Appendix B

SAFEbus Detailed Protocol Description

SAFEbus is the only backplane or local area data network to become a standard (ARINC 659) that provides fail-op / fail-safe fault tolerance with near unity coverage for all of its components — signal lines, terminations, interface electronics, clock sources, and power supplies. This coverage includes tolerating a Byzantine fault. SAFEbus provides a time-based protocol that delivers messages with a precision on the order of 100 nanoseconds over a backplane network.

B.0.1 Background

In the late 1980s, there was a push towards integration of multiple functions on a common computing and I/O platform, also referred to as Integrated Modular Avionics (IMA). This push was due to the advantages of IMA systems over then prevalent federated architectures. The advantages are decreased size, cost, and weight, increased reliability, less-frequent maintenance, and more flexibility. The success of an IMA system hinges on a backplane bus connecting Line Replaceable Modules (LRMs). The backplane bus must be designed to support the dual requirements of *space and time determinism.* These requirements are derived from the concept of "robust partitioning" that prevents functions on a common platform from adversely influencing each other, even when some functions may be faulty. Honeywell designed SAFEbus as a backplane for the Aircraft Information Management System (AIMS), which is the IMA part of the avionics for the Boeing B-777 airplane.

Boeing provided some additional design requirements. One requirement was for the number of days the Boeing 777 could be dispatched without maintenance following a failure. The goal was to allow a plane with a failure in an AIMS component to follow its normal schedule, which will eventually bring it to a maintenance base. This requirement meant both that individual components of the AIMS system had to be reliable and that the system as a whole had to be fault-tolerant. A second design requirement was that the backplane-bus interface not force complexity on the functions in an LRM. Some LRMs might be high-performance processors, but others might be simple hardwired logic. A third requirement, implicit in the notion of an integrated cabinet, was that the design support a multi-processor architecture. In particular, the backplane had to provide adequate net throughput for the initial set of functions together with 50 percent extra capacity to allow for growth. Fourth, the integrity requirements for the avionics system as a whole meant that the backplane bus had to exhibit total fault containment. There had to be less than one chance in a billion per hour of operation that an error occurring within the backplane system would be passed undetected from or to application software. Finally, the design had to be one that would support the certification of the system and the re-certification of modified functions. In particular, the design could not be one that would force the re-certification of all functions when only one function was modified. Honeywell designed SAFEbus because no existing backplane bus met these requirements. SAFEbus builds on the Multi-Processor Flight Control System (M^2FCS) [6] research done for the United States Air Force.

B.1 Outline

The SAFEbus protocol is heavily dependent on its hardware. In order to understand the SAFEbus protocol, one needs to know the basic building blocks of the hardware architecture. The SAFEbus interface logic within each LRM uses paired hardware; each half of the pair consists of a Bus Interface

Unit (BIU) ASIC, a Table Memory, an Intermodule Memory (IMM) and Backplane Transceivers. This logic is paired to provide immediate fault detection and containment. The backplane bus lines are configured in a unique fault-tolerance topology that lies somewhere between quad redundancy and dual-dual redundancy[7]. This topology simultaneously provides high integrity and availability (see Figure B21). SAFEbus consists of two Self-Checking Buses (SCBs), A and B, called bus pairs. Each SCB is itself composed of two buses, x and y. Figure B22 presents a nomenclature of the different bus-related items. One of the BIUs in an LRM transmits data on one of the busses in each SCB, and its partner BIU transmits on the other bus within each SCB. The data on any two busses which come from different BIUs are compared at each receiving LRM. Only bit-for-bit identical data are written into the Intermodule memories. A transmitting LRM checks its transmission using a local loopback. That is, the receiving circuitry in the transmitting LRM also checks what is actually put on the bus for errors. Such self-checking ensures a babbling LRM will be self-detected and will remove itself from SAFEbus. This removal is enforced within an LRM by having each BIU control the other BIU's drivers. If either BIU thinks it should not be transmitting, neither BIU can transmit. Each bus consists of three wires, two for data and one for clock. Thus, the entire SAFEbus set of buses uses a total of 12 wires. The data is transmitted synchronously, two bits at a time, at 30 MHz (for throughput of 60 Mb/s). SAFEbus uses the "wired OR" capable Backplane Transceiver Logic (BTL) defined in the IEEE 1194 standard. The wired-OR drivers and the use of extended clock pulse widths allows SAFEbus to do some out-of-band signaling that support some features of the protocol.

The bus time is divided into "windows" of various sizes. Each window is sized to contain either one message (of from 1 to 256 32-bit words), or a synchronization pulse, or some fixed idle time. A set of static cyclic schedules define the sequence of windows, the size of each window, which LRM(s) may transmit during each window, and which LRM(s) may receive a message from the window. Each window ends with a small, fixed intermessage gap time. A typical intermessage gap is 2 to 4 clock periods. Fixed idle time may be inserted to adjust the duration of the full cycle or the time between messages, to a precision of 30 ns.

Messages that are to be transmitted or have been received over the backplane are placed in buffers in Intermodule memories, which are pseudo dual port memories shared by the host(s) and the BIUs. This organization permits a simple host interface, because all the hosts can view SAFEbus as a shared multi-port memory.

B.2 Protocol Services

B.2.1 Communication Services

SAFEbus provides data communication with very low jitter (on the order of 100 ns) that is fail-op / failsafe, even with a Byzantine failure.

The data for a message that is to be transmitted over SAFEbus is calculated typically by independent self-checking pair Hosts. As with all self-checking pairs, this data is bit-for-bit identical in the fault free case. The Hosts write this data into buffers in the Intermodule Memories. Associated with each buffer is a Buffer Control Word (BCW). Whenever a host completes assembling a message buffer in the Intermodule Memory, it sets a bit in the BCW to say that the buffer is ready for transmission.

The SAFEbus protocol is driven by a sequences of commands stored in the BIUs' Table Memories (see Section B.2.18). Each command corresponds to a single window on the bus. The command indicates whether the BIU should transmit, receive, or ignore the message in that window. The BIUs in every LRM on SAFEbus are synchronized to equivalent points in their respective tables

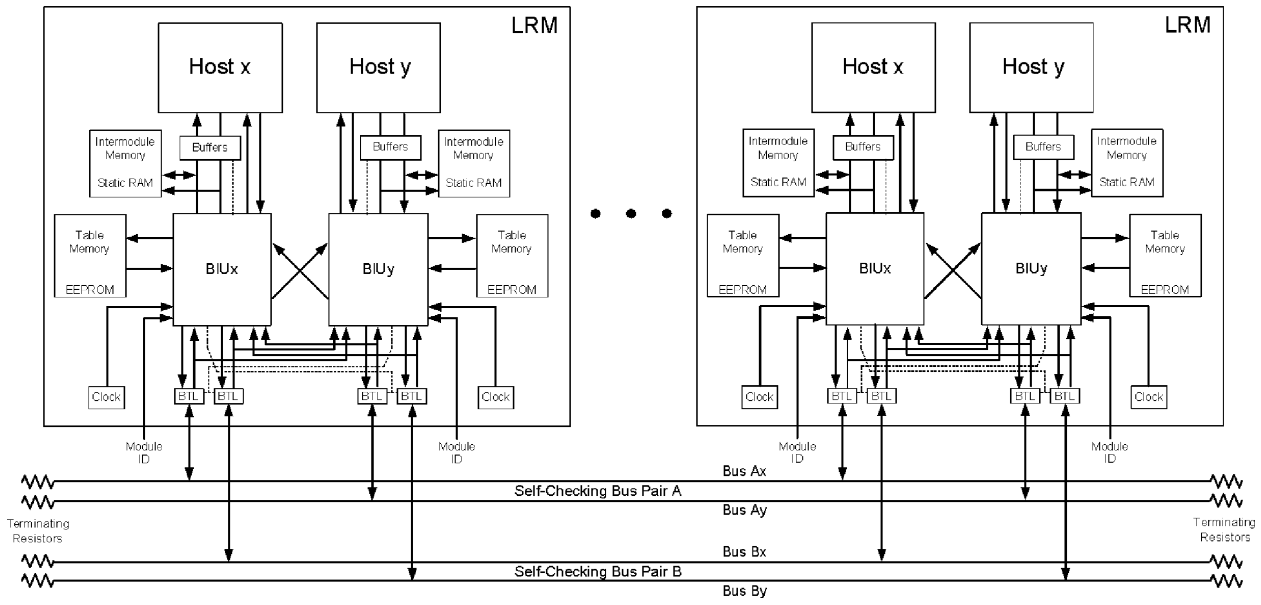


Figure B21. SAFEbus Interface Logic

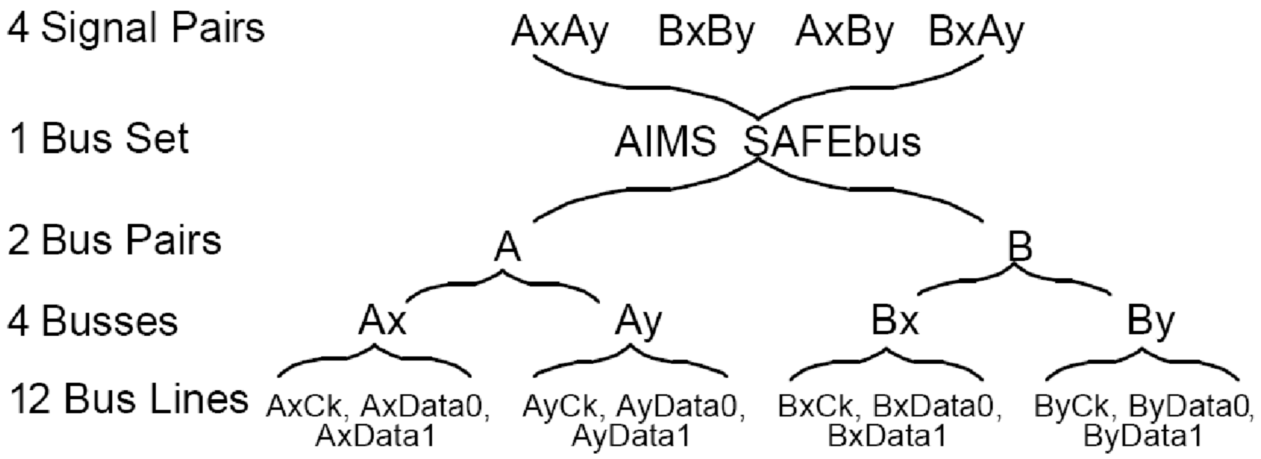


Figure B22. SAFEbus Nomenclature

Term	Explanation
BIU	Bus Interface Unit
Command	Defines what a BIU does during a Window (TX, RX, skip, sync, send interrupt to host)
Frame	A cyclical repeating sequence of windows (NB: in SAFEbus and avionics generally, "frame" refers to a repetition of an execution cycle; whereas, in non-avionics data communication, "frame" is often roughly synonymous with "message")
IMM	Inter-Module Memory (shared message buffer memory between the BIU and its host)
Gap	The constant reserved minimum idle time between transmissions on the bus
LRM	Line Replaceable Module (a node on SAFEbus)
Message	A single, unique transmission of data on SAFEbus; has a length (in words) fixed at design time
Table	One or more sequences of commands controlling one or more frames plus their resync jump points, and a BIU Configuration area
Table Memory	The nonvolatile memory that stores one or more tables
Window	The bus time reserved for a message, sync pulse, or idle plus the trailing intermessage gap; has a duration (in bit times) fixed at design time

Table B2. SAFEbus Terminology

and mechanisms are provided to quickly attain synchronization if it is ever lost. The tables also contain the local address (in the IMM) of the data to be transmitted or received. The commands in each BIU's Table are organized into multiple frames. Each frame controls a repetitive sequence of windows which has a fixed total period.

According to a schedule stored in its Table Memory, each BIU checks their Intermodule Memory for the buffer assigned to the next time window. If that buffer's BCW says the buffer is ready for transmission, the BIU begins pre-fetching the message from the Intermodule Memory. An LRM broadcasts its messages on the four buses when it is scheduled to do so. The two BIUs in an LRM are sync'ed to within 2 bits of each other via the SAFEbus protocol on the busses (no "backdoor" sync between them). Because a BIU doesn't know if it is faster or slower than its partner, it turns on its partner's drivers two bit times before its first bit transmission and off two bit times after its last bit transmission for each message. Off-line scheduling ensures that messages from different sources never collide, taking into account worse case clock drifts, metastability behavior, resync intervals, read timing errors, LRM positions along the bus (the speed of light does make a difference),... Time windows for messages can be set up such that up to four LRMs share a time window in a Master/Shadow arrangement, using a mini-slotting scheme to arbitrate for that window.

Receiving BIUs write validated input data into buffers within each of their Intermodule Memories. At the completion of a message, the BCWs associated with each buffer are updated with status that includes whether the buffer has valid data and what time it arrived. The fact that the BIUs are synchronized to protocol time allows each of them to independently write identical BCW timestamps into each one of their Intermodule Memories without doing an exchange of the received time between the BIUs.

One of the benefits of the table-driven protocol is extremely high efficiency. Control applications typically generate short messages, and most serial protocols perform poorly when messages are short. Efficiencies of between 10% and 30% are typical. In contrast, the SAFEbus protocol is over 89% efficient for a continuous stream of 32-bit messages. Ethernet messages with the same payload would be less than 5% efficient. Because buffer addresses are kept in the tables, they do not need to be transmitted on the bus. The use of transmit and receive commands in the individual tables eliminates the need to send source or destination LRM addresses. Within a message, all clock periods contain data (zero overhead). And, because transmissions are scheduled, no transmission time is consumed arbitrating between contending BIUs (with the rare exception of the optional use of Master/Shadow that typically consume about 9 bit times per arbitration).

B.2.2 Determinism and Partitioning

SAFEbus' determinism and support for robust partitioning warrants more detailed examination, since no other protocol provides these features to this extent. When a system has of functions with different levels of criticality, the functions must be partitioned both in space and in time. Space partitioning means that it no function can prevent another from obtaining adequate memory space and that the memory space assigned to one function cannot be corrupted by the behavior of another function. Memory-management units (MMUs) are usually adequate for simple uniprocessor main memory. But, problems arise with multi-port memory (including network interfaces) and "memory" that must be shared (CPU registers, cache, I/O registers). Time partitioning means that one function's demand for shared hardware resources will never prevent another function from obtaining a specified level of service and, more importantly, that the timing of a function's access to these resources will not be affected by variable demand or by failure of another function.

Any protocol that includes a destination memory address in a message is a space-partitioning problem. It is extremely difficult to verify correct address usage in a partitioned multiprocessor. To ensure correct usage, the BIU would have to duplicate the typical processor MMU function. Then, a difficult protocol would have to be implemented to ensure all BIUs used the same MMU information.

Protocols that use contention arbitration cannot be made strictly time-deterministic. Such arbitration is meant to ensure that when two modules contend for the bus, the one with the highest priority request is granted access. But minor jitter in the execution of functions can change which modules contend for the bus on any given bus cycle. As a result, the order in which the modules obtain access can vary from one arbitration to another in ways that cannot be predicted at design time.

SAFEbus achieves both time and space partitioning by placing all message location (IMM) and bus-timing information in its Table Memories that are frozen at design time. This Table information is held in the BIUs' Table Memories where it cannot be corrupted by any errant software or communications errors. The contents of these memories can be changed only by a very well guarded interface, see Section B.2.15

To extend SAFEbus' time determinism to include the functions' software, the software execution can be synchronized with the execution of the commands in the bus Table. Thus, the software is at the same point during the same bus transmission window in every frame. One benefit is that message latencies can be reduced to insignificance. Results can be scheduled to be transmitted just

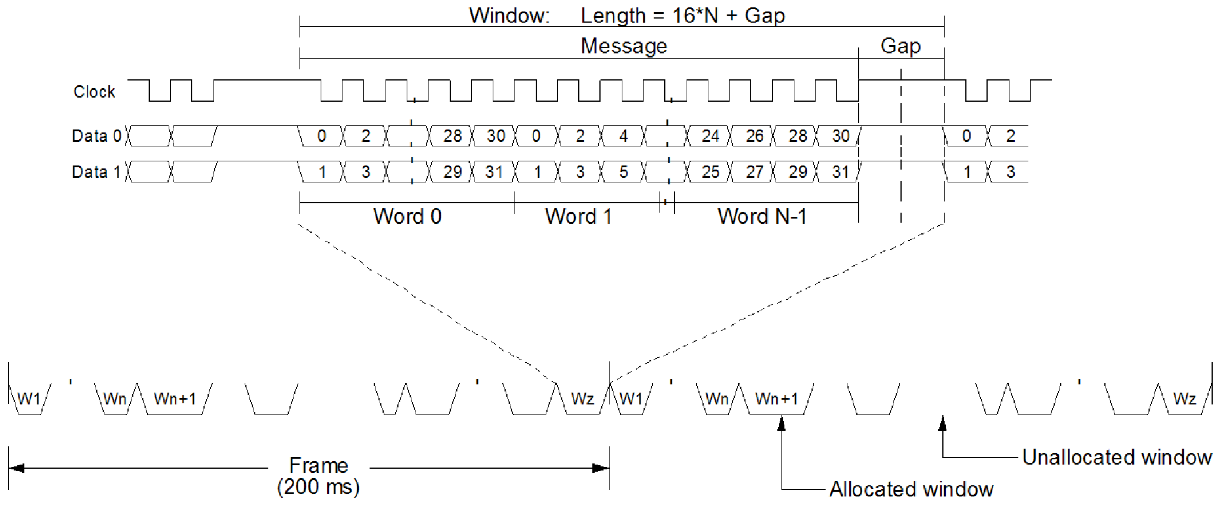


Figure B23. Basic Message Structure

after they are generated and input data can be delivered just before it is needed (software never has to ask for input data to transferred over SAFEbus). A second benefit is that there is less latency jitter on cabinet outputs, which means that a SAFEbus IMA can be used in tighter control loops. A third benefit is that double buffering is rarely necessary because it is possible to schedule the transmission of a data block for a time when it is known that software will not be reading it or modifying it. The elimination of double buffers means the Intermodule memories can be smaller and memory access faster. While the use of software synchronized to the bus and single buffers is the preferred operation, SAFEbus does allow for asynchronous software and double buffering.

B.2.3 Data-Message Structure

There are two data-message types: Basic and Master/Shadow. The Basic message structure has been chosen to maximize the efficiency of data transmissions. The Master/Shadow structure supports data transfers by redundant or aperiodic functions.

Basic Message Structure Basic messages have a simple structure (see Figure B23). Each message consists of a string of 1 to 256 32-bit data words followed by a programmable intermessage gap of two to nine bit times. The Master/Shadow mechanism allows LRMs or applications to be reconfigured or spared without disturbing the traffic pattern on the bus. Master/Shadow windows are identified by a field in the associated Table command. As many as four transmitters can be assigned to one Master/Shadow window. Time-slot arbitration determines which of the transmitters actually gets control of the window. If the Master is alive and has fresh data to send, it starts transmitting at the beginning of the window. The first Shadow begins transmitting "delta" bit times into the window, only if the Master did not use its opportunity to transmit. The second Shadow begins transmitting two delta bit times into the window, only if the Master and the first Shadow did not use their opportunities to transmit. Finally, the third Shadow begins transmitting three delta bit times into the window, only if none of the other candidate transmitters use their opportunities to transmit. Delta is a programmable value that is typically set at one bit time larger than the selected intermessage gap (values from three to ten bit times may be selected). The selected value depends on the propagation characteristics of the backplane. Examples of the transmission over SAFEbus when the Master or third Shadow transmits are shown in Figure B24.

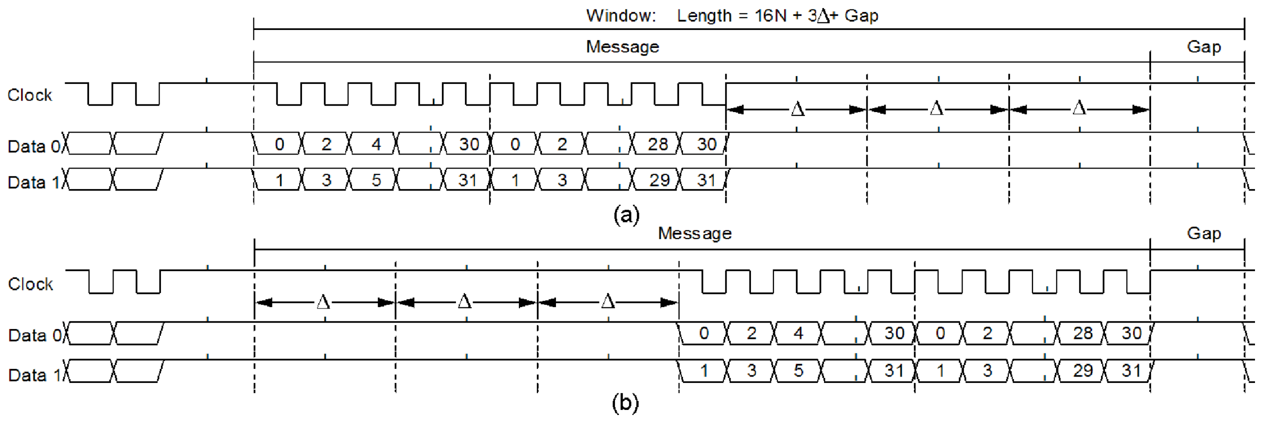


Figure B24. Master/Shadow Message Structure: (a) Master transmits (b) Shadow 3 transmits

Time-slot arbitration could re-introduce non-determinism, but strict measures have been taken to eliminate this danger. First, extra bit times in the window and a restriction on the size of the message guarantee that message transmission will be completed within the assigned time window, no matter what happens during arbitration. Thus, the time window remains the same size no matter which transmitter "wins" the arbitration. Second, recipients of a Master/Shadow message always place the data in the same memory location, no matter which transmitter wins the arbitration. Third, delta can be made large enough to guarantee that the candidate transmitters will never mistake a busy bus for an idle one and begin transmitting in error. Fourth, recipients of a message from this window will be alerted to the presence of this message at exactly the same time after the end of the window, regardless of which LRM wins the arbitration. This allows completely transparent redundancy among the Master and the Shadows.

The Master/Shadow mechanism also can be used for sharing bandwidth among asynchronous functions. Of course, partitioning is not maintained in such Windows.

B.2.4 Bus Encoding

To improve error-detection coverage, data on the four SAFEbus serial lines are encoded in four different ways. Data on Bus Ax have normal polarity. Data on bus Bx are inverted. On bus Ay, every other bit is toggled, starting with the second bit. Bus By is the inverse of bus Ay. This is illustrated in Figure B25. This encoding can be seen as a form of "encryption" in which the four buses Ax, Ay, Bx, and By are XORed with the running keys "0000...", "0101...", "1111...", and "1010..." respectively [15].

One type of failure that this encoding catches is bus-to-bus shorts. With identical data on all four buses, a short between the buses could not be detected until another failure occurred such that the failure propagated from the bus with the second fault to the bus with which it was shorted. Having a first fault lay dormant for indefinitely long periods of time and then to make its appearance exactly when a second failure occurs, cannot be tolerated. This problem is largely ignored by homogeneous fault-tolerant architectures. One way to mitigate this problem is to do periodic scrubbing of the buses by putting different data on the buses that would ordinarily be the same and see that they actually are received as different. There are a number of problems with this approach. The scrubbing has to disrupt normal communication and it has to disable some of the fault tolerance features. The latter is dangerous and must be invoked only with a complex set of interlocks. Because of these complications, describing is not continuous and there

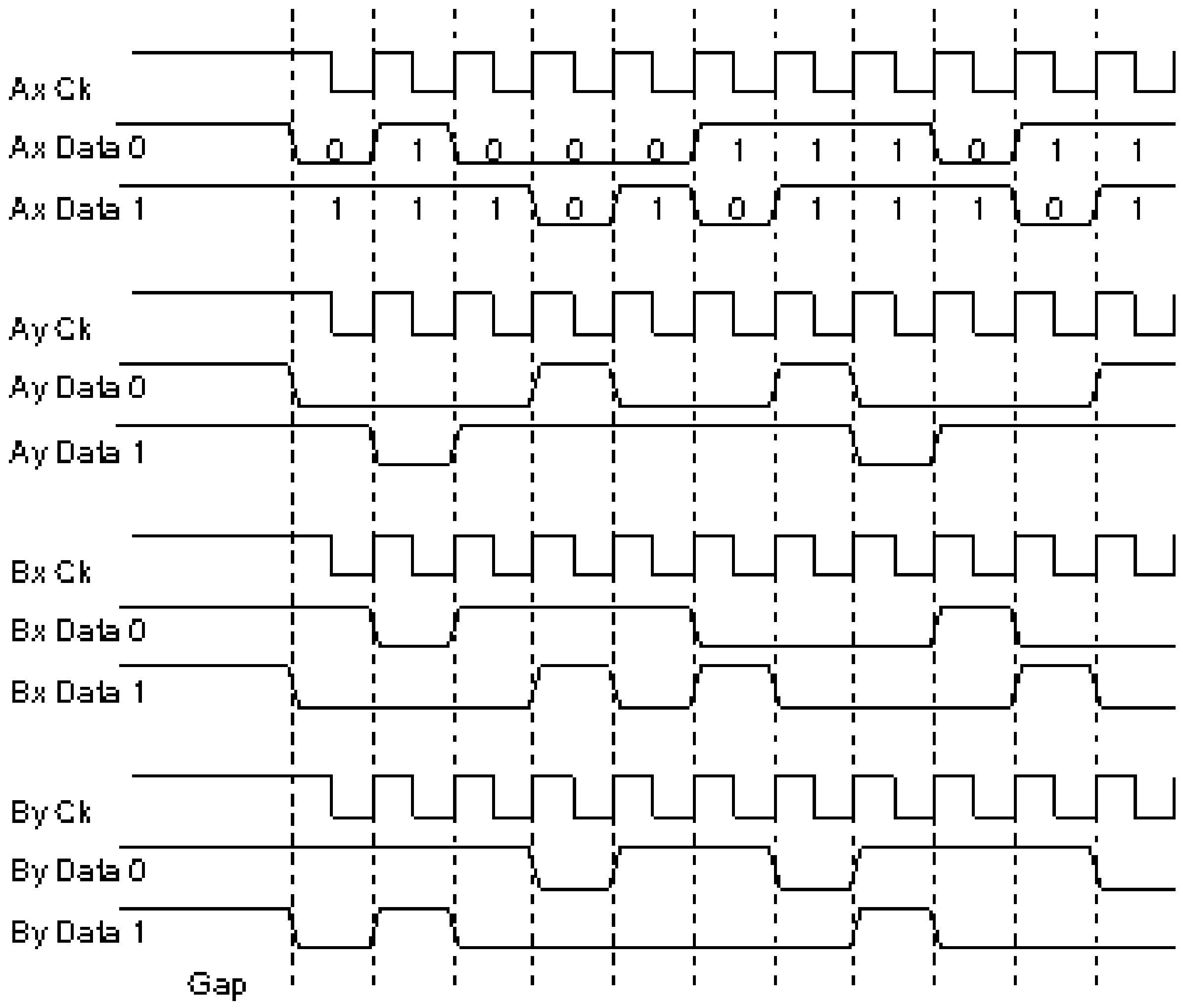


Figure B25. Bus Encoding Example

is an engineering trade-off between the structuring the protocol of scrubbing vs. exposure time to these latent faults. The use of this bus encoding detects these shorts within two bit times of the short onset and without requiring scrubbing.

This encoding scheme can detect unipolar transient upsets that affect several data lines simultaneously. It also allows quick detection of bus collisions caused by a malfunctioning BIU pair (the specific dual-fault scenario that is beyond the basic SAFEbus fault hypothesis). Because bus lines are "wired OR," if a faulty LRM tries to transmit at the same time as another LRM, illegal encodings appear within two bit times of the LRMs starting to transmit differing data.

An additional virtue of this encoding scheme is that power consumption is independent of the data being transmitted. Two bus lines are always high and two are always low (constant average DC power). When the data change, two of the buses change state and two do not (constant average AC power). Because power consumption is constant, the power supply does not have to be designed for a worst-case data pattern. The fact that bus pair A and the B are inverses of each other provide some of the characteristics of differential signaling, without an additional doubling of the number of signal lines required.

B.2.5 Out-of-Band Signaling Pulses

Because SAFEbus messages are pure data (no message-framing overhead) and all data values are used, the only way that protocol specific information can be conveyed is through the use of out-of-band signaling. One method SAFEbus has for out-of-band signaling uses the clock lines. For data transfer, the clock lines are alternating low for one half bit time and high for one half the time (as depicted in Figure B23). For out-of-band signaling, the clock lines are driven low for four bit times, creating a uniquely identifiable pulse, called the Sync Pulse. There are four possible variants of this signal depending on the values of the Data0 and Data1 lines during this pulse. Two of these variants are used for synchronization, one of them is used for a debugging mechanism (see Section B.2.9), and one is not used.

The other SAFEbus out-of-band signaling method drives data lines low while the clock lines remain high. Of the three variants that are possible with this method, SAFEbus only uses one. It drives Data0 low to indicate Initial Resync (see Section B.2.7).

B.2.6 Clock Synchronization

SAFEbus' Long Resync and the Short Resync messages both perform precision (sub bit time level) clock synchronization. The purpose of this synchronization mechanism is to maintain separation of adjacent messages in the presence of oscillator drift and keep the two BIUs in an LRM within two bit times of each other. Since the mechanism is the same for both and the Short Resync message is simpler, only the Short Resync message will be discussed here. The additional functionality of the Long Resync message is described in Section B.2.7. The Short Resync message as shown in Figure B26. It consists of the Sync Pulse on the clock lines and both data lines being high. To provide availability, all LRMs transmit the Sync Pulse. The multiple drives are combined into a single pulse by the "wired OR" action of the open-collector BTL drivers. While all LRMs are scheduled to transmit this pulse at the same time, only one needs to succeed. Because of clock drift, each of the LRMs might turn on their drivers at slightly different times. However, these resync pulses happen often enough such that the drift can never be more than the width of the pulse. Thus, this pulse can appear to be from four bit times up to eight times wide. When a BIU sees this pulse, it freezes its internal time, and at the end of the pulse it releases the freeze. It uses an internal effective 4x clock to sample the clock lines. This is shown in Figure B27. Because each BIU's time was frozen, it's time will have fallen behind real-time. The BIUs then enter a catch-up phase where the internal time counters are incremented at a 2x rate until the internal time has caught up with real-time. The duration of this catch-up phase is equal to the freeze duration. This allows all BIUs to have identical times after each resync event without ever causing time to go backwards in any BIU.

The aggregate effect of this resync mechanism is that the slower of the two BIUs in the quickest LRM pace the system timebase. Each BIU maintains a counter (called SAFEbus Time) driven by its synchronization-corrected oscillator. The synchronization mechanisms make the values in these counters identical in all BIUs with respect to the time that protocol events happen (e.g., the receipt of a message). The Time value may be used to time stamp data.

B.2.7 Restart, Re-Integration, Integration

Synchronization States Figure B29 shows the major synchronization states of a BIU and their transitions. The major states are:

Initializing While in this state, the BIU performs such operations as Table Memory CRC checks, BIU Configuration Area loading, and IMM tests. The Full-resolution SAFEbus Time register

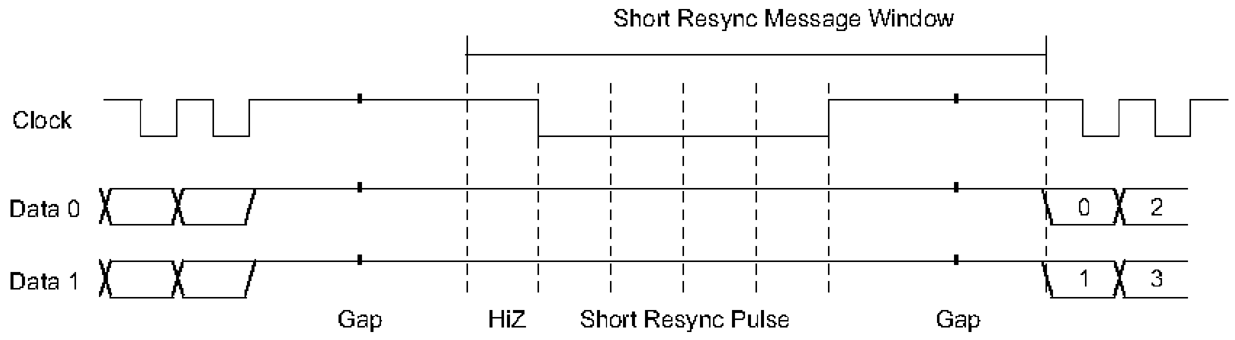


Figure B26. Short Re-Sync Message

value is not valid in this state. IMM access is disabled during much of the initialization process, while the BIU performs IMM pattern testing.

Out_of_Sync While in this state, the BIU hunts for resynchronization messages transmitted over SAFEbus, and attempts to synchronize with them if they are present. If enough time elapses without a synchronization message being seen, the BIU issues an Initial Sync pulse to start up the backplane. The Full-resolution SAFEbus Time register value is not valid in this state.

In_Sync While in this state, the BIU executes command sequences out of a Table Command Sequence Area in the Table Memory. It transmits when executing an appropriate transmit command (and data is fresh), and receives data when executing a receive command. Synchronization is maintained via the transmission and reception of programmed synchronization messages. The Full-resolution SAFEbus Time register value is valid in this state.

Halted This is a sub-state of the In_Sync state. The BIU enters the Halted state when it is executing in debug mode, and encounters a breakpoint, or completes a single step operation.

Disconnected While in this state, the BIU suspends all transmission and reception activity on SAFEbus. The Host may still read and write memory and BIU registers, but no backplane data will get written into memory. The BIU will enter this state if commanded to by the host, or if initialization fails, or if it receives a Long Resync message with mismatching Version while in the Out_of_Sync state. It leaves this state if commanded by the host or if an Initial Sync Pulse is detected on SAFEbus. The Full-resolution SAFEbus Time register value is not valid in this state.

Synchronization Messages Three uniquely identifiable transmission patterns are provided to support bit-level and frame-level synchronization of the SAFEbus backplane. The Initial Sync Message is used to initialize the SAFEbus after a power-up or in the pathological case of a cabinet-wide loss of synchronization. The Short Resync Message is provided to maintain bit-level synchronization between all BIUs in the cabinet by correcting for oscillator drift between BIUs. The Long Resync Messages are provided to allow lost modules to regain synchronization with an active bus. Long Resync Messages come in two variants. The Entry Resync variant is provided simply to allow lost modules to resync to the current frame. The Frame Change variant is provided to switch between different frame programs in the current Table. Both Versioned and Unversioned forms of the Long Resync Messages exist. Long Resync Messages also implement the bit-level resynchronization operation (as provided by the Short Resync message).

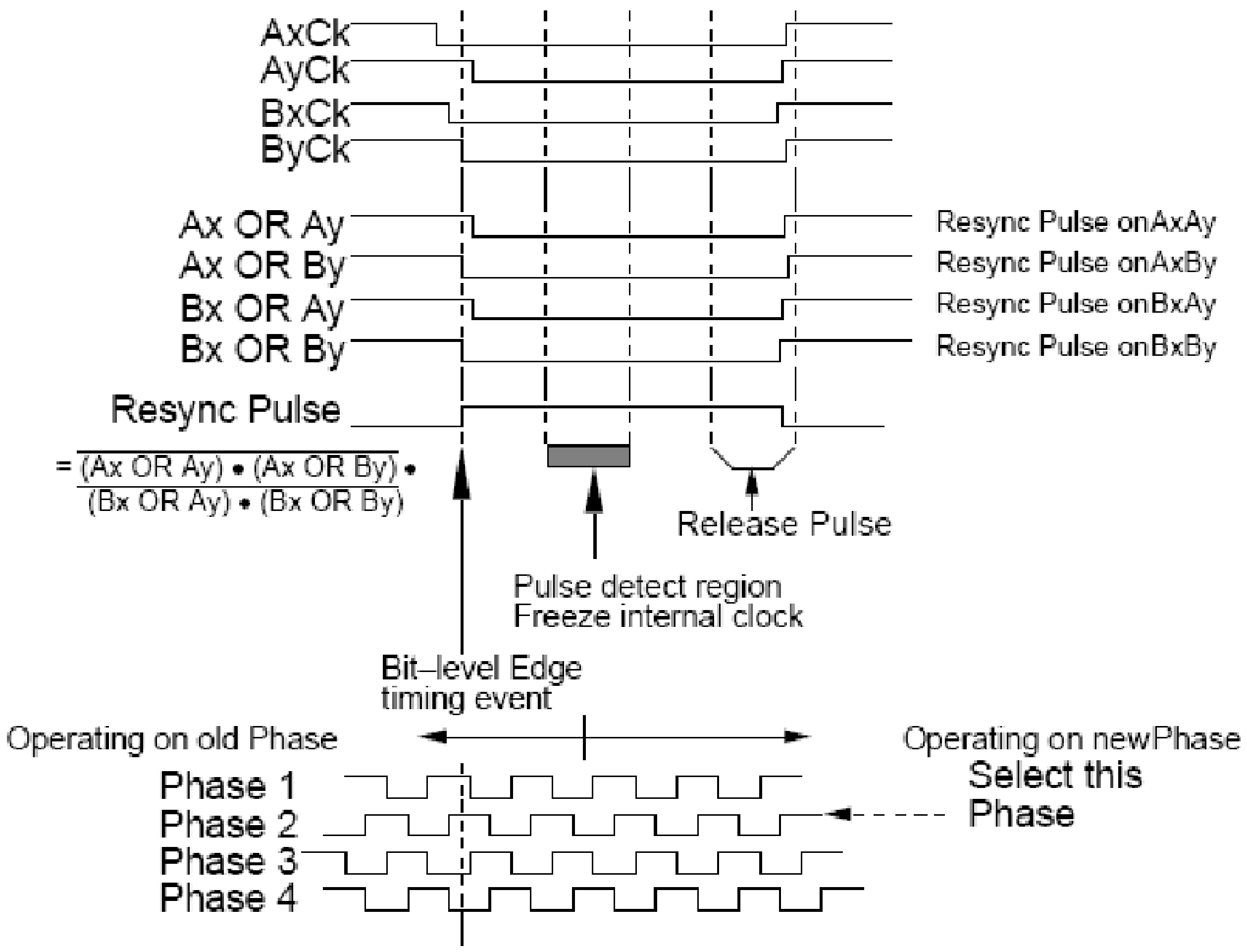


Figure B27. Resynchronization Pulse Timing

Long Resync The structure of the Long Resync Message (Figure B30) is separated into two distinct sub-windows. The Long Resync Pulse sub-window starts with a unique Long Resync pulse identified by a low level on the clock lines with all associated Data0 lines low. The pulse is nominally four bit-times long followed by a Maximum Gap. The Long Resync Pulse is transmitted by all BIUs that are executing either a Transmit or Receive Long Resync command.

The second part of the Long Resync Message is the Long Resync Information sub-window. A single, unique BIU must transmit in the Long Resync Information sub-window. This consists of an 8-bit Resync Code (a value from 0-255), a 1-bit Versioned Frame indicator, 1 bit of reserved space, a 4-bit Cabinet position, 7 bits of reserved space, 43 bits containing SAFEbus Time, and a 32-bit Table Version. The Resync Code allows the BIU to determine which of 256 locations in the Table it should jump to in order to align itself with the other BIUs. In software, this would be called an indirect jump. The code indexes into a Resync Jump Table (see Figure B33) which contain addresses that point to the main Table Command Sequence Area command that should follow this Long Resync message in the normal sequence of command execution. The Version Frame indicator, Cabinet position, and Table Version are used as part of versioning enforcement as described in SectionB.2.20. The SAFEbus Time field is used to supply this information to LRMs receiving this message while not in sync.

Long Resync messages can be sent Master/Shadow, which combines the properties of both these

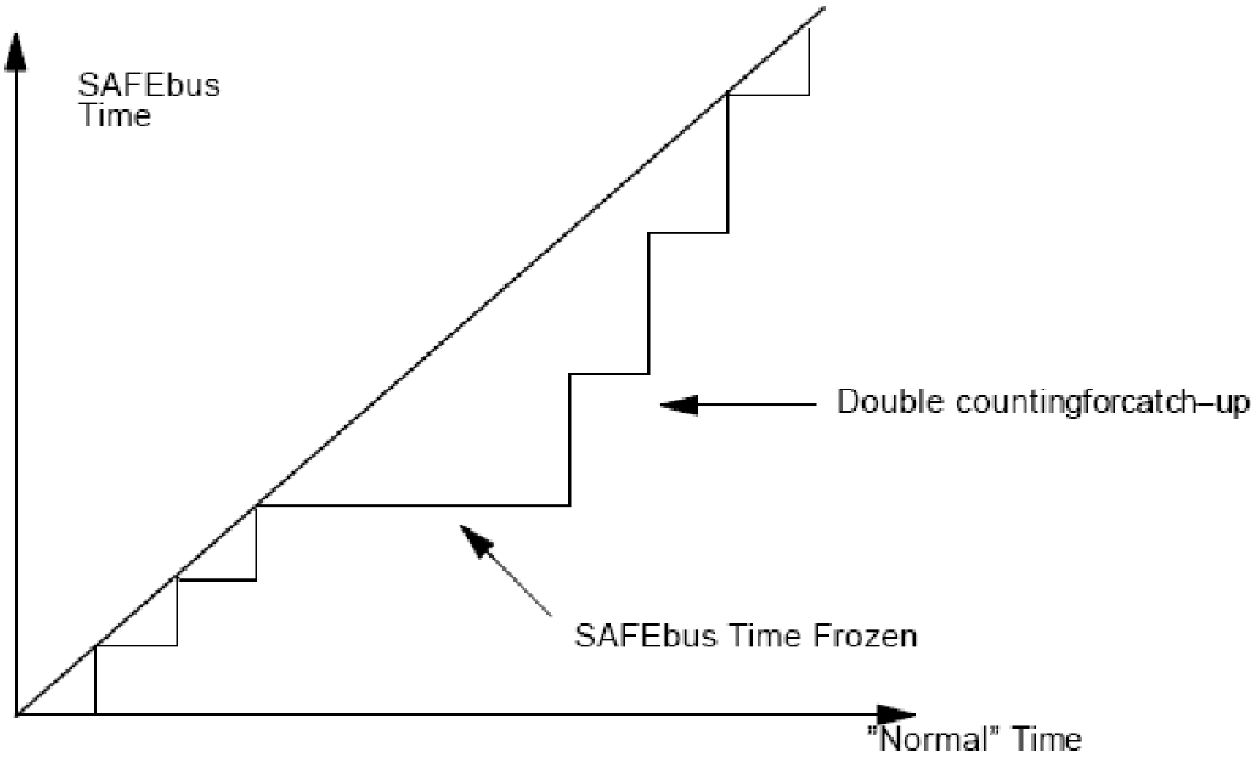


Figure B28. Time Adjustment

message types. This benefits the multiway trade-off among fault tolerance, bandwidth used, and resynchronization latency.

Frame Change A variant of the Long Resync message is the Frame Change message. The message form is identical. The differences are in the way that it is used. In the Long Resync message, the Resync Code points to next command in the normal sequence. In the Frame Change message, the Code points elsewhere, usually to another frame. The Frame Change is a conditional jump. An LRM which is scheduled to transmit a Frame Change cannot do so unless its host writes a code into the BIU which matches the Frame's code (a "lock and key" mechanism). A tightly controlled mechanism is provided to switch between frames based on explicit commands generated by Level A OS software using the "lock and key". Normally, "lock and key" fault-containment mechanisms are very weak. However, in the case of SAFEbus, this mechanism is protected against hardware failures by the use of self-checking pair hardware and it is protected against software failures through use of an MMU which limits the lock's access only to the Level A software.

If the Fame Change message is not transmitted, the "jump" is not taken and the command/window sequence continues with the command which is next after the Frame Change in the table. If the Fame Change message is transmitted, the "jump" is taken and all LRMs fetch their next command from the location pointed to by the Frame Change's Resync Code. Use of this feature is described in Section B.2.16.

Initial Sync Initial Sync messages (Figure B31) are transmitted by an Out_of_Sync BIU, that waits longer than the Initial Sync Wait Limit without seeing any Resync Pulses on the backplane. All BIUs which are out of sync at this point will use the Initial Sync message to synchronize into

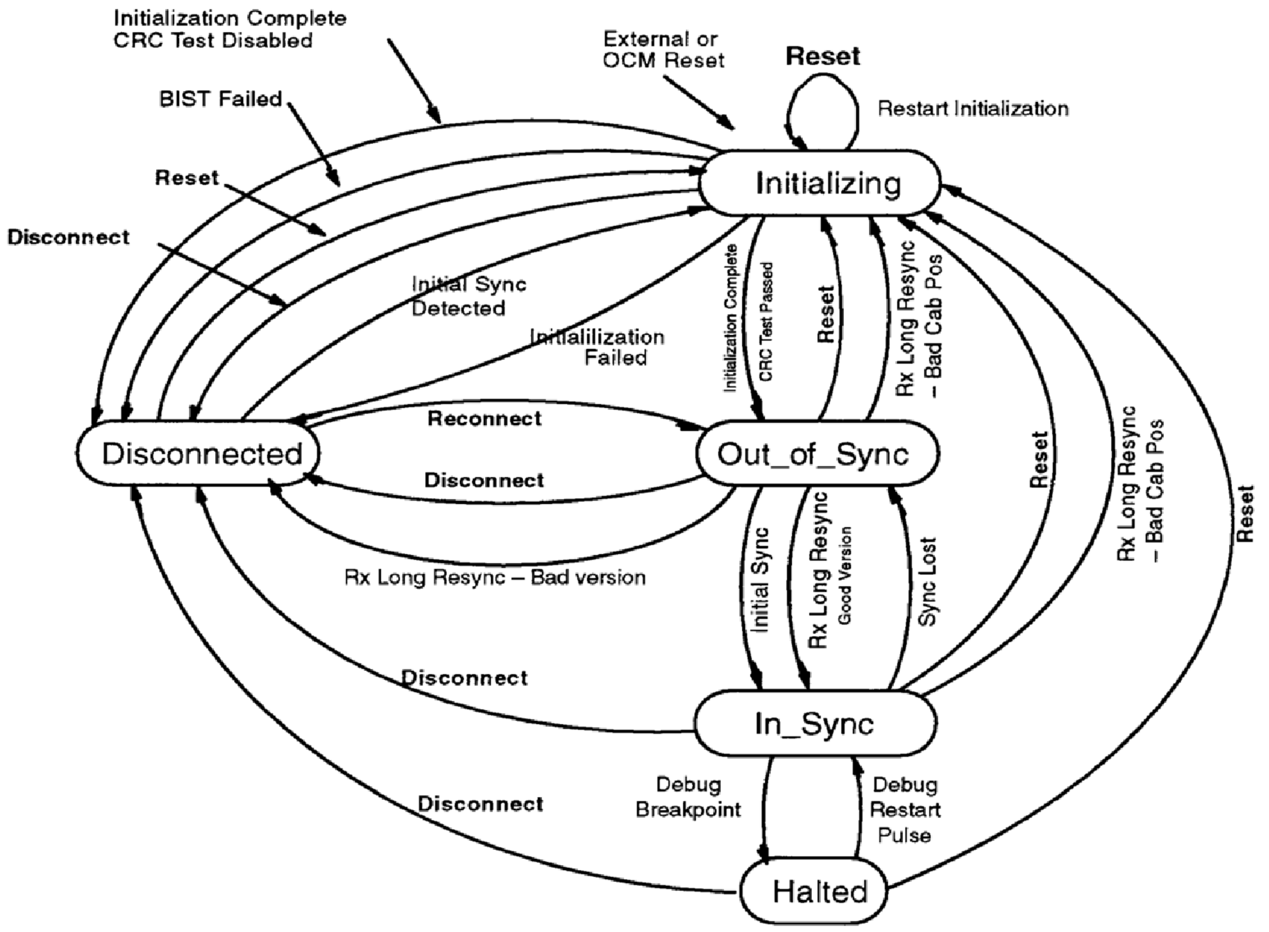


Figure B29. Synchronization State Diagram

an Unversioned Initial Frame. The Initial Sync message starts with the Initial Sync Pulse. It has a unique pattern identified by a signal pair of busses with a low level on the Data0 line and a high level on the clock line for at least 2 bit times. In addition, for a bus to be considered part of an Initial Sync Pulse signal pair, the Data0 line must have been high at one point during the time while the BIU waited for the Initial Sync Wait Limit. The pulse is nominally four bit-times long.

To more accurately resync the clocks, and to allow the BIU time to fetch the command for the first message in the Initial Frame, the Initial Sync Pulse is followed by a Long Resync message with no BIU transmitting the Information sub window. The missing information is assumed to be zero. In particular, the Resync Code is zero and the SAFEbus Time is zero. The BIU will enter the In_Sync state at the first bit time of the Gap which separates the idle data portion of the Long Resync message from the first window of the Initial Frame. The first bit time of the Initial Frame occurs (106 + 6Maxδ + 2MaxGap) bit times after the leading edge of the Long Resync pulse which follows the Initial Sync Pulse.

B.2.8 Diagnostic Services

SAFEbus uses masking fault tolerance, which does not need the complicated diagnostic services required by other protocols, e.g. it does not need to keep track of membership. However, it does supply a rich set of diagnostic mechanisms and information for maintenance purposes. This

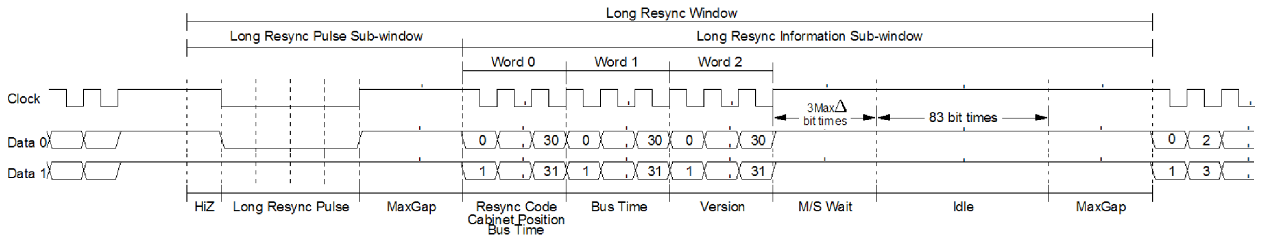
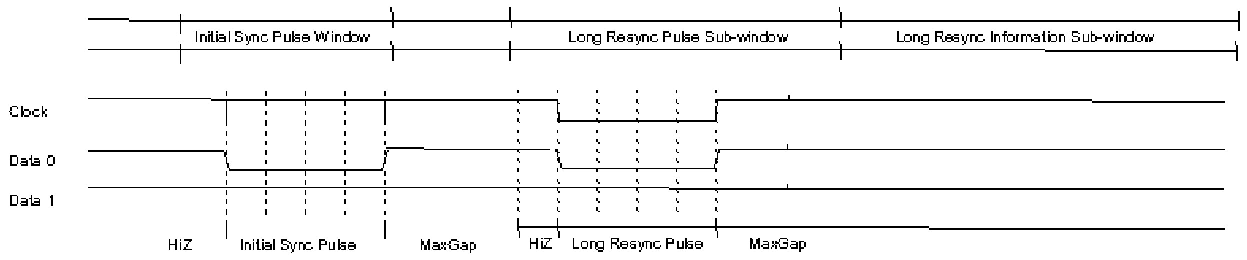


Figure B30. Long Re-Sync Message



Initial Sync Pulse Window
Long Resync Pulse Sub-window
Long Resync Information Sub-window
Clock
Data 0
Data 1
HiZ
Initial Sync Pulse
MaxGap
HiZ
Long Resync Pulse
MaxGap

Figure B31. Initial Sync Message

includes full BIST capability via dual (x and y) IEEE 1149.1 (JTAG) test busses. The x JTAG bus connects only to the x BIU and the y JTAG bus connects only to the y BIU in order to prevent fault propagation from the two sides of the self-checking pair. For commanded BIST, the BIU goes off line and scans a set of pseudo-randomly generated test vectors into the logic, clocks it through the logic and accumulates the result of the clocking into the BIST Signature Result register. The BIST also includes scrubbing of all the critical fault-tolerant circuitry within the BIUs. For scrubbing protection mechanisms, hardware is invoked to test if the protection mechanism is really invoked when a failure occurs.

B.2.9 Debugging Mechanisms

Another unique feature of SAFEbus is its ability to breakpoint and single step an entire system, including the processors connected to SAFEbus. This is the same as the breakpoint and single step functionality commonly seen in software debuggers, but on a systemwide basis. The reason this is possible is because SAFEbus acts as the central clock for the system, providing the timing ticks that the operating systems of the processors attached to it use for dispatching tasks. If the SAFEbus freezes, so do all of the processors.

A breakpoint can be set for any time within the global SAFEbus timeline. A breakpoint can also be initiated by driving the Ck, Data0, and Data1 lines low simultaneously for at least four bit times. To resume from a breakpoint, some LRM transmits a Short Resync pulse.

Because setting a breakpoint can be dangerous during normal operation, breakpoints are only enabled for special test table versions. This is indicated by the upper two bits of the Table Version being “11”.

B.2.10 Fault Isolation

SAFEbus and its self-checking approach provides near perfect coverage. The checking at the receiving end provides near perfect error detection coverage for many faults, including Byzantine faults [13]. It provides better coverage than signature-based error detection techniques (such as CRCs) [16] while simultaneously not incurring the overhead of these schemes.

Recapping Figure B21: SAFEbus consists of four buses (Ax, Ay, Bx, and By) that connect several self-checking pair LRMs. Each of the four buses and each half of an LRM are independent fault containment zones. If there are N LRMs, there are 2*N + 4 fault containment zones. Each of the BTL bus interface parts is in its associated bus' fault containment zone and gets its power supply from that bus. Thus, the fault containment zone boundaries are between the BTL parts and the BIU parts. Names ending in x and y denote redundant parts for integrity. Bus names beginning with A and B denote redundant bus pairs for availability. The BTL drivers are connected such that BIUx transmits only on Ax and Bx; and, BIUy transmits only on Ay and By. Thus, a faulty BIUx can contaminate only buses Ax and Bx; and, a faulty BIUy can contaminate only buses Ay and By.

The dashed lines in this diagram are control signals. In particular, BIUx enables BIUy's bus drivers and BIUy enables BIUx's bus drivers. Thus, an LRM can't transmit unless both BIUx and BIUy agree to do so. The set of busses is fail-op, fail-stop. Each LRM is fail-stop. N redundant LRMs are N-1 fail-op, fail-stop. The major failure scenario that SAFEbus does not cover is two simultaneous active faults in the same LRM that are somehow complementary to escape the bit-for-bit checking and cross-coupled driver enables. This has a probability that has been calculated to be much less than 10^{-10}.

B.2.11 Babble Protection

To detect errors, the transmitting LRM checks what it actually puts on the bus. If a BIU sees a miscompare, it stops transmitting and it disables its partner's drivers. The dual nature of this comparison ensures that a babbling module cannot stay on the bus. This is an availability feature rather than an integrity feature. Integrity fault containment is done by the receivers. This availability feature is only applicable to Master/Shadow Windows. In order to prevent loss of resources due to transient failures, LRMs are allowed to restart for a limited time by implementation of a strike counter. This strike counter is incremented for each fault found in a specific interval (say a maximum of three failures are allowed within ten minutes). This ensures that transient faults are dealt with in a constructive manner and permanent or intermittent faults are isolated after the LRM has hit the strike counter limit.

B.2.12 Byzantine Protection

SAFEbus has a unique way of tolerating Byzantine faults. Because the transfer of a message from one LRM to another LRM uses 4 fault zones, it is possible for it to tolerate one Byzantine fault. The BTL receivers are cross-linked to the two BIUs such that each receiving BIU gets a copy of the message from all four buses. This can be seen as the first round of the classical Byzantine exchange. Each BIU creates two 4-bit status vectors, collectively called the "syndrome", for each 16 bits received within a message.[B2]. The first vector has a bit for each bus saying whether anything came in from that bus. The second vector is the result of the comparisons: Ax = Ay, Bx = By, Ax = By, Ay = Bx. The BIUs exchange their syndromes. From these 8 bits, the two BIUs can determine which (if any) of the data bus inputs have arrived error free. If there is such an error-free source, both BIUs selected it as the source data. This can be seen as the second round of the

[B2]The granularity of 16 bits was chosen as an engineering trade-off between the desire for a smaller size to increase availability and the desire for larger size to minimize metastability errors and to make it easier to meet timing constraints on the exchange between the BIUs classical Byzantine exchange. This prevents Byzantine failures arriving from outside a pair from confusing a pair into thinking that one of the halves of the pair is faulty. If a message arrives with uncorrectable errors, the BCW associated with its buffer is updated to that status. A self-checking pair host reading a BCW status that indicates an error is not allowed to read the data buffer because differing data in the two Intermodule Memories could cause the host pair to split.

While the syndrome exchange prevents a Byzantine fault from splitting a pair, an additional mechanism is needed for Byzantine agreement among pairs. Before SAFEbus, Byzantine algorithms either did a full exchange of an entire message's content or used signatures. The problem with the former method is the large amount of bandwidth it requires. The problem with the latter method is that it does not provide full coverage. SAFEbus introduced a new method: hierarchical Byzantine agreement. In this method, a lower-level agreement prevents Byzantine faults from affecting a pair, as described above. An upper-level agreement only needs to send one bit of information from every receiving LRM of the message. This bit would indicate whether the LRM rejected the message as being faulty or not. Because the SAFEbus granularity is 32 bits, it can send 32 bits just as well as one bit. Using 32 bits allows this upper level of exchange to be implemented with no additional hardware and with no software. All that is done is to have the buffer for this upper-level round of exchange be placed into the IMM such that the data word of this buffer overlaps with the BCW for the original message. Thus, what is exchanged on this upper-level is the BCW of the original message. If the original message was rejected, the BCW is zero.

B.2.13 Availability Vs. Integrity Trade-Off

The syndrome exchange mechanism includes an option to select a preference for availability or integrity, for those cases where there is not a generally applicable best choice. For example, if Ax = Ay and Bx = By; but Ax $\neq$ By and Bx $\neq$ Ay. This can only happen if there are at least two identical bit errors (after decode). A receiver cannot tell if pair A or pair B is correct. For integrity, both have to be thrown out because neither can be trusted. For availability, either A or B could be arbitrarily chosen.

B.2.14 Zombie Module Protection

Each LRM that plugs into SAFEbus must contain a Table of commands that is compatible with all other LRMs on that SAFEbus. One mechanism for preventing LRMs with an incompatible Table would be to have the LRMs exchange Table version information at startup. However, such a scheme would not cover the scenario where an LRM is dead or comatose at startup and then "wakes up" in middle of critical operation with an incompatible Table. This is called the "zombie module" problem. To solve this, SAFEbus requires all LRMs joining a SAFEbus that has active traffic to compare the Table version information in a Long Resync message to its own. It is allowed to join the network only if the versions are compatible. See Section B.2.20 for details on version enforcement.

B.2.15 Configuration Services

The SAFEbus Tables are loaded into each BIU's Table Memory using the same dual IEEE 1149.1 (JTAG) test busses that are used to support BIST. These Table Memory images can contain multiple tables so that one LRM can play several different roles depending on which slot and in which cabinet the LRM is located. This mechanism can reduce the number of spares required to be held and maintenance facilities. For example, the Boeing 777 AIMS cabinets had one I/O module (IOM) design that was used in eight places. This reduced the number of types of spares required from eight down to one, and reduced the total number of spares needed to be held in the logistics pipeline. In theory, this could also allow for on-board sparing and manual reconfiguration. Each slot in every SAFEbus cabinet has five slot ID pins that can allow up to 32 slots per cabinet. To prevent wrong-ID masquerade faults, these pins are protected by parity and cross compare between the two halves of a pair. To eliminate the need for Cabinet ID pins on every slot, SAFEbus allows some subset of LRMs within the cabinet to have these pins and then these LRMs broadcasts this cabinet identification in their Long Resync messages' Cabinet Position field. With the existing four bits of Cabinet Position and five bits four slot ID, SAFEbus can currently accommodate up to 512 slots systemwide. The Long Resync message has seven spare bits adjacent to this field that could be used to expand this to a total of 65,536 systemwide slots.

After the tables have been loaded, SAFEbus can be used to download software and other data. To optimize bandwidth usage, special frames can be used for these downloads (as described in Section B.2.16).

B.2.16 Frame Changes

This section provides an example for frames to do hardware initialization, software initialization, data loading, built-in self test (BIT), and normal application communication; plus, the Frame Change transitions between them. The following is a brief textual description of each frame.

1. The Hardware Initialization (HW Init) frame follows Arinc 659's definition for the Initialization frame and, as such, is an Unversioned frame. The only data traffic during the HW Init frame is the transfer of Version Messages from all LRMs. The HW Init frame is approximately 117 microseconds in duration.

2. The Software Initialization (SW Init) frame provides time for the processors' environment to be set-up before the functional partitions are dispatched in the Flight frame. This allows the environment to be set-up before the functional partitions are dispatched in the Flight frame.

3. The Built in Test (BIT) frame is a Versioned frame that provides time to perform power-up BITE.

4. The Flight frame is a Versioned frame where the software partitions run.

5. The Dataload frame is a Versioned frame that provides the Dataload function, using the majority of the SAFEbus bandwidth. While dataloading may be performed in the Flight frame, a dataloading session using the Dataload frame is much quicker.

Figure B32 shows these frames and possible transitions. Note that the Frame Changes are unidirectional. This means, for example, that once the Flight frame starts executing, it cannot be Frame Changed to any other frame. In order to go from the Flight frame to another frame, the SAFEbus must be reset. This is an additional safety precaution against accidental transitions out of flight frame.

B.2.17 Protocol Parameterization

B.2.18 Table Memory

As shown in Figure B33, a SAFEbus Table is divided into three areas: Resynchronization Jump Table, Table Command Sequence Area, and BIU Configuration Area. The Resynchronization Jump Table is used by the BIU to rapidly locate the address in the Table Memory corresponding to the Resync Code (see Section B.2.7). The majority of the Table Memory is used for the cyclic command

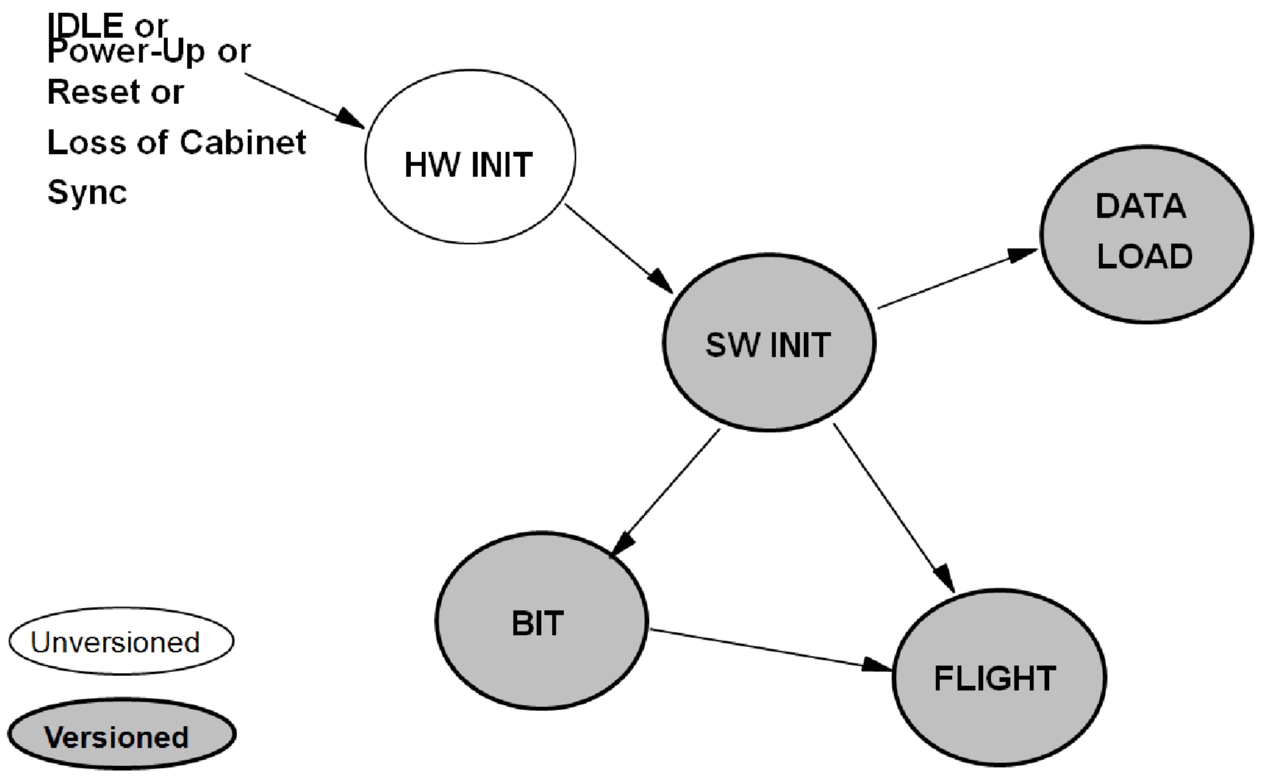


Figure B32. Example Frames and Their Frame Change Transitions

sequences that control frames. More than one frame schedule can be present for different system "modes" (such as system initialization, ground check-out, flight operation, software loading, etc.), with each mode having a different schedule. The different frames for different modes are contained within a single frame. Therefore, this selection is in addition to the selection of different tables depending on location (slot and cabinet) roles. The BIU Configuration Area contains information for BIU customization options such as memory speeds, host interface characteristics, selection of availability or integrity as preferred for those cases where one is not universally preferred over the other, intermessage gap, Master/Shadow delta, and SAFEbus Time increment rate. The contents of the Table Memories which are associated with the two BIUs on a single module are bit-for-bit identical. However, the Table Memory contents are different for each module on SAFEbus. This because the local IMM addresses can be different and the commands are different for TX versus RX, Master versus Shadow, etc.

B.2.19 Frame Description Language

SAFEbus uses a Frame Description Language to define the contents of each frame. This is an intermediary language that can be produced by off-line schedulers. A back-end tool specific to each BIU design translates this FDL into a bit image that can be loaded into Table Memories. This language has been standardized by ARINC 659 to allow decoupling between vendors of scheduling tools and vendors of BIU hardware. For AIMS, a software tool parses a database of ICD information and generates a schedule that has much more freedom than schedulers for most time triggered protocols.