A CONCEPTUAL DESIGN FOR A RELIABLE OPTICAL BUS (ROBUS)

Paul S. Miner, Mahyar Malekpour, and Wilfredo Torres

NASA Langley Research Center, Hampton, VA

{p.s.miner, m.r.malekpour, w.torres-pomales}@larc.nasa.gov

Abstract

The Scalable Processor-Independent Design for Electromagnetic Resilience (SPIDER) is a new family of fault-tolerant architectures under development at NASA Langley Research Center (LaRC). The SPIDER is a general-purpose computational platform suitable for use in ultra-reliable embedded control applications. The design scales from a small configuration supporting a single aircraft function to a large distributed configuration capable of supporting several functions simultaneously. SPIDER consists of a collection of simplex processing elements communicating via a Reliable Optical Bus (ROBUS). The ROBUS is an ultra-reliable, time-division multiple access broadcast bus with strictly enforced write access (no babbling idiots) providing basic fault-tolerant services using formally verified fault-tolerance protocols including Interactive Consistency (Byzantine Agreement), Internal Clock Synchronization, and Distributed Diagnosis. The conceptual design of the ROBUS is presented in this paper including requirements, topology, protocols, and the block-level design. Verification activities, including the use of formal methods, are also discussed.

Introduction

The Scalable Processor-Independent Design for Electromagnetic Resilience (SPIDER) is a general-purpose fault-tolerant architecture being designed at NASA Langley Research Center to support laboratory investigations into various recovery strategies from transient failures caused by electromagnetic effects. The core of the SPIDER architecture is the Reliable Optical Bus (ROBUS). As part of an effort partially sponsored by the FAA, the ROBUS is being developed in accordance with RTCA DO-254: *Design Assurance Guidance for Airborne Electronic Hardware*.

The SPIDER is a family of general-purpose computational platforms suitable for use in ultra-reliable embedded control applications. Towards this end, the design is intended to scale from a small configuration supporting a single aircraft function to a moderately large configuration capable of supporting several functions simultaneously. The computational platform is suitable for use for any safety-critical (Level A) aircraft function.

SPIDER Overview

The SPIDER architecture is intended to support a collection of *N* simplex general purpose Processing Elements (PEs) communicating over a Reliable Optical Bus (ROBUS). One logical view of the SPIDER Architecture is depicted in Figure 1.

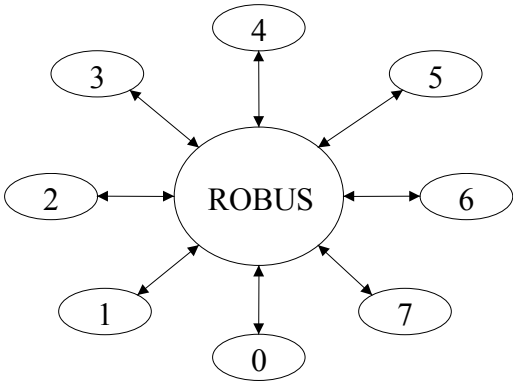

Figure 1: SPIDER Logical View

The ROBUS logical behavior is a time-division multiple access (TDMA) broadcast bus. In order to guarantee reliable communication among the good PEs, the bus needs to be protected against any bad PE monopolizing its capacity. Furthermore, the communication model must

support several fundamental services. The essential goal is to ensure reliable communication between all pairs of fault-free PEs in the system. This will enable the development of several fault-tolerance strategies combining the individual PEs. For example, Figure 2 illustrates a SPIDER configuration with three PEs in a Triple Modular Redundant (TMR) configuration, four PEs in a dual-dual configuration and a single simplex PE.

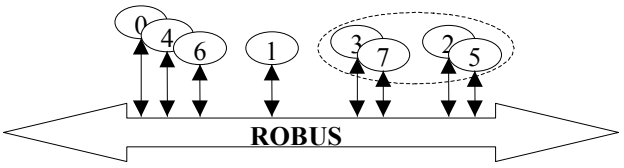

Figure 2: Sample SPIDER Configuration

ROBUS Requirements

The ROBUS must provide the following capabilities:

1. All good nodes will observe an identical sequence of messages on the ROBUS
2. The ROBUS will provide a reliable time reference for all attached nodes
3. The ROBUS will provide correct and consistent diagnostic information to all attached nodes
4. For a 10 hour mission,

$$P(\text{ROBUS Failure}) < 10^{-10}$$

The first three requirements are desirable functional characteristics of a reliable bus. The final requirement is motivated by the fact that the ROBUS may support communication for several functions whose failure could be catastrophic. Since the bus should not be a dominant source of failure, we have set the reliability requirement to be significantly greater than would be required for any aircraft function. Rushby presents a comparison of several architectures with similar requirements [1].

Allocation of Requirements

In this section, we address the implications these requirements have on the available design choices. Requirement 4 implies that the ROBUS will have internal redundancy. It is not possible to meet this reliability goal without replication. This implies that all of the other services must be guaranteed in the presence of a bounded number of internal ROBUS component failures.

Reliability

In order to satisfy Requirement 4, we have constructed semi-Markov models that use the same fault assumptions as the fault-tolerance protocols. Death states in the Markov models correspond to violations of fault assumptions.

Fault Assumptions

There are at least two approaches to reasoning about faults and failures in a digital system. One is to postulate possible component failures and then assess the resulting impact on the system. Alternatively, one may assume that all faults have potentially devastating consequences and then design the system relative to this worst case assumption. Our approach is closer to the latter, but we will allow some variation into the potential impact of faults. We have modified the fault-classification strategy used in the development of the Multiprocessor Architecture for Fault-Tolerance (MAFT) [2]. Faults are classified based on the observable characteristics to other nodes within the system. The system is partitioned into Fault Containment Regions (FCR) that ensure independence of random physical failures. The failure status of an FCR is then one of four mutually exclusive possibilities:

- A **good** node behaves according to specification
- A **benign** faulty node only sends messages that are detectably faulty, including nodes that have failed silent
- A **symmetric** faulty node may send arbitrary messages, but does so the same way to each receiver
- An **asymmetric** faulty node may send different arbitrary messages to different receivers

This provides a global classification of the fault status of a collection of nodes. This classification is useful for the analysis of the various fault-tolerant protocols in the system. However, the protocols themselves cannot have complete knowledge of the current failure status of the other nodes in the system, so the protocols

cannot make decisions based upon this classification.

The protocols must make use of local knowledge about the fault status. Each FCR in the design will maintain a local determination of which FCRs are *trusted*. Only information from trusted FCRs will be considered during a vote. For the protocols to work properly, a good FCR's local view of which nodes to trust must satisfy the following properties:

1. Good nodes always trust other good nodes
2. When a vote function is computed, no good node trusts any benign-faulty node
3. If FCR *i* is not asymmetric-faulty, then good nodes agree on whether or not *i* is trusted

Static Schedule

The simplest solution to Requirement 1 is to first use a static communication schedule for the ROBUS. The initial prototype uses a round robin schedule where each PE has equal access to the ROBUS. However, all analysis is based upon the weaker assumption that all nodes agree on the communication schedule. This will allow us to explore dynamic scheduling algorithms for later instances of the SPIDER architecture.

Interactive Consistency

Since the ROBUS must have internal redundancy to achieve the reliability requirements, we also need an interactive consistency protocol to satisfy requirement 1. In a redundant computer system, it is necessary to ensure that all single-source data items are consistently replicated among the redundant computational elements. Otherwise, a single faulty source may be able to overwhelm the system. There are several published algorithms for ensuring interactive consistency; the first fully general solution is by Pease et al [3]. Interactive consistency requirements are:

Agreement -- All non-faulty receivers agree on the single source data value received

Validity -- If the originator of the data is non-faulty, then all non-faulty receivers receive the transmitted value

Protocols that satisfy these requirements assume that the participants are synchronized within a known skew.

Clock Synchronization

Requirement 2 demands a fault-tolerant clock synchronization protocol for the ROBUS. The general requirements for clock synchronization are:

Precision---There is a small constant d such that for any two clocks that are good at real time t: $|C_1(t)-C_2(t)| < d$

Accuracy---All good clocks maintain an accurate measure of the passage of time

These properties are sufficient to ensure the ROBUS satisfies requirement 2.

Many synchronization protocols are round-based. The participants in the protocol periodically exchange clock readings to compute an adjustment for the next round. For such protocols, two conditions are sufficient to ensure precision and accuracy.

Bounded Delay---All good clocks start each round, k, within a bounded duration of real time

Bounded Adjustment---There is an upper bound on the magnitude of the adjustment a good clock makes in a round

Bounded delay merely means that the net effect of all the computed adjustments maintains the precision of the system. Bounded adjustment preserves the accuracy of the synchronized clocks. The adjustment bound should be significantly less than the duration of a round.

Diagnosis

Algorithms for clock synchronization and interactive consistency may be designed to operate correctly under several different fault-assumptions. In order to meet requirement 3, the ROBUS will support distributed diagnosis algorithms. For diagnosis to be useful, we require all good nodes to agree on the results of the diagnosis protocol. This will be ensured by exchanging the diagnostic data

using the interactive consistency protocol. The role of a diagnosis algorithm is to identify failed nodes within the system. The goal of a diagnosis algorithm is to ensure the following properties:

Correctness---Every FCR diagnosed as faulty by a good FCR is indeed faulty

Completeness---Every faulty FCR is eventually diagnosed as faulty

If the fault model includes Byzantine (asymmetric) faults, it is impossible to guarantee both of these properties [4]. There always exist fault scenarios where it is known that there is a fault in the system, but it is impossible to identify precisely which FCR is faulty. In such cases, either correctness or completeness must be sacrificed. If correctness is sacrificed, then some good nodes may be declared faulty and removed from the system. If completeness is sacrificed, then actively faulty nodes may remain in the system. The choice of which property to guarantee is open to debate. For the ROBUS we have chosen to ensure correctness and make the diagnosis as complete as possible.

ROBUS Conceptual Design

ROBUS Topology

The ROBUS consists of a collection of *N* Bus Interface Units (BIUs) and *M* Redundancy Management Units (RMUs) connected as a complete bipartite graph $K_{N,M}$. The *N* BIUs will each have a bi-directional link to a single processing element (PE). A ROBUS consists of *N* + *M* distinct fault containment regions, one for each BIU and RMU. An implementation may choose to include a PE in the same FCR as its associated BIU, but this is not required. The choice of whether to combine a BIU and PE in a single FCR is guided by the reliability model. A PE may contain substantially more hardware than a BIU. If so, its failure rate will dominate the BIU failure rate. We may be able to ensure better system reliability if the BIU and PE are in separate FCRs.

The topology of the communication structure for a SPIDER with a $ROBUS_{N, M}$ is shown in Figure 3.

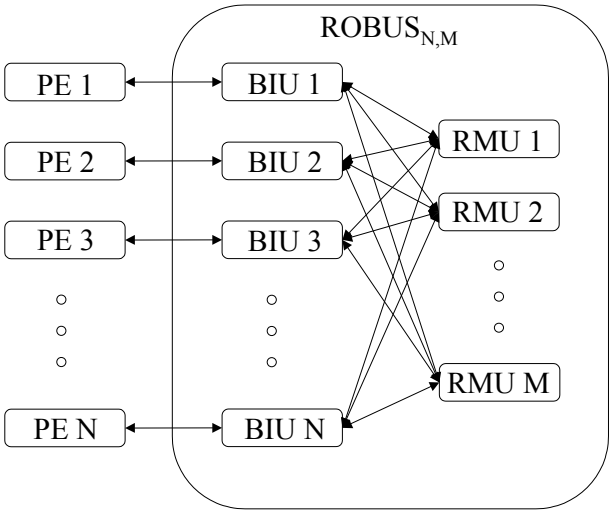

Figure 3: SPIDER Architectural Structure

According to the global fault model, we can partition the BIUs into four disjoint sets based upon their fault classification. Let GB denote the good BIUs, BB the benign faulty BIUs, SB the symmetric faulty BIUs, and AB the asymmetric faulty BIUs. We similarly partition the RMUs. Let GR denote the good RMUs, BR the benign faulty RMUs, SR the symmetric faulty RMUs, and AR the asymmetric faulty RMUs. The **maximum fault assumption** for the ROBUS protocols is:

1. $|GB| > |AB| + |SB|$
2. $|GR| > |AR| + |SR|$
3. $|AR| = 0$ or $|AB| = 0$

These fault combinations will determine the death states in the SURE reliability model. The critical path in the reliability model will be due to fault assumption 3. We can easily add enough redundancy to make the probability of ROBUS failure due to fault assumption 1 or 2 insignificant.

Reliability Models

The ROBUS protocols are designed to work whenever the above fault assumptions are satisfied. We have developed an ASSIST script to generate SURE models for various ROBUS configurations. The SURE program computes bounds on the solution of a (semi) Markov model. In addition to SURE, the programs PAWS and STEM compute exact solutions of Markov models. The programs

SURE, PAWS, and STEM all use the same input format.

The reliability models for the ROBUS calculate the probability that any of the three fault assumptions are violated for a given duration mission and hardware fault arrival rate. Since the design is in a conceptual stage, we are using generic order-of-magnitude approximations for the fault arrival rates.

The ASSIST script prompts for the number of BIUs and RMUs and then generates a user-modifiable SURE model. The SURE user must specify the percentage of faults that are benign or symmetric for both the BIUs and the RMUs. Additionally, the SURE user may provide diagnostic coverage probabilities for each class of symmetric and asymmetric faults. Finally, the user may specify a recovery rate for the diagnosable faults. The only recovery mechanism included in the model is graceful degradation. Faulty units that are correctly diagnosed are removed from the system. The model does not include recovery from transient faults. All faults are assumed permanent. We intend to add transient faults in the future. The following table enumerates the parameters for the generated model:

Table 1: Parameters for the Reliability Models

Parameter	Description	Default
λ_B	BIU Fault Arrival Rate	10^{-6}/hour
λ_R	RMU Fault Arrival Rate	10^{-6}/hour
Time	Duration of Mission	10 hours
B_B	Probability that a BIU fault is benign	0
S_B	Probability that a BIU fault is symmetric	0
A_B	Probability that a BIU fault is asymmetric	1 - (B_B + S_B)
B_R	Probability that an RMU fault is benign	0
S_R	Probability that an RMU fault is symmetric	0
A_R	Probability that an RMU fault is asymmetric	1 - (B_R + S_R)
D_{SB}	Probability that a symmetric BIU fault is diagnosable	0
D_{AB}	Probability that an asymmetric BIU fault is diagnosable	0
D_{SR}	Probability that a symmetric RMU fault is diagnosable	0
D_{AR}	Probability that an asymmetric RMU fault is diagnosable	0
α	Rate of diagnosis and reconfiguration (for all diagnosable faults)	1/second

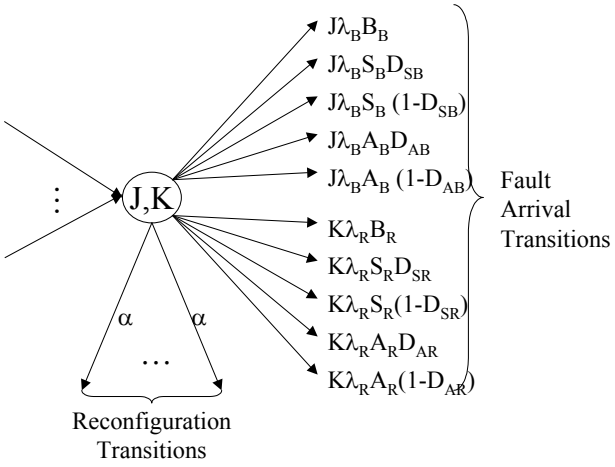

Figure 4: Transitions in ASSIST Model

The default failure rate is based on historical failure rates for a single VLSI device. In Kopetz ([5], page 121), the failure rate for a high quality chip is claimed to be better than 10^{-7}/hour.

Figure 4 shows the transitions generated by the ASSIST script when the system is in a state with J good BIUs and K good RMUs. The fault arrival transitions cover the various possibilities when a new fault arrives. If there are any diagnosable faults present in the current state, then appropriate reconfiguration transitions are generated. The only reconfiguration strategy is graceful degradation. After a non-benign fault is diagnosed, the protocols can ignore it. Reconfiguration is modeled by converting diagnosed faults to benign faults.

The ASSIST script has been validated both by hand inspection and by solving the generated model with parameters set to extreme cases. These

examples are easy to check by hand using either the algebraic SURE bounds [6] or combinatorial analysis.

Interactive Consistency Protocol

The Interactive Consistency (IC) protocol is designed to satisfy requirement 1. If all communication uses this protocol, then all good Processing Elements will observe the same sequence of data. This protocol is also used to reliably exchange diagnostic data among the good nodes within the ROBUS.

For the Interactive Consistency protocol, we assume that all FCRs are synchronized within a known skew and that the implementation can avoid adverse effects due to this skew. Also, every FCR knows the communication schedule. An informal description of the protocol is as follows:

1. PE *j* transmits its message *v* to BIU *j*, in accordance with the agreed schedule
2. BIU *j* broadcasts *v* to all RMUs
3. For each RMU *k*, if RMU *k* does not receive a correctly formatted message from BIU *j*, then it broadcasts *source error* to all BIUs, otherwise it broadcasts the value v_k to all BIUs
4. Each BIU collects the values received (v_1 , ..., v_M). If a BIU does not receive a correctly formatted message from RMU *k*, it removes RMU *k* from its set of *trusted* RMUs
5. Each BIU determines if there is a majority among the values received from the *trusted* RMUs
6. If BIU *l* determines that a majority of *trusted* RMUs sent the same value v_{maj}, BIU *l* transmits v_{maj} to PE *l*. Otherwise, BIU *l* transmits *no majority* to PE *l*

Theorem: This protocol satisfies both **Agreement** and **Validity** assuming the maximum fault assumption holds.

Proof of Agreement: There are two cases to consider,

Case 1: |AR| = 0

Since there are no asymmetrically faulty RMUs, all good BIUs agree on which RMUs to *trust*. All good BIUs receive the same vector of values in step 4. Thus, in steps 5 and 6, each good BIU will determine the same value to forward to its PE.

Case 2: |AB| = 0

In this case, the BIU broadcasting in step 2 cannot be asymmetrically faulty, so all good RMUs will broadcast the same value in step 3. Fault assumption 2 ensures that there are more good RMUs than the combined total of asymmetric faulty and symmetric faulty RMUs. Since every benign faulty RMU is manifest-faulty to every good BIU, all benign faulty RMUs will be ignored. The good RMUs form a majority, so the value they broadcast in step 3 will be the same as the value transmitted to the PEs in step 6.

Proof of Validity: Validity follows immediately from the proof of Case 2 for Agreement.

- If BIU *j* is good, then all good RMUs will correctly forward its value in step 3
- If BIU *j* is benign faulty, then all good RMUs will broadcast *source error* in step
- If BIU *j* is symmetric faulty, then all good RMUs will forward the value received in step 3

There is a useful corollary to **Validity** that will aid in diagnosis.

Corollary: If a good BIU receives invalid data (i.e. *source error* or *no majority*) as a result of executing the Interactive Consistency protocol, then the originating BIU is faulty.

By a symmetric argument, we can use the same protocol (steps 2 through 5) to exchange data between RMUs. This capability is needed to exchange diagnostic information.

Diagnosis Protocol

The ROBUS diagnosis protocols are based on the MAFT approach to on-line diagnosis presented by Walter, et al [7]. The MAFT protocol can be abstractly subdivided into two phases: local diagnosis and global diagnosis. In the local diagnosis phase, each node monitors the behavior of all other nodes. From these observations, it constructs an error syndrome that identifies those nodes that it believes to be faulty. The global diagnosis phase consists of an interactive consistency exchange to reliably distribute the accusations followed by a voting step to make a

globally consistent decision based upon the set of all local accusations.

In the MAFT architecture, all nodes are identical and the nodes are completely connected. In the ROBUS, there are two different kinds of nodes, the RMUs and the BIUs. In addition, there are no direct links between a pair of BIUs or between a pair of RMUs. The global diagnosis phase had to be modified to accommodate the ROBUS characteristics. For the ROBUS protocol, there are several levels of diagnostic information:

1. **Suspicions:** Node *k* suspects nodes *i* and *j* when it knows that at least one of *i* or *j* is bad, but does not have sufficient information to accuse either
2. **Accusations:** Node *k* accuses node *j* when it has sufficient evidence to conclude that node *j* is faulty
3. **Declarations:** Node *k* declares node *j* to be faulty when it knows that all good nodes of the *same kind* as *k* have sufficient evidence to conclude that *j* is faulty
4. **Convictions:** A node is convicted when *all* good nodes have declared it faulty

The principal distinction between accusations and declarations is that accusations only depend upon local knowledge, but declarations depend upon common knowledge. This common knowledge is a side effect of the protocols. We can now make precise the notion of *trusted* nodes as employed by the IC and Synchronization protocols.

A node is considered *trusted* if it has not been accused, declared, or convicted. The voting functions in the IC protocol and the synchronization protocol only consider messages from trusted sources.

A node is considered *undeclared* if it has not been declared or convicted. This classification is needed for some of the votes employed by the diagnosis protocol.

For the diagnosis protocol to work, all accusations must satisfy the following property:

If node k accuses node j, then at least one of node k or node j must be faulty

A direct consequence is the following property:

If any good node accuses node j, then node j is faulty

This ensures that every good node is *trusted* by all good nodes. We also require that, at the time of any vote, no benign-faulty node be *trusted* by any good node. Finally, we allow good nodes to disagree concerning asymmetric faulty nodes, but we require that they agree on whether to trust symmetric-faulty nodes.

There are several ways for a node to make an accusation. These include both direct error checking by the receiving node and sufficient disagreement with voted results. Suspicions against node *k* are promoted to accusations when it is known that *k* is suspected in conjunction with at least one good node.

A node may make a declaration in at least two ways. First, the interactive consistency protocol provides partial diagnostic information. If the result of an interactive consistency exchange is any sort of error, then all good receiving nodes know that the originator is faulty. In this case, since all good nodes know the source is faulty, they all *declare* the source faulty. The second mechanism for making declarations is based upon distributed diagnostic information. If there is sufficient evidence in a consistent set of accusations to conclude that a node is faulty, then that node is declared faulty. A set of accusations is consistent if all good nodes (of the same kind) agree on the contents of that set of accusations. There is sufficient evidence to conclude that a node is faulty if it accuses itself or if it is accused by a majority of *undeclared* nodes.

The ROBUS diagnosis protocol is as follows:

1. All nodes gather accusations against all other nodes
2. All nodes gather declarations based on the properties of the interactive consistency protocol
3. The BIUs periodically exchange their accusations with all other BIUs using the interactive consistency protocol. If a majority of *undeclared* BIUs accuse a node, that node is declared faulty
4. The RMUs periodically exchange their accusations with all other RMUs using the interactive consistency protocol. If a majority

of *undeclared* RMUs accuse a node, that node is declared faulty

5. All BIUs broadcast their declarations to all RMUs. The RMUs perform a majority vote of the declarations received from *trusted* BIUs
6. All RMUs broadcast their declarations to all BIUs. The BIUs perform a majority vote of the declarations received from *trusted* RMUs
7. Any node declared faulty by either a majority of *trusted* BIUs or a majority of *trusted* RMUs is convicted
8. The BIUs forward the list of convicted nodes to the PEs

Since the current system is designed using the assumption that all faults are permanent, any node that is convicted is permanently isolated from the rest of the system. The protocol is being modified to remove this assumption.

Formal proofs of the distributed diagnosis protocol are described in [8].

The protocol ensures that:

- Every declared node is convicted
- Every benign faulty node is declared
- Every symmetric faulty nodes accused by some good node is declared
- Every node accused by a trusted majority of nodes is declared

Both the distributed diagnosis and interactive consistency protocols are synchronous. It is essential that the inherent asynchrony between any pair of nodes be bounded. Further, the design must ensure that the relative skew be masked.

Clock Synchronization

The SPIDER clock synchronization protocol is an event-based protocol. Periodically, good clocks will generate events indicating that it is time to start the next round. The protocol is designed to ensure that the events generated by good clocks are echoed in such a way that all good clocks will reset within a short time of each other. The duration of a round is approximately P ticks. An informal description of the protocol follows:

RMU:

Process 1: When time to resynchronize for round k, broadcast (init, k) to all BIUs

Process 2: If *Accept?*(echo, k) then broadcast (echo, k) to all BIUs and reset counter for round k

BIU:

Process 1: If *Accept?*(init, k) then broadcast (echo, k) to all RMUs

Process 2: If *Accept?*(echo, k) then reset counter for round k (and transmit reset to PE)

The fault-tolerance is in the definition of function *Accept?*. It selects the middle event from the *trusted* sources. The times to resynchronize and values for resetting counters are selected to accommodate the inherent communication delays.

Communication between independently clocked synchronous systems is necessarily imprecise. If a node sends a message at time t, it will be received by all good nodes during the time interval $[t + d, t + d + e]$. Here d denotes the minimum communication delay and e is a bound on the error. The dominant source of error is due to discretization; e is always larger than the duration of one clock tick. Other factors that contribute to this error term are jitter, drift, and slight differences in communication delay due to various causes (e.g. temperature effects, differences in wire length, etc.).

Lemma: All good RMUs reset their clocks within $2e$ of each other.

Proof: There are two cases.

Case 1: $|AB| = 0$

In this case, the echo broadcast events generated by BIU process 1 are totally ordered. All good RMUs will accept in response to (essentially) the same event. The relative time difference that two good RMUs can observe this event is bounded by e.

Case 2: $|AR| = 0$

In this case, the init events generated by the RMUs are totally ordered. All good BIUs will accept init (and broadcast echo) within e of each other. Since accept is bounded by good events, e is the maximum skew effect an asymmetric fault can have. Adding another e for the inherent imprecision in communication ensures that all good RMUs will accept within $2e$ of each other.

A symmetric argument bounds the separation of good BIUs. The separation of any BIU/RMU pair is bounded by 3*e* leading to the following result:

Theorem: The synchronization protocol satisfies bounded delay.

The protocol also guarantees bounded adjustment, thus leading to:

Theorem: The synchronization protocol guarantees **precision** and **accuracy**.

Informal proofs of these properties can be found in [9]. Machine checked proofs are in progress.

ROBUS Architecture

The ROBUS consists of two primary design elements: the BIU and the RMU. The block structure of these devices is presented below.

BIU Block Model

The block structure of a BIU is depicted in Figure 5. There are several functional blocks. The Input Unit is responsible for de-skewing data messages, and accusing any source that transmits an invalid message. It also directs all synchronization messages to the Synchronization Unit. The Route & Vote unit performs the core functions of the interactive consistency protocol. It either relays messages or votes results. The Synchronization Unit implements the event voter needed for the synchronization protocol. When enough synchronization messages have arrived, it signals the Control Unit to take appropriate action. The Diagnostics Unit maintains all of the diagnostic state information. It generates a vector of trusted sources based on the current accusations, declarations, and convictions. It also performs the voting required in the diagnosis protocol. The Output Unit selects the appropriate source for the next broadcast message. The Control Unit realizes the steps of all of the ROBUS protocols and maintains the schedule and timer. Finally, the PE Interface manages communication with the attached PE.

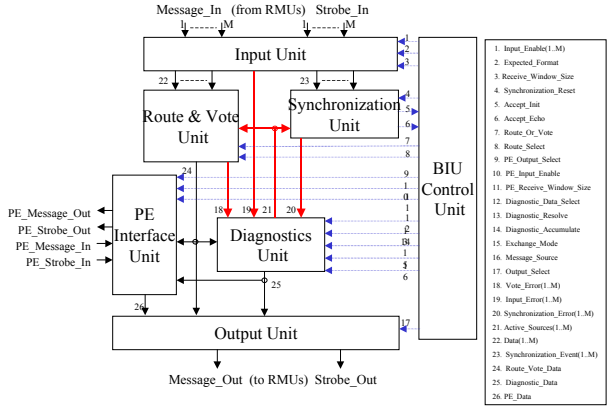

Figure 5: BIU Block Model

RMU Block Model

The block structure of the RMU design is depicted in Figure 6. Its structure is quite similar to the BIU. The main difference is that the RMU does not have an interface to the PE. The operation of the Control Unit is also different, due to the differing roles in the protocols.

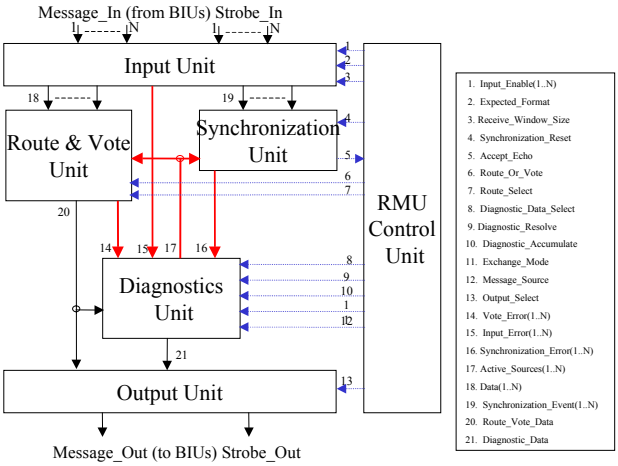

Figure 6: RMU Block Model

Verification Issues

We are developing the ROBUS in accordance with guidance found in RTCA DO-254. Since the ROBUS is intended to support any aircraft function whose failure would be catastrophic, the ROBUS is being developed to design assurance level A.

DO-254 requires that the design assurance for any level-A device employ approaches found in appendix B. Two of these are relevant to this development.

The primary design-assurance strategy for the ROBUS is the use of formal methods. The application of formal methods is targeted to early life-cycle application. The emphasis is upon formal proof of the critical fault-tolerance protocols. We have complete machine-checked proofs of the interactive consistency protocol and the distributed diagnosis protocol. These proofs are described in detail by Geser and Miner [8]. The formal verification of the synchronization protocol is incomplete. The main properties have been checked, but there is still a need to put the pieces together.

The other relevant strategy from DO-254 appendix B is the use of elemental analysis. We have selected the TransEDA tool VN-cover to support this analysis, but have not yet carried out this verification exercise. Our preliminary analysis indicates that coverage of VHDL code using Focused Expression Coverage is mathematically equivalent to MC/DC coverage.

Concluding Remarks

In this paper, we have presented a conceptual design of a family of fault-tolerant architectures. The SPIDER architecture provides a flexible framework for building fault tolerant applications. The primary mechanism for ensuring fault-tolerance in the SPIDER family of architectures is the Reliable Optical Bus (ROBUS). The ROBUS reliably provides several key fault tolerant capabilities. It provides an interactive consistency protocol to enable reliable communication in the presence of arbitrarily malicious failures. It provides consistent diagnostic information so all nodes can make consistent reconfiguration decisions. Finally, it provides an underlying fault-tolerant synchronization mechanism to provide a reliable time source, and provide a means to construct synchronous protocols on top of the ROBUS. These protocols have been formally verified to provide the greatest possible assurance that they are correct.

The SPIDER protocols all have very simple descriptions. However, their interactions are quite complex. We have presented a block model illustrating one hardware realization of these protocols. This design has been implemented on a laboratory prototype and testing is currently in progress.

References

[1] Rushby, J., 2001, A Comparison of Bus Architectures for Safety-Critical Embedded Systems, www.csl.sri.com/~rushby/abstracts/buscompare

[2] Kieckhafer, R. M., C. J. Walter, A. M. Finn, and P. M. Thambidurai, 1988, The MAFT Architecture for Distributed Fault Tolerance, *IEEE Transactions on Computers, 37* (4), 398-405.

[3] Pease, M., R. Shostak, and L. Lamport, 1980, Reaching Agreement in the Presence of Faults, *Journal of the ACM, 27* (2), 228-234.

[4] Shin K, and P. Ramanathan , 1987, Diagnosis of Processors with Byzantine Faults in a Distributed Computing System. In 17th Fault-Tolerant Computing Symposium, 55-60.

[5] Kopetz, H., 1997, *Real-Time Systems: Design Principles for Distributed Embedded Applications,* Kluwer Academic Publishers, Boston.

[6] Butler, R. W., and A. L. White, 1988, SURE Reliability Analysis: Program and Mathematics, *NASA Technical Paper, 2764*.

[7] Walter, C. J., P. Lincoln, and N. Suri, 1997, Formally Verified On-Line Diagnosis, *IEEE Transactions on Software Engineering, 23* (11), 684-721.

[8] Geser A, and P.S. Miner, 2002, A Formal Correctness Proof of the SPIDER Diagnosis Protocol. In: Carreno V., Munoz C., and Tahar S., eds. Track B Proceedings of the 15th International Conference on Theorem Proving and Higher Order Logics, 71-86.

[9] Miner, P.S., M. Malekpour, and W. Torres-Pomales, 2002, *ROBUS Conceptual Design,* NASA Technical Memorandum (To Appear), Hampton, VA.