

---

# Start-up with modified 8B10B scheme

Alistair Coles

Presented to P1394B  
August 6-8, 1997

# Background

- During Beta-mode start-up, handshake signals are exchanged between 1394B PHYs so that bit, codeword and scrambler synchronization can occur.

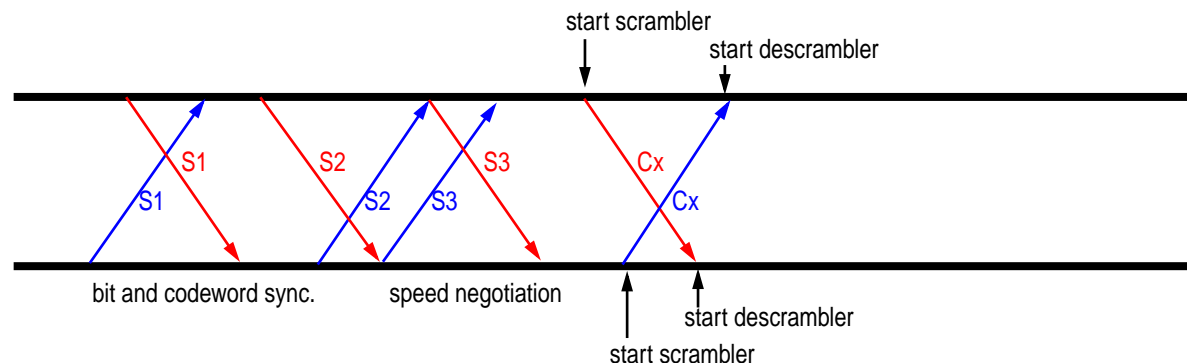
(Ref: Colin's presentation in May and section 9.2 of draft 0.04).

- In June the following handshake patterns were proposed:

$S1(ESL) = [K28.5, D10.2] = [0011111010][0101010101]$

$S2(ESR) = [K28.5, D21.5] = [0011111010][1010101010]$

- These patterns allow bit and codeword synchronization. Scrambler would sync. on first occurrence of a Cx codeword after S2.



---

# Concerns with June proposal:

- S1, S2 and S3 sequences are repetitive, causing spectral lines.

Should only last ~1us and therefore not be an EMC problem, but is there an alternative ?

- Scrambler/descrambler synchronization requires reseeding at transition from S3 to Cx's : “once and for all” process.

May be advantages to a “blind training” procedure in which rx. descrambler learns state of tx. scrambler, *regardless of state the tx. scrambler.*

---

# New proposal

- S1 (ESL) = REQUEST/GRANT (i.e. stream of Cx's derived from scrambled control state 0001).
- S2 (ESR) = IDLE (i.e. stream of Cx's derived from scrambled control state 0000).

*Bit, codeword and scrambler synchronization can be achieved during S1 and S2.*

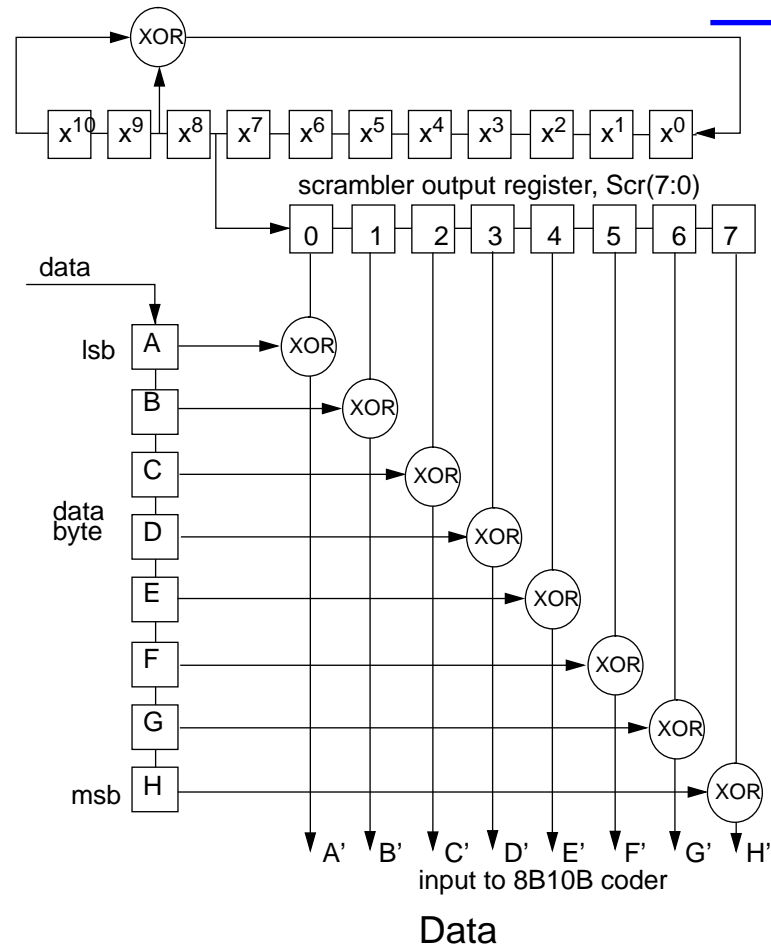
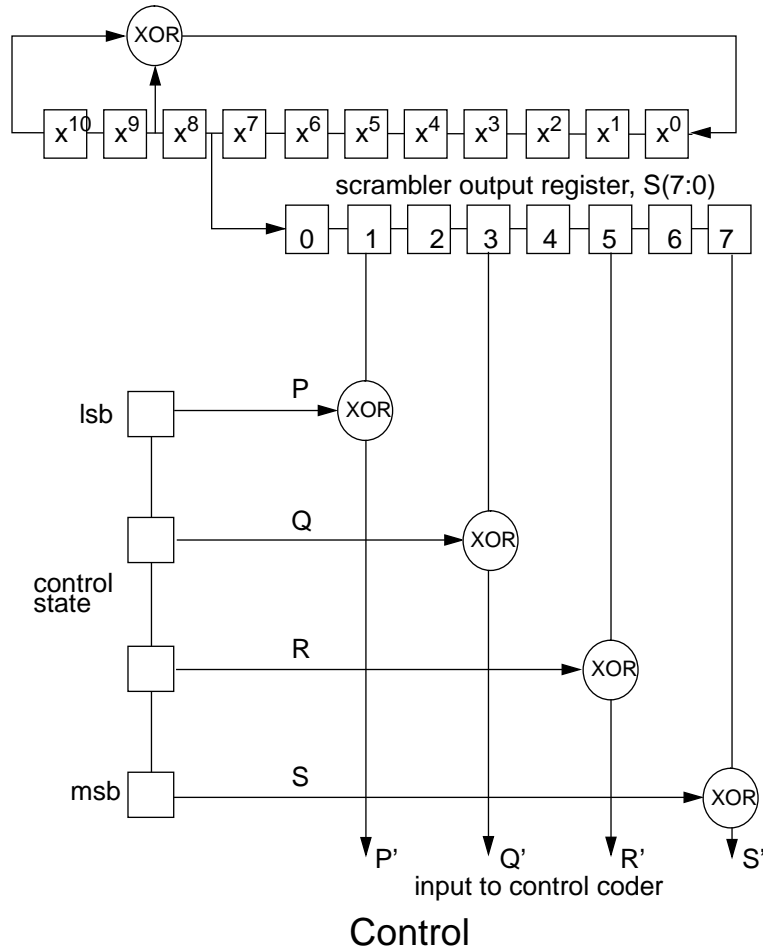
## Codeword Synchronization

- The control codeword set (C0 -> C15) contains the following comma characters that can be used to identify the codeword boundary:

C4 = 0010001111

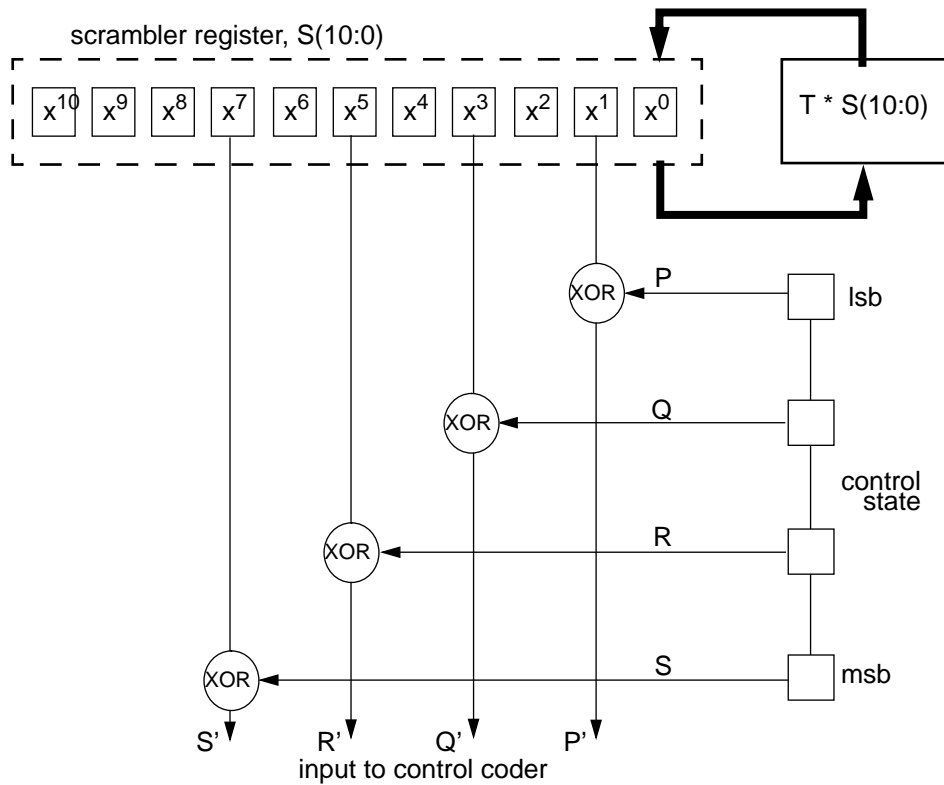
C11 = 1101110000

# Scrambler (1)



Note: Shown separately for clarity. In practice, a single scrambler performs both functions

# Scrambler (2)



Scrambler can also be implemented in parallel.

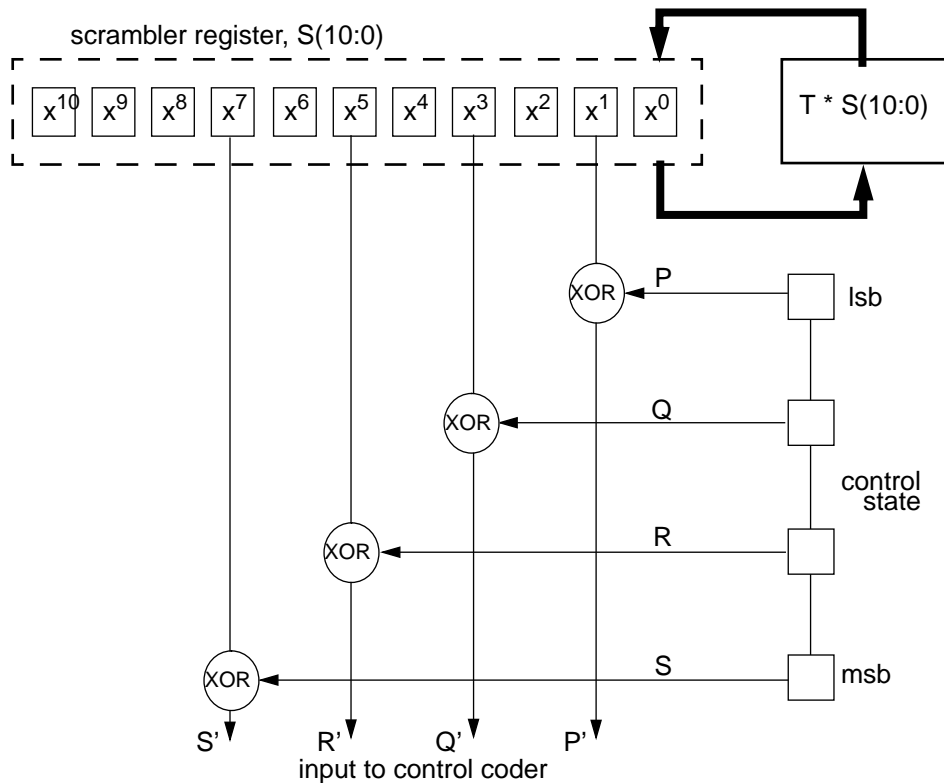
The update function T for the  $x^{11}+x^9$  scrambler is defined by:

$$\begin{aligned}
 S_{k+1}(10) &= S_k(2) \\
 S_{k+1}(9) &= S_k(1) \\
 S_{k+1}(8) &= S_k(0) \\
 S_{k+1}(7) &= S_k(10) \wedge S_k(8) \\
 S_{k+1}(6) &= S_k(9) \wedge S_k(7) \\
 S_{k+1}(5) &= S_k(8) \wedge S_k(6) \\
 S_{k+1}(4) &= S_k(7) \wedge S_k(5) \\
 S_{k+1}(3) &= S_k(6) \wedge S_k(4) \\
 S_{k+1}(2) &= S_k(5) \wedge S_k(3) \\
 S_{k+1}(1) &= S_k(4) \wedge S_k(2) \\
 S_{k+1}(0) &= S_k(3) \wedge S_k(1)
 \end{aligned}$$

$S_k(i)$  is state of  $i$ th bit of scrambler register at time  $k$ .

$\wedge$  = exclusive OR.

# Scrambler output during start up



	$S1$ (REQUEST)	$S2$ (IDLE)
SRQP	0001	0000
$S'R'Q'P'$	$S(7), S(5), S(3), \overline{S(1)}$	$S(7), S(5), S(3), S(1)$

While  $S1$  and  $S2$  are transmitted, scrambled control values contain regular samples of scrambler output register.

These samples can be used to synchronize descrambler in receiver.

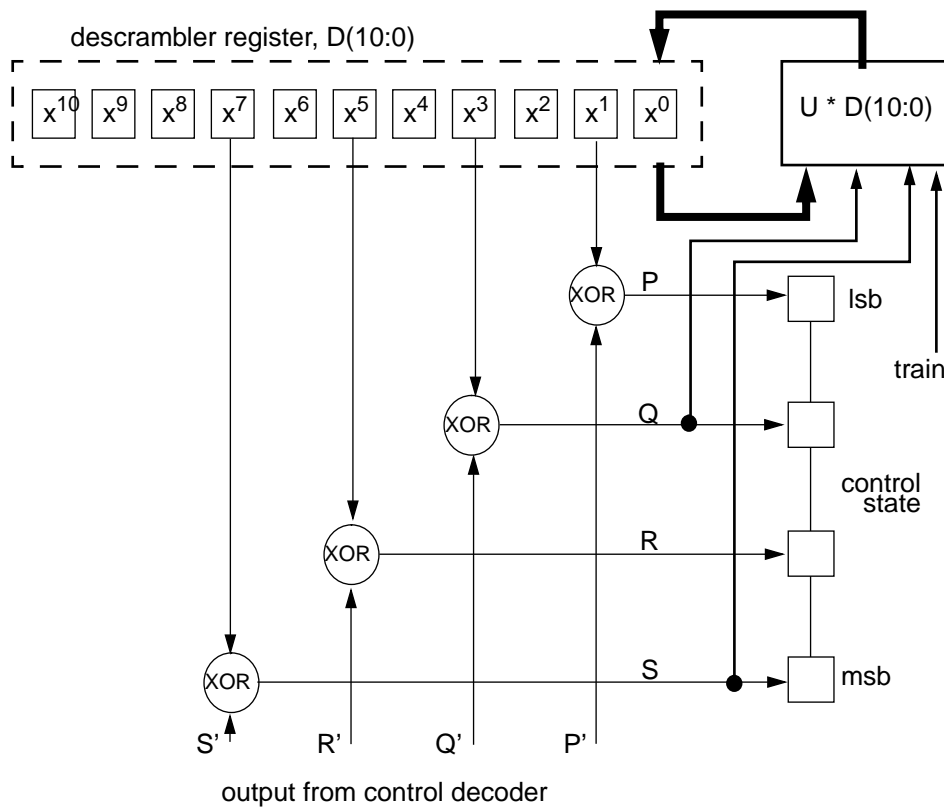
---

# — Descrambler Synchronization

- During start up (and at any other time IDLE is received), decoded control values contain regular samples of transmitter scrambler state. (2 samples per received codeword).
- These samples can be used to synchronize the descrambler, using extremely simple hardware.
- 11 samples of scrambler state are required to guarantee synchronization of descrambler, i.e. synchronization achieved in 6 codewords duration.
- This is an example of a distributed sample scrambler.

See: “Synchronization of shift register generators in distributed sample scramblers”, Kim, S.C. and Lee, B. G., IEEE Trans. Comms., vol.42, Feb 1994, pp.1400-1408.

# Descrambler



During REQUEST and IDLE, S and Q are samples of tx. scrambler, i.e.  $S_k(7)$  and  $S_k(3)$ .

The update function U for the descrambler is defined by:

$$D_{k+1}(10) = D_k(2)$$

$$D_{k+1}(9) = D_k(1)$$

$$D_{k+1}(8) = D_k(0)$$

$$D_{k+1}(7) = D_k(10) \wedge D_k(8)$$

$$D_{k+1}(6) = D_k(9) \wedge D_k(7) \wedge (S \& \text{train})$$

$$D_{k+1}(5) = D_k(8) \wedge D_k(6)$$

$$D_{k+1}(4) = D_k(7) \wedge D_k(5) \wedge (S \& \text{train})$$

$$D_{k+1}(3) = D_k(6) \wedge D_k(4)$$

$$D_{k+1}(2) = D_k(5) \wedge D_k(3) \wedge (Q \& \text{train})$$

$$D_{k+1}(1) = D_k(4) \wedge D_k(2)$$

$$D_{k+1}(0) = D_k(3) \wedge D_k(1) \wedge (Q \& \text{train})$$

'train' enables descrambler training.

# Speed Negotiation

- For speed negotiation, each node issues its speed capability by sending a repeating sequence of control states of the form:

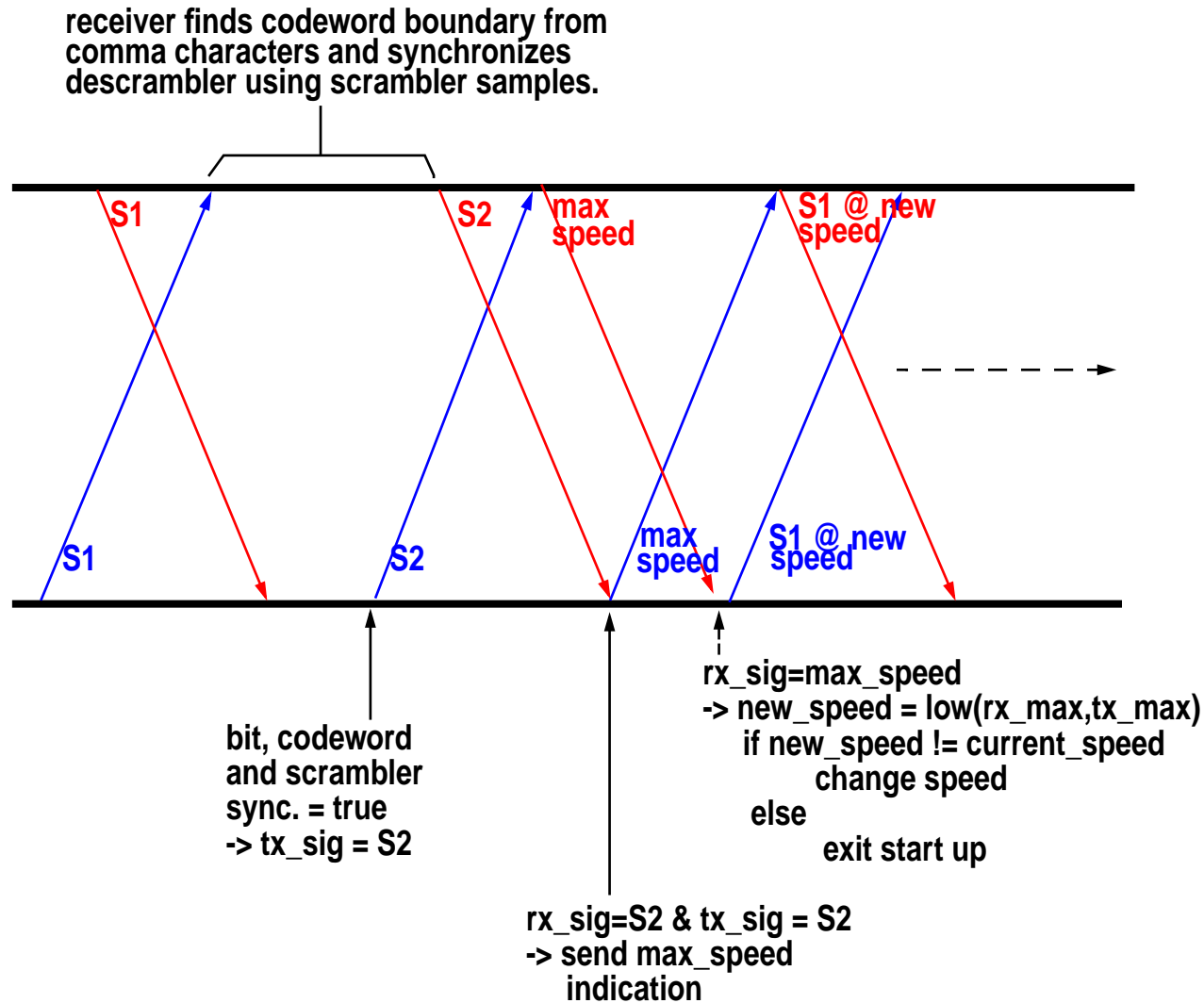
...ESCAPE SPEED+ SPEED+ SPEED+ ESCAPE SPEED+...

The number of SPEED+ states following each ESCAPE indicates the maximum speed capability of the node, in a similar way to the number of SPEED states following data prefix indicates the rate of the following packet:

S100	ESCAPE
S200	ESCAPE SPEED+
S400	ESCAPE SPEED+ SPEED+
S800	ESCAPE SPEED+ SPEED+ SPEED+
S1600	ESCAPE SPEED+ SPEED+ SPEED+ SPEED+

- Each node compares received max. speed indication with its own max. speed, and changes speed to lower of the two (unless this equals the current speed).
- ESCAPE defined as control state 1100 (currently spare).

# Start up sequence



---

# — Possible enhancement to start up

- IBM 8B10B code as implemented for Fibre Channel and Gigabit Ethernet uses K28.5 character for codeword sync.

To maintain this, insert K28.5 characters into S1 and S2 control codeword scheme (e.g. one K28.5 character in every 16 Cx characters).

- This would also allow S1 and S2 to be distinguished from IDLE and REQUEST during normal arbitration.

-> Potential for one node to request Start Up.

---

# — Root contention

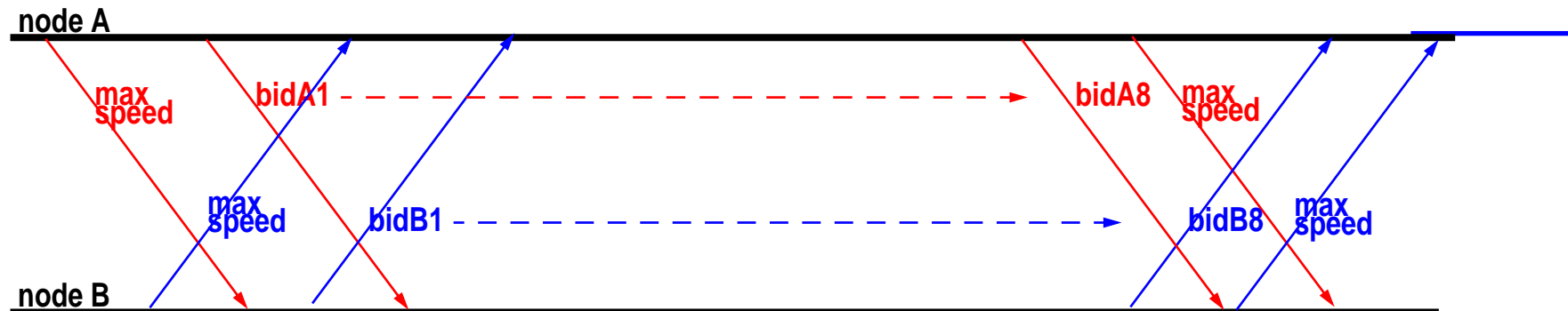
- Root contention can be resolved prospectively during start up by nodes negotiating in advance their behaviour in the event of a contention. Negotiation not needed after each Bus Reset.
- Concurrent with speed negotiation, nodes could exchange random “bids” to establish which will become root in the event of contention.

e.g. bid for “I will become root” = ESCAPE, CHILD\_NOTIFY

bid for “I will become child” = ESCAPE, PARENT\_NOTIFY

- Bids are inserted in max. speed indication stream. After issuing first bid, each node waits to receive a bid.
  - If the two bids are different, then contention is resolved.
  - If the two bids are the same, node issues a new bid.
- Round trip delay can be eliminated by nodes issuing a burst of say 8 bids, and comparing these sequentially with received bids. First pair of differing bids resolves contention.

# Root contention example

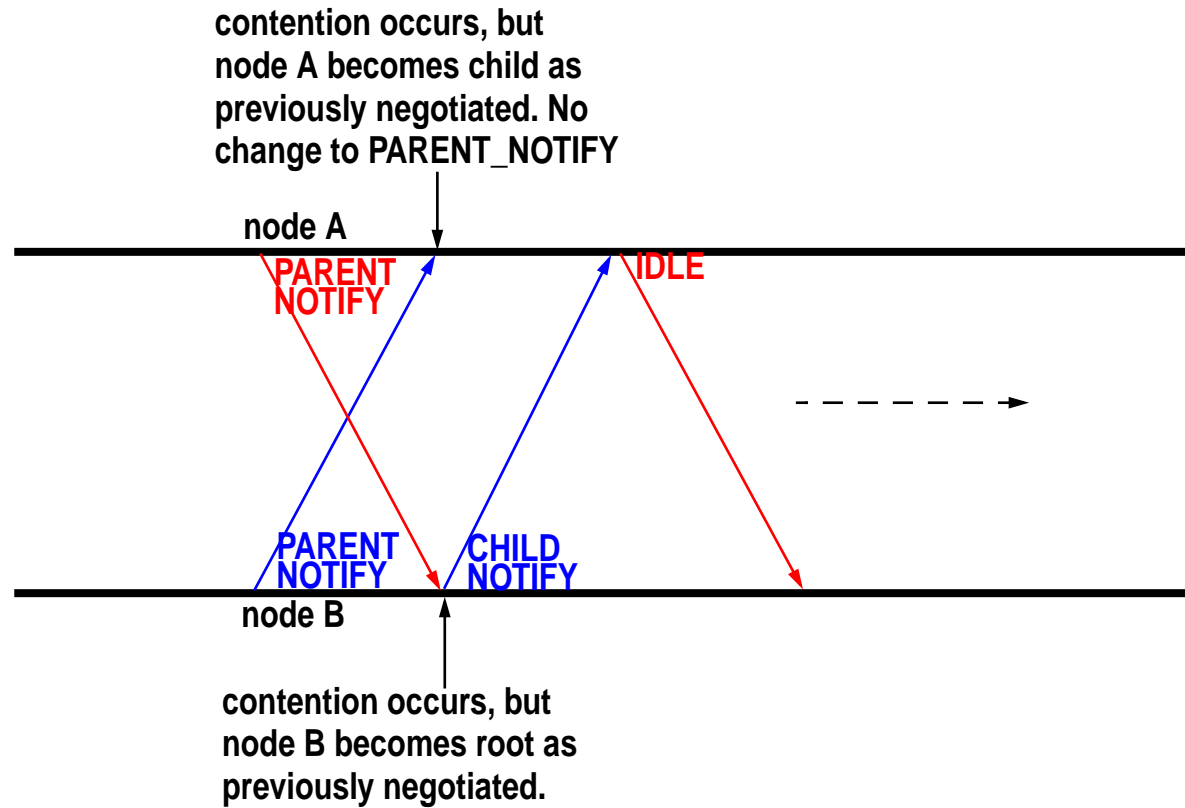


each bid is of form [ESCAPE, PARENT\_NOTIFY] or [ESCAPE, CHILD\_NOTIFY]

i	bid Ai	bid Bi	outcome
1	PARENT_NOTIFY	PARENT_NOTIFY	no result
2	CHILD_NOTIFY	CHILD_NOTIFY	no result
3	CHILD_NOTIFY	CHILD_NOTIFY	no result
4	PARENT_NOTIFY	CHILD_NOTIFY	A = child, B= root
5	CHILD_NOTIFY	PARENT_NOTIFY	don't care
6	PARENT_NOTIFY	CHILD_NOTIFY	don't care
7	PARENT_NOTIFY	PARENT_NOTIFY	don't care
8	CHILD_NOTIFY	PARENT_NOTIFY	don't care

- 1/256 chance of no result after 8 bids: if so, each node issues new bids and process repeats.

# Root contention example (cont.)



---

# Summary

---

- During start-up proposed patterns are:

S1 (ESL) = REQUEST = control state 0001

S2 (ESR) = IDLE = control state 0000

Easy codeword and scrambler synchronization.

- At end of S2 (ESR), transmit max. speed capability in form:

ESCAPE, SPEED+ .... ESCAPE, SPEED+ .....

Number of SPEED+ states indicates max. speed. Both nodes jump to maximum common speed capability.

- Alternative Root Contention resolution method described using prospective bidding during start-up.

Note: other recent proposals to solve long distance root contention also work with modified 8B10B coding.