P1394a Power Management Proposal

Intel Mobile Technology Lab

Proposal for IEEE 1394 Suspend/Resume Capability

Intel Corporation

1. Introduction

The IEEE 1394-1995 serial bus provides some node power-management capability, in that the node's link can be turned on with the Link-On packet, and then subsequently turned off through a write transaction to the *linkoff* bit of a node's STATE_SET register. However, this bus standard requires that a node's physical layer (PHY) must remain powered and active, so that a node remains capable of relaying packets which are addressed to other nodes on the cable. (Leaf-nodes on a bus are an exception to this requirement, since they have no nodes located further "downstream" on the bus. Such nodes are free to turn off their PHY, as well as their link-layer, as appropriate).

With the preceding architecture, node power consumption can be reduced to levels dictated by PHY power consumption. With today's PHY implementations, this is on the order of 1W or less per node. Since contemporary link-layer implementations consume a similar amount of power, the Link-On/Link-Off controls can be used to roughly halve the node power consumption (relative to its full operating state).

Unfortunately, this degree of power reduction is inadequate to support the needs of mobile systems, whose total power dissipation in a "suspended" state may be 100 mW or less. 1394-1995 power management capabilities are also probably inadequate for "green" desktop PCs, which must constrain their "sleeping" power-consumption to, say, 3W total (e.g. to satisfy Energy Star or Blue Angel certification requirements).

In a future high-speed 1394b PHY (800+ Mbits/sec) with a projected per-port active consumption of 300 mW and a per-port inactive consumption of 100 mW, a 4-port PHY with all ports inactive still consumes almost half a Watt. This significant portion of the sleeping platform's power budget is simply wasted.

This proposal addresses the power-management needs of mobile and other power-constrained 1394 implementations. The proposal suggests mechanisms which should reduce 1394 subsystem power to a small fraction of what it is today.

1.1 Scope of Proposal

"Power management" of a 1394 bus actually requires a variety of distinct capabilities, including

- The ability to control power distribution from a 1394 device to other bus-resident devices.
- The ability to put a node into a reduced-power "Standby" mode, in which bus traffic can still traverse the node's PHY, as usual.
- The ability to put a 1394 bus, and all attached devices, into a very low-power "suspended" state. A node in this state may be able to resume normal operation in response to receiving a "wake-up" signal.
- The ability for software to enable or disable individual PHY ports.
- The ability for host operating software to control the transition of 1394 devices between a set of well-defined node power states.

The first of these capabilities is described in a 1394 Trade Association document titled "**1394 TRADE ASSOCIATION Power Specification Part 1: Cable Power Distribution Cable Power Distribution**". The remaining items all relate to the definition and control of low-power 1394 states; these are described in a 1394 Trade Association document titled "**1394 TRADE ASSOCIATION Power Specification Part 2: Power Management**" (under development).

This proposal treats *only* aspects of 1394 power management which require extensions or changes to the 1394-1995 bus standard. All other (higher-level) facets of 1394 power management are dealt with in Trade Association documents.

Control of the power-state of nodes is performed by a bus “power manager” agent, using the low-level mechanisms (e.g. newly-defined packet types) described in this proposal. The methods by which the manager applies these low-level mechanisms is **TBD**. The protocols and specific functions used by this manager lie outside the scope of this document.

1.2 Definitions

“Suspend” and “resume” are ill-defined terms in today’s computer industry. We adopt the following usage:

- ***“Standby”*** refers to a state in which the link and higher layers of a node (i.e. its internals) are in a very low-power sleeping state, while the node’s PHY continues to operate as usual. This allows the node to conserve power while continuing to serve as a repeater of bus traffic. While in Standby, a node’s PHY draws power from the 1394 cable to which it is attached. If such power is unavailable, the bus is broken into independent segments at that point.
- “***Suspend***” refers to a state in which a node has saved its internal state (to battery-backed or low-refresh-rate RAM, to disk or to some other non-volatile memory), and has then stopped normal operation. In this state, the node is incapable of doing application-level processing, and even most system software is not executed. Instead, the node retains just enough intelligence to recognize the occurrence of one or more types of prescribed “***wake-up***” events, whose occurrence can be used to trigger resumption of the normal operating state. Triggering of wake-up can be caused by reception of a ring-indicate signal by a telephony device; reception of a wake-up packet by a LAN adapter; a change in the charge-state of a device’s battery; a request from a node-resident processor; etc. The wake-event can reach a node over the 1394 bus, or it can originate within a node (e.g. from an internal timer).
- The process of returning a suspended system to its normal operating state is referred to as “***resume***” processing. This entails restoration of the system’s state from suspend storage, followed by resumption of execution of system software and application software where these were interrupted by entry into the suspend state.

1.2.1 1394 ACPI Definitions

Table 1A describes the ACPI device power states which can be supported by a 1394 device which includes a proposed suspend capability. In this table, resume latency (i.e. time taken to return to the D0 normal operating state) increases toward the bottom of the table. Also, the amount of power consumption decreases toward the bottom of the table.

Device Power-State	**Descriptor**	**PHY Power**	**Link Power**	**Device Context**	**Wake-Up Available**
D0	Run	ON	ON	Saved	No
D1	Standby	ON	OFF	Saved	Yes
D2	Suspend	ON_LP	OFF	Lost	Yes
D3	Off	OFF	OFF	Lost	No

Table 1A: 1394 ACPI Device Power States

For the link-layer, the “OFF” state corresponds to the link-layer power-state of a node whose *linkoff* STATE_SET register bit has been set. For the PHY layer, the “ON-LP” state corresponds to the proposed new PHY signaling state, in which the PHY cannot accept or relay packets. Instead, a PHY in the ON_LP state is only capable of signaling or propagating a wake-up event indication.

Note that all node context information is preserved in the Standby state, for both the 1394 layers and the node device(s). This allows a very rapid return to the Run state.

In this proposal, the terminology in the “Descriptor” column of the table is used to describe the state of devices on the bus. ACPI power-states could be used instead, but these labels are less suggestive.

2. STANDBY

As described above, the PHY of a node which is in Standby continues to receive (cable) power and function normally, while the node’s link and other internals are in a sleeping state. A node can enter Standby either autonomously (e.g. a notebook computer suspending itself due to inactivity), or in response to reception of a command from another “power manager” node on the 1394 bus.

The only requirements for a node to support autonomous Standby are that the node implementation provide separate power controls for the PHY and the link (including the ability for the PHY to draw power from the cable), and that the node’s PHY must have stand-alone capability to relay bus traffic which is addressed to other nodes. To support this power-source switching, the PHY needs to provide an output signal which is asserted when a node receives a Standby packet.

If a node is to have an optional capability to enter Standby on command from another node, the node’s PHY must be capable of recognizing a (newly-defined) Standby command packet. Reception of this packet causes a Standby-capable node to enter Standby as soon as possible. That is, entry into Standby is not an interlocked process. Another node can determine when the target node has entered Standby by reading that node’s Power State register (*PHY resident; location and contents TBD*). Read-access to this register does not automatically trigger resumption of normal node processing.

A node may receive a Standby request on any of its ports. In each such case, that node proceeds into the Standby state. However, the receiving node does not propagate the request to any other nodes. That is, Standby is a node-specific state; multiple nodes must individually be put into Standby.

Computer nodes need autonomous Standby capability, but probably do not need to implement the optional externally-commanded Standby capability. Other classes of 1394 devices (e.g. cameras, VCRs, etc.) may implement both autonomous and externally-commanded entry into Standby.

A standing-by node resumes normal operation in response to reception of a transaction for which it is the target node. That is, resumption occurs automatically as a side-effect of attempted communication with the standing-by node. Resumption takes some time, since power and then contextual state information must be restored to the node. During this time, the node responds to transactions with an “ACK tardy” acknowledge, as proposed by Microsoft. This suffices to inform other nodes that they should retry communications later, till they receive a normal acknowledge.

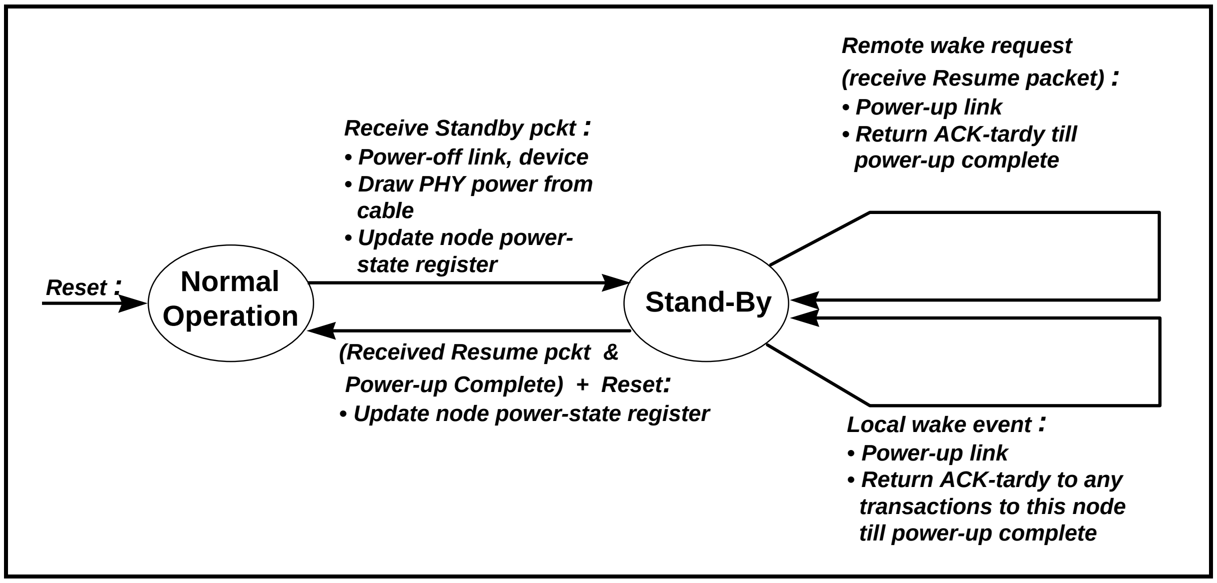

Remote wake request
(receive Resume packet) :
• Power-up link
• Return ACK-tardy till power-up complete
Receive Standby pckt :
• Power-off link, device
• Draw PHY power from cable
• Update node power-state register
Reset :
Normal Operation
Stand-By
(Received Resume pckt & Power-up Complete) + Reset:
• Update node power-state register
Local wake event :
• Power-up link
• Return ACK-tardy to any transactions to this node till power-up complete

Figure 2A: Node Standby Entry/Exit

3. BUS SUSPEND

There are cases in which it is desirable to reduce 1394 bus power consumption below the level that is attainable through node Standby. For example, a notebook computer may need to maximize its operating time while on battery power, or a sleeping “green” desktop computer may need to conserve most of its sleep-state power budget for use by subsystems other than the 1394 bus. Suspending the bus, rather than turning it off, has the advantage of a shorter resume latency; resume-related restoration of state-information is faster than full reconfiguration of the bus. In such cases, it is necessary to turn off power to the node’s PHY, as well as to higher layers.

In mobile systems, it is common to suspend nearly all of the platform’s functions, while awaiting the occurrence of a wake event. In that state, only a very-low-power system power manager block (i.e. keyboard controller) remains powered and active, so that it can detect wake events and bring the system back to normal operation. In such a situation, all devices on a 1394 bus would be put into the suspended state. Since a 1394 bus consists of a set of constituent ports, this puts the bus into suspend.

In the resulting “suspended” state, the node needs to retain the ability to detect and record topology changes (attach/detach events), and to resume its normal operating state. In addition, the node may need an optional capability to generate “wake events” which can cause the bus to resume normal operation.

These capabilities lie outside the scope of the 1394-1995 standard, and are candidates for inclusion in 1394a. The specific proposed changes to 1394a are described in the following sections.

3.1 PHY Architecture for Suspend

The proposed suspend state is a bus-level state, in that the bus does not support normal packet-based signaling while in this state. Instead, a suspended bus is only capable of detecting attach/detach events, and of propagating a simple resume request throughout the bus. All of the node PHYs on a suspended bus must be in a (newly-defined) very low-power suspended state, in which they are almost completely powered off.

In suspend, a node keeps TpBias powered on its ports in the range $0.7v \leq TpBias < 0.9v$, and the node also keeps its connect-detection circuitry powered. This allows the node to detect attach/detach notification events and latch them locally, while sinking only about 0.2 mA of current per port.

A new "Topology Changed" PHY-register bit (*location TBD*) is also needed to capture an attach or detach event which may occur while the bus is suspended. If this bit is set in any node when a bus resume begins, that node generates a bus reset. This causes re-enumeration to occur, thus accounting for the topology change. Note that if no PHY has its Topology Changed bit set at resume time, no bus reset is generated. This avoids needless reconfiguration of the bus. In either case, resume processing clears each node's Topology Changed bit as a side-effect.

3.2 Wake-up from Suspend

A suspended bus can be brought back to normal operation by having any node generate a wake-up event. A node does this by returning TpBias from its suspend value-range of $0.7v \leq TpBias < 0.9v$ (which indicates that the attached node is connected and in the suspend state) to the range $TpBias \geq 1.0v$ (which indicates that the attached node is connected and in a normal operating state). The wake-event does not cause the port's Topology Changed bit to be set, thus avoiding automatic assertion of a bus reset on resumption of the bus.

When a node's PHY receives the preceding TpBias resume signal on any of its enabled ports, the PHY repeats that signal on all of its other enabled ports. (Recall that the 1394a specifications include per-port enable/disable capability). In this way, the wake event propagates to all enabled portions of the bus. Conversely, portions of the tree which have been disabled do not receive, and are not affected by, the wake signal.

This wake mechanism is global to the bus, and is not a means for routing a wake-event from a given source-node to a particular wake-target node. When a bus resume occurs, the bus begins consuming power in all nodes which are attached to enabled ports. (Per-port disable supports device-tree "pruning", or diabling of bus branches and associated current-draw, to reduce the size of the current-draw step which occurs on resume). Since wake-event propagation involves more than just a pair of peer nodes, software polling must be used to determine the source and target of a wake event.

3.3 Suspend Entry/Exit Protocol

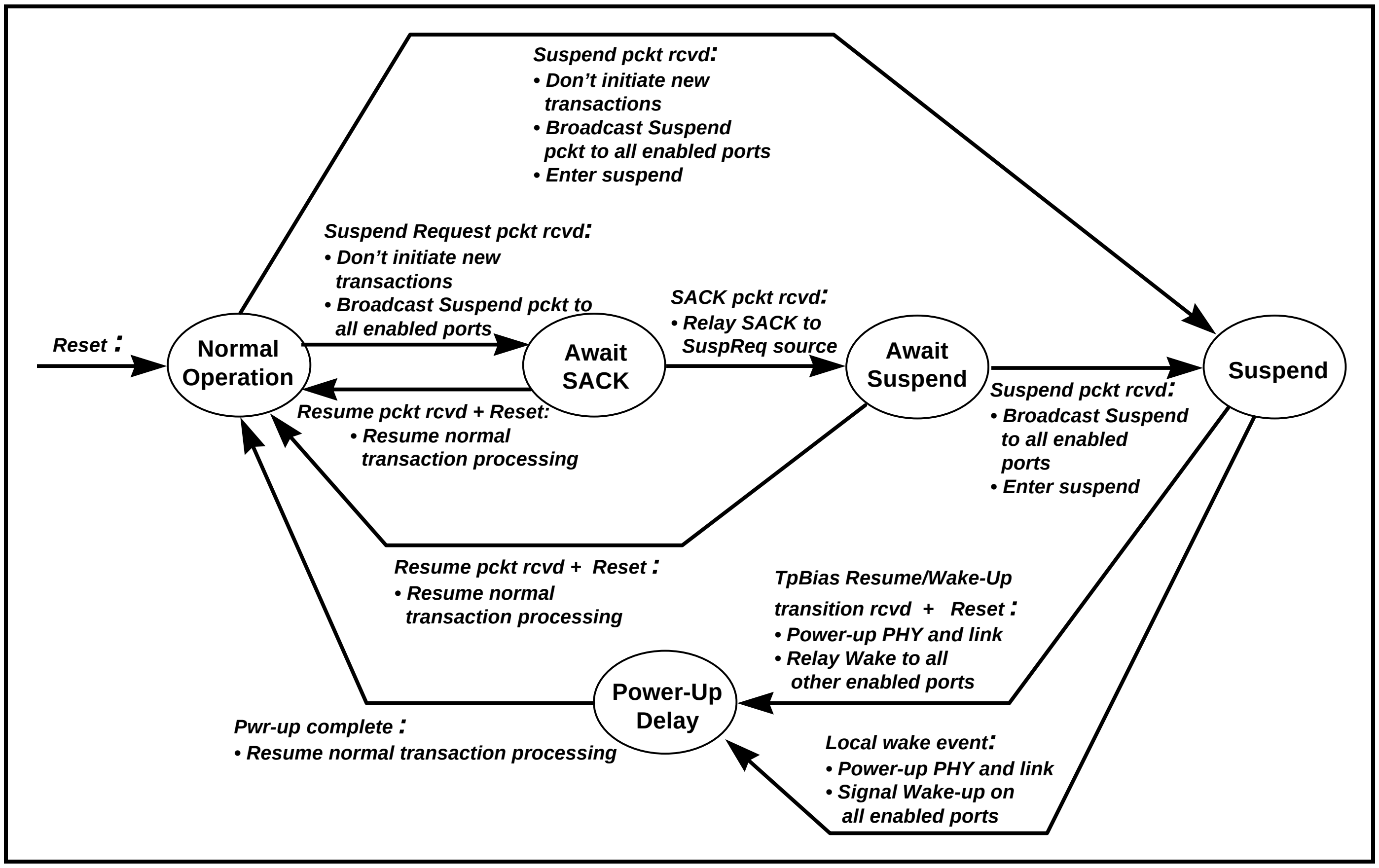
1394 suspend is a bus-wide state, which affects the operation of all nodes on that bus. Accordingly, making the bus unavailable for normal usage by suspending it is an action that can only be initiated by the bus power manager. It does not make sense to give other nodes this far-reaching capability.

Each of the devices on a bus needs the ability to hold off entry of the bus into the suspended state. This allows all bus-resident devices adequate time to prepare for entry into suspend, during which time the bus remains available for transactions. Interlocked entry into suspend is accomplished through a suspend request/acknowledge hand-shake protocol (see Figure 3A).

The bus suspend manager initiates suspension of the bus by broadcasting a (newly-defined) Suspend Request packet to each of its enabled ports. (If a port is disabled, that port's TpBias is inactive and any attached device is effectively disconnected from the bus). Any node attached to one of these ports begins to prepare for suspension. In addition, each such node relays the Suspend Request packet to each of its enabled ports (other than the one on which it received the Suspend Request). This process continues until the request packet has been repeated throughout the bus tree (i.e. the request reaches all leaf nodes).

Figure 3A: Bus Suspend Protocol

When a node has completed its own preparation for suspension, it checks to see that it has


Suspend pckt rcvd:
• Don't initiate new transactions
• Broadcast Suspend pckt to all enabled ports
• Enter suspend
Suspend Request pckt rcvd:
• Don't initiate new transactions
• Broadcast Suspend pckt to all enabled ports
SACK pckt rcvd:
• Relay SACK to SuspReq source
Reset :
Normal Operation
Await SACK
Await Suspend
Suspend
Resume pckt rcvd + Reset:
• Resume normal transaction processing
Suspend pckt rcvd:
• Broadcast Suspend to all enabled ports
• Enter suspend
Resume pckt rcvd + Reset :
• Resume normal transaction processing
TpBias Resume/Wake-Up transition rcvd + Reset :
• Power-up PHY and link
• Relay Wake to all other enabled ports
Power-Up Delay
Pwr-up complete :
• Resume normal transaction processing
Local wake event:
• Power-up PHY and link
• Signal Wake-up on all enabled ports

received a (newly-defined) Suspend Acknowledge packet from each of the ports on which it relayed the Suspend Request. Until this condition is satisfied, the node remains in a wait-state. Once suspend acknowledges have arrived at all such ports, the node transmits its own Suspend Acknowledge packet to its parent node, on the port through which it received the Suspend Request. Once the power manager has received all of its expected acknowledges, the manager knows that the entire bus is ready for suspension.

If a node is participating in a bus suspension, it cannot initiate a new transaction after it receives a Suspend Request packet. In addition, the bus power manager can use a suspend time-out to detect if the bus is unable to enter suspend within a reasonable (programmable) period.

If a time-out occurs, the manager can return all devices to their normal operating state by issuing a (newly-defined) Resume packet through each of its enabled. As each attached device receives the packet, it aborts its preparation for suspension. In addition, each node relays the Resume packet to each of its own enabled ports (except for the port on which it received the Resume packet). This process eventually reaches all leaf nodes, at which point the bus is again fully operational.

The bus suspend manager has the responsibility of actually transitioning the bus into the suspend mode. The manager does this by sending a (newly-defined) Suspend packet to each of its enabled ports. After doing this, the manager node puts itself into the suspended state.

As each child receives the Suspend packet, it repeats the Suspend command to each of its own enabled ports (except for the port on which it received that command), until the packet reaches all leaf-nodes on the bus. Immediately after relaying the received Suspend packet, each node transitions into the suspended state. When this process is complete, all nodes on the bus are in the suspended state.

3.4 Automatic Power-State Changes

One might consider whether there are circumstances in which 1394 nodes might automatically change *their own* power states, such as suspending themselves. Several particular cases deserve mention:

1) A node deciding to turn off its link following some period of not participating in transactions involving that node;
2) A node deciding to turn off both its PHY and its link once all of its ports have been disabled (using the 1394a per-port disable capability); and
3) The bus power manager deciding to suspend the bus once all nodes have entered Standby.

In the first case, the node has not been involved in recent bus traffic, as determined by node-resident logic (e.g. an inactivity timer). Since the node must continue to support its bus packet-relay responsibilities, its PHY must continue to be powered, as in Standby. However, the node is free to remove power from its link. The only requirement is that if the bus has a power-manager, the node should notify the manager of its intent prior to turning its link off. (This is discussed in the section on State-Change Notification).

In the second case, the node has split the bus by turning off *all* of its ports. Dealing with the consequences of this (especially coordinating the consequences of this with the bus power manager) are beyond the scope of this proposal. However, as in the first case, the node in question needs to notify the manager of its intent, if this is possible. Once this has been done, the node can turn off its PHY and link. Note that detection of this condition must be done by logic within that node, since bus communication with the power manager is impossible with all ports disabled.

The third case can arise due to all nodes individually deciding to enter Standby, or due to commands from the power manager. Having all nodes in Standby would save link power, though PHY power would still be consumed (to allow signaling to occur through the node). No node other than the bus power manager is aware of the occurrence of this situation, since only the manager knows the state of all nodes. Thus, the manager would need to be the agent by which the bus was suspended in this situation.

3.5 Node Power-State Change Notification

If a node changes its power-state in response to a command from the bus power manager, the manager knows both the old and the new states of the target node. The manager can poll the target's power-state bits to determine when the commanded change has been completed. In contrast, whenever a node changes its power-state on its own initiative (i.e. not in response to a command from the bus power manager), the changing node must notify the manager of this change. Failure to do so can cause the power manager to make incorrect assumptions about the power consumption of the changing node.

A (newly-defined) reporting mechanism is needed to allow nodes to report self-initiated power-state changes. This mechanism consists of a Power Change packet which describes the node's old and new power-states. Since various nodes can serve as bus power manager, each node needs a PHY register which can be loaded with the ID of the power manager. Then, a node can use the contents of that register to address a Power Change packet to the manager.

Note that for implementation symmetry, it may be useful to require that a node send a power-state notification packet to the power manager whenever that node changes power-state. That is, the node would send notification to the manager even if the *manager* commanded the state-change, rather than only if the node initiated its *own* state-change. Notification of a manager-commanded state-change would serve as the recipient's acknowledgement of the manager's command.

3.6 Suspend Preparation

When an entire 1394 bus is to be suspended, the nodes on the bus must be prepared for suspension. This involves two different activities:
1) Selection and enabling of wake-up events in nodes which can produce such events; and
2) Saving of device context.

These activities are performed at appropriate points in the suspend process; they are reversed as part of the resume process. Programming of wake-up events and the state-save process are described below.

3.6.1 Wake-Event Selection

In preparation for node suspension, the bus suspend manager can prepare the node to be suspended for wake-event generation. That is, the manager can select events which, if they occur, will serve as wake-up stimuli for other nodes. This programming involves specification of two pieces of information: the ID of the node which is to receive the event notification, and the type of event which is to cause wake-up.

We refer to the intended recipient of a wake-event as the wake "target". (Recall that 1394 is a peer-to-peer bus which can support multiple wake-event sources and multiple wake targets). Targets are simply identified through their bus node ID.

Different types of events may cause a suspended device to wake. Some of these are device-class specific, and are not discussed here; others are generic. The latter type includes events such as node attach or detach, device-generated interrupt, device ring-indicate, LAN wake-up received, and critical device battery/power state.

To simplify the writing of suspend/resume software, it may be advantageous to standardize a set of "event" registers, which can be implemented on any wake-source node. ***Figure 3B*** proposes an architecture for these registers.

Here, each of the set of device wake events is ANDed with a corresponding bit in an Event Mask register, and the result is captured in an Event register. The programmable Mask register is used to enable wake-up assertion on specific wake events. All of the Event register bits are ORed to form a single WAKE bit.

In addition, each of the Event registers is accessible through the node's 1394 interface. This allows programming of the Mask register, writing to the Event-Clear register to clear active Event register bits, and reading of the Event register to ascertain the nature of an active wake event. It may also be desirable to add an Event Set register through which Event register bits can be set; this would allow software simulation of wake events (e.g. for test purposes).

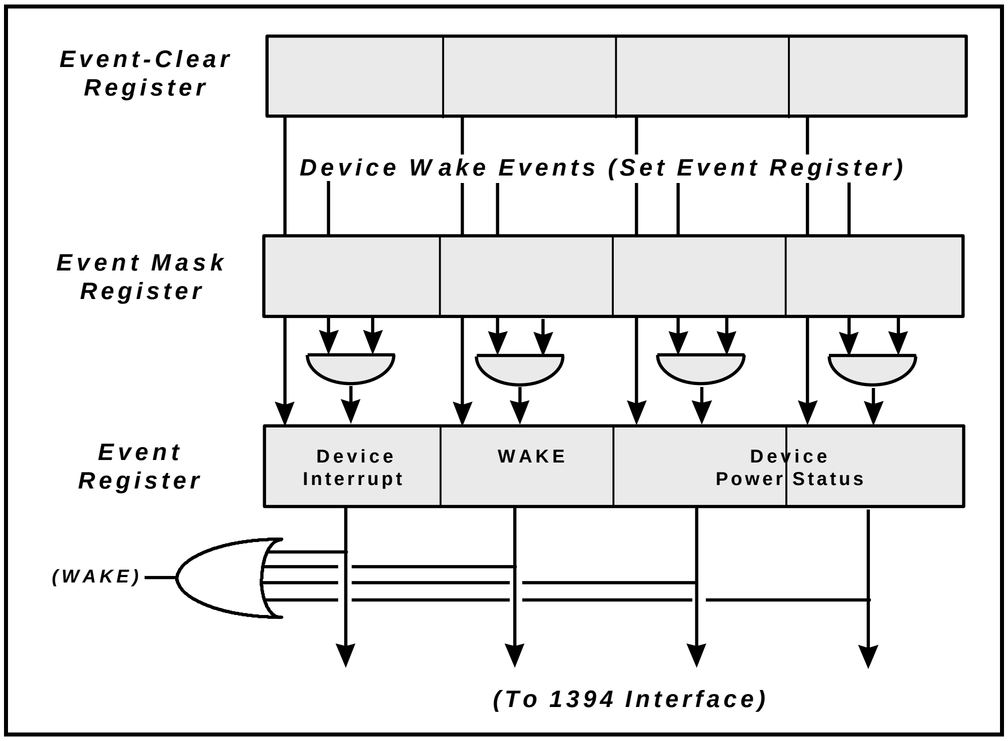


Figure 3B: Node Wake-Event Registers

3.6.2 State-Save

Per the proposed 1394 ACPI definitions, device context is lost when a device is suspended. This means that the device state (configuration and perhaps application state) must be saved in non-volatile storage prior to entry into suspend. Then, when that device is prepared for resumption of normal activity, its state can be restored by copying it back from non-volatile storage into its pre-suspend locations.

State-saving and restoration is probably best done by a bus's current suspend-manager node. That node knows the power-state of each node on the bus at a given moment, and it also controls the power-state transitions of each node. Saving and restoring node state is a logical extension of these control responsibilities.

The specific way in which device state is saved depends upon which ACPI S-state (system "sleep" state) is to be entered. Device state is preserved in states S1 and S2, and state is lost in S3, S4 and S5. The contents of system memory are preserved in S1-S3, but are lost in S4 and S5. This means that 1394 device state is saved in system memory ("save to RAM", or "STR") when the system enters S3; but device state is saved to disk ("STD") if the system is to enter state S4.

4. BACKWARD COMPATIBILITY

The proposed suspend/resume mechanism builds on the existing 1394a mechanisms. Suspend-capable nodes can coexist with "legacy" nodes which lack this capability.

Nodes which are not suspend-capable do not respond to Suspend Request, Suspend Acknowledge or Suspend packets. For such nodes, the suspend-acknowledge time-out mechanism will result in issuance of a Suspend Failed on any attempt to put the bus into a suspended state.

Since bus-level suspend/resume relies on use of the 1394 cable signals in a new operating mode, suspend is only effective for systems which contain only suspend-capable nodes. That is, all nodes on the cable must be able to enter the suspend state, so that the cable signaling mode can change to support suspend/resume. If a bus contains a mix of suspend-capable and non-suspendable nodes, the cable signals must retain their normal packet-propagation capabilities. In that case, nodes can be put into individual-node Standby, but the bus as a whole cannot be put into a Suspend state.

Implementation of the proposed suspend logic implies partitioning of a node's PHY, as well as of its attached device. This is necessary so that only a small portion of the node can be kept powered during suspend. It may be desirable to provide a suspend power rail which is separate from the normal operating power rail, though this is an implementation decision.

5. Expected Impact on 1394a Specifications

The proposed suspend/resume mechanism requires that several new packet types be defined (Suspend Request, Suspend Grant, Suspend, Resume Node and Resume). These can perhaps be incorporated into one new "Suspend control" packet, which would contain a bit-field which could be encoded to convey the various stages and test results of the suspend process. The details of this have yet to be worked out.

Similarly, this proposal describes the sequence of operations which support the suspend and resume mechanisms. However, the detailed contents, location and organization of needed supporting registers have not yet been established.

In addition, the suspend/resume capability redefines the usage of the bus signal lines while the bus is in the suspended state. This is, however, an additional capability which extends the existing 1394a specifications. Use of this signaling mode should not affect the operation of the bus in its normal packet-based signaling mode.

Table 5A captures the suspend/resume-related changes which are proposed for inclusion in the 1394a specifications.

Capability	Feature	Comments
(All power states)	PHY register bits to reflect present node power state	Must reflect following states: normal, standby, was-suspended.
Standby	Link power on/off signal from within node	Allows node to control link power consumption (e.g. period of inactivity)
Standby	Logic to detect all ports disabled and automatically turn off PHY and link	Complements per-port disable
Standby	Self-Standby packet	Notify power manager when node decides to put itself into Standby (it may be enough to just return Ack Tardy)
Standby	External resume command	Need new Standby packet; Need PHY Power State register (read doesn't cause resume)
Standby	Standby packet	New packet type (command for receiving node to enter Standby ASAP)
Standby	Standby Packet Received signal	PHY output needed to support switching of PHY power source
Suspend	Add "Standing By" power-status bit	Register location TBD
Standby	Automatic resume (from Standby) on reception of any packet targeted at standing-by node	
Standby	ACK Tardy response code returned whenever node is in, or exiting, Standby	

Suspend	Redefinition of connect-detect voltage levels (add “suspended” level)	Detached: $TpBias < 0.6v$; Undefined: $0.6v \le TpBias < 0.7v$ Suspended: $0.7v \le TpBias < 0.9v$ Undefined: $0.9v \le TpBias < 1.0v$ Attached: $TpBias \ge 0.9v$
Suspend	Maintain $0.7v \le TpBias < 0.9v$ while node is suspended	Allows attach/detach detection, wake-up assertion
Suspend	Define PHY signal to cause assertion of wake-up transition on TpBias	New PHY signal
Suspend	Add “Suspended” power-status bit	Register location TBD
Suspend	PHY internal power-plane partitioning to allow attach/detach and wake processing while other logic is unpowered	Requires new silicon
Suspend	Topology Changed PHY-register bit (capture attach/detach which occurs during suspend)	Location TBD
Suspend	Suspend Request packet	New packet type (request to prepare receiving node for suspend)
Suspend	Suspend Acknowledge packet	New packet type (Acknowledgment that receiving node is ready to enter suspend)
Suspend	Suspend packet	New packet type (command to immediately and unconditionally suspend receiving node)
Suspend	Resume packet	New packet type (command to have receiving node abort entry into suspend)
Suspend	Suspend entry/exit state-machine	Controls transition in signaling protocol (including initiation of new transactions after receiving Suspend Request)
Suspend	Suspend Request, Suspend Acknowledge and Suspend packet routing logic	Routes packets between receiving and repeat ports
Wake/Resume	Signal wake-up by returning TpBias from suspend level to $\ge 1.0v$	Allows attach/detach detection, wake-up assertion
Wake/Resume	Wake/Resume TpBias transition-relay logic	Creates or repeats TpBias Wake/ Resume transition to output ports

Figure 5A: Proposed Changes to 1394a

6. Contact Information

Please submit all suggestions, comments and concerns to:

Claude A. Cruz
Intel Mobile Technology Lab
2111 N.E. 25th Ave., Mailstop JF3-202
Hillsboro, Oregon 97214
(503) 264-7109
E-Mail: Claude_Cruz@ccm.jf.intel.com