Wireless Reconfigurability of Fault-Tolerant Processing Systems

by

Melinda Y. Tang
S.B. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2007

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September, 2008

Author____________________________
Department of Electrical Engineering and Com~~puter~~ Science
July 31, 2008

Certified by____________________________ [illegible]
Joseph A. Kochocki
Charles Stark Draper Laboratory
Thesis Supervisor

Certified by____________________________ [illegible]
Barbara H. Liskov
Institute Professor
Thesis Advisor

Accepted by________
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

This page intentionally left blank

Wireless Reconfigurability of Fault-Tolerant Processing Systems

by

Melinda Y. Tang

Submitted to the Department of Electrical Engineering and Computer Science on July 31, 2008, in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This thesis examines the use of wireless data buses for communication in a real-time computer system designed for applications with high reliability requirements. This work is based on the Draper Laboratory Software Based Redundancy Management System (SBRMS), which is a fault-tolerant system architecture that uses data exchange and voting via Ethernet connections between redundant hardware components to detect and recover from faults. For this thesis, a triplex redundant system was developed that utilized the key features from the SBRMS design, including commercial-off-the-shelf hardware components and robust software partitioning, while replacing the standard Ethernet connection with a wireless Ethernet connection. The implementation of this system is presented and the benefits and potential applications of such a system are discussed.

Technical Supervisor: Joseph A. Kochocki
Title: Principle Member of the Technical Staff, Charles Stark Draper Laboratory

Thesis Advisor: Barbara H. Liskov
Title: Institute Professor, Massachusetts Institute of Technology

This page intentionally left blank

ACKNOWLEDGEMENTS

I would like to start by thanking Barbara Liskov, whose comments and advice were of great help during the writing of this thesis. Thanks to everyone at Draper Lab for their help during my stay here: to Joe Kochocki for his guidance throughout this year, Mark Lyon and Erik Antelman for their technical advice, and a special thanks to Terry Sims for taking me on late in the year and providing great insight and support.

Thanks to my family for giving me a place to go when I needed to get away, for bringing me food to make sure I was eating well, and for reminding me that it's okay to take a break every once in a while. I couldn't have done this without your constant support, encouragement, and patience.

And to all of my friends who made my experience at MIT beyond anything I could have imagined, this is for you. To Burton1, thanks for giving me a family and a home away from home these past 5 years. To the grad students of SPCrew, thanks for always making sure I didn't work too hard and for showing me what a "typical Wednesday night" really is. To the occupants of the 141-Table, thanks for the banter, the late night conversations, the MacG runs, and of course, the unpredictable drama that made life a little more interesting. To Patricia, thanks for being there for me for the past 18 years and for always lending a sympathetic ear when I need someone to talk to. Even from thousands of miles away, you've managed to help make my long days in lab more bearable. To Jenwoo, thanks for being my partner in crime ever since freshmen orientation. We've experienced so much together, and I don't think I could have survived MIT without you by my side. May we continue to speak only in acronyms as we live our lives full circle. And finally, to Steph, I don't think I can properly express in words how much I've learned from you. Between the countless nights and unusual encounters in the reading room, the hundreds of orders from Thailand Café, and all the random nonsensical outbursts of song and dance…you certainly helped make this thesis-writing process a surprisingly fun time. Thanks for the laughs, the distractions, and thanks for making my last two years at MIT the most memorable ones.

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under the Reliable Computing Architecture Development and Prototyping Independent Research and Development (IR&D) program, charge no. 21817-001.

Melinda Tang
July 31, 2008

This page intentionally left blank

Contents

List of Figures

Chapter 1

Introduction

As we move forward in this digital age, safety-critical real-time computing has become increasingly important in applications such as medical devices, automotive controls, and spacecraft systems. In these applications, human safety is a major concern, and so it is especially crucial that the systems operate reliably even in the event of a failure. This thesis is related to fault-tolerance within the realm of guidance, navigation, and control systems. These systems that operate air, space, land, and sea vehicles must be highly reliable and they often must remain reliable and self-contained for long periods of time and under harsh conditions. With such applications where a failure of one or more functions could be catastrophic, a well-designed fault-tolerant processing system is necessary to ensure mission success and safety of the crew and vehicle.

The techniques for building fault-tolerant systems have been established for several decades, and now more attention is being placed on making these systems not only reliable but also flexible and cost-effective. To reduce development and maintenance costs of these systems and to improve reusability and scalability, providers of these systems look to commercial-off-the-shelf (COTS) components based on industry standards instead of custom, proprietary components. Thus, designers can reuse components from mission to mission and no longer need to redesign entire systems for each particular application.

Another area to explore in terms of adaptable and cost-effective fault-tolerant systems is dynamic reconfiguration. When a permanent fault is detected, the system might be able to reconfigure either to replace the failed component or to isolate it from the rest of the system while it undergoes recovery. Reconfiguration might also occur without the presence of a fault; if the fault-tolerance requirements of an application change, the system should be able to dynamically adjust to those changes. This way, rather than having multiple system designs for multiple applications, a single design can be used for many applications simply by reconfiguring to meet particular needs as they come along.

This thesis focuses on the concept of reconfigurability of fault-tolerant systems utilizing wireless data buses. Wireless connections between onboard avionics components allow for more mobility than wired connections, and networks can be formed spontaneously as more computers come within range of each other. With this freedom, wireless provides an ease of reconfigurability that is not feasible for wired systems. As an example, when two orbiting vehicles dock together, their separate fault-tolerant systems can communicate wirelessly with one another and reconfigure to form a new system, with additional computing resources contributing to the overall redundancy management and fault detection. Thus by combining the wireless aspect with COTS hardware, we have the basis for extremely adaptable and cost-effective fault-tolerant systems.

1.1 Wireless Avionics

Wireless technology has progressed to the point where it is now considered a realistic option for communication in aerospace applications. The main advantage is the elimination of signal and information cabling, which in a typical system contribute to the overall system weight and, as a result, affect the payload and performance of the vehicle [1]. Reducing the amount of cabling between components and thus the overall weight, results in lower fuel costs and less power required to operate the vehicle. Another benefit is the overall improvement in onboard mobility obtained through ease of equipment change-out as well as placement flexibility [2]. Not only does wireless provide more freedom onboard an aircraft, but it also opens the door for applications that call for communication between two or more separate vehicles.

Today's armed forces are already trying to utilize modern networking technology for battlefield communications. In a setting with no existing communications infrastructure, they must bring their own networks with them, set them up on the fly, and have the networks automatically adapt to the constant movements of their components. Each node in these "ad-hoc" networks has intelligence to discover other nearby nodes, dynamically determine optimal paths for forwarding data to other nodes, and automatically heal breaks in the network due to node failures or changes in connectivity. Wireless ad-hoc networking can be used, for example, with Unmanned Aerial Vehicles (UAVs) where multiple UAVs can form a network to effectively collect and forward data to each other as well as to ground nodes at a base station [3].

Present systems are typically high-cost because they implement wireless communication using specialized radios in designated military radio bands [4]. However, wireless networking for avionics applications is becoming more attractive due to the recent development and availability of low-cost COTS radio technologies such as the IEEE 802.11 standards. This thesis project utilizes 802.11g-compliant wireless devices to enable communication between separate channels of a fault-tolerant system.

1.2 Research Overview

This thesis project builds upon an existing fault-tolerant computer system developed by Draper Laboratory. The original system called the Fault-Tolerant Parallel Processor (FTPP) was built for NASA's X-38 Crew Return Vehicle, and featured four redundant flight processors that communicated via specialized hardware [5]. That system was later redesigned to eliminate all custom proprietary hardware components. The Software-Based Redundancy Management System (SBRMS) prototype was developed, which utilized a partitioned operating system running on COTS hardware and a communication protocol implemented entirely in software. The prototype consisted of three redundant flight processors linked using point-to-point Gigabit Ethernet connections. The FTPP and SBRMS systems are described in detail in Chapter 3.

This research is part of an effort at Draper Laboratory to examine the feasibility of a dynamically reconfigurable wireless fault-tolerant system. This work explores the benefits of using wireless data buses to achieve this goal. Building upon the component architecture developed for the SBRMS prototype, we present the design and implementation of a three-channel COTS system that uses a wireless 802.11g connection in place of the Gigabit Ethernet links.

1.3 Thesis Outline

Chapter 2 presents background information on the subject of fault-tolerance. It also describes some commonly utilized fault-tolerant architectures and their pros and cons. Chapter 3 gives detailed descriptions of the X-38 FTPP and the SBRMS prototype systems that this work is based on. Chapter 4 presents a conceptual overview of wireless reconfigurable fault-tolerance. It explains the design of the wireless fault-tolerant system and includes a description of the hardware components that were used. It also presents the performance results obtained and compares these results with those of the SBRMS system. Chapter 5 concludes this thesis with a summary of the work done and suggestions for future research.

Chapter 2

Background

This chapter provides background information on the subject of fault-tolerance. Section 2.1 discusses some common fault-tolerance concepts. Section 2.2 then describes typical fault-tolerant architectures and their advantages and disadvantages.

2.1 Fault Tolerant Concepts

Fault-tolerant systems are applied in avionics, medical, nuclear power, and other applications when the consequences of failure of the operations executed by system are intolerable due to hazards to human health or risk of mission failure and system and/or mission loss. This section describes some fault-tolerant computing concepts and terminology often used in the context of avionics systems.

2.1.1 Fault Classifications

Fault-tolerance requirements of a system are determined by the consequences of different failure conditions. While the objective of a fault-tolerant system is to allow the system to continue to operate when it fails, or *Fail Operational*, a computer in an avionics system can fail in one of two ways: *Fail Passive* or *Fail Active* [6]. A Fail Passive condition can be thought of as a loss of function, where, once the system determines that data is no longer valid, the output actuators are signaled to assume some predetermined, desired state. A Fail Active condition, on the other hand, is a system malfunction rather than a loss of function; data seems to be normal but is actually incorrect, and thus output actuators assume some active but uncontrolled state. Systems have different requirements depending on the duration that a loss of function or malfunction can be tolerated. While some systems might react catastrophically to a brief loss of function, for others even a malfunction condition can be acceptable for a certain period of time. Understanding the consequences of the different failure conditions is essential in order to properly match a particular fault-tolerant architecture to a system's needs.

2.1.2 Redundancy

The most common techniques for achieving fault-tolerance in safety-critical systems use redundant hardware components. Each redundant unit in a system is referred to as a *channel* or *fault containment region* (*FCR*). A channel typically contains a processor, memory, I/O interfaces, and communication interfaces to other channels. The FCRs are electrically isolated from one another, each with its own separate power supply and clocking mechanism. An FCR will operate correctly regardless of logical or electrical faults outside of the region, and conversely a fault within an FCR cannot propagate outside of that region [7].

One purpose of redundancy is to provide fail-operational functionality. If one hardware component goes down, then one of the redundant components can be brought in to continue operation of the system. Redundancy combined with management software can also help with detecting faults. Since identical hardware running identical software should logically produce the same outputs, by comparing the outputs and seeing if they are the same or not, we can determine if there has been a fault in one of the channels.

The obvious disadvantages of redundancy are the increases in cost, size, and weight of the system. Adding redundant computers increases the overall complexity of the system and could possibly decrease the system throughput and bandwidth. Also there is a higher fault arrival rate to the system since there are more components that can fail. These are the tradeoffs that need to be considered when looking to improve reliability using redundancy.

2.1.3 Synchronization

An important design consideration for fault-tolerant systems is whether or not the components of the system should operate *synchronously* or *asynchronously*. For synchronous systems, sampling of sensor measurements must occur simultaneously and identical software must be executed by each channel at the same time. For asynchronous systems, the channels can operate autonomously; they read sensor values and execute controls but are not required to perform these functions at a specific point in time [8].

The advantage of synchronous fault-tolerant systems is that faults can be detected using simple bit-for-bit matching (exact agreement) of the outputs [6]. Exact agreement is possible since channels executing identical software on the same inputs should produce the same outputs if there are no faults. The disadvantage of a synchronous system is that it requires all of the channels to have a consistent time clock. It is possible to wire all of the channels with the same electrical clock signal, but this is not fault-tolerant since it introduces a single point of failure. Thus, for fault-tolerant systems each channel uses and maintains its own logical clock. Since local processor clocks will drift over time, we need some method, such as a distributed clock synchronization algorithm, to keep all channels in agreement over the global time. These algorithms are often difficult to implement, especially as the number of channels increases, and sometimes can cause more clock drift [9].

Asynchronous systems are attractive because the channels are uncoupled and no clock synchronization schemes are required. However, since channels are not synchronized and are non-deterministic, redundant processors might not produce identical outputs. Thus we can no longer use simple bit-for-bit voting, but rather we are limited to performing approximate agreement methods and reasonableness tests on outputs to detect faults [10]. Approximate agreement methods include voting schemes such as Median Value Selection (MVS), which selects the median numerical value out of all the channel outputs. If, for example, the output of one channel in a three-channel system is faulty and turns out much larger or much smaller than the other two outputs, through MVS it will not be selected and the system will operate correctly. Since approximate agreement methods do not explicitly detect and report failures, undetected failures can build up until the system can no longer operate.

The FTPP and SBRMS are both synchronous systems.

2.1.4 Cross Channel Communication

In order for channels to exchange data, there needs to be some kind of communication mechanism between all of the channels. This *Cross Channel Data Link* (*CCDL*) provides the means for a multi-channel system to perform tasks such as comparing sensor data among channels for identifying faults or exchanging clock information for maintaining synchronization [10]. Generally, the tighter the coupling between channels, the more important the physical CCDL medium and data exchange methods become.

The X-38 FTPP channels are connected by fiber optic cables while channels in the SBRMS architecture use Gigabit Ethernet connections. This thesis utilizes a wireless 802.11g CCDL.

2.2 Fault-Tolerant Architectures

A typical avionics control system consists of sensor inputs, computation, and outputs to system actuators. Depending on its fault-tolerance requirements, a system will have a particular architecture which will determine the type of computations that occur before sensor data gets sent to the actuators. This section gives a brief overview of typical fault-tolerant architectures and their benefits. More detailed explanations of these architectures can be found in [6].

2.2.1 Simplex

The most basic computing architecture is the classic control *simplex* system, shown in Figure 2-1, which provides no fault-tolerant features. This type of system is used when occasional loss of function or malfunction conditions are considered acceptable and lead to no critical loss. Timers or built-in-test (BIT) units can be added to help detect some, but not all, failures. If a failure is detected, then a hardware mechanism is signaled to disengage the malfunctioning computer from the system. See Figure 2-2. These BIT features increase the probability that the computer will fail passive rather than fail active.

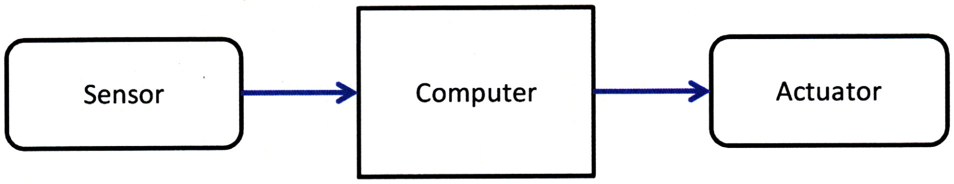


Figure 2-1: Simplex with no fault-tolerance functionality. Sensor data gets computed into an output value that is sent directly to the actuators.

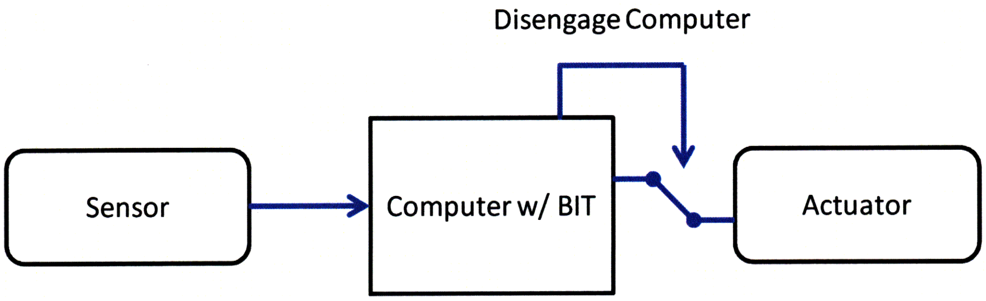


Figure 2-2: Simplex with Built-In-Test. In the event that the BIT detects a fault, the computer is disengaged from the system until it can be repaired or switched out.

2.2.2 Dual Standby

Simple redundancy can be implemented as a *dual standby system*, seen in Figure 2-3, where one of the redundant computers acts as the primary channel and the other acts as the backup. If a fault is detected in the primary, then the backup will be engaged to replace it. The backup can be considered "hot", where it is powered continuously and can be automatically used as a replacement when necessary, or "cold", where it is only powered on when necessary but as a result takes longer to engage. Dual standby systems can be used to prevent loss of function, but they provide no more protection against malfunctions than a simplex system with BIT features.

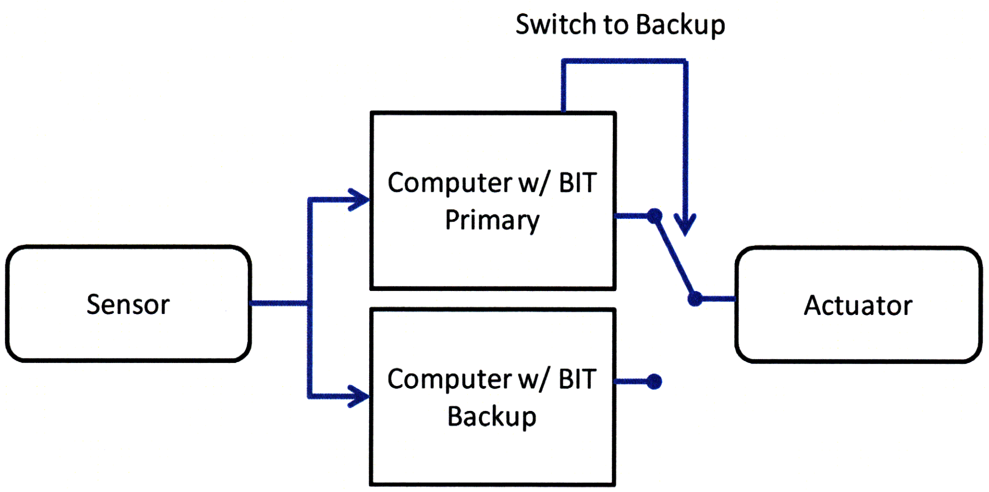


Figure 2-3: Dual standby. Should the primary channel fail, a backup is available to take its place.

2.2.3 Self-Checking Pair

A *self-checking pair* (*SCP*) system serves as a dual redundancy architecture that does prevent malfunctions. In this case, there are two processors executing the same software using the same inputs and they compare outputs to determine if there is a fault. If the processors produce identical outputs (exact agreement) or agree within some predefined boundary (approximate agreement), then the output value is sent to the actuators. This is shown in Figure 2-4. Though a self-checking pair can detect malfunctions, it is twice as likely as a simplex system to experience loss of function since there are two computers needed to get one output. A modified approach could allow the system to continue to operate using one computer after a fault is detected. This is known as a *self-checking pair with simplex fault down*, but this again does not protect against malfunctions after the first fault occurs. This type of system has application if the system can be supervised after the fault or will only be operated for a short time, minimizing the chance of a second failure. Multiple self-checking pairs can be used to provide backup to the first pair, but this rapidly adds size and weight to the system and complexity in terms of keeping track of information from each processor pair.

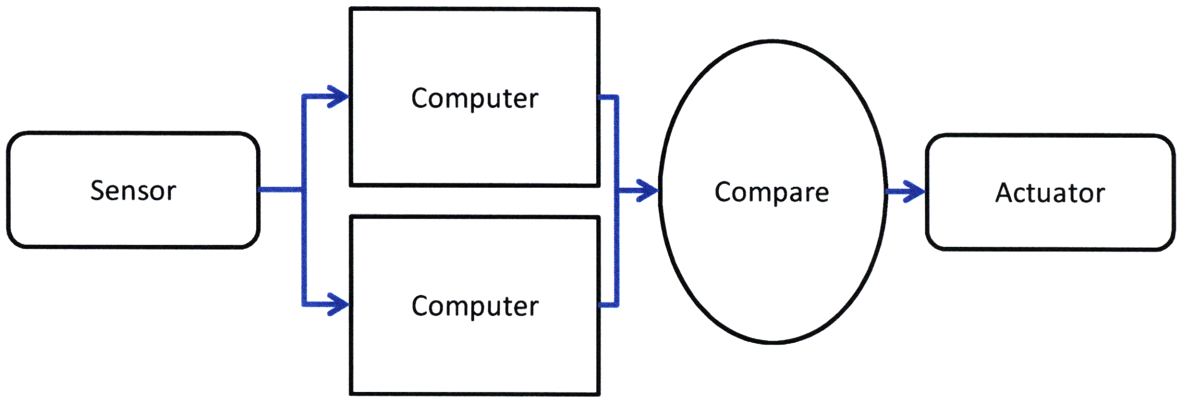


Figure 2-4: Self-checking pair. Two computers perform computations on the same sensor inputs and compare their outputs to determine if there is a fault.

2.2.4 N-Modular Redundancy

Another approach that provides low probability of both loss of function and malfunction faults is *N-Modular Redundancy* (*NMR*). In this type of architecture, there are N≥3 channels that compare their outputs to determine the overall output of the system. This can be done using bit-for-bit majority voting or MVS, depending on whether the system is synchronous or asynchronous and whether N is even or odd (if N is even, simple majority voting does not work). The comparison can be done either with an external voting device or can be distributed into the computers themselves. However, the use of an external voting device may not be a "true" NMR since it introduces a single point of failure in the voter. In the *distributed NMR* case, dedicated point-to-point CCDLs connect each pair of channels, and the channels exchange application outputs over the CCDLs and then the outputs are voted within each channel.

A *Triple Modular Redundancy* (*TMR*) system, illustrated in Figure 2-5, is an NMR setup where N=3, that helps protect against malfunction or loss of function after a single failure (though 100% fault-tolerance is not possible). TMR can be configured to fault down to a self-checking pair or simplex to help prevent both malfunction and loss of function after a second failure. Voting is also used to maintain consistent state data within the different channels, but this exchange of data can introduce a special type of failure known as the *Byzantine General's Problem* [11]. In this situation, a channel may not only give incorrect results, but also give those results inconsistently when exchanging data with the other channels. A Byzantine fault can result in no agreement being made on the outputs if two channels receive inconsistent input data from the third computer. TMR systems cannot prevent malfunction following a single Byzantine failure. However, a quad-modular redundant (QMR) system can effectively prevent both loss of function or malfunction following any two standard faults and one Byzantine fault.

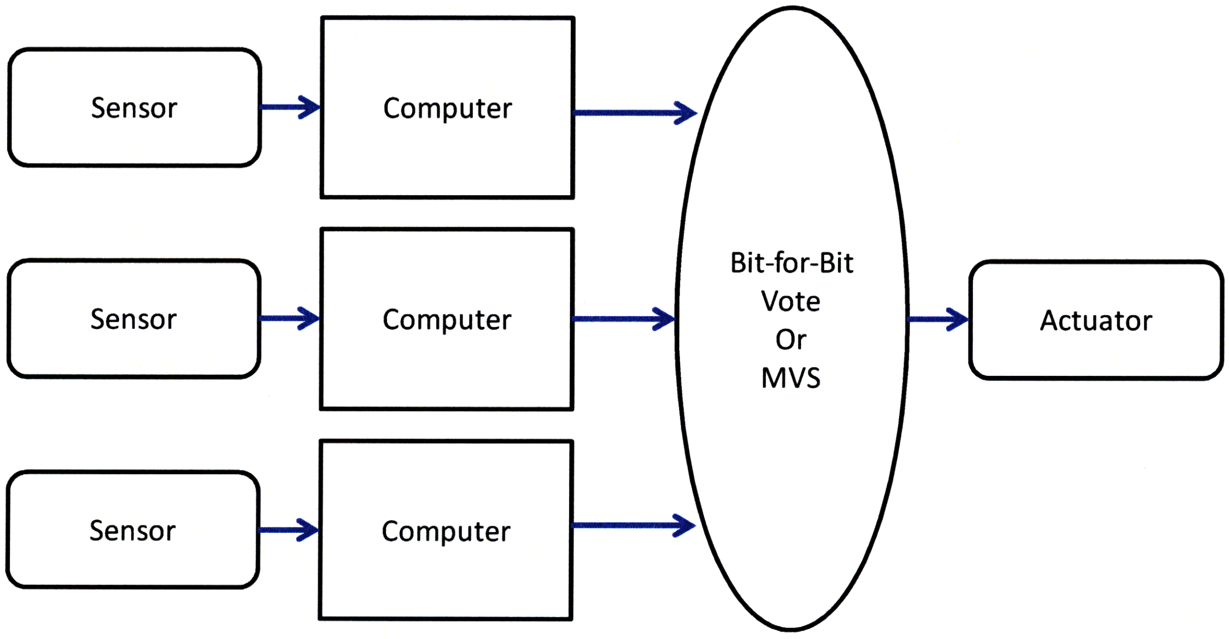


Figure 2-5: Triple Modular Redundancy. Using an external voting device, three channels compare their outputs to determine what value gets sent to the actuator.

Since a TMR system protects against both loss of function and malfunction, it is generally considered more reliable than a single SCP. Multiple SCPs are required to provide the same protection as a single TMR or QMR system. Also, unlike an SCP, TMR allows the system to identify exactly which channel has failed; as long as any two channels produce the same output, an output from a channel that is different from the agreeing pair will be considered faulty. However, ensuring that all the channels of an NMR system use identical inputs and stay synchronized can become very complex as more channels are added. Thus the decision between using NMR or SCP is a trade-off between complexity and the number of channels needed.

The X-38 FTPP architecture contains one distributed QMR set and five simplex processors. The SBRMS design uses a distributed TMR architecture. These are described in Sections 3.1 and 3.2.1 respectively.

Chapter 3

Related Work

This chapter presents two of the fault-tolerant computer systems previously developed at the Charles Stark Draper Laboratory that this research is based on. Section 3.1 gives an overview of the X-38 Fault-Tolerant Parallel Processor. That system was later redesigned and developed into the Software-Based Redundancy Management System, which is described in Section 3.2

3.1 X-38 Fault-Tolerant Parallel Processor

The Fault-Tolerant Parallel Processor (FTPP) is a Byzantine-resilient quad-redundant avionics system that was developed by Draper Laboratory for use on NASA's X-38 experimental Crew Return Vehicle. This system is based on COTS processor boards that are interconnected with specialized voting hardware known as the *Network Element*. The following descriptions of the FTPP architecture and Network Element design draw from [5], [12], and [13].

The X-38 FTPP is made up of five FCRs. All five channels contain one simplex (non-redundant) processor known as the Instrumentation Control Processor (ICP), while only four of the channels contain one processor of the quad-redundant Flight Critical Processor (FCP). The ICPs perform input and output with devices outside of the FTPP, including sensors and actuators. The FCP uses the data collected by the ICPs to perform the actual guidance, navigation, and control (GN&C) operations and all functions that require fault-tolerance.

In the X-38 avionics architecture, a set of sensors is only connected to a single channel. The ICP on each channel distributes its sensor values to the other channels, and then the sensor data gets voted and passed along to the quad-redundant FCP. The FCP uses the voted sensor values to execute the GN&C code and calculate the actuator commands. The output from the FCP is voted again before being sent back to the ICPs that control the actuators. This process is depicted in Figure 3-1.

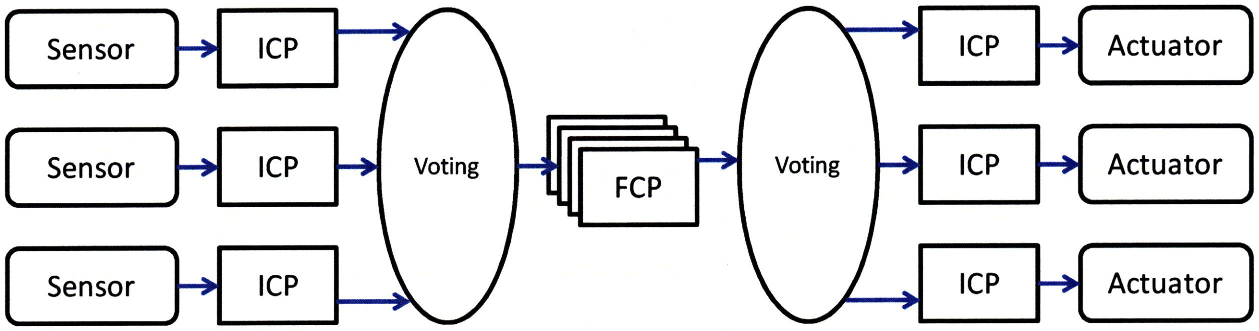


Figure 3-1: Conceptual X-38 FTPP architecture.

All processor-to-processor communication in the FTPP, whether it is between processors in the same FCR or between separate FCRs, is facilitated by specialized hardware modules known as the Network Element (NE). Each of the five FCRs contains one NE and the NEs are connected in a complete graph topology. The NE is the heart of the fault-tolerant functionality of the FTPP. It is responsible for performing message exchanges and voting all the data such as between the ICPs and FCP as described above. The NEs are also responsible for maintaining clock synchronization between channels so that they may perform the data exchanges at the correct times.

Figure 3-2 illustrates the physical layout of the X-38 FTPP.

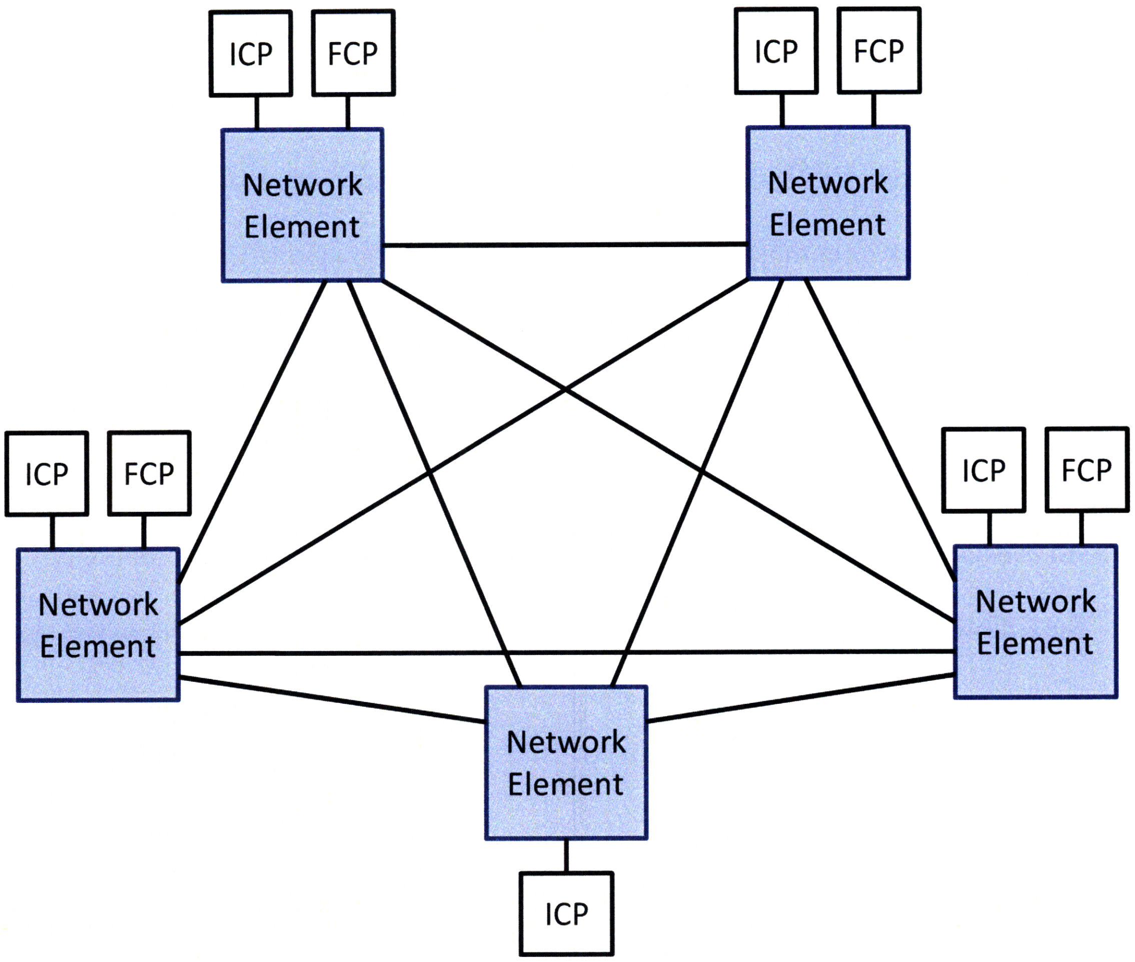


Figure 3-2: Physical X-38 FTPP Architecture. Each grouping of Network Element and processors is a fault containment region.

3.2 Software-Based Redundancy Management System

The Software-Based Redundancy Management System (SBRMS) was developed in an effort to reduce the complexity and cost of the FTPP. The main aspect of the SBRMS is that, unlike the X-38 FTPP, it utilizes only COTS hardware based on industry standards and no custom or propriety hardware. The fault-tolerance features of the FTPP that had been provided by the hardware-based NE are now implemented entirely in software.

The advantage of a software-based solution is the improvement in flexibility over a system that utilizes propriety hardware. The more of a system's functionality that is implemented in software, the less dependent the system becomes on specific hardware components and the fewer number of physical components in the system that can fail. A software implementation is also easier to build and make changes to, leading to lower development and maintenance costs. The tradeoff in migration of functionality from the NE to the redundancy management software in the SBRMS design is the additional CPU throughput required to execute that function, as well as additional software development complexity.

The following sections describe the SBRMS architecture and key features of the system's design.

3.2.1 SBRMS Architecture

The SBRMS utilizes a TMR architecture, with three redundant channels that are connected to each other using point-to-point IEEE 802.3 Standard Gigabit Ethernet CCDLs. Each channel contains a Flight Computer (FC), and simulated sensors and actuators are connected to each channel via a data bus. The FCs act as hardware fault containment regions and are each made up of a single board computer and various I/O modules. They are completely isolated from each other except for the CCDLs that connect them. The CCDL Ethernet device driver on the single board computer uses raw Ethernet Media Access Control (MAC) frames. For performance reasons, standard Internet Protocol (IP) and Transmission Control Protocol (TCP), which are usually responsible for ensuring reliable data delivery in a network, are not used.

The SBRMS takes a partitioned NMR approach which uses robust partitioning to allow each processor to host multiple applications. Each channel uses a COTS real-time operating system (RTOS) that complies with the ARINC 653 industry standard for robust partitioning, which provides very strong separation between processes running on a single processor. Thus this system can take what used to be separate flight critical computers and mission management computers and integrate them into a single vehicle management computer. Instead of having a channel that consists of separate physical processors such as the ICP and FCP of the X-38 FTPP, each channel contains a single processor and the various software applications are hosted in different partitions but share the same computing resources (CPU, memory, I/O, etc.). Each partition can be looked at as a software fault containment region. The benefit of this partitioned approach is that reducing the number of physical processors used in the system reduces the size, weight, and power requirements of a vehicle, as well as complexity (fewer parts).

Figure 3-3 illustrates an overview of the SBRMS architecture. There are three redundant FC channels called FC1, FC2, and FC3, and they are connected to each other using dedicated point-to-point Gigabit Ethernet CCDLs. Each FC is an SBC750GX single board computer with a PowerPC 750GX processor. The figure depicts redundant GN&C software running in one partition and channel-specific software running in the 2nd partition on each FC, but in the actual implementation there are multiple partitions. Figure 3-4 shows the actual SBRMS laboratory setup.

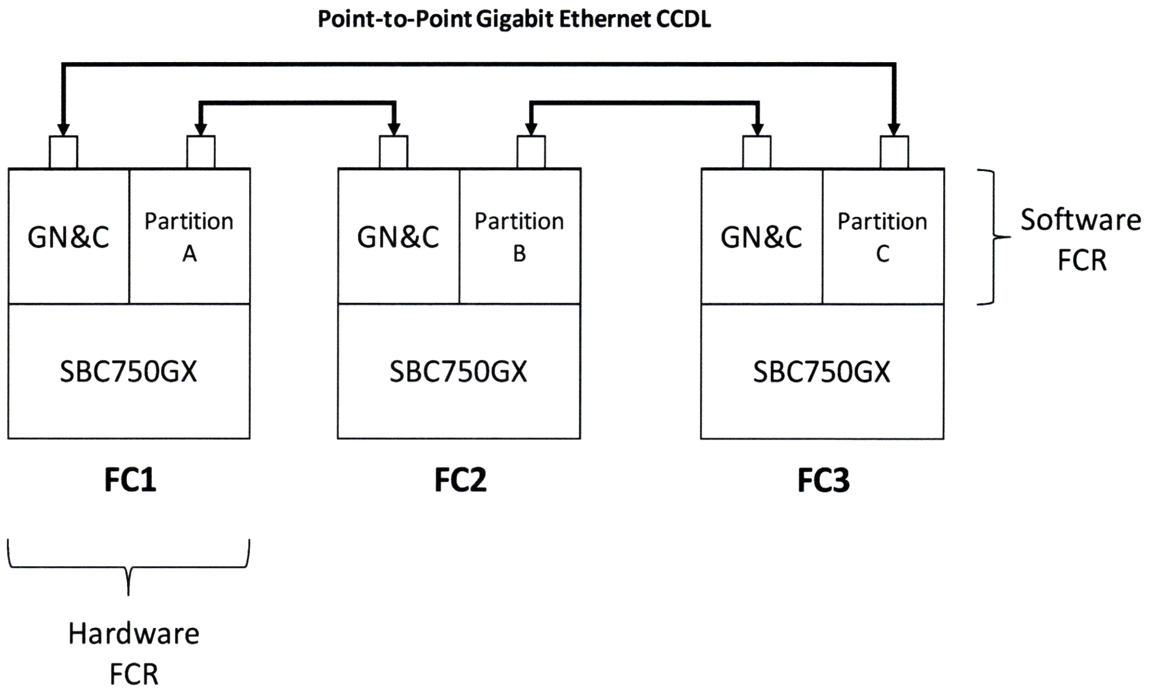


Figure 3-3: SBRMS System Overview

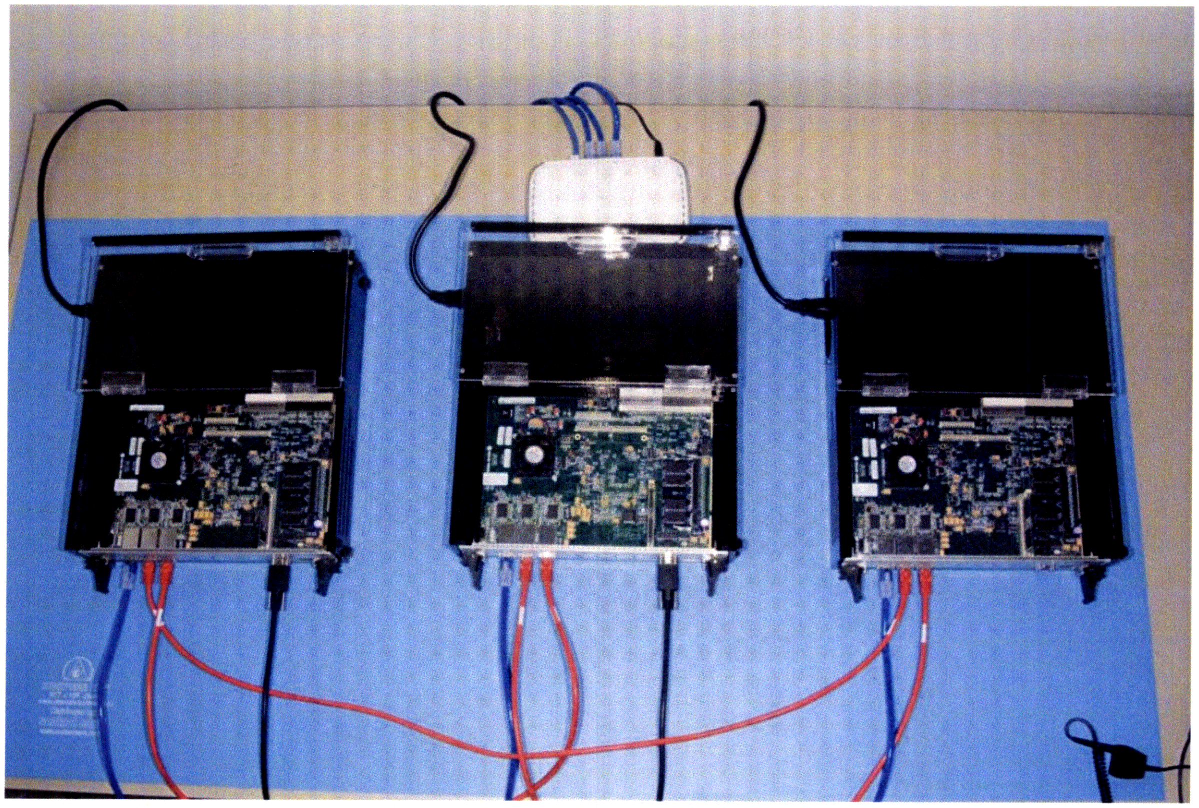


Figure 3-4: SBRMS Lab Prototype (Courtesy of Draper Laboratory)

3.2.2 Frame Synchronization

The SBRMS uses a *frame synchronous* approach in order to maintain clock synchronization between channels and to facilitate fault-tolerant redundancy management. All of the FCs synchronize their local clocks to a global clock determined through a distributed, Byzantine fault-tolerant clock synchronization algorithm [9]. The FCs divide each time into *major and minor frames*, where major frames are divided into minor frames and minor frames are divided into kernel ticks. The durations of the frames are determined by the longest and shortest partition periods respectively, and then the FCs schedule which partition applications will execute within each frame. Since the channels operate synchronously, then they will execute the same fixed partition schedule (nearly) simultaneously. This ensures that all channels obtain the same sensor inputs at the same time, execute the same application software at the same time using those inputs, and finally have application outputs ready and available for voting at the same time. As a result, the system can use simple bit-for-bit majority voting of sensor inputs and application outputs to detect faults.

Figure 3-5 shows the basic operation of the frame synchronous approach. The GN&C process executes at 50 Hz, or a period of 20 ms, which defines the duration of the minor frame. Before the GN&C partition executes, each channel's sensor inputs are distributed to all the other channels and voted. Then identical GN&C application code executes on each channel using the voted inputs. Finally, the GN&C outputs from each channel are distributed to all channels and voted. Other applications may execute within the minor frame after the GN&C processing is done, as defined by the partition schedule. This sequence is then repeated continuously.

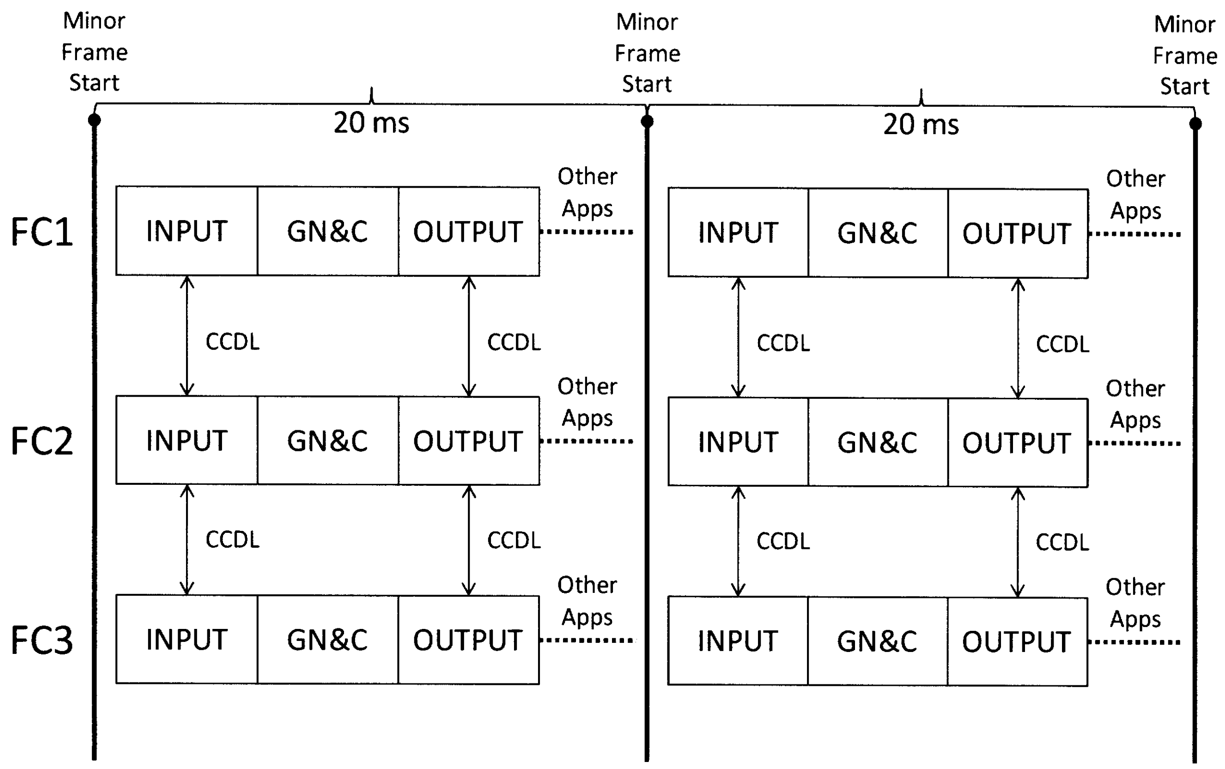


Figure 3-5: SBRMS Frame synchronous timing

The backbone of the SBRMS implementation of frame synchronization is the core operating system (OS) kernel tick. Kernel ticks allow the core OS to keep track of the passage of time on the processor, and nearly all core OS and partition operations that deal with time use kernel ticks. A kernel tick is a hardware interrupt that is provided by the PowerPC decrementer (DEC) register. Code executing on the PowerPC loads a value into the DEC register and then the register starts to decrement this value at a frequency that is determined by the system clock. When the value in the register becomes negative, a decrementer interrupt is generated. When this occurs, the assigned interrupt handler resets the DEC register, advances the kernel time by one tick, and then evaluates the partition schedule to see if the current partition should continue execution or if a different partition is scheduled to gain control of the processor.

To maintain synchronization, the three channels in the system use message exchanges over the CCDLs to determine how to adjust each local DEC value. At the start of the partition window, each channel starts polling for messages and periodically broadcasts a synchronization message to the other channels. When two messages are received on a particular channel, they are relayed to the other channels and the local clock is updated. The difference between when messages are *expected* to arrive and when they *actually* arrive determines the adjustment that needs to be made to each local clock. By doing this, the DEC interrupts of all the channels will be aligned and the partition schedule will be executed at the same time (with some bounded error) across the channels.

The current SBRMS implementation uses a 4000 Hz DEC interrupt frequency, which equates to an interrupt every 250 μs between interrupts or 80 interrupts per 20 ms minor frame.

3.2.3 Fault-Tolerant Data Exchange

Fault-tolerant data exchange in the SBRMS is similar but not identical to the data exchanges provided by the NE for the X-38 FTPP. Data exchange in the SBRMS is table-driven using system configuration files so that the schedule of exchanges is defined at design time. The actual exchanges are done differently for input processing versus output processing, as are described in the following sections.

3.2.3.1 Sensor Input Processing

For sensor input processing, the SBRMS uses a modified two-round exchange process. It is modified because there are only three channels and in order for a system to tolerate one

Byzantine fault using two rounds of data exchange, at least four channels are required. To remedy this, each channel calculates a digital signature to be sent with its sensor data to the other channels. This is a form of message authentication that allows the system to determine if a fault has occurred to a channel based on whether or not the signature matches the data attached to it.

During Round 1 of the exchange, each FC obtains readings from its own set of sensors through the local data bus and sends this data along with its digital signature to the other FCs via CCDL. Thus at the end of Round 1, each FC has the data that it received from its own sensors, as well as the sensor data and signatures received from the other two FCs. In Round 2, each FC reflects the data it received in Round 1 back to the other FCs. At the end of Round 2, each FC has two sets of data for each sensor collected over the two rounds, and using signatures they can determine the correct value of the sensor inputs. Figures 3-6 through 3-9 walk through an input processing scenario where data gets corrupted during one of the exchange rounds.

At the start of input processing (Figure 3-6), each FC obtains data from its local sensor. The sensor values for FC1, FC2, and FC3 are 4, 5, and 6 respectively.

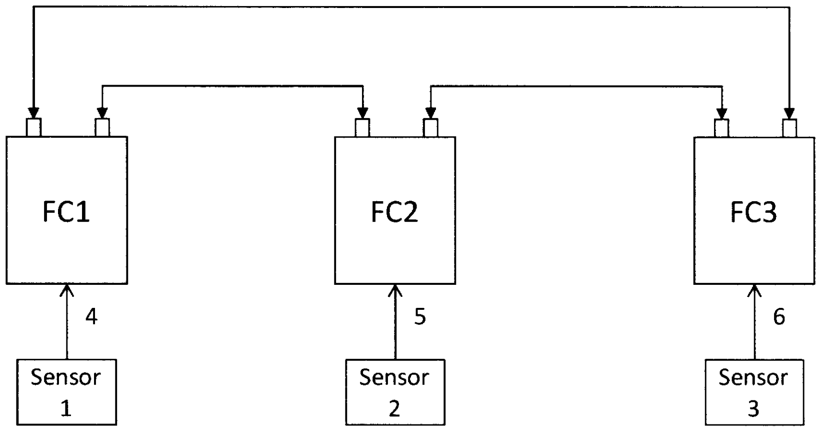


Figure 3-6: Start of input processing

During Round 1 of the exchange (Figure 3-7), each FC calculates a digital signature "S" for its local sensor data and sends that signature and the data to each of the other FCs.

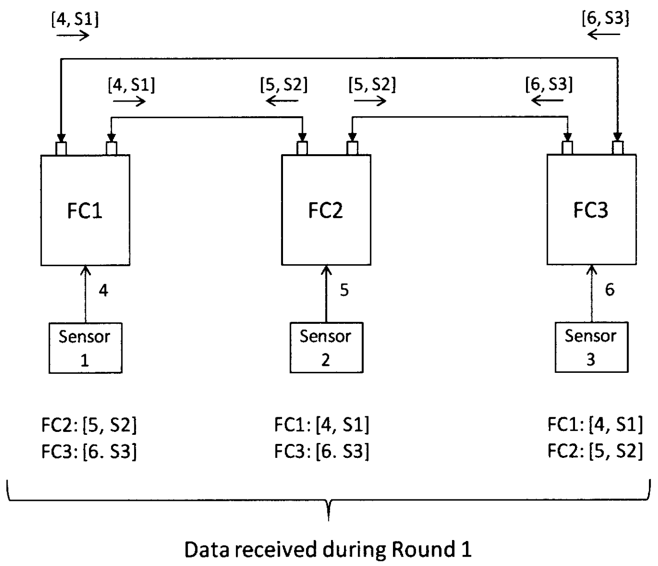


Figure 3-7: Round 1 exchange

In Round 2 of the exchange (Figure 3-8), the data and signatures received by a channel during Round 1 are reflected back to the other channels. In this scenario, the FC2 transmitter corrupts the original FC3 data during the reflection to FC1, and sends a value of "7" instead of the correct value of "6".

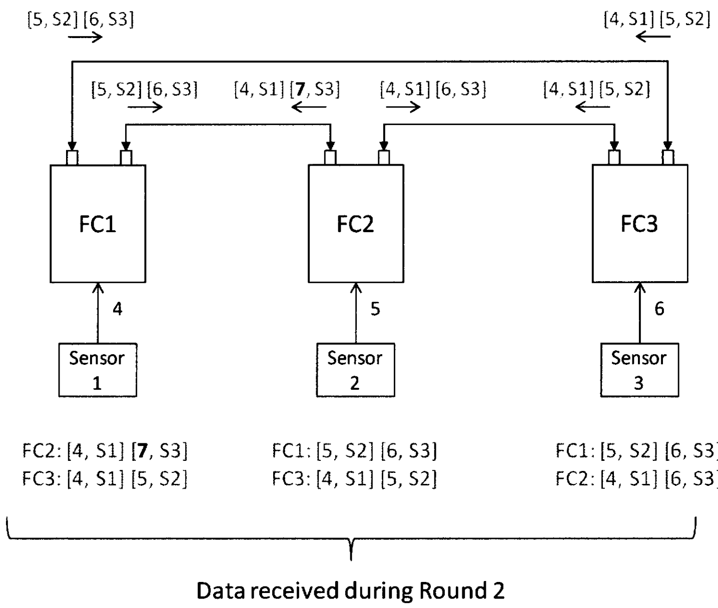
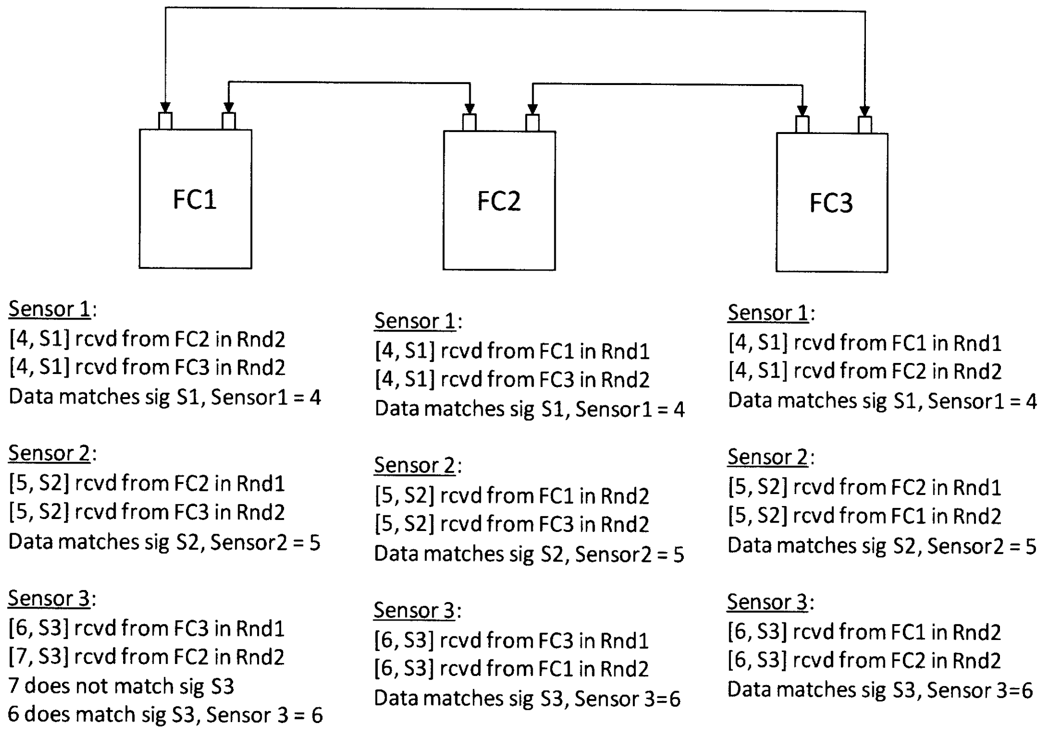


Figure 3-8: Round 2 exchange

However, by matching the data with the generated signatures, each FC can determine the correct sensor input values. Since the value of "7" sent to FC1 does not match the signature generated by FC3 for that sensor, that value is discarded and the correct value of "6" sent by FC3 to FC1 is used. See Figure 3-9.



Sensor 1:
[4, S1] rcvd from FC2 in Rnd2
[4, S1] rcvd from FC3 in Rnd2
Data matches sig S1, Sensor1 = 4

Sensor 2:
[5, S2] rcvd from FC2 in Rnd1
[5, S2] rcvd from FC3 in Rnd2
Data matches sig S2, Sensor2 = 5

Sensor 3:
[6, S3] rcvd from FC3 in Rnd1
[7, S3] rcvd from FC2 in Rnd2
7 does not match sig S3
6 does match sig S3, Sensor 3 = 6

Sensor 1:
[4, S1] rcvd from FC1 in Rnd1
[4, S1] rcvd from FC3 in Rnd2
Data matches sig S1, Sensor1 = 4

Sensor 2:
[5, S2] rcvd from FC1 in Rnd2
[5, S2] rcvd from FC3 in Rnd2
Data matches sig S2, Sensor2 = 5

Sensor 3:
[6, S3] rcvd from FC3 in Rnd1
[6, S3] rcvd from FC1 in Rnd2
Data matches sig S3, Sensor 3=6

Sensor 1:
[4, S1] rcvd from FC1 in Rnd1
[4, S1] rcvd from FC2 in Rnd2
Data matches sig S1, Sensor1 = 4

Sensor 2:
[5, S2] rcvd from FC2 in Rnd1
[5, S2] rcvd from FC1 in Rnd2
Data matches sig S2, Sensor2 = 5

Sensor 3:
[6, S3] rcvd from FC1 in Rnd2
[6, S3] rcvd from FC2 in Rnd2
Data matches sig S3, Sensor 3=6

Figure 3-9: Match data and signatures to determine correct sensor values

3.2.3.2 Application Output Processing

Unlike sensor input processing, the outputs from application partitions are voted through only a one round exchange since the replicated GN&C application on each channel should produce the same output. Each FC obtains a value from its GN&C application, and then the three FCs exchange their values. Thus at the end of the round, each FC has the value computed by all three applications, and bit-for-bit majority voting can be performed to determine the correct output. The voted value is then passed to the I/O processing partition which manages the transmission of that value to the actuator connected to the FC. Figures 3-10 and 3-11 walk through an output processing scenario where one FC computes an incorrect result.

The GN&C software on each FC produces an output command and each FC exchanges that output data with all the other FCs. In this scenario, FC1 and FC3 produce the correct output command value of "7", while FC2 produces the incorrect command value of "8".

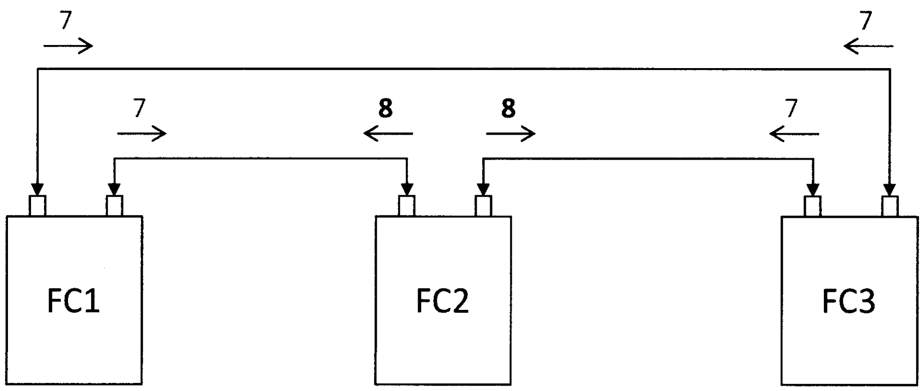
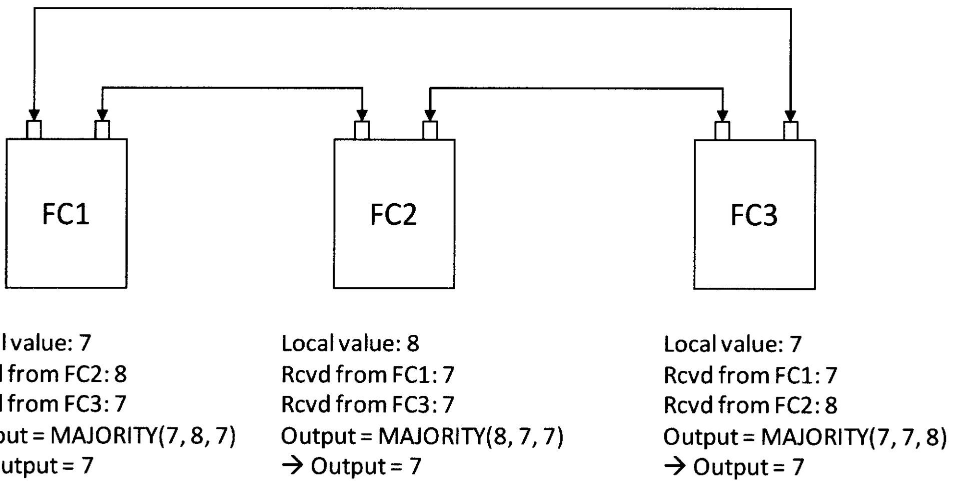


Figure 3-10: Output exchange

After the output values are exchanged, each FC uses majority vote on the data received from all of the FCs to determine the correct value of the command output. Since both FC1 and FC3 produced the same output value, the incorrect value of "8" gets voted out and the value of "7" is chosen as the correct output.



Local value: 7	Local value: 8	Local value: 7
Rcvd from FC2: 8	Rcvd from FC1: 7	Rcvd from FC1: 7
Rcvd from FC3: 7	Rcvd from FC3: 7	Rcvd from FC2: 8
Output = MAJORITY(7, 8, 7)	Output = MAJORITY(8, 7, 7)	Output = MAJORITY(7, 7, 8)
→ Output = 7	→ Output = 7	→ Output = 7

Figure 3-11: Each FC uses majority vote to determine the correct value of the command output

This page intentionally left blank

Chapter 4

Wireless System Design

The aim of this thesis work is to explore the concept of wireless reconfigurable fault-tolerant (WRFT) systems. In particular, the idea is to be able to reuse most of the work already done on the Draper Laboratory SBRMS design, with the only major difference being the use of a wireless CCDL in place of the Gigabit Ethernet CCDL. The use of wireless reduces mass and power requirements of a vehicle, improves mobility of the system, and allows networks to be formed and reconfigured spontaneously. Therefore by combining the main aspects of the SBRMS design—COTS hardware and robust partitioning—with the benefits of wireless, we have the basis for extremely flexible and cost-effective fault-tolerant solutions.

This chapter discusses the design and implementation of a wireless system based on SBRMS. First, Section 4.1 discusses the concept of reconfigurable fault-tolerant systems. Section 4.2 gives an overview of the proposed system design. Section 4.3 details the hardware used in this work. Section 4.4 provides details of the wireless implementation. Finally, Section 4.5 presents an analysis of the performance of the system.

4.1 Wireless Reconfigurability Concept Summary

The idea behind reconfigurable fault-tolerance is that a system should not need to have a permanently defined fault-tolerant architecture. That is, rather than have a system that is strictly TMR or strictly an SCP with simplex fault down, the system should be able to dynamically reconfigure to match certain architectures depending on the situation. As a result, a single design can be used to satisfy the fault-tolerance requirements of a variety of applications simply by reconfiguring to meet particular needs as they come along. For example, a QMR system can be reconfigured as a TMR system with one spare, as dual SCPs, or as one SCP with two spares, etc. These various configurations are illustrated in Figure 4-1.

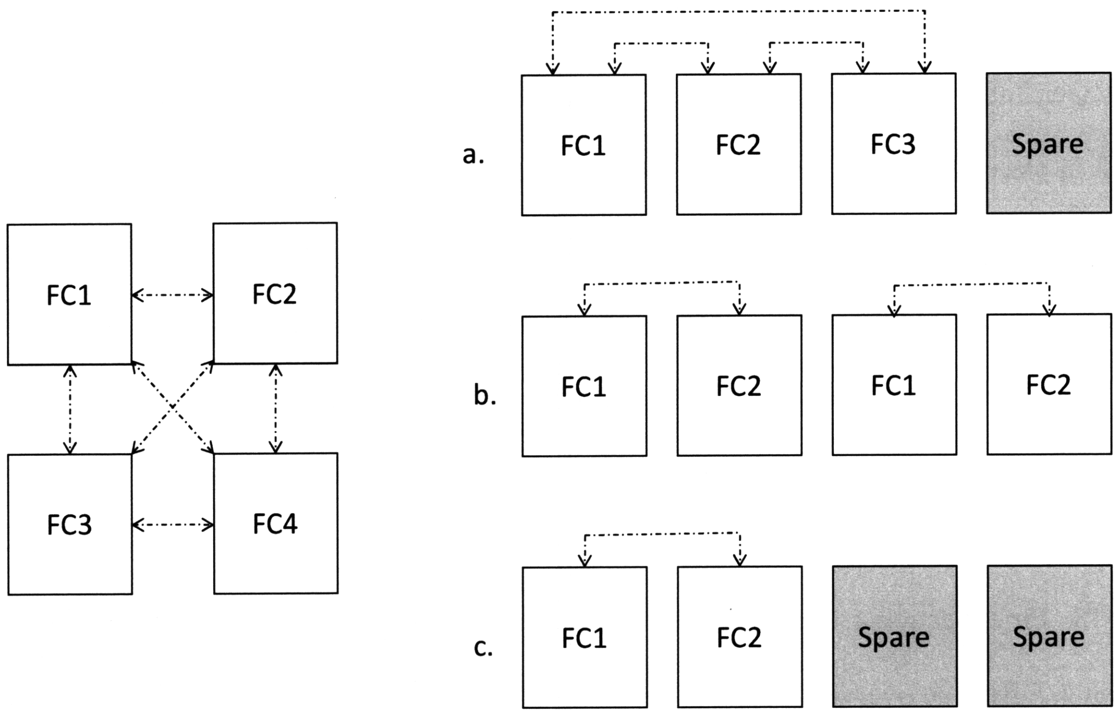


Figure 4-1: A QMR system can be reconfigured to form:

a) TMR with 1 spare

b) Dual self-checking pairs

c) One self-checking pair with 2 spares

Reconfiguration is often used as a failure response mechanism in multi-channel systems. In the event of a fault detection in one channel, that channel is isolated from the rest of the system and the remaining channels reconfigure their communication and voting protocols as necessary so as to not send/receive data to/from the faulty channel. The SCP with simplex fault down architecture is a very basic example of this. In a wired system, it is simple enough to be able to isolate a channel by programmatically disabling the links to that channel and turning off all communication going into it. A more difficult problem is that of introducing a new channel into the system such that all current channels recognize the new one and can communicate with it. This seems complicated for a wired system because components need to be physically rewired in order for the new channel to be included. It is more feasible if the channels involved are connected wirelessly since they can communicate and reconfigure on the fly, as long as they are within range of each other, and we do not need to worry about physical connections. Then for example, if a simplex computer is brought within range of a TMR set, the system should reconfigure to include the additional channel in the voting exchange, thus forming a QMR system.

The ability to distribute fault-tolerant system services and redundancy management over a wireless network opens the door for new types of fault-tolerant applications. Systems on separate vehicles can use this capability to communicate with each other across the physical boundaries that separate them, and reconfigure to form one overall system. With this, multiple UAVs could combine to form one assimilated vehicle or a lunar rover could dock to a base station, and the systems would still be able to maintain their fault-tolerant functionality and operate reliably. See Figure 4-2 for a sample application.

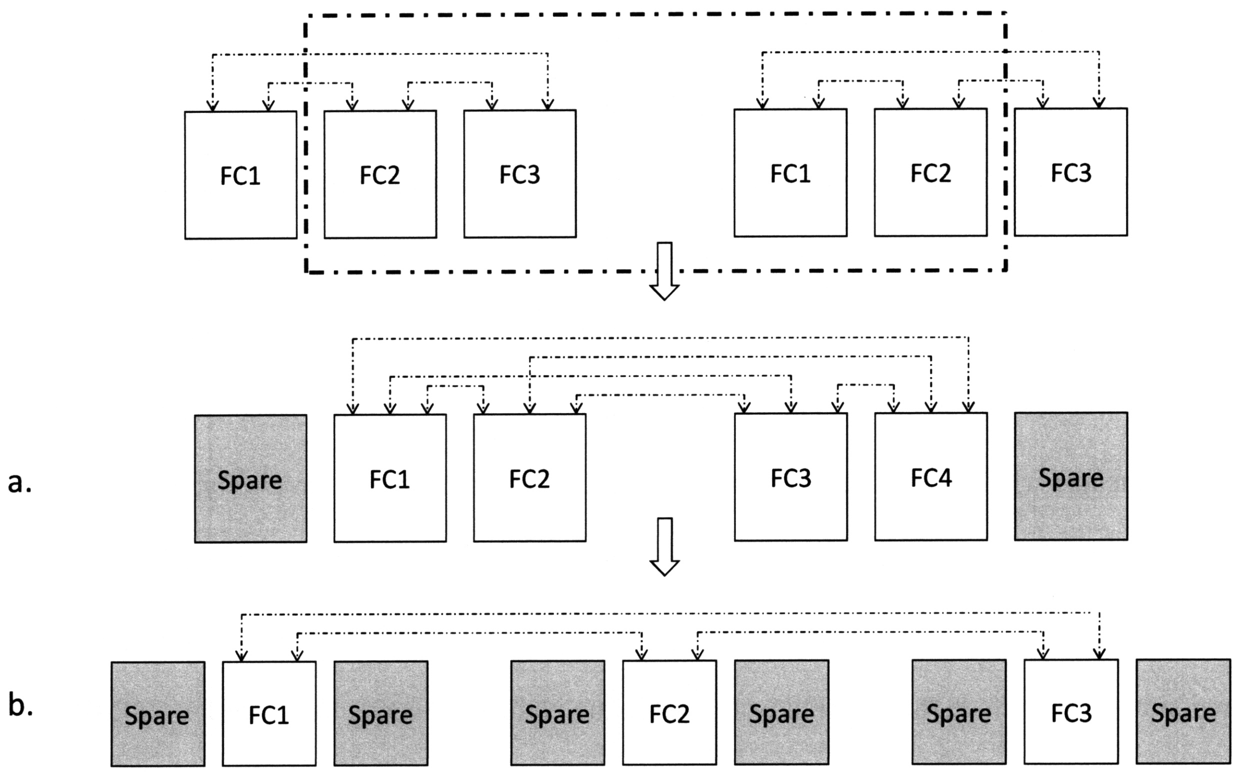


Figure 4-2: a) Two triplex UAVs combine to form single UAV with quad voting

b) A third UAV is introduced, and one channel from each vehicle is used in an overall triplex system

4.2 Design Overview

The goal for developing a WRFT system is to be able to combine the benefits of wireless with the benefits of the already implemented SBRMS. The increase in flexibility from taking a software-based approach instead of using specialized hardware, as well as the decrease in costs from utilizing a partitioned OS, are among the advantages of SBRMS that we would like to maintain in future systems. As much as possible we would like to preserve the main features of the SBRMS design. By reusing the existing design, we not only save on development time and costs, but we can also demonstrate the portability and modularity of the SBRMS implementation.

For example, switching out the Gigabit Ethernet CCDL for a wireless one, we can show that the design is not dependent on specific hardware.

The proposed setup of the WRFT system, like the SBRMS design, uses a TMR architecture with robust partitioning and consists entirely of COTS hardware components. Each channel is still an SBC750GX single board computer running the Wind River Systems VxWorks 653 RTOS, but replacing each point-to-point Gigabit Ethernet CCDL is a pair of Linksys WET54G wireless bridges (one on each end of a connection).

4.3 Channel Hardware

This section details the various hardware components that make up a channel in the proposed WRFT system.

4.3.1 Wind River SBC750GX

The SBC750GX is a single board computer in a 6U CompactPCI form factor that contains one IBM PowerPC 750GX microprocessor. The other features of the SBC750GX board include:

- Processor bus running at 133 MHz
- Marvell MV64360 system controller
- 512MB of DDR266 SDRAM
- 64MB of Flash Memory
- Three Gigabit Ethernet ports via RJ-45 connectors
- Front panel GPIO header
- PMC connectors

Figure 4-3 shows a block diagram of these and other of the board's major components. Figure 4-4 is an image of the actual SBC750GX board.

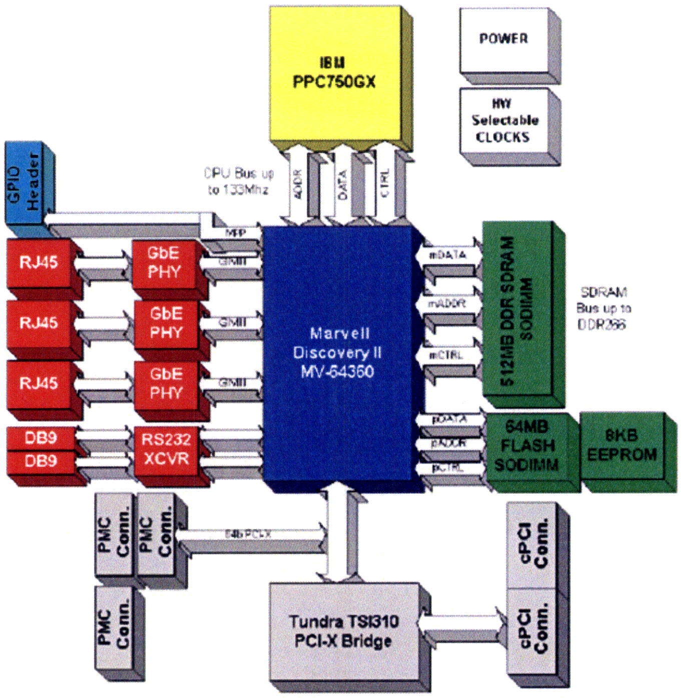


Figure 4-3: SBC750GX Block Diagram (Courtesy of Wind River Systems)

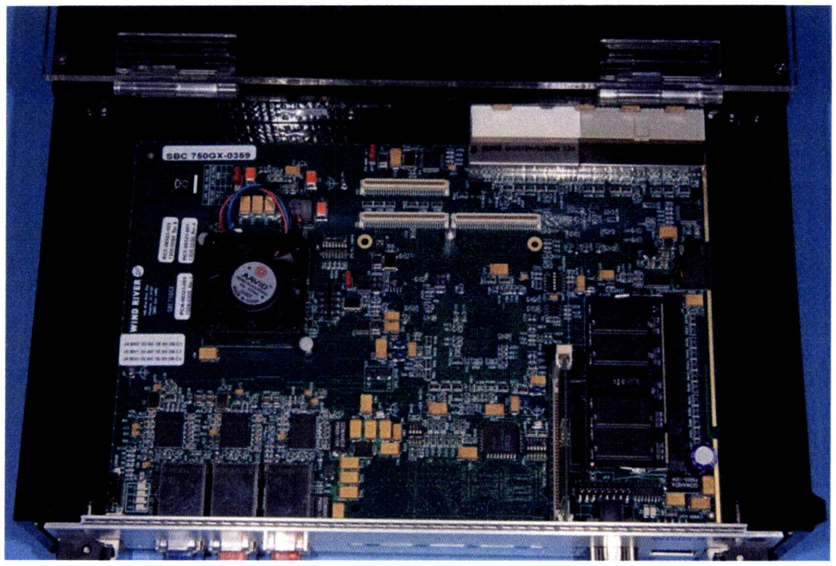


Figure 4-4: SBC750GX (Courtesy of Wind River Systems)

4.3.2 IBM PowerPC 750GX

The PowerPC 750GX is a 32-bit implementation of the IBM PowerPC family that operates at 933 MHz. This processor has a Memory Management Unit (MMU) that the VxWorks 653 RTOS uses for space partitioning. The 750GX also has a decrementer register that the RTOS uses to cause an interrupt after a specified delay. This is used to manage scheduling and synchronization.

4.3.3 Marvell MV64360

The Marvell MV64360 system controller provides a single-chip solution for PowerPC based applications. On the SBC750GX it interfaces to the CPU bus, DDR SDRAM, GPIO header, Flash memory, serial port, and Ethernet transceivers. The MV64360 chip has three Ethernet ports, each with its own Media Access Control (MAC) logic.

4.3.4 Linksys WET54G

The Linksys WET54G is a wireless Ethernet bridge that gives wireless connectivity to any wired Ethernet-equipped device. The main specifications include:

- Operates in the 2.4GHz frequency spectrum with throughput up to 54Mbps
- Complies with IEEE 802.11b/g standard
- 10/100 Auto-Crossover (MDI/MDI-X) port

The WET54G was chosen because it is an industry standard (IEEE 802.11b/g) COTS product that provides easy integration into the existing system. One key feature is that it is driver-free, so that it will work on any platform and under any operating system, including the VxWorks 653

OS used in SBRMS. Since that particular OS is uncommonly used for commercial applications, it is difficult to find wireless hardware that it can support without looking into high-cost specialized devices.

Another important aspect of the WET54G bridge is that it interfaces with Ethernet-ready devices, such as the SBC750GX, directly through the device's Ethernet port. The bridge provides wireless network connectivity by plugging into a device's Ethernet port via a standard RJ-45 cable. The device itself is blind to the presence of the bridge; it sends packets out through the Ethernet port the same way it would with a wired connection, and it is the responsibility of the bridge to direct those packets to the right destination. This feature is useful for comparison with the SBRMS implementation, which also uses the SBC750GX Ethernet ports to enable communication between channels.

4.4 Implementation Details

Each of the three FCs in this system consists of an SBC750GX board and two Linksys WET54G Ethernet bridges. The two WET54G bridges are connected to the SBC750GX via RJ-45 cable to its first two Ethernet ports (port0 and port1). The third Ethernet port (port2) is connected to the development and test Windows PC.

Although the FCs only send raw Ethernet MAC packets to each other and TCP/IP are not used (as implemented in the SBRMS), the configuration parameters of the bridges require an assigned IP address. Each of the WET54G bridges is also configured to clone the MAC address of the device it is connected to. Since each Ethernet port on the SBC750GX has its own MAC address, a bridge connected to a particular port will be associated with that port's address and will only accept packets destined for that address. Each bridge on an FC is paired to one bridge on another FC, and these two bridges are dedicated to communicating only with each other. To ensure this, the two bridges in a pair are configured to communicate with each other via ad-hoc connection on a particular wireless channel, with each pair set to a different channel. Figure 4-5 depicts an overview of the system layout.

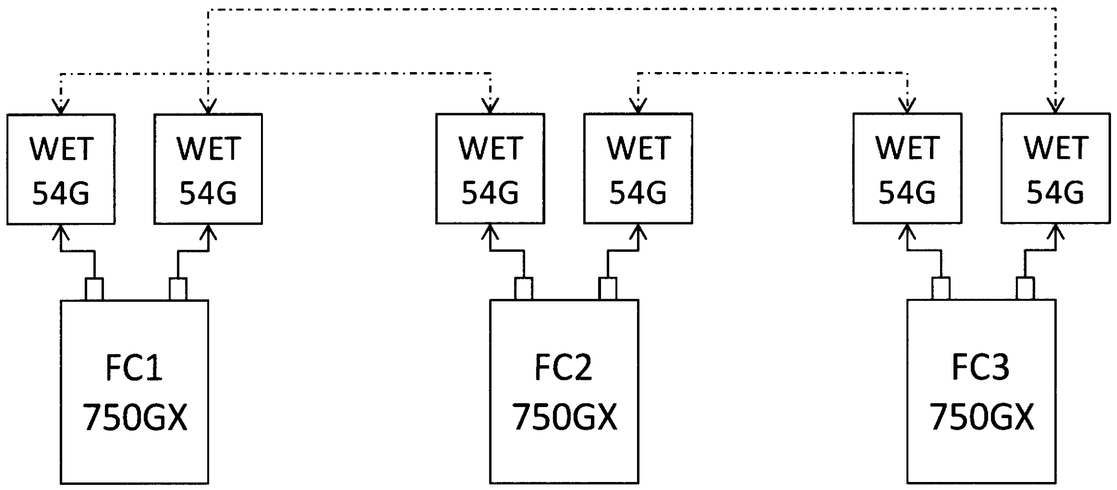


Figure 4-5: Layout of the wireless system. Each FC is a 750GX board with two WET54G bridges attached. Each WET54G device is paired with a bridge on another channel.

Hard-coded configuration tables define the destination port MAC addresses that each FC can send to as well as what port an FC should transmit out of to reach another particular FC. For example, FC1 should transmit out of port0 to reach FC2 and out of port1 to reach FC3 (see Table 4-1). When an FC goes to transmit a packet to another FC, it looks in the tables to find what MAC address to send to and which local port to transmit out of; any packets sent out of that port are then broadcast over a particular wireless channel by the WET54G bridge. The bridge on the receiving end is listening for packets on that same channel and will accept the transmitted packet since it is destined for that FC's address.

FC #	FC Linked to Port0	FC Linked to Port1
FC1	FC2	FC3
FC2	FC1	FC3
FC3	FC2	FC1

Table 4-1: Sample configuration table showing which channels are linked to each channel's Ethernet ports

4.5 Performance Analysis

This section describes the results of tests performed to evaluate the use of a wireless CCDL in place of the Gigabit Ethernet CCDL. For these tests, data was collected by the development and test Windows PC; each SBC750GX communicates with the host PC via an RJ-45 Ethernet connection as well as an RS-232 serial connection. The Ethernet connection is used by the Wind River Workbench tools on the PC for debugging purposes. The serial connection is used to send text outputs from a process running on a channel to a terminal window on the host PC.

4.5.1 IP Ping Test

To test the basic functionality of the WET54G bridges when used with the SBC750GX boards, two FCs were set up to communicate with each other using a simple ping function provided by VxWorks. The ping routine tests that a remote host is reachable by sending ICMP echo request packets, and waiting for replies. For this test, each of the two FCs used has an appropriately configured (as described in Section 4.4) wireless bridge plugged into Ethernet port 0 of the SBC750GX. Each FC sends ping messages to the IP address of port 0 on the other FC, and waits for a response. The reception of a response message proves successful wireless connectivity between the two FCs. For this test, 100 ICMP packets with a size of 64 bytes were sent at 1 second intervals with a 5 second timeout period. This was repeated over 5 trials to determine an average rate of packet loss. The results for the wireless connection compared with the wired Ethernet connection are summarized in Table 4-2.

	Packet Loss	
	Wired	**Wireless**
Trial 1	0%	5%
Trial 2	0%	3%
Trial 3	0%	10%
Trial 4	1%	6%
Trial 5	0%	5%
Average	0.2%	5.8%

Table 4-2: Summary of ping test results

By demonstrating successful transmission and reception of ICMP packets, this test confirmed the functionality of the WET54G bridges and the capacity of our selected channel hardware for wireless communication. However, although the channels are capable of communicating wirelessly, there is a noticeable difference in the reliability of the wireless connection compared to the wired connection.

4.5.2 Raw MAC Test

The previous test demonstrated wireless connectivity, but relied on IP for simplicity reasons (use of built-in VxWorks functions) to send and receive messages. However, to improve performance and to be able to compare to the SBRMS implementation, the system should be able to send raw Ethernet MAC packets without the use of IP. To demonstrate this capability this test utilized the SBRMS synchronization code, which identified the SBC750GX Ethernet ports by MAC address instead of IP address. As described in Section 3.2.2, the synchronization process involves the broadcast of a message from one channel to the others, and the execution of a clock adjustment procedure upon successful receipt of the message. The Wind River Workbench Debugger was used to control execution of this synchronization process and monitor the data exchanges. By stepping through the code, if control ever entered the part of the code dependent on the reception of the sync message from another channel, then we know the wireless transmission succeeded. With the wireless CCDL in place, we were successfully able to send and receive raw data packets between each pair of channels by carefully controlling when a channel would attempt to transmit a packet and when the other channels would start polling to receive the packet.

Although this method was useful in determining the ability of the channels in our system to communicate wirelessly without the use of IP, the actual complete synchronization process could not be executed. Using the Debugger we can carefully control when certain tasks are executed so we do not have to worry about timing windows, schedules, or how long it takes to deliver a message. In real-time however, in order to properly adjust its local clock, a channel needs to have an idea of what time to expect to receive a packet. That expected time is determined by the time it takes (on average) to deliver a message from one channel to another.

Since the bandwidth and throughput capabilities of a wireless 802.11g connection differ from those of a wired Ethernet connection, we cannot guess the same expected delivery time and thus we cannot directly apply the same frame synchronization protocol as used in SBRMS. In the future, if we can accurately quantify the delivery time of a packet going from one channel to another wirelessly, then the timing of the synchronization tasks and the partition schedule can be adjusted as necessary. Once synchronization is achieved, the remainder of the fault-tolerant functionality provided by the SBRMS, such as input and output voting, can be utilized.

This page intentionally left blank

Chapter 5

Conclusions

Wireless technology has progressed to the point where it can be considered as an option for communication in aerospace applications. It is becoming especially attractive due to the development and availability of low-cost wireless solutions such as the IEEE 802.11 standard. This thesis examined the possibility of reconfigurable fault-tolerant systems that use this type of low-cost wireless technology. We explored the benefits of using wireless data buses to achieve system reconfigurability and proposed and implemented a design of a three-channel system that utilizes wireless communication.

Building upon the component architecture of previously designed systems, basic tests were performed to successfully demonstrate the capability of 802.11g wireless communication in a system developed entirely with COTS hardware running a partitioned real-time operating system. However, more research needs to be done to find a wireless solution with higher reliability than the particular hardware chosen for this work. Also, testing needs to be done on the timing differences between the wired and wireless connections before the data exchanges between the three channels can be performed synchronously. Once we can synchronize the channels, then we can begin to implement voted data exchanges and the other fault detection, isolation, and recovery mechanisms required for a fully fault-tolerant system.

5.1 Future Work

5.1.1 Reconfigurability

If we can develop this work to the point where we have a synchronized TMR system with fault-tolerant data exchanges, then we will have the basic infrastructure in place to demonstrate wireless reconfigurability of the system. The methods for reconfiguration after a faulty channel is detected and removed from the rest of the system have already been established by the SBRMS implementation. Now, work needs to be done to be able to bring a new channel *into* the system. We can look to research done in wireless ad-hoc networking to determine the best way for channels to discover other channels and establish communication with each other. Once a new channel has been discovered, then all of the channels can appropriately update their data exchange protocols depending on the total number of channels in the system.

5.1.2 Wireless Technology

Although we have demonstrated the functionality of a wireless CCDL, the wireless connection itself is not fault-tolerant. The data exchange algorithms of this system need to be further developed in order to account for the higher rate of packet loss compared to the wired system. Features such as acknowledgement of received packets along with retransmission timers should be explored and incorporated into the system to ensure reliable wireless data transfer.

Another area to explore in order to reach the same performance level as the SBRMS design is the choice of wireless device. The Linksys WET54G bridges were selected because of their low cost and ease of integration into the existing system. However, there are other wireless technologies available that could greatly improve performance of the system in terms of throughput. Table 5-1 presents just a few possible wireless technologies that can be explored.

Technology	Transfer Speed	Range
802.11g	54 Mbit/s	30 m
802.11n	200 Mbit/s	50 m
WiMax 802.16	70 Mbit/s	4 mi
Wireless USB	110 Mbit/s	10 m
Bluetooth 3.0 (proposed)	480 Mbit/s	unknown

Table 5-1: Comparison of various wireless technologies

5.1.3 Ruggedized Hardware

The hardware components used for this thesis were selected for low-cost prototyping purposes in a lab environment, and thus they are commercial-grade products and are not ruggedized for a space environment. After the basic system gets further evaluated and more performance metrics are collected, we can look into the more expensive space-ready COTS hardware components.

This page intentionally left blank

Bibliography

[1] J. N. Yelverton, "Wireless Avionics," *Proc. 14th Digital Avionics Systems Conference*, pages 95-99, 1995.

[2] George Studor, "Fly-by-Wireless: A Revolution in Aerospace Vehicle Architecture for Instrumentation and Control," 2007.

[3] C. Elliot, B. Heile, "Self-Organizing, Self-Healing Wireless Networks," *Proc. International Conference on Personal Wireless Communications*, pages 355-362, 2000.

[4] T. X. Brown, B. Argrow, C. Dixon, S. Doshi, R. Thekkekunnel, D. Henkel, "Ad Hoc UAV Ground Network (AUGNet)," *Proc. AIAA 3rd Unmanned Unlimited Technical Conference*, pages 20-23, 2004.

[5] R. Racine, M. LeBlanc, and S. Beilin, "Design of a Fault-Tolerant Parallel Processor," *Proc. 21st Digital Avionics Systems Conference*, pages 13D2/1 - 13D2/10, 2002.

[6] R. Hammett, "Design by Extrapolation: An Evaluation of Fault Tolerant Avionics," *IEEE Aerospace and Electronic Systems Magazine*, vol. 17, no. 4, pp. 17-25, April 2002.

[7] J.H. Lala and R.E. Harper, "Architectural Principles for Safety-Critical Real-Time Applications," *Proc. IEEE*, vol. 82, no. 1, pages 25-40, Jan. 1994.

[8] R. Hammett, "Flight-Critical Distributed Systems: Design Considerations," *IEEE Aerospace and Electronic Systems Magazine*, 18(6): pages 30-36, June 2003.

[9] T. K. Srikanth and S. Toueg, "Optimal Clock Synchronization," *Journal of the ACM*, vol. 34, no. 3, pages 626-645, July 1987.

[10] F. Martin and D. Adams, "Cross Channel Dependency Requirements of the Multi-Path Redundant Avionics Suite," *Proc. Aerospace and Electronics Conference*, IEEE, pages 200 - 206, 1992.

[11] Lamport, L., Shostak, R., and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, pp. 382-401, July 1982.

[12] S. Beilin, "Maintaining Network Element Synchronization in the X-38 Fault-Tolerant Parallel Processor," Draper Laboratory, 2002.

[13] R. Harper, J. Lala, "Fault-Tolerant Parallel Processor," *Journal of Guidance, Control, and Dynamics*, vol. 14, no.3, pages 554-563, May-June 1991.