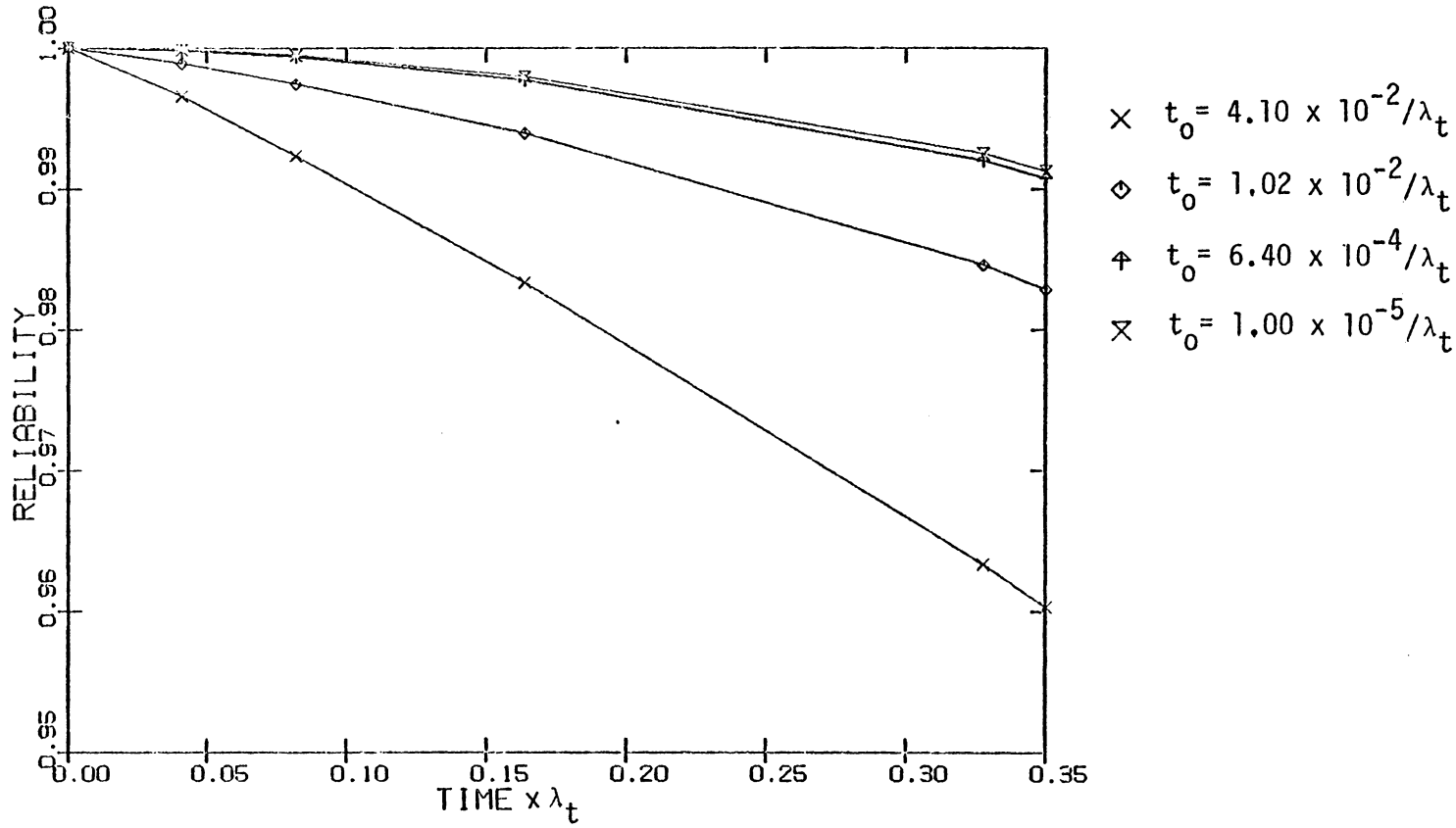

FIGURE 5.6: Reliability Plots for N = 3, $\lambda_p = 0.04\lambda_t$.

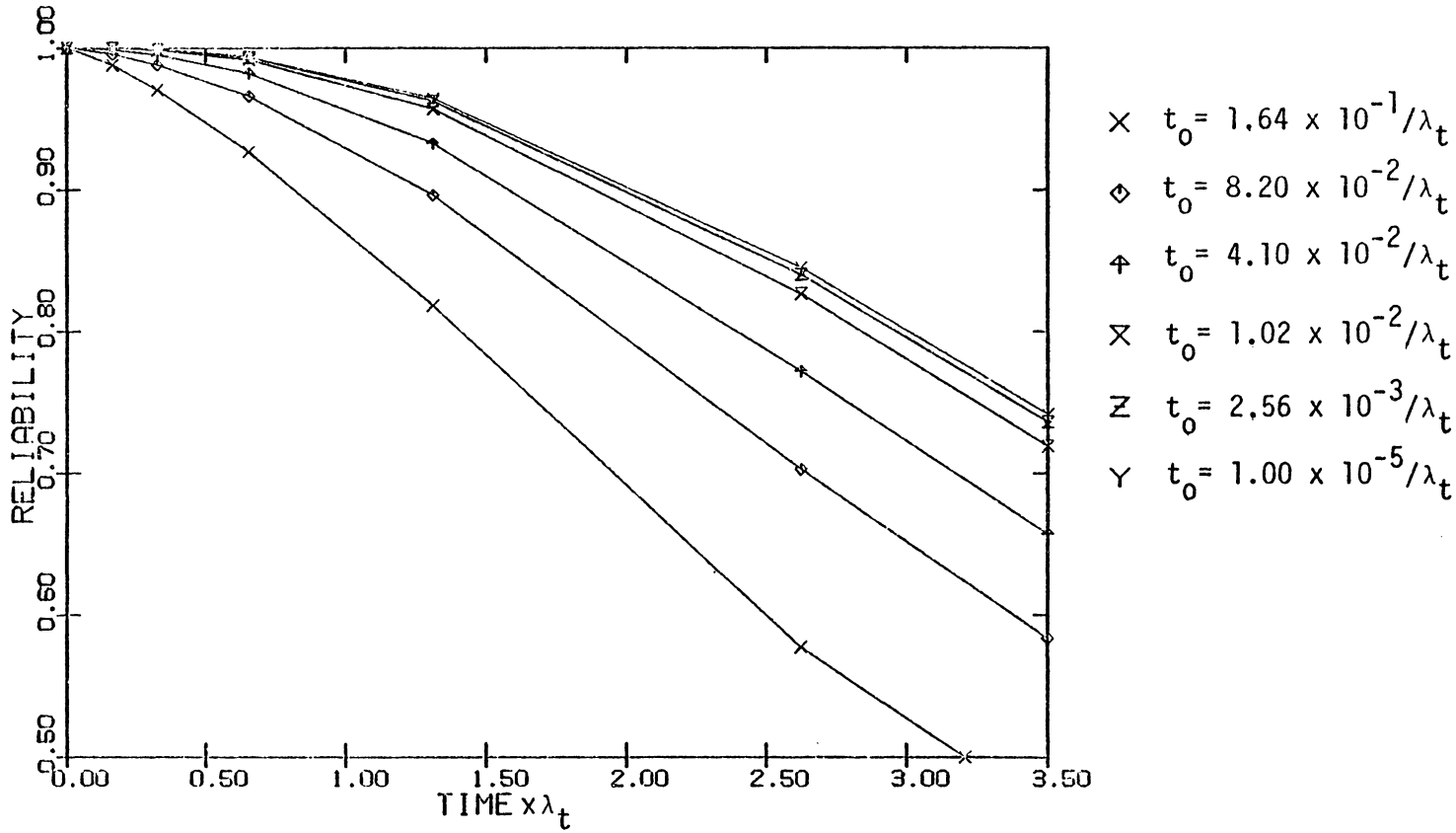

FIGURE 5.7: Reliability Plots for N = 4, $\lambda_p = 0.1\lambda_t$.

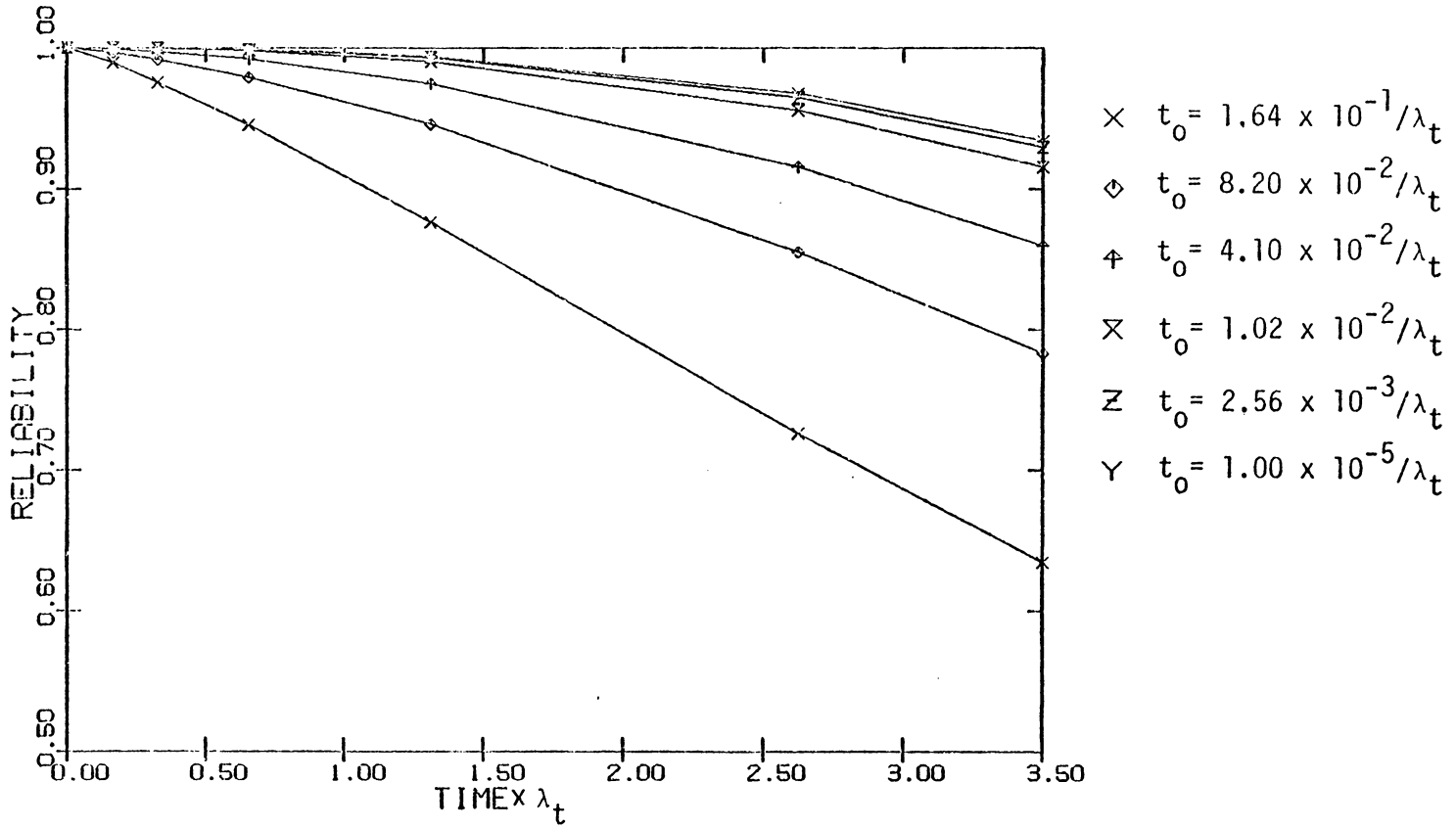

FIGURE 5.8: Reliability Plots for N = 4, $\lambda_p = 0.04\lambda_t$.

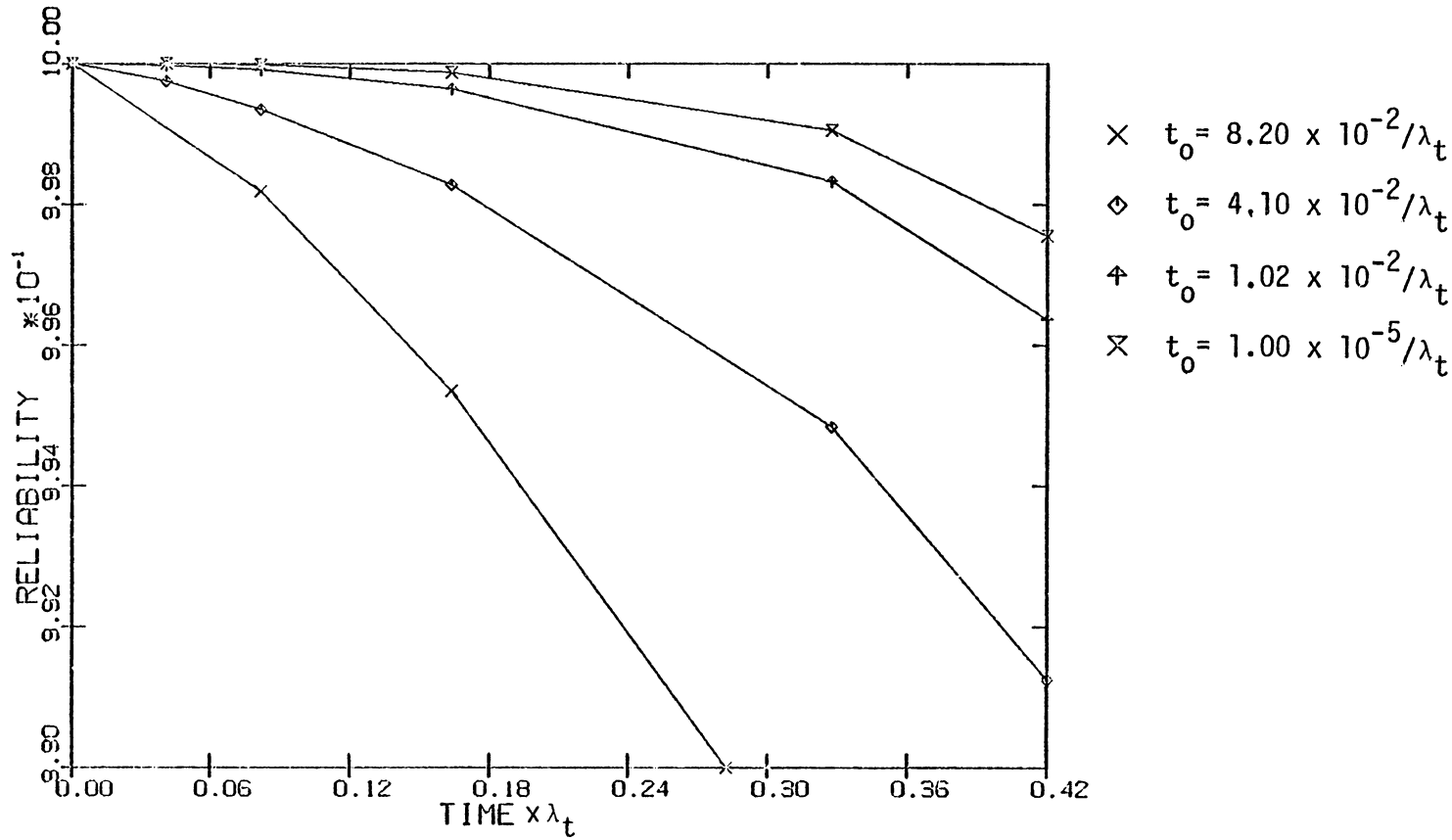

FIGURE 5.9: Reliability Plots for N = 4, $\lambda_p = 0.1\lambda_t$.

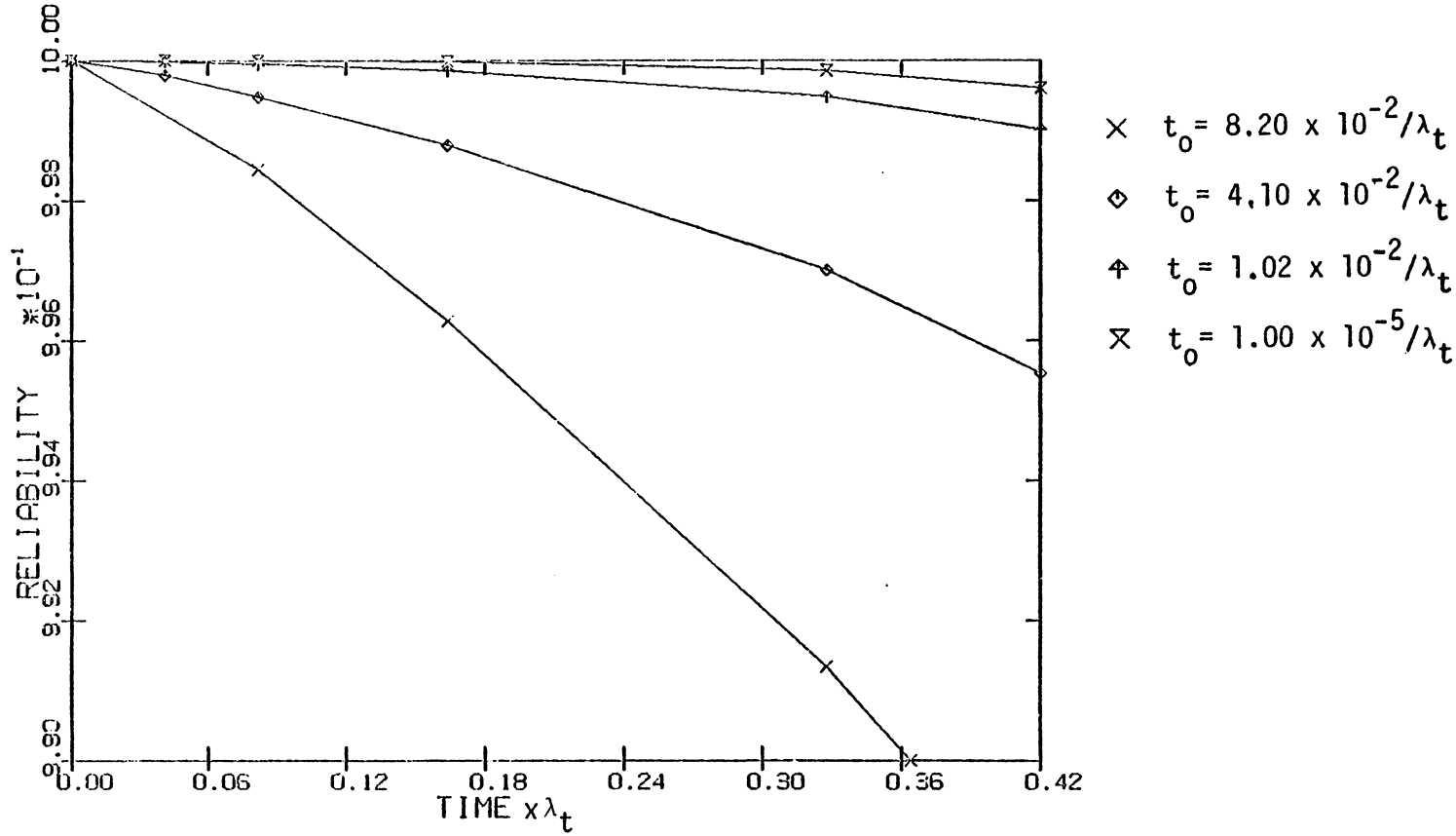

FIGURE 5.10: Reliability Plots for N = 4, $\lambda_p = 0.04\lambda_t$.

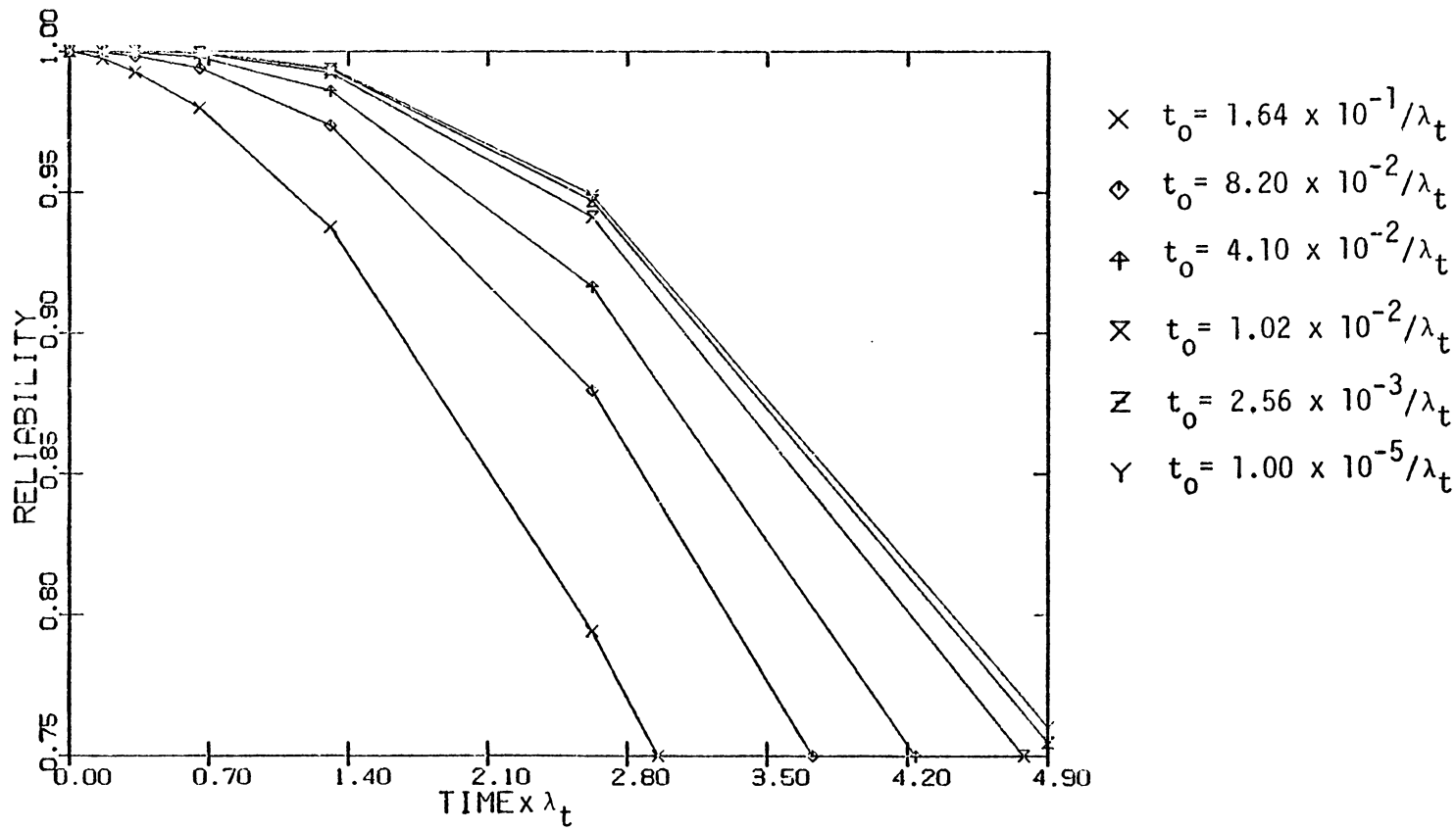

FIGURE 5.11: Reliability Plots for N = 5, $\lambda_p = 0.1\lambda_t$.

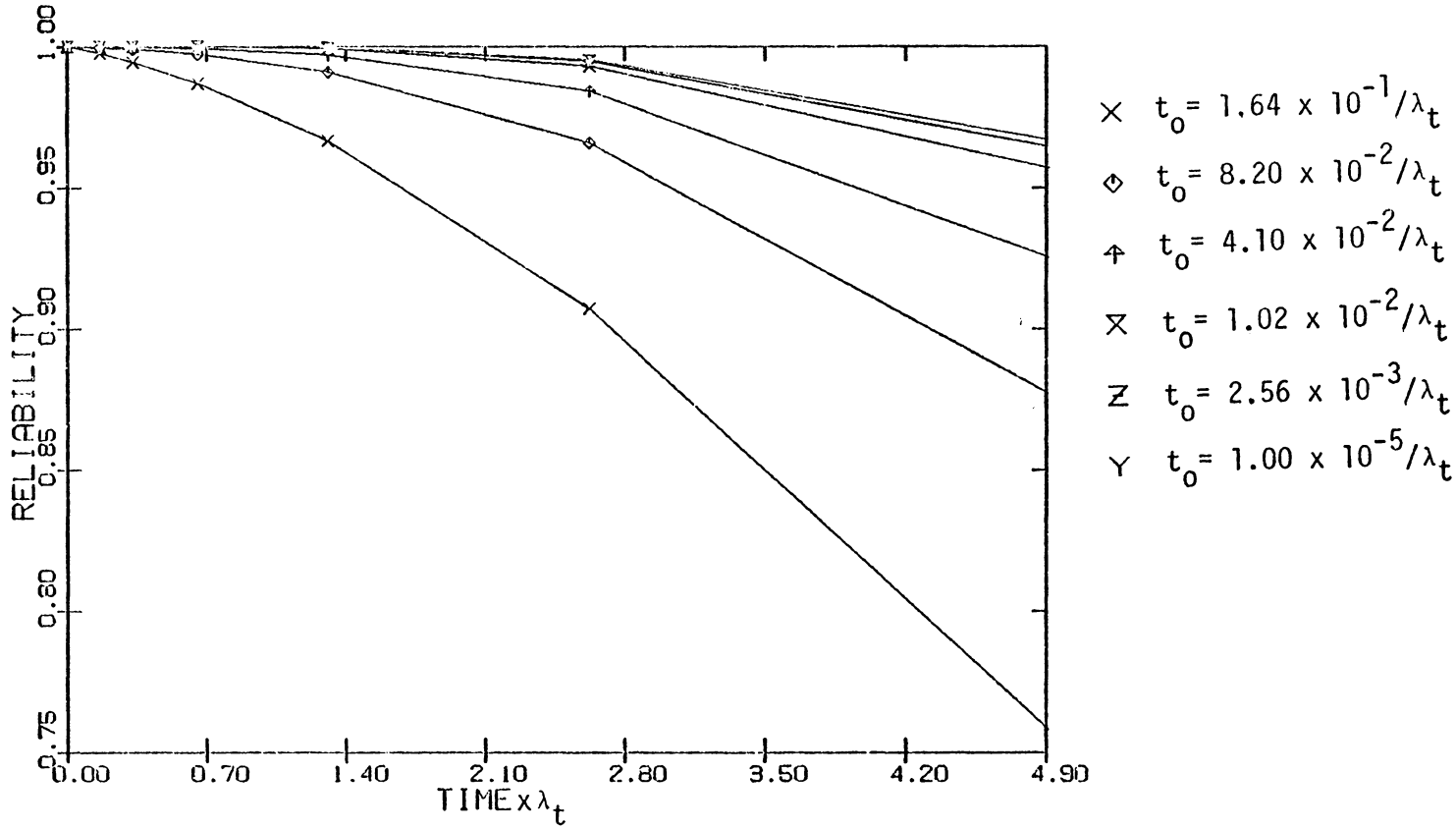

FIGURE 5.12: Reliability Plots for N = 5, $\lambda_p = 0.04\lambda_t$.

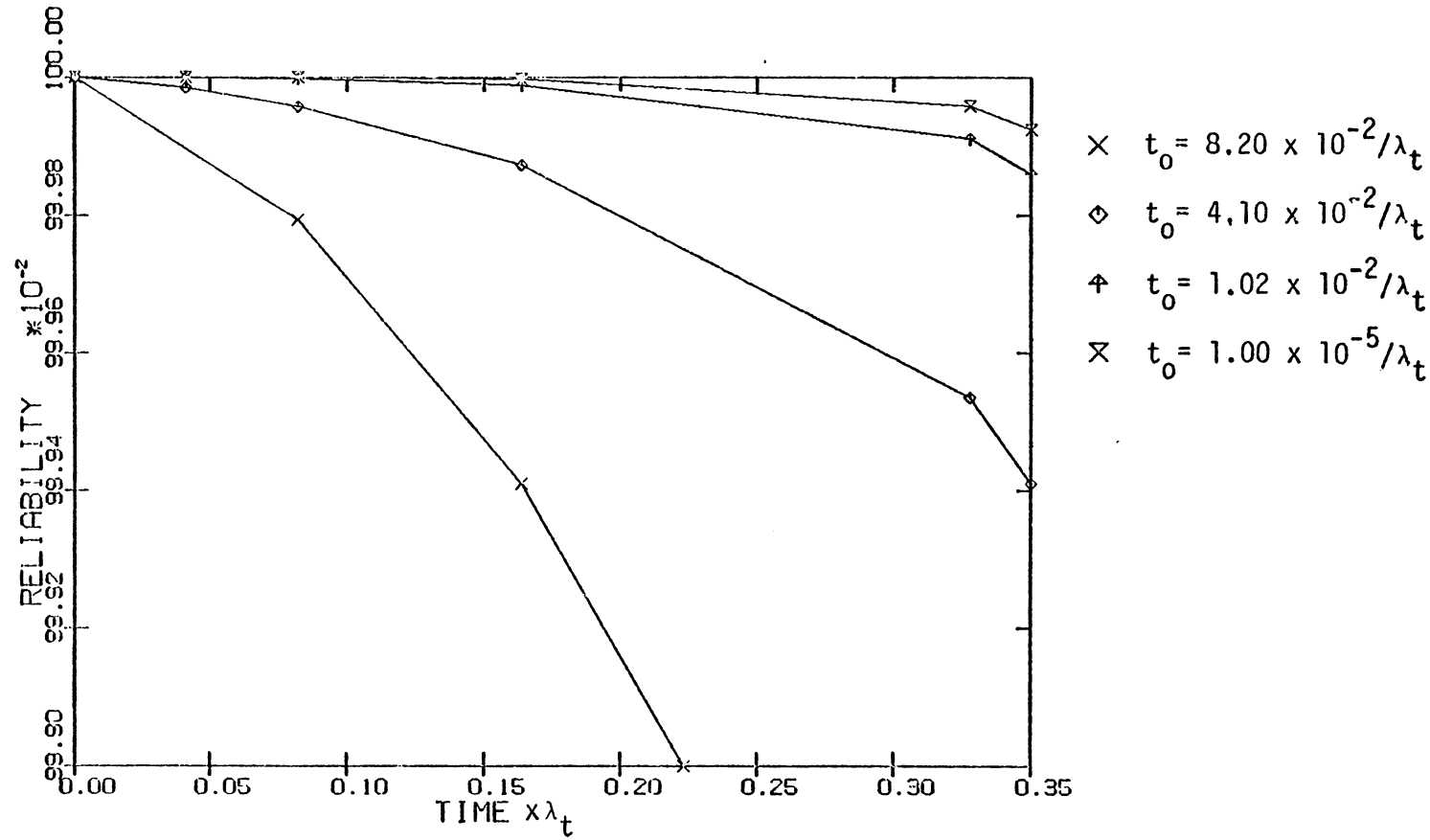

FIGURE 5.13: Reliability Plots for N = 5, $\lambda_p = 0.1\lambda_t$.

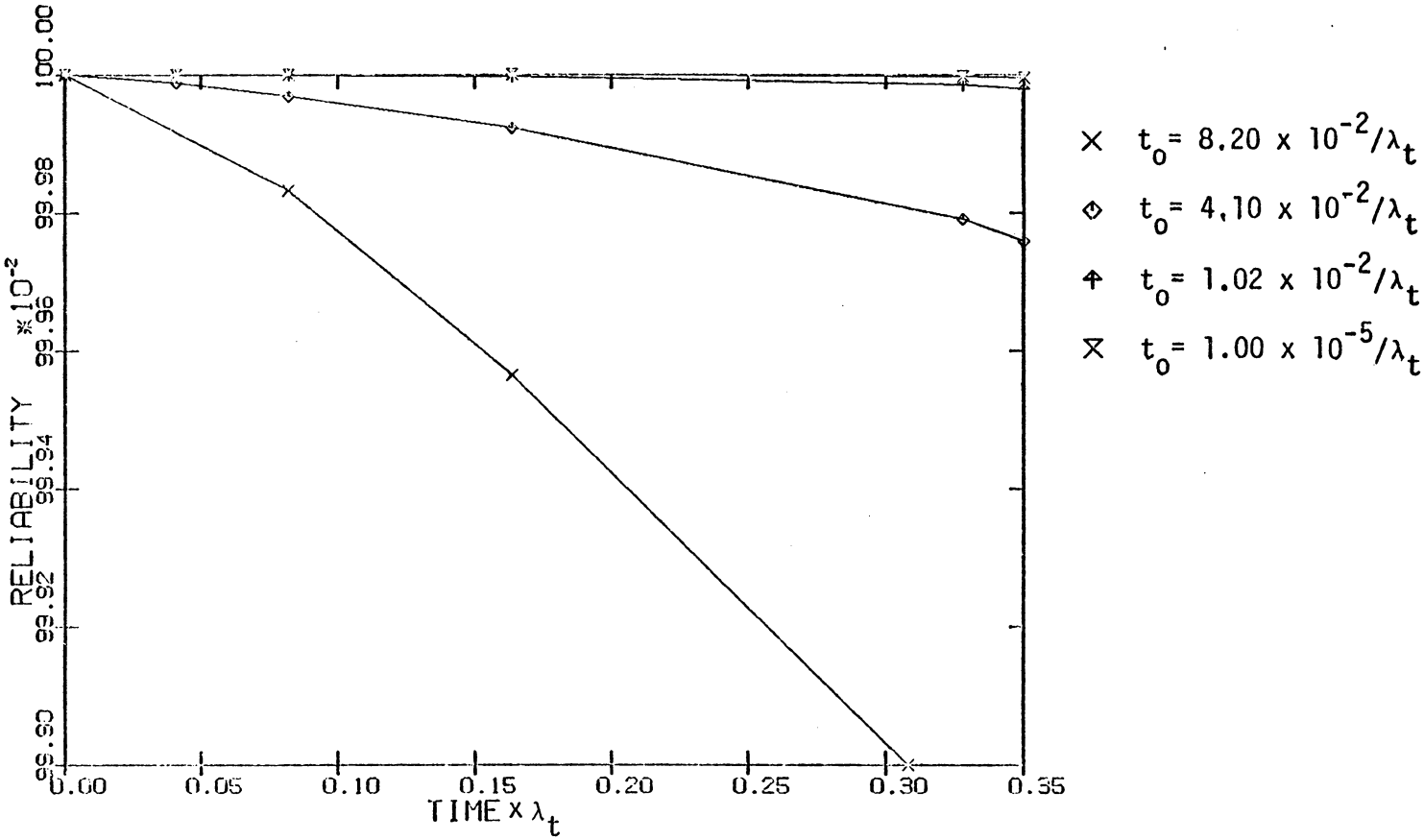

FIGURE 5.14: Reliability Plots for N = 5, $\lambda_p = 0.04\lambda_t$.

Chapter VI

CONCLUSIONS

6.1 SUMMARY

The dynamic redundancy methods that are at present widely used in fault tolerant designs can be considered to be incident driven. In such designs, it is the detection of an error that triggers the attempt to restore the system to the correct operational state. In this dissertation, we have proposed and analyzed an alternative approach, one where the redundant system periodically restores itself so as to correct errors before they build up to the point of system failure.

The proposed periodically self restoring redundancy (PSRR) scheme is introduced in Chapter One. In general, an N redundant PSRR system employs N computing units (CU's) operating redundantly in synchronization. Each CU has all the computational capabilities of the desired fault tolerant system. System input is simultaneously provided to all N CU's. System output can be recognized from among the N outputs available from the CU's based on certain decision rules. These decision rules provide the correct system output even if the output of some of the CU's is erroneous.

The CU's in a PSRR system can fail due to both permanent and transient faults. While those with permanent faults cannot be reliably resynchronized with the rest of the system, CU's that fail due to transients can be restored to the operational state. To achieve this, the N

CU's in the system periodically communicate their state information to each other and resynchronize themselves to a mutual consensus state. If a certain number of CU's in the system are operational, this consensus is assured of being the correct operational state. The restoration is initiated by a non-maskable interrupt from a fault tolerant clock and is executed by each CU out of ROM. This ensures that a CU that may have failed due to a transient still executes the restoration program in synchronization with the other CU's in the system. It is, therefore, restored to the operational state, if enough other CU's in the system are operational.

It should be clear from the above description that PSRR systems are particularly effective at handling transient faults. This is an important advantage because transient faults are believed to occur much more frequently than permanent faults. PSRR systems can also tolerate a certain number of permanent faults because the correct output can be recognized from among the N outputs available from the system even in the presence of some failed CU's.

It should be clear that the reliability of such a system depends on how often the CU's are restored. This is decided by the computation - restoration (C-R) cycle time. A C-R cycle consists of a computing interval, during which the system does useful computation, and a restoration interval during which failed CU's are restored. Shorter C-R cycles imply more frequent restoration. This results in increased system reliability because it reduces the possibility of system failure due to an accumulation of failed CU's.

In Chapter Two we analyze triple redundant PSRR systems that employ a simple majority vote to decide the consensus state and system output. A reliability model for such systems is developed, based on the theory of Markov Chains. The model considers failures due to both transient and permanent faults. Making realistic simplifying assumptions, closed form expressions for system reliability and mean time to failure (MTTF) are derived. These expressions are in terms of the failure probabilities for individual CU's due to transient and permanent faults during a C-R cycle, the C-R cycle time, and the mission time. As expected, the expressions show that both system reliability and the mean time to failure increase significantly with more frequent restoration.

Reliability and MTTF expressions are also derived in Chapter Two to account for the possibility of failures in the system CU's at the start of the mission. This is important because in practice it is impossible to guarantee that the system is free of failures at the initial time. These expressions show that the possibility of permanent faults in the CU's at the start of the mission has a greater impact on system reliability than the possibility of initial CU failures due to transients. This again is to be expected, as failures due to transients will be restored during the first restoration interval while permanent failures stay in the system for all time and continue to degrade reliability.

General N redundant PSRR systems are analyzed next in Chapter Three. A weighted plurality vote is defined for deciding the consensus

CU state and system output in such systems. This ensures an operational system as long as there are two correctly operating CU's, and in some cases allows operation down to a single good CU. A Markov model for N redundant PSRR systems employing such a weighted plurality vote is developed. The model is completely general and makes no simplifying assumptions. Algorithms for evaluating system reliability and MTTF are given. These algorithms require estimates of the failure probabilities for the system CU's over the computing and restoration intervals, and the length of these intervals. The model is general enough to allow both constant and time varying CU failure rates. The possibility of failed CU's at the start of the mission is also incorporated into the model.

The reliability model in Chapter Three assumes that during the restoration interval, the weighted plurality vote on the CU states can be implemented Chapter Four shows how this can be achieved. An interconnection network is proposed to support the communication between the CU's during the restoration process. Then a restoration algorithm is developed for execution by each of the system CU's in synchronization during the restoration interval. The algorithm ensures that any CU that executes it correctly is restored to the desired weighted plurality state.

Having established the feasibility of PSRR systems the trade offs available to a system designer are next analyzed in Chapter Five. It is found that the reliability of PSRR systems can be improved either by

more frequent restoration, which degrades the computational performance of the system or by increasing the level of redundancy, which increases cost. These tradeoffs suggest that PSRR systems can be implemented to meet given specifications in several different ways. A procedure is presented in Chapter Five that allows evaluation of the level of redundancy needed by a PSRR system, based on a given CU, to meet desired performance and reliability specifications. It is suggested that this procedure can be carried out for all CU's that are likely candidates in the design of such a system. The best choice based on criteria such as cost, the margin of safety in reliability and/or performance can then be used to implement the system.

Thus, this dissertation provides a systematic approach for implementing fault tolerant systems to meet desired reliability and performance specifications. Such a design procedure is made possible by the fact that the proposed PSRR scheme lends itself to reliability modeling in terms of parameters that can be readily estimated with reasonable accuracy. This is not true of most present day fault tolerant systems because they employ dynamic redundancy. System reliability in such designs is a very sensitive function of coverage, which is virtually impossible to predict with enough accuracy during the design stage to get meaningful reliability estimates. As a result the present day design of fault tolerant systems is an expensive iterative process involving design modification and testing until the desired reliability is achieved.

In addition to making possible the systematic design of fault tolerant systems, PSRR systems have another significant advantage. Because they employ high level redundancy and need only minimal specialized hardware, they can be largely implemented from proven off-the-shelf components. This should result in substantial savings in component costs and design time.

A disadvantage of the proposed system is that it is unable to repond to interrupts during the restoration intervals. This may disqualify PSRR systems from use in some real time control applications. However, it may be possible to overcome the problem by operating two PSRR systems in parallel such that their restoration intervals do not overlap. This would ensure that at least one system is always available to service real time interrupts.

6.2 SUGGESTIONS FOR FUTURE STUDY

The obvious next step in the study of PSRR systems is the implementation of such a design. The hardware can then be used to validate the reliability models developed in this dissertation.

As an analytical exercise, it may be useful to obtain the reliability of general N redundant PSRR systems that employ weighted plurality voting for restoration but use a simple majority among the CU outputs to decide the system output. This is a relevant problem because the combinational circuitry required to carry out the weighted plurality vote on the CU outputs is very complex. It may be more practical to obtain

the system output by implementing a simple majority vote on each bit of the output word from the N CU's.

At a more general level, the possibility of combining the proposed PSRR scheme with other redundancy schemes is suggested for investigation. Such an attempt should make use of the fact that PSRR systems are very effective at handling transient faults while traditional redundancy techniques were developed largely to combat permanent failures. A synthesis of the two approaches may yield an even more versatile fault tolerant design.

REFERENCES

1. J. von Neuman, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", Annuals of Mathematical Studies, Princeton University Press, Vol. 54, pp. 43-98, 1956.
2. J. H. Wensley, et.al. "SIFT: Design and Analysis of a Fault Tolerant Computer for Aircraft Control:, Proc. IEEE, Vol. 66, No. 10, Oct. 1978, pp. 1240-1255.
3. A. Avizienis, "Design of Fault Tolerant Computers", Proc. AFIPS 1967 FJCC, Vol. 31, pp. 733-743, 1967.
4. A. E. Cooper and W. T. Chow, "Development of on-board Space Computer Systems", IBM Journal of Research Development, Vol. 20, No. 1, pp. 5-19, Jan. 1976.
5. F. P. Mathur and P. T. deSousa, "Reliability Models of NMR Systems", IEEE Transactions on Reliability, Vol. R-24, pp. 108-113, 1975.
6. W. W. Peterson and E. J. Weldon, Error Correcting Codes, MIT Press, Cambridge, MA, 1971.
7. R. A. Short, "The attainment of reliable digital systems through the use of redundancy--A survey", IEEE Computer Group News, Vol. 2, pp. 2-17, Mar. 1968.

8. D. K. Pradhan and J. J. Stiffler, "Error Correcting Codes and Self Checking Circuits", Computer, Vol. 13, pp. 27-38, March 1980.

9. W. G. Bouricius, et.al., "Reliability Modeling Techniques for Self-Repairing Computer Systems", Proceedings ACM Ann. Conf., San Francisco, CA, August 1969, pp. 295-309.

10. S. J. Bavuso, J. J. McGough and F. Swern, "Latent Fault Modeling and Measurement Methodology for Application to Digital Flight Control". Advanced Flight Control Symposium USAF Academy, Colorado Springs, CO, August 1981.

11. J. J. Stiffler, "Fault Coverage and the Point of Diminishing Returns", Journal of Design Automation and Fault Tolerant Computing, Vol. 2, No. 4, October 1978.

12. W. N. Toy, "Fault-Tolerant Design of Local ESS Processors", Proceedings of the IEEE, Vol. 66, No. 10, Oct. 1978, pp. 1126-1145.

13. F. P. Mathur and A. Avizienis, "Reliability Analysis and Architecture of a Hybrid-Redundant Digital System", AFIPS Conf. Proc., Vol. 36, pp. 375-383, May 1970.

14. P. T. deSousa and F. P. Mathur, "Sift-Out Modular Redundancy", IEEE Transaction on Computers, Vol. C-27, No. 7, pp. 624-627, July 1978.

15. J. Losq, "A Highly Efficient Redundancy Scheme: Self-Purging Redundancy", IEEE Transactions on Computers, Vol. C-25, pp. 569-578, June 1976.

16. M. M. Dickinson, et.al., "Saturn V Launch Vehicle Digital Computer and Data Adapter", AFIPS Conf. Proc., Vol. 26, 1964 FJCC, pp. 501-516.

17. A. A. Avizienis, et.al., "The STAR (Self-Testing and Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design", IEEE Trans. Computers, Vol. C-20, No. 11, Nov. 1971, pp. 1312-1321.

18. J. J. Stiffler, et.al., "CARE II Final Report Phase I", Vols. 1 and 2, NASA GR-159123, 1979.

19. A. L. Hopkins, T. Basil Smith and J. H. Lala, "FTMP - A Highly Reliable Fault Tolerant Multiprocessor for Aircraft", Proceedings of the IEEE, Vol. 66, No. 10, October 1978.

20. D. P. Siewiorek, V. Kini, H. Mashburn, S. R. McConnel and M. Tsao, "A Case Study of C.mmp,Cm*, and C.vmp: Part I--Experiences with Fault Tolerance in Multiprocessor Systems", Proceedings of the IEEE, Vol. 66, No. 10, pp. 1178-1199, October 1978.

21. D. Tasar and V. Tasar, "A Study of Intermittent Faults in Digital Computers", AFIPS Conf. Proc., 1977 NCC, pp. 807-811.

22. A. Avizienis, "Fault Tolerant Systems", IEEE Transactions on Computers, Vol. C-25, No. 12, December 1976, pp. 130H-12.

23. W. M. Daly, et.al., "A Fault-Tolerant Digital Clocking System", Digest of Papers--1973 Int'l Symp. Fault-Tolerant Computing, Palo Alto, CA, June 1973, pp. 17-22.

24. J. G. Kemeny and J. L. Snell, Finite Markov Chains, D. Van Nostrand Company, Inc. Princeton, NJ, 1969.

25. C. L. Liu, Introduction to Combinatorial Mathematics, McGraw-Hill, New York, 1968.

26. H. C. Rickers and P. F. Manno, "Microprocessor and LSI Microcircuit Reliability-Prediction Model", IEEE Transactions on Reliability, Vol. R-29, No. 3, August 1980.

27. B. L. Amstadter, "Reliability Mathematics, McGraw-Hill, New York, 1971.

28. S. R. McConnel, D. P. Siewiorek and M. M. Tsao, "The Measurement and Analysis of Transient Errors in Digital Computer Systems" Proceedings Ninth International Symposium on Fault Tolerant Computing, IEEE, New York, 1979, pp. 67-70.

29. X. Castillo and D. P. Siewiorek, "A Workload Dependent Software Reliability Prediction Model", Proceedings 12th International Symposium on Fault Tolerant Computing, IEEE, New York, 1982, pp. 279-286.

The vita has been removed from the scanned document

THE DESIGN OF PERIODICALLY SELF RESTORING REDUNDANT SYSTEMS

by

Adit D. Singh

(ABSTRACT)

Most existing fault tolerant systems employ some form of dynamic redundancy and can be considered to be incident driven. Their recovery mechanisms are triggered by the detection of a fault. This dissertation investigates an alternative approach to fault tolerant design where the redundant system restores itself periodically to correct errors before they build up to the point of system failure. It is shown that periodically self restoring systems can be designed to be tolerant of both transient (intermittent) and permanent hardware faults. Further, the reliability of such designs is not compromised by fault latency.

The periodically self restoring redundant (PSRR) systems presented in this dissertation employ, in general, N computing units (CU's) operating redundantly in synchronization. The CU's communicate with each other periodically to restore units that may have failed due to transient faults. This restoration is initiated by an interrupt from an external (fault tolerant) clocking circuit. A reliability model for such systems is developed in terms of the number of CU's in the system, their failure rates and the frequency of system restoration. Both transient and permanent faults are considered. The model allows the estimation of system reliability and mean time to failure. A restoration al-

gorithm for implementing the periodic restoration process in PSRR systems is also presented. Finally a design procedure is described that can be used for designing PSRR systems to meet desired reliability specifications.