12-1-1993

Reliability analysis of triple modular redundancy system with spare

Khalid A. Al-Kofahi

Reliability Analysis of Triple Modular Redundancy System with Spare

by

Khalid A. Al-Kofahi

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved By : Graduate Advisor Dr. P.V. Reddy

Department Chairman - Dr. Roy Czernikowski

Professor of Electrical Engineering Dr. J. E. Palmer

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK
December 1993

Title of thesis

Reliability Analysis of Triple Modular Redundancy
System with Spare

I Khalid A. Al-Kofahi hereby grant permission to the Wallace Memorial Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: 12 /22 / 1993 Signature of Author

Abstract

Hardware redundant fault-tolerant systems and the different design approaches are discussed. The reliability analysis of fault-tolerant systems is usually done under permanent fault conditions. With statistical data suggesting that up to 90% of system failures are caused by intermittent faults, the reliability analysis of fault-tolerant systems must concentrate more on this class of faults. In this work, a reconfigurable Triple Modular Redundancy (TMR) with spare system that differentiates between permanent and intermittent faults has been built. The reconfiguration process of this system depends on both the current status of its modules and their history. Based on this, a different approach for reliability analysis under intermittent fault conditions using Markov models is presented. This approach shows a much higher system reliability compared to other redundant and non-redundant configurations.

Table Of Contents

Table Of Contents *(continued)*

List of Figures

List of Figures *(continued)*

CHAPTER ONE
INTRODUCTION

1.1 INTRODUCTION

The widespread use of computers in almost every aspect of life motivates the need for more reliable computers, especially in such applications where computer failures may cause great financial or human tragedies. Although practicing more conservative design approaches and using more reliable hardware and software components, does increase the reliability of computer systems, computer failures still happen. These failures are caused by different factors, from harsher environmental conditions to user abuse. Even in a favorable environment, computer systems nowadays are much more sophisticated and contain a larger number of hardware and software components, which are bound to fail, making the overall probability of a system failure even larger.

With faults being unavoidable, the trend is to design computers that can tolerate faults and prevent them from causing errors and system failures. Before discussing some of the fault-tolerant computing techniques, one should understand common faults, their types, distribution, causes, and extents.

1.2 Faults: Types, Causes, and Distribution

A fault is a physical defect, or an erroneous state of hardware or software components, which may cause failure or system error.

When designing a fault-tolerant system it is important to identify the faults that may occur, their types, causes, effects, extents and distribution. Then designers should decide what action(s), if any, should be taken in respose to a fault. This decision depends on many factors such as: system failure cost versus fault-tolerant design cost, repair cost, application cruciality, fault distribution, etc.

Faults may be permanent, intermittent, or transient. Permanent faults result in forcing the system, or part of it, into a faulty state. Intermittent and transient faults occur occasionally caused by unstable hardware or software conditions.

Causes of Faults

Faults have different causes. Understanding these causes enables the designers to anticipate and hence, tolerate them. The first cause of faults is incomplete, vague, or incorrect hardware and/or software specifications. The second cause of faults is implementation mistakes. These mistakes happen during the translation of hardware and software specifications into a system. The third and most crucial cause of faults is hardware defects. The fourth cause of faults is external factors, such as harsher environmental conditions, temperature extremes, fluctuating in the supply voltage, interfaces, user abuse or mistakes, etc.

Faults Distribution

Faults happen at any time of system operation. But different types of faults are dominant at different stages of the system life. As shown in figure 1.1, a system life can be divided into three stages. The first stage is the infant mortality period. During this period systems usually have high failure rate due to either component defects or manufacturing mistakes. It is a common procedure to burn in the systems before puting them into operation. The next stage is the normal life period. This period is characterized by a constant failure rate. The last stage is the wear-out period, where the failure rate starts increasing again due to hardware aging. Of course, the boundaries between these stages are not clear cut and may differ from one system to another. Harsher environmental conditions, for example, may cause the system to wear-out earlier. It is important to notice that fault-tolerant systems are usually designed to tolerate faults only during their normal life period.

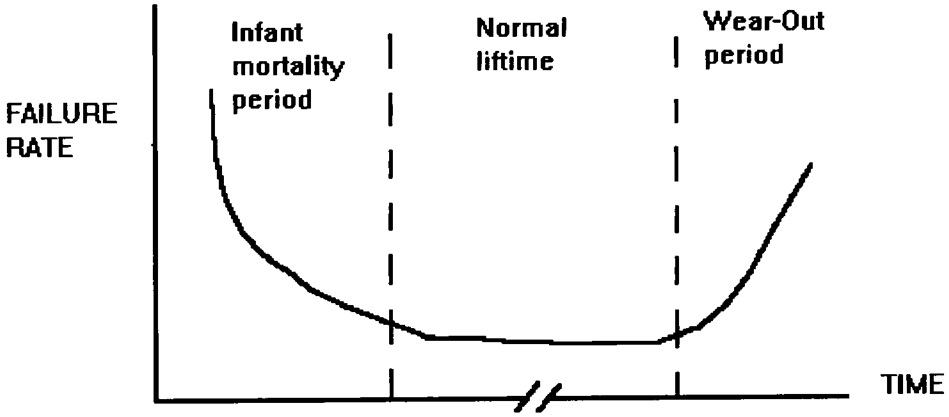


Figure 1.1

Bathtub curve describing failure rates as a function of time

1.3 Fault Tolerance

Fault-tolerance is the system ability to continue its operation correctly despite the existence of a fault(s).

Although fault-tolerant systems are usually described as being either highly reliable or highly available, there are other attributes to fault-tolerant systems. Such as performance, safety, maintainability, and/or testability. Fault-tolerant systems may be designed to achieve some or all of these requirements.

1.3.1 Reliability : R(t)

Reliability is the conditional probability that a system will perform correctly during the time interval [0, t], given that it was operational at the beginning of the interval (t = 0).

Reliability is considered to be the most important factor in applications where system failures are not acceptable, either because of their consequences, or because systems cannot be repaired, as in satellites. Highly reliable systems usually contain some redundant hardware, which will enable the system to continue its operation without interruption upon system failure (as in the case of hot standby systems).

1.3.2 Availability : A(t)

Availability is the probability that a system is operational correctly at any time t.

The goal here is to make the system as available to the user as possible. Availability is typically used as a figure of merit in systems where short duration failures do not have serious consequences. Because availability can be defined as operation time divided by total life time, a system can be highly available while having frequent failures, as long as these failures have short duration and repair times. The use of spares during the system down and repair times is very common in highly available systems.

1.3.3 Performability : P(L,t)

> Performability is the probability that a system performance is at, or higher than some level L at time t.

One of the drawbacks of some fault-tolerance techniques is lower system performance; this is obvious in majority voting systems where some processor time must be wasted to synchronize the processors. This measure is used to ensure that the system performance does not fall below a certain level L.

1.3.4 Safety : S(t)

> Safety is the probability that a system will not fail into a state that may disturb the operation of other related systems, or endanger the people associated with it.

Fault-tolerant systems differ in the way they respond to a fault. Most of fault-tolerant systems follow more conservative design approaches, pass through different quality control tests (to avoid faults caused by specification and implementation mistakes), and are designed to handle harsher environmental conditions and external disturbances. With this being done, designers are left with two choices in dealing with faults: either mask them or tolerate them. Fault Masking is the process of preventing faults from causing errors and system failures. Majority voting systems are a typical example of such technique; another example is the use of error detecting and correcting codes. Fault Tolerance, on the other hand, requires fault detection, location, confinement, and recovery (usually through reconfiguration). In either case, some form of redundancy is required.

1.4 Redundancy

> Redundancy is the addition of extra hardware, software, information, or repetition that is not needed for normal system operation.

The addition of redundant resources does not come free and may degrade the system performance, especially in the case of software and time redundancy. Therefore, a trade off between the redundant design cost versus the system failure cost must be made to decide what form and level of redundancy is needed. In this section we will breifly discuss the different kinds of redundancy that are commonly used.

Software Redundancy

Software redundancy is the use of extra software beyond the system's normal operation need. One example is the software added to produce error correcting and detecting codes.

Information redundancy

Information redundancy can be seen in all error correcting and detecting codes, where extra information, parity, check sum, m-of-n codes, duplication of words, etc, are added for the purpose of fault-tolerance. It is also worth mentioning that information redundancy involves both software and hardware redundancy.

Time Redundancy

Time Redundancy is useful in systems where speed is less important, or in applications that do not form a computational challenge to the system and do not require a short response time. The basic idea of this form of redundancy is the repetition of computation in a way that will detect faults. For example, faults with short duration (intermittent and transient) can be detected if the computation is repeated at different times.

Hardware Redundancy

Hardware redundancy is becoming more popular due to the decreasing cost, size, and power requirements of hardware components. Hardware redundancy is used to mask faults and prevent them from generating errors, or to detect, locate, and recover from faults. The earliest (and maybe the most common) form of this redundancy is the Triple Modular Redundancy (TMR).

1.4.1 Triple Modular Redundancy : TMR

Triple modular redundancy is the first form of hardware redundancy techniques, introduced by J. Von Neuman in 1956. Figure 1.2 shows the basic configuration of this form. The output in such designs agrees with the majority (2-out-of-3) of the processors (or modules). This means that the system can tolerate (mask) single module failures only. The reliability of such designs cannot be higher than the voter reliability. A voter failure is considered to be a common point failure. To overcome this problem the voter can be triplicated as shown in figure 2.3.

Another majority voting technique is the N-Modular Redundancy, NMR, which is the general case of TMR. In this method N modules (usually are odd number) are used instead of three to enable the system to tolerate more than a single module failure.

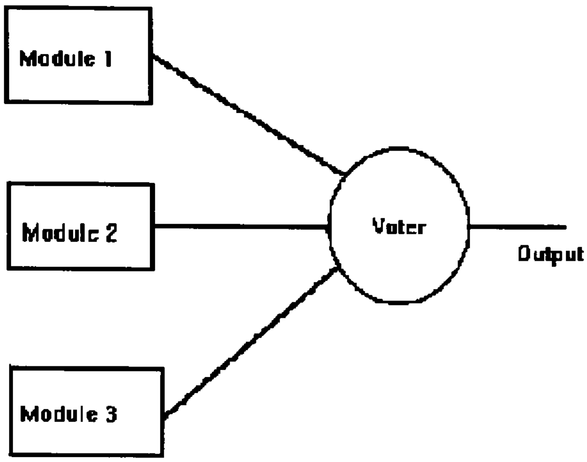


Figure 1.2

TMR basic diagram

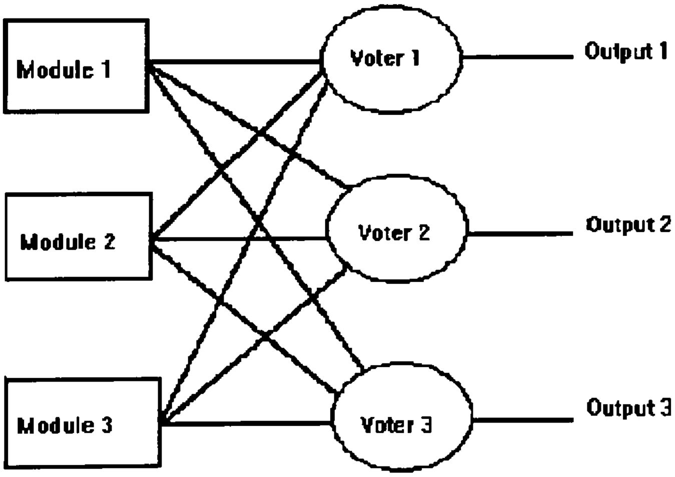


Figure 1.3

TMR with Triplicate voters

Another common technique is the TMR with spare, as shown in Figure 1.4. It consists of a TMR system, spare, switching circuitry, and some fault detection and location hardware. The basic modules start the voting process; then, upon a module failure, the spare will be considered in the voting process. If these modules are isolated from each other, the failed module can be replaced or repaired without interrupting the system's operation. Obviously this system is more reliable and available than a regular TMR (as we will see in later chapters).

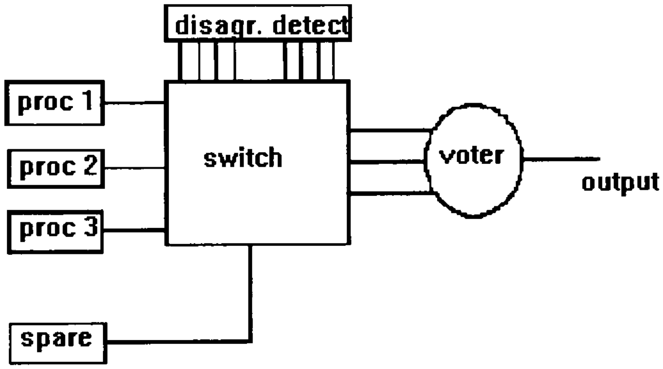


Figure 1.4

A TMR System with Spare

1.5 Problems with majority voting techniques

In addition to common point failure, majoty voting techniques have other major problems. For example, adding redundant modules does not necessarily improve the reliability of a system over its simplex counterpart as one would first expect. To illustrate this, consider a TMR system. The system is functioning correctly if:

all three modules are functioning correctly,

or two of the modules are operational and one is not.

So, if we denote the system's reliability by R_{sys} , and the module's reliability by R, then

$$
\begin{aligned}
R_{sys} &= R^3 + (\text{ 3 choose 2 }) R^2 (1-R) \\
&= R^3 + (3! / (2! * 1!)) R^2 (1-R) \\
&= 3R^2 - 2R^3 \qquad (1.1)
\end{aligned}
$$

And the crossover (intersection) point will be

$$
\begin{aligned}
Rsys &= R \\
0 &= 3R^2 - 2R^3 - R
\end{aligned}
$$

solving yields:

$R = 0, 1/2, \text{ or } 1.$

The above result suggests that, using three modules with reliability of 0, 1/2, or 1 in a TMR system will not improve the system's reliability over its simplex counterpart at all. Furthermore, using modules with reliability less than 0.5 in a TMR system will worsen the overall system reliability with respect to its simplex counterpart. Figure 1.5 shows the results of the same analysis for different majority voting systems.

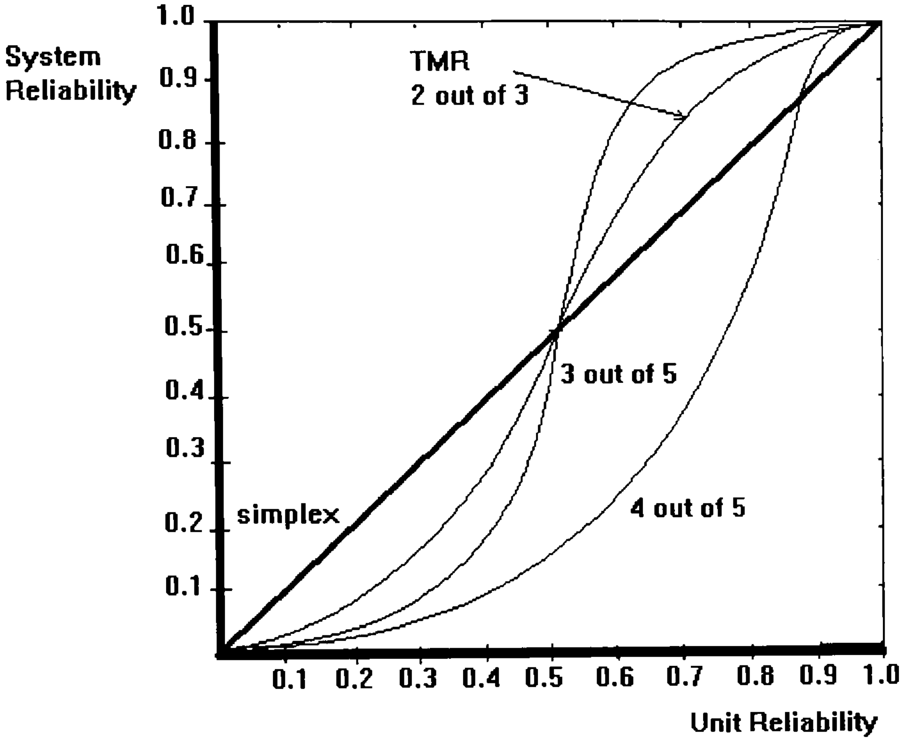


Figure 1.5

Unit Reliability versus System Reliability

(B. S. Dhillon 1987)

CHAPTER TWO
THEORETICAL BACKGROUND

2.1 INTRODUCTION

Evaluation measures are needed to determine whether a fault-tolerant system will achieve its goals without actually building it. These measures will also help designers decide what fault-tolerant techniques are most suitable for their applications. Section 2.2 introduces some of the evaluation measures available. Sections 2.3, 2.4, and 2.5 introduce some of the most common tools used to analyze fault-tolerant systems. Section 2.6 is a comparison between different fault-tolerant systems. Finally, section 2.7 discusses the effect of intermittent faults on fault-tolerant systems.

2.2 EVALUATION MEASURES

Several measures are available for evaluating fault-tolerant systems. These measures can be divided into two categories: quantitative and qualitative. Although qualitative measures are, to an extent, subjective in nature, quantitative measures give numbers that can be used to compare different systems. Usually a collection of these measures are needed to fully describe a system. The most common measures are: Failure Rate, Reliability, Availability, Mission Time, Mean Time To Failure, Mean Time Between Failure, Mean Time To Repair, Fault Coverage, Safety, and Cost.

2.2.1 Failure Rate Z(t)

Failure rate, or hazard rate, is defined as the total number of system failures per time period. During the system normal life time failures have a constant rate of occurrence (see figure 1.1, bath-top curve), hence Z(t) = Lambda (L). The most common technique used for estimating failure rates is the United States Department of Defense (USDOD) MIL-HDBK-217 standards, which predicts the constant failure rate of an Integrated Circuit (IC) to be:

$$\mathbf{L = F_L\, F_Q(C_1\, F_T + C_2\, F_E)\, F_P} \quad \text{failures per million hours.}$$

Where,

F_L : Learning Factor, this factor represents the level of confidence in the fabrication process. Devices fabricated using a new and yet unproved process are assigned a learning factor of 10, while those produced using a proven process are assigned a factor of 1.

F_Q Quality Factor, this factor represents the level of the device screening and testing. Typical values vary from1 to 300.

F_T : Temperature Factor, this factor depends on the device technology, operating temperature, and power dissipation. For example, the temperature factor (F_T) for bipolar circuits is given by :

$$F_T = 0.1\, e^{-4794\left(\left(\frac{1}{T_j+273}\right)-\left(\frac{1}{298}\right)\right)}$$

Where,

T_j Junction temperature, in degrees Celsius.

F_E Environmental Factor, this factor represents the harshness of the operating environment. Typical values vary form 0.2 to 10.0

F_P Pin Factor, this factor is a function of the number of pins on the IC. Typical values ranges from 1.0 to 1.2.

C_1, C_2 : Complexity Factors, these factors are functions of the number of gates in a logic circuit.

Table 2.1 shows some failure rate values computed using MIL-HDBK-217B standards.

No. of Logic gates	Failure rate (Failures/million hours)
50	0.1527
100	0.2312
200	0.3655
500	1.4483
1000	14.4880

Table 2.1, Failure rates calculated using MIL-HDBK-217B ($F_L = 1$, $F_Q = 16$, $F_T = .35$, $F_E = 0.2$, $F_P = 1$). Johnson 1989.

2.2.2 Reliability R(t)

Recall that Reliability is the conditional probability that a component (or a system) will operate correctly throughout the interval $[t_0,t]$ given that it was operational at time t_0. Consider a system put into operation at time t_0 and tested at time t.

Let, **N** be the total number of system components.

N_f(t) be the number of failed components at time t.

N_0(t) be the number of operating components at time t.

Then,

$$R(t) = \frac{N_o(t)}{N}$$

and

$$R(t) + \frac{N_f(t)}{N} = 1$$

$$R(t) = 1 - \frac{N_f(t)}{N}$$

$$\frac{dR(t)}{dt} = \frac{-1}{N}\frac{dN_f(t)}{dt}$$

or

$$\frac{dN_f(t)}{dt} = -N\frac{dR(t)}{dt}$$

Now, since Z(t) = (1/**N**$_0$(t)) d**N**$_f$(t)/dt

then,

$$Z(t) = \frac{-N}{N_o(t)}\frac{dR(t)}{dt}$$

$$= \frac{-1}{R(t)}\frac{dR(t)}{dt}$$

hence,

$$\frac{dR(t)}{dt} = -Z(t)\,R(t), \quad \text{substituting L for Z(t) yields:}$$

$$\frac{dR(t)}{dt} = -L\,R(t)$$

Solving yields:

$$R(t) = e^{-Lt} \qquad (2.1)$$

Equation (2.1) is known as the exponential failure law.

2.2.3 Mean Time To Failure (MTTF)

Mean time to failure Is the expected operation time of a system before its first failure. **MTTF** can be measured experimentally. For example, consider N identical systems put into operation at time t_0, and at time t_i system i encounters its first failure, then

$$\text{MTTF} = (t_1 + t_2 + \ldots + t_N) / N$$

Or, let f(t) be the failure density function, then

$$MTTF = \int_0^\infty t \; f(t)\, dt$$

Using integration by parts and the fact that f(t) = d(1- R(t))/dt, yield:

$$MTTF = \int_0^\infty R(t)\, dt$$

As an example, Consider a simplex system with

$$R(t) = e^{-Lt}$$

The MTTF for this system is:

$$\text{MTTF} = 1/L \qquad (2.2)$$

Mean Time To Repair & Mean Time Between Failures (MTTR & MTBF)

Repair rate (MTTR) is defined as the average number of repairs per hour. Although the expected value of repair rate cannot be found directly as failure rate, it is a common assumption in systems with small failure rates, that repair rate = failure rate. Now, if we denote repair rate by m then,

$$MTTR = 1/m$$

and

$$MTBF = MTTF + MTTR$$

2.2.4 Fault Coverage

Fault coverage is a measure of the system's ability to detect, locate, confine, and recover from faults. The most important aspect of fault coverage is the fault recovery coverage, which is sometimes used to denote fault coverage in general. Mathematically, it is defined as the conditional probability that the system will recover given that a fault exists, or,

$$C = P\text{(fault recovery | fault exists)}$$

Fault coverage is not easy to calculate, because it usually requires developing a list of all possible faults and then deciding what factor of these faults can be detected, located, confined, and recovered from. This may require exhaustive testing of the system with a very large number of test vectors.

2.2.5 Mission Time MT(r)

Mission time is an estimate for the time at which the system's reliability falls below some level r. For example, a non-redundant system that follows the exponential failure law has a reliability

$$R(t) = e^{-Lt}$$

To find MT(r) for this system, set $r = e^{-Lt}$ and solve for t. Solving yields:

$$t = \frac{-\ln(r)}{L}$$

or,

$$MT(r) = \frac{-\ln(r)}{L} \qquad (2.3)$$

A simple example will show the importance of this measure. Consider a non-redundant system with failure rate L = 0.002 failures/hour. The mission time for this system at a reliability level of 0.95 is:

$$MT(0.95)_{simplex} = [\ -\ln\ (0.95)\ ]\ /\ 0.002 = 25.64 \text{ hours.}$$

Now, consider a TMR system with the same failure rate. The system's mission time at the same reliability level as before is (R(t) is given by equation 1.1):

$$0.95 = 3\ e^{(-0.004\ t)} - 2e^{(-0.006\ t)}$$

Solving for t gives:

$$MT(0.95)_{TMR} = 145 \text{ hour.}$$

The previous result states that at a failure rate of 0.002 failure/hour. a TMR system is expected to operate 5.45 times longer than a single-module simplex system before its reliability falls bellow 0.95

2.3 RELIABILITY MODELING

Loosely used to denote evaluation criteria for fault tolerant systems, reliability is one of the most important system attributes. System reliability can be measured experimentally (as seen earlier). But this requires the availability of a sufficiently large population of such systems, and one may wait for years for the expected failures to happen, which is totally impractical. Hence the importance of reliability analysis. Reliability analysis can be done under various assumptions, such as failure to exhaustion and failure with repair. Failure to exhaustion assumes that all system components (modules) fail before any repair can take place. Systems operating under this assumption can be modeled using combinatorial modeling techniques. Failure with repair, on the other hand, involves the modeling of two concurrent processes, the failure process and the repair process. Markov modeling is one of the most popular techniques for this kind of system.

2.3.1 Combinatorial Modeling

Combinatorial modeling divides the system into non-overlapping modules. Each module is assigned a probability of working P_i (or $R_i(t)$), then some probabilistic techniques are used to enumerate all possible ways for the system operation. System reliability is defined as the sum of the modules' reliabilities in all these different ways. This technique makes the following assumptions :

1. Module failures are independent.
2. Failed modules yield incorrect results.
3. The system fails when all working modules do not form a way that is sufficient for system operation.
4. A failed system cannot return to correct operation by any further failures.

These assumptions are suitable for modeling random hardware failures in a system. But when failures are caused by some global factors, the first assumption, for example, is not accurate. To analyze systems reliabilities, combinatorial modeling categorize systems into series and parallel systems, and a combination of these.

Series Systems

A series system can be seen as a system that has no redundancy at all, where all system modules are necessary for correct system operation. Consider the series system shown in figure 2.1. Its reliability is given by:

$$R(t)_{series} = R_1(t)\, R_2(t) \,.....R_{N-1}(t)\, R_N(t) \qquad (2.4)$$

Furthermore, if each module satisfies the exponential failure law, then

$$R(t)_{series} = e^{-L_1 t} + e^{-L_2 t} + \ldots.. + e^{-L_N t}$$

or,

$$R(t)_{series} = e^{-L_{system}t}$$

where,

$$L_{system} = L_1 + L_2 + ... + L_N$$

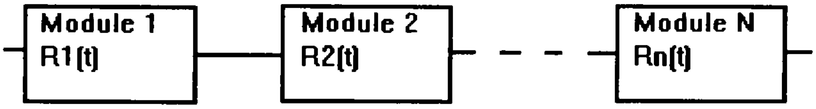


Figure 2.1 A series system of N modules

Parallel Systems

A parallel system of N modules (such as the one in figure 2.2) can be seen as a system with a redundancy level of (N 1), where one operating module is sufficient for correct system operation. To analyze such systems, let us define the unreliability of module i to be, $Q_i(t) = 1 - R_i(t)$.

It is obvious that the system of figure 2.2 will fail if and only if all its N modules fail, or,

$$Q_{Parallel}(t) = Q_1(t)\ Q_2(t) \ldots\ldots\ Q_N(t)$$

Or,

$$R_{Parallel}(t) = 1 \quad Q_{Parallel}(t)$$
$$= 1 - [\ (1-R_1(t)\ )\ (1-R_2(t)\ ) \ldots (1-R_N(t)\ )] \qquad (2.5)$$

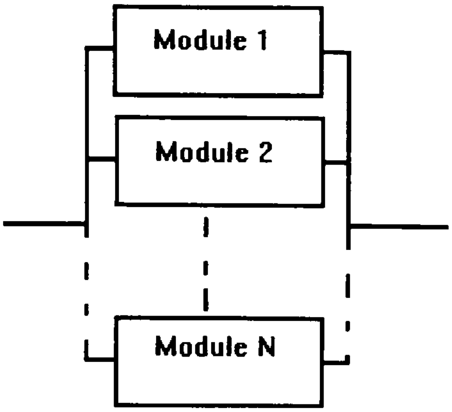


Figure 2.2 A parallel system of N modules

Series/Parallel Systems

Some fault-tolerant systems can be modeled as a combination of series and parallel systems. Modules with high failure rates or those more critical to the application are usually configured in parallel. Those with infrequent failures or those with tolerable failure results are usually configured in series. The reliability of a system depends (in addition to other factors) on the way its modules are configured. To illustrate this, consider a redundant system consisting of two processors, A & B, and two memory modules, C & D, with one processor and one memory module being needed for system operation. Figure 2.3 shows two ways of configuring these modules.

- The system in figure 2.3-a requires the combination of either A-C, or B-D for its correct operation, hence represents two series modules, A-C and B-D, configured in parallel.

$$R_{fig2.3\text{-}a}(t) = 1\text{-}\ [(\ (1\text{-}R_A(t)\ R_C(t))\ )(\ 1\text{-}\ (R_B(t)\ R_D(t))\ )]$$

- The system in figure 2.3-b requires the combination of any processor and any memory modules for its correct operation, and so it represents two parallel modules configured in series.

$$R_{fig2.3\text{-}b}(t) = [\ 1\text{-}\ (1\text{-}R_A(t))(1\text{-}R_B(t))\ ][\ 1\text{-}\ (1\text{-}R_C(t))(1\text{-}R_D(t))\ ]$$

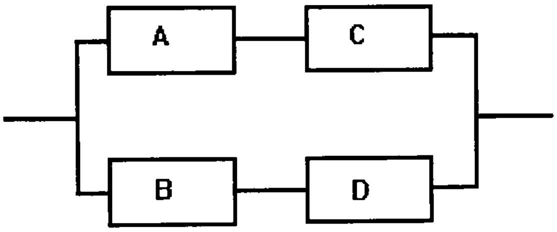


Figure 2.3-a Parallel series configuration

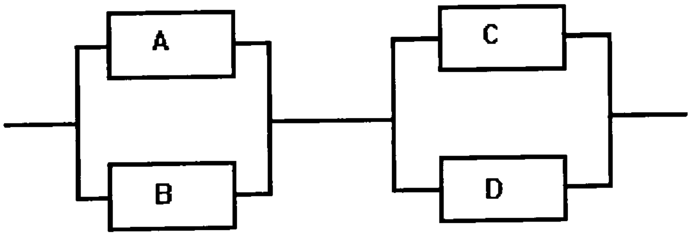


Figure 2.3-b Series parallel Configuration

To simplify the comparison, assume that

$$R_A(t) = R_B(t) = R_C(t) = R_D(t) = R$$

then,

$$
\begin{aligned}
R_{fig2.3\text{-}a} &= 1 - (1-R^2)(1-R^2) \\
&= 2R^2 \quad R^4
\end{aligned}
$$

and,

$$
\begin{aligned}
R_{fig2.3\text{-}b} &= [\,1 - (1-R)(1-R)\,][\,1 - (1-R)(1-R)\,] \\
&= [\,2R \quad R^2\,][\,2R - R^2\,] \\
&= 4R^2 - 4R^3 + R^4
\end{aligned}
$$

It is obvious that the reliability of the system in figure 2.3-b ($R_{fig2.3\text{-}b}$) is larger than that of figure 2.3-a ($R_{fig2.3\text{-}a}$). This result shows that the reliability of a fault tolerant system is depends on its configuration.

Modeling a TMR System

A TMR system can be modeled using combinatorial modeling techniques by enumerating all possible ways for system operation. Consider a TMR system with three modules A, B, and C configured in a majority voting fashion. The system requires any two of these to be operational for its correct operation. Assuming that the voter has a reliability of 1.0, the TMR reliability is given by :

$$
\begin{aligned}
R_{TMR}(t) = {} & R_A(t)\, R_B(t)\, R_C(t) + R_A(t)\, R_B(t)\,(1-R_C(t)) \\
& + R_A(t)\, R_C(t)\,(1-R_B(t)) + R_B(t)\, R_C(t)\,(1-R_A(t))
\end{aligned}
$$

Now, if $\quad R_A(t) = R_B(t) = R_C(t) = R(t)$

then, $\quad R_{TMR}(t) = 3R^2(t) - 2R^3(t)$

The above result agrees with the formula derived earlier (equation 1.1). We can follow the same analysis to find the reliability of any N-out-of-M system, where N operational modules are required for correct system operation.

In addition to the assumption that module failures are independent, which is inaccurate in some cases, combinatorial modeling has other problems. One of the major problems is the perfect fault coverage assumption, which means that the detection of a failed module in the system has a probability of 1.0. Another problem is the assumption that the reconfiguration process is also a perfect one and happens in zero time units. Furthermore, the modeling of complex systems can be extremely difficult. Finally, combinatorial modeling techniques cannot model systems with repair and sometimes require very restrictive assumptions.

2.3.2 Markov Modeling

Markov modeling is a powerful technique for analyzing systems. The basic concepts of the Markov process model are the system state and the state transition. The system state fully describes the system status at any given instant of time. The state transition describes the behavior of the system as modules fail and are repaired. A system of N modules with each module being either working or in failure will have 2^N states. There are two types of Markov models: discrete time models and continuous time models. Discrete time models assume that all state transitions occur at fixed intervals of time. Continuous time models allow state transitions to occur in a random fashion. In this section the term Markov model refers to continuous time model.

One of the most important assumptions in Markov modeling is that the state transitions depend only on the current state. This means that the time spent in a given state does not affect the probability of the next transition or the probability of remaining in the current state. Furthermore, failure rates are constant and do not depend on the time spent in any state. Thus the model agrees with the exponential failure law.

As an example, consider a non-redundant system consisting of one module. If the system has a constant failure rate L (obeys the exponential failure law), then given that the system was operational at time t, the probability of system failure at time t+dt is:

$$1 - e^{-Ldt}$$

Substituting the exponential series expansion for the exponential term above yields:

$$-e^{-Lt} = 1-[\,1+(-Ldt)+\frac{(-Ldt)^2}{2!}+\ldots]$$

And for small values of dt, the above expression reduces to

$$-e^{-Lt} \approx Ldt$$

Therefore, the probability of system failure within the time period dt is approximately L dt Figure 2.4 is a graphical representation of our simplex system with failure rate L and repair rate m. The state probabilistic equations for the system in figure 2.4 are:

$$P_1(t + dt) = (1 - Ldt)\, P_1(t) + mdt\, P_2(t)$$

$$P_2(t + dt) = (1 - mdt)\ P_2(t) + Ldt\ P_1(t)$$

Where, $P_i(t)$ is the probability that the system is at state i at time t, and $P_i(t + dt)$ is the probability that the system is at state i at time t + dt.

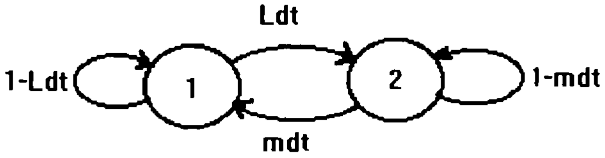


Figure 2.4 Two state differential Markov model

Rearranging the above equations and dividing by dt produce:

$$[\ P_1(t + dt)\quad P_1(t)\ ] / dt \ = \ - L\ P_1(t) + \ m\ P_2(t)$$
$$[\ P_2(t + dt) - P_2(t)\ ] / dt \ = \ L\ P_1(t) \quad m\ P_2(t)$$

Taking the limit as dt approaches zero produces the following simultaneous differential equations:

$$\frac{dP_1(t)}{dt} = -L P_1(t) + mP_2(t)$$
$$\frac{dP_2(t)}{dt} = \ LP_1(t) - mP_2(t)$$

These equations are known as Chapman-Kolmogorov equations. They can be written directly from the transition diagram without the self loops. Consider figure 2.5, in which the change in state (1) is the flow coming from state (2) times the probability of being at state (2), minus the flow out of state (1) times the probability of being at state (1).

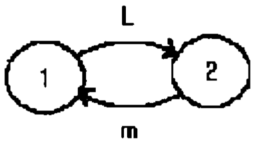


Figure 2.5

Two state Continuous time Markov model

The previous equations can be solved using LaPlace transforms. Performing LaPlace transforms gives:

$$sP_1(s) - P_1(0) \;=\; -LP_1(s) + mP_1(s)$$
$$sP_2(s) - P_2(0) \;=\; LP_1(s) - mP_1(s)$$

Since the system is assumed to be operational at time t = 0, then $P_1(0) = 1$, and $P_2(0) = 0$. Substituting these values in the above equations yields:

$$(s + L)\, P_1(s) \qquad m \quad P_2(s) \;=\; 1$$
$$-L \quad P_2(s) \;+\; (s + m)\; P_2(s) \;=\; 0$$

Solving for $P_1(s)$ and $P_2(s)$ gives:

$$P_1(s) = \frac{s+m}{s^2 + Ls + ms}$$
$$P_2(s) = \frac{L}{s^2 + Ls + ms}$$

Performing the partial fraction expansions yields:

$$P_1(s) = \frac{m/(L+m)}{s} + \frac{L/(L+m)}{s+L+m}$$

$$P_2(s) = \frac{L/(L+m)}{s} + \frac{L/(L+m)}{s+L+m}$$

Performing the inverse LaPlace transforms yields:

$$P_1(t) = \frac{m}{L+m} + \frac{L}{L+m} e^{-(L+m)t}$$

$$P_2(t) = \frac{L}{L+m} + \frac{L}{L+m} e^{-(L+m)t} \qquad (2.6)$$

Equations 2.6 describe the probabilities of the two system states. $P_1(t)$ is the probability that the system is operational at any time t (or known as the system's Availability). $P_2(t)$ is the probability that the system is in a failed state at any time t. One interesting feature of these equations is that both of them approach a constant value as t approaches infinity

$$P_1(\text{infinity}) = m / (L+m)$$

$$(2.7)$$

$$P_2(\text{infinity}) = L / (L+m)$$

P_1(infinity) is known as the steady state availability $A_{ss}(t)$. Furthermore, if we are only interested in the system's steady state status, the state equations can be rewritten as

$$0 = -LP_1 + mP_2$$

$$0 = LP_1 - mP_2$$

Solving these equations with the extra condition ($P_1 + P_2 = 1$), gives us the same result as those in equations (2.7).

To calculate the system's reliability, we need to modify the Markov model of figure 2.5 such that the system's failed state is a trap state (i.e. no repair is taking place). Doing so produces the following equations :

$$\frac{dP_1(t)}{dt} = -L\,P_1(t)$$

$$\frac{dP_2(t)}{dt} = L\,P_1(t)$$

Performing LaPlace transforms produces:

$$P_1(s) = \frac{1}{s+L}$$

$$P_2(s) = \frac{L}{s(s+L)}$$

Performing partial fraction expansions and the inverse LaPlace transforms yield:

$$R(t) = P_1(t) = \mathrm{e}^{-Lt}$$

$$(t) = P_2(t) = 1 - \mathrm{e}^{-Lt}$$

Notice that the above results agree with our original assumption that the system obeys the exponential failure law.

Modeling a TMR System

Consider a TMR system, where only two of the three processors are necessary for correct system operation. The Markov model for such a system with failure rate L and no repair is shown in figure 2.6.

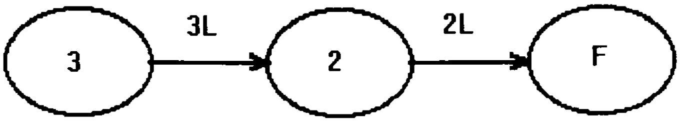


Figure 2.6 Markov model for TMR system

In the figure above, the states have the following meanings:

state 3 : the three processors are operational.

state 2 : two processors are operational.

state F : the system failed.

Taking LaPlace transforms of the state equations yields:

$$sP_3(s) - P_3(0) = -3LP_3(s)$$

$$sP_2(s) \quad P_2(0) = 3LP_3(s)$$

$$sP_F(s) \quad P_F(0) = 2LP_2(s)$$

Substituting the initial conditions $P_3(0) = 1$, $P_2(0) = 0$, and $P_F(0) = 0$, and performing the partial fraction expansions yield:

$$P_3(s) = \frac{1}{s+3L}$$

$$P_2(s) = \frac{3L}{(s+2L)(s+3L)}$$

$$P_F(s) = \frac{6L^2}{s(s+2L)(s+3L)}$$

Performing the inverse LaPlace transforms yields:

$$P_3(t) = e^{-3Lt}$$

$$P_2(t) = 3e^{-2Lt} - 3e^{-3Lt}$$

$$P_F(t) = 1 \quad - 3e^{-2Lt} + 2e^{-3Lt}$$

The reliability of the system is equal to the probability that the system is in either state (3) or state (2), therefore

$$\begin{aligned} R_{TMR}(t) &= P_3(t) + P_2(t) \\ &= e^{-3Lt} + 3e^{-2Lt} - 3e^{-3Lt} \\ &= 3e^{-2Lt} + 2e^{-3Lt} \end{aligned}$$

Now, if we let $R(t) = e^{-Lt}$, then the above equation becomes:

$$R_{TMR}(t) = 3R^2(t) \quad 2R^3(t)$$

This final result matches the one produced earlier using combinatorial modeling.

2.4 AVAILABILITY MODELING

System availability is an important factor in the analysis of fault-tolerant systems. In many cases the main concern is not how long a computer can operate without any failure (reliability), but it is whether it will be available when needed. As mentioned earlier, availability can be approximated as the system's operation time divided by the total time elapsed since the system started operation, or,

$$A_{ss}(t) = (\text{operation time}) \,/\, (\text{operation time} + \text{repair time}) \qquad (2.8)$$

The above equation emphasizes the importance of short repair times (rate) in highly available systems.

Markov models can be used to calculate system availability. But since in many cases we are only interested in the steady state availability $A_{SS}(t)$, another simpler technique is usually used. Consider a simplex system with failure rate L and repair rate m, if we assumed that the system experienced N failures during its life time, then

MTTF = 1/L

and

MTTR = 1/m

Furthermore,

system operation time = N(MTTF) = N/L,

system repair time = N(MTTR) = N/m

Substituting these values in equation (2.8) yields:

$$A_{ss}(t) = \frac{1/L}{\frac{1}{L} + \frac{1}{m}}$$

$$= \frac{m}{m + L} \qquad (2.9)$$

This last result agrees with the formula obtained earlier for $A_{SS}(t)$ using Markov models (see equation 2.7). As an example, consider the simplex system modeled in figure 2.5, and let the failure rate L = 0.01 (MTTF = 100 hr), and the repair rate m = 0.1 (MTTR = 10 hrs). Then from equation (2.9) , the system's steady state availability is 0.909090909. Using equation (2.6), we get the availability as:

$$A(t) = P_1(t) = (0.9090909) + (0.0909090)\, e^{-0.11t}$$

A plot for the above result is shown in Figure 2.7

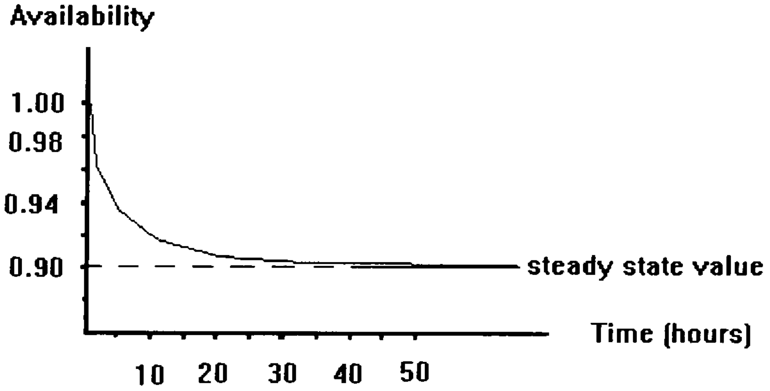


Figure 2.7 Availability as a function of time

As shown in figure 2.7, the system approaches its steady-state availability value in a short period of time, hence the importance of this value.

2.5 SAFETY MODELS

The definition of the word "safe" itself depends on the application. With this in mind we will divide each system's failed state into two states, Safe Failed (SF), and Unsafe Failed (UF). The distinction between these two is whether the fault was detected by the system or not, hence the importance of fault detection coverage C. As an example, consider a simplex system with failure rate L and fault detection coverage C. The Markov module for this system is shown in Figure 2.8.

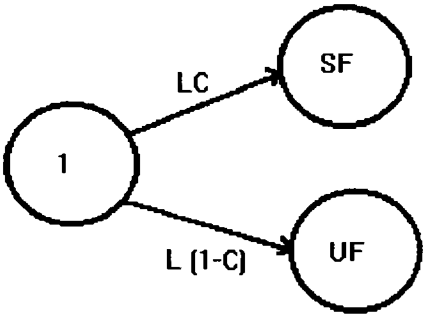


Figure 2.8 Markov model for safety calculations.

Taking LaPlace transforms of the state equations with $P_1(0) = 1$, $P_{SF}(0) = P_{UF}(0) = 0$, and solving yield:

$$P_1(s) = \frac{1}{s+C}$$

$$P_{SF}(s) = \frac{C}{s} - \frac{C}{s+L}$$

$$P_{UF}(s) = \frac{1-C}{s} - \frac{1-C}{s+L}$$

Performing the inverse LaPlace transforms yields:

$$P_1(t) = e^{-Lt}$$

$$P_{SF}(t) = C - C\,e^{-Lt}$$

$$P_{UF}(t) = (1-C) - (1-C)\,e^{-Lt}$$

Finally, the Safety of the system is:

$$S(t) = P_1(t) + P_{SF}(t)$$

$$= e^{-Lt} - C\,e^{-Lt} + C$$

The previous equations agree with the ideas we have built so far. For example, the system reliability is given by:

$$R(t) = P_1(t) = e^{-Lt}$$

Furthermore, at time $t = 0$ (initially) the system safety is $S(0) = 1$, and as time approaches infinity the system safety approaches C (the fault coverage). The next section is a full reliability analysis of a TMR with spare system, 3-out-of-5, and a 3-out-of-4 majority voting systems.

2.6 SYSTEM COMPARISON

To show the importance of the techniques encountered so far, the reliability of different fault-tolerant systems will be analyzed. As an example, consider the following systems: 3-out-of-4 majority voting, 3-out-of-5 majority voting, and TMR with spare systems (this last one was implemented as part of this thesis work).

3 - out of - 4 System

A 3-out-of-4 majority voting system requires 3 operational processors for its correct operation. Assume that each processor has a failure rate of L, and no repair is taking place, then the Markov model is shown in Figure 2.9.

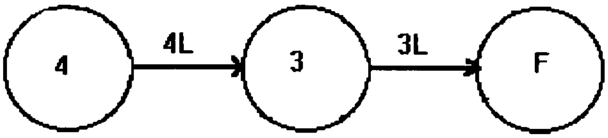


Figure 2.9 3-out of-4 Markov model

In figure 2.9, the states have the following meanings:

state 4 : all processors are operational.

state 3 : three processors are operational.

state F : system failed.

Taking the LaPlace transforms of the state equations and solving yield:

$$P_4(s) = \frac{1}{s+4L}$$

$$P_3(s) = \frac{4L}{(s+3L)(s+4L)}$$

$$P_F(s) = \frac{12L^2}{s(s+3L)(s+4L)}$$

Performing the partial fraction expansions yields:

$$P_4(s) = \frac{1}{s+4L}$$

$$P_3(s) = \frac{4}{s+3L} - \frac{4}{s+4L}$$

$$P_F(s) = \frac{1}{s} - \frac{4}{s+3L} + \frac{3}{s+4L}$$

Solving for R(t) (R(t) = 1 - P_F(t)), yields:

$$R(t) = 4e^{-3Lt} - 3e^{-4Lt}$$

2.6.2 3-out of-5 System

A 3-out-of-5 Majority voting system requires three operational processors at any time for its correct operation. Assuming that each processor has a failure rate of L, and no repair is taking place, yield the Markov model of figure 2.10.

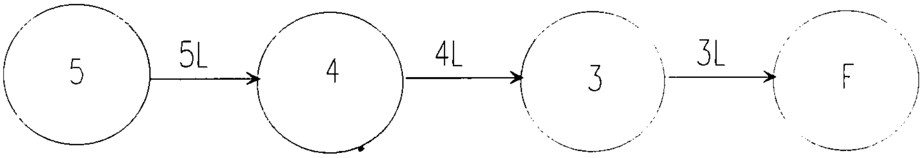


Figure 2.10 3-out-of-5 Markov model

Writing the state equations and performing the LaPlace transforms yield :

$$\begin{aligned}
sP_5(s) &- P_5(0) = -5LP_5(s) \\
sP_4(s) &- P_4(0) = -4LP_4(s) + 5LP_5(s) \\
sP_3(s) &- P_3(0) = -3L\,P_3(s) + 4LP_4(s) \\
sP_F(s) &- P_F(0) = 3LP_3(s)
\end{aligned}$$

Solving for R(t) (R(t) = P_5(t) + P_4(t) + P_3(t)) with the the initial conditions :

$P_5(0) = 1, P_4(0) = 0, P_3(0) = 0, P_F(0) = 0$

yields:

$$R(t) = 6e^{-5Lt} - 15e^{-4Lt} + 10e^{-3Lt}$$

2.6.3 TMR with Spare System

Using the same assumptions used for the previous systems, gives the Markov model shown in figure 2.11.

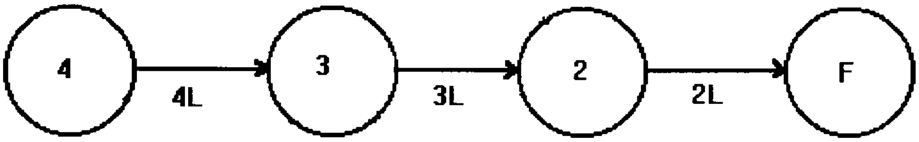


Figure 2.11 Markov model for TMR with spare system

In the above figure the states have the following meanings:

state 4 : all three processors and the spare are operational.

state 3 : only three processors are operational

state 2 : only two processors are operational.

state F : the system failed.

Taking LaPlace transforms of the state equations yields:

$$sP_4(s) - P_4(0) = -4L\, P_4(s)$$

$$sP_3(s) - P_3(0) = 4L\, P_4(s) - 3L\, P_3(s)$$

$$sP_2(s) - P_2(0) = 3L\, P_3(s) - 2L\, P_2(s)$$

$$sP_F(s) - P_F(0) = 2L\, P_2(s)$$

Solving for the system's reliability (R(t) = 1 - P_F(t)) with the initial conditions,

$$P_4(0) = 1, P_3(0) = 0, P_2(0) = 0, P_F(0) = 0,$$

we get :

$$R(t) = 3e^{-4Lt} - 8e^{-3Lt} + 6e^{-2Lt}$$

This value is obviously larger than the reliability values of the previous systems. Figure 2.12 is a comparison between the systems considered. From figure 2.12 we can notice the following:

- Although both the TMR with spare and the 3-out-of-4 systems contain the same level of redundancy, the first is more reliable than the second. Furthermore, it is also more reliable than the 3-out-of-5 system which contains a higher level of redundancy (4 redundant processors).

- Recall that the reliability graphs of TMR and Simplex systems intersect at a reliability value of 0.5 (see figure 1.5). The addition of an extra processor to the TMR may shift this intersection point up (as it is the case with the 3-out-of-4 system), or down (as it is the case with the TMR with spare system). The up-shifting means that the redundant system processors must have reliability values larger than 0.5 to make the redundant system more reliable than the simplex one. The down shifting loosens this requirement on the redundant system.

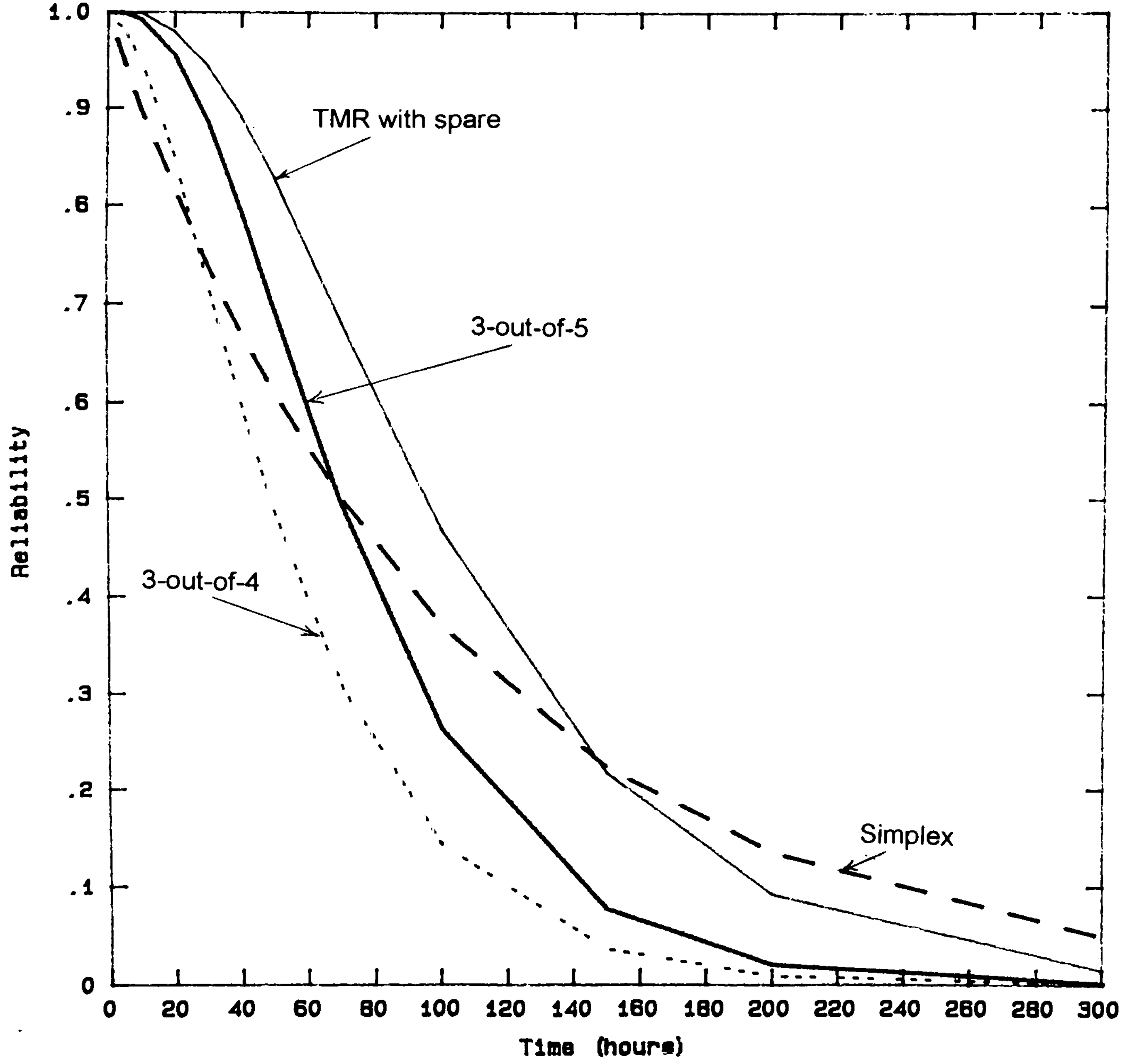
- Notice that the reliability of the TMR with spare system becomes smaller than that of the simplex system after a certain point in time. This suggests that the TMR with spare system is more suitable for short life applications (In chapter 4 we will come to a different conclusion).

To further study the characteristics of these systems, the MTTF values were calculated for all of them. The results shown below were calculated with L = 0.01 failure/hour.

System	MTTF (hours)
Simplex	100
TMR	83
3-out-of-4	58
3-out-of-5	78
TMR with spare	108

The above results suggest that although the TMR with spare system has a small advantage over the simplex system, the Simplex is better than the other systems. However one should not jump into such inaccurate conclusion. To explain the above results, take another look at figure 2.12. The MTTF can be seen as the area under the reliability curve, and that area is larger for the simplex system than for the 3-out-of-4 and the 3-out-of-5 systems. So, if the intended application has a short life time the 3-out-of-5 system, for example, is better than the simplex. This suggests that the MTTF alone is not an accurate measure of a system. Instead, a measure that takes into consideration the intended application life is needed, such as the Mission Time MT(r). Table 2.2 lists some values for MT(0.90) of different systems at a failure rate L = 0.01.

Figure 2.12
(L = 0.01 fault/hour)



System Comparison

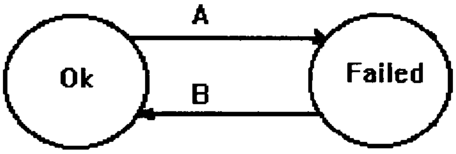
System Configuration	MT(0.90)
Simplex	10.53 hours
TMR	21.90 hours
3-out of-4	15.0 hours
3-out-of-5	28.2 hours
TMR with Spare	38.5 hours

Table 2.2 MT(0.90) for different systems with L = 0.01

From table 2.2 one can see that the TMR with spare system has a mission time improvement factor of 3.66, 1.76, 2.57, and 1.37 over Simplex, TMR, 3-out of-4, and 3-out-of-5 systems respectively. And all of the TMR, 3-out-of-5, and 3-out-of-4 systems are superior to the simplex system. Finally notice that all the previous measures favored the TMR with spare system over the other considered systems.

2.7 INTERMITTENT FAULTS

In the previous analysis we only considered the effect of hard/permanent faults on fault-tolerant systems. But since experimental data suggests that approximately 90% of system faults are intermittent, the effect of this class of faults must be considered. An intermittent fault can be defined as a fault with temporary behavior. It may be caused by lose/dry connection(s), temperature sensitive elements, external interferences, etc. The difficulty with intermittent faults arises from their temporarily beahviour. Figuer 2.13 is an intermittent fault model. The transition rate B (in figure 2.13) has a big effect on the intermittent faults detection coverage. A full intermittent faults analysis will be done in chapter 4 (system analysis).



A : Intermittent faults rate
B : Intermittent faults disappearing rate.

Figure 2.13 Intermittent faults modeling

CHAPTER THREE
SYSTEM DESCRIPTION

3.0 SYSTEM DESCRIPTION

To study and illustrate some of the design issues involved in fault-tolerant systems, a fault-tolerant system was built. The system consists of two reconfigurable Triple Modular Redundancy (TMR) with spare modules (see figure 3.1). The system's application is a two main-substreet-intersections traffic lights controller (see figure 3.2), with each module controling one intersection. The system's terminal displays the processors status/failure data and is shared by both modules. This chapter describes the system's hardware, operation, and some of the design problems that were encountered and the approach used in solving them.

3.1 SYSTEM HARDWARE

The system consists of two modules (see figure 3.3 in the appendix). Each module contains the following elements:

- Three processors (proc1, 2, and 3) all of them are Motorola MC68705.
- A spare processor, also MC68705.
- Controller processor, Motorola MC68705.
- Switching circuitry.

  Voting circuitry.
- Disagreement detection circuitry.
- Shift register.
- Traffic Lights.

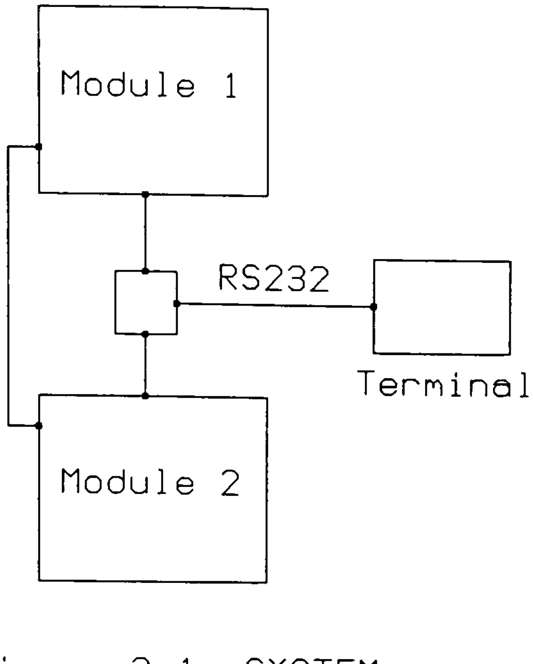


Figure 3.1 SYSTEM

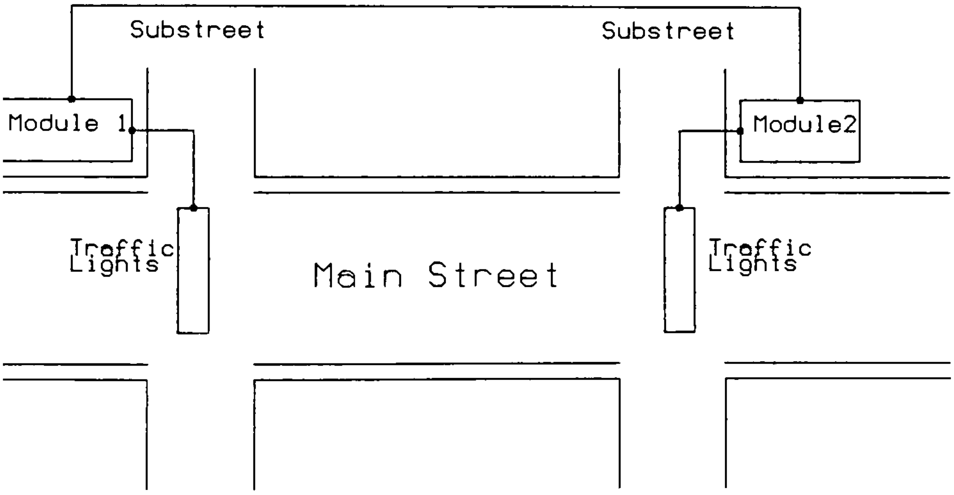


Figure 3.2 APPLICATION

The Processors and the Spares

The processors and the spares are Motorola MC68705. They execute the application program (i.e. traffic light controlling). Processors 1, 2, and 3 execute the exact same program. The Spares execute slightly different programs. A listing of these programs is shown in the appendix.

The processors deliver their outputs (traffic lights control signals) serially. Delivering the data serially requires each processor to produce another two control lines for the serial data control (shift registers serial clock and load signals), a total of three. This approach is better than delivering the outputs in parallel. The reason is that delivering the data in parallel requires each processor to produce six output signals (for the six traffic lights at each intersection). Each of these must pass through a voting stage, a disagreement detection stage, and a switching circuitry.

One of the major problems in majority voting systems is processor synchronization. To address this problem, we chose to use a separate clock for each processor rather than a common clock. This will be the subject of section 3.3.

The Controller

The controller is also a Motorola MC68705. It collects the processors' disagreement data (or faults), displays their status and failure data, and reconfigures the module based on faults information. As seen from figure 3.3, the controller I/O ports are occupied as follows:

- Twelve ports d1...d12 (or PA0..PA7, and PB0..PB3 respectively) to collect fault information (disagreement detection).
- Three lines for module reconfiguration s1, s2, and s3 (or PB5, PB6, PB7).
- Two lines to regulate the collection of processor fault data H2, H3 (or PB4, and PC0).
- One line for the reset signal (PC1).
- One line to regulate the sharing of the RS232 bus (PC2).
- One line to send data to the terminal (PC3).

Switching Circuitry

The switches (see figure 3.4) are used to determine which processor participates in the voting process. The controller controls this choice using the control lines s1, s2, and s3. Each module contains nine switches (a module contains three processors with three output lines from each, a total of nine). For example, consider the top switch in figure 3.3, this switch controls the voter input F1, to be either PA0 from the first processor (if s1 = 0), or PA0 from the spare (if s1 = 1).

Voting Circuitry

The voters (see figure 3.5) are 2-out-of-3 majority voters, which means that the voter output agrees with at least two of its input signals. Each module contains the following voters (refer to figure 3.3):

- The traffic lights data voter (labeled H1, the top one in figure 3.3).
- The serial clock voter (labeled H2).
- The load signal voter (labeled H3).

Disagreement Detection Circuitry

A disagreement detector is simply an XOR that produces a high (1) output if its inputs are not the same and low (0) otherwise (see Figure 3.6). A disagreement detector is used to detect whether a processor output agrees with the voter output (majority output) or not. Each module contains the following groups of disagreement detectors:

- Data lines disagreement detectors determine whether PA0 from each processor, agrees with the voter output (H1) or not, and produce d1, d2, d3, andd4.
- Serial clock disagreement detectors determine whether PB0 from each processor agrees with the voter output (H2) or not, and produce d5, d6, d7, and d8.
- Load signal disagreement detectors determine whether PB1 from each processor agrees with the voter output H3 or not, and produce d9, d10, d11, and d12.

The disagreement detectors d1, d5, and d9 detect the faults of processor #1. The disagreement detectors d2, d6, and d10 detect the faults of processor #2. The disagreement detectors d3, d7, and d11 detect the faults of processor #3. Finally, The disagreement detectors d4, d8, and d12 detect the faults of the spare processor.

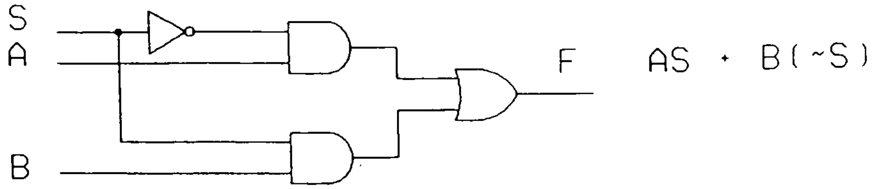


Figure 3.4 Switch

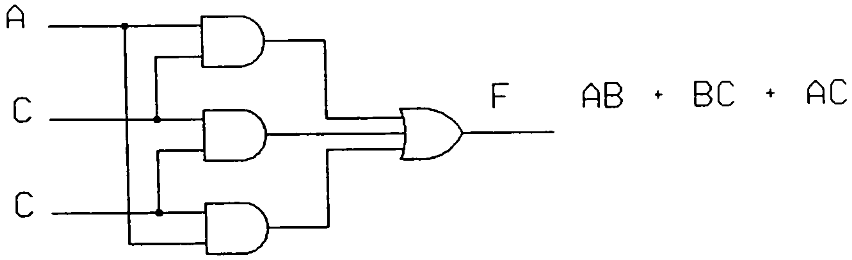


Figure 3.5 Voter

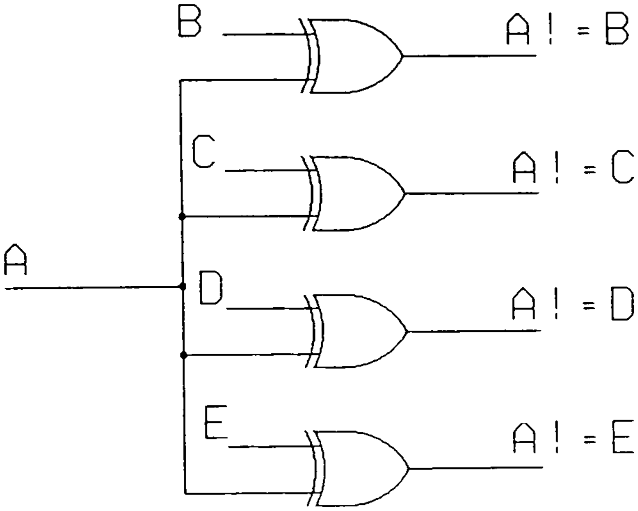


Figure 3.6 Disagreement Detectors

Shift Register

The shift register used is a serial-in parallel-out shift register with buffers, MC74HC595. The inputs of the shift register are the outputs of the voters H1, H2, and H3 (for the data, serial clock, and load signals respectively), and its outputs are connected to the traffic lights (LED's), as shown in figure 3.3.

3.2 SYSTEM OPERATION

As mentioned earlier, the system controls two main-substreet-intersections traffic lights (see figure 3.2). The main street green light is five times longer than the substreet green light. The system contains two modules, module one (the one to the left in figure 3.2) and module two. Each module controls one intersection. The modules are almost identical except in that module two tries to follow module one (so that the two main street traffic lights are both green), and in that module one is responsible for controlling the RS232 bus.

Each module continues its normal operation if it has at least two operational processors (including the spare). Otherwise, it goes into flashing by simply resetting the processors in a specified time interval. This flashing approach is safer than making the "faulty" processors execute a "flashing" procedure.

The Controller

Each module contains one controller; the controllers are almost identical except for the differences mentioned earlier. Each controller has its own reset switch. After receiving a reset signal, the controller from module one sends the header message to the terminal and the operation continues as follows:

First, each controller resets the processors in its module (at PC1), and initializes the total number of failures encountered by each processor, so far, to zero. Then it waits for the serial clock to arrive (H2 at PB4). During each serial clock (H2) the controller tests the serial clock disagreement lines (d5, d6, d7, and d8) and saves the results. Then it tests the serial data disagreements lines (d1, d2, d3, and d4) and saves the results. All this is done before the next serial clock pulse arrives. Since there are six lights (i.e. six data bits and clock pulses will be generated), the controller repeats this operation six times, and of course sums each processor's faults and saves the results. Then the controller waits for the load signal (H3) to arrive. When it arrives the controller tests the parallel load disagreement lines (d9, d10, d11, and d12) , and saves the results.

While the processors are waiting for the current traffic light time slot to elapse, the controller starts analyzing the collected data, and tests whether any of the processors encountered a permanent fault or exceeded the maximum number of intermittent faults allowed. Depending on these results the controller may decide to reconfigure the module, go to flashing, or continue its normal operation. The decision is made as follows:

- If a processor, P_i, encounters a permanent fault at any of its output lines, then P_i is a faulty processor and the system is reconfigured (i.e. the spare processor is considered in the voting process instead of P_i).

- If a non-faulty processor P_i encounters N intermittent faults then,

  if N > maximum limit , then P_i is faulty and the system is reconfigured.

  if N <= maximum limit, then the system continues its normal operation.

- If two processors fail then, the Module fails:

  if this is the first module failure, flash traffic lights four times and restart the module (This is done to make the system more available).

  if this is the second module failure, then go to flashing indefinitely until Module is reset by the supervisor.

- After every M (time units) clear all operational processors' fault records, and start counting from zero.

The controller considers any two or more consecutive faults on any line to be permanent, and intermittent otherwise. The maximum number of intermittent faults allowed (N faults in M time units) is set depending on: the expected/measured intermittent faults rate, whether the faults are casued by global conditions or not, and on the system application.

The Processors

The processors, in each module, execute the same program with the spare program being slightly different in the synchronization procedure. After receiving the reset signal from the controller, a processor sends an all-off signal to the traffic lights for half a second, then it sends a green signal for the main street lights and red for the substreet lights, and continues from there in a weighted round robin manner. Module one processors send their status to module two processors, so that the later will try to follow the operation of the first.

As mentioned earlier, one of the major problems that faces majority voting systems is to synchronize the processors for the voting process to be a success. The following section considers this problem and our approach in detail.

3.3 SYNCHRONIZATION

Synchronizing voting processors and keeping them synchronized is one of the problems that faces all majority voting systems. In our system we used a separate clock for each processor to address this problem. One of the major causes of this problem is that even without any processor failures, and even if the processors started operation at the same clock cycle, these processors will eventually go out of synchronization due to the fact that even the best quartz crystals have a margin of error of 0.01% - 0.5 %. Of course the error may be very small in a well designed system operating in a controlled environment, but still it is there and it will affect the system operation. Furthermore, since it is very necessary in many applications to use timers, a one per million clock error, for example, will be magnified by the timer prescalar factor, resulting in an N per million clock error; a more serious situation.

Of course the clock sources errors are not the only cause of the synchronization problem; faults usually drive processors out of synchronization. One may not have a great concern about a processor being driven out of synchronization by a permanent fault since the processor is faulty anyway, but this argument does not hold when we are dealing with intermittent faults, especially with most studies suggesting that up to 90% of system failures are due to intermittent faults. With this being the case, system designers need to make sure that intermittent faults do not drive processors out of synchronization, hence having the effect of permanent faults.

In our system the synchronization problem effect was very clear. The system was first built using a common clock source and it was working almost fault free. But when we tried to use separate clock sources, the voting process was rarely a success and (on the average) the system's MTTF was 2 seconds. Studying the behavior of the system showed that even if the processors started at the same clock cycle (after the reset), they will be hundreds of clock cycles out of synchronization four to five seconds later. Our search for the cause of the problem led us to the timer prescaling factor (a factor of 128) which magnifies the crystal errors 128 times. After setting this factor to one, the system's MTTF improved to a system failure every 20 to 30 seconds. These failures were mainly caused by crystal errors driving the processors out of synchronization. The crystal errors were caused by stray capacitances and noises on the circuit board. After isolating the crystals from the system common ground, the system's

MTTF improved to a system failure every five to seven minutes. Finally, synchronizing the processors produced an almost failure free system.

Our Approach

Although we had a plenty of system time that can be spent on processor synchronization (due to the application), we tried to synchronize the processors as efficiently as possible. Figure 3.7 shows the extra hardware connections that has to be made between the processors to synchronize them. These connections are needed to enable each processor to send an " I'm ready" synchronization message to other processors, and to receive such messages from them. Notice also that these connections are arranged in a way that will allow each processor to treat ports C and B connections as " me first, then the others in ascending order". This arrangement allows the processors to execute the same program and enables them to physically replace each other.

Figure 3.8 is a flowchart of the synchronization procedure "synchronize" which is executed by the processors one, two, and three. Figure 3.9 is a flowchart of the same procedure executed by the spare processor.

Finally, to prevent the processors from being driven out of synchronization, the synchronization procedure is executed every one second and also before any voting process. This may seem to be too costly, but the time spent on the synchronization procedure itself does not exceed 40 cycles.

PROCESSORS' SYNCHRONIZATION

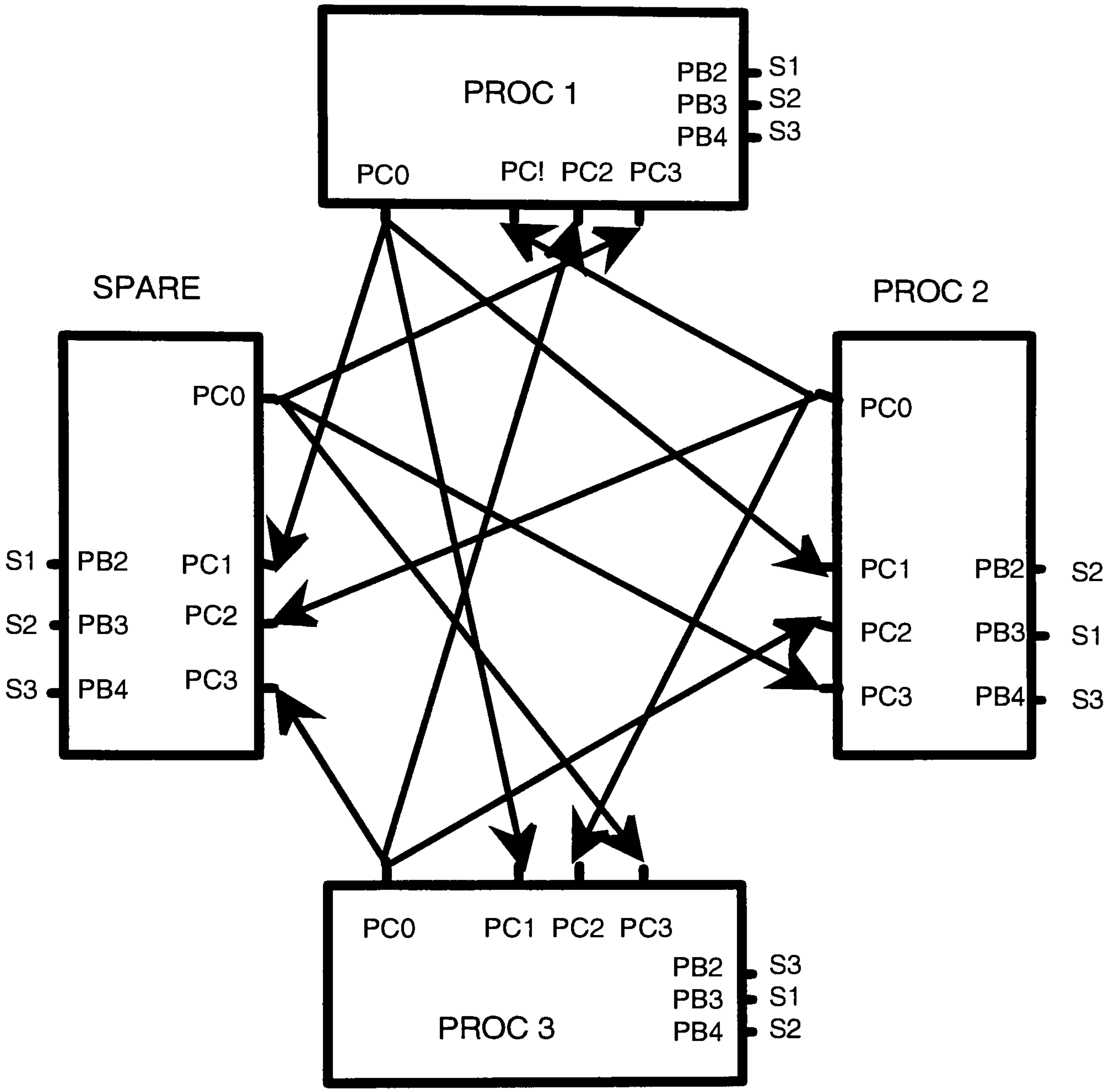


Figure 3.7 The extra connections needed to synchronize the processors

3.4 THE SYSTEM

The system is mounted on a single board, except for the reset circuitry and the crystals, which are mounted on separate boards. The voters, switches, buffers, and the disagreement detectors were implemented using programmable logic arrays. Figure 3.10 (in the appendix) shows the actual pin connections for each of the system's modules. It is easy to understand the analogy between figure 3.10 and figure 3.3 since we used the same labeling in both figures.

In the following paragraph, a circuit is labeled after its output. For example, a switch with an output line labeled F1 will be called switch F1. U1 through U6 in each module are PALCE16V8 programmable logic arrays:

- U1 : Implements the first five switches (F1,....,F5).
- U2 : Implements the switches (F6,....,F9).
- U3 : Implements the three voters (H1, H2, and H3).
- U4 : Implements the data disagreement detectors (d1,...,d4).
- U5 : Implements the serial clocks disagreement detectors (d5,...,d9).
- U6 : Implements the load signals disagreement detectors (d10,...,d12), and the three buffers between the controller (PC1), and the processors reset circuits.

For a full listing of processors', controllers', spares', PLA's programs see the appendix.

PROCEDURE SYNCHRONIZE
PROCESSORS 1, 2, & 3

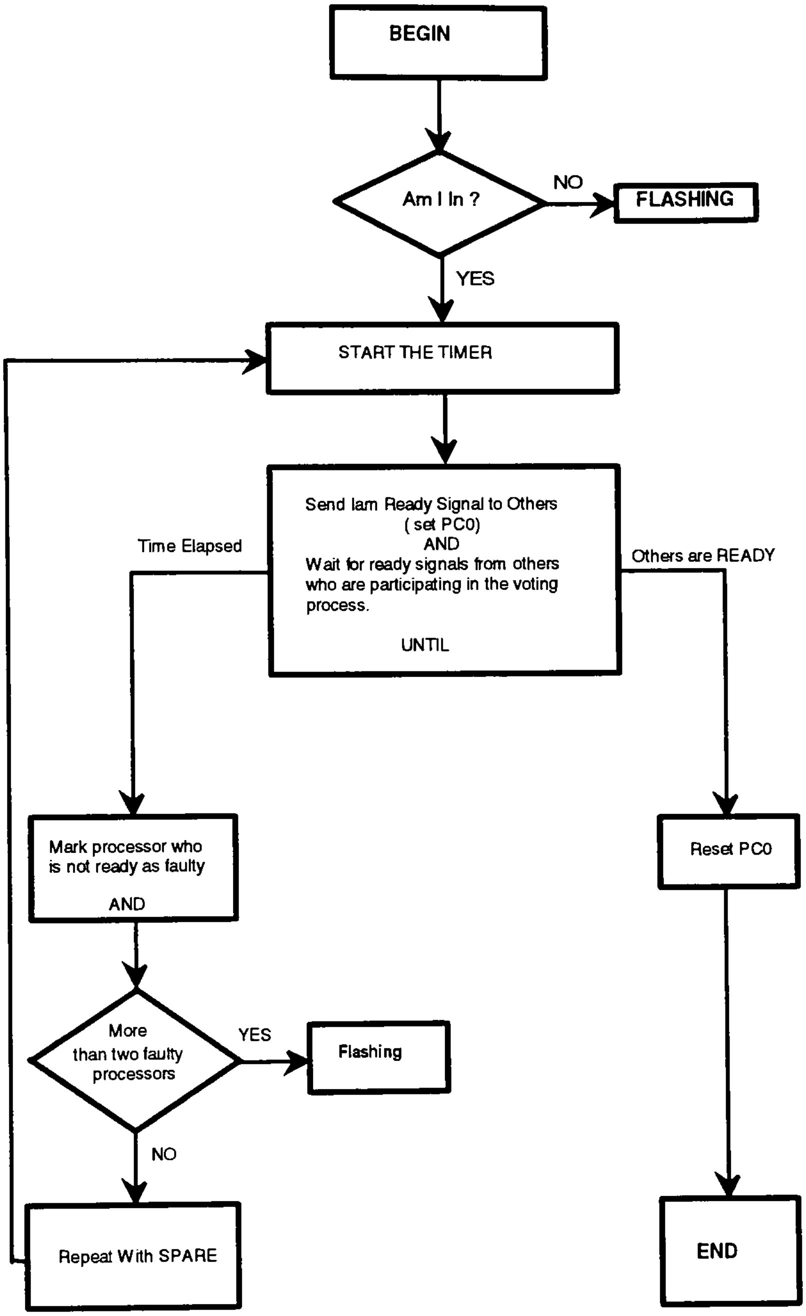

BEGIN
Am I In ?
NO
FLASHING
YES
START THE TIMER
Send Iam Ready Signal to Others
(set PC0)
AND
Wait for ready signals from others who are participating in the voting process.
UNTIL
Time Elapsed
Others are READY
Mark processor who is not ready as faulty
AND
Reset PC0
More than two faulty processors
YES
Flashing
NO
Repeat With SPARE
END

Figure 3.8

PROCEDURE SYNCHRONIZE SPARE

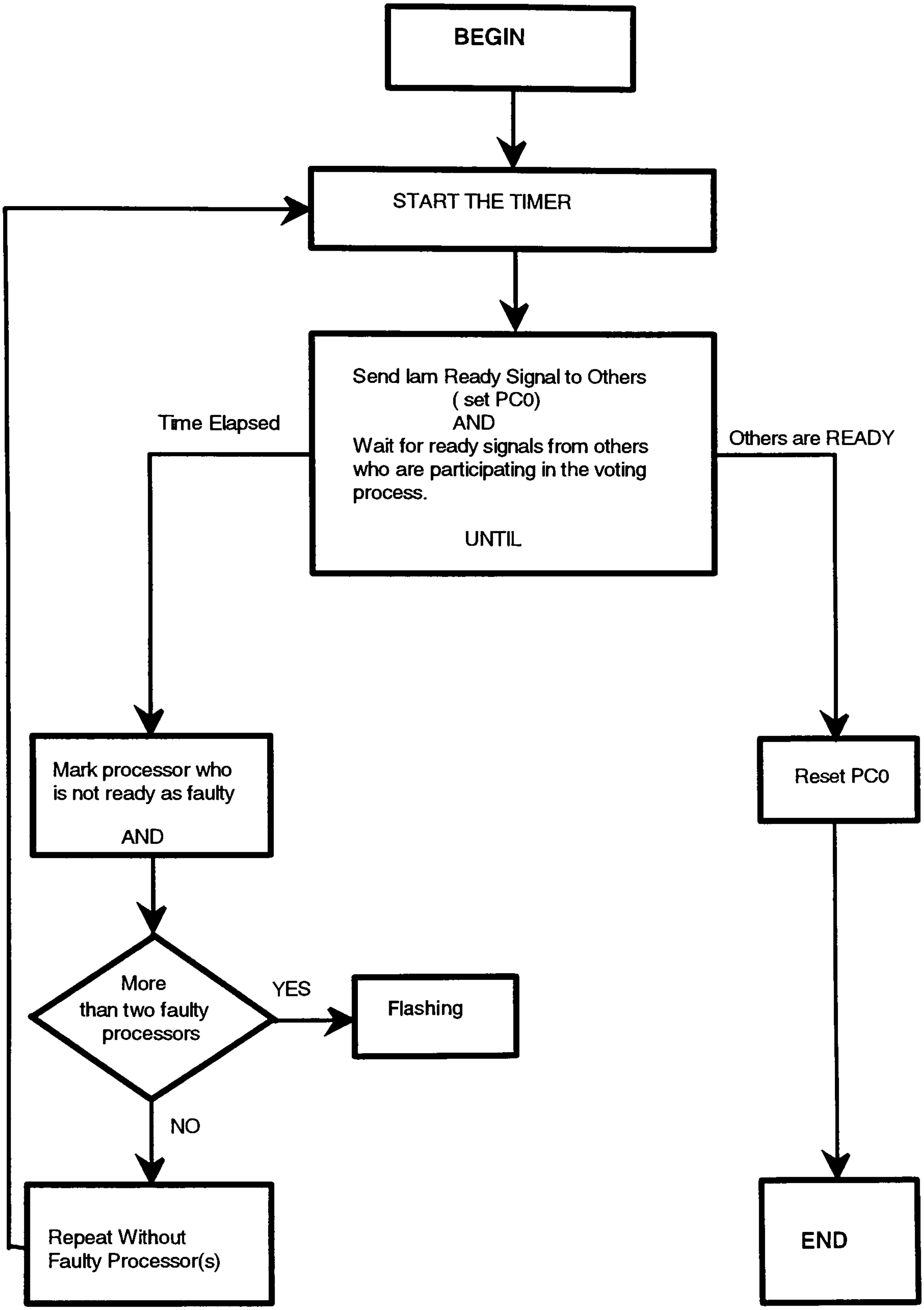

BEGIN
START THE TIMER
Send Iam Ready Signal to Others
(set PC0)
AND
Wait for ready signals from others who are participating in the voting process.
UNTIL
Time Elapsed
Others are READY
Mark processor who is not ready as faulty
AND
Reset PC0
More than two faulty processors
YES
Flashing
NO
Repeat Without Faulty Processor(s)
END

Figure 3.9

3.4 THE SYSTEM

The system is mounted on a single board, except for the reset circuitry and the crystals, which are mounted on separate boards. The voters, switches, buffers, and the disagreement detectors were implemented using programmable logic arrays. Figure 3.10 shows the actual pin connections for each of the system's modules. It is easy to understand the analogy between figure 3.10 and figure 3.3 since we used the same labeling in both figures.

In the following paragraph, a circuit is labeled after its output. For example, a switch with an output line labeled F1 will be called switch F1. U1 through U6 in each module are PALCE16V8 programmable logic arrays:

- U1 : Implements the first five switches (F1,....,F5).
- U2 : Implements the switches (F6,....,F9).
- U3 : Implements the three voters (H1, H2, and H3).
- U4 : Implements the data disagreement detectors (d1,...,d4).
- U5 : Implements the serial clocks disagreement detectors (d5,...,d9).
- U6 : Implements the load signals disagreement detectors (d10,...,d12), and the three buffers between the controller (PC1), and the processors reset circuits.

For a full listing of processors', controllers', spares', PLA's programs see the appendix.

CHAPTER FOUR
RELIABILITY ANALYSIS

4.1 INTRODUCTION

As mentioned earlier, statistics show that up to 90% of system failures are caused by intermittent faults. With this being the case, the reliability analysis of fault-tolerant systems will not be accurate without taking this class of faults into consideration. In this chapter we discuss the reliability of our system while concentrating on intermittent faults and their effects.

4.2 SYSTEM MODELING

A processor in our system is considered to be faulty if it encounters a permanent fault or more than N intermittent faults in M hours. Permanent faults are defined as those existing during two or more consecutive decisions, while Intermittent faults do not. The values of the parameters N and M should be set depending on: the intermittent fault rate, intermittent fault causes, and the system application.

The Markov model of our system is shown in figure 4.1, with the intermittent faults being modeled for one processor only due to graph complexity. The intermittent fault models for the other processors are the same as the one shown. In figure 4.1, states Q1, Q2, and Q3 represent the system with four, three, or two working processors, respectively. And state x (x = 1,..,N) represents a processor with x intermittent faults.

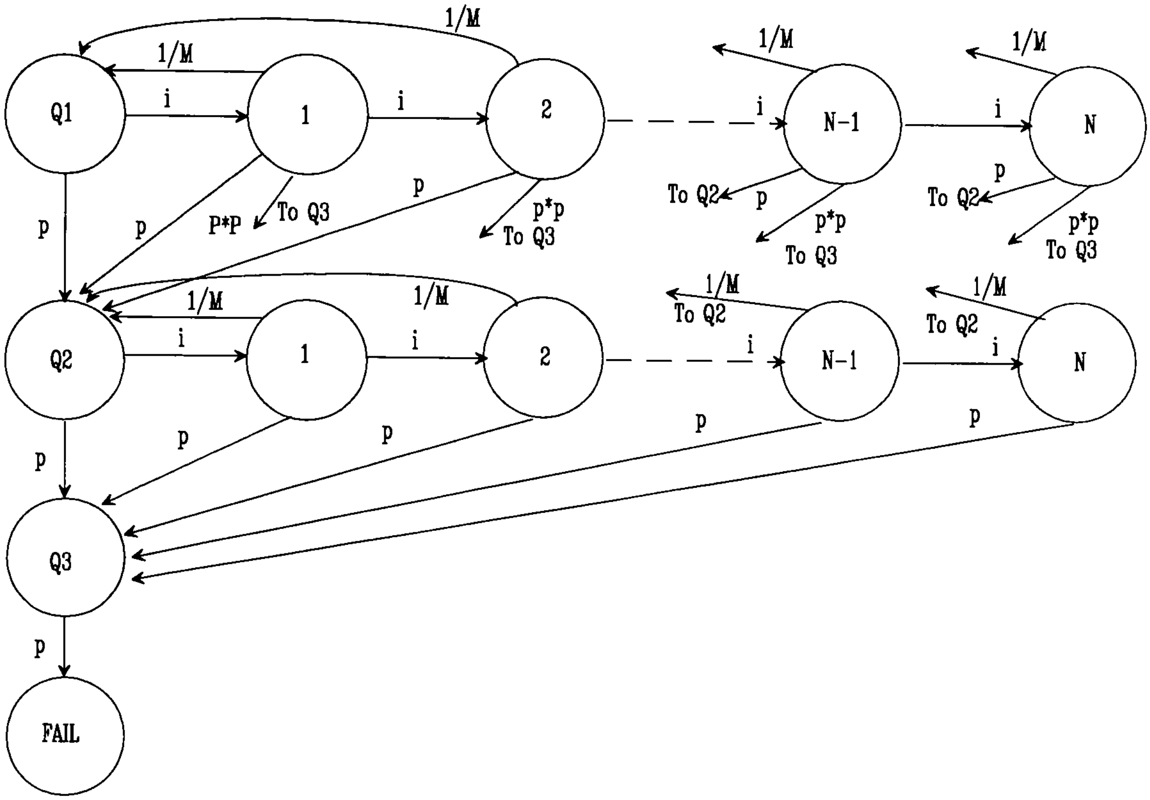


Figure 4.1 System Modeling

i: Intermittent Faults rate

p: Permanent Faults rate

The state equations for the above model are very difficult to solve, and they will become much more difficult if we consider the state equations for all the remaining processors. Hence, let us try to find an equivalent simpler model for the one in figure 4.1. First, consider the models in figures 4.2 and 4.3. State Q1 in figure 4.3 is the same as that in figure 4.2, and state (1) in figure 4.3 represents the combination of states (1) through (N) in figure 4.2. Our goal now is to find A (in figure 4.3) such that these two models are equivalent.

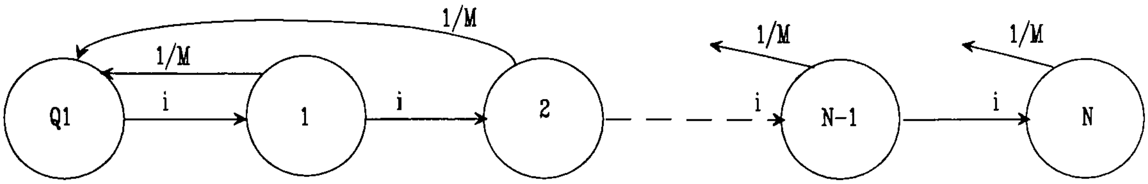


Figure 4.2

Markov model for a processor with intermittent faults only

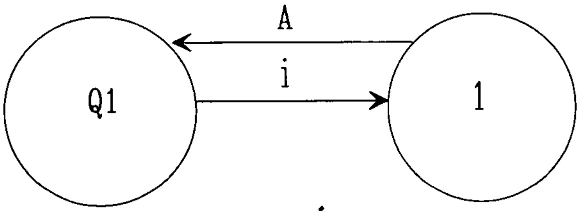


Figure 4.3

Equivalent module of figure 4.2

Consider the model in figure 4.2. The equation for state Q1 is given by:

$$\frac{dQ_1(t)}{dt} = -i \quad Q_1(t) + \frac{1}{M}P_1(t) + \frac{1}{M}P_2(t) + ... + \frac{1}{M}P_N(t)$$

$$= -i \quad Q_1(t) + \frac{1}{M}(P_1(t) + P_2(t) + ... + P_N(t))$$

$$sQ_1(t) - Q_1(0) = -iQ_1(s) + \frac{1}{M}(P_1(s) \quad + P_2(s) + ... + P_N(s)) \qquad (4.1)$$

In equation (4.1), P_i represents the state i (i = 1,..,N). The equation for state Q1 in the model of figure 4.3 is given by:

$$\frac{dQ_1(t)}{dt} = -i\,Q_1(t) + A\,P_1(t)$$

or

$$sQ_1(s) - Q_1(0) = -i\,Q_1(s) + A\,P_1(s) \tag{4.2}$$

Now, we want to find the value of A in equation (4.2) such that equations (4.1) and (4.2) are equivalent. To do so, we write every $P_i(s)$ in equation (4.1) in terms of $P_1(s)$, and then equate the two equations and solve for A. Solving with the initial conditions $P_1(0) = P_2(0) = ... = P_N(0) = 0$, gives the following (refer to figure 4.2):

for state 2:

$$s\,P_2(s) - P_2(0) = i\,P_1(s) - (\frac{1}{M}+i)\,P_2(s)$$

or (4.3)

$$P_2(s) = \frac{i}{s+\frac{1}{M}+i}\,P_1(s)$$

and for state 3:

$$s\,P_3(s) - P_3(0) = i\,P_2(s) - (\frac{1}{M}+i)\,P_3(s)$$

or

$$P_3(s) = \frac{i}{s+\frac{1}{M}+i}\,P_2(s)$$

Substituting for P2(s) (from equation 4.3) in the previous equation, we get:

$$P_3(s) = [\frac{i}{s+\frac{1}{M}+i}]^2\,P_1(s) \tag{4.4}$$

Or in general:

$$P_N(s) = \left(\frac{i}{s+\frac{1}{M}+i}\right)^{N-1} P_1(s) \qquad (4.5)$$

Substituting the values for $P_1(s)$, $P_2(s)$, ..., $P_N(s)$ in equation (4.1) yields:

$$sQ_1(s) - Q_1(0) = iQ_1(s) + \frac{1}{M}P_1(s)\left[\sum_{k=0}^{k=N-1}\left(\frac{i}{s+\frac{1}{M}+i}\right)^k\right] \qquad (4.6)$$

Hence, the value of A in equation (4.2) is:

$$A(s) = \frac{1}{M}\sum_{k=0}^{k=N-1}\left(\frac{i}{s+\frac{1}{M}+i}\right)^k \qquad (4.7)$$

Taking the inverse LaPlace transforms yields:

$$A(t) = \frac{1}{M} + \frac{1}{M}\sum_{k=1}^{k=N-1}\frac{i^k t^{(k-1)}}{(k-1)!}e^{-at} \qquad (4.8)$$

where, a = i + p + 1/M .

Finally, combining the intermittent fault models for all processors, gives us the Markov model for our system (figure 4.4).

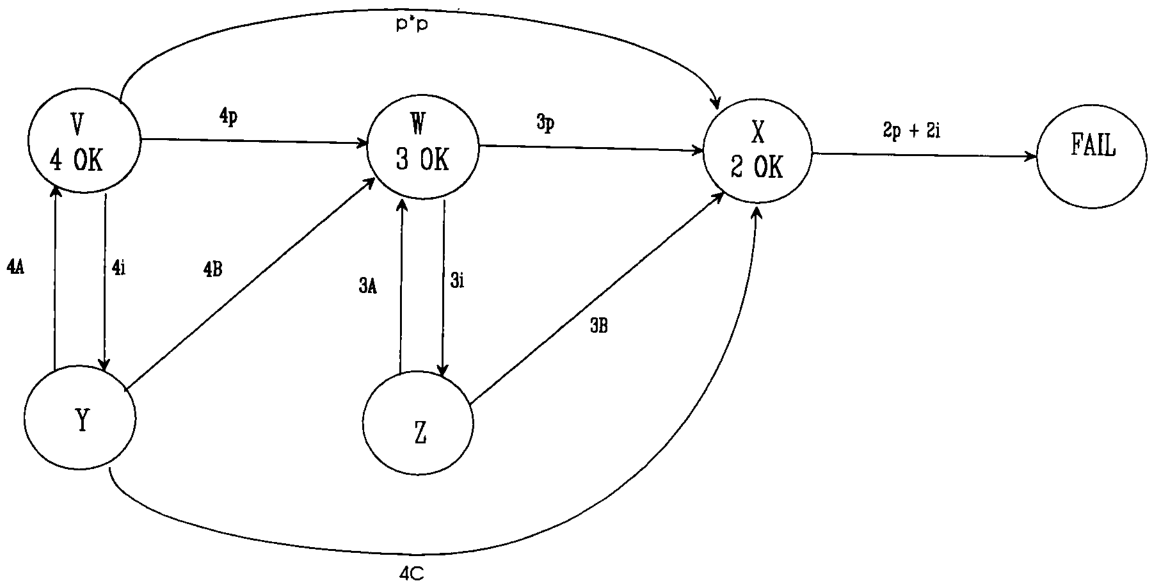


Figure 4.4

Equivalent model of figure 4.1

In figure 4.4, A is given by equation (4.8). B and C were obtained following the same analysis used for A, and they are given by:

$$B(t) \;=\; p+p\sum_{k=1}^{k=N-1} \frac{i^k t^{(k-1)}}{(k-1)!}\, e^{-at} \;+\; \frac{i^N t^{(N-2)}}{(N-2)!}\, e^{-at} \qquad (4.9)$$

$$C(t) \;=\; p^2+p^2\sum_{k=1}^{k=N-1} \frac{i^k t^{(k-1)}}{(k-1)!}\, e^{-at} \qquad (4.10)$$

where, a = i + p + 1/M .

This concludes the modeling of our system.

Recall that in section 2.6 the reliability of our system was calculated under permanent fault conditions. Now, let us repeat the same analysis under intermittent fault conditions.

4.3 RELIABILITY ANALYSIS

Assume that the system encounters intermittent faults only (p = 0); then the transitions V to W, V to X, W to X, and Y to X in figure 4.4 will disappear. Furthermore, the transition rate C(t) will be equal to zero, A(t) will stay the same, and B(t) will be given by:

$$B(t) = \frac{i^N t^{(N-2)}}{(N-2)!} e^{-at}$$

where, a = i + 1/M in both A(t) and B(t).

Before writing the state equations and solving for the system's reliability, there is one last issue that has to be dealt with. In the model of figure 4.4 the state transitions are not constants. This means that our system does not obey the exponential failure law. The most straightforward solution to this problem is to approximate these state transitions using constant values, but can we? To answer this question, let us try to interpret the meaning of these state transitions. A(t) represents the intermittent fault disappearance rate, or (as described in chapter 2) the transition from the failed state to the pseudo-failed state (see figure 2.13). A(t) can be thought of as being the "repair rate." B(t) represents the processors failure rate due to intermittent faults only. Figures 4.5, 4.6, 4.7, and 4.8, are plots of A(t) and B(t) for different values of M and N.

Figure 4.5: shows the effect of the parameter M on B(t). As expected, the probability that a processor will encounter more than N faults in M time units is bigger for larger values of M (longer time intervals).

Figure 4.6 : shows the effect of the parameter N on B(t). One can see how changing N changes the peak value for B(t) and its location. A larger value of N means that a processor can tolerate more intermittent faults, hence B(t) has a smaller peak value. The location of this peak is at:

$$t = \frac{N-2}{i+\frac{1}{M}} \quad hours \tag{4.11}$$

Figures 4.7 and **4.8** show the effect of both M and N on A(t). Notice that A(t) peaks at the same location regardless of M and N values. The steady state value of A(t) is 1/M. Figure 4.8 shows the relationship between the speed at which A(t) reaches its steady state and the value of the parameter N.

To approximate these functions (A(t) and B(t)) by constants, keep in mind that in reliability analysis the safest and the most reliable approach is the one based on the worst condition assumptions. So, in our system we will assume that the failure rate , B(t), is constant at its maximum value, and the "repair rate," A(t), is constant at its minimum value. This will give us a model similar to those of chapter 2.

The state equations for the model of figure 4.4 are given below:

$$(s+4i)\,V(s) - 4A\,Y(s) = 1 \tag{4.12}$$

$$(s+4A+4B)\,Y(s) - 4i\,V(s) = 0 \tag{4.13}$$

$$(s+3i)\,W(s) - 4B\,Y(s) - 3A\,Z(s) = 0 \tag{4.14}$$

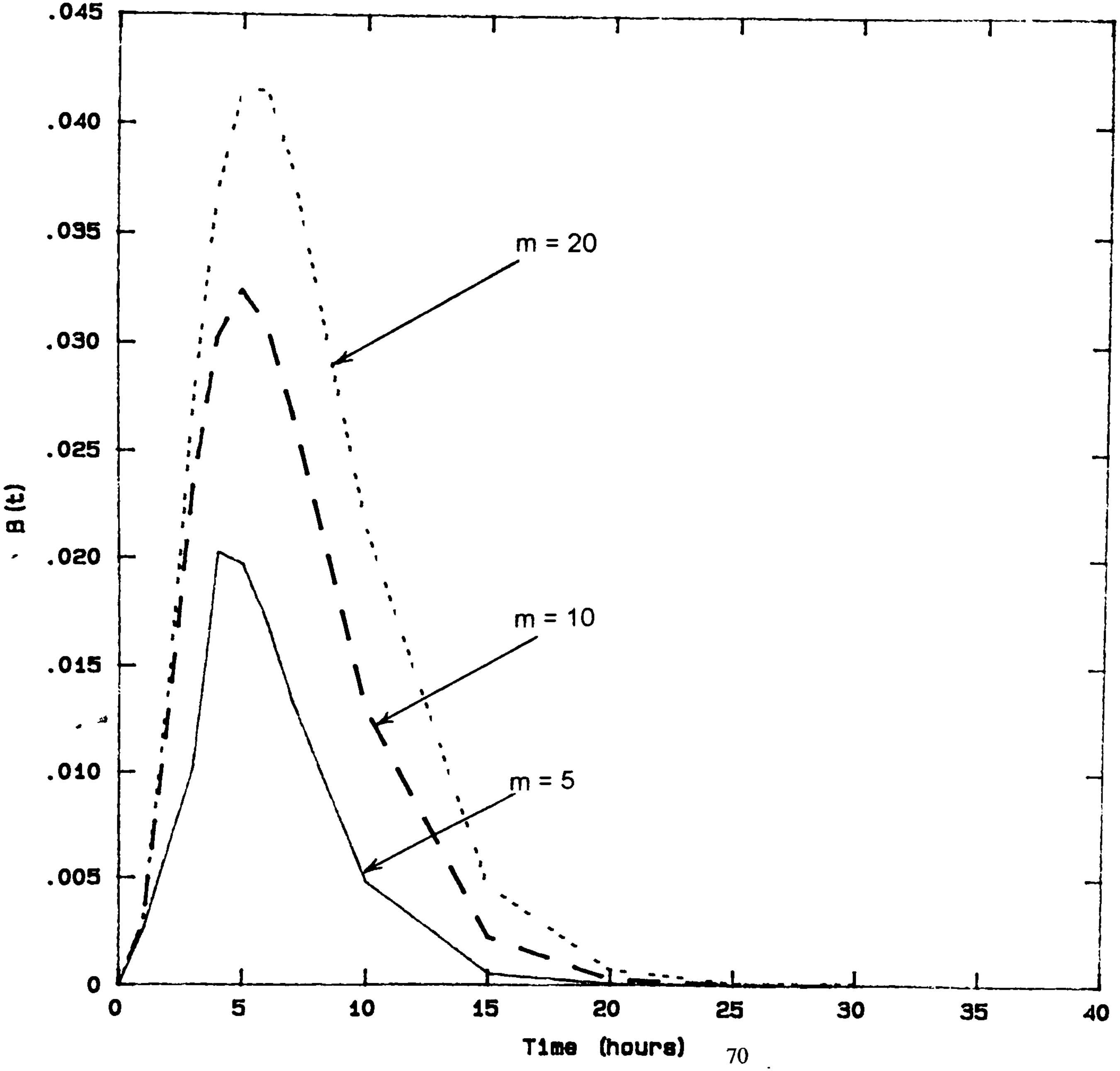
$$(s+3A+3B)\,Z(s) - 3i\,W(s) = 0 \tag{4.15}$$

$$(s+2i)\,X(s) - 3B\,Z(s) = 0 \tag{4.16}$$

$$s\,F(s) - 2i\,Z(s) = 0 \tag{4.17}$$

Figure 4.5

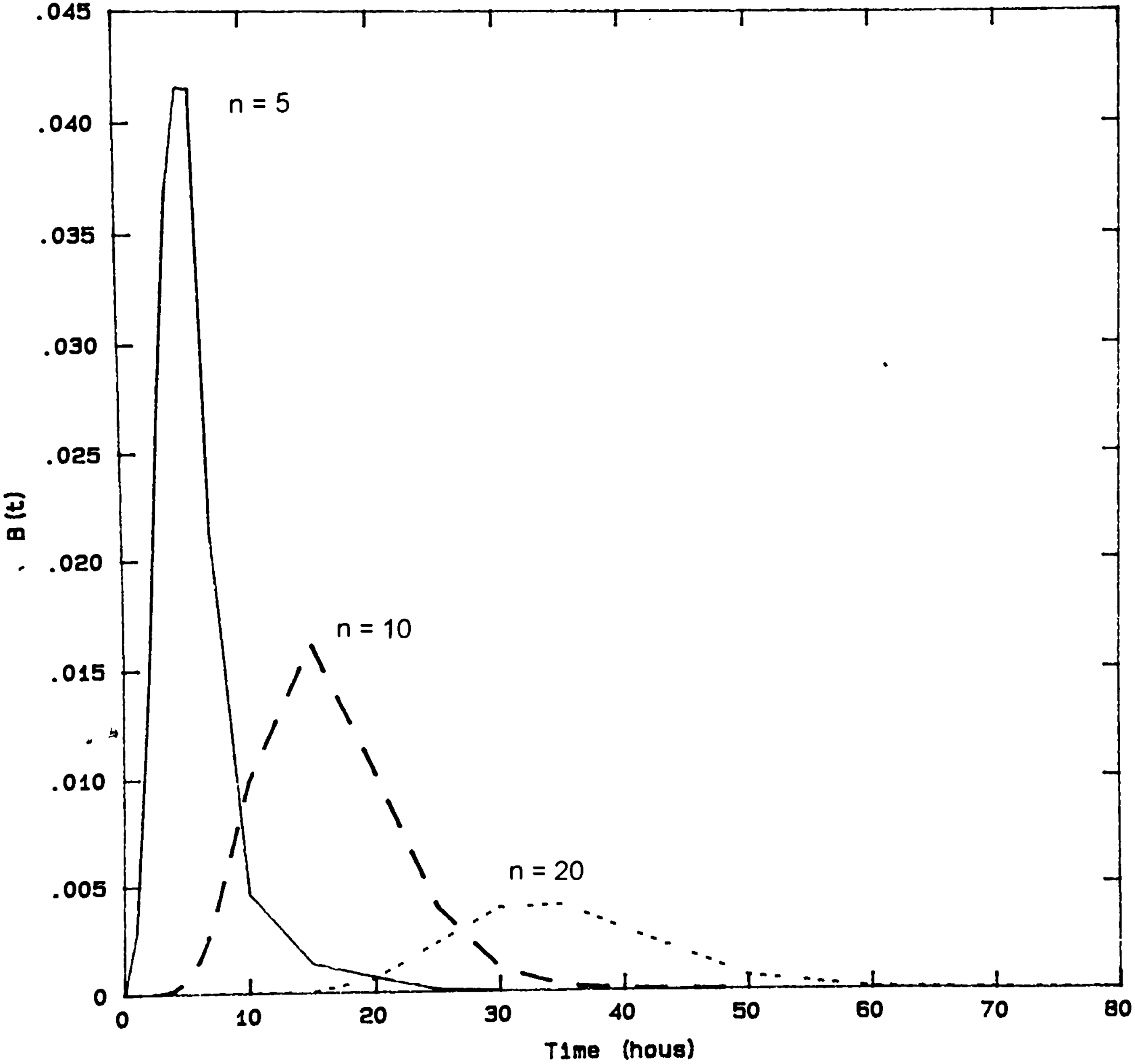
i = 0.5, n = 5, m = 5, 10, 20



The effect of the parameter M on B(t)

Figure 4.6

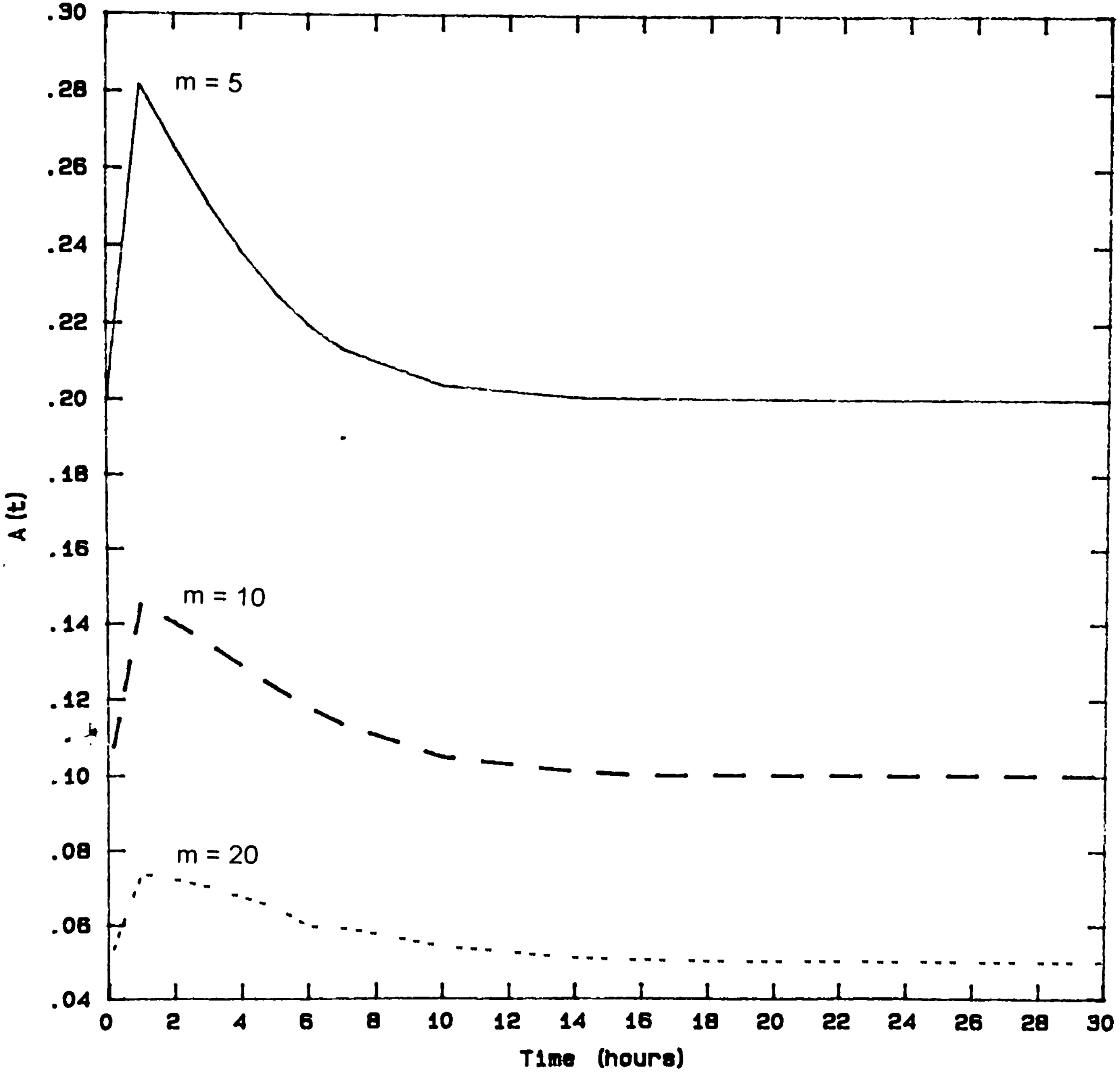
i = 0.5, m = 20, n = 5, 10, 20



The effect of the parameter N on B(t)

Figure 4.7

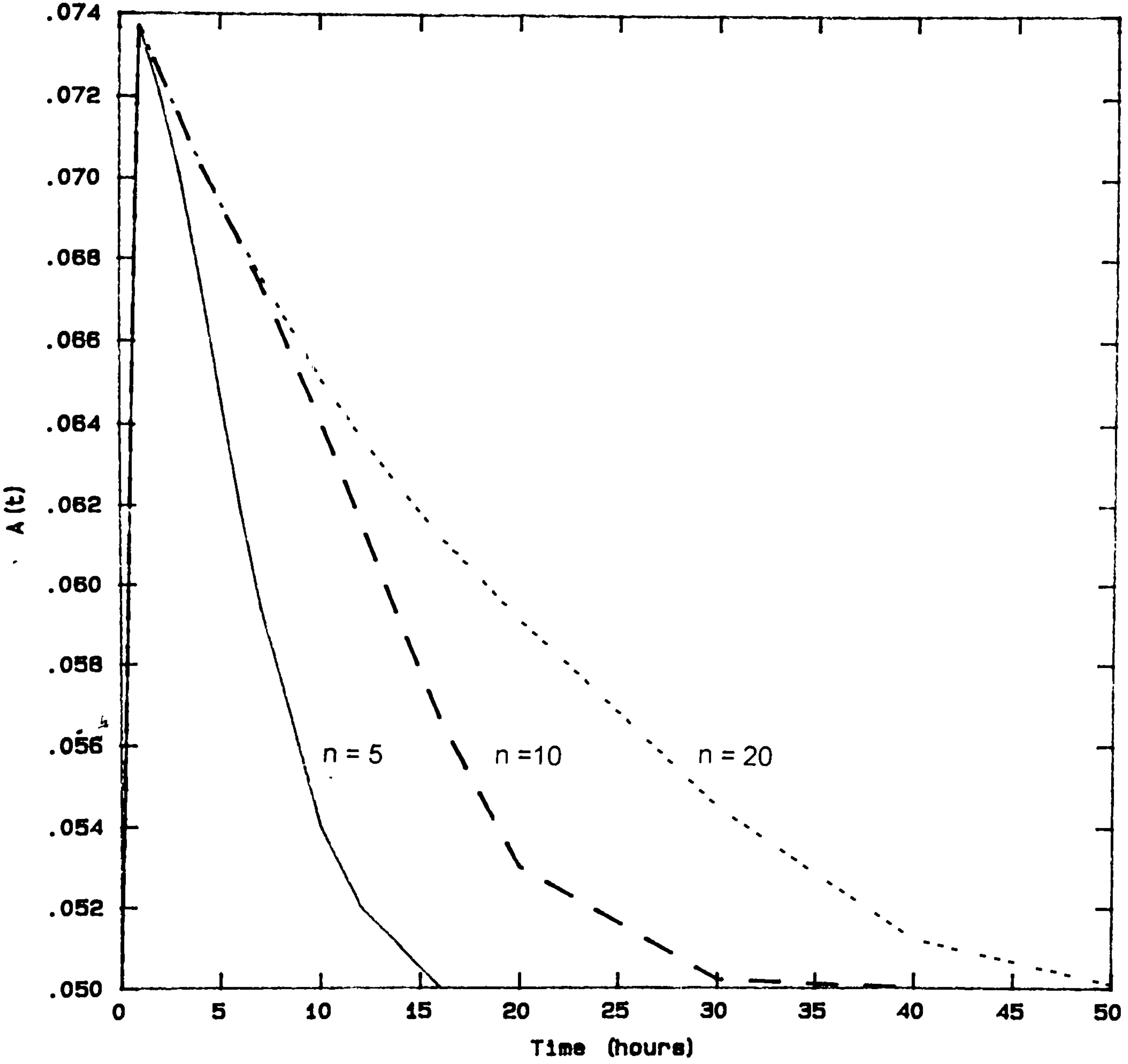
1 = 0.5, n = 5, m = 5, 10, 20



The effect of the parameter M
on A(t)

Figure 4.8

i = 0.5, m = 20, n = 5, 10, 20



The effect of the parameter N
on A(t)

From equations (4.12) and (4.13) we get:

$$V(s) = \frac{(s+4A+4B)}{(s+4i)(s+4A+4B)-16iA} \tag{4.18}$$

$$Y(s) = \frac{4i}{(s+4i)(s+4A+4B)-16iA} \tag{4.19}$$

From equation (4.15) we get:

$$W(s) = \frac{(s+3A+3B)}{3i} Z(s) \tag{4.20}$$

And from equation (4.16) we get:

$$-4B\,Y(s) \;+\; (s+3i)\,W(s) \;-\; 3A\,Z(s) \;=\; 0 \tag{4.21}$$

Substituting for Y(s) and W(s) in the above equation yields:

$$-4B\left(\frac{4i}{(s+4i)(s+4A+4B)-16iA}\right) + (s+3i)\left(\frac{s+3A+3B}{3i}\right) Z(s) - 3A\,Z(s) = 0$$

Or,

$$Z(s) = \left[\frac{16iB}{(s+4i)(s+4A+4B)-16iA}\right]\left[\frac{3i}{(s+3i)(s+3A+3B)-9iA}\right] \tag{4.22}$$

Substituting the value for Z(s) above in equation 4.16 and rearranging give:

$$X(s) = \left[\frac{3B}{s+2i}\right]\left[\frac{16iB}{(s+4i)(s+4A+4B)-16iA}\right]\left[\frac{3i}{(s+3i)(s+3A+3B)-9iA}\right] \tag{4.23}$$

Substituting the value for X(s) above in equation 4.17 and rearranging give:

$$F(s) = \left[\frac{2i}{s}\right]\left[\frac{3B}{s+2i}\right]\left[\frac{16iB}{(s+4i)(s+4A+4B)-16iA}\right]\left[\frac{3i}{(s+3i)(s+3A+3B)-9iA}\right]$$

Or,

$$F(s) = \frac{288\,i^3 B^2}{s(s+2i)(s+C_1)(s+C_2)(s+C_3)(s+C_4)} \tag{4.24}$$

where,

$$C_1 = 2(A+B+i) - 2\sqrt{A^2 + B^2 + 2AB + 2iA - 2iB}$$

$$C_2 = 2(A+B+i) + 2\sqrt{A^2 + B^2 + 2AB + 2iA - 2iB}$$

$$C_3 = \frac{3(A+B+i) - 3\sqrt{A^2 + B^2 + 2AB + 2iA - 2iB}}{2}$$

$$C_4 = \frac{3(A+B+i) + 3\sqrt{A^2 + B^2 + 2AB + 2iA - 2iB}}{2}$$

Notice that $C_3 = \frac{3}{4}C_1$, and $C_4 = \frac{3}{4}C_2$

Hence,

$$F(s) = \frac{288 i^3 B^2}{s(s+2i)(s+C_1)(s+C_2)(s+\frac{3}{4}C_1)(s+\frac{3}{4}C_2)} \qquad (4.25)$$

Because we are only interested in the reliability of the system, we only need to solve equation (4.25). First, taking the partial fraction expansions yields:

$$F(s) = \frac{K_1}{s} + \frac{K_2}{s+2i} + \frac{K_3}{s+C_1} + \frac{K_4}{s+C_2} + \frac{K_5}{s+\frac{3}{4}C_1} + \frac{K_6}{s+\frac{3}{4}C_2} \qquad (4.26)$$

where,

$$K_1 = \frac{256 i^2 B^2}{C_1^2\ C_2^2}$$

$$K_2 = \frac{288 i^3\ B^2}{(-2i)(C_1 - 2i)(C_2 - 2i)(\frac{3}{4}C_1 - 2i)(\frac{3}{4}C_2 - 2i)}$$

$$K_3 = \frac{1152 i^3 B^2}{C_1^2 (2i - C_1)(C_2 - C_1)(\frac{3}{4}C_2 - C_1)}$$

$$K_4 = \frac{1152 i^3 B^2}{C_2^2 (2i - C_2)(C_1 - C_2)(\frac{3}{4}C_1 - C_2)}$$

$$K_5 = \frac{1536\, i^3\, B^2}{(\frac{-3}{4})C_1^{\,2}\,(2i - \frac{3}{4}C_1)(C_2 - \frac{3}{4}C_1)(C_2 - C_1)}$$

$$K_6 = \frac{1536\, i^3\, B^2}{(\frac{-3}{4})C_2^{\,2}\,(2i - \frac{3}{4}C_2)(C_1 - \frac{3}{4}C_2)(C_1 - C_2)}$$

Taking the inverse LaPlace transforms yields:

$$F(t) = K_1 + K_2\, e^{-2it} + K_3\, e^{-C_1 t} + K_4\, e^{-C_2 t} + K_5\, e^{-\frac{3}{4}C_1 t} + K_6\, e^{-\frac{3}{4}C_2 t} \qquad (4.27)$$

And finally, the system reliability is given by:

$$R(t) = 1 - F(t) \qquad (4.28)$$

Now, consider a 3-out-of-5 majority voting system. This system involves the same level of hardware redundancy as our system (4 redundant processors). Assume that this system encounters intermittent faults only at a rate i failure/hour. Furthermore, assume that the intermittent faults have a maximum duration of one system decision (this was the definition of intermittent faults in our system). With these assumptions being made, the 3-out-of-5 system will fail only if it encounters three intermittent faults at the same time. To clarify, assume that from the fault-free state a processor (in the 3-out-of-5 system) encountered an intermittent fault putting the system in the four-working/one-faulty state. Then if during the next system decision another processor encountered another intermittent fault, this will also put the system in the four-operational/one-faulty state. This behavior is due to the assumption we made, which implies that the first faulty processor is now operational caused by the intermittent fault disappearance. Considering this, the system can be modeled with two states only: the all operational state and the system failed state. The transition from the first state to the second is i^3. Finally, the reliability of the 3-out-of-5 system is given by:

$$R(t)= e^{-i^3 t}$$

Figure 4.9 shows a comparison between our system, a simplex system, and a 3-out-of-5 system, with i = 0.5 intermittent faults/hour and without any repair. Although the assumptions we made favor the 3-out-of-5 system, our system is still more reliable than the other systems.

Figures 4.10 and 4.11 show the effect of the values M and N on the reliability of our system. As expected, decreasing M (or increasing 1/M, the "repair rate") improves the reliability of our system, and increasing N (the intermittent fault tolerance capability) also improves the system's reliability. In practice these values should be set depending on different factors, such as:

- The intermittent fault rate.
- The intermittent to permanent fault ratio.
- The percentage of intermittent faults that are caused by common factors (to all processors) to the total number of intermittent faults.
- The cruciality of the application.

Figure 4.9

1 = 0.5, p = 0, n = 10, 1/m = 0

System Comparison

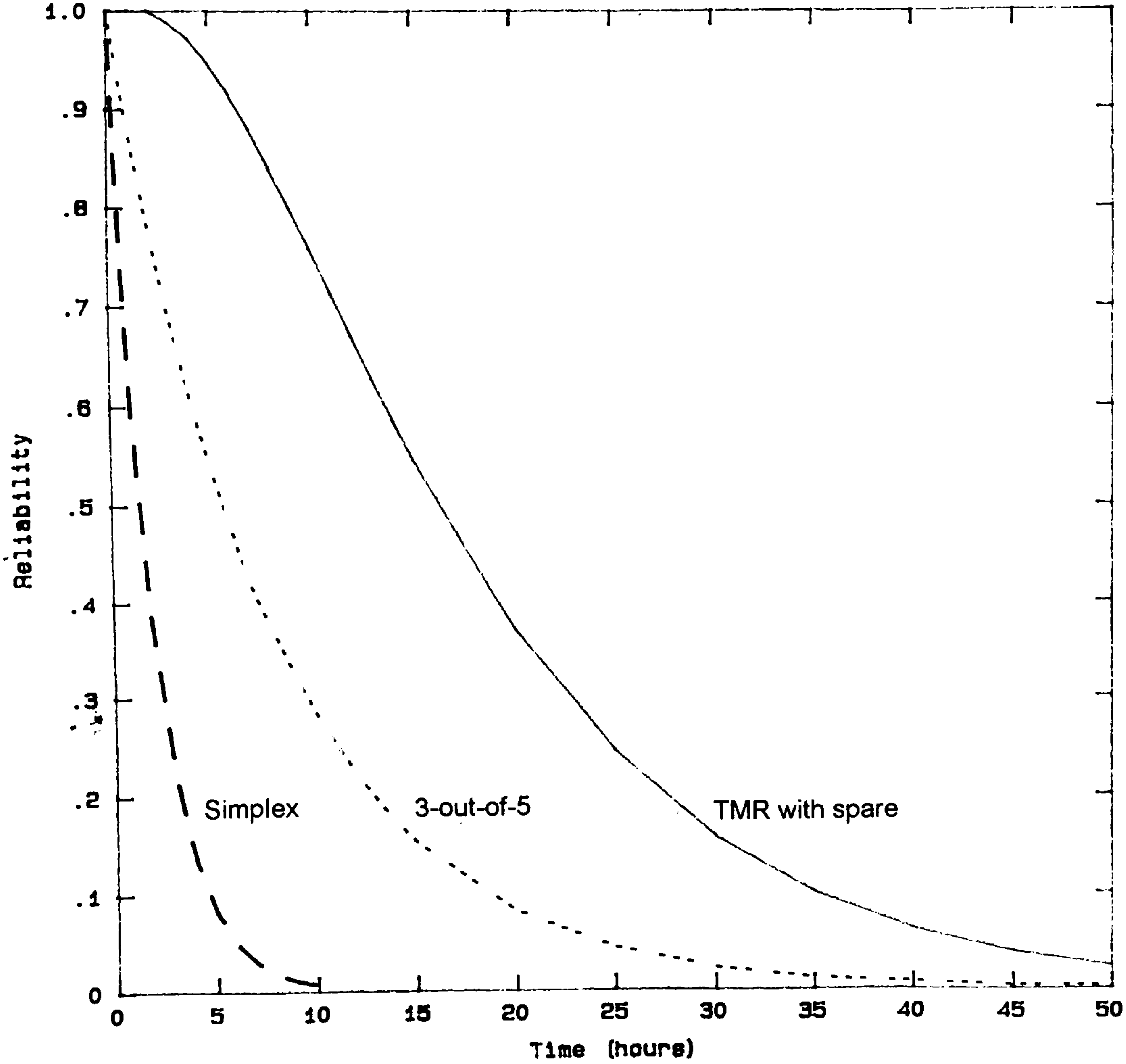
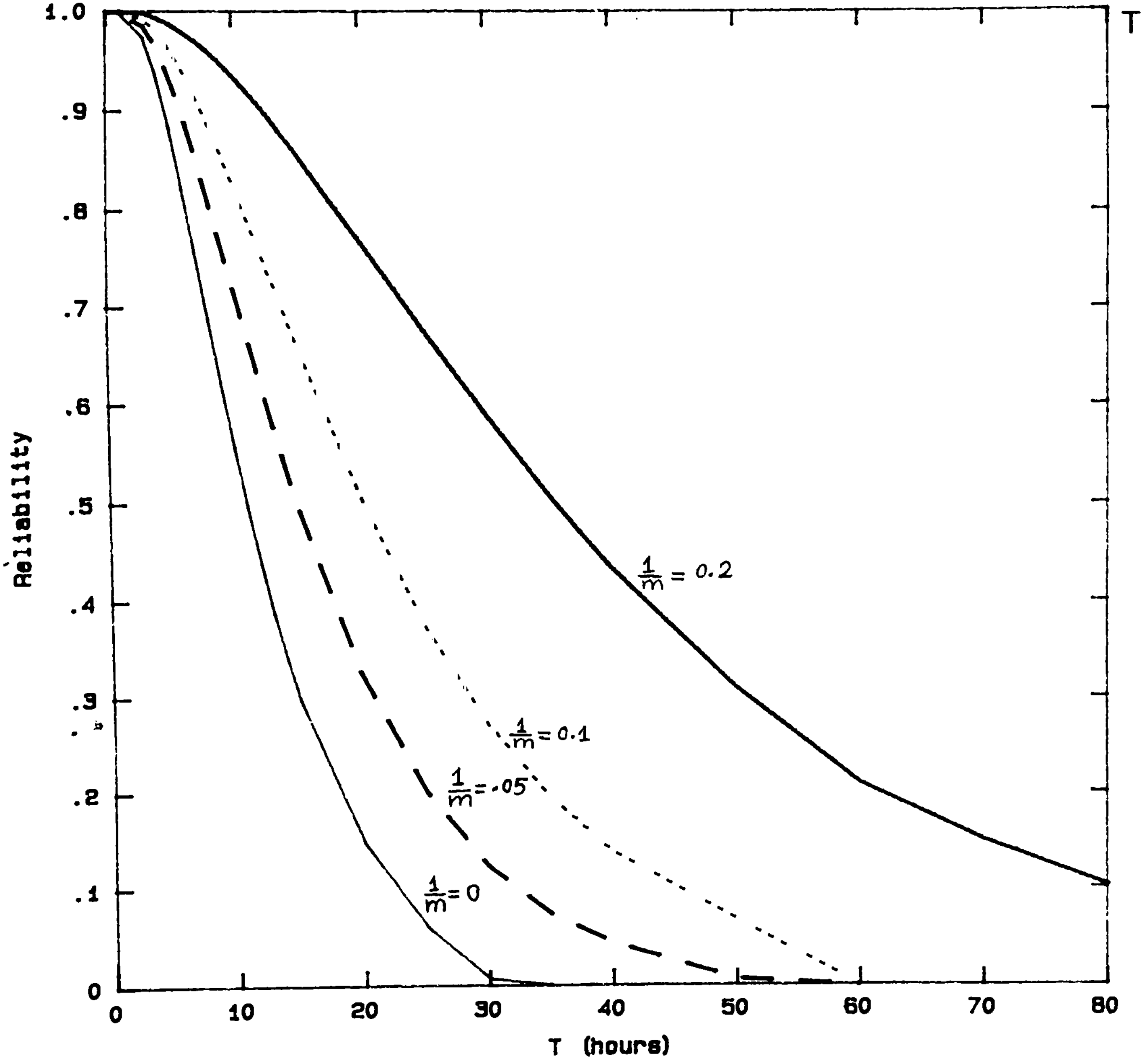


Figure 4.10

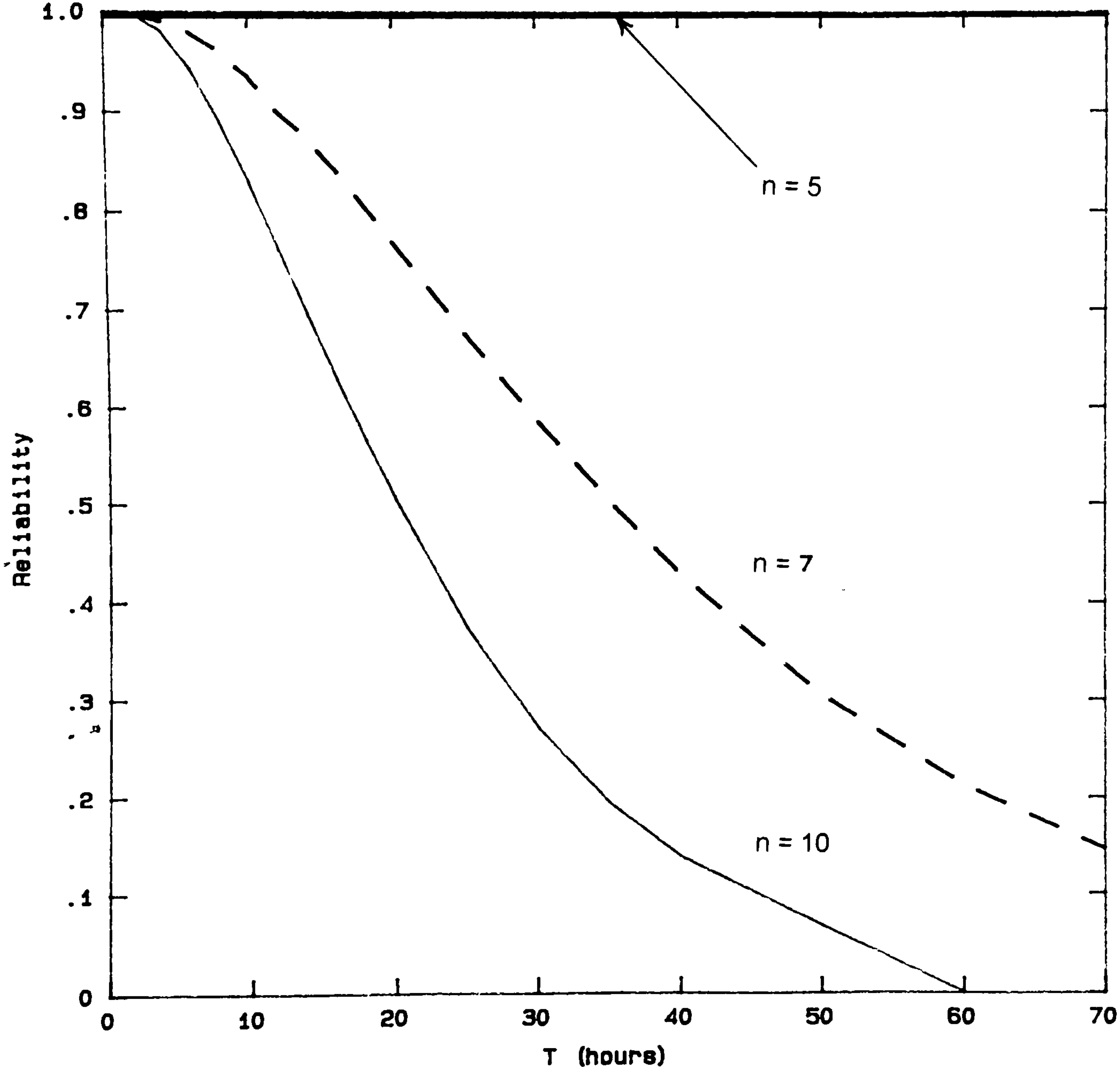
1 = 0.5, n = 5, 1/m = 0, 0.05, 0.1, 0.2



The effect of the parameter M an system Reliability

Figure 4.11

1 = 0.5, 1/m = 10, n = 5, 7, 10



The effect of the parameter N
system Reliability

5.0 Conclusion

The techniques and design issues involved in fault tolerant systems were discussed. Evaluation criteria were presented with some emphasis on reliability analysis.

The reliability of hardware fault-tolerant systems is usually done under permanent fault conditions. The effects of intermittent faults on system reliability are not fully considered. With statistical data suggesting that up to 90% of system failures may be caused by intermittent faults, reliability analysis techniques must concentrate more on this class of faults.

To realize the techniques and the design issues discussed, a reconfigurable Triple Modular Redundancy (TMR) with spare system was built. The general problem of synchronization in majority voting systems, causes and solutions, was presented and discussed. A design and reliability analysis approaches were introduced & implemented. The TMR with spare system was designed to tolerate a higher level of intermittent faults. This level is controlled by the parameters M & N which should be set based on the intermittent faults rate and their causes. A system with intermittent faults caused by global conditions affecting all its modules should use lower values of the parameter N. This is because the intermittent faults in this system will have a higher overlapping probability which may cause an undetectable system errors. The effect of these parameters on the system's reliability is shown in figures 4.10 & 4.11. Although these parameters reflect the level of confidence in the processors (i.e. a processor is considered fault free despite its encountering N intermittent faults in M hours), the parameter M (for example) can be thought of as the processors repair rate which explains the reliability value of one when N = 1/M in figure 4.11.

Reliability analysis of the system under permanent fault conditions showed a higher system reliability compared to other systems especially for short time applications (see figure 2.12). Reliability analysis under intermittent fault conditions showed a very big improvement over other redundant (3-out-of-5) and non-redundant (simplex) systems without any restriction by the intended application life (see figure 4.9 and compare it to figure 2.12). Furthermore, keep in mind that these results were obtained despite the worst case assumptions for our system and a favorable assumptions for the 3-out-of-5 system which emphasizes the importance of these results.

Although this work did not contain any actual fault statistics, the design and reliability approaches presented in this work are expected to be useful and, hopefully, more accurate than the traditional techniques once the values of the parameters M & N are carefully set. Determining the optimal values of these parameters can be the subject of some further work in this area, which may also involve the use of techniques presented in modeling non-overlapping permanent faults in fault-tolerant systems.

References

[1] Barry Johnson, "Design and Analysis of Fault-Tolerant Digital Systems", Addison Wesley, Mass. 1989.

[2] A. E. Barbour and A. S. Wojcik, "A General Constructive Approach to Fault-tolerant Design Using Redundancy", IEEE Trans. on Comp. Jan. 1989.

[3] D. Siewiorek, "Reliability Modeling of Compensating Module Failures in Majority Voted Redundancy", IEEE Trans. on Comp. May 1975.

[4] D. Siewiorek, G. York, and Y. X. Zhu, "Compensating Faults in Triple Modular Redundancy", Digest of Papers of the 15th Fault Tolerant Computing Symposium, June 1985.

[5] Charles Stroud and Ahmed Barbour, "Reliability, Testability, and Yield of Majority Voting VLSI", IEEE Proceedings, May 1990.

[6] Flavin Cristian, "Understanding Fault-Tolerant Distributed Systems", Communications of the ACM, February 1991.

[7] Robert Geist and Kishov Trivedi, "Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques. IEEE Trans. on Comp. July 1990.

[8] Daniel Siewiorek, "Fault-Tolerance in Commercial Computers", IEEE Trans. on Comp., July 1990.

[9] Guy A. S. Wingate and Clive Preece, "A Software-Implemented Fault-Tolerant Techniques for Microprocessor Controllers", IEEE 1993 Proceedings., Annual Reliability and Maintainability Symposium.

[10] M. Balakrishuan and C. S. Raghavendra, "An Analysis of a Reliability Model for Repairable Fault-Tolerant Systems", IEEE Trans. On Comp. March 1993

[11] M. A. Friedman, P. Y. Tran, and P. L. Gaddard, "Reliability Techniques for Combined Hardware and Software Systems", RL-TR-92-15, Final Report. Rome Laboratory, Air Force system Command. February 1992.

[12] Lavon B. Page and Jo Ellen Perry, " A Model for System Reliability with Common-Cause failures", IEEE Trans. on Reliability, October 1989.

[13] Yinong Chen and Tinghuai Chen, "Implementing Fault-Tolerance via Modular Redundancy with Comparison", IEEE Trans. on Reliability, June 1990

[14] Daniel Barbara and Hector Garcia-Molina, "The Reliability of Voting Mechanisms", IEEE Trans. On Comp. October 1987.

Appendix

```
**************************************************************************
****    Program  : cont1.asm
****    By       : Khalid Al-kofahi
****    Date     : May 25, 1993
****    Descrption :
****            This program is executed by the controller of module 1. It tests all three processors and
****            the spare for intermittent and permanent faults, and reconfigure the module according-
****            ly. It also controlls the RS232 to the terminal and displays the headder messages and
****            the processors' fault data. The operation proceeds as follows :
****            - First display headder message and reset all processors ( procedures mess1 & first).
****            - Wait for the serial clock, when recieved : test the serial data and the serial clock
****            disagreement detectors ( d1 .. d8 ). Repeat six times ( for the six clock cycles) and store
****            the results in d1_p,d2_p,d3_p,d4_p,c1_p,c2_p,c3_p,c4_p respectively.
****            - Wait for the laod signal, when recieved : test the load signals disagreement detectors
****             ( d9...d12) and store the resutl in l1,l2,l3,l4 respectively.
****            - Note :
****                    di_p : # of faults encountered during the last decision on processor i data line.
****                    ci_p : # of faults encountered during the last decision on processor i clock line.
****                     li_p : # of faults encountered during the last decisionon processor i load line.
****                     di   : total # of faults on processor i data line.
****                     ci   : total # of faults on processor i clock line.
****                      li  : total # of faults on processor i load line.
****                       ( processor # 4 is the spare).
****
****            - Check all the di's, ci's and li's registers and reconfigure if needed.
****            - Upon module failure, flash the module four times and start all over ( if this is the first
****            failure) else go to flash indifinitely.
****
****            Online comments clarify some of the detailes in the program.
**************************************************************************

        org     $784
        fcb     $07

        org     $7fe    * start at location $100 after reset
        fcb     $01
        fcb     $00
        end
port_a  equ     $000    * refer to figure 3.3 for actual connections
port_b  equ     $001
port_c  equ     $002
timer   equ     $008
a_cntl  equ     $004
b_cntl  equ     $005
c_cntl  equ     $006
t_cntl  equ     $009

t_mode  equ     $47     * external clock, 128 prescalar
a_mode  equ     $00
b_mode  equ     $e0
c_mode  equ     $fe

d1      equ     $10
c1      equ     $11
```

```
l1      equ     $12
d2      equ     $13
c2      equ     $14
l2      equ     $15
d3      equ     $16
c3      equ     $17
l3      equ     $18
d4      equ     $19
c4      equ     $1A
l4      equ     $1B
d1_p    equ     $1C
c1_p    equ     $1D
l1_p    equ     $1E
d2_p    equ     $1F
c2_p    equ     $20
l2_p    equ     $21
d3_p    equ     $22
c3_p    equ     $23
l3_p    equ     $24
d4_p    equ     $25
c4_p    equ     $26
l4_p    equ     $27
cont    equ     $28             * cont : contains N.
cont1   equ     $29             * conti : a flag register to test for permanent faults on processor i
cont2   equ     $2A             * load lines, if it is == 2, then processor i has permanent fault on
cont3   equ     $2B             * its load line. It is cleared every two cycles.
cont4   equ     $2C
cont5   equ     $2D             * contains the number of faulty processors
temp1   equ     $2E
temp2   equ     $2F
res     equ     $30
dis     equ     $31
status  equ     $32
again   equ     $33
reg1    equ     $34
reg2    equ     $35             * contains the data to be printed as ascii
tx      equ     $36             * transmit register
acc     equ     $37
indx    equ     $38
max     equ     15              * maximum number of intermittent faults allowed (M)
cr      equ     $0D
lf      equ     $0A
spc     equ     $20

        org     $100
        sei
        lda     #a_mode
        sta     a_cntl
        lda     #b_mode
        sta     b_cntl
        lda     #c_mode
        sta     c_cntl
        lda     #t_mode
        sta     t_cntl
        clr     port_a
        clr     port_b          *select original processors ( proc 1, 2, & 3)
        clr     port_c
        clr     again

        bset    3,port_c        * tx = high
*******************************************************************

main    bclr    2,port_c        * hold RS232
        jsr     delay1
        jsr     delay1
        jsr     mess1
        bset    2,port_c        * release RS232
        jsr     first           * reset all processors
loop1   jsr     secnd
        ldx     #6
wait    brclr   4,port_b,wait   * wait for the serial clock
        jsr     delay
        jsr     test1           * test d1...d8 lines
        decx
        bne     wait            * repeat six times

wait2   brclr   0,port_c,wait2  * wait for load signal
        jsr     delay
        jsr     test2           * test d9 .. d12

        jsr     proc1           * test processor 1 lines
        jsr     proc2           * test processor 2 lines
        jsr     proc3           * test processor 3 lines
        jsr     proc4           * test spare processore lines
        clr     cont5           * will contain the number of faulty processors

        brclr   1,status,p2
        inc     cont5           * processor 1 is faulty
        bset    5,port_b        * reconfigure with spare
p2      brclr   2,status,p3
        inc     cont5           * processor 2 is faulty
        bset    6,port_b        * reconfgure with spare
p3      brclr   3,status,p4
        inc     cont5           * processor 3 is faulty
        bset    7,port_b        * reconfigure with spare
p4      brclr   4,status,p5
        inc     cont5
p5      lda     cont5
        cmp     #2              * number of faulty proeccors ?
        blo     ok
        bra     rest            * module fail
ok      jsr     display         * write status message on screen
        dec     cont
        bne     loop1
        bra     main            * continue

***************************************************************************
jumpp   bra     main            * just an intermediate jump
```

```
*******************************************************************
rest    lda     again           * is it the first failure
        cmp     #1
        beq     flash           * this the second failure, goto flash and stay there
        inc     again           * increase number of module failures

        bclr    2,port_c        * hold RS232
        jsr     delay1
        jsr     delay1
        jsr     mess6           * print message
        bset    2,port_c        * release RS232

        lda     #4              * flash 4 times then restart
        sta     acc

rest1   bclr    1,port_c        * reset processors
        jsr     delay1
        bset    1,port_c
        ldx     #24             * length of reset signal
rest2   jsr     delay1
        decx
        bne     rest2

        dec     acc
        bne     rest1
        bclr    1,port_c        * send reset
        clr     port_b          * select original processors
        bra     jumpp
*****************************************************************************
*** If here then module already failed *************************************************

flash   jsr     delay1          * 0.1305 sec
        jsr     delay1
        jsr     mess7           * cr & lf
        jsr     mess7
        jsr     mess5
        bset    2,port_c        * release RS232

flash1  bclr    1,port_c
        jsr     delay1
        bset    1,port_c
        ldx     #24
loop    jsr     delay1
        decx
        bne     loop
        bra     flash1
****************************************************************************

first   bclr    1,port_c        * reset signal
        jsr     delay1
        bset    1,port_c
        clrx
loop2   clr     $10,x
        incx
        cpx     #35
        bne     loop2
        lda     #100
        sta     cont            * initialize N to 100
        rts

**********************************************************************
secnd   clr     d1_p
        clr     d2_p
        clr     d3_p
        clr     d4_p
        clr     c1_p
        clr     c2_p
        clr     c3_p
        clr     c4_p
        clr     cont5
        rts
**********************************************************************
*** test1 : test d5.. d8 & d1..d4 ********************************************

test1   brclr   4,port_a,t2     *d5
        inc     c1_p
t2      brclr   5,port_a,t3     *d6
        inc     c2_p
t3      brclr   6,port_a,t4     *d7
        inc     c3_p
t4      brclr   7,port_a,t5     *d8
        inc     c4_p
t5      brclr   0,port_a,t6     *d1
        inc     d1_p
t6      brclr   1,port_a,t7     *d2
        inc     d2_p
t7      brclr   2,port_a,t8     *d3
        inc     d3_p
t8      brclr   3,port_a,t9     *d4
        inc     d4_p
t9      rts

******************************************************************************
*** test2 :  test d9..d12 ***********************************************************

test2   brclr   0,port_b,a1     *d9
        inc     l1_p
a1      brclr   1,port_b,a2     *d10
        inc     l2_p
a2      brclr   2,port_b,a3     *d11
        inc     l3_p
a3      brclr   3,port_b,a4     *d12
        inc     l4_p
a4      rts

******************************************************************************
***  test processor 1 ***********************************************************
```

```
proc1   lda     #d1
        sta     temp1         * temp1 = $10 : address of d1
        add     #3
        sta     temp2         * temp2 = $13 : address of d2
        jsr     test3         * returns wether the processor is faulty or not ( bit 0 of register res)
        brclr   0,res,ok1     * if not cleared, the processor is faulty
        bset    1,status      * a flag for the main to indicate processor 1 is faulty
ok1     lda     11_p
        cmp     #0
        beq     p1end
        inc     cont1         * cont1 is clered every two consecutive cycles, if it is = 2, then
        lda     cont1         * the load line of this processor has a permanent fault
        cmp     #2
        bne     p1end
        clr     cont1
        clr     11_p
        dec     11
p1end   rts

****************************************************************************
**** proc2 : the same as proc1 but this tests processor 2 instead *************************

proc2   lda     #d2
        sta     temp1
        add     #3
        sta     temp2
        jsr     test3
        brclr   0,res,ok2
        bset    2,status
ok2     lda     12_p
        cmp     #0
        beq     p2end
        inc     cont2
        lda     cont2
        cmp     #2
        bne     p2end
        clr     12_p
        clr     cont2
        dec     12
p2end   rts

**********************************************************************
*** proc3: the same as proc1 but this tests proecssor 3 lines . *********************

proc3   lda     #d3
        sta     temp1
        add     #3
        sta     temp2
        jsr     test3
        brclr   0,res,ok3
        bset    3,status
ok3     lda     13_p
        cmp     #0
        beq     p3end
```

```
         inc     cont3
         lda     cont3
         cmp     #2
         bne     p3end
         clr     l3_p
         clr     cont3
         dec     l3
p3end    rts

*******************************************************************
**** proc4 : the same as proc1 but this test the spare lines . **********************

proc4    lda     #d4
         sta     temp1
         add     #3
         sta     temp2
         jsr     test3
         brclr   0,res,ok4
         bset    4,status
ok4      lda     l4_p
         cmp     #0
         beq     p4end
         inc     cont4
         lda     cont4
         cmp     #2
         bne     p4end
         clr     l4_p
         clr     cont4
         dec     l4
p4end    rts

*******************************************************************
**** test3 : tests the registers between the adresses temp1 & temp2 . these registers contain the
****         processors' faults information. **********************************

test3    clr     res
         ldx     temp1             * contains the address of di , i = 1,2,3,4
loop3    lda     $c,x
         cmp     #2                * two consecutive faults = permanent
         blo     b1
         bset    0,res             * processor is faulty  due to permanent faults
b1       add     0,x
         sta     0,x
         incx
         cmp     #max              *  processor's intermittent faults > N ?
         bls     b2
         bset    0,res             * processor is fualty due to intermittent faults
b2       cpx     temp2             * finished with this processor data ?
         bne     loop3
         rts
*****************************************************************
**** convert data in reg2 to ascii and call send for screen display ****************

ascii    lda     reg2
```

```
        and     #$f0
        lsra
        lsra
        lsra
        lsra
        cmp     #9
        bls     h1
        add     #7
h1      add     #$30
        sta     tx
        jsr     send            * send the MSD

        lda     reg2
        and     #$0f
        cmp     #9
        bls     h2
        add     #7
h2      add     #$30
        sta     tx
        jsr     send            * send LSD
        rts
*****************************************************************
**** show processors' fault information ************************************

display inc     dis
        lda     dis
        cmp     #7
        beq     wait3
        cmp     #5
        bne     wait4
        bclr    2,port_c        * hold RS232
        jsr     delay1
        jsr     delay1
        jsr     disp1           * show processors' status
        bset    2,port_c        * release RS232
        bra     wait4
wait3   clr     dis
        inc     dis
wait4   rts
*****************************************************************
*** show processors' status. statrting at $10 (or d1) and ending at $1b (or l4) **********

disp1   jsr     mess7           * cr, lf
        lda     #spc            * space
        sta     tx
        jsr     send            * print space
        sta     tx
        jsr     send
        clrx
disp2   lda     $10,x           * fault information at location $10+x
        sta     reg2
        jsr     ascii           * convert them to ascii
        lda     #spc            * space
        sta     tx
        jsr     send
        sta     tx
        jsr     send
        incx
        cpx     #12             * did I reach 14 or not yet ?
        bne     disp2
        brclr   1,status,m2     * is processor 1operational ?
        jsr     noo             * no it is not, display N in its column on the screen.
        bra     m3
m2      jsr     yess            * yes it is , then disply Y in its column on the screen.
m3      brclr   2,status,m4     * is processor 2 operational ?
        jsr     noo
        bra     m5
m4      jsr     yess
m5      brclr   3,status,m6     * is processor 3 operational ?
        jsr     noo
        bra     m7
m6      jsr     yess
m7      ldx     #5
        jsr     space
        lda     cont5
        cmp     #0
        beq     m8
        brset   4,status,m8     * is the spare considered in the voting  process ?
        jsr     yess            * yes it is, then display Y in its column on the screen.
        bra     m9
m8      jsr     noo             * no it is not, display N in its column on the screen.
m9      ldx     #6
        jsr     space           * print space, number of times as in reg. x.
        lda     #1
        sta     reg2
        jsr     ascii
        rts
*****************************************************************
**** this procedure sends the data in register tx to the screen. one start bit , two stop bits.

send    sta     acc
        lda     #08
        sta     reg1
        bclr    3,port_c        * send a start bit
        jsr     delay2          * 80 cycles ( one bit)
tx3     ror     tx
        bcs     tx1
tx0     bclr    3,port_c
        bra     tx4
tx1     bset    3,port_c
tx4     jsr     delay           * 70 cycles
        dec     reg1
        bne     tx3

        inc     reg1            * dummy delay
        nop
        nop
tx5     bset    3,port_c        *send stop bits  (2)
```

```
        jsr     delay2
        jsr     delay2
        lda     acc
        rts
************************************************************************
**** the following procedures are for display purposes only ***********************

mess1   jsr     mess7           * cr , lf
        jsr     mess7
        clrx
more    lda     mes1,x
        cmp     #$24            * all messages ends with $24 as a flag
        beq     done
        sta     tx
        jsr     send
        incx
        bra     more
done    jsr     mess7
        rts

yess    clrx
more2   lda     yes,x
        cmp     #$24
        beq     done2
        sta     tx
        jsr     send
        incx
        bra     more2
done2   rts

noo     clrx
more3   lda     no,x
        cmp     #$24
        beq     done3
        sta     tx
        jsr     send
        incx
        bra     more3
done3   rts

mess5   jsr     mess7           * cr, lf
        clrx
more5   lda     mes5,x
        cmp     #$24
        beq     done5
        sta     tx
        jsr     send
        incx
        bra     more5
done5   jsr     mess7           * cr, lf
        rts

mess6   stx     indx
        jsr     mess7           * cr, lf
        clrx
more6   lda     mes6,x
        cmp     #$24
        beq     done6
        sta     tx
        jsr     send
        incx
        bra     more6
done6   jsr     mess7
        ldx     indx
        rts

space   lda     #spc            * print k spaces, k = contents of reg. x
space1  sta     tx
        jsr     send
        decx
        bne     space1
        rts

mess7   stx     indx          * cr & lf
        ldx     #0
more7   lda     mes7,x
        cmp     #$24
        beq     done7
        sta     tx
        jsr     send
        incx
        bra     more7
done7   ldx     indx
        rts
*******************************************************************
**** all the delay procedures are the same but they have different lengths

delay   lda     #6              * 70 cycles
loop4   nop
        deca
        bne     loop4
        nop
        rts

delay2  lda     #7              * 80 cycles
loop5   nop
        deca
        bne     loop5
        nop
        rts

delay1  bclr    7,t_cntl        * 0.1305 SEC, used to regulate the traffic lights timing
        lda     #$ff
        sta     timer
self    brclr   7,t_cntl,self
        rts
*******************************************************************
```

```
        org     $600
mes1    FCC     /   P1      P2      P3    Spare  /
        FCB     $0D,$0A
        FCC     / A  B  C  D  E  F  G  H  I  J  K  L /        * see figure 3.3 for meaning
        FCC     /  P1  P2  P3   Spare  Module/
        FCB     $24
yes     FCC     / Y /
        FCB     $24
no      FCC     / N /
        FCB     $24
mes5    FCC     / FATAL ERROR ======> Module -1- FLASHING <======/
        FCB     $24
mes6    FCC     / ---- Module -1- First Failure, Try Again ----/
        FCB     $24
mes7    FCB     $0D,$0A,$24
```

```
********************************************************************************
****     Program : cont2.asm
****     BY      : Khalid All-kofahi
****     Date    : May 25, 1993
****     Description:
****             This program is executed by the controller of module2. It is almost the same as
****             cont1.asm, the only difference is that this controller does not write the header messages,
****             and it waits for the first controller ( cont1.asm) to release the RS232 to be able to use it.
*********************************************************************************

         org     $784
         fcb     $07

         org     $7fe
         fcb     $01
         fcb     $00
         end
port_a   equ     $000
port_b   equ     $001
port_c   equ     $002
timer    equ     $008
a_cntl   equ     $004
b_cntl   equ     $005
c_cntl   equ     $006
t_cntl   equ     $009

t_mode   equ     $47
a_mode   equ     $00
b_mode   equ     $e0
c_mode   equ     $fa

d1       equ     $10
c1       equ     $11
l1       equ     $12
d2       equ     $13
c2       equ     $14
l2       equ     $15
d3       equ     $16
c3       equ     $17
l3       equ     $18
d4       equ     $19
c4       equ     $1A
l4       equ     $1B
d1_p     equ     $1C
c1_p     equ     $1D
l1_p     equ     $1E
d2_p     equ     $1F
c2_p     equ     $20
l2_p     equ     $21
d3_p     equ     $22
c3_p     equ     $23
l3_p     equ     $24
d4_p     equ     $25
c4_p     equ     $26
```

```
l4_p    equu    $27
cont    equ     $28
cont1   equ     $29
cont2   equ     $2A
cont3   equ     $2B
cont4   equ     $2C
cont5   equ     $2D
temp1   equ     $2E
temp2   equ     $2F
res     equ     $30
dis     equ     $31
status  equ     $32
again   equ     $33
reg1    equ     $34
reg2    equ     $35    * data to printed as ascii
tx      equ     $36
acc     equ     $37
indx    equ     $38
max     equ     15
cr      equ     $0D
lf      equ     $0A
spc     equ     $20

        org     $100
        sei
        lda     #a_mode
        sta     a_cntl
        lda     #b_mode
        sta     b_cntl
        lda     #c_mode
        sta     c_cntl
        lda     #t_mode
        sta     t_cntl
        clr     port_a
        clr     port_b
        clr     port_c
        clr     again

        bset    3,port_c        * tx = high
******************************************************************

main    jsr     first
loop1   jsr     secnd
        ldx     #6
wait    brclr   4,port_b,wait
        jsr     delay
        jsr     test1
        decx
        bne     wait

wait2   brclr   0,port_c,wait2
        jsr     delay
        jsr     test2
        jsr     proc1
        jsr     proc2
        jsr     proc3
        jsr     proc4
        clr     cont5

        brclr   1,status,p2
        inc     cont5
        bset    5,port_b
p2      brclr   2,status,p3
        inc     cont5
        bset    6,port_b
p3      brclr   3,status,p4
        inc     cont5
        bset    7,port_b
p4      brclr   4,status,p5
        inc     cont5
p5      lda     cont5
        cmp     #2
        blo     ok
        bra     rest
ok      jsr     display
        dec     cont
        bne     loop1
        bra     main

***************************************************************************
jumpp   bra     main
***************************************************************************
rest    lda     again
        cmp     #1
        beq     flash
        inc     again
        lda     #4              * flash 4 times then restart
        sta     acc
rest1   bclr    1,port_c
        jsr     delay1
        bset    1,port_c
        ldx     #24
rest2   jsr     delay1
        decx
        bne     rest2
        dec     acc
        bne     rest1
        bclr    1,port_c        * send reset
        clr     port_b          * select original processors
rs      brclr   2,port_c,rs     * wait for RS232
        jsr     mess6
        bra     jumpp
******************************************************************************
flash   nop
rs2     brclr   2,port_c,rs2    * wait for RS232
        jsr     mess7
        jsr     mess7
        jsr     mess5
flash1  bclr    1,port_c
        jsr     delay1
        bset    1,port_c
        ldx     #24
loop    jsr     delay1
        decx
        bne     loop
        bra     flash1
*****************************************************************************

first   bclr    1,port_c
        jsr     delay1
        bset    1,port_c * reset
        clrx
loop2   clr     $10,x
        incx
        cpx     #35
        bne     loop2
        lda     #100
        sta     cont
        rts
****************************************************************************

secnd   clr     d1_p
        clr     d2_p
        clr     d3_p
        clr     d4_p
        clr     c1_p
        clr     c2_p
        clr     c3_p
        clr     c4_p
        clr     cont5
        rts
***************************************************************************

test1   brclr   4,port_a,t2
        inc     c1_p
t2      brclr   5,port_a,t3
        inc     c2_p
t3      brclr   6,port_a,t4
        inc     c3_p
t4      brclr   7,port_a,t5
        inc     c4_p
t5      brclr   0,port_a,t6
        inc     d1_p
t6      brclr   1,port_a,t7
        inc     d2_p
t7      brclr   2,port_a,t8
        inc     d3_p
t8      brclr   3,port_a,t9
        inc     d4_p
t9      rts
****************************************************************************
```

```
test2   brclr   0,port_b,a1
        inc     l1_p
a1      brclr   1,port_b,a2
        inc     l2_p
a2      brclr   2,port_b,a3
        inc     l3_p
a3      brclr   3,port_b,a4
        inc     l4_p
a4      rts
*************************************************************************

proc1   lda     #d1
        sta     temp1
        add     #3
        sta     temp2
        jsr     test3
        brclr   0,res,ok1
        bset    1,status
ok1     lda     l1_p
        cmp     #0
        beq     p1end
        inc     cont1
        lda     cont1
        cmp     #2
        bne     p1end
        clr     cont1
        clr     l1_p
        dec     l1
p1end   rts

*************************************************************************

proc2   lda     #d2
        sta     temp1
        add     #3
        sta     temp2
        jsr     test3
        brclr   0,res,ok2
        bset    2,status
ok2     lda     l2_p
        cmp     #0
        beq     p2end
        inc     cont2
        lda     cont2
        cmp     #2
        bne     p2end
        clr     l2_p
        clr     cont2
        dec     l2
p2end   rts
*************************************************************************

proc3   lda     #d3
```

```
        sta     temp1
        add     #3
        sta     temp2
        jsr     test3
        brclr   0,res,ok3
        bset    3,status
ok3     lda     l3_p
        cmp     #0
        beq     p3end
        inc     cont3
        lda     cont3
        cmp     #2
        bne     p3end
        clr     l3_p
        clr     cont3
        dec     l3
p3end   rts

****************************************************************************

proc4   lda     #d4
        sta     temp1
        add     #3
        sta     temp2
        jsr     test3
        brclr   0,res,ok4
        bset    4,status
ok4     lda     l4_p
        cmp     #0
        beq     p4end
        inc     cont4
        lda     cont4
        cmp     #2
        bne     p4end
        clr     l4_p
        clr     cont4
        dec     l4
p4end   rts

****************************************************************************

test3   clr     res
        ldx     temp1
loop3   lda     $c,x
        cmp     #2
        blo     b1
        bset    0,res
b1      add     0,x
        sta     0,x
        incx
        cmp     #max
        bls     b2
        bset    0,res
b2      cpx     temp2
```

```
        bne     loop3
        rts
*************************************************************************

ascii   lda     reg2
        and     #$f0
        lsra
        lsra
        lsra
        lsra
        cmp     #9
        bls     h1
        add     #7
h1      add     #$30
        sta     tx
        jsr     send    * send the MSD

        lda     reg2
        and     #$0f
        cmp     #9
        bls     h2
        add     #7
h2      add     #$30
        sta     tx
        jsr     send
        rts
*************************************************************************

display inc     dis
        lda     dis
        cmp     #7
        bne     wait3
        jsr     disp1
        clr     dis
        inc     dis
wait3   rts
*************************************************************************
disp1   nop
rs3     brclr   2,port_c,rs3     * wait for RS232
        jsr     mess7           * cr, lf
        lda     #spc
        sta     tx
        jsr     send
        sta     tx
        jsr     send
        clrx
disp2   lda     $10,x
        sta     reg2
        jsr     ascii
        lda     #spc            * space
        sta     tx
        jsr     send
        sta     tx
        jsr     send
        incx
        cpx     #12
        bne     disp2
        brclr   1,status,m2
        jsr     noo
        bra     m3
m2      jsr     yess
m3      brclr   2,status,m4
        jsr     noo
        bra     m5
m4      jsr     yess
m5      brclr   3,status,m6
        jsr     noo
        bra     m7
m6      jsr     yess
m7      ldx     #5
        jsr     space
        lda     cont5
        cmp     #0
        beq     m8
        brset   4,status,m8
        jsr     yess
        bra     m9
m8      jsr     noo
m9      ldx     #6
        jsr     space
        lda     #2
        sta     reg2
        jsr     ascii
        rts
*****************************************************************************

send    sta     acc
        lda     #08
        sta     reg1
        bclr    3,port_c        * send a start bit
        jsr     delay2          * 80 cycles
tx3     ror     tx
        bcs     tx1
tx0     bclr    3,port_c
        bra     tx4
tx1     bset    3,port_c
tx4     jsr     delay           * 70 cycles
        dec     reg1
        bne     tx3

        inc     reg1            * dummy delay
        nop
        nop
tx5     bset    3,port_c        *send stop bits
        jsr     delay2
        jsr     delay2
        lda     acc
        rts
```

```
****************************************************************************

mess1   jsr     mess7           * cr , lf
        jsr     mess7
        clrx
more    lda     mes1,x
        cmp     #$24
        beq     done
        sta     tx
        jsr     send
        incx
        bra     more
done    jsr     mess7
        rts

yess    clrx
more2   lda     yes,x
        cmp     #$24
        beq     done2
        sta     tx
        jsr     send
        incx
        bra     more2
done2   rts

noo     clrx
more3   lda     no,x
        cmp     #$24
        beq     done3
        sta     tx
        jsr     send
        incx
        bra     more3
done3   rts

mess5   jsr     mess7           * cr, lf
        clrx
more5   lda     mes5,x
        cmp     #$24
        beq     done5
        sta     tx
        jsr     send
        incx
        bra     more5
done5   jsr     mess7           * cr, lf
        rts

mess6   jsr     mess7           * cr, lf
more6   lda     mes6,x
        cmp     #$24
        beq     done6
        sta     tx
        jsr     send
        incx
        bra     more6
done6   jsr     mess7
        rts

space   lda     #spc
space1  sta     tx
        jsr     send
        decx
        bne     space1
        rts

mess7   stx     indx            * cr & lf
        ldx     #0
more7   lda     mes7,x
        cmp     #$24
        beq     done7
        sta     tx
        jsr     send
        incx
        bra     more7
done7   ldx     indx
        rts
*************************************************************************

delay   lda     #6                * 70 cycles
loop4   nop
        deca
        bne     loop4
        nop
        rts

delay2  lda     #7               * 80 cycles
loop5   nop
        deca
        bne     loop5
        nop
        rts

delay1  bclr    7,t_cntl         * 0.1305 SEC
        lda     #$ff
        sta     timer
self    brclr   7,t_cntl,self
        rts
*************************************************************************

        org     $600
mes1    FCC     /   P1      P2       P3      Spare  /
        FCB     $0D,$0A
        FCC     / A  B  C  D  E  F  G  H  I  J  K  L /
        FCC     /  P1  P2  P3    Spare /
        FCB     $24
yes     FCC     / Y /
        FCB     $24
no      FCC     / N /
```

```
         FCB    $24
mes5     FCC    / FATAL ERROR ======> Module -2- FLASHING <====== /
         FCB    $24
mes6     FCC    / ---- Module -2- First Failure, Try Again ---- /
         FCB    $24
mes7     FCB    $0D,$0A,$24
```

```
****************************************************************************
* ***   Program : processor.asm
****    By      : Khalid Al-kofahi
****    Date    : May 25,1993
****    Description :
****            This program is executed by processor 1, 2, & 3. The program proceeds in the following
****            manner:
****            - After reset, send all of signal to the traffic lights.
****            - Traffic lights start at Nr-Eg, and continues looping in a weighted time slices fashion.
****            - Before each voting process ( changing the traffic lights ), all processors are
****             synchronized ( call procedure synch).
****              While wating for the current time slice to elapse, synchronize the processors every 0.5
****            seconds.
****              Refer to figure 3.7 and 3.8 for further  information about this program.
****              The lines being voted upon are :
****                    - PA0 -- serial data out : 6 - bits
****                    - PB0 -- serial clock.
****                    -  PB1 -- parallel clock (load).
****************************************************************************


        org     $784
        fcb     $07

        org     $7fe            * start here after reset
        fcb     $01
        fcb     $00
        org     $7f8            * timer interrupt service routine
        fcb     $02
        fcb     $50
        end
port_a  equ     $000
port_b  equ     $001
port_c  equ     $002
timer   equ     $008
a_cntl  equ     $004
b_cntl  equ     $005
c_cntl  equ     $006
t_cntl  equ     $009

t_mode  equ     $47
a_mode  equ     $ff
b_mode  equ     $03
c_mode  equ     $01

Nr_Eg   equ     $0C             * North red - East green
Nr_Ey   equ     $0A             * North red - East yellow
Nr_Er   equ     $09             * North red - East red
Ng_Er   equ     $21             * North green - East red
Ny_Er   equ     $11             * North yellow - east red

status  equ     $12
count   equ     $13
count1  equ     $14
```

```
count2  equu    $15


        org     $100
        sei
        lda     #a_mode
        sta     a_cntl
        lda     #b_mode
        sta     b_cntl
        lda     #c_mode
        sta     c_cntl
        lda     #t_mode
        sta     t_cntl

        clr     port_c

        jsr     synch
        lda     #0
        sta     port_a
        jsr     shift
        jsr     delay

main    jsr     synch           * synchronize with other processors
        lda     #Nr_Er
        sta     port_a          * traffic lights data is stored at port_a
        jsr     shift           * send serial data
        lda     #2
        sta     count           * the length of time slice = 2 * del_sn
        jsr     del_sn          * delay as sepecified by count

        jsr     synch           * synch again
        lda     #Nr_Eg          * next light
        sta     port_a
        jsr     shift
        lda     #4
        sta     count           * time slice = 4 * del_sn
        jsr     del_sn

        jsr     synch
        lda     #Nr_Ey
        sta     port_a
        jsr     shift
        lda     #2
        sta     count
        jsr     del_sn

        jsr     synch
        lda     #Nr_Er
        sta     port_a
        jsr     shift
        lda     #2
        sta     count
        jsr     del_sn
        jsr     synch
        lda     #Ng_Er
        sta     port_a
        jsr     shift
        lda     #4
        sta     count
        jsr     del_sn

        jsr     synch
        lda     #Ny_Er
        sta     port_a
        jsr     shift
        lda     #2
        sta     count
        jsr     del_sn
        bra     main            * continue looping

**********************************************************************
***     procedure del_sn : delay as specified by count. and keep the processors
***                             synchronized.
**********************************************************************

del_sn  jsr     synch
        jsr     delay           * 512800 cycles
        jsr     delay
        dec     count
        bne     del_sn
        rts

******************************************************************
***     procedure Synch:  This procedure synchronize the processors . for a flowchart see figure 3.8
******************************************************************

synch   brset   2,port_b,flash          * I am out goto flashing.

        bclr    7,t_cntl
        lda     #$ff                    * maximum waiting period.
        sta     timer

        bclr    6,t_cntl
        cli

        brset   3,port_b,t2             * porc at PC1 is out
        brset   4,port_b,t3             * proc at PC2 is out

t1      bset    0,port_c                * send I am ready signal .
wait    lda     port_c
        and     #$ff
        cmp     #$ff                    * PC- 3,2,1,0   = 1111
        beq     done
        brset   0,status,inter          * a processor(s) is not ready, time out is reached.
        bra     wait

inter   brclr   1,status,t2             * the processor at PC1 is too late. don't wait any longer
```

```
        brclr   2,status,t3         * the processor at PC2 is too late. don't wait any longer
        bra     done

t2      bset    0,port_c            * if all others ( except the one at PC1) are ready, then done
wait2   lda     port_c
        and     #$fd
        cmp     #$fd                * PC- 3,2,1,0     = 11X1
        beq     done
        bra     wait2


t3      bset    0,port_c            * if all others (except the one at PC2)  are ready, then done
wait3   lda     port_c
        and     #$fb
        cmp     #$fb                * PC- 3,2,1,0     = 1X11
        beq     done
        bra     wait3

done    sei
        bset    6,t_cntl
        jsr     delay3              * in case others did not read me yet
        bclr    0,port_c            * clear I am ready signal
        rts

****************************************************************************
*** procedure shift : shifts the data and serial clock serially, and then sends the load signal
****************************************************************************

shift   ldx     #6

send  bset      0,port_b            * serial clock of width 200 cycle
        jsr     delay2              * 100 cycle
        jsr     delay2
        bclr    0,port_b

        jsr     delay2
        jsr     delay2
        lsr     port_a              * send data ( six times)
        decx
        bne     send

        bset    1,port_b            * parallel clock of width 400 cycle at PB1
        jsr     delay2
        jsr     delay2
        jsr     delay2
        jsr     delay2
        jsr     delay2
        bclr    1,port_b
        rts


****************************************************************************
*** procedure flash : if  I am faulty or more than two faulty processors then flash all the time
```

```
*****************************************************************************

flash   lda     #0                      * all off
        sta     port_a
        jsr     shift
        jsr     delay

        lda     #Nr_Er                  * Nr-Eg
        sta     port_a
        jsr     shift
        jsr     delay
        bra     flash                   * stay here

*****************************************************************************

delay   lda     #100                    * delay1 = 512800 cycles
        sta     count1
loop1   lda     #255
        sta     count2
loop2   dec     count2
        bne     loop2
        dec     count1
        bne     loop1
        rts

delay2  lda     #10                     * delay of 108 cycle
loop4   nop
        deca
        bne     loop4
        rts

delay3  lda     #5                      * 58 cycles
loop5   nop
        deca
        bne     loop5
        rts

*****************************************************************************
***     timer int. service routine: If I am here then maximum waiting time is reached without
        recieving all the " I am ready " signals, mark which processor is late ( status) and return.
*****************************************************************************

        org     $250
        bset    5,port_b
        bclr    7,t_cntl
        ldx     port_c          * which one is late ?
        lda     port_b
        and     #$1c            * 0001 1100  , s1,s2,s3 --> PB2,3,4
        cmp     #0
        bhi     flash           * more than two faulty processors ? , if yes goto flash.
        txa
        and     #$0e
        sta     status
        bset    0,status
```

```
bclr    5,port_b
rti
end
```

```
*****************************************************************************
****    Program: Spare.asm
****    By        : Khalid Al-kofahi
****    Date      : May 25, 1993
****    Description :
****              This program is executed by the spare processors in both modules. It is the same as the
****              program processor.asm ( executed by processors 1, 2, & 3) except in the procedure
****              synch. For a flow chart of this procedure refer to figure 3.9 and refer to figure 3.7 for the
****              hardware connections.
*****************************************************************************

          org     $784
          fcb     $07

          org     $7fe
          fcb     $01
          fcb     $00
          org     $7f8              * timer interrupt routine
          fcb     $02
          fcb     $50
          end
port_a    equ     $000
port_b    equ     $001
port_c    equ     $002
timer     equ     $008
a_cntl    equ     $004
b_cntl    equ     $005
c_cntl    equ     $006
t_cntl    equ     $009

t_mode    equ     $47
a_mode    equ     $ff
b_mode    equ     $03
c_mode    equ     $01

Nr_Eg     equ     $0C
Nr_Ey     equ     $0A
Nr_Er     equ     $09
Ng_Er     equ     $21
Ny_Er     equ     $11

status    equ     $12
count     equ     $13
count1    equ     $14
count2    equ     $15


          org     $100
          sei
          lda     #a_mode
          sta     a_cntl
          lda     #b_mode
          sta     b_cntl
          lda     #c_mode
```

```
        sta     c_cntl
        lda     #t_mode
        sta     t_cntl

        clr     port_c

        jsr     synch
        lda     #0
        sta     port_a
        jsr     shift
        jsr     delay

main    jsr     synch
        lda     #Nr_Er
        sta     port_a
        jsr     shift
        lda     #2
        sta     count
        jsr     del_sn

        jsr     synch
        lda     #Nr_Eg
        sta     port_a
        jsr     shift
        lda     #4
        sta     count
        jsr     del_sn

        jsr     synch
        lda     #Nr_Ey
        sta     port_a
        jsr     shift
        lda     #2
        sta     count
        jsr     del_sn

        jsr     synch
        lda     #Nr_Er
        sta     port_a
        jsr     shift
        lda     #2
        sta     count
        jsr     del_sn

        jsr     synch
        lda     #Ng_Er
        sta     port_a
        jsr     shift
        lda     #4
        sta     count
        jsr     del_sn

        jsr     synch
        lda     #Ny_Er
        sta     port_a
        jsr     shift
        lda     #2
        sta     count
        jsr     del_sn
        bra     main
******************************************************************************
***     procedure del_sn  *******************************************************

del_sn  jsr     synch
        jsr     delay
        jsr     delay
        dec     count
        bne     del_sn
        rts

************************************************************************  *****
***     procedure Synchronize ***************************************************

synch   bclr    7,t_cntl
        lda     #$ff
        sta     timer

        bclr    6,t_cntl
        cli

        brset   2,port_b,t2                 * proc 1 is faulty
        brset   3,port_b,t3                 * proc 2 faulty
        brset   4,port_b,t4                 * proc 3 is faulty

t1      bset    0,port_c                    * I am ready.
wait    lda     port_c
        and     #$ff
        cmp     #$ff                        * PC0, PC1, PC2,PC3 = 1111
        beq     done
        brset   0,status,inter
        bra     wait

inter   brclr   1,status,t2                 * processor 1 is too late (time out)
        brclr   2,status,t3                 * processor 2 is too late ( time out)
        brclr   3,status,t4                 * processor 3 is too late ( time out)
        bra     done

t2      bset    0,port_c                    * I am ready
wait2   lda     port_c
        and     #$fd                        * PC- 3,2,1,0     = 11X1
        cmp     #$fd
        beq     done
        bra     wait2

t3      bset    0,port_c                    * I am ready
wait3   lda     port_c
        and     #$fb                        * PC- 3,2,1,0     = 1X11
        cmp     #$fb
        beq     done
        bra     wait3

t4      bset    0,port_c                * I am ready
wait4   lda     port_c
        and     #$f7                    * PC- 3,2,1,0     = X111
        cmp     #$f7
        beq     done
        bra     wait4

done    sei
        bset    6,t_cntl
        jsr     delay3                  * in case others did not read me yet
        bclr    0,port_c
        rts
********************************************************************************
*** procedure shift ***************************************************************

shift   ldx     #6

send   bset     0,port_b                * serial clock of width 100 cycle
        jsr     delay2
        jsr     delay2
        bclr    0,port_b

        jsr     delay2
        jsr     delay2
        lsr     port_a                  * send data
        decx
        bne     send

        bset    1,port_b                * parallel clock of width 400 cycle
        jsr     delay2
        jsr     delay2
        jsr     delay2
        jsr     delay2
        jsr     delay2
        bclr    1,port_b
        rts

*****************************************************************************

delay   lda     #100              * delay1 = 512800 cycles
        sta     count1
loop1   lda     #255
        sta     count2
loop2   dec     count2
        bne     loop2
        dec     count1
        bne     loop1
        rts

delay2  lda     #10     * delay of 108 cycle
loop4   nop
```

```
        deca
        bne     loop4
        rts

delay3  lda     #5
loop5   nop
        deca
        bne     loop5
        rts

*************************************************************************
flash   lda     #0
        sta     port_a
        jsr     shift
        jsr     delay

        lda     #Nr_Er
        sta     port_a
        jsr     shift
        jsr     delay
        bra     flash
        end

************************************************************************
*****   timer int. service routine *********************************************

        org     $250
        bset    5,port_b
        bclr    7,t_cntl
        ldx     port_c
        lda     port_b
        and     #$1c
        cmp     #0
        bhi     flash
        txa
        and     #$0e
        sta     status
        bset    0,status
        bclr    5,port_b
        rti
        end
```

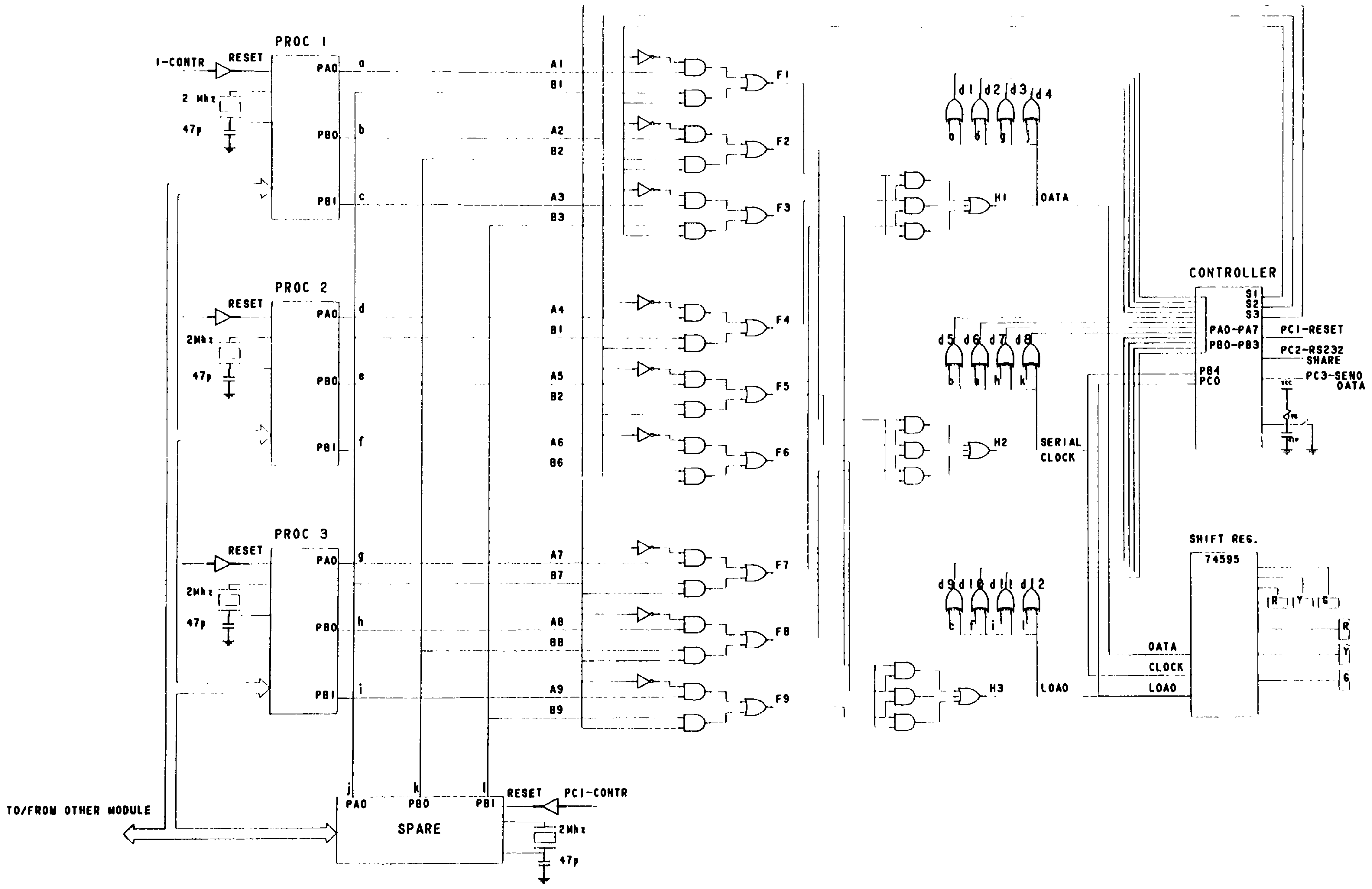


Figure 3.3 A MODULE

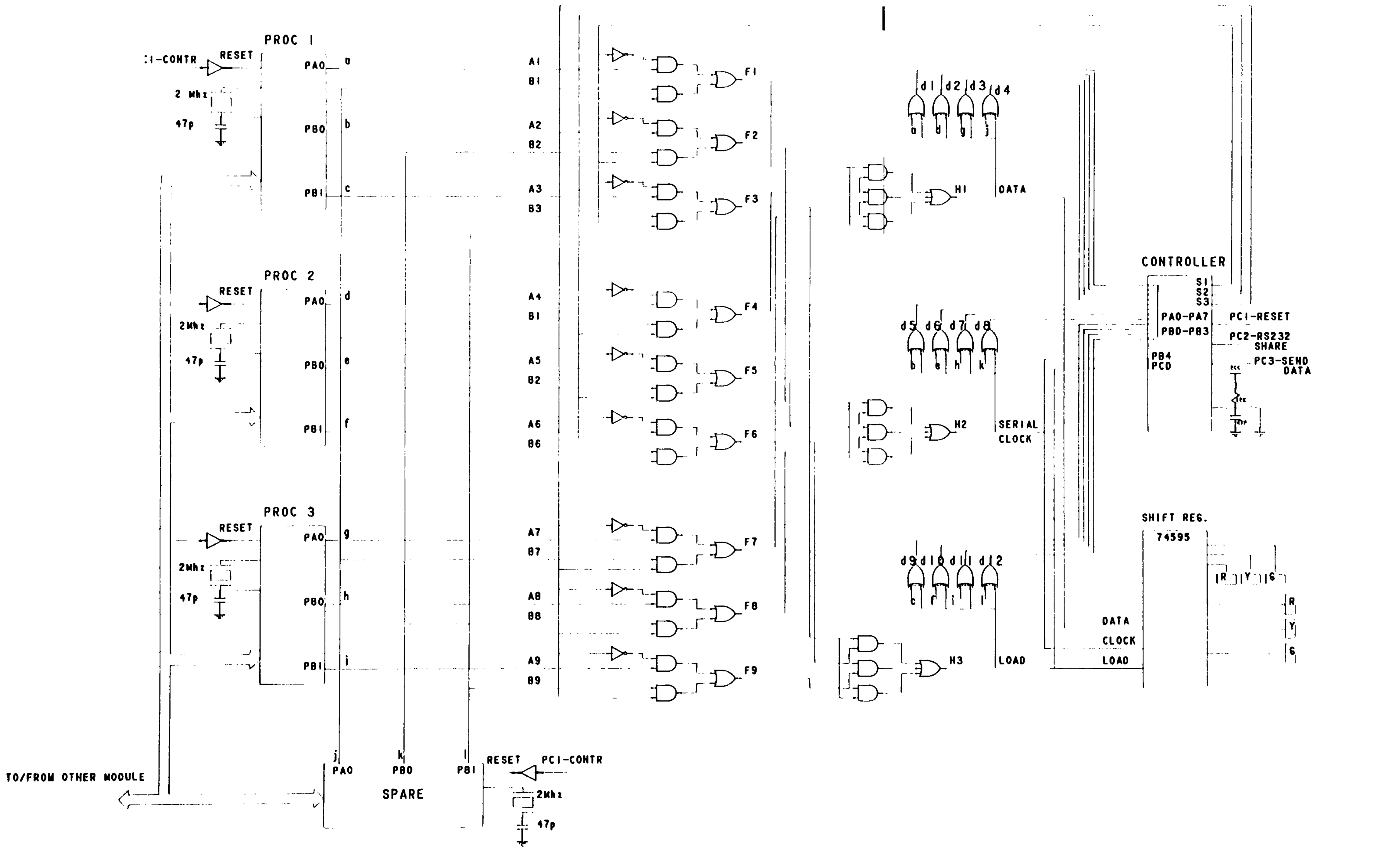


Figure 3.3 A MODULE

MOTOROLA

MC68705P3

Advance Information

8-BIT EPROM MICROCOMPUTER UNIT

The MC68705P3 Microcomputer Unit (MCU) is an EPROM member of the M6805 Family of low-cost single-chip microcomputers. The user programmable EPROM allows program changes and lower volume applications in comparison to the factory mask programmable versions. The EPROM versions also reduce the development costs and turnaround time for prototype evaluation of the mask ROM versions. This 8-bit microcomputer contains a CPU, on-chip CLOCK, EPROM, bootstrap ROM, RAM, I/O, and a TIMER.

Because of these features, the MC68705P3 offers the user an economical means of designing an M6805 Family MCU into his system, either as a prototype evaluation, as a low-volume production run, or a pilot production run.

HARDWARE FEATURES:

- 8-Bit Architecture
- 112 bytes of RAM
- Memory Mapped I/O
- 1804 Bytes of User EPROM
- Internal 8-Bit Timer with 7-Bit Prescaler
  - Programmable Prescaler
  - Programmable Timer Input Modes
  - External Timer Interrupt
- Vectored Interrupts — External, Timer, and Software
- Zero-Cross Detection on $\overline{\text{INT}}$ Input
- 20 TTL/CMOS Compatible Bidirectional I/O Lines (8 Lines are LED Compatible)
- On-Chip Generator
- Master and Power-On Reset
- Complete Deyelopment System Support on EXORciser
- Emulates the MC6805P2 and MC6805P4 (Except for V_{SB})
- Bootstrap Program in ROM Simplifies EPROM Programming

SOFTWARE FEATURES:

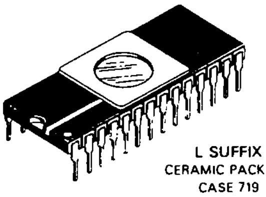
- Similar to M6800 Family
- Byte Efficient Instruction Set
- Easy to Program
- True Bit Manipulation
- Bit Test and Branch Instructions
- Versatile Interrupt Handling
- Versatile Index Register
- Powerful Indexed Addressing for Tables
- Full Set of Conditional Branches
- Memory Usable as Registers/Flags
- Single Instruction Memory Examine/Change
- 10 Powerful Addressing Modes
- All Addressing Modes Apply to EPROM, RAM, and I/O

hs document contains information on a new product Specifications and information herein
re subject to change without notice

HMOS

(HIGH-DENSITY, N-CHANNEL
DEPLETION LOAD,
5 V EPROM PROCESS)

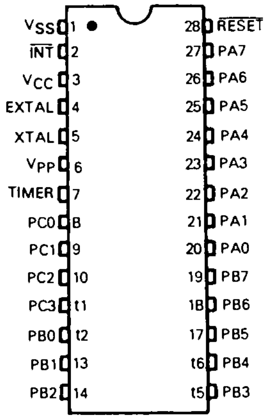
8-BIT EPROM MICROCOMPUTER



L SUFFIX
CERAMIC PACKAGE
CASE 719

S SUFFIX
CERDIP PACKAGE
ALSO AVAILABLE

PIN ASSIGNMENT



GENERIC INFORMATION
(f = 1.0 MHz, T_A = 0 to 70°C)

Package Type	Generic Number
Ceramic L Suffix	MC68705P3L
Cerdip S Suffix	MC68705P3S