From: Dr. David V. James
Email: davej@lsi.sel.sony.com
Phone: +1.408.955.6295
To: p1394a Ballot Response Committee
Data: February 2, 1999
RE: BusyA/B retry, Document #99-002r0

Attached is a proposal for revisions of the A/B retry protocols, which is believed to fix perceived problems with the existing 1394-1995 specification. At the last meeting, I accepted the action item of generating this type of proposal, with the assistance of Farrell Ostler and Jerry Hauck.

Farrell and I ended up generating this proposal, after several days of iterative designs. To our belief, it fulfills the requirements of Jerry Hauck.

However, Jerry was unavailable during this time and has therefore not approved this draft. To expedite the process, we are posting this draft for general BRC review.

DVJ

1. Congestion management

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 producer shall eventually tag its oldest request and its oldest response, with a reservation-requested label. A high performance producer may tag multiple requests (or responses) with reservation-requested labels, if each of these is directed to a different consumer.

The basis of the reservation protocols is as follows:

1) Age labels. The producer 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 other are busied.

1.2 Producer reservation request

The producer's handling of transmission labels is specified by the state machine of table 1. Although not listed in this table, the producer 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**		**new state**	**transmitted send.rt**
—	Reset completion	1	DONE	—
DONE	newer packet available	2	SEND	retry_X
SEND	TimeOfDeath()||!AckBusy();	3	DONE	—
	oldest&&ack_busy_X	4	—	retry_1
	oldest&&ack_busy_A	5	—	retry_A
	oldest&&ack_busy_B	6	—	retry_B
	!oldest&&AckBusy();	7	—	retry_X

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

Row 2: The packet is initially sent with a retry_X indication.

Row 3: 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 4: 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 5: The oldest packet inherits the ack_busy_A acknowledge, accepting this reservation assignment.

Row 6: The oldest packet inherits the ack_busy_B acknowledge, accepting this reservation assignment.

Row 7: The newer packets continue to retry with their initial retry_X label.

1.3 Consumer reservation filters

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

1) No resynchronization. The existing protocols doesn't automatically resync when producer and consumer 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 producer (once every 4 arbitration intervals) is inconsistent with that of the consumer (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. We propose to post both to the reflector, to solicit comments from a broader audience.

1.3.1 Inbound reservation filter, design A

This 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. Acceptance 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—Consumer 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			
	Queue()==FULL&&ra!=0&&send.rt==retry_A	6	ac=0	ack_busy_A	
	Queue()==FULL&&ra==0&&send.rt==retry_A	7	ac=0,Add(ra)		
	!(Queue()!=FULL&&ra==0)&&send.rt=retry_B	8	—	ack_busy_B	
	!(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
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			
	Queue()==FULL&&rb!=0&&send.rt=retry_B	17	ac=0	ack_busy_B	
	Queue()==FULL&&rb==0&&send.rt=retry_B	18	ac=0,Add(rb)		
	!(Queue()!=FULL&&rb==0)&&send.rt=retry_A	19	—	ack_busy_A	
	!(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;	—	USE_A

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 producer'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.

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.

1.3.2 Inbound reservation filter, design 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	
	send.rt==retry_B	6	—	ack_busy_A	IRD1
	Queue()==FULL&&send.rt==retry_1	7			
	Queue()==FULL&&send.rt==retry_A	8	ac=0;		
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()	
	Queue()==FULL&&send.rt==retry_A	13	ac=0;	ack_busy_A	
	ArbResetGap&&ac!=AC	14	ac+=1;	—	
	(ArbResetGap&&ac==AC)	15	—	—	IRD2
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	
	send.rt==retry_A	20	—	ack_busy_B	IRD3
	Queue()==FULL&&send.rt==retry_1	21			
	Queue()==FULL&&send.rt==retry_B	22	ac=0;		
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()	
	Queue()==FULL&&send.rt==retry_B	27	ac=0;	ack_busy_B	
	ArbResetGap&&ac!=AC	28	ac+=1;	—	
	(ArbResetGap&&ac==AC)	29	—	—	IRD0

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.

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.