A Single Fault-Tolerant Dual Channel Controller

Thomas J. Lowery
University of Central Florida
A SINGLE FAULT-TOLERANT DUAL CHANNEL CONTROLLER

BY

THOMAS J. LOWERY
B.S., Rochester Institute of Technology, 1980

RESEARCH REPORT

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering in the Graduate Studies Program of the College of Engineering, University of Central Florida, Orlando, Florida

Spring Term
1984
ABSTRACT

The advent of the VLSI technology makes it feasible to offer a once expensive system attribute called fault-tolerance to a wide variety of applications. This can be accomplished by using off-the-shelf single board computers and peripherals as the heart of the system. Custom design boards can then be added to meet the specific requirements of each application.
ACKNOWLEDGEMENT

This paper was made possible by continued support from Mr. Warren M. Anderson and greatly aided by his depth of knowledge in all areas of design. I would also like to thank Dr. Ayee Goundan for his support throughout the project. Finally, I would like to thank Dr. Fred O. Simons, Jr. for his assistance in completing this paper.
# TABLE OF CONTENTS

INTRODUCTION ............................................. 1

Chapter

I. ARCHITECTURE SELECTION ................................... 3
II. THE CHANNEL STATES ......................................... 6
III. THE CONTROL FRAME OR LOOP ............................. 9
IV. CHANNEL HARDWARE DEFINITION ........................... 11
V. SELECTION OF INTER-CHANNEL COMMUNICATIONS LINK ...... 14
VI. RESULTS AND CONCLUSIONS ............................... 24

REFERENCES ................................................. 26
APPENDIX .................................................. 27
INTRODUCTION

Fault-tolerance refers to the ability of a system to survive in the presence of one or more faults. In this paper a fault is defined to be any hardware malfunction or failure which disables the system to perform to the specified limits.

Fault-tolerance can be exploited to make a system more reliable, available and safe. Fault-avoidance techniques can also be used to achieve the same goals; but only fault-tolerance can provide continued system operation in the presence of a fault. Fault-avoidance techniques avoid, or more appropriately postpone, the occurrence of faults by using high reliability parts, less thermal stress, good mechanical contact, etc. Also with today's technology, fault-tolerance can be less expensive than fault-avoidance for similar requirements. Quick fault-repair techniques (which use low Mean-Time-To-Repair (MTTR), skilled personnel, etc.) are also no longer effective for similar reasons.

Fault-tolerance implies detection, isolation and repair of a fault without noticeable interruption to the system operation. Repair here simply refers to switching in a (redundant) good unit and switching out the failed unit. Fault-tolerance requires some form of hardware and/or software redundancy. A system component is called redundant if it is entirely superfluous for the operation of the fault-free system. The level of redundancy directly influences
the degree of fault-tolerance achievable. The redundancy level is selected based on the application requirements.

In any redundant system, provisions must be made for managing or operating redundant elements. Redundancy management refers to the built-in functions of a redundant system which performs (1) fault detection and isolation, and (2) recovery. Recovery refers to reconfiguration and initialization of the remaining good components for continued operation. If continued operation is not possible, a safe shutdown is desirable.

This report describes the development of a duplex fault-tolerant controller. The duplex architecture involves two channels both executing the same software in loose synchronism with each other. An arbiter determines which of the two channels will control the plant. A system based on this concept can survive the loss of one channel and still maintain operation.
CHAPTER I

ARCHITECTURE SELECTION

One of the main objectives of this R&D project was to develop a baseline architecture that will be applicable to many of our applications. Considering that this was the department's first attempt to design a fault-tolerant system, the interest was in selecting an architecture that can be upgraded for higher performance and reliability characteristics. Consequently, the dual channel architecture was considered as the prime candidate. The input, processing and output elements of a simplex or non-redundant controller is referred to as a channel. Then, roughly stated, a duplex or dual redundant controller has two channels; a triplex controller has three channels, and so on. Duplex redundancy is attractive because of the following reasons:

a. If duplex redundancy is sufficient for designing a control system, there would be obvious savings in hardware, power and therefore costs, compared to a higher level of redundancy.

b. Study of duplex architecture aids in building better performing triplex or higher redundant systems. For example, failure of one channel reduces a triplex system to a duplex. Methods must be developed to handle this situation.

Based on the above considerations, a prototype duplex baseline architecture was developed using Intel's Multibus compatible boards. Off-the-shelf hardware was used in the prototype design wherever
possible so that minimum effort was spent in custom hardware design. It is possible to upgrade the baseline architecture to a triplex configuration or to reduce it to a simplex configuration. The prototype is serving as a test vehicle for planning and testing the redundancy management operating system and the diagnostic software. The prototype is also being used as a tool for investigating new fault tolerant architectural concepts.

The block diagram of a redundant dual channel configuration is shown in Figure 1. There are basically three functional units in the duplex configuration: Channel-0, Channel-1, and the arbiter. The channels are identical to one another, and any one of them is fully capable of controlling the system. The arbiter serves as the judge for deciding which channel should be allowed to control the system. In the configuration shown in Figure 1, the channels are fully redundant, except for the non-critical input output logic which is shared in order to minimize hardware. It should be noted that several variations to the configuration shown are possible. For instance, the channels can be fully redundant (as opposed to sharing non-critical functions). The current prototype has been implemented with fully redundant dual channels.
Figure 1. Block Diagram of the Duplex Architecture
CHAPTER II

THE CHANNEL STATES

During the course of system operation, the channels pass through what are called states. The STATE of a channel refers to the status of the channel at a given instant in time. The transition from one state to another is caused by an EVENT. For instance, the event "fault" causes the channel to change from "good" state to "faulty" state. In reality, each of the channels can assume one of numerous states. For simplicity, each channel is assumed to be in any of the following three states:

ON-LINE: The channel is fault-free and receives all inputs and performs processing. It can control all critical and non-critical outputs of the system. The on-line channel is also referred to as the primary channel.

STANDBY: The channel is fault-free and receives the other channel inputs through the shared memory access and performs processing. However, it uses its own inputs for fault detection by comparison. The standby channel is also called the secondary channel.

OFF-LINE: The channel is declared to contain a malfunction. It can control none of the critical or non-critical outputs. The off-line channel is also referred to as the faulty channel.
An on-line channel goes to the off-line state when the self-test functions detect a fault in that channel. This will cause the other channel which is in the standby state to move to the on-line state. When the malfunction is repaired, the channel is allowed to assume any of the other two states. The transitions between the channel states is illustrated in Figure 2. Note that an off-line (or faulty) channel can be manually forced on-line. This will make the other channel relinquish the on-line state independent of its health. In other words, manual selection takes precedence over any automatic channel state assignment.

Both channels cannot be on-line at the same time, and both channels cannot be in standby at the same time; therefore these are mutually exclusive states. Both channels in the off-line state at the same time amounts to a system failure.
ON-LINE: CONTROL ALL CRITICAL AND NON-CRITICAL OUTPUTS
STANDBY: CONTROL NO OUTPUTS, BUT PERFORM ALL CALCULATIONS IN PARALLEL WITH ON-LINE CHANNEL
OFF-LINE: CONTROL NONE. (FAULTY STATE)

Figure 2, Illustration of Channel States
CHAPTER III

THE CONTROL FRAME OR LOOP

Each channel has self-contained hardware and software to fully perform its functions. The software consists of three parts: the supervisory programs, self-test programs, and application programs. The supervisory program or executive manages the execution of application and self-test software. In a typical control system, the application and self-test program are executed periodically, where the period depends on the requirements of the particular application. The repetition cycle is referred to as the control frame or control loop. The execution of the control frame is often initiated by an external or internal time reference called the sync interrupt. This concept is illustrated in Figure 3. Several variations to the control loop shown here are possible, however.

As noted in Figure 3, the comparison of both channel inputs and calculated outputs serves as a key fault-detection scheme. In addition to reducing the burden of dedicated self-test routines, this scheme detects the fault at the earliest time, thereby inhibiting the propagation of the fault to the output.

It should be noted that there is a significant overhead for performing on-line self-tests and parameter comparisons. This overhead should be kept within tolerable limits in order to provide maximum availability of the system to application programs.
CONTINUOUSLY MONITOR POWER SUPPLIES & CLOCKS

POWER ON

CHECK HARDWARE USING FAULT DETECTION TESTS

ISOLATE FAULT, RECONFIGURE & REPORT

SYNC INTERRUPT?

NO

INPUT

COMPARE INPUTS WITH INPUTS OF OTHER CHANNEL

MISMATCH?

YES

PROCESS

COMPARE OUTPUTS WITH OUTPUTS OF OTHER CHANNEL

MISMATCH?

YES

OUTPUT

Figure 3. Typical Control Loop Flow with Self-Test
CHAPTER IV

CHANNEL HARDWARE DEFINITION

Each channel has a Multibus structure as shown in Figure 4. The Multibus interface is defined as a flexible structure used to interface a standard family of Multibus compatible boards. It supports direct address ability up to one megabyte through 20-bit address and 8-bit to 16-bit data transfers. The bus structure is built upon the master-slave concept where the master device in the system takes control of the Multibus interface and the slave device, upon decoding its address, acts upon the command provided by the master. Handshaking between master and slave allows modules of different speeds to use the Multibus interface. It also has the ability to connect multiple master modules for multiprocessor configurations. The Multibus interface provides control signals for connecting multiple masters either in a daisy chain or parallel priority scheme.

The initial Multibus configuration has one bus master, an 8086/8087 16-bit single board computer, fully Multibus compatible with all existing 8-bit or 16-bit peripheral boards. The 8086/8087 co-processors have access to an on-board local bus as well as the Multibus. The local bus architecture includes a minimum of 4K bytes of RAM, 16K bytes of EPROM, a programmable asynchronous-synchronous...
serial communication channel, a 24-bit programmable parallel I/O port, 3 programmable 16-bit interval timers/event counters, and eight levels of programmable interrupt control. It also has Multibus control logic for multimaster configuration and system expansion.

Each channel has a Multibus interface memory capability of 128K bytes of RAM, 64K bytes of EPROM, and 8K bytes of dual port RAM. The dual port RAM is accessible from the local channel and from the redundant channel and serves as the communication link between the two channels. The reason for choosing the dual port RAM as the communication link is discussed in the next chapter. Each channel has 144 programmable discrete I/O lines and four serial communications I/O ports. The I/O ports include interval timers and programmable interrupt controllers. There are 32 input lines and six analog output lines. An analog comparator monitors power supply bus voltages for out-of-tolerance conditions. Each channel will latch in a fault condition caused by the power supplies associated with the channel. The latched fault(s) are sent to the Arbiter. A fault condition is reset by the channel manual reset, system manual reset or the power up reset.
Figure 4. Channel Electronics Block Diagram
CHAPTER V

SELECTION OF THE INTER-CHANNEL COMMUNICATIONS LINK

The selection of the type of communication link to be used in the fault-tolerant design was of prime importance. The communication link would be used by each channel to arrive at a health opinion about the other channel. The four types of designs discussed were (1) Serial Link, (2) Parallel Link, (3) FIFO Memory Link, (4) Dual Port Memory Link. These four alternatives were ranked against nine different criteria and the results are shown in Table 1.

As can be seen from Table 1, the dual port and FIFO link were given a higher ranking for speed than were the serial or parallel link. This is because in the dual port and FIFO configuration, the processors need not interrupt each other to exchange information. The dual port memory was also ranked the highest in the categories of flexibility, ease of software implementation, and CPU overhead. The dual port link will allow a warm-start of the off-line processor. If a warm-start feature were needed, the off-line processor would have to acquire current data from the failed processor. The flexibility of the dual port link allows this feature to be implemented. It should also be pointed out that the dual port link is much more flexible from the standpoint of redesign. That is to say, it is easier to
change from a dual port link to one of the other three configurations than vice versa. This is because the dual port link is more complex in terms of interconnections and hardware. The dual port link ranks higher in terms of overhead because the CPU takes much less time to access it than the other three links. In terms of software implementation, it is easier to access memory than to interface with a serial or parallel link, thus the dual port and FIFO rank the highest.

The serial link ranks the highest in terms of component count, interface connections, channel isolation, ease of hardware implementation, and growth. The serial link requires the fewest number of components to implement because it is not necessary to select a particular area of memory. The interface connections between the two processors is simpler because there are fewer communications lines between the processors allowing for better channel isolation and reliability. Because of the fewer components and fewer interface connections, the hardware implementation of the serial link is much easier.

The serial link is ranked highest in terms of growth. Growth is defined as the ability to expand from a duplex system to a triplex or n-plex system. Because of the fewer interface connections and ease of hardware implementation, the serial link has the greatest potential for growth.

Table 1 also shows that the sum of the rankings (∑R_i) for each of the alternatives is quite close. In order to make a proper
### Table 1

**Ranking of Criteria (R_i)**

<table>
<thead>
<tr>
<th>Selection Criteria</th>
<th>Serial Link</th>
<th>Parallel Link</th>
<th>FIFO Memory Link</th>
<th>Dual-Port Memory Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>.25</td>
<td>.50</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Flexibility</td>
<td>.50</td>
<td>.50</td>
<td>.25</td>
<td>1.00</td>
</tr>
<tr>
<td>Ease of SW Implementation</td>
<td>.25</td>
<td>.25</td>
<td>.75</td>
<td>1.00</td>
</tr>
<tr>
<td>Ease of HW Implementation</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
<td>.50</td>
</tr>
<tr>
<td>Overhead</td>
<td>.25</td>
<td>.50</td>
<td>.75</td>
<td>1.00</td>
</tr>
<tr>
<td>Channel Isolation</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
<td>.75/.50</td>
</tr>
<tr>
<td>Component Count</td>
<td>1.00</td>
<td>.75</td>
<td>.75</td>
<td>.25</td>
</tr>
<tr>
<td>Interface Connection</td>
<td>1.00</td>
<td>.75</td>
<td>.75</td>
<td>.50/.25</td>
</tr>
<tr>
<td>Growth</td>
<td>1.00</td>
<td>.50</td>
<td>.75</td>
<td>.25</td>
</tr>
<tr>
<td>Σ R_i</td>
<td>6.25</td>
<td>5.75</td>
<td>6.50</td>
<td>6.25/6.00</td>
</tr>
</tbody>
</table>

### Table 2

**Weight Factor of Criteria (W_i)**

<table>
<thead>
<tr>
<th>Weights</th>
<th>Sum of Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7/49 = .143</td>
</tr>
<tr>
<td>9</td>
<td>9/49 = .184</td>
</tr>
<tr>
<td>5</td>
<td>5/49 = .102</td>
</tr>
<tr>
<td>5</td>
<td>5/49 = .102</td>
</tr>
<tr>
<td>8</td>
<td>8/49 = .163</td>
</tr>
<tr>
<td>6</td>
<td>6/49 = .122</td>
</tr>
<tr>
<td>4</td>
<td>4/49 = .082</td>
</tr>
<tr>
<td>3</td>
<td>3/49 = .061</td>
</tr>
<tr>
<td>2</td>
<td>2/49 = .041</td>
</tr>
</tbody>
</table>

<p>| 49 | 49 = .101 |</p>
<table>
<thead>
<tr>
<th>Selection Criteria</th>
<th>Serial Link</th>
<th>Parallel Link</th>
<th>FIFO Memory Link</th>
<th>Dual-Port Memory Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>.036</td>
<td>.072</td>
<td>.143</td>
<td>.143</td>
</tr>
<tr>
<td>Flexibility</td>
<td>.092</td>
<td>.092</td>
<td>.046</td>
<td>.184</td>
</tr>
<tr>
<td>Ease of SW Implementation</td>
<td>.026</td>
<td>.026</td>
<td>.077</td>
<td>.102</td>
</tr>
<tr>
<td>Ease of HW Implementation</td>
<td>.102</td>
<td>.102</td>
<td>.077</td>
<td>.051</td>
</tr>
<tr>
<td>Overhead</td>
<td>.041</td>
<td>.082</td>
<td>.122</td>
<td>.163</td>
</tr>
<tr>
<td>Channel Isolation</td>
<td>.122</td>
<td>.122</td>
<td>.092</td>
<td>.092/.061</td>
</tr>
<tr>
<td>Component Count</td>
<td>.082</td>
<td>.062</td>
<td>.062</td>
<td>.021</td>
</tr>
<tr>
<td>Interface Connections</td>
<td>.061</td>
<td>.046</td>
<td>.046</td>
<td>.031/.015</td>
</tr>
<tr>
<td>Growth</td>
<td>.041</td>
<td>.021</td>
<td>.031</td>
<td>.010</td>
</tr>
<tr>
<td>( \Sigma R_i W_i )</td>
<td>( .603 )</td>
<td>( .625 )</td>
<td>( .696 )</td>
<td>( .797/.750 )</td>
</tr>
</tbody>
</table>
selection, it was necessary to assign a weight factor \((W_i)\) to each of the criterion. It was felt that because initially this project would concern itself with the research and development of a duplex fault-tolerant system, the heaviest weights should be assigned to flexibility, speed, and overhead. When the project reaches the point where a triplex or n-plex architecture is needed, new weight factors \((W_i)\) will have to be assigned to the selection matrix. The weight factors \((W_i)\) are shown in Table 2.

The results of the selection matrix \((R_i W_i)\) are shown in Table 3. The results indicate that the dual port memory link should be used during the initial phase of the project.

An investigation into the types of dual port memory sharing techniques led to four possible configurations. These four configurations are described below.

a. Prioritized Access

This technique multiplexes the two devices' address, control and data buses by assigning priorities. When the high priority device accesses the shared memory, it always gets control of the multiplexer. Meanwhile, the low priority device idles until the high priority device releases memory control.

b. Arbitrated Access

This technique utilizes arbitration logic to determine which of two devices needs memory service first. This logic provides ready/wait signals to the devices to avoid conflicts between access requests. But because some processors and peripherals cannot enter a wait state, the arbitrated access method does not suit many memory-sharing
applications. (In an exception, though, clever design of an arbitrated dual port the Intel ISBC 86/12 permits two CPU's to access one memory with a maximum of one wait state.)

c. Synchronous Access

This technique enables two CPU's to generate concurrent requests for shared memory in a complicated timing arrangement. This approach is device-dependent, however and only a few µP's such as the 6800 and 6502 exhibit suitable timing parameters. These µP's furnish fixed periodic intervals during which memory access cannot take place, thus, permitting interleaved memory sharing.

d. Asynchronous Sampling Method.

In this technique the two devices do not compete for access to the shared RAM array. Instead, under precise clock timing, input multiplexers and input and output latches alternately allocate memory access to one of the two ports for data, address and control transactions.

The Prioritized Access Method is unsuitable for the FTA design because the high priority device always gets control of the multiplexer when it requests the memory. This could result in the low priority device being interrupted in the middle of a read cycle.

The Synchronous Access technique cannot be used because the 8086 microprocessor does not exhibit the type of timing parameters needed for this method.

The Arbitrated Method for dual port access has been implemented on Intel's 86/12 board. This method allows the RAM to be shared by the host and an alien CPU. When accessing the dual port RAM, the
host CPU has priority. In this situation, the alien access is held off until the host CPU has completed its particular read or write operation. The alien CPU will enter a wait state until it receives a transfer acknowledge signal from the dual port RAM. It will then gain control of the dual port, and access by the host CPU is held off until completion of the READ operation. The host processor will not gain control of the dual port until a transfer acknowledge signal is generated or the alien no longer generates a dual port address request. The maximum number of wait states either processor should incur is one. If the transfer acknowledge signal failed to be generated, both CPU's could be forced into a permanent wait state.

A number of other signals are also passed between the arbitration logic and the host and alien CPU's. A failure in one CPU could cause a bogus signal to hang up the other CPU. For this reason it was felt that the arbitrated dual port is unsuitable for the FTA design.

The Asynchronous Access technique has a simple hardware approach, but relatively intricate timing parameters. In order to determine if this technique could be used in the FTA design, a timing analysis was done. This analysis and the system configuration used for the analysis are shown in the remainder of this chapter.

The analysis shows that if the dual port is accessed from the local bus, the dual port board can use the input muxes as buffers. This will decrease the propagation delays and allow you to use a slower memory. The analysis shows that a memory with a 70 nsec access time will be adequate.
The major advantage of the asynchronous method is that it operates independently of the attached devices and grants virtually instant memory access. A failure in one CPU will have no affect on the dual port or the other CPU. This alone would indicate that of the four type of dual port methods available, only the asynchronous method is suitable for a fault tolerant design.

Asynchronous Dual-Port Timing Analysis

As can be seen from Figure 5, for a memory READ cycle in which the clock's sampling edge just misses the start of Device-A's access cycle ($\Delta t$ exceeds 0), the next Port-A cycle must complete before the end of Device-A's access cycle (TAC). Consequently, as $\Delta t$ approaches 0, $3T_{\text{sample}}$ approaches TAC. You thus need at least two complete Port-A cycles for the shortest possible Device-A access cycle. Because Port-B cycle occurs between two Port-A cycles, however, $T_{\text{sample}}$ is approximately 1/3 of TAC.
Given the system block diagram shown in Figure 6, the memory access time (TAM) can be computed using the following equations.

\[
TAM = \frac{TAC}{3} - T_{PD \text{ mux}} - T_{PD \text{ input}} - T_{PD \text{ output}}
\]

Where \( T_{sample} = \frac{TAC}{3} \)

And \( T_{PD} = \) time of propagation delay

\( T_{AC} = T \) address to valid data delay - \( T_{PD8283} - T_{PDLS645} - T_{PD8205} + \) Logic

\( T_{AC} = 410 \) nsec - 30 nsec - 15 nsec - 30 nsec

\( T_{AC} = 335 \) nsec \( T_{sample} = \frac{335}{3} = 111.67 \) nsec

\[
TAM = \frac{335}{3} - T_{PD \text{ mux}} - T_{PD \text{ input}} - T_{PD \text{ output}}
\]

\( TAM = (\frac{335}{3} - 6 - 17 - 18) \) nsec

\( TAM = 70.66 \) nsec

Note that \( T_{sample} \) was calculated using maximum propagation delays. Thus 70 nsec memory will be adequate for this design.
Figure 6. System Block Diagram
CHAPTER VI
RESULTS AND CONCLUSIONS

The goal of this research project was achieved by building a baseline architecture in the lab that followed closely the architecture discussed in Chapter IV.

One of the major problems encountered during this project was getting the communication link functioning properly. As pointed out in Chapter IV, the communication link used was a dual port RAM. The block diagram of the dual port is shown in Appendix A. We found that the dual port RAM was highly susceptible to noise, particularly the input latches and multiplexers. This problem was eliminated by increasing the ground strapping between chassis and separating the multiplexers and latches into two chips rather than using a combined mux latch chip.

The arbiter used in the design of this controller was an 8085 with 2K of EPROM and a large number of I/O ports. The initial design goal for the arbiter electronics was to provide a flexible design which could be easily implemented without hardware redesign. Optimizing the failure rate by minimizing the gate count of the arbiter was not a consideration. In a real life application, the arbiter would be composed of highly reliable parts, or be a redundant system, or both.
The dynamic testing of the system was accomplished through the use of three microprocessor development systems with in-circuit emulation. The results of testing indicate that our current design will work properly as a generic fault-tolerant controller. Specific applications can be achieved through changes in the software. The current software development project is expected to continue for the next 18 months. When the project is completed it will be used to improve the reliability, availability and safety in machinery and process control applications.
REFERENCES


APPENDIX I
DUAL-PORT RAM
Dual-Port RAM Communication Link