

## Proposed P1394b Arbitration C-Code

```
typedef union {
    struct {
        union {
            quadlet dataQuadlet;
            dataBit dataBits[32];
            unsigned char dataBytes[4];
            struct { // First self-ID packet
                quadlet type:2;
                quadlet phy_ID:6; // Physical_ID
                quadlet :1; // Always 0 for first self-ID packet
                quadlet L:1; // Link active
                quadlet gap_cnt:6; // Gap count
                quadlet sp:2; // Speed code
                quadlet :2;
                quadlet c:1; // Isochronous resource manager contender
                quadlet pwr:3; // Power class
                quadlet p0:2; // Port 0 connection status
                quadlet p1:2; // Port 1 connection status
                quadlet p2:2; // Port 2 connection status
                quadlet i:1; // Initiated reset
                quadlet m:1; // More self-ID packets...
            };
            struct { // Subsequent self-ID packets
                quadlet :8;
                quadlet ext:1; // Nonzero for second and subsequent self-ID packets
                quadlet n:3; // Sequence number
                quadlet :2;
                quadlet pa:2; // Port connection status...
                quadlet pb:2; // pa pb pc pd pe pf pg ph
                quadlet pc:2; // Self-ID packet 2 P3 P4 P5 P6 P7 P8 P9 P10
                quadlet pd:2; // Self-ID packet 3 P11 P12 P13 P14 P15 --- --- ---
                quadlet pe:2;
                quadlet pf:2;
                quadlet pg:2;
                quadlet ph:2;
                quadlet :2;
            };
            struct { // PHY configuration packet
                quadlet :2;
                quadlet root_ID:6; // Intended root
                quadlet R:1; // If set, root_ID field is valid
                quadlet T:1; // If set, gap_cnt field is valid
                quadlet gap_cnt:6; // Gap count
                quadlet :16;
            };
            struct { // Extended PHY packets (ping and other remote packets)
                quadlet :2;
                quadlet :6; // Physical_ID
                quadlet ext_type:4; // Extended type
                quadlet page:3; // Page_select
                quadlet port:4; // Port_select
                quadlet reg:3; // Register address (add 0b1000 if paged register)
                union {
                    quadlet data:8; // Register data (remote reply)
                    struct { // Remote confirmation
                        quadlet fault:1; // Copies of equivalent PHY register bits...
                        quadlet connected:1;
                        quadlet bias:1;
                        quadlet disabled:1;
                        quadlet ok:1; // Confirm command accepted or not
                        quadlet cmnd:1; // Remote command
                    };
                };
            };
        };
    };
    union {
        quadlet checkQuadlet;
        dataBit checkBits[32];
    };
};
```

## Proposed P1394b Arbitration C-Code

```
} PHY_PKT;

int out_req(int port){
    //combine best asynch and isoch request from other ports and own request
    //return(request)
}

void idle_actions() {
    while !(new_req || IMM_REQ || receive_grant || data_coming)
        for (i=1; i<NPORT; i++) {
            bportT[i] = out_req(i); // get current request state for each port
            BPORT_CONTROL.request(i); // send request
        }
}

void new_request_actions() { //new request received from link
    //update own_request with new request from link
    //combine latest isoch and asynch requests in own_request
}

void grant_actions() {
    // if (root)
    //     wait for a request
    // if (own_req > best_request)
    //     grant_self = TRUE;
    // else if (best_request > 0)
    //     grant the port with the best request
    // else
    //     grant parent port
    //     send request code on all ports
    // if arb_reset
    //     send arb_reset on all ports
}

void start_rx_packet() { // Send data prefix and do speed signaling
    // currently for beta only environment and doesn't handle error cases
    // S400 and below have different data prefix requirements in mixed environment

    arb_timer = 0;
    //wait_event(BPORT_SPEED.indication);
    wait_event(BPORT_DATA.indication);
    // center FIFO by waiting for data to arrive before transmitting data_prefix?
    for (i=0; i<NPORT; i++) {
        if (i <> receive_port) {
            bportTspeed[i] = bportRspeed[receive_port];
            BPORT_SPEED.request(i); // For beta transmission, this serves as data prefix
        }
        bportT[receive_port] = out_req(receive_port);
        BPORT_CONTROL.request(receive_port); // Send requests on receive port
    }
}

void receive_actions() {
    // currently doesn't handle several error cases
    unsigned byte_count = 0, i;
    PHY_packet rx_phy_pkt;

    concatenated_packet = FALSE;
    PH_DATA.indication(DATA_PREFIX); // Send notification of bus activity
    start_rx_packet(); // Start up receiver and repeater and repeat data prefix and speed signal
    PH_DATA.indication(DATA_START, bportRspeed(receive_port)); // Send speed indication
    do {
        if (test_event(BPORT_DATA.indication)) { // Normal data, send to link layer and repeat
            PH_DATA.indication(data_byte);
            for (i=0; i<NPORT; i++) // Repeat out other ports
                if (i <> receive_port) {
                    bportT[i] = bportR[receive_port];
                    BPORT_DATA.request(i);
                }
            bportT[receive_port] = out_req(receive_port);
        }
    }
}
```

## Proposed P1394b Arbitration C-Code

```
        BPORT_CONTROL.request(receive_port); // Send requests on receive port
        if (byte_count < 8) // Accumulate first 8 bytes
            rx_phy_pkt.bytes[byte_count] = bportR[receive_port];
        byte_count++;
        wait_event(BPORT_CLOCK.indication);
    }
} while (!test_event(BPORT_CONTROL.indication));

switch(bportR[receive_port]) { // Check end of packet indicator
    case DATA_PREFIX:
        concatenated_packet = TRUE; // Re-entering receive_actions() will handle DATA_PREFIX
        break;

    case DATA_END:
        PH_DATA.indication(DATA_END); // Normal end of packet
        for (i=0; i<NPORT; i++) // Repeat out other ports
            if ((i <> receive_port) && active[i]) {
                bportT[i] = DATA_END;
                BPORT_CONTROL.request(i);
            }
        bportT[receive_port] = out_req(receive_port);
        BPORT_CONTROL.request(receive_port); // Send requests on receive port
        break;

    case DATA_END_ERR:
        PH_DATA.indication(DATA_END_ERR); // End of bad packet
        for (i=0; i<NPORT; i++) // Repeat out other ports
            if ((i <> receive_port) && active[i]) {
                bportT[i] = DATA_END_ERR;
                BPORT_CONTROL.request(i);
            }
        bportT[receive_port] = out_req(receive_port);
        BPORT_CONTROL.request(receive_port); // Send requests on receive port
        break;
}

receive_grant = FALSE;
wait_event(BPORT_CLOCK.indication);
if (test_event(BPORT_CONTROL.indication)) {
    switch(bportR[receive_port]) { // Check grant indicators
        case GRANT:
            receive_grant = TRUE;
            break;

        ARBRST_GRANT:
            receive_grant = TRUE;
            arb_reset = TRUE;
            PH_DATA.indication(ARBITRATION_RESET_GAP); // Alert link
            break;

        case ARBRST:
            arb_reset = TRUE;
            PH_DATA.indication(ARBITRATION_RESET_GAP); // Alert link
            for (i=0; i<NPORT; i++) // Repeat out other ports
                if (i <> receive_port) {
                    bportT[i] = ARBRST;
                    BPORT_CONTROL.request(i);
                }
            bportT[receive_port] = out_req(receive_port);
            BPORT_CONTROL.request(receive_port); // Send request on the receive port
            break;
    }
}

if ((byte_count == 8) && (rx_phy_pkt.data.Quadlet == ~rx_phy_pkt.check.Quadlet)) // Check PHY packet for good format
    decode_phy_packet(rx_phy_pkt); // Parse valid phy packets
}

void start_tx_packet(speedcode speed) { // Send data prefix and do speed signaling
```

## Proposed P1394b Arbitration C-Code

```
// currently for beta only environment and doesn't handle error cases
// S400 and below have different data prefix requirements in mixed environment
int signal_port = NPORT, i;

for (i = 0; i < NPORT; i++) {
    if (phy_response) {
        bportTspeed[i] = speed; // Almost always S100
        BPORT_SPEED.request(i); // For beta transmission, this serves as data prefix
    } else if (disable_notify[i]) {
        bportTspeed[signal_port=i] = DISABLE_NOTIFY;
        BPORT_CONTROL.request(i);
    } else if (suspend[i]) {
        bportTspeed[signal_port=i] = SUSPEND;
        BPORT_CONTROL.request(i);
    } else {
        bportTspeed[i] = speed;
        BPORT_SPEED.request(i); // For beta transmission, this serves as data prefix
    }
}

void transmit_actions() { // Send a packet as link transfers it to the PHY
    int bit_count = 0, i;
    boolean end_of_packet = FALSE;
    phyData data_to_transmit;
    PHY_packet tx_phy_pkt;

    receive_port = NPORT; // Impossible port number ==> PHY transmitting
    start_tx_packet(speed); // Send data prefix & speed signal
    // Assume speed has been set correctly by link from PH_ARB.request
    if (isbr) // Avoid phantom packets...
        return;
    PH_ARB.confirmation(WON); // Signal grant on Ctl[0:1]
    while (!end_of_packet) {
        PH_CLOCK.indication(); // Tell link to send data
        data_to_transmit = PH_DATA.request(); // Wait for data from the link
        if (data_to_transmit == DATA_END) {
            end_of_packet = TRUE; // End of packet indicator
            for (i=0; i<NPORT; i++) { // Send DATA_END on all ports
                bportT[i] = DATA_END;
                BPORT_CONTROL.request(i);
            }
        } else {
            for (i=0; i<NPORT; i++) { // Send DATA on all ports
                bportT[i] = data_to_transmit;
                BPORT_CONTROL.request(i);
            }
            if (byte_count < 8) // Accumulate possible PHY packet
                rx_phy_pkt.bytes[byte_count] = data_to_transmit;
            byte_count++;
        }
    }
    if ((byte_count == 8) && (tx_phy_pkt.data.Quadlet == ~tx_phy_pkt.check.Quadlet)) // Check PHY packet for good format
        decode_phy_packet(tx_phy_pkt); // Parse valid phy packets
}
```