

Net reset and refresh sequences as submitted to the p1394.1 committee

Dr. David V. James, Sony
3300 Zanker Road, MS SJ3C1
San Jose, CA 95134-4591
Phone: 408 955-6295
FAX: 408 955-4591
Email: dvj@alum.mit.edu

October 12 1999

**This contribution is one of several, presented for independent review.
An overall contribution, which provides the context for multiple contributions,
is also provided in BR047R08.**

Techniques for invoking a net reset or net refresh, which updates bridge routes, are needed.
This contribution proposes what appears to be the most robust and best documented alternative.
The working group is requested to confirm this direction or propose a documented alternative.

High performance Serial Bus bridges

1. Introduction

1.15 Bus initialization information

1.15.1 Bus reset information

The net refresh protocols assume that topological information is available at the completion of each bus reset. This information is used to detect changes in the attached nodes, and also identifies its portal's "neighbor", allowing messages can be sent from one neighbor to the next. After a bus reset completes, each bridge portal is aware of the following information:

- 1) Phy lists. A list of local-bus phys (that normally represent nodes) is known:
 - a) Ordered. This list has a topology-dependent root-independent circular ordering. The ordered list is obtained from topology information derived from selfId packets (see xx).
 - b) Classified. Nodes are classified into categories:
 - i) Portal. This node is a bus-bridge portal.
 - ii) Other. This node is not a bus-bridge portal.
- 2) Parsing of the lists allows the bus reset to be classified as follows:
 - a) SAME. The same nodes appear to be attached.
 - b) DIFF. Different nodes appear to be attached.
 - c) MORE. More nodes appear to be attached.
 - d) LESS. Less nodes appear to be attached.

TBD—Define the bridge and bridge-aware bits in the second quadlet's selfId packets.

1.15.2 Next-neighbor ordering

NOTE—The following protocols are physical layer dependent and may be different on other Surreal interconnects.

The net refresh protocols are designed to yield a predictable logical topology for a give physical cable topology, regardless of which nodes are selected to be root. The bus refresh protocols involve communications between portals and their neighbors. To ensure stability, the “neighbor” portal is defined by the topology, independent of phyId assignments, as described in this subclause.

Neighbors are defined by relative ringId values, where each node derives ringId values from the observed selfId packets. The assignment of ringId values is based on a conceptual routing of signals, illustrated in figure 1. In figure 1, the labels ‘a’, ‘b’, ‘c’ identify phy ports ‘0’, ‘1’, and ‘2’ respectively, and these ports have an implied internal ordering, listed below:

```
port[a].in=>port[b].out  
port[b].in=>port[c].out  
port[c].in=>counter=>port[a].out
```

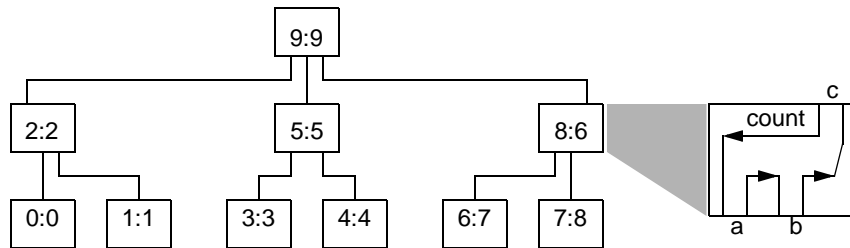


Figure 1—Persistent ringId ordering

Each node assigns conceptual ringId values to the others, starting with ringId=0 on its own port[a] output. The ringId values are assigned by tracing a path through other phys, incrementing the ringId when passing through the port[a] output. This assignment strategy always yields the same next-neighbor selections, despite changes in the selected-root assignment.

1.15.3 Bus refresh messages

The protocols used for bus refresh were designed with the following assumptions in mind:

- 1) Messages. Messages are distributed to other bus-local portals in the following fashion:
 - a) Ordered. The topology-dependent root-independent portal ordering is available after bus reset. This allows each node to circulate messages by sending them in a next-neighbor ordering.
 - b) Idempotent. Messages are idempotent responseless writes, so failures can be simply retried.

After the bus reset completes, each portal sends messages to the next portal, allowing messages to flow in a circular direction, as illustrated in figure 2. In this context, “next” means the portal with the next larger ringId assignment.

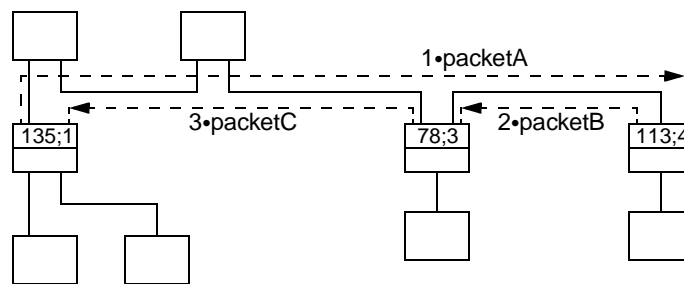


Figure 2—Portal-to-portal message paths

This form of sequential/circular writes is the basis for reliable net refresh communications. Writes that incur errors or busy indications are simply (and immediately) retried until successful. Confirmations are provided by allowing these writes to circulate through other portals until they return to their origin.

The constant sending of responseless writes ensures their successful completion, without mandating special fault-retry protocols. The circular nature of the communication allows the originator of these writes to verify their completion. Such communications are sufficient for reliable broadcasts, but are more flexible because write payloads can be modified as they pass through connected portals.

1.16 Net refresh/reset messages

The protocols used for net refresh were designed with the following assumptions in mind:

- 1) Precedence. A 2-bit precedence allows software to specify prime-portal preferences. This value is prepended to the portal’s EUI to generate a stable primeId identifier. The local-bus portal with the largest primeId identifier is selected to become the net’s prime portal.
- 2) Messages. Messages are distributed to other bus-local portals in the following fashion:
 - a) Ordered. The topology-dependent root-independent portal ordering is available after bus reset. This allows each node to circulate messages by sending them in a next-neighbor ordering.
 - b) Idempotent. Messages are idempotent responseless writes, so failures can be simply retried.
- 3) Selected victims. Normally only a few (victim node) transactions are terminate prematurely.
- 4) Constructive reconnect. A net is minimally disrupted by transient disconnect/reconnect sequences.

1.17 Net refresh

The process of initializing a net topology involves formation of a spanning tree by circumscribing the paths through bus bridge portals, as illustrated in figure 3. Each portal communicates with its adjacent neighbor by writing messages into a standardized CSR location. During the final state of a net refresh, the portal-to-portal messages flow in the direction of the solid arrows.

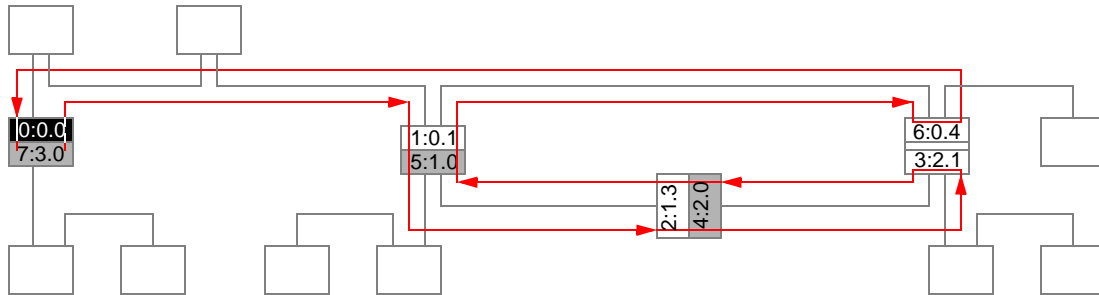


Figure 3—Circumscribed net topology (legend in 2.4.1)

For stability, the node with the largest primeId is selected to become the prime portal; the primeId is a concatenation of a 2-bit preference number and the portal's 64-bit EUI. The prime portal's EUI serves as a context identifier, where the context refers to the bridge routing tables and virtualId-address assignments.

1.17.1 Refresh identifiers

A 66-bit primeId identifier is used to select the prime portal. This value consists of a preference field prepended to an EUI-64 value, as illustrated in figure 4.

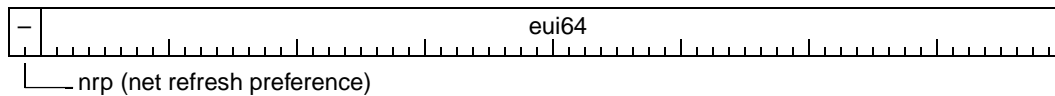


Figure 4—Format of primeId values

The 2-bit *nrp* (net refresh precedence) is a software settable parameter that is normally intended to select the prime portal.

If two or more *nrp* values are the same, the 64-bit *eui64* field is used as a tie breaker in the prime portal selection process. This *eui64* values is a copy of unique identifier located in the node's *bus_info_block* ROM.

Note that the primeId identifier does not include dynamic precedence information (such as the number of attached nodes or buses), as this would increase net refresh and can be counter productive (see E.1).

6. Net refresh

TBD—This clause can be viewed as a preliminary straw-man proposal. It provides an existence proof for how net refresh can be simply and reliably performed. Optimizations for common cases and encoding specifications are currently missing.

A “net refresh” refers to the sequence of actions that assign busId addresses and establish the bus bridge routing tables. The term “net refresh” is used because the effects of a net refresh on the bridge portals within the net is similar to the effects of a bus reset on the nodes attached to the bus.

As is true for bus resets, a net refresh is not necessarily disruptive. When possible, the net refresh maintains the previous busId assignments, bridge portal routing tables, established isochronous channels, and queued subactions.

6.1 Net refresh objectives

Net refresh design strategies are listed below. Note that the strategies for breaking loops and identifying buses is similar to algorithms used to identify nodes and break loops on a point-to-point based “bus”.

- 1) Stable. Bridge routing topologies do not change unless the physical cable topology changes.
- 2) Loop breaking involves concurrent algorithms:
 - a) Portal selections. Alpha portals are selected as follows:
 - i) On each net, a primary bus with a prime-alpha portal is selected.
 - ii) On each secondary bus, the alpha portal is the primary-bus-path portal.
 - b) Tree configuration. Bus bridge portals are set to enable a spanning-tree topology.
 - c) Portal IDs. Sequential portalId assignments yield distinctive labels while safely aging packets.
- 3) Two forms of net refresh are supported:
 - a) Net refresh. A net refresh is nondisruptive, but generates stale bus_ID addresses.
 - b) Net reset. A net reset is disruptive, because it recycles stale bus_IDs (forces discarding of related EUI-to-nodeId transactions). Two forms net reset are specified:
 - i) Lazy net reset. Transactions progress during the recycling of stale bus_IDs.
 - ii) Hard net reset. Transactions are stalled during the recycling of stale bus_IDs.

The intent of (1) is to prevent intermittent hard-to-diagnose errors, by avoiding changes the location of disabled redundant-bridge connections, when physical cables have not changed. Otherwise, the unfortunate activation of a marginal/redundant bridge could generate unexpected errors.

When performing resets, each bridge portal is assumed to be aware of the bridge-portal neighbor (if any) with the next larger phyId address (see xx). Local daisy chained sequences are then possible; for example would be portalA sends messages to portalB, portalB sends messages to portalC, and portalC returns messages to portalA.

Net refreshes are multi-phase operations, listed below. The initial phases select the prime portal and detect addressing conflicts; the final phase assigns busId assignments, establishes routing tables, and (when necessary) purges asynchronous bridge queues.

- 1) Acquire. Periodic acquisition messages are distributed to other bus-local portals.
- 2) Breach. The acquired portals sequentially extend their acquisitions to adjacent buses.
- 3) Commit. When the breach phase completes, the prime portal sends commit messages, allowing bus numbers and routing tables to be updated.

6.2 Hard net refreshes

This subclause discusses hard net refreshes, which leave the net in a known initial state. Hard resets are designed to be robust, so they may be disruptive. Most of the bus portal state is ignored, to ensure the net refresh success despite previously corrupted state. State could become corrupted by a variety of unforeseen circumstances, including static electricity or alpha particles.

The process of initializing a net topology involves formation of a spanning tree by circumscribing the paths through bus bridge portals, as illustrated in figure 5. For stability, the node with the largest primeId is selected to become the prime portal. This prime portal's EUI is also the basis for the contextId that is distributed among the nodes. In this figure, the solid arrows indicate the propagation of messages from their prime portal source.

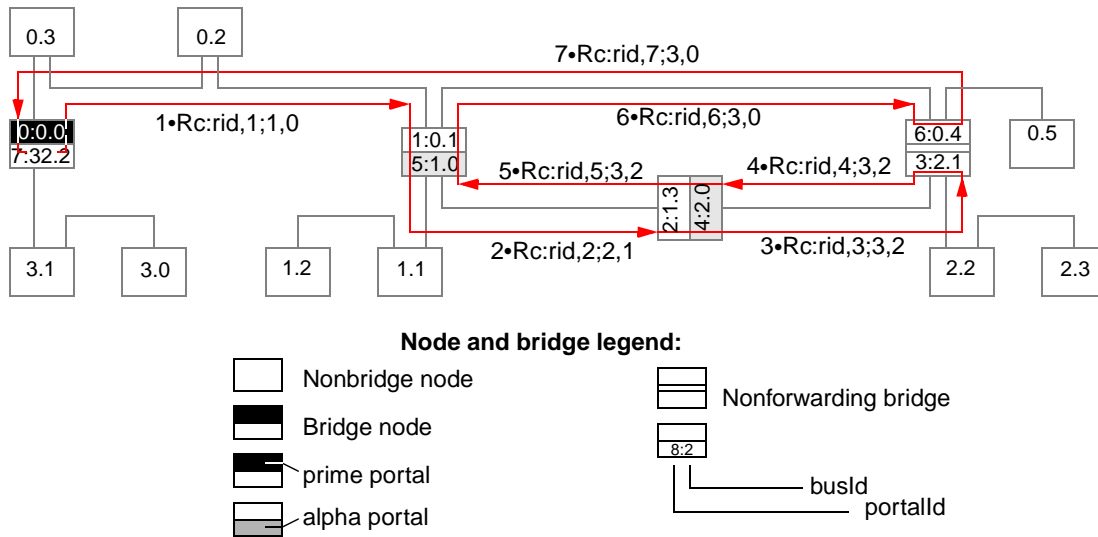


Figure 5—Hard net refreshes, messaging paths

The numbers within bridge portals identify their sequentially assigned *portalId* and locally shared *busId* values respectively. For example, the “1.0” portal has *portalId*=1 and *busId*=0 and values. The *portalId* has no effects on the portal’s operation, but is saved for diagnostic purposes. The *busId* affects the assignment of virtualId addresses and the routing of packets.

This clause illustrates hard net refreshes initiated by the prime portal, for simplicity and clarity, although a hard net refresh can be initiated by any portal. If initiated by a non-prime portal, the initial net-reset messages (not illustrated) would eventually propagate the prime portal, which (due to its larger *primeId* identifier) would become responsible for completing the net refresh, as illustrated in the remainder of this subclause.

The leading “•” characters are used to label the time sequencing of events. The “1•” label is applied to the first change that occurred, the “2•” label is applied to the second most recent change, etc.

6.2.1 Local bus acquisition

A net refresh doesn't invoke bus resets, but starts with *resetAcquire* messages sent between bus bridge portals. The first of these messages are sent from the prime portal and circulate through subordinate portals on the primary bus, as illustrated by the dotted lines in figure 6.

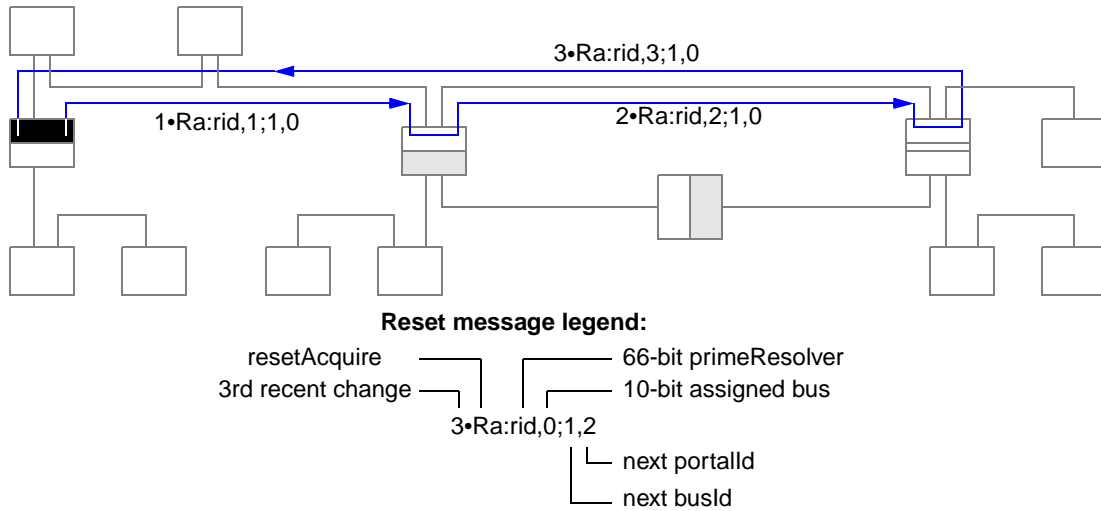


Figure 6—Hard net refreshes, primary bus acquisition

In this and following illustrations, the net refresh packets have “Ra:rid,0;1,2” labels. The “Ra” name is an abbreviation for resetAcquire, the “Rb” name (shown later) is an abbreviation for resetBreach, and the “Rc” name (shown later) is an abbreviation for resetCommit. The *rid* name refers to the *resetId* of the packet source and the “2” value is the next available *portalld*; the “1” value is the next available *busld* address and the “0” value is the current *busld* assignment.

For robustness, the *portalld* value in net-reset packets is incremented when passing through not-yet-enumerated bus-bridge portals. This ensures the eventual demise of rogue resets, by allowing them to be aged until dead. Reset messages are sent periodically, once each arbitration interval, until the net refresh completes so that (in the absence of prime-portal's resets) other portals can timeout and attempt to become prime portals. The timeout is triggered by the absence of observed net-reset packets during eight successive arbitration intervals.

6.2.2 Adjacent bus acquisition

The returned resetAcquire packet allows the prime portal to propagate a resetBreach indication to its “downstream” neighbor. The resetBreach initiates the acquisition of the first remote bus, as illustrated in figure 7.

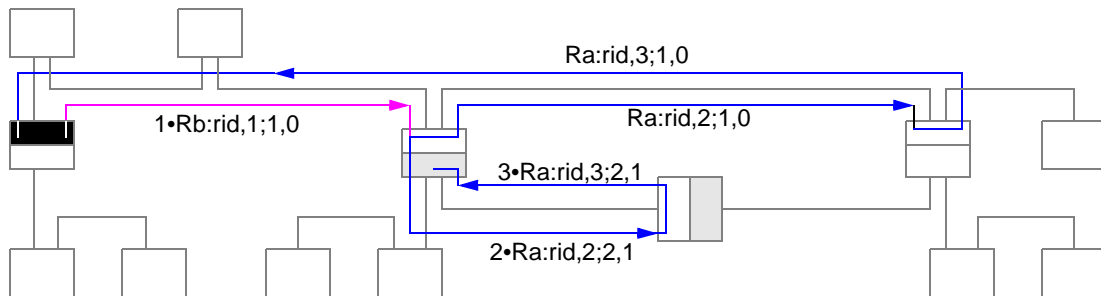


Figure 7—Adjacent bus acquisition

6.2.3 Remote bus acquisition

The next portal on the adjacent bus observes the returning resetAcquire indication and propagates a resetBreach indication to its adjacent bus. That resetAcquire circulates and acquires the third bus portals, as illustrated in figure 8.

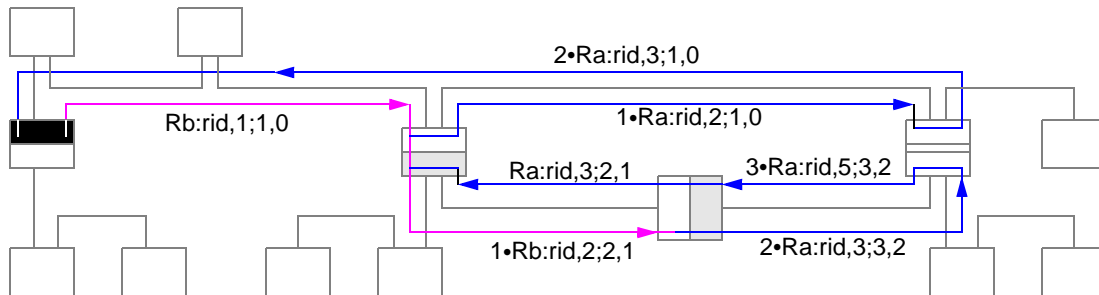


Figure 8—Remote bus acquisition

6.2.4 Looped breach propagation

The resetBreach propagates through portals that connect to a previously acquired bus, thereby starting the path back towards the prime portal, as illustrated in figure 9.

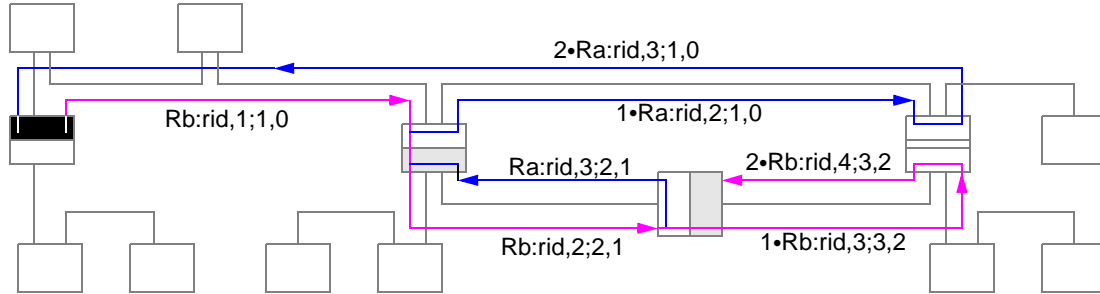


Figure 9—Looped breach propagation

6.2.5 Breach phase completion

The resetBreach indications eventually retrace the previously established paths and return to the prime portal, as illustrated in figure 10.

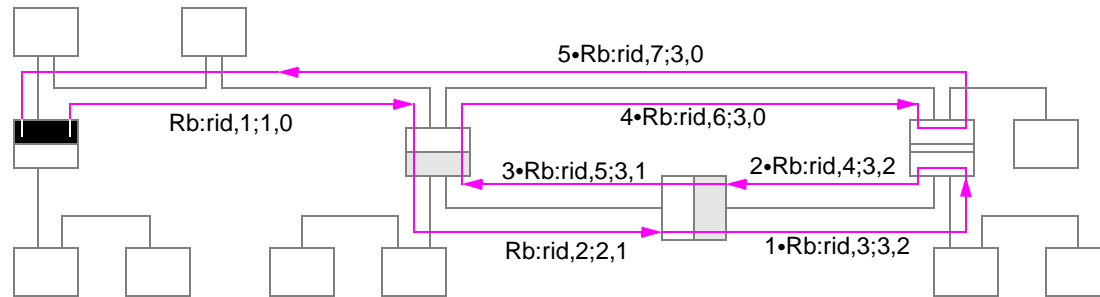


Figure 10—Breach phase completion

6.2.6 Commit phase

From its previously observed selfId packets, the fourth bus is known to have only one bridge portal and its *busId=3* value assignment occurs without generating any Serial Bus transactions. Then, the resetCommit messages propagate through previously established paths, assigning busId addresses and establishing routing tables, as illustrated in figure 11.

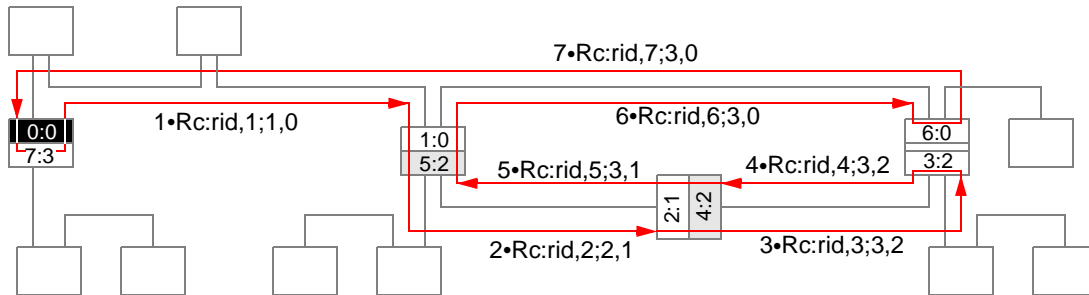


Figure 11—Commit phase completes

6.2.7 Hard net refresh effects

During the resetCommit phase, bridge portals discard previously queued (and now possibly stale) subactions. In addition, each source is quarantined by setting bytes of the QUARANTINE register to their initial zero-valued state. The intent is to prohibit remote-bus accesses until the requester has discarded previous (and possibly inconsistent) EUI-to-virtualId translations.

Upon receiving a resp_data_error/pack_quarantined response, the requester discards previous net-transaction state, including active transaction state and EUI-to-nodeId mappings. The requester then restores the alpha portal's QUARANTINE state and retries virtually addressed requests.