

## p1394a Proposal

BRC,

Following is the busyA/B retry protocol clarifications proposed at the Oregon 22Feb98 BRC meeting. My assumption is that option A and option B will be posted for review. Based on the last meeting, option A may be incorporated into p1394a, and option B text will be saved for incorporation in a 1394TA white paper.

DVJ

-----  
All,

As per our 02Feb99 conversation, here is a document to be added to the p1394a list of documents. We assume there will be appropriate editorial license to integrate the text. I would prefer to have the state transition remain a table (rather than a stick figure), as I believe its easier to read and maintain.

FYI, opinions on the second option are mixed. Farrell feels it provides a useful reference design; I would just as soon discard it. However, we both feel (I think) that the first design should be the recommended design. I'm assuming Farrell will correct me if I have misinterpreted his statements...

DVJ

```
*****
* Dr. David V. James           * Tel: (408) 955-6295           *
* Sony Research Laboratories   * Fax: (408) 955-5180           *
* LSI Systems Laboratory      * email: davej@lsi.sel.sony.com *
* 3300 Zanker Road, M/S SJ-3D3 *      dvj@ibm.net (home)       *
* San Jose, CA 95134          *                                  *
*****
```

All,

Regarding the "second option", it is believed to be the behavior of an inbound dual-phase node that was intended to be specified by IEEE 1394-1995. As such, if there is general agreement that it is the right 1394-1995 model, it can be used by implementors (and their customers) to check the adequacy of pre-1394a implementations; so it seems it would be good to have it published in some place.

(There was also a version of it that includes the reservation-counting optimization; Dave discovered a weakness in it in that it does not recover as gracefully from a spuriously injected reservation claimed by some arbitrary sender as does the recommend "first option", which benefits from being designed with reservation counting in mind -- instead of as an afterthought.)

Regards,  
Farrell Ostler

-----  
Philips Semiconductors | MS55 | 9201 Pan American Fwy NE | Albuquerque NM 87113  
Tel: 505-822-7791 | Fax: 505-822-7836  
email: Farrell.Ostler@abq.sc.philips.com | seri: ostler@sphp01  
-----

## 1. Congestion management

Revisions to the inbound busyA/B retry state machines (page 191, 1394-1995), and clarifications to the busyA/B outbound state machines, are proposed for the following reasons:

- 1) No resynchronization. The existing protocols doesn't automatically resync when outbound and inbound state machines have inconsistent reservation histories. State machines need to remain wait for a retry-timeout so that any outstanding (but unaccounted for) reservations will be serviced.
- 2) Retry timeouts. The retry timeout for the outbound queue (once every 4 arbitration intervals) is inconsistent with that of the inbound queue (once every 3 arbitration intervals) based on some interpretations.
- 3) No counts. Its unclear how to extend the current specification to incorporate reservation counts.

To fix these known problems and clarify the definition, new inbound state machines are proposed for p1394a. Both of these are thought to be correct and complete. Option A is preferred by the authors of these proposals (Farrell Ostler and David James), but option B is more consistent with past nomenclature and may therefore be more acceptable to reviewers.

### 1.1 Receive-queue reservations

#### 1.1.1 Receive-queue reservations

To ensure forward progress, fair arbitration protocols and fair acceptance protocols are necessary. Fair arbitration protocols allow each nodes to transmit packets; fair acceptance protocols ensures that transmitted packets will eventually be accepted, rather than busied. Acceptance protocol fairness is based on reserving future queue space to older sets of requests, where the age of a request is based on the time of its initial retry.

Allocation reservations have timeouts, so that the reservation can be reclaimed in the absence of retries. Reservation timeouts may occur when an active node is reset, when transmission errors corrupt the packet header, or if the packet can't be transmitted before its time-of-death is reached. The minimal interval between retries is specified, to avoid falsely triggering one of these missing-retry timeouts.

For fairness, each outbound queue shall eventually tag its oldest request and its oldest response, with a reservation-requested label. A high performance outbound queue may tag multiple requests (or responses) with reservation-requested labels, if each of these is directed to a different target on the local bus.

The basis of the reservation protocols is as follows:

- 1) Age labels. The outbound queue initially labels its oldest retries with a reservation-requested retry\_1 label; newer requests are labeled with a retry\_X label.
- 2) Assignment. Reservations are assigned to retry\_1 packets, by returning an ack\_busy\_A or ack\_busy\_B label.
- 3) Servicing. The oldest of the retry\_A/retry\_B packet retries are allowed to consume queue space while others are busied.

## 1.2 Outbound reservation request

The outbound queue's handling of transmission labels is specified by the state machine of table 1. Although not listed in this table, the outbound queue shall retry a previously busied subaction with no more than AC intervening arbitration gaps.

**Table 1—State transition table for reservation assertion**

inputs		Row	results	
old state	ack-returned condition		transmitted send.rt	new state
—	Reset completion	1	—	DONE
DONE	not necessarily oldest packet available	2	retry_X	SEND
	known oldest packet available	3	retry_1	
SEND	TimeOfDeath()  !AckBusy();	4	—	DONE
	oldest&&ack_busy_X	5	retry_1	SEND
	oldest&&ack_busy_A	6	retry_A	
	oldest&&ack_busy_B	7	retry_B	
!oldest&&AckBusy();	8	retry_X		

Request and response reservations shall be processed independently, so that two packets (the oldest request and response) can have reservations. The 4 fairness-interval period limit allows outbound queues to periodically transmit oldest request, oldest response, newer request, and newer response. Sending of newer subactions, particularly ones with different destination\_ID addresses, reduces the effects of a congested inbound queue on unrelated traffic.

**Row 1:** Reservation assignments are discarded during a bus reset.

**Row 2:** Before being retried, a not known-to-be oldest packet is sent with a retry\_X indication.

**Row 3:** Before being retried, a known-to-be oldest packet is sent with a retry\_1 indication.

**Row 4:** The packet is no longer retried when its time-of-death timeout has been exceeded or when a busy acknowledge (ack\_busy\_X, ack\_busy\_A, or ack\_busyB) is not returned.

**Row 5:** The oldest packet changes from retry\_X for retry\_1 when busied for the first time. The intent is to request a reservation on the next retry.

**Row 6:** The oldest packet inherits the ack\_busy\_A acknowledge, accepting this reservation assignment.

**Row 7:** The oldest packet inherits the ack\_busy\_B acknowledge, accepting this reservation assignment.

**Row 8:** The newer packets continue to retry with their initial retry\_X label.

### 1.3 Inbound reservation filter, option A

The inbound reservation filter has two states: USE\_A and USE\_B. The 'A' and 'B' reservations have precedence when in the USE\_A and USE\_B states respectively, as specified in table 2. Request and response packets pass through separate independent reservation filters, to avoid queue-dependency livelocks and starvation conditions. Reservation filters remain in these states for a minimum of four arbitration intervals, so that any outstanding (but unaccounted for) reservations will be serviced.

**Table 2—Inbound reservation filter, design A**

old state	condition	Row	action	ack	new state
—	Reset completion	1	ra=rb=ac=0	—	USE_A
USE_A	Queue()!=FULL&&send.rt==retry_A	2	Sub(ra,RC)	AckDone()	USE_A
	Queue()!=FULL&&ra==0&&send.rt==retry_B	3	Sub(rb,RC)		
	Queue()!=FULL&&ra==0&&send.rt==retry_1	4	—		
	Queue()!=FULL&&ra==0&&send.rt==retry_X	5	—	ack_busy_A	
	Queue()==FULL&&ra!=0&&send.rt==retry_A	6	ac=0		
	Queue()==FULL&&ra==0&&send.rt==retry_A	7	ac=0,Add(ra)	ack_busy_B	
	!(Queue()!=FULL&&ra==0)&&send.rt=retry_B	8	—		
	!(Queue()!=FULL&&ra==0)&&send.rt=retry_1	9	Add(rb,RC)		
	!(Queue()!=FULL&&ra==0)&&send.rt==retry_X	10	—	ack_busy_X	
	ArbResetGap&&ac!=AC	11	ac+= 1;	—	
	ArbResetGap&&ac==AC	12	ac=1,ra=0;	—	
USE_B	Queue()!=FULL&&send.rt==retry_B	13	Sub(rb,RC)	AckDone()	USE_B
	Queue()!=FULL&&rb==0&&send.rt==retry_A	14	Sub(ra,RC)		
	Queue()!=FULL&&rb==0&&send.rt==retry_1	15	—		
	Queue()!=FULL&&rb==0&&send.rt==retry_X	16	—	ack_busy_B	
	Queue()==FULL&&rb!=0&&send.rt=retry_B	17	ac=0		
	Queue()==FULL&&rb==0&&send.rt=retry_B	18	ac=0,Add(rb)	ack_busy_A	
	!(Queue()!=FULL&&rb==0)&&send.rt=retry_A	19	—		
	!(Queue()!=FULL&&rb==0)&&send.rt=retry_1	20	Add(ra,RC)		
	!(Queue()!=FULL&&rb==0)&&send.rt==retry_X	21	—	ack_busy_X	
	ArbResetGap&&ac!=AC	22	ac+= 1;	—	
	ArbResetGap&&ac==AC	23	ac=1,rb=0;	—	

Notes:

```
#define AC 4 // Reservation timeout interval
#define Add(a,b) (a+= (a!=b))
#define Sub(a,b) (a-= (a!=b&&a!=0))
// RC is implementation dependent
```

The *ac* counter counts the number of consecutive arbitration gaps, and is limited by the *AC* value that specifies the outbound queue's worst-case retry delay.

The *ra* counter counts the number of *A* reservation assignments; the *rb* counter counts the number of *B* reservation assignments. The *RC* value, that specifies the size of these counters, is implementation dependent; in the absence of a counter, the state machines shall behave as though *RC* were equal to 1.

NOTE—An 8-bit counter is expected to be sufficient to handle the largest number of reservation requests envisioned by the currently active 1394 related standards.

**Row 1:** Reservation assignments are discarded during a bus reset.

**Row 2, row 13:** Current reservation-assigned packets are always accepted.

**Row 3, row 14:** Later reservation-assigned packets are accepted, if current reservations have been serviced.

**Row 4, row 15:** Later reservation-requested packets are accepted, if current reservations have been serviced.

**Row 5, row 16:** Reservationless packets are accepted, if current reservations have been serviced.

**Row 6, row 17:** A busied current reservation-assigned packet clears the retry timeout counter.

**Row 7, row 18:** Unexpected current reservation assignments are honored.

**Row 8, row 19:** Later reservations are busied, while current reservations are outstanding.

**Row 9, row 20:** When full or reserved, a reservation-requested packets obtains a later reservation.

**Row 10, row 21:** When full or reserved, a reservationless packets is busied without reservations.

**Row 11, row 22:** Reservation timeout interval is measured in units of arbitration reset gaps.

**Row 12, row 23:** Later reservations become current when the reservation timeout is reached.

## 1.4 Inbound reservation filter, option B

This reservation filter has four states, corresponding to the four states in the 1394-1995 specification, as documented in table 3. Acceptance filters remain in the IRD1 and IRD3 states for a minimum of four arbitration intervals, so that any outstanding (but unaccounted for) reservations will be serviced.

**Table 3—Consumer reservation filter, design B**

old state	condition	Row	action	ack	new state
—	Reset completion	1	—	—	IRD0
IRD0 (accept all except retry_B)	Queue()!=FULL&&send.rt==retry_X	2	—	AckDone()	IRD0
	Queue()!=FULL&&send.rt==retry_1	3			
	Queue()!=FULL&&send.rt==retry_A	4			
	Queue()==FULL&&send.rt==retry_X	5	—	ack_busy_X	IRD1
	send.rt==retry_B	6	—	ack_busy_A	
	Queue()==FULL&&send.rt==retry_1	7	ac=0;	—	
	Queue()==FULL&&send.rt==retry_A	8			
IRD1 (accept retry_A only)	send.rt==retry_X	9	—	ack_busy_X	IRD1
	send.rt==retry_1	10	—	ack_busy_B	
	send.rt==retry_B	11	—	—	
	Queue()!=FULL&&send.rt==retry_A	12	—	AckDone()	IRD2
	Queue()==FULL&&send.rt==retry_A	13	ac=0;	ack_busy_A	
	ArbResetGap&&ac!=AC	14	ac+=1;	—	
	(ArbResetGap&&ac==AC)	15	—	—	
IRD2 (accept all except retry_A)	Queue()!=FULL&&send.rt==retry_X	16	—	AckDone()	IRD2
	Queue()!=FULL&&send.rt==retry_1	17			
	Queue()!=FULL&&send.rt==retry_B	18			
	Queue()==FULL&&send.rt==retry_X	19	—	ack_busy_X	IRD3
	send.rt==retry_A	20	—	ack_busy_B	
	Queue()==FULL&&send.rt==retry_1	21	ac=0;	—	
	Queue()==FULL&&send.rt==retry_B	22			
IRD3 (accept retry_B only)	send.rt==retry_X	23	—	ack_busy_X	IRD3
	send.rt==retry_1	24	—	ack_busy_A	
	send.rt==retry_A	25	—	—	
	Queue()!=FULL&&send.rt==retry_B	26	—	AckDone()	IRD0
	Queue()==FULL&&send.rt==retry_B	27	ac=0;	ack_busy_B	
	ArbResetGap&&ac!=AC	28	ac+=1;	—	
	(ArbResetGap&&ac==AC)	29	—	—	

Notes:

```
#define AC 4 // Reservation timeout interval
```

The *ac* counter counts the number of consecutive arbitration gaps, and is limited by the *AC* value that specifies the producer's worst-case retry delay.

Proposed A/B retry

The *ra* counter counts the number of A reservation assignments; the *rb* counter counts the number of B reservation assignments. The *RC* value, that specifies the size of these counters, is implementation dependent; in the absence of a counter, the state machines shall behave as though *RC* were equal to 1.

**Row 1:** Reservation assignments are discarded during a bus reset.

**Row 2, row 16:** Reservationless packets are accepted.

**Row 3, row 17:** Reservation-requested packets are accepted.

**Row 4, row 18:** Reservation-assigned packets are accepted; reservation count is adjusted.

**Row 5, row 19:** Reservationless packet is busied, while space is unavailable.

**Row 6, row 20:** Reservation-missed packets gets current reservation assignment, servicing begins.

**Row 7, row 21:** Reservation-requested packet gets reservation assignment, servicing begins.

**Row 8, row 22:** Reservation-assigned packet keep their reservation, servicing begins.

**Row 9, row 23:** Reservationless packets are busied.

**Row 10, row 24:** Reservation-requested packet gets reservation assignment, but is busied.

**Row 11, row 25:** Reservation-assigned packet keeps it later reservation.

**Row 12, row 26:** Reservation-assigned packets are accepted.

**Row 13, row 27:** Reservation-assigned packet is busied, when space is unavailable.

**Row 14, row 28:** Arbitration interval timeout is measured in units of arbitration gaps.

**Row 15, row 29:** Later reservations become current when the reservation timeout is reached.