Design of a Fault-Tolerant COTS-Based Bus Architecture

Savio N. Chau
Jet Propulsion Laboratory, Pasadena

Leon Alkalai, *Member IEEE*
Jet Propulsion Laboratory, Pasadena

Ann T. Tai, *Member IEEE*
IA Tech Inc., Los Angeles

John B. Burt
Jet Propulsion Laboratory, Pasadena

Key Words — COTS, IEEE 1394 Bus, I^2C bus, Long-life deep space mission, Bus network reliability, Fault-tolerant bus architecture.

Summary & Conclusions — This paper describes our approach to using commercial-off-the-shelf (COTS) products in highly reliable systems. The methodology calls for multi-level fault-protection. The methodology realizes that COTS products are often 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. This methodology allows more freedom for design trade-offs among the fault-protection levels, which can result in less complicated designs than the traditional strictly-enforced fault-containment policy. This paper covers our experiences & findings on the design of a fault-tolerant avionics bus architecture comprised of two COTS buses, the IEEE 1394, and the I^2C, for the avionics system of X2000 program at the Jet Propulsion Laboratory. The X2000 design is judicious about ensuring the fault-tolerance provisions do not cause the bus design to deviate from commercial standard specifications, so that the economic attractiveness of using COTS is preserved. The hardware & software designs of the X2000 fault-tolerant bus are being implemented, and flight hardware will be delivered to the Europa Orbiter missions. This work provides an example of how to construct a highly reliable system with low-cost COTS interfaces.

1. INTRODUCTION

Acronyms

COTS	commercial off-the-shelf
I^2C Bus	Inter integrated-circuit bus
JPL	Jet Propulsion Laboratory
RAM	random-access memory
GMRAM	giant magnetoresistive RAM
FeRAM	ferro electric RAM

In recent years, COTS products have found many applications in space exploration. The attractiveness of COTS is that low-cost hardware & software products are widely available in the commercial market. By using COTS through-out the system, one can appreciably reduce both the development cost, the recurring cost, and most importantly, the integration-and-test/equipment 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, long-term survivable system based on COTS that are not developed with high-reliability in mind. This paper reports our experience of using COTS buses to implement a fault-tolerant avionics system for the Deep Space System Technology Program (also known as X2000) at the JPL. The X2000 avionics system design emphasizes architectural flexibility & scalability, so that it can be reused for multi-missions, 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 the NASA JPL [2]. The main focus of CISM is the development of highly integrated, reliable, capable micro-avionics systems for deep space, long-term survivable, autonomous robotic missions [3, 4]. The X2000 Program is also participating in the software-architecture development 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 shown in figure 1 is a distributed, symmetric system of multiple computing-nodes and device-controllers that share a common redundant bus architecture. Most notably, all interfaces used in this distributed architecture are based on COTS:

- the local computer bus is the Peripheral Component Interface (PCI) bus;
- the system-buses are the IEEE 1394 bus and the I^2C bus.

Within each node, there is also a separate subsystem I^2C bus for sensors and instruments control (see figure 1). This paper focuses on the architecture of the system bus.

Section 2 outlines a methodology of applying COTS for highly reliable system. Based on this methodology, section 3 presents the current baseline for the X2000 First Delivery avionics system architecture. Section 4 describes how the X2000 Program uses COTS to implement a fault-tolerant bus architecture for a scalable and distributed system. Section 5 reports the status of implementation of this bus architecture.

The hardware & software development of the COTS-based fault-tolerant bus architecture is underway at JPL. A model-based quantitative evaluation [14] shows that, for an 11-year mission, the reliability of a 32-node instance of the COTS-based fault-tolerant bus architecture can remain greater than 0.9999 at the end of mission. In comparison, the reliability of a non fault-tolerant COTS-based bus of the same size is less than 0.86 at the end of a mission with the same duration.

2. USING 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 is used to ensure that no 'effects of a fault within a containment region' can propagate to the other regions. JPL has a policy of single fault tolerance in most of the spacecraft design. This policy requires dual redundancy of fault containment regions.

While these techniques have been very successful, they cannot be easily applied in a COTS environment, because 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 example,

- the popular VME backplane bus does not even have parity bit to check the data & address [11];
- IEEE 1394 bus (cable implementation) adopts a tree topology in which a 'single node or link failure' partitions the bus.

These fundamental weakness hinder rigorous enforcement of fault containment. Worse yet, it is very difficult to modify COTS because:

1. The suppliers of COTS products have no interest in changing their design, add any overhead, or sacrifice their performance for a narrow market of high reliability applications.
2. Any modification renders the COTS incompatible with commercial test equipment or software, and therefore drastically diminishes the economic benefits of COTS.

Therefore, 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 avionics system has used multi-level fault protection to achieve high reliability; its 4 levels are:

Level 1: Native Fault Protection

Most COTS bus standards have some limited fault detection capabilities. These capabilities should be exploited as the first line of defense.

Level 2: Enhanced Fault Protection

An additional 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 heartbeats, watchdog timer, and additional layer of error checking code. It is important to ensure that the added fault tolerance mechanisms do not affect the basic COTS functions. This level is the most convenient one to implement provisions for fault injections.

Level 3: Fault Protection by 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 failure. One example is the tree topology of the IEEE 1394 bus. Once the bus is partitioned by a failed node, the nodes in different segments cannot communicate with each other to coordinate a recovery effort. This is a fundamental problem of the bus topology which cannot be solved by the enhanced fault-partition techniques. To compensate for such fundamental weaknesses, a different bus with different topology must be used to provide a communication path among the nodes under those fault conditions, so that they can coordinate the fault isolation & recovery. In particular, the I^2C bus, which has a multi-drop bus topology, is used in the X2000 architecture to assist the IEEE 1394 fault isolation & recovery.

For buses using tree technologies, it is necessary to add backup connections in the bus to tolerate failed nodes or links. In X2000, backup connections are added to the IEEE 1394 bus (figure 3). These connections are usually disabled to avoid loops, which are prohibited in the IEEE 1394 Standard. The backup connections can be selectively enabled during fault recovery.

Level 4: Fault Protection by System Level Redundancy

The X2000 avionics system architecture is symmetric and thus provides inherent redundancy. In addition, the IEEE 1394/I^2C bus set is replicated for system level fault containment. The redundant bus set is in either ready or dormant states, depending on the recovery time and other system requirements. In either case, the redundant bus set is a necessary resource for the fault recovery process.

3. OVERVIEW OF THE X2000 AVIONICS ARCHITECTURE

Figure 1 shows the X2000 avionics architecture. It is comprised of multiple Compact PCI based nodes connected by a fault-tolerant system bus. A node can be either:

- a flight computer,
- a global non-volatile mass memory,
- a subsystem microcontroller, or

· a science instrument.

The fault-tolerant system bus is comprised of two COTS buses:

· IEEE 1394 [5, 6],

· 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 be enhanced by adding circuit boards to its compact PCI bus [9]. The spacecraft functions that are handled by the X2000 architecture are:

· Spacecraft command and data handling,

· Telemetry collection, management and downlink spacecraft navigation and control,

· Science data storage and on-board science processing,

· Power management and distribution,

· Autonomous operations for on-board planning, scheduling, autonomous navigation fault-protection, isolation and recovery, *etc*,

· Interfacing to numerous device drivers: both dumb and intelligent device drivers.

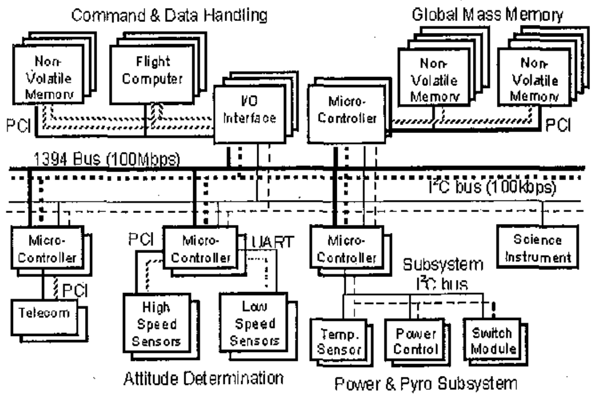

Figure 1: X2000 Avionics System Architecture

3.1 The IEEE 1394 Bus

The IEEE 1394 bus is the artery of the system, and can transfer data at 100, 200, or 400 Mbps. The X2000 First Delivery Project implements only the 100 Mbps data rate. The IEEE 1394 bus has two kinds of implementations: cable and backplane. The cable implementation has adopted a tree topology; the backplane implementation has a multi-drop bus topology. From the topological viewpoint, many designers at JPL are more interested in the backplane implementation because it resembles the 1553 bus used in the Cassini project [13]. Unfortunately, although products of the backplane 1394 bus are available [12], it is not widely supported in the commercial industry, and thus cannot take full advantage of COTS. On the other hand, the cable implementation has been enjoying a much wider commercial support. It has better performance than the backplane implementation [5]. Therefore, the cable implementation has been selected for the X2000.

The IEEE 1394 bus has 3 layers of protocol,

· physical,

· link,

· transactions,

and supports 2 modes of data transaction,

· isochronous: guarantees on-time delivery but does not require acknowledgment;

· asynchronous: 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 node can request, and is allocated, a portion of the bus bandwidth at the bus initialization. Once every 125 μsec (called isochronous cycle), each isochronous node has to arbitrate, but is guaranteed a time slot, to send out its isochronous messages if it has been allocated a portion of the bus bandwidth. At the beginning of each isochronous cycle, the root sends out a cycle-start message and then the isochronous transaction follows. After the isochronous transaction is the asynchronous transaction.

Asynchronous messages are not guaranteed to be sent within an isochronous cycle. Therefore, a node might have to wait several isochronous cycles before its asynchronous massage can be sent. The asynchronous transaction uses 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 many isochronous cycles, depending on,

· how much of each cycle is used by the isochronous transactions,

· 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 goes through an initialization process in which each node gets a node ID. The root (cycle master), bus manager, and isochronous resource manager are 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 for acquiring & maintaining 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; thus any nodes can be elected to take these roles if they have the capability.

3.2 The I^2C Bus

The I^2C bus, developed by the Philips Semiconductor [8], is a simple bus with a data rate of 100 kbps. It has a more traditional multi-drop topology. The I^2C bus has 2 open-collector signal lines:

· a data line (SDA),
· 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 follow; they are followed by a read/write bit. The target node can acknowledge the receipt of the data by holding down the SDA line in the next clock (called acknowledgment bit). After that, 8 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 1-bit loses arbitration to the node producing a 0-bit. The clock signals during arbitration are a synchronized 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,
· system level: the bus is used to assist the IEEE 1394 bus to isolate and recover from faults;
· subsystem level: a separate I^2C bus is used to collect engineering data from sensors, and to send commands to power switches or other equipment.

3.3 Description of Nodes

There are three basic types of nodes in the system:
· flight computer node,
· microcontroller node,
· non-volatile memory node.

The flight computer node consists of a high-performance Power PC 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 that, to conserve power, the microcontroller has lower performance and less memory. It is used to interface sensors & instruments with the IEEE 1394 and I^2C bus.

The non-volatile memory node has 4 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 a 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 in the non-volatile memory node are controlled by a microcontroller with the IEEE 1394 and I^2C bus interfaces.

The complete system is housed in a standard compact PCI backplane chassis with 3U boards.

4. DESIGN OF COTS FAULT-TOLERANT BUS

The COTS fault-tolerant bus architecture is comprised of the IEEE 1394 and the I^2C buses. A very detailed tradeoff study was conducted at the beginning of the X2000 First Delivery project to select the buses [16]. 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,
· availability of commercial ASIC cores (referred to as Intellectual Properties or IPs in industry).

The advantages of commercial IPs are that they are reusable and can be integrated in ASIC 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,
· strong commercial support.
The Applied Physics Laboratory (APL) of the Johns Hopkins University has developed a rad-hard I^2C based sensor interface chip called Temperature Remote I/O (TRIO) for the X2000 First Delivery Project.

Although the IEEE 1394 and I^2C buses are very attractive in many aspects, they are not ideal buses in the classical fault-tolerance sense. The 1394 bus has limited fault-detection capabilities, 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 acknowledgment bit after every byte transfer. However, it is selected mainly because of their low cost and commercial support. Managing the tradeoffs effectively is the characteristic of our approach to using COTS for highly reliable systems; the techniques to compensate for their weakness in fault tolerance is the main focus of this research.

4.1 Failure Modes in the Data Bus of Spacecraft Avionics Systems

The most common or critical failure modes for data buses in spacecraft avionics systems 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 occurring' or 'critical to the survival of the spacecraft'.

- Invalid Messages: Messages sent across the bus contain invalid data.
- Non-Responsive: An anticipated response to a message does not occur or return in time.
- Babbling: Communication among nodes is blocked or interrupted by uncontrolled data stream.
- Conflict of Node Address: More than one node has the same identification.

4.2 Overall Strategy of the COTS Fault-Tolerant Bus Design

Applying the methodology in section 2, the overall strategy of the COTS fault-tolerant bus design can be established, and is shown in figure 2. The strategy first uses the native fault-tolerance features of the IEEE 1394 and I^2C buses to detect fault-occurrences. An additional layer of hardware & software fault-tolerance enhances the fault detection & recovery capability of each bus. Then the IEEE 1394 and I^2C buses assist each other to isolate & recover from difficult faults. The entire set of IEEE 1394 and I^2C buses are duplicated at the system level to provide necessary redundancy for fault recovery.

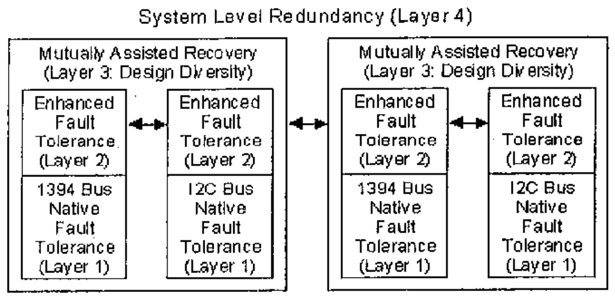
System Level Redundancy (Layer 4)
Mutually Assisted Recovery (Layer 3: Design Diversity)
Enhanced Fault Tolerance (Layer 2)
1394 Bus Native Fault Tolerance (Layer 1)
I2C Bus Native Fault Tolerance (Layer 1)

Figure 2: COTS Bus Architecture Fault-Tolerance Strategy

When a fault occurs in the primary bus set, it is detected by either the native or enhanced fault protection (layer 1 & 2). Simple recovery procedures such as retry and bus reset are first attempted. If the simple procedures cannot correct the problem, then the backup set of buses is activated and the system operations are transferred to the backup bus (layer 4)[1]. At this point, the system can have more time to diagnose the failed bus-set. If the IEEE 1394 bus fails, the I^2C bus is used to diagnose it (layer 3). After the faulty node or connection is identified, the system removes it from the bus topology by disabling the connections attached to the node, and enable selected backup connections around it. Similarly, if the I^2C bus fails, the IEEE 1394 bus can be used to diagnose it (layer 3). If a faulty I^2C bus interface is found, the system can disable the bus interface by commanding the node to shut off its I2C bus transmitter. The repaired bus-set becomes the backup. Implementation of this bus architecture allows the IEEE 1394 and I^2C buses to be switched to their backups independently to enhance recovery flexibility. In some scenarios, both the primary and backup IEEE 1394 buses are partitioned by a failed node Then, the switchover to the backup bus fails. The primary bus must be diagnosed and repaired first so that the system operations can resume. After that, the backup bus can be repaired in the background. Details of each layer in figure 2 are explained as follows.

[1]This assumes that the backup bus is healthy. Section 4.6 discusses bus-switching in more detail

Layer 1: Native Fault Detection

The basic fault detection mechanisms of the IEEE 1394 and I^2C buses, *eg*, CRC and acknowledgment are used to detect invalid messages or non-responsive failure modes.

Layer 2: Enhanced Fault Detection & Recovery

A layer of hardware & software fault tolerance is used to detect more difficult failure modes such as babbling and 'conflict of node addresses' in the IEEE 1394 and I^2C buses. Some low-level fault-recovery mechanisms are implemented in each bus.

Layer 3: Fault Isolation & Recovery 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 which partitions the bus network and cuts off communication between the sub-trees. The I^2C bus is used to assist the fault isolation & recovery by maintaining the communication of all nodes. Similarly, if the shared medium of the I^2C bus fails, the 1394 bus can assist in the fault isolation & recovery of the I^2C bus.

Layer 4: Fault Protection by System Level Redundancy

The entire set of IEEE 1394 and I^2C buses are duplicated to provide redundancy for fault recovery. For long-life missions, only one set of the buses is activated in normal operation. If one of the buses in the primary bus-set fails, the backup set of buses is activated, and the system operations are transferred to the backup buses. After that, the failed bus set is diagnosed & repaired. Even though either one of the buses in the primary set can be independently switched to its backup-bus, it is preferred to have the entire bus-set switched. This ensures that the diagnostic operations of the failed bus are transparent to the system operations.

Specific implementation techniques of each layer are described in sections 4.3 – 4.6.

4.3 Native Fault-Containment Regions

This section highlights the basic fault detection mechanisms of the IEEE 1394 and I^2C buses.

4.3.1 Highlights of the IEEE 1394 bus fault-tolerance mechanisms

The 1394 bus standard has many built-in fault detection mechanisms; they are summarized in this section. Details of the acknowledgment and response packet error codes are described in [5, 6].

1. Data and packet header CRCs for both isochronous & asynchronous transactions.

2. Acknowledgment packets that 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 that 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 P1394a [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,
- 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 isolating a failed processor during fault recovery.

4.3.2 Highlights of the I^2C bus fault-detection mechanisms

The only fault-detection mechanism of the I^2C bus 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 issues a stop condition to abort the transaction. Similar situations 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 stops sending data. Subsequently, the master node can issue a stop condition to terminate the transaction if the master node is still functional.

4.4 Enhanced Fault-Containment Regions

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

4.4.1 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 & polling. Heartbeat is effective for detecting root failure; polling can detect individual node failures.

Since the cycle master (root) of the 1394 bus always sends an isochronous cycle-start message every 125 μsec (average), it is reasonable 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 detect missing cycle-start and report to the bus manager of the IEEE 1394 bus via the I^2C bus. Then the bus manager initiates the fault-isolation process by sending an I^2C message to interrogate the health of each node.

Other failure modes can be detected by this method. For example, multiple roots generate more than 1 hardware heartbeat (cycle start) within an isochronous cycle. By comparing the actual heartbeat interval with a minimum anticipated heartbeat interval, the multiple heartbeats can be detected.

The cycle-start can detect only hardware-level faults because it is automatically generated by the link layer hardware. Therefore, polling should be used to detect faults in the transaction or application layers.

Polling is effective in detecting non-responsive nodes. If the non-responsive node is not the root, the hardware heartbeat will not detect its failure. On the other hand, polling can easily detect it non-responsiveness. Therefore, the polling is preferred over the hardware heartbeat in detecting non-responsive nodes.

The root node can 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 needs to be 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. The data field of the original isochronous message contains the source node ID; thus 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 application layer to monitor this message. Upon receiving the message, each node resets its fail-silence timer. If one of the nodes babbles because of a link-layer or application-layer failure, the fail-silence message is blocked or corrupted. This causes the fail-silence timer in each node to time out. Subsequently, the fail-silence timer disables the hardware of its own link layer and thus inhibits the node from transmitting or receiving messages (the ability of the physical layer to pass on a message is unaffected). Eventually, after a waiting period, the link layers of all nodes including the babbling node are disabled and the bus becomes quiet again. At this time, another timer in the root unmutes the root itself and sends a Link-on packet (a physical layer packet) to individual nodes. Upon receiving the Link-on packet, the physical layer of a node sends a signal to awaken its link layer. If a node causes the bus to fail again while its link layer is re-enabled, it is identified as the failed node and is not enabled again. If the root itself is the babbling node, other nodes detect the unmute timeout and issue a 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 include the following types.

· CPU Watchdog Timer: A hardware timer to monitor the health of the host CPU (microprocessor or microcontroller). This watchdog timer is an incremental counter and must be reset by the CPU periodically. If the CPU fails to reset this watchdog, an overflow occurs which then triggers a local reset.

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

4.4.2 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 uses special hardware message 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 allows a fast node to send data to a slow node. However, a failed receiving node can cause a stuck-at-low fault on the SCL signal, so that the sending node might have to wait indefinitely. To recover from this failure mode, every node needs to include 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 disables 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) is used to disable the failed node.

Fail Silence

One node 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 monitor this message with an I^2C bus fail-silence timer. Upon receiving the message, each node resets 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 times-out. Subsequently, the bus transmitters of each node are 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, are disabled and the bus is quiet again. At this time, another timer in the controlling master node unmutes the node itself and sends a message to re-enable the other nodes individually. If a node causes the bus to fail again while it is enabled, it is identified as the failed node and is not enabled again. If the Controlling Master itself is the failed node, other backup nodes, such as the bus manager or the isochronous resource manager of the IEEE 1394 bus, detect the unmute timeout, and promote themselves as the controlling master according to a pre-determined priority.

4.5 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 3 failure-modes that can be isolated & recovered by the cooperation of the buses are listed here.

Non-Responsive Failures

In the IEEE 1394 bus, when a node or one of its links fails in the non-responsive mode, it cannot respond to requests, and messages cannot 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. If the processor or link layer of the node fails, it is the only non-responsive mode. If its physical layer fails, then all the nodes in the sub-tree under it 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 if the fault is in the physical layer because the tree topology of the bus has been partitioned into 2 or 3 segments by the failed node. The nodes in a segment cannot communicate with the nodes in the other segments. Consequently, the root node cannot command some of the nodes to change bus topology if they belong to another segment. 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 complicated, 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 first interrogates the health of the nearest non-responsive node (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^{2}C messages to the other nodes' and 'command them to connectivity their links to bypass the failed node'. This is done by disabling the active connections attached to the failed node and enabling the backup connections to reconnect the separated segments. If the prime-suspect node is fault-free, then the root can repeat the interrogation (and recovery procedure) on the other nodes in the separated sub-trees.

Similarly, if a node in the I^{2}C 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^{2}C bus interface,
- request the target node to retransmit the message,
- command the target node to shut off its bus transmitter if retransmission fails.

IEEE 1394 Bus Physical Layer Babbling

The fail-silence technique is effective in handling babbling-failures in the I^{2}C bus and in the link or application layers in the IEEE 1394 bus. However, it is not effective in handling babbling in the physical layer of the IEEE 1394 bus. The physical layer of the IEEE 1394 bus is rather complicated and contains state machines; thus a transient fault could cause it to babble. Such failures cannot be handled by fail-silence, because if the physical layer is silenced, it cannot pass-on messages, and thus causes bus partitioning. In this case, each node can check its own physical layer (*eg*, read the physical layer registers). If the physical layer of a node is faulty, the processor of the node can issue a physical-layer reset to correct the problem. If the physical-layer fault is permanent, then the node has to inform the root node via the I^{2}C bus. Subsequently, the root node can command other nodes via the I^{2}C bus to reconfigure the bus topology to bypass the failed node.

Conflict of Node Addresses

The address of any node in the IEEE 1394 or I^{2}C 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 is corrupted by bus conflict from the two nodes, and any write-transaction goes to both nodes, and can have unpredictable consequences. Hence, it is difficult to disable the fault node by the bus itself. However, with the redundant IEEE 1394/I^{2}C bus set, this kind of failure can be handled through using one bus to isolate and then disable a faulty node on the other bus, so that the erroneously duplicated node address can be eliminated.

4.6 Fault Protection by System-Level Redundancy

The COTS bus set is duplicated to provide system-level fault protection. Using the redundant COTS bus set to handle faults is explained in section 4.2.

To enhance the effectiveness of system-level redundancy, a special tree topology called stack-tree [14] is employed. This topology permits the IEEE 1394 bus in the backup bus set to initially connect the nodes in such a way that any branch node in the IEEE 1394 bus of one bus set is a leaf node in the other bus set. As mentioned in section 4.2 and described in detail in [14], backup connections (disabled) are added to the IEEE 1394 bus in each bus set to allow the bus network to be reconfigured via port-disabling and port-enabling. In particular, when a node failure occurs, it is first examined whether the failed node is a branch node in the active bus and a leaf node in the backup bus. If this is the case, recovery is accomplished by switching to the backup bus and then reconfiguring the failed bus in background. However, if the failed node is a branch node for both buses, then the recovery begins with reconfiguring the failed active bus. Should another node failure occur subsequently, the same recovery rule is applied.

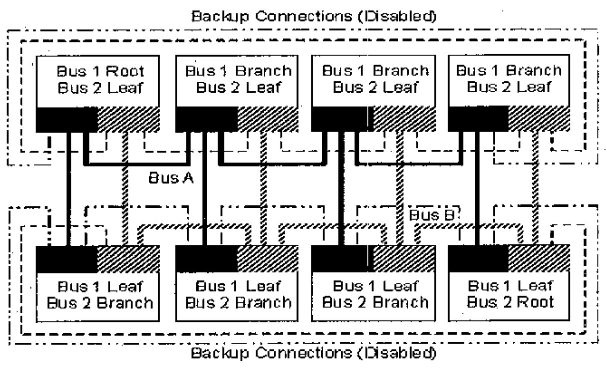

Figure 3: Stack-Tree Topology of IEEE 1394 Bus

4.7 Fault Recovery under Catastrophic Failure Conditions

Under catastrophic failure conditions such as bus power failure, both COTS bus sets can fail such that 'all communication among the nodes' is lost. To re-establish the communication, each node can execute a distributed recovery procedure that consists of a sequence of link enable/disable steps. The 'enabled links of all the nodes in each step' form 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 rather tight synchronization among all the nodes, which is very difficult to achieve when all bus communications are lost. Because the cause of the catastrophic failure might 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. CURRENT STATUS OF THE FAULT-TOLERANT COTS-BASED BUS ARCHITECTURE IMPLEMENTATION

The X2000 Program at JPL has already implemented some of the fault-tolerance techniques in this paper; the rest of the techniques are still being implemented. The implemented techniques include:

- the native fault-detection features of the buses,
- fail silence,
- protocol enhancement,
- watchdog timers [15].

A testbed will be completed in early year 2000, and several levels of simulation models are being developed. The flight version of the system will be delivered to the Europa Orbiter mission by the end of the year 2001.

As the testbed and simulation models are completed, the design techniques in each level 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 implementation and test results will be reported in future papers.

ACKNOWLEDGMENT

The research in this paper was carried out by the JPL, California Institute of Technology, under a contract with the US National Aeronautics and Space Administration. It has been done for the X2000 Integrated First Delivery Project, DSST Future Deliveries Project, and Center for Integrated Space Micro-Systems.

REFERENCES

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

[2] L. Alkalai, "NASA center for integrated space microsystems",*Proc. Advanced Deep Space System Development Program Workshop on Advanced Spacecraft Technologies*, 1997 Jun; California.

[3] L. Alkalai, "A roadmap for space microelectronics technology into the new millennium", *Proc.* 35th *Space Congress*, 1998 Apr; Florida.

[4] L. Alkalai, S. Chau, "Description of X2000 avionics program", *Proc.* 3rd *DARPA Fault-Tolerant Computing Workshop*, 1998 Jun; California.

[5] IEEE 1394, *Standard for a High Performance Serial Bus*, 1995 Jan; IEEE.

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

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

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

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

[10] IEEE P1394A, *Standard for a High Performance Serial Bus* (Supplement), Draft 2.0, 1998 Mar; IEEE.

[11] W. Peterson, *The VMEbus Handbook*, Expanded 3rd ed, 1993; VFEA Int'l Trade Assoc.

[12] J. Marshall, "Building standard based COTS multiprocessor computer systems for space around a high speed serial bus network", *Proc.* 17th *Digital Avionics Systems Conf*, 1998 Nov; Washington.

[13] J. Donaldson, "Cassini orbiter functional requirements book: Command and data subsystem", *JPL Document* CAS-4-2006, 1994 Jun 28; Jet Propulsion Laboratory, California.

[14] A.T. Tai, S.N. Chau, L. Alkalai, "COTS-based fault tolerance in deep space: Qualitative and quantitative analyses of a bus network architecture", *Proc.* 4th *IEEE High-Assurance System Eng'g Symp*, 1999 Nov; Washington DC.

[15] H. Luong, "X2000 first delivery digital input and output (DIO) ASIC", *JPL Document* D-16931, 1999 Mar; Jet Propulsion Laboratory, California.

[16] S.N. Chau, *Bus Trade Study (Summary and Raw Data)*, DSST Project Library, Jet Propulsion Laboratory, California

AUTHORS

Dr. Savio N. Chau; Jet Propulsion Laboratory; Pasadena, California 91109 USA.
Internet (e-mail): savio.chau@jpl.nasa.gov

Savio N. Chau received his PhD in Computer Science from UCLA. He is developing scalable multi-mission avionics system architectures for the X2000 Program, and has been investigating techniques to apply low-cost commercial technologies for long-life spacecraft. His interests include distributed system architecture, fault tolerance, and design-for-testability.

Dr. Leon Alkalai; Center for Integrated Space Microsystems (CISM); Jet Propulsion Laboratory; Pasadena, California 91109 USA.
Internet (e-mail): leon.alkalai@jpl.nasa.gov

Leon Alkalai joined JPL in 1989 after receiving his PhD in Computer Science from UCLA. Since then, he has worked on numerous technology-development tasks including advanced microelectronics miniaturization, advanced microelectronics packaging, reliable and fault-tolerant architectures. He was one of the NASA appointed co-leads on the New Millennium Program Integrated Product Development Teams for Microelectronics Systems. He is currently the director of CISM.

Dr. Ann. T. Tai; IA Tech; 10501 Kinnard Ave; Los Angeles, California 90024 USA.
Internet (e-mail): a.t.tai@ieee.org

Ann T. Tai received her PhD in Computer Science from UCLA. Her research interests include development & application of performance & dependability models, and fault-tolerant system-architecture design. She authored the book, *Software Performability: From Concepts to Applications*.

John B. Burt; Jet Propulsion Laboratory; Pasadena, California 91109 USA.
Internet (e-mail): john.burt@jpl.nasa.gov

John B. Burt received his BS & MS in Computer Science from the Univ. of Southern California. He is working on fault protection and hardware-software interface issues for the X2000 Program. His interests include spacecraft autonomous fault detection and response designs.

Manuscript TR1999-404 received: 1999 May 12;
revised: 1999 August 26, October 18.
Responsible editor: M.R. Lyu
Publisher Item Identifier S 0018-9529(99)10381-6