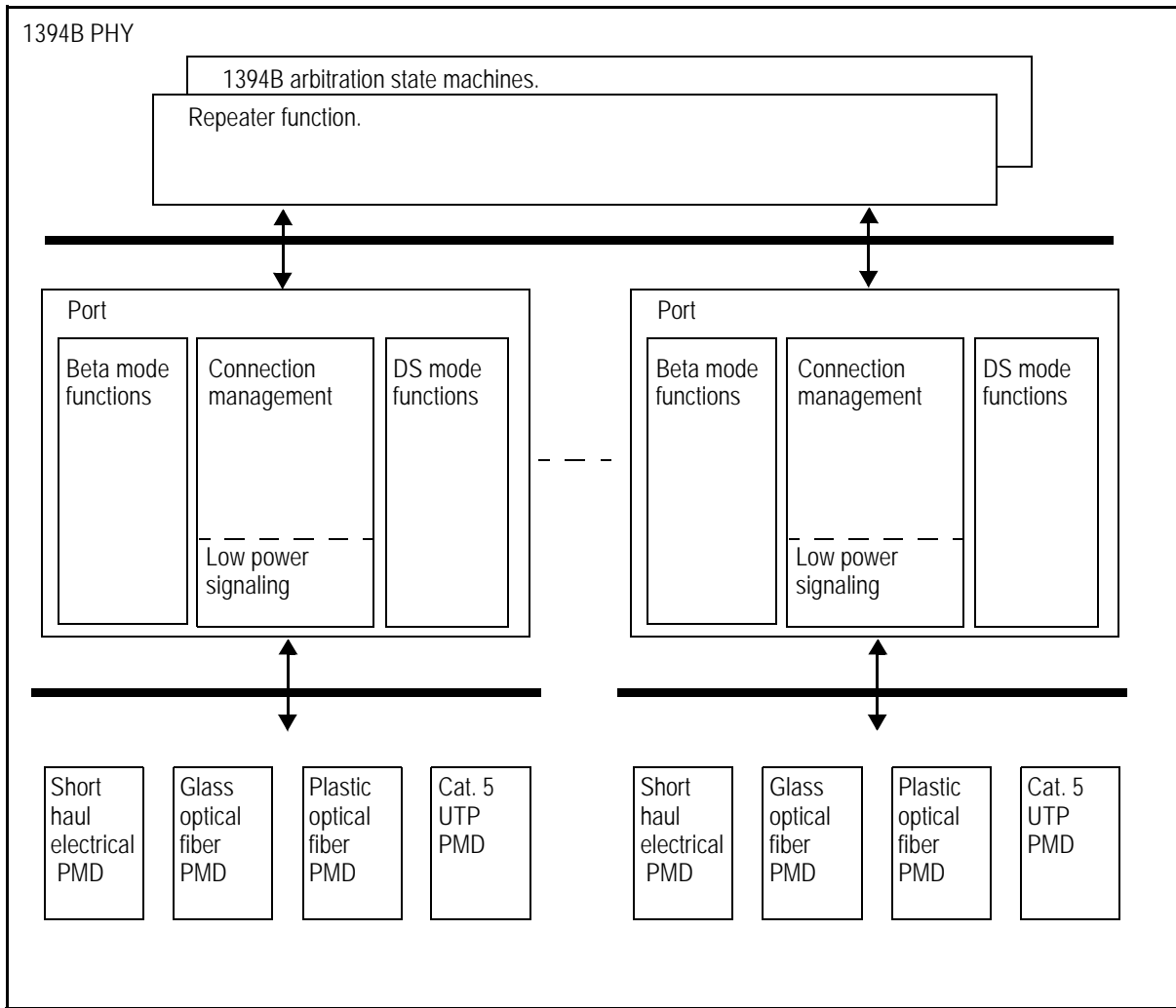


## 9. Connectivity Management

### 9.1 Overview

This section specifies the port state machines, handshake protocol and communications used between ports. The port abstraction is the primary interface inside the node between IEEE Std. 1394-1995 protocols and the P1394b gigabit protocols

The model for a port is shown below



The Connection Management function interacts with the PMD, Beta mode functions, DS mode functions and the repeater function by a set of services in each case..

## 9.2 Port characteristics

The port is the device that controls the flow of data bits onto and off of the 1394 wires. The port state machine must detect the present or absence of its neighbor port, establish reliable communications with the other port in the most suitable mode of signaling, and must manage low-power modes of operation, i.e., sleep modes.

### 9.2.1 Definitions

- Port and port state machine are interchangeable terms when the context make the choice clear
- DS mode means the normal IEEE Std. 1394-1995 form of electrical signaling and handshaking
- Beta mode means the 8b/10b, full duplex form of binary encoded signaling
- Connection means the cabled interconnect between two ports
- Sleep-Connected is a very low power signaling mode indicating continuing connection between ports
- Suspend means go into a low power mode of operation while maintaining the ability to process Sleep-Connected signaling
- Resume is a signal requiring the port to resume full speed, full power operation
- Beta chirp (aka tone) is a legal 8b/10b encoded signal (TBD) which indicates the maximum speed the port is capable of sending and receiving. Desirable qualities are low power operation, low-duty cycle, no spectral peaks, detection without phase locking to bit and quadlet boundaries, and sensing of the maximum port speed capability does not require bit and quadlet boundary phase locking.
- Sleep-connected chirp and resume chirp are (TBD) with the same constraints as the beta chirp

### 9.2.2 Requirements

- a) The port must start up in DS mode if either the port or the port to which it is connected is not beta capable
- b) The port must start up in beta mode if both the ports are beta capable
- c) The port must run (the "operational speed") at the fastest speed of which both ports are capable
- d) The port must support all speeds of packet at or below its maximum speed through the use of data padding
- e) The port must support Suspend, Resume, and Sleep-Connected signaling
- f) The port must detect and manage all connect/disconnect events on the connection

### 9.2.3 Port properties

A P1394b port

- a) may be capable of operating at any speed range from S100 upwards
- b) may be capable of operating at any speed range from S800 upwards
- c) shall be capable of operating in Beta mode
- d) when capable of operating at S100, shall be capable of DS at S100-S400, may be capable of Beta mode at S100-S400
- e) shall not be brought out to a 1394-1995 connector unless capable of operating in DS mode
- f) may be connected directly to a suitable transceiver for long haul connection (thus also providing DC isolation)
- g) may use DC (e.g. capacitive or galvanic) isolation when operating in Beta mode

## 9.3 Connection Management Services

Connection management services are provided at the interface between the CM function and the repeater function. The method by which these services are communicated is not defined by this standard.

### 9.3.1 CM status indication (CM\_STATUS.indication)

The CM function uses this service to indicate the status of the port to the repeater function. The following parameters are communicated via this service:

- a) Port mode. This parameter shall contain the mode of the port. The following values are defined for this parameter:
  - 1) DS. The port is operating in DS mode.
  - 2) BETA. The port is operating in beta mode.
- b) Port speed. This parameter shall contain the operating speed of the port. The value of this parameter may be any valid 1394B operating speed (see [tbd]).
- c) Port status. This parameter shall contain the status of the port. The following values are defined for this parameter:
  - 1) DISCONNECTED. The port is currently disconnected.
  - 2) CONNECTED. The port is connected.
  - 3) [FAULT. Something has gone wrong with this port. ]
  - 4)

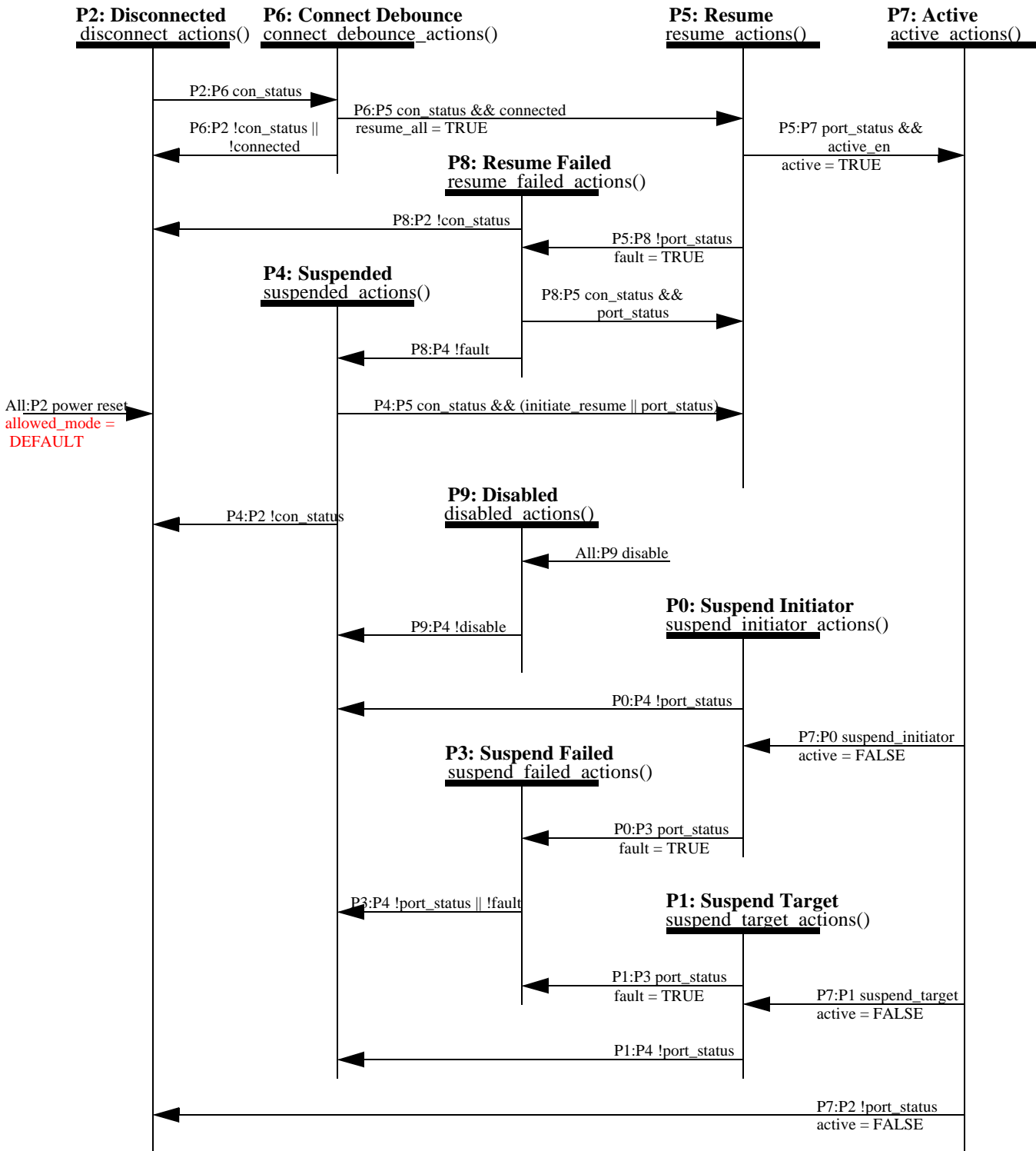
### 9.3.2 CM action request (CM\_ACTION.request)

The repeater function uses this service to control the connection management function of the port. The CM function shall service the request immediately upon receipt. The following parameters are communicated via this service:

- a) Action. This parameter shall indicate the action to be taken by the port connection management function. The following values are defined for this parameter:
  - 1) SUSPEND.
  - 2) RESUME.
  - 3) DISABLE.
  - 4) ENABLE
  - 5) NULL (no action)
- b) Port mode. This parameter shall indicate the mode in which the port is permitted to operate. The following values are defined for this parameter:
  - 1) DS. Forces the port to attempt connection only in DS mode.
  - 2) BETA. Forces the port to attempt connection only in Beta mode.
  - 3) DEFAULT. The port may attempt to connect in any mode.
  - 4) NULL (no change to port mode)
- c) [others?]

### 9.4 Port State machine

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66



## 9.5 Port state actions

### 9.5.1 P2: Disconnect state

```

1 void disconnected_actions () {
2
3
4
5
6 void disconnected_actions () {
7     port_power = TRUE;
8     DS_mode = Beta_mode = FALSE;
9     DSPORT_ACTIVATE.request(INACTIVE);
10    BPORT_ACTIVATE.request(INACTIVE);
11    CM.STATUS.indication(DISCONNECTED); // tell the repeated machine
12    PMD_CONTROL.request (PMD_disable);
13        // This sets TPA and TPB outputs to high impedance, and turns off TpBias
14        // and responds with a status indication
15    wait(port_functional);
16    active_en = FALSE; // prevents port from being active until bus reset
17    connection_in_progress = FALSE;
18    if (child)//parent still connected?
19        isbr = TRUE; // yes - arbitrate for short reset
20    else
21        ibr = TRUE; // no - core transits to R0
22    if ((connected || fault) && chg_int_en)
23        PH_EVENT.ind(INTERRUPT);
24    connected = FALSE;
25    con_status = FALSE;
26    fault = FALSE;
27    port_power = FALSE;
28    wait_event(PMD_CONTROL.indication (PMD_status)); // wait for status indication
29    PMD_CONTROL.request (toning);
30        // Start to send a well-spaced (low power) pulse-toning on TPB (specification TBD).
31        // simultaneously, start to listen
31    while (!con_status) {
32        connection_timer = 0;
33        action = mode = NULL;
34        wait_event(PMD_CONTROL.indication (PMD_status) ||
35                connection_timer > CONNECT_DEBOUNCE/2 ||
36                CM_ACTION.request(action, mode));
37        // wait for a change in status or a timeout
38        // timeout must be less than CONNECT_DEBOUNCE,
39        // but must be long enough to give us a chance of detecting a tone
40        // to avoid prematurely going into DS mode on a DC connection
41        if action = DISABLE disable = TRUE; //and a sharp exit to P9:Disabled
42        else {
43            if mode != NULL {
44                allowed_mode = mode;
45                if allowed_mode = DS PMD_CONTROL.request(stop_toning);
46            }
47            if (allowed_mode != DS) Beta_mode = PMD_status && tone_detected;
48            if (!Beta_mode && (allowed_mode != BETA)) DS_mode = PMD_status && PMD.con_status;
49            con_status = Beta_mode || DS_mode;
50        }
51    }
52 }
53 }
54
55
56
57
58
59
60
61
62
63
64
65
66

```

## 9.5.2 P6:Connect Debounce

```

1
2
3 void connect_debounce_actions () {
4     port_power = TRUE;
5     wait(port_functional);
6     connection_in_progress = TRUE;
7     connect_timer = 0;
8     if (Beta_mode) {
9         // Continue sending tones
10        wait (connect_timer >= CONNECT_TIMEOUT); // unlike DS mode, we cannot see resets yet
11        PMD_CONTROL.request (stop_toning);
12        PMD_CONTROL.request (send_status);
13        wait_event(PMD_CONTROL.indication(PMD_status);
14        if (PMD_status & tone_detected) {
15            // check that we have received a tone since entering this state
16            // Code to exchange timed tones to determine the operating speed (precise method TBD).
17            // result is left in port_operating_speed
18            // Note, by this time the port is powered up, so we can use normal
19            // timers and signalling.
20            connected = TRUE;
21        }
22    }
23    else { // DS mode
24        PMD_CONTROL.request(send_status);
25        wait_event(PMD_CONTROL.indication(PMD_status));
26        if (PMD_status & TpBias) {
27            wait(2 * BIAS_HOLD_NOTIFY);
28            PMD_CONTROL.request(send_status);
29            wait_event(PMD_CONTROL.indication(PMD_status);
30        }
31        if (PMD_status & TpBias) {
32            //TpBias handshake - need to wait to see TpBias at the far end first
33            PMD_CONTROL.request (TpGen);
34            wait ( connected ||
35                connect_timer >= (isolated_node) ? 2 * CONNECT_TIMEOUT: CONNECT_TIMEOUT);
36            PMD_CONTROL.request(send_status);
37            wait_event(PMD_CONTROL.indication(PMD_status);
38            con_status = PMD_status & PMD.con_status
39            if (con_status) connected = TRUE;
40        }
41        if (isolated_node)
42            ibr = TRUE;
43        else
44            isbr = TRUE;
45        if (connected && chg_int_en)
46            PH-EVENT.in(INTERRUPT);
47        connection_in_progress = FALSE;
48    }
49 }
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66

```

### 9.5.3 Resume

```

1 void resume_actions () {
2
3     port_power = TRUE;
4     wait (port_functional);
5     if (Beta_mode) {
6         PMD_CONTROL.request(send_status);
7         wait_event(PMD_CONTROL.indication(PMD_status);
8         if (PMD_status = resume_tone_detected)
9             resume_target = resume_all = TRUE;
10        PMD_CONTROL.request (resume_tone);
11        if (fault && chg_int_en)
12            PH_EVENT.ind (INTERRUPT); // notify higher layers of !fault
13        fault = FALSE;
14        connect_timer = 0;
15        port_status = FALSE;
16        // port_status is now a logical signal, not a physical copy of TpBias
17        // loosely, it means normal two way communications are established
18        // need to know if we are a resume target
19        while ((!port_status) && (connect_timer < DETECT_MIN)) {
20            wait ( (connect_timer >= DETECT_MIN) || PMD_CONTROL.indication(PMD_status));
21            port_status = PMD_status & resume_tone_detected
22        }
23        PMD_CONTROL.request (PMD_disable) //turn the toning off
24        if (port_status) { // train
25            BPORT_ACTIVATE.request (ACTIVE, port_operating_speed);
26            BPORT_TRAIN.request (TRAIN);
27            for (i=0; i < Training_time; i++) {
28                wait_event (BPORT_CLOCK.indication());
29                BPORT_CONTROL_request(IDLE);
30            }
31            BPORT_TRAIN.request (LOCK);
32            port_synchronized = TRUE;
33        }
34    }
35    else { // DS mode
36        DSPORT_ACTIVATE.request (ACTIVE)
37        PMD_CONTROL.request(send_status);
38        wait_event(PMD_CONTROL.indication(PMD_status);
39        port_status = PMD_status & TpBias;
40        if (port_status)
41            resume_target = resume_all = TRUE;
42            // This port is a resume target and resume all suspended ports
43        PMD_CONTROL.request(TpGen) // drive TpBias to resume target
44        if (fault && chg_int_en)
45            PH_EVENT.ind (INTERRUPT); // notify higher layers of !fault
46        fault = FALSE;
47        connect_timer = 0; // start timer
48        while ((!port_status) && (connect_timer < DETECT_MIN)) {
49            wait ( (connect_timer >= DETECT_MIN) || PMD_CONTROL.indication(PMD_status));
50            port_status = PMD_status & TpBias
51        }
52        // wait for target to drive TpBias
53        // NB drop out to Resume failed if it does not
54    }
55 }
56 }
57
58
59
60
61
62
63
64
65
66

```

## 9.5.4 Active state

```

1 void active_actions () {
2
3     active = TRUE;
4     initiate_resume = resume_target = FALSE;
5     CM.STATUS.indication (CONNECTED) // tell the repeater that all is now OK
6     if (chg_int_en)
7         PH_EVENT.ind (INTERRUPT); // notify higher layers of !suspend
8     action = NULL
9     control = IDLE
10    while (active && port_status && (action == NULL) && !(control == SUSPEND)) {
11        wait_event(CM_ACTION.request(action, port_mode) ||
12                 PMD_CONTROL.indication(PMD_status) ||
13                 BPORT_ERROR.indication(port_error) ||
14                 BPORT_CONTROL.indication(control))
15        if (DS_mode) port_status = PMD_status & TpBias;
16        suspend_initiator = action & suspend_action;
17        suspend_target = control == SUSPEND;
18        disable = action & disable_action;
19        if port_error & bad_link { // need to think through really what we should
20            // do on bad link - maybe some form of link reset??
21            // stop all activity
22            CM.STATUS.indication(DISCONNECTED); // tell the repeater machine so that we
23            // take control
24            BPORT_TRAIN.request (TRAIN);
25            for (i=0; i < Training_time; i++) {
26                wait_event (BPORT_CLOCK.indication());
27                BPORT_CONTROL_request(IDLE);
28            }
29            BPORT_TRAIN.request (LOCK);
30            port_synchronized = TRUE;
31            wait_event (timeout || BPORT_ERROR.indication(port_error))
32            if (port_error & bad_link) port_status = FALSE// still bad
33            if (port_status) CM.STATUS.indication(CONNECTED)// otherwise hand back over
34        }
35    }
36 }
37
38
39
40

```

## 9.5.5 Disabled

```

41 void disabled_actions () {
42     port_power = TRUE; // Power up the port
43     wait (port_functional); // Clock is running
44     if (DS_mode) DSPORT_ACTIVATE.request(INACTIVE);
45     if (Beta_mode) BPORT_ACTIVATE.request(INACTIVE);
46     if (chg_int_en)
47         PH_EVENT.ind (INTERRUPT); // notify higher layers of disable
48     PMD_CONTROL.request (PMD_disable);
49     port_power = FALSE;
50     while (disable) {
51         action = mode = NULL;
52         wait_event (CM_ACTION.request(action, mode));
53         if action = ENABLE disable = FALSE; //and exit P9:Disabled
54         else if mode != NULL allowed_mode = mode;
55     }
56 }
57
58
59
60
61
62
63
64
65
66

```

## 9.5.6 Suspended

```

1 void suspended_actions () {
2
3     port_power = TRUE; // Power up the port
4     wait (port_functional); // Clock is running
5     if (chg_int_en)
6         PH_EVENT.ind (INTERRUPT); // notify higher layers of !disable or !fault or suspend
7     fault = FALSE;
8     CM.STATUS.indication(DISCONNECTED); // tell the repeater machine so that we can take
9         //control
10
11     if (Beta_mode) {
12         handshake = FALSE
13         while (!handshake) {
14             control_word = NULL
15             wait_event (BPORT_CLOCK.indication(), BPORT_CONTROL.indication(control_word));
16             if (control_word = NULL) BPORT_CONTROL.request(SUSPEND);
17             else handshake = control_word == SUSPEND;
18         }
19         BPORT_ACTIVATE.request(inactive);
20         BPORT.CONTROL.request(PMD_disable);
21     }
22     else { // DS mode
23         PMD_CONTROL.request(PMD_disable) // drive TpBias low and then release
24         con_status = FALSE
25         while (!con_status) { // wait to eliminate false disconnect
26             wait_event(PMD_CONTROL.indication (PMD_status)); // wait for status indication
27             con_status = PMD_status && PMD.con_status;
28         }
29     }
30     port_power = FALSE; // turn off bias generator
31 }

```

## 9.5.7 Resume fail

```

32
33 void resume_failed_actions () {
34
35     initiate_resume = FALSE;
36     if (chg_int_en)
37         PH_EVENT.ind (INTERRUPT); // notify higher layers of fault
38     if (Beta_mode) {
39         // TBD - but can we get here in Beta mode? - but at least:-
40         BPORT_ACTIVATE.request(inactive);
41         BPORT.CONTROL.request(PMD_disable);
42     }
43     else { // DS mode
44         PMD_CONTROL.request(PMD_disable) // drive TpBias low and then release
45         con_status = FALSE
46         while (!con_status) { // wait to eliminate false disconnect
47             wait_event(PMD_CONTROL.indication (PMD_status)); // wait for status indication
48             con_status = PMD_status && PMD.con_status;
49         }
50     }
51     port_power = FALSE; // return to suspend state
52 }
53
54 }
55
56
57
58
59
60
61
62
63
64
65
66

```

### 9.5.8 Suspend initiator

```

1 void suspend_initiator_actions () {
2
3     connect_timer = 0;
4     suspend_initiator = FALSE; // clear the suspend request
5
6     if (DS_mode) {
7         while (!(connect_timer >= NOTIFY_HOLD) || (!port_status)) {
8             wait (connect_timer >= NOTIFY_HOLD || PMD_CONTROL.indication(PMD_status)
9                 if (connect_timer) < NOTIFY_HOLD) port_status = PMD_status & TpBias
10            }
11        else { // Beta mode - wait to get a SUSPEND handshake
12            handshake = FALSE
13            while (!handshake) {
14                control_word = NULL
15                wait_event (BPORT_CLOCK.indication(),
16                    BPORT_CONTROL.indication(control_word));
17                if (control_word = NULL) BPORT_CONTROL.request(SUSPEND);
18                else handshake = control_word == SUSPEND;
19            }
20        if (!port_status) //TpBias driven low by target
21            PMD_CONTROL.request(disable); // drive TpBias
22        }
23    }
24 }

```

### 9.5.9 Suspend failed

```

25 // can only get here in DS mode???
26 void suspend_failed_actions () { // legacy target
27     if (chg_int_en)
28         PH_EVENT.ind (INTERRUPT); // notify higher layers of fault
29     port_power = FALSE;
30 }
31
32
33
34

```

### 9.5.10 Suspend target

```

35 void suspend_target_actions () {
36     suspend_target = FALSE;
37     connect_timer = 0; // start the timer
38     if (Beta_mode) {
39         // TBD
40     }
41     else { // DS mode
42         TpGen = LOW; // drive TpBias low to Initiator
43         wait ((connect_timer >= BIAS_HOLD) || (!port_status));
44         // wait for initiator to drive TpBias
45     }
46 }
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66

```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66

**9.6 Node states**

**9.6.1 Port register RX**

**9.6.2 Suspend Target**

**9.6.3 Core suspended**

**9.6.4 Core resume**

**9.6.5 Disable received**

**9.7 Rationale (informative)**

Debounce

We stay in this state until we detect a connection. We transmit a tone on TPB and listen on TPA. TPA is set to high impedance.

A 1394-1995 node at the far end will assert TpBias on its TPA, but will not see TpBias on its TPB, and so will not think it is connected. A P1394a node on the other end will sense con\_status and start its debounce timeouts. We must ensure that we move into the Debounce state in less time than a CONNECT\_DEBOUNCE.

If we are a bilingual port, we monitor for Connect\_status (on TPA)

The set of possible port connections that a bilingual port may be connected to are:-

**Figure 9-1—Connect Status value in various connection scenarios**

	Connect_status	tone exchange	action
No connection	No	fails	stay in P2
DC connection to P1394a	Yes	fails	set DS
DC connection to bilingual	Yes	succeeds	set Beta
DC connection to Beta only	probably	succeeds	set Beta
AC connection to bilingual	No	succeeds	set Beta
AC connection to Beta only	No	succeeds	set Beta
DC connection to optical transceiver	No - must be biased so as not to trigger the Con detector	fails	stay in P2
AC connection to optical transceiver	No	fails	stay in P2

In the case that the answer is yes, we go to the next state.

In the case that the answer is No (see above table for when this will happen) we start to ping. If we hear a ping or we see connect status, then we go to the next state. This ping operation must consume very little power.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66