The Design of a Fault-Tolerant COTS-Based Bus Architecture

Savio N. Chau Leon Alkalai
John B. Burt
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Ann T. Tai
IA Tech, Inc.
10501 Kinnard Avenue
Los Angeles, CA 90024

April 15, 1999

Abstract

In this paper, we report our experiences and findings on the design of a fault-tolerant bus architecture comprised of two COTS buses, the IEEE 1394 and the I^2C. This fault-tolerant bus is the backbone system bus for the avionics architecture of the X2000 program at the Jet Propulsion Laboratory. COTS buses are attractive because of the availability of low cost commercial products. However, they are not specifically designed for highly reliable applications such as long-life deep-space missions. The X2000 design team has devised a multi-level fault tolerance approach to compensate for this shortcoming of COTS buses. First, the approach enhances the fault tolerance capabilities of the IEEE 1394 and I^2C buses by adding a layer of fault handling hardware and software. Second, algorithms are developed to enable the IEEE 1394 and the I^2C buses assist each other to isolate and recovery from faults. Third, the set of IEEE 1394 and I^2C buses is duplicated to further enhance system reliability. The X2000 design team has paid special attention to guarantee that all fault tolerance provisions will not cause the bus design to deviate from the commercial standard specifications. Otherwise, the economic attractiveness of using COTS will be diminished. The hardware and software design of the X2000 fault-tolerant bus are being implemented and flight hardware will be delivered to the ST4 and Europa Orbiter missions.

Keywords: COTS, IEEE 1394, I^2C, fault-tolerant bus architecture, space applications
Principal Contact: Savio N. Chau, Savio.N.Chau@jpl.nasa.gov

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

1 Introduction

In recent years, commercial-off-the-shelf (COTS) products have found many applications in space exploration. The attractiveness of COTS is that low cost hardware and software products are widely available in the commercial market. By using COTS through out the system, we expect to significantly reduce both the development cost as well as the recurring cost of the system. On the other hand, COTS are not specifically developed for highly reliable applications such as long-life deep-space missions. The real challenge is to deliver a low-cost, highly reliable and long-term survivable system based on COTS that are not developed with high-reliability in mind. In this paper, we report our experience of using COTS buses to implement a fault-tolerant avionics system for the Advanced Spacecraft System Development Program (also known as X2000) at the Jet Propulsion Laboratory. The X2000 avionics system design emphasizes on architectural flexibility and scalability, so that it can be reused for multi-missions in order to reduce the cost of space exploration [1]. The advanced avionics technologies that enable the X2000 program are being developed at the newly established Center for Integrated Space Microsystems, (CISM), a Center of Excellence at NASA's Jet Propulsion Laboratory [2]. The main focus of CISM is the development of highly integrated, reliable, and capable micro-avionics systems for deep space, long-term survivable, autonomous robotic missions [3][4]. In addition, the X2000 Program is participating in a software architecture development effort called the Mission Data System architecture (MDS), which brings within a common framework the software for both on-board avionics as well as on-ground operations.

The X2000 architecture is a distributed, symmetric system of multiple computing nodes and device drivers that share a common redundant bus architecture. Most notably, all interfaces used in this distributed architecture are based on COTS. That is, the local computer bus is the Peripheral Component Interface (PCI) bus; the "system" bus is the IEEE 1394 high-speed bus; and the engineering bus is the I^2C bus (see Figure 1). In the following sections, we first outline a methodology of applying COTS for highly reliable system. Based on this methodology, we describe the current baseline for the X2000 First

Delivery avionics system architecture. This is followed by how the X2000 Program is using COTS to implement a fault-tolerant bus for a scalable and distributed system.

2 A Methodology of Applying COTS for Highly Reliable System

JPL has a long history of successfully applying fault protection techniques in space exploration. One of the most important techniques used by JPL, in design of space vehicle fault protection, is fault containment. Traditionally, a spacecraft is divided into fault containment regions. Rigorous design effort is used to ensure no effects of a fault within a containment region will propagate to the other regions. Furthermore, JPL has a policy of single fault tolerance in most of the spacecraft design. This policy reqiures dual redundancy of fault containment regions.

While these techniques have been very successful in the past, they may not be easily applied in a COTS environment. The reason is that COTS are not developed with the same level of rigorous fault tolerance in mind. Hence, there are many fundamental fault tolerance weakness in COTS. For examples, the popular VME bus does not even have parity bit to check the data and address [11]. Another example is the IEEE 1394 bus (cable implementation) adopts a tree topology in which a single node or link failure will partition the bus. These fundamental weakness will hinder rigorous enforcement of fault containment. Worse yet, it is almost impossible to make modifications to COTS in general. There are two reasons. First, the suppliers of COTS products have no interest to change their design, add any overhead, or sacrifice their performance for a narrow market of high reliability applications. Second, any modification will render the COTS incompatible with commercial test equipment or software, and therefore diminish the economic benefits of COTS drastically. Therefore, it is obvious that fault tolerance cannot easily be achieved by a single layer of fault containment regions that contains COTS.

The COTS-based bus architecture of the X2000 has employed a multi-level fault protection methodology to achieve high reliability. The description of each fault protection level in the methodology is given as follows:

Level 1: Native Fault Containment – most of COTS bus standards have some limited fault detection capabilities. These capabilities should be exploited as the first line of defense.

Level 2: Enhanced Fault Containment – addition layer of hardware or software can be used to enhance the fault detection, isolation, and recovery capabilities of the native fault containment region. Examples are watchdog timer or additional layer of error checking code. It is important to ensure that the added fault tolerance mechanisms will not affect the basic COTS functions. This level is also the most convenient level to implement provisions for fault injections.

Level 3: Fault Protection by Component Level Design-Diversity – many COTS have fundamental fault tolerance weakness that cannot simply be removed by enhancing the native fault protection mechanisms. These weakness usually are related to single points of failures. One example is the tree topology of the IEEE 1394 bus. Once the bus is partitioned by a failed node, no watchdog timer or extra layer of protocol can reconnect the bus. Similar examples include buses using other point-to-point topologies. In order to compensate for such fundamental weaknesses, complementary types of buses may be used to implement this level of fault protection In particular, the I^2C bus, which has a multi-drop bus topology, is used in the X2000 architecture to complement the IEEE 1394 fault isolation and recovery.

Another example of design-diversity to compensate for COTS reliability in X2000 is the use of flash memory for the Non-Volatile Memory. The flash memory has the density required by X2000, but it has been observed that a single high energy particle can corrupt an entire block in the flash memory. To handle such failure mode with error correcting codes alone may not meet the reliability requirement. Therefore, in order to compensate for this weakness, a more robust but much lower density GMRAM or FeRAM is used to store critical state data in stead of the flash memory.

Level 4: Fault Protection by System Level Redundancy – the Level 3 fault containment regions will be replicated for system level fault containment. The redundant fault containment regions can be either in ready or dormant states, depending on the recovery time and other system requirments. If they are in ready state, voting or comparison of outputs among the regions will provide one more level of fault detection. In either case, the redundant regions are necessary resources for the fault recovery process.

In order to aid the discussion on how this methodolgy is applied to the X2000 bus architecture, an overview of the overall X2000 avionics system architecture and the common failure modes in space applications are given in the next section.

3 Overview of the X2000 Avionics Architecture

The X2000 avionics architecture is shown in Figure 1. It is comprised of a number of Compact PCI based "nodes" connected by a fault-tolerant system bus. A "node" can either be a flight computer, a non-volatile memory, a subsystem microcontroller, or a simple sensor interface. The fault-tolerant system bus is comprised of two COTS buses, the IEEE 1394 [5][6] and I^2C [7][8]. Both buses are multi-master and therefore support symmetric scalable and distributed architectures. Due to the standard electrical interface and protocol of the COTS buses, nodes complying with the bus interfaces can be added to or removed from the system without impacting the architecture. The capability of each node can also be enhanced by adding circuit boards to the its compact PCI bus [9]. An overview of the spacecraft functions that are handled by this architecture features are given in the following.

- Power management and distribution
- Science data storage and on-board science processing
- Telemetry collection, management and downlink spacecraft navigation and control
- Autonomous operations for on-board planning, scheduling, autonomous navigation fault-protection, isolation and recovery, etc.

- Interfacing to numerous device drivers: both "dumb" as well as "intelligent" device drivers

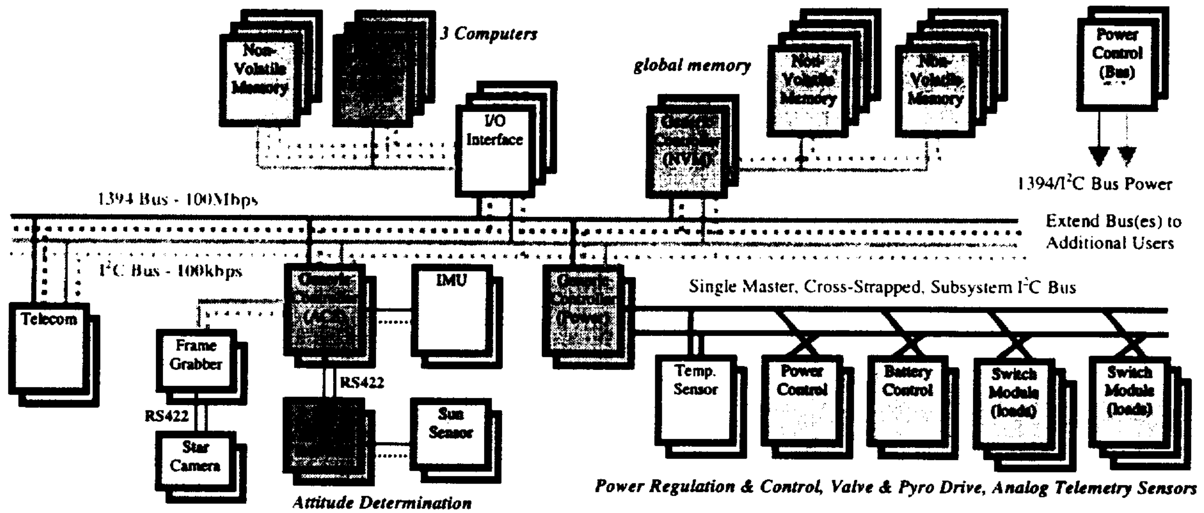

Figure 1: X2000 Avionics System Architecture

The IEEE 1394 Bus

The IEEE 1394 bus is the artery of the system and is capable to transfer data at 100, 200, or 400 Mbps. The IEEE 1394 bus has two implementations, cable and backplane. The cable implementation has adopted a tree topology and the backplane implementation has a multi-drop bus topology. From the topological point of view, many designers at JPL are more interested in the backplane implementation because it resembles the 1553 bus used in the Cassini project [13]. Unfortunately, it was found that although the backplane 1394 bus has been implemented by the aerospace industry [12], it is not widely supported in the commercial industry and thus will not be able to take the full advantage of COTS. On the other hand, the cable implementation has been enjoying a much wider commercial support. It has also better performance than the backplane implementation [5]. Therefore, the cable implementation has been selected for the X2000.

The IEEE 1394 bus has two modes of data transactions, the isochronous transaction and the asynchronous transactions. The isochronous transaction guarantees on-time delivery but does not require acknowledgment, while the asynchronous transaction requires acknowledgment but does not guarantee

on-time delivery. Isochronous messages are sent through "channels" and a node can talk on or listen to more than one isochronous channel. Each isochronous channel can request and will be allocated a portion of the bus bandwidth at the bus initialization. Once every 125 microseconds (called isochronous cycle), each isochronous channel has to arbitrate but is guaranteed a time slot to send out its isochronous messages. At the beginning of each isochronous cycle, the root sends out a cycle start message and then the isochronous transaction will follow. After the isochronous transaction is the asynchronous transaction. Asynchronous message is not guaranteed to be sent within an isochronous cycle. Therefore, a node may have to wait a number of isochronous cycles before its asynchronous massage can be sent out. The asynchronous transaction employs a fair arbitration scheme, which allows each node to send an asynchronous message only once in each fair arbitration cycle. A fair arbitration cycle can span over many isochronous cycles, depending on how much of each cycle is used up by the isochronous transactions and how many nodes are arbitrating for asynchronous transactions. The end of a fair arbitration cycle is signified by an Arbitration Reset Gap.

During the bus startup or reset, the bus will go through an initialization process in which each node will get a node ID. In addition, the root (cycle master), bus manager, and isochronous resource manager will be elected. The root mainly is responsible for sending the cycle start message and acts as the central arbitrator for bus requests. The bus manager is responsible to acquire and maintain the bus topology. The isochronous resource manager is responsible for allocating bus bandwidth to isochronous nodes. The root, bus manager, and isochronous resource manger are not pre-determined, so that any nodes can be elected to take these roles as long as they have the capability.

The I^2C Bus

The I^2C bus is a simple bus with a data rate of 100 kbps. It has a more traditional multi-drop topology. The I^2C bus has two open-collector signal lines: a data line (SDA) and a clock line (SCL). Both signal lines are normally pulled high. When a bus transaction begins, the SDA line is pulled down before the SCL line. This constitutes a start condition. Then the address bits will follow, which is

followed by a read/write bit and then an acknowledgment bit. The target node can acknowledge the receipt of the data by holding down the acknowledgment bit. After that, eight bits of data can be sent followed by another acknowledgment bit. Data can be sent repeatedly until a stop condition occurs, in which the source node signals the end of transaction by a low-to-high transition on the SDA line while holding the SCL line high.

The I^2C uses collision avoidance to resolve conflicts between master nodes contending for the bus. If two or more masters try to send data to the bus, the node producing a 'one' bit will lose arbitration to the node producing a 'zero' bit. The clock signals during arbitration are a synchronised combination of the clocks generated by the masters using the wired-AND connection to the SCL line.

There are two applications of the I^2C bus in this architecture. In the system level, it is used to assist the IEEE 1394 bus to isolate and recover from faults. In the subsystem level, a separate I^2C bus is used to collect engineering data from sensors and send commands to power switches or other equipment.

Description of Nodes

There are three basic types of nodes in the system: flight computer, microcontroller node, and non-volatile memory node. The flight computer node is consisted of a high-performance processor module (250 MIPS); 128 Mbytes of local (DRAM) memory; 128 Mbytes of non-volatile storage for boot-up software and other spacecraft state data; an I/O module for interfacing with the IEEE 1394 and I^2C buses. All modules communicate with each other via a 33 MHz PCI bus. The microcontroller node is very similar to the flight computer node except the microcontroller has lower performance and less memory to conserve power. The non-volatile memory node has four slices, each slice contains 256 Mbytes of flash memory and 1 Mbytes of GMRAM. The flash memory has much higher density and is suitable for block data storage. However, it has limited number of write cycles and is susceptible to radiation effects. The GMRAM has unlimited write cycles and is radiation tolerant, but its density is much lower than flash. In X2000, the flash memory is used for software codes and science data storage while the GMRAM is used

to store spacecraft state data. The non-volatile memory slices are controlled by a microcontroller with an IEEE 1394 and I^2C bus interfaces.

4 Design of the COTS Fault-Tolerant Bus

As it is mentioned that the COTS fault-tolerant bus is comprised of the IEEE 1394 and the I^2C buses. A very detail trade study was conducted at the beginning of the X2000 First Delivery project to select the buses. At the end, the IEEE 1394 bus was selected because of its high data rate (100, 200 or 400 Mbps), multi-master capability, moderate power consumption, strong commercial support, relatively deterministic latency, and the availability of commercial ASIC cores (referred to as Intellectual Properties or IPs in industry). The advantages of IPs are that they are reusable and can be integrated in ASICs and fabricated by rad-hard foundry to meet radiation requirements. The I^2C bus was selected because of its very low power consumption, multi-master capability, availability of ASIC IPs, adequate data rate (100 kbps) for low speed data, simple protocol, and strong commercial support. APL has even developed a rad-hard I2C based sensor interface chip.

Although the IEEE 1394 and I^2C buses are very attractive in many aspects, it was recognized early in the design activity that they are not ideal buses in the classical fault tolerance sense. The 1394 bus has limited fault detection features, and has no explicit fault recovery mechanisms such as built-in redundancy or cross strapping. In particular, the 1394 bus has a tree topology that can easily be partitioned by a single node or link failure. The I^2C bus has almost no built-in fault detection except an acknowledgement bit after every byte transfer. However, they are preferred over the other fault-tolerant buses mainly because of their low cost and commercial support. To effectively manage the trade-offs is the characteristic of our approach to using COTS for highly reliable systems; and the techniques to compensate for their weakness in fault tolerance is the main focus of this paper.

4.1 Failure Modes in Data Bus of Spacecraft Avionics Systems

At this point, it is worthwhile to identify the most common or critical failure modes for data buses in spacecraft avionics systems, which are the targets of the fault tolerance techniques described in this paper. NASA/JPL always performs failure mode effect and criticality analysis (FMECA) for every spacecraft design. Based on those experiences, the following failure modes for data buses in avionics systems have been identified as either frequently occur or critical to the survival of the spacecraft.

1. *Invalid Messages*: Messages sent across the bus contains invalid data.
2. *Non-Responsive*: An expected response to a message does not return in time.
3. *Babbling*: Communication among nodes is blocked or interrupted by uncontrolled data stream.
4. *Collision*: More than one node has the same identification.

4.2 Overall Strategy of COTS Fault-Tolerant Bus Design

The overall strategy of the COTS fault-tolerant bus design applies the methodology mentioned in Section 2 as follows.

Step 1: Native Fault Containment – The basic fault detection mechanisms of the IEEE 1394 and I^2C buses such as CRC and acknowledgment are used to detect invalid messages or non-responsive failure modes.

Step 2: Enhanced Fault Containment – A layer of hardware and software is added to enhance the fault detection and recovery capability of the IEEE 1394 and I^2C buses. The extra layer of hardware is added to the ASIC containing the IPs of the IEEE 1394 and I^2C buses.

Step 3: Fault Protection by Design Diversity – Since the IEEE 1394 bus adopts a tree topology, it is very difficult to isolate or recover from a failed node or link because the bus network is partitioned and communication between the sub-trees is cut off. The I^2C bus is used to assist the fault isolation and recovery by maintaining the communication of all nodes. Similarly, if the shared medium of the I^2C bus fails, the 1394 bus can be used to assist the isolation and recovery of the I^2C bus.

Step 4: Fault Protection by System Level Redundancy – The set of COTS buses (i.e., IEEE 1394 and I^2C bus set) is duplicated. If the primary COTS bus set has failed due to a single-point-failure between the IEEE 1394 and I^2C buses, the backup COTS bus set will be activated to assist the fault isolation and recovery of the primary bus set. The I^2C bus in the backup COTS bus set will be activated first since it is much easier to initialized. Also the backup I^2C bus can provide communication among the nodes to facilitates the isolation of the single-point-failure in the primary bus set. If the single-point-failure can be isolated and removed by bus reconfiguration, then the primary bus set will resume normal operation. Otherwise, the backup IEEE 1394 bus will be activated. With both the backup IEEE 1394 and I^2C buses activated, the backup bus set can replace the primary bus set.

In some circumstances, only the IEEE 1394 bus in the primary bus set can be recovered and the I^2C bus is still affected by the single-point-failure (i.e., due to the multi-drop topology of the I^2C bus). The backup I^2C bus and the primary IEEE 1394 bus can work together as a single bus set.

In rare occasions, the backup I^2C bus also fails while trying to assist the primary bus set to isolate the single-point-failure. The backup IEEE 1394 bus can be activated to assist the fault isolation and recovery of the primary bus set as well as handling the workload.

4.2 Native Fault Containment Regions

The basic fault detection mechanisms of the IEEE 1394 and I^2C buses are highlighted in this section.

4.2.1 Highlights of the IEEE 1394 Bus Fault Tolerance Mechanisms

The 1394 bus standard has many built-in fault detection mechanisms. A summary of these mechanisms is given below. The details of the acknowledgment and response packet error codes can be found in [5] and [6].

1. Data CRC and packet header CRC for both isochronous and asynchronous transactions
2. Acknowledgment packets include error code to indicate if the message has been successfully delivered in asynchronous transactions
3. Parity bit to protect acknowledgment packets
4. Response Packets include error code to indicate if the requested action has been completed successfully in asynchronous transactions
5. Built-in timeout conditions: response timeout for split transaction, arbitration timeout, acknowledgment timeout etc.

A very import feature in the latest version of the IEEE 1394 standard (IEEE 1394a [10]) is the capability to enable or disable individual ports (a port is the physical interface to a link). With this feature, every node in the bus can disable a link connected to a failed node and enable a backup link to bypass the failed node. This feature is the basis of the IEEE 1394 bus recovery in this bus architecture.

Another feature in the IEEE 1394 standard is the keep-alive of the physical layer with cable power. This feature allows the link layer hardware and the host processor to be powered off without affecting the capability of the physical layer to pass on messages. This is useful for insolating a failed processor during fault recovery.

4.2.2 Highlights of the I^2C Bus Fault Detection Mechanisms

The only fault detection mechanism is the acknowledgment bit that follows every data byte. When a node (master) sends data to another node (slave), and if the slave node is able to receive the data, it has

to acknowledge the transaction by pulling the data line (SDA) to low. If the slave node fails to acknowledge, the master node will issue a stop condition to abort the transaction. Similar situation can happen when the master node requests data from a slave node. If the master fails to acknowledge after receiving data from the slave, the slave will stop sending data. Subsequently, the master node can issue a stop condition to terminate the transaction if it is still functional.

4.3 Enhanced Fault Containment Regions

Several mechanisms are added to enhance the fault detection and recovery capability of the IEEE 1394 and I^2C buses. They are discribed in the following.

4.3.1 Enhanced Fault Tolerance Mechanisms for IEEE 1394 Bus

Heartbeat and Polling

The X2000 architectural design enhances the fault detection capability of the 1394 bus with heartbeat and polling. Heartbeat is effective for detecting root failure while polling can be used to detect individual node failures. Since the cycle master (root) of the 1394 bus always sends out an isochronous cycle start message every 125 μs (average), it is natural to use the cycle start message as the heartbeat. All other nodes on the bus monitor the interval between cycle start messages. If the root node fails, other nodes on the bus will detect missing cycle start and initiate fault isolation process (to be discussed in later sections). However, cycle start can only detect hardware level faults since it is automatically generated by the link layer. Therefore, a software heartbeat should be used to detect faults in the transaction or application layers.

Other failure modes can also be detected by this method. For example, multiple roots will generate more than one hardware heartbeat (i.e., cycle start) within an isochronous cycle. By comparing the actual heartbeat interval with a minimum expected heartbeat interval, the multiple heartbeats can be detected. More discussions about the multiple root detection can be found in the next two sections.

Furthermore, software heartbeat is effective in detecting babbling nodes. If the fault causing the node to babble is in software, it is possible that the hardware heartbeat may appear to be valid since the cycle start is automatically generated by the link layer hardware. On the other hand, the software fault is likely to affect the software heartbeat. Therefore, the software heartbeat is preferred over the hardware heartbeat in detecting babbling nodes.

In addition to heartbeat, the root node can also send polling messages periodically to individual nodes by asynchronous transaction. Since asynchronous transaction requires acknowledgment from the target node, a node failure can be detected by acknowledgment timeout.

Isochronous Acknowledgment

Sometimes, acknowledgment is desirable for isochronous transactions, especially when the isochronous transaction requires on-time and reliable delivery. Therefore, a confirmation message type is added to the application layer, so that the target node can report any isochronous transaction errors to the source node. The confirmation message itself can be either an isochronous or asynchronous transaction, depending on the time criticality. Furthermore, the data field of the original isochronous message contains the source node ID, so the target node knows where to report the isochronous transaction errors. If the confirmation message contains an error code, the source node can retransmit the message in isochronous or asynchronous mode as appropriate.

Link Layer Fail-Silence

The root node of the IEEE 1394 bus periodically sends a "fail silence" message to all nodes; every node in the bus has a *fail silence timer* in the link layer to monitor this message. Upon receiving the message, each node will reset its *fail silence timer*. If one of the nodes babbles because of a link layer or application layer failure, the fail silence message will be blocked or corrupted. This will cause the *fail silence timer* in each node to time out. Subsequently, the *fail silence timer* will disable the hardware of its own link layer and thus inhibit the node from transmitting or receiving messages (note: the ability of the

physical layer to pass on message is unaffected). Eventually, after a waiting period, the link layers of all nodes including the babbling node will be disabled and the bus will become quiet again. At this time, another timer in the root will "unmute" the root itself and send a *Link-on* packet, which is a physical layer packet, to individual nodes. Upon receiving the *Link-on* packet, the physical layer of a node will send a signal to wake up its link layer. If a node causes the bus to fail again while its link layer is re-enabled, it will be identified as the failed node and will not be enabled again. If the root itself is the babbling node, other nodes will detect the unmute timeout and issue bus reset.

Watchdog Timers

The IEEE 1394 standard has specified many watchdog timers. Additional watchdog timers that are related to fault detection of the IEEE 1394 bus have been identified as follows.

CPU Watchdog Timer: A hardware timer to monitor the health of the host CPU (i.e., the microprocessor or microcontroller). This watchdog timer is an incremental counter and need to be reset by the CPU periodically. If the CPU fails to reset this watchdog, an overflow will occur which then will trigger a local reset.

Heartbeat Lost Timer (HLT): Triggered by lost of heartbeat or CPU. It times out after a programmable value is decremented to zero.

Poll Response Timer (in Root Node): A software timer monitor the response time of polling message on the 1394 bus.

4.3.2 Enhanced Fault Tolerance Mechanisms for I^2C Bus

Protocol Enhancement

A layer of protocol is added to the I^2C bus. This protocol includes a byte count after the address and two CRC bytes after the data. X2000 design also utilizes especial hardware messages commands to control critical functions. For these messages, command is sent followed by its complement to provide one more layer of protection.

Byte Timeout

The I^2C bus permits a receiving node (slave or master) to hold down the clock signal (SCL) as a means to slow down the sending node (master or slave). This is to allow a fast node to send data to a slow node. However, it is possible that a failed receiving node causes a stuck-at-low fault on the SCL signal, so that the sending node may have to wait indefinitely. To recover from this failure mode, every node has a *byte timeout* timer to monitor the duration of the SCL signal. When the *byte timeout* timer in a node (including the faulty node) expires, it will disable the circuitry of the SDA and SCL transmitters. After all nodes have disabled their SDA and SDL transmitters, a recovery procedure similar to that in the fail-silence mechanism (see next) will be used to disable the failed node.

Fail Silence

One of the nodes in the I^2C is designated as the controlling master. The controlling master periodically sends a "fail silence" message to all I^2C nodes. All nodes will monitor this message with an *I^2C bus fail silence timer*. Upon receiving the message, each node will reset its *I^2C bus fail silence timer*. If one of the nodes is babbling so that the fail silence message is blocked or delayed, the *I^2C bus fail-silence timer* of each node will time out. Subsequently, the bus transmitters of each node will be disabled to inhibit any transmission of messages. However, the bus receiver of each node is still enabled so that it can receive commands for fault recovery later on. After a waiting period, the bus transmitters of all nodes including the babbling node will be disabled and the bus will be quiet again. At this time, another timer

in the controlling master node will "unmute" the node itself and send a message to re-enable the other nodes individually. If a node causes the bus to fail again while it is enabled, it will be identified as the failed node and will not be enabled again. If the root itself is the failed node, other backup nodes will detect the unmute timeout and promote themselves as the controlling master according to a pre-determined priority.

4.4 Fault Protection by Design Diversity

By working together, the IEEE 1394 and I^2C buses can isolate and recover from many faults that might not be possible if each bus is working alone. The failure modes that can be handled by the coorpration of the buses are as follows.

Non-Responsive Failures

In the IEEE 1394 bus, when a node or one of its links fails in the non-responsive mode, it will not be able to respond to requests and messages will not be able to pass through the node. The existence of the failure can easily be detected by the bus timeout, message re-transmission, heartbeat, or polling. In general, the failed node is relatively easy to isolate because all the nodes in the sub-tree under it will become non-responsive to the requests from the root node. Therefore, the prime suspect is usually the non-responsive node nearest to the root. However, to recover from the fault is not trivial because the tree topology of the bus has been partitioned in to two or three segments by the failed node. The nodes in each segment will not be able to communicate with the nodes in the other segments. Consequently, the root node will not be able to command the nodes in the other segments to change bus topology. It might be possible to devise distributed algorithms so that each node can try different link configurations to re-establish the connectivity. However, these algorithms usually are rather complicate and their effectiveness is difficult to prove.

Under these circumstances, the I^2C bus can facilitate the communication among all the nodes. The root node will first interrogate the health of the nearest non-responsive node (i.e., the prime suspect)

through the I^2C bus. If the node does not respond or if its response over the I^2C bus indicates any internal or physical connection failures, then the root node can send I^2C messages to the other nodes and command them to reconfigure their links to bypass the failed node. If the prime suspect node is fault-free, then the root can repeat the interrogation (and recovery procedure) on the other nodes in separate segments.

Similarly, if a node in the I^2C bus becomes non-responsive, the source node can interrogate the health of the target node through the IEEE 1394 bus, command the target node to reset its I^2C bus interface, and request the target node to retransmit the message.

IEEE 1394 Bus Physical Layer Babbling

The fail-silence technique is effective to handle babbling failures in the I^2C bus and in the link or application layers in the IEEE 1394 bus. However, the physical layer of the IEEE 1394 bus is rather complicate and contains state machines, it is possible that a transient fault would cause it to babble. A particular dangerous type of babbling is the continuous reset because any node in the IEEE 1394 bus is able to issue bus reset. Such failures cannot be handled by fail-silence. It is because if the physical layer is silenced, it will not be able to pass on messages and thus cause bus partitioning. In this case, each node can check its own physcial layer (e.g., read the physical layer registers). If the physical layer is faulty, the processor of the node can issue a physical layer reset to correct the problem. However, if the physical layer fault is permanent, then the node has to inform the root node via the I^2C bus. Subsequently, the root node can command other nodes via the I^2C bus to reconfigure the bus topology to bypass the failed node.

Collision of Node Addresses

The address of any node in the IEEE 1394 or I^2C buses can be corrupted by permanent fault or single event upset. If the faulty address coincides with an existing node address, any read transaction to that address will be corrupted by bus conflict from the two nodes, and any write transaction will go to both nodes and may have unpredictable consequences. Hence, it is difficult to disable the fault node by

the bus itself alone. However, with the redundant IEEE 1394/I^{2}C bus set, this kind of failures can be handled through using one bus to disable a faulty node on the other bus, so that the erroneously duplicated node address can be eliminated.

4.5 Fault Protection by System Level Redundancy

The COTS bus set is duplicated to provide system level of fault protection. The procedure of using the redundant COTS bus set to handle faults has been explained in Section 4.2.

To further enhance the effectiveness of the system level redundancy, the IEEE 1394 bus in the backup COTS bus set connects the nodes in such a way that any branch node in primary bus set is a leaf node in the backup bus set and vice versa. In other words, there is no node that is a branch node for both buses. This is shown in Figure 2. Hence, a failed node can only partition the bus in which it is a branch node. For the other bus, the failure only represents the loss of a leaf node, but the main body of the tree structure is not affected.

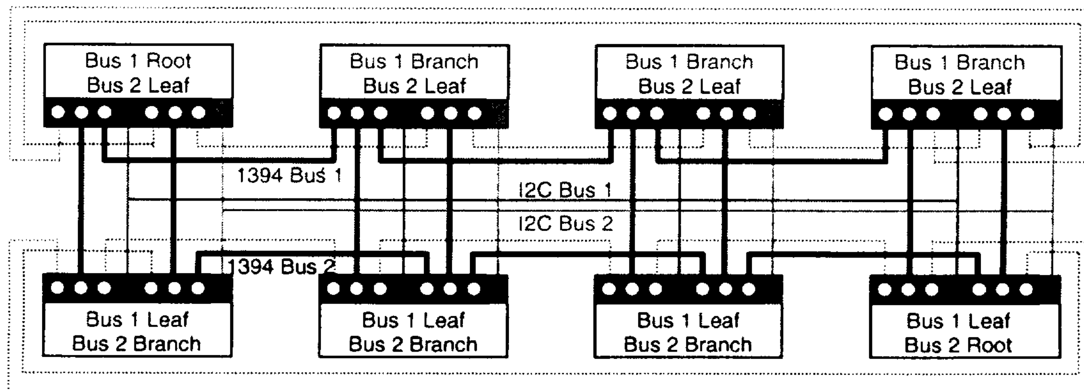

Figure 2: Configuration Diversity of Redundant IEEE 1394 Bus

4.6 Fault Recovery under Catastrophic Failure Condition

Under catastrophic failure conditions such as bus power failure, both COTS bus sets may fail such that all communications among the nodes are lost. To re-establish the communication, each node can execute a distributed recovery procedure that consists of a sequence of link enable/disable activities. The enabled links of all the nodes in each step of the procedure forms a bus configuration. If the critical

nodes of the system can communicate with each other in one of the bus configurations, further fault recovery procedures can follow. Unfortunately, this approach requires reasonably tight synchronization among all the nodes, which is very difficult to achieve when all bus communications are lost. Furthermore, since the cause of the catastrophic failure may not be within the avionics system, there is no guarantee that the distributed recovery procedure will succeed. Therefore, this approach is only the last recourse to save the spacecraft.

5 Summary and Future Work

We have desribed our approach to using COTS in highly reliable systems. Our methodology calls for a multi-level fault protection techniques. The methodology realizes that COTS are not developed with high reliability in mind. Nevertheless, by using multi-level fault protection, the same level of reliability as the traditional full-custom fault tolerance approach can be achieved. In fact, this methodology allows more freedom for design trade-off among the fault protection levels. This can result in less complicate designs than the traditional strictly enforced fault containment policy. Examples of how the methodoloy is realized in the X2000 bus architecture are also given. The design techniques in each levels of fault protection will be verified by fault injection under various fault scenarios. The effectiveness of the multi-level fault protection methodology will be measured by its fault coverage. The distributed algorithm to handle catastrophic failures will also be refined in the near future.

Reference:

[1] L. Alkalai and A. T. Tai, "Long-life deep-space applications," *IEEE Computer*, vol. 31, pp. 37–38, Apr. 1998.

[2] L. Alkalai, "NASA Center for Integrated Space Microsystems," in *Proceedings of Advanced Deep Space System Development Program Workshop on Advanced Spacecraft Technologies*, (Pasadena, CA), Jun. 1997.

[3] L. Alkalai, "A roadmap for space microelectronics technology into the New Millennium," in *Proceedings of the 35th Space Congress*, (Cocoa Beach, FL), Apr. 1998.

[4] L. Alkalai and S. N. Chau, "Description of X2000 Avionics Program," in *Proceedings of the 3rd DARPA Fault-Tolerant Computing Workshop*, (Pasadena, CA), Jun. 1998.

[5] IEEE 1394, *Standard for a High Performance Serial Bus*. Institute of Electrical and Electronic Engineers, Jan. 1995.

[6] D. Anderson, *FireWire System Architecture, IEEE 1394*. PC System Architecture Series, MA: Addison Wesley, 1998.

[7] D. Paret and C. Fenger, *The I^2C Bus: From Theory to Practice*. John Wiley, 1997.

[8] Philips Semiconductor, *The I^2C-Bus Specification Version 2.0*, Philips Semiconductor, Dec. 1998.

[9] T. Shanley and D. Anderson, *PCI System Architecture*, Addison Wesley, 1995.

[10] IEEE P1394A, *Standard for a High Performance Serial Bus (Supplement)*, Draft 2.0. Institute of Electrical and Electronic Engineers, Mar. 1998.

[11] W. Peterson, "The VMEbus Handbook: Expanded Third Edition,", VFEA International Trade Association, 1993

[12] J. Marshall, "Building Standard Based COTS Multiprocessor Computer Systems for Space Around a High Speed Serial Bus Network," in the *Proceedings of the 17th Digital Avionics Systems Conference*, (Bellevue, Washington), 1998.

[13] J. Donaldson, "Cassini Orbiter Functional Requirements Book: Command and Data Subsystem," *JPL Document CAS-4-2006*, June 28, 1994