# INTERRUPTS ON RESUME

98-019r0

Dave Scott
davidx_j_scott@ccm.intel.com

There appears to be general agreement that the generation of interrupts during resume is broken in two places. One is in the case of a resume command and the other is in the case of a fault. The problem is that an interrupt could be generated with no change in a port's connected, bias, fault or disabled bits. The upper layers would not have any indication of which port generated the interrupt. The following are the recommended changes to fix these two problems.

## Recommended Changes to Table 6-2, Draft 1.5

| | | | | |
|---|---|---|---|---|
| Fault | 1 | rw | 0 | Set to one if an error is detected during a suspend or resume operation.- Clearing this bit, clears both the resume and suspend error.~~A write of one to this bit clears it to zero.~~ |

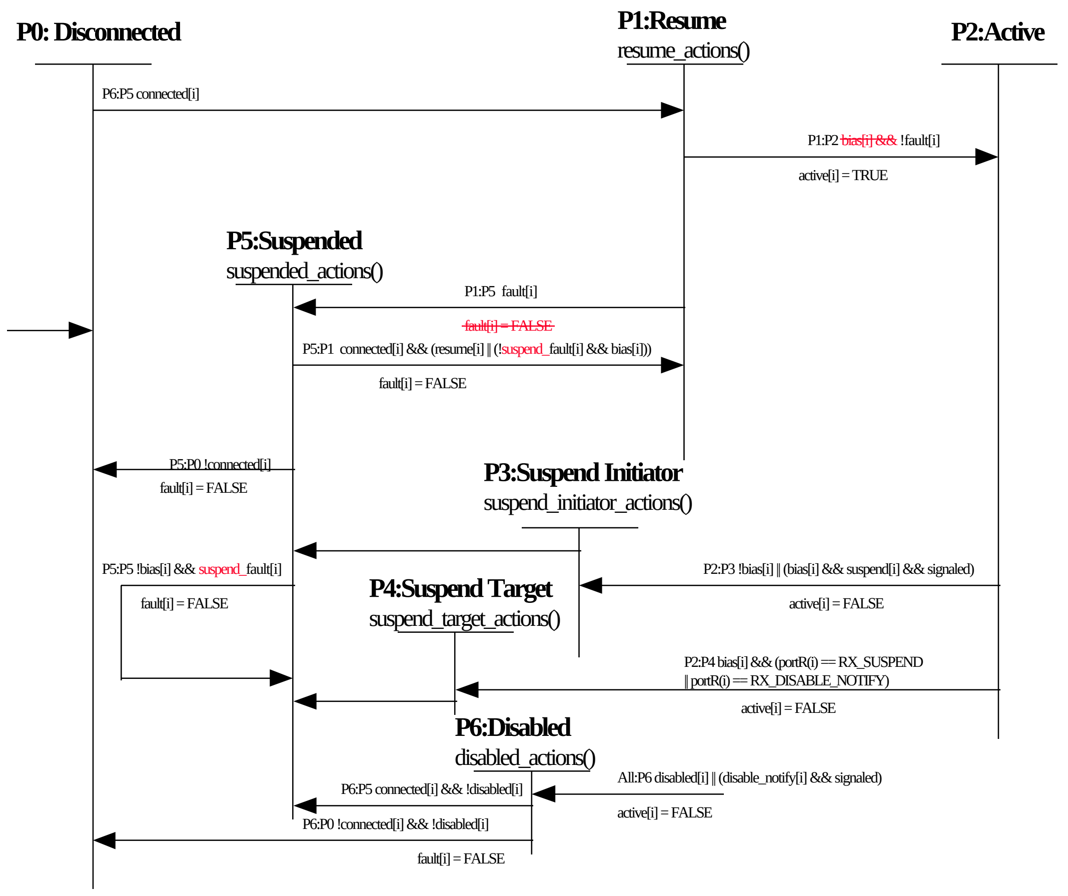
## Recommended Changes to Table 7-17, Draft 1.5

```
boolean resume_fault[NPORT];  // Set when its peer port does not participate in resume.
boolean suspend_fault[NPORT]; // Set when its peer port does not participate in suspend.
```

(Continued on the next page)

## PORT SUSPEND/RESUME STATE DIAGRAM

Recommend Changes to Figure 7-20, Draft 1.5


P0: Disconnected
P1:Resume
resume_actions()
P2:Active
P6:P5 connected[i]
P1:P2 ~~bias[i] &&~~ !fault[i]
active[i] = TRUE
P5:Suspended
suspended_actions()
P1:P5 fault[i]
~~fault[i] = FALSE~~
P5:P1 connected[i] && (resume[i] || (!suspend_fault[i] && bias[i]))
fault[i] = FALSE
P5:P0 !connected[i]
fault[i] = FALSE
P3:Suspend Initiator
suspend_initiator_actions()
P5:P5 !bias[i] && suspend_fault[i]
fault[i] = FALSE
P4:Suspend Target
suspend_target_actions()
P2:P3 !bias[i] || (bias[i] && suspend[i] && signaled)
active[i] = FALSE
P2:P4 bias[i] && (portR(i) == RX_SUSPEND || portR(i) == RX_DISABLE_NOTIFY)
active[i] = FALSE
P6:Disabled
disabled_actions()
All:P6 disabled[i] || (disable_notify[i] && signaled)
active[i] = FALSE
P6:P5 connected[i] && !disabled[i]
P6:P0 !connected[i] && !disabled[i]
fault[i] = FALSE

### Recommended Changes to Clause 7.10.4.1, Draft 1.5

**Transition P1:P2.** If the PHY port ~~is both connected,~~ did not fault during the resume handshake ~~and observes TpBias~~, it transitions to the active state.

**Transition P1:P5.** A resuming PHY port that ~~remains connected to its peer PHY port but fails to observe TpBias~~ faults during the resume handshake transitions to the suspended state. ~~The fault condition is cleared so that subsequent detection of TpBias may cause the port to resume.~~

**Transition P5:P1.** Either of two conditions cause a suspended PHY port to transition to the resuming state: a) a nonzero value for the port's *resume* variable or b) the detection of *bias* if the port~~'s~~ has not faulted during the preceding suspend transaction~~Fault bit is zero~~. A port's *resume* variable may be set indirectly as the result of the resumption of other PHY ports.

**Transition P5:P5.** If the port transitioned from the active state to~~entered~~ the suspended state in a faulted condition (*i.e.*, TpBias was still present), the fault is cleared if and when TpBias is removed by the peer PHY.

(Continued on the next page)

# Recommended Changes to Table 7-32, Draft 1.5

```
resume_actions(int i) {
    while (suspend_in_progress()) // Let any other suspensions complete
        ; // (we’ll resume those ports)
    connect_timer = 0;
~~    if ((int_enable[i] || resume_int) && !port_event) {~~
~~        port_event = TRUE;~~
~~        if (link_active && LPS)~~
~~            PH_EVENT.indication(INTERRUPT);~~
~~        else~~
~~            PH_EVENT.indication(LINK_ON);~~
~~    }~~

    connect_detect_valid[i] = FALSE; // Bias renders connect detect circuit useless
    tpBias(i, 1); // Generate TpBias
    if (resume[i] == 0 && !boundary_node)
        for (j = 0; j++; j < NPORT)
            if (!active[j] && !disabled[j] && connected[j])
                resume[j] = TRUE; // Resume all other suspended ports
    else
        resume[i] = TRUE; // Guarantee resume_in_progress() returns TRUE
    while (((connect_timer < BIAS_HANDSHAKE) && !bias[i]) || bus_initialize_active)
        ; // Wait for peer PHY to generate TpBias
    resume_fault[i] = ~bias[i]; // Resume attempt failed if TpBias is absent
        if (resume_fault[i])
            activate_connect_detect(i, 0); // restore usefulness of connect detect circuit
        else {                   // Connection restored to active state
            if ((int_enable[i] || resume_int) && !port_event) {
                // Notify LINK of port going active soon
                port_event = TRUE;
            if (link_active && LPS)
                PH_EVENT.indication(INTERRUPT);
            else
                PH_EVENT.indication(LINK_ON);
~~    if (bias[i]) { // Connection restored to active state?~~
            while ((connect_timer < 3 * RESET_DETECT) && !bus_initialize_active)
                ;
            if (!bus_initialize_active) { // No other node initiated reset?
                if (boundary_node) // Can we arbitrate?
                    isbr = TRUE; // Yes, don’t wait any longer
                else {
                    while ((connect_timer < 7 * RESET_DETECT) && !bus_initialize_active)
                        ; // Let’s wait a little longer...
                    if (!bus_initialize_active)
                        ibr = TRUE; // Sigh! We’ll have to use long reset
                }
            }
        }
    }
~~    fault[i] = ~bias[i]; // Resume attempt failed if TpBias is absent~~
~~    if (fault[i]) // If so, restore usefulness of connect detect circuit~~
~~        activate_connect_detect(i, 0);~~
    resume[i] = FALSE; // Resume attempt complete
}

void suspend_initiator_actions(int i) {
    connect_timer = 0;          // Used to debounce bias or for bias handshake
    if (!suspend[i]) {          // Unexpected loss of bias?
        suspend[i] = TRUE;      // Insure suspend_in_progress() returns TRUE
        if (child[i])           // Yes, parent still connected?
            isbr = TRUE;        // Arbitrate for short reset
        else
            ibr = TRUE;         // Transition to R0 for reset
        while (connect_timer < BIAS_DEBOUNCE)
            ; // Time for bias to stabilize
    }
    while ((connect_timer < BIAS_HANDSHAKE) && bias[i])
        ; // Wait for suspend target to deassert bias
    suspend_fault[i] = bias[i]);        // Suspend handshake refused by target?
    activate_connect_detect(i, BIAS_HANDSHAKE); // Also guarantees handshake timing
}
```