RC23408 (W0411-032) November 3, 2004
Electrical Engineering

IBM Research Report

8B/10B Encoding and Decoding for High Speed Applications

Albert X. Widmer
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Oct. 21, 2004

8B/10B Encoding and Decoding for High Speed Applications

Albert X. Widmer

IBM T.J. Watson Research Center
1101 Kitchawan Rd. Route 134, Yorktown Heights, NY 10598-0218
914-945-2047, widmer@us.ibm.com

Summary

The 8B/10B Encoder and Decoder circuit designs presented here attempt to deliver the best possible speed and exploit the characteristics of the standard cell library for the IBM CMOS-7S or similar technology (Ref. 4) to the fullest. An implementation with both a single CoDec circuit and for parallel circuits is shown.

Introduction

The idea of serialized, self-timed chip I/O is gaining momentum. It is expected to replace many wide bidirectional and shared buses by separate high-speed, serial communications type lanes. The IBM *8B/10B transmission code of Ref. 2* is well entrenched in the industry for this kind of application because of its easy implementation and good transmission characteristics. A clear trend is also to higher transmission speed which traditionally has been handled by multiple byte CoDec circuits in parallel. Cideciyan (Ref. 3) teaches an improved method of operating multiple encoders in parallel.

The improvements described in this report go well beyond Ref. 3. *New vector classifications* have been developed for the mapping to and from the encoded domain which leads to coding and decoding equations with generally shorter delay. The code itself is unchanged. The disparity for both the coding and decoding process is handled by separate rather than shared (Ref. 2) classifications and the extra number of circuits required is surprisingly small. While traditional circuits present the running disparity at the end of a byte or block of bytes together with the encoded or decoded data, this design takes advantage of the fact that it is not necessary to immediately know the polarity of the running disparity at the start of the encoding or decoding and error checking processes. So instead of the ending disparity, the *starting disparity of the current byte together with certain characteristics of the current byte are passed along to the next byte interval* via latches or delay elements. The starting disparity of the next byte is then computed during the early part of the next cycle concurrently with bit mapping, error checking, and disparity classification of the new byte. These two innovations will allow many applications to operate with a single CoDec circuit rather than two or four, or two instead of four. For very short range applications, the reduced latency of a single CoDec may be more important than the circuit savings. Increased latency with multiple CoDec circuits is associated with single byte lanes because it takes extra time to assemble and disassemble several bytes in parallel if not needed for other reasons.

For implementations with *multiple* CoDec circuits operating in parallel, it is also not necessary to obtain the ending disparity of the respective multiple byte blocks, instead, the starting disparity of the current block may be derived from the disparity value at some preceding reference point and certain characteristics of the bytes up to the new reference point.

- For the *encoding* process, the running disparity from the preceding reference point is changed if the *number of unbalanced* coded vectors up to the new reference point is odd and remains unchanged otherwise. The same is true for the number of balanced vectors if the total number of vectors back to the reference is even. However, it seems that the ending running disparity of a block can be obtained without increasing the delay in the critical path.
- The *decoding* process does not appear to be addressed explicitly by Ref. 3. Disparity plays no role in the bit mapping for decoding, but the conformance with *disparity rules* is checked for error detection purposes and this logic path can be one of the bottlenecks in some of the traditional designs. The disparity check circuits classify the encoded bytes into those which are *disparity dependent or independent* (which is a different thing from being balanced or not). Disparity independent bytes are ignored. Disparity dependent bytes assume either a *positive or a negative exit disparity* depending solely on the bit patterns in the byte regardless of the running disparity at the start of the byte. For error checking purposes, *the running disparity at any point is equal to the exit disparity of the most recent disparity dependent byte.* For the example of four parallel bytes, this can be determined by 4 independent sets of combinational logic simultaneously, i.e. the disparity in front of byte #3 is obtained without knowledge of the disparity on front of the preceding bytes, as is required for some designs with longer delays. The disparity at the front of the 4-byte block is derived from the disparity value at the front of the last byte (#3) of the preceding block and the disparity characteristics of that byte. An improvement in the delay margin can be obtained by deferring this evaluation to the next block cycle. This solution is not taught by Ref. 3.

General Improvements for Higher Speed

The speed improvements are accomplished by the following steps:

1. New *coding and decoding tables* have been generated with new classifications. While the old classifications relied heavily on 4-variable functions, the new classifications rely more on 3-variable functions which can be implemented with less delay.
2. Separate functions have been defined which address specifically the *disparity* aspects. The *balance* of coded bytes is determined directly from the input bits rather than indirectly from the absence of positive or negative encoded 6B or 4B vectors. The original design for the control of the disparity and complementation relies on the same primitive classifications as are used for the support of bit encoding and decoding to save circuits.

3. For encoding, the disparity at the front of any vector is determined by a starting reference and any of the following four techniques:
   a. For a block of an even number of vectors, the disparity remains unchanged from the front end to the tail end, if there is an even number of balanced vectors, and the disparity at the tail end is the complement of the starting disparity if the number of balanced vectors is odd.
   b. For a block of an odd number of vectors, the disparity remains unchanged from the front end to the tail end, if there is an odd number of balanced vectors, and the disparity at the tail end is the complement of the starting disparity if the number of balanced vectors is even.
   c. Regardless of the number of bytes in a block, the disparity at the tail end is the complement of the disparity at the front end if the number of unbalanced vectors is odd, and the disparity remains unchanged if the number of unbalanced vectors is even. Reliance on unbalanced vectors generally requires more logic delay.
   d. An alternative approach depends on the propagation of the disparity from vector to vector and from byte to byte which requires the most time.
5. For decoding, the current disparity is assumed to be equal to the exit disparity of the next previous disparity dependent byte.
6. Rather than computing the ending disparity of a byte or block and then passing it on to the next block as a starting disparity, the starting disparity of the current byte or block may be computed based on the disparity at a reference point in the preceding block and the disparity characteristics of the bytes in between.
7. Particular attention has been devoted to the critical paths for the S-Function as defined in Note 2 of Table II and the complement functions. Some serial gating and large fan-in in the critical path has been avoided by gating upstream in multiple non-critical paths.
8. The first logic implementation choice throughout is NAND-gates with the least delay, second choice is NOR-gates. For the same reason, XNOR gates are preferred over XOR gates. Logic polarities are rigged to make the best use of the preferred circuits.
9. The more critical signals have been wired to the A-inputs which are the top inputs of the symbols, except for some XOR and XNOR gates for which the bottom inputs have less delay.
10. The current design has less sharing of basic logic elements and intermediate functions if it requires serial steps. For instance at the input, the A≠B function is generated directly from the A and B inputs instead of from intermediate logic expressions with multiple use.

Notation

The signal names used in the equations of this document do not reflect any logic levels, they are to be interpreted as abstract logic statements. However, in the circuit diagrams, the signal names may be prefixed with the letter P or N to indicate whether the function is true at the upper or lower level, respectively. The P and N prefixes are normally not used for net names which start with P and N, respectively, unless it could cause confusion and misinterpretations (e.g. NNDFS6=PPDFS6 is not reduced to avoid apparent contradictions such as NDFS6=PDFS6). Net numbers starting with n or m are true at the lower level and take the P prefix if true at the upper level.

8B/10B Encoder

The Table I, 5B/6BEncoding, has changed entries in the column ‘Coding Class’. The 5B vector sets L03 and L30 include all source vectors S5 with 3 logical zeros or ones, respectively, in the bit positions A, B, and C. The symbols L12 and L21 indicate 1 logical one and 2 logical zeros or vice versa, respectively, in the same bit positions. The new symbols are formally defined as follows:

1. The vector sets L03 and L30 comprise all S5 vectors which have 3 zeros or 3 ones, respectively, in the bits ABC.

   L03 = A’•B’•C’

   L30 = A•B•C

   The vector sets L12 and L21 comprise all S5 vectors which have 1 one and 2 zeros or 2 ones and 1 zero, respectively, in the bits ABC.

   L12 = A•B’•C’ + A’•B•C’ + A’•B’•C

   L21 = A’•B•C + A•B’•C + A•B•C’

Coder Equations, Table I and II.

Conceptually, coding is done in two steps. The translation to the primary vectors is executed first. A second step for the subset of the disparity dependent coded vectors determines whether the alternate, complemented vectors must be used to meet the disparity rules. Disparity dependent vectors have a plus sign or a minus sign in the DR column of the tables.

Generation of Primary Vectors

The logic equations necessary for the translation to the primary vectors can be read directly from the columns ‘Coding Class’, ‘Primary abcdei’, and ‘Primary fghj’ of Table I and Table II. In the ‘Primary’ columns, all plain bits are the same as the corresponding input bit values ABCDE. The bold and underlined bits are forced to the complemented value indicated. The i and j bits are assumed to be normally 0. If there are two classification entries on a single line separated by a comma, the second expression applies to the last bold 0 or 1 in the Primary column. The extracted logic equations from the Tables are listed below:

The symbols • and + represent the Boolean AND and OR functions, respectively. The apostrophe (’) represents negation.

a = A

b = B•(L30•D)’ + L03•D’

c = C + L03•D’ + L03•D•E = C + L03•(D’ + D•E) = C + L03•(D’ + E)

$d = D \bullet (L30 \bullet D)'$

$e = E \bullet (L03 \bullet D)' + L12 \bullet D' \bullet E' + L03 \bullet D \bullet E'$

$i = L21 \bullet D' \bullet E' + L12 \bullet [(D \neq E) + K] + L03 \bullet D' \bullet E + L30 \bullet D \bullet E$

$f = F \bullet [F \bullet G \bullet H \bullet (S+K)]'$

$g = G + F' \bullet G' \bullet H' = G + F' \bullet H'$

$h = H$

$j = (F \neq G) \bullet H' + F \bullet G \bullet H \bullet (S + K)$

8B/10B Disparity Control

The column *'DR Class'* (DR = Required Running Disparity) in the tables above classifies the vectors according to the plus sign and the minus sign entries in the DR column which indicates the required disparity at the front of the primary encoded vector. The expressions PDRS6, PDRS4 and NDRS6, NDRS4 represent a positive or negative required disparity, respectively, at the start of the 6B or 4B vectors.

PDRS6

The set of 6B vectors with a plus sign in the DR column is referred to as PDRS6.

$PDRS6 = L03 \bullet (D + E') + L30 \bullet D \bullet E' + L12 \bullet D' \bullet E'$

NDRS6

The set of 6B vectors with a minus sign in the DR column is referred to as NDRS6.

$NDRS6 = L30 \bullet (D' + E) + L03 \bullet D' \bullet E + L21 \bullet D \bullet E + K$

PDRS4

$PDRS4 = F' \bullet G' + (F \neq G) \bullet K$

NDRS4

$NDRS4 = F \bullet G$

In the circuit diagrams, the signal names PDFS6, PDFS4 and NDFS6 and NDFS4 represent the actual running disparity at the front of the 6B and 4B vectors, respectively.

CMPLS6 and CMPLS4

If the polarities of DF and DR do not match, a complement signal is generated which selects the alternate vector.

$CMPLS6 = NDFS6 \cdot PDRS6 + PDFS6 \cdot NDRS6$

$CMPLS4 = NDFS4 \cdot PDRS4 + PDFS4 \cdot NDRS4$

BALS6 and BALS4

The column *'DB Class'* (Block Disparity) identifies all coded vectors which are balanced by a 0 in the DB column.

The set of 6B vectors with a zero in the DB column is referred to as BALS6.

BALS6 = L21•(D' + E') + L12•K'•(D + E) + L30•D'•E'

The set of 4B vectors with a zero in the DB column is referred to as BALS4.

BALS4 = (F ≠ G) + F•G•H'

An encoded byte is balanced if BALS6 and BALS4 are either both true of both false.

BALBY = (BALS6=BALS4)

Table I. 5B/6B Encoding

Name	ABCDE K	Coding Class	Primary a b c d e i	Alternate a b c d e i	DR Class	DR	DB Class	DB
D0	0 0 0 0 0 0	L03•D’	0 $\underline{1}$ $\underline{1}$ 0 0 0	1 0 0 1 1 1	L03•(D+E’)	+		
D1	1 0 0 0 0 0	L12•D’•E’	1 0 0 0 $\underline{1}$ 0	0 1 1 1 0 1	L12•D’•E’	+		
D2	0 1 0 0 0 0	L12•D’•E’	0 1 0 0 $\underline{1}$ 0	1 0 1 1 0 1	L12•D’•E’	+		
D3	1 1 0 0 0 0	L21•D’•E’	1 1 0 0 0 $\underline{1}$			±	L21•(D’+E’)	0
D4	0 0 1 0 0 0	L12•D’•E’	0 0 1 0 $\underline{1}$ 0	1 1 0 1 0 1	L12•D’•E’	+		
D5	1 0 1 0 0 0	L21•D’•E’	1 0 1 0 0 $\underline{1}$			±	L21•(D’+E’)	0
D6	0 1 1 0 0 0	L21•D’•E’	0 1 1 0 0 $\underline{1}$			±	L21•(D’+E’)	0
D7	1 1 1 0 0 0		1 1 1 0 0 0	0 0 0 1 1 1	L30•(D’+E)	–	L30•D’•E’	0
D8	0 0 0 1 0 0	L03•D•E’	0 0 0 1 $\underline{1}$ 0	1 1 1 0 0 1	L03•(D+E’)	+		
D9	1 0 0 1 0 0	L12•(D≠E)	1 0 0 1 0 $\underline{1}$			±	L12•(D+E)•K’	0
D10	0 1 0 1 0 0	L12•(D≠E)	0 1 0 1 0 $\underline{1}$			±	L12•(D+E)•K’	0
D11[1]	1 1 0 1 0 0	L21•D•E’	1 1 0 1 0 0			±	L21•(D’+E’)	0
D12	0 0 1 1 0 0	L12•(D≠E)	0 0 1 1 0 $\underline{1}$			±	L12•(D+E)•K’	0
D13[1]	1 0 1 1 0 0	L21•D•E’	1 0 1 1 0 0			±	L21•(D’+E’)	0
D14[1]	0 1 1 1 0 0	L21•D•E’	0 1 1 1 0 0			±	L21•(D’+E’)	0
D15	1 1 1 1 0 0	L30•D	1 $\underline{0}$ 1 $\underline{0}$ 0 0	0 1 0 1 1 1	L30 • D•E’	+		
D16	0 0 0 0 1 0	L03•D’, L03•D’•E	0 $\underline{1}$ $\underline{1}$ 0 1 $\underline{1}$	1 0 0 1 0 0	L03•D’•E	–		
D17[2]	1 0 0 0 1 0	L12•(D≠E)	1 0 0 0 1 $\underline{1}$			±	L12•(D+E)•K’	0
D18[2]	0 1 0 0 1 0	L12•(D≠E)	0 1 0 0 1 $\underline{1}$			±	L12•(D+E)•K’	0
D19	1 1 0 0 1 0		1 1 0 0 1 0			±	L21•(D’+E’)	0
D20[2]	0 0 1 0 1 0	L12•(D≠E)	0 0 1 0 1 $\underline{1}$			±	L12•(D+E)•K’	0
D21	1 0 1 0 1 0		1 0 1 0 1 0			±	L21•(D’+E’)	0
D22	0 1 1 0 1 0		0 1 1 0 1 0			±	L21•(D’+E’)	0
D/K23	1 1 1 0 1 x		1 1 1 0 1 0	0 0 0 1 0 1	L30•(D’+E)	–		
D24	0 0 0 1 1 0	L03•D•E, L03•D	0 0 $\underline{1}$ 1 $\underline{0}$ 0	1 1 0 0 1 1	L03•(D+E’)	+		
D25	1 0 0 1 1 0		1 0 0 1 1 0			±	L12•(D+E)•K’	0
D26	0 1 0 1 1 0		0 1 0 1 1 0			±	L12•(D+E)•K’	0
D/K27	1 1 0 1 1 x		1 1 0 1 1 0	0 0 1 0 0 1	L21•D•E	–		
D28	0 0 1 1 1 0		0 0 1 1 1 0			±	L12•(D+E)•K’	0
K28	0 0 1 1 1 1	L12•K	0 0 1 1 1 $\underline{1}$	1 1 0 0 0 0	K	–		
D/K29	1 0 1 1 1 x		1 0 1 1 1 0	0 1 0 0 0 1	L21•D•E	–		
D/K30	0 1 1 1 1 x		0 1 1 1 1 0	1 0 0 0 0 1	L21•D•E	–		
D31	1 1 1 1 1 0	L30•D, L30•D•E	1 $\underline{0}$ 1 $\underline{0}$ 1 $\underline{1}$	0 1 0 1 0 0	L30•(D’+E)	–		

1. S = 1 for L21•PDFS6•D•E’

2. S = 1 for L12•NDFS6•D’•E

Table II. 3B/4B Encoding

Name	F G H K M	Coding Class	Primary f g h j	Alternate f g h j	DR Class	DR	DB Class	DB
D/Kx.0[1]	0 0 0 x 0	F'•G'•H'	0 $\underline{1}$ 0 0	1 0 1 1	F'•G'	+		
Dx.1	1 0 0 0 0	(F≠G)•H'	1 0 0 $\underline{1}$			±	(F≠G)	0
K28.1	1 0 0 1 0	(F≠G)•H'	1 0 0 $\underline{1}$	0 1 1 0	(F≠G)•K	+	(F≠G)	0
Dx.2	0 1 0 0 0	(F≠G)•H'	0 1 0 $\underline{1}$			±	(F≠G)	0
K28.2	0 1 0 1 0	(F≠G)•H'	0 1 0 $\underline{1}$	1 0 1 0	(F≠G)•K	+	(F≠G)	0
D/Kx.3[1]	1 1 0 x 0		1 1 0 0	0 0 1 1	F•G	–	F•G•H'	0
D/Kx.4[1]	0 0 1 x 0		0 0 1 0	1 1 0 1	F'•G'	+		
Dx.5	1 0 1 0 0		1 0 1 0			±	(F≠G)	0
K28.5	1 0 1 1 0		1 0 1 0	0 1 0 1	(F≠G)•K	+	(F≠G)	0
Dx.6	0 1 1 0 0		0 1 1 0			±	(F≠G)	0
K28.6	0 1 1 1 0		0 1 1 0	1 0 0 1	(F≠G)•K	+	(F≠G)	0
Dx.P7	1 1 1 0 0		1 1 1 0	0 0 0 1	F•G	–		
Dx.A7[2]	1 1 1 0 0	F•G•H•S	$\underline{0}$ 1 1 $\underline{1}$	1 0 0 0	F•G	–		
Ky.A7[3]	1 1 1 1 0	F•G•H•K	$\underline{0}$ 1 1 $\underline{1}$	1 0 0 0	F•G	–		

1. Kx is restricted to K28

2. S = L21•D•E'•PDFS6 + L12•D'•E•NDFS6

3. Ky is restricted to K23, K27, K28, K29, K30

Circuit Implementation of 8B/10B Encoding

A circuit block identified by eXGS_s10encode for 8B/10B encoding is illustrated in FIGS. 1A and 1B which are part of a single circuit with shared net names, i.e. a net name in one diagram may also find use in the other without the definition of input and output pins.

Notation for net names in the encoding circuit diagrams: The letters 'a' and 'o' within net-names refer to the Boolean AND and OR functions, respectively. The letter 'n' within a name negates the preceding parameter. The letters 'e' and 'ue' represent the symbols '=' and '≠', respectively. The capital letters ABCDEFGHK represent the uncoded input bits and the lower case letters abcdeifghj represent the coded format.

Bit Encoding

A bit encoding implementation according to the Tables I and II and the related equations and design principles is illustrated in the circuit diagram 8B/10B Bit Encoding of FIG. 1A.

Disparity Control

The circuit for the control of the disparity is shown in FIG. 1B. The output signal PBALBY assumes the upper level for a balanced byte.

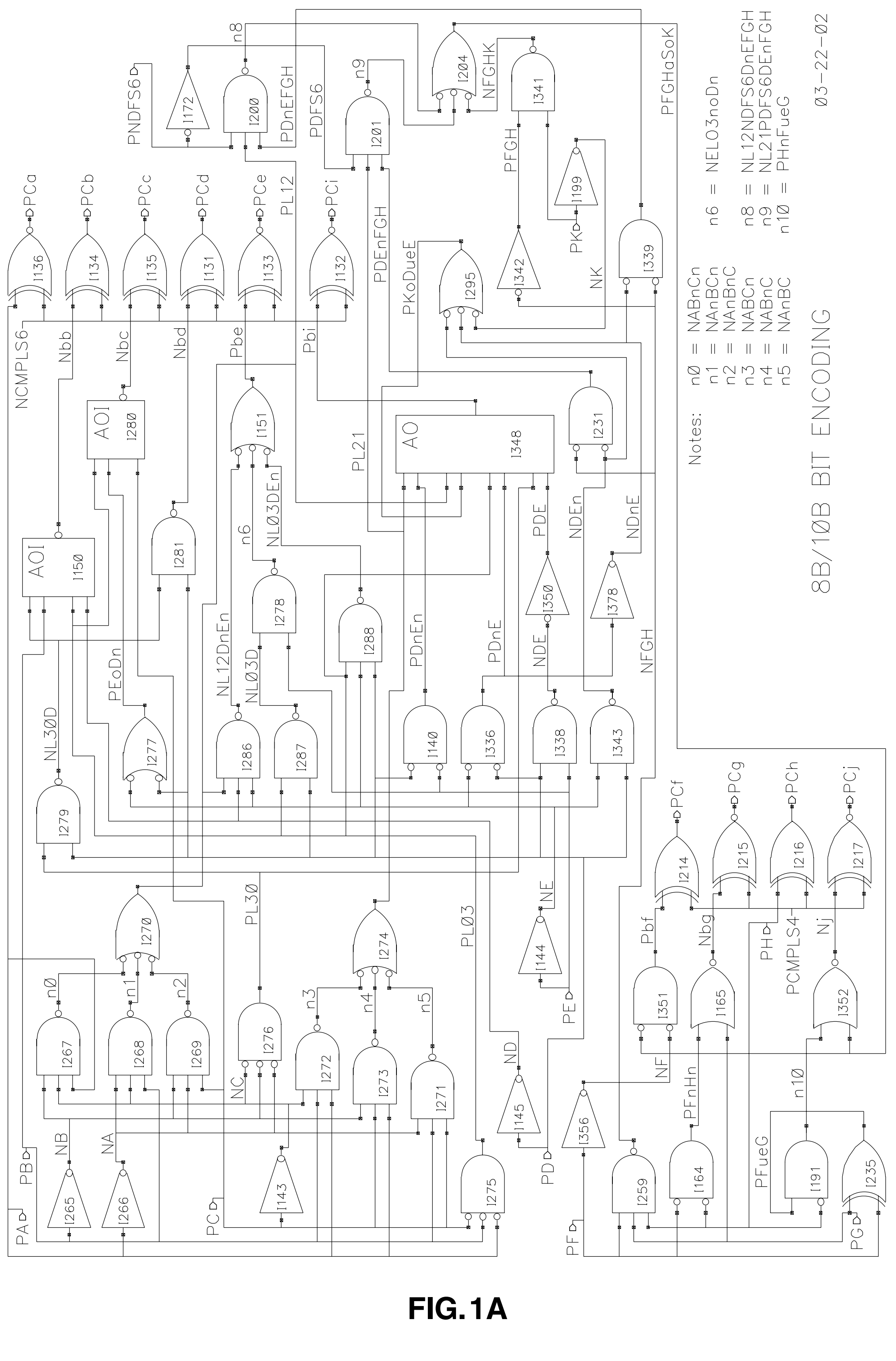

8B/10B BIT ENCODING
Notes:
n0 = NABnCn
n1 = NAnBCn
n2 = NAnBnC
n3 = NABCn
n4 = NABnC
n5 = NAnBC
n6 = NELO3noDn
n8 = NL12NDFS6DnEFGH
n9 = NL21PDFS6DEnFGH
n10 = PHnFueG
03-22-02
PA
PB
PC
PD
PE
PF
PG
PH
PK
PCa
PCb
PCc
PCd
PCe
PCi
PCf
PCg
PCh
PCj
PNDFS6
PDFS6
PDnEFGH
NFGHK
PFGHaSoK
PFGH
NK
PKoDueE
PDEnFGH
PL12
PL21
PL30
PL03
NCMPLS6
PCMPLS4
Nbb
Nbc
Nbd
Pbe
Pbi
Pbf
Nbg
Nj
AOI
AO
NL30D
PEoDn
NL12DnEn
NL03D
NL03DEn
PDnEn
PDnE
NDE
PDE
NDEn
NDnE
NFGH
NA
NB
NC
ND
NE
NF
PFnHn
PFueG

FIG. 1A

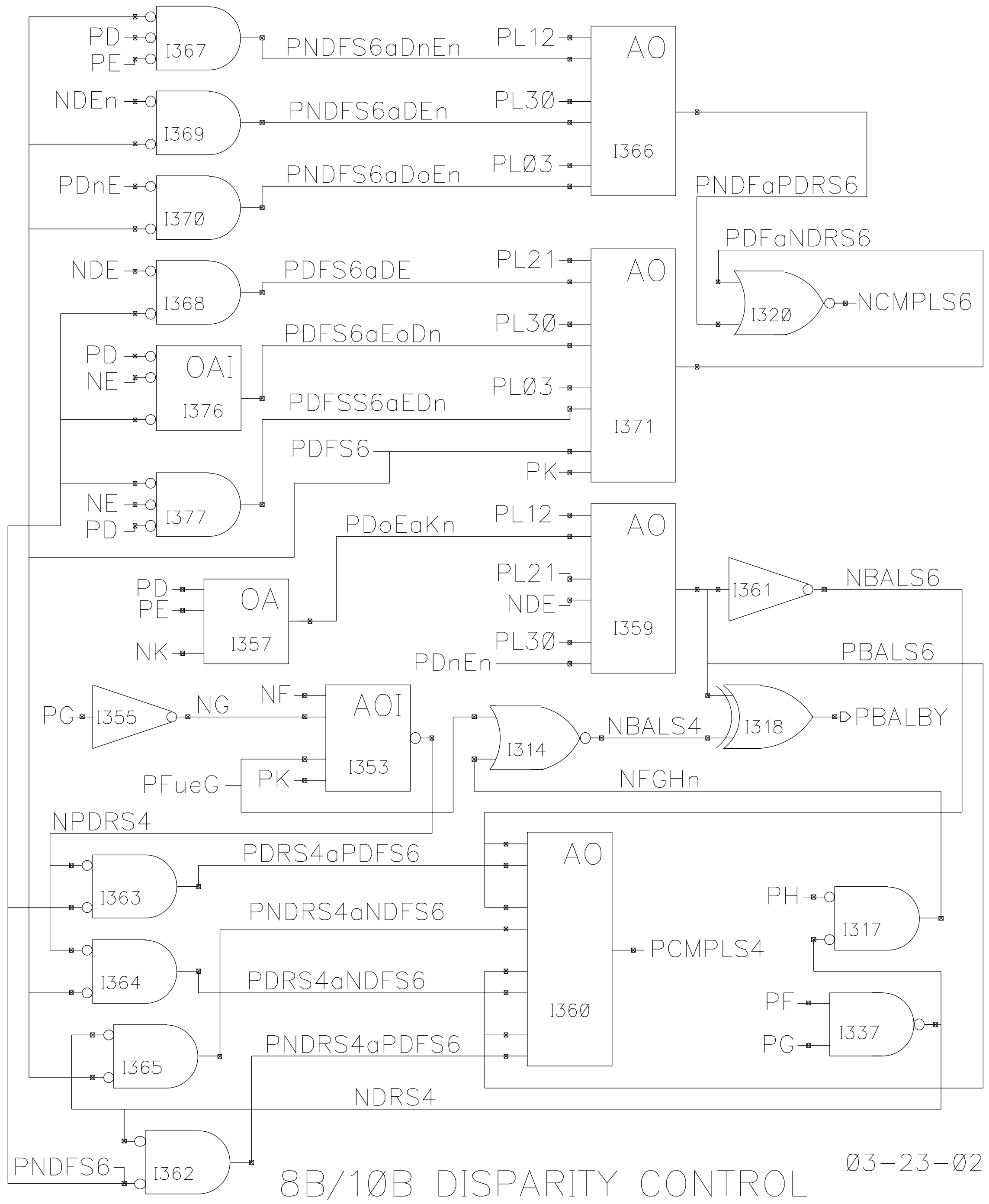


FIG.1B

A reduction in the combined delay of a 5B/6B and a 3B/4B encoder or of several 8B/10B encoders operating in parallel results from the methodology used to determine the disparity at any vector boundary as shown at the bottom of FIG.1B and in FIG.2. Given a starting disparity such as NDFBY (Negative Disparity in Front of a Byte), the running disparity at any subsequent vector boundary remains unchanged if the combined number of S6 and S4 vectors between the two points is even and the number of balanced vectors is even, otherwise it assumes the complementary polarity. This is in contrast to the more obvious techniques which observe the disparity as it propagates from vector to vector. The expression NDFS4 represents a negative running disparity in front of the 4B vector.

The 8B/10B encoder has an output which indicates whether the coded 10-bit byte is balanced or not, but there is no output to indicate the ending disparity. Generally, the starting disparity for a vector is determined from the disparity of a prior reference point according to the rules described above.

Disparity Circuit for 1-Byte Encoder, Fast Version

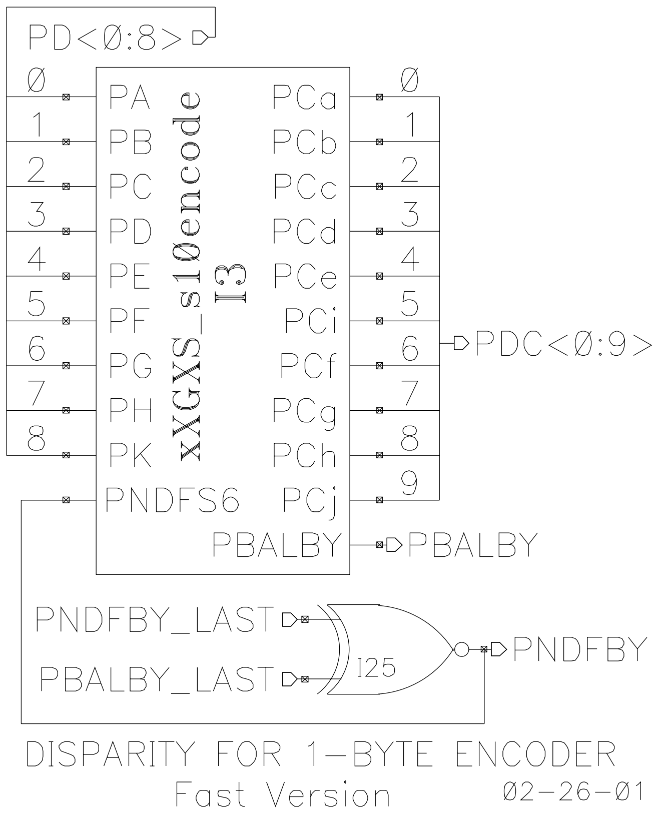


FIG.2

The circuit of FIG. 2 takes advantage of the fact that it is not necessary that the starting disparity must be known immediately for the encoding process. Since the evaluation of the running disparity at the end of a byte may be in the critical delay path, the final operations for determining the starting disparity PNDFS6 of the next byte are deferred to the next byte interval to be executed while initial bit encoding independent of the running disparity is performed. The cost of doing this is to pass along two parameters rather than just one to the next byte interval for an increase in the timing margin by an amount equal to the delay of a XNOR2 gate.

The circuit diagram of FIG. 2 provides the starting disparity PNDFBY and an encoded byte disparity 'PBALBY' for the current byte based on these identical two parameters carried over from the preceding byte. At the end of each byte cycle, the signals PNDFBY and PBALBY are stored in two latches with outputs PNDFBY_LAST and PBALBY_LAST, respectively. The respective latches are not shown since their timing is identical to or closely related to the timing for the data output latches. These parameters are used for the determination of the starting disparity of the next byte. The signal PNDFBY is at the upper level for a negative running disparity in front of the new byte.

Disparity Circuit for 1-Byte Encoder, Slower Version

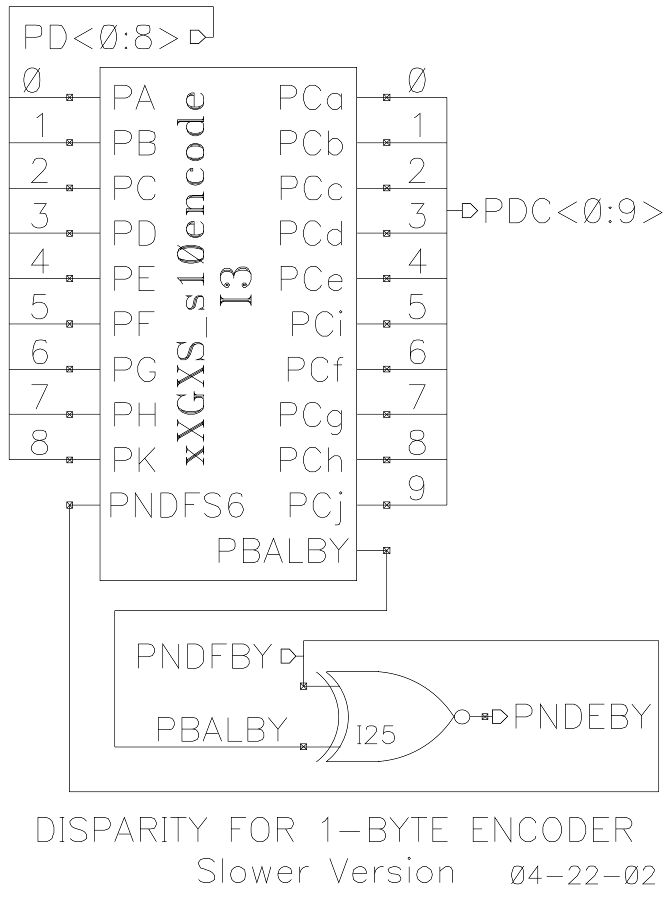


FIG. 3

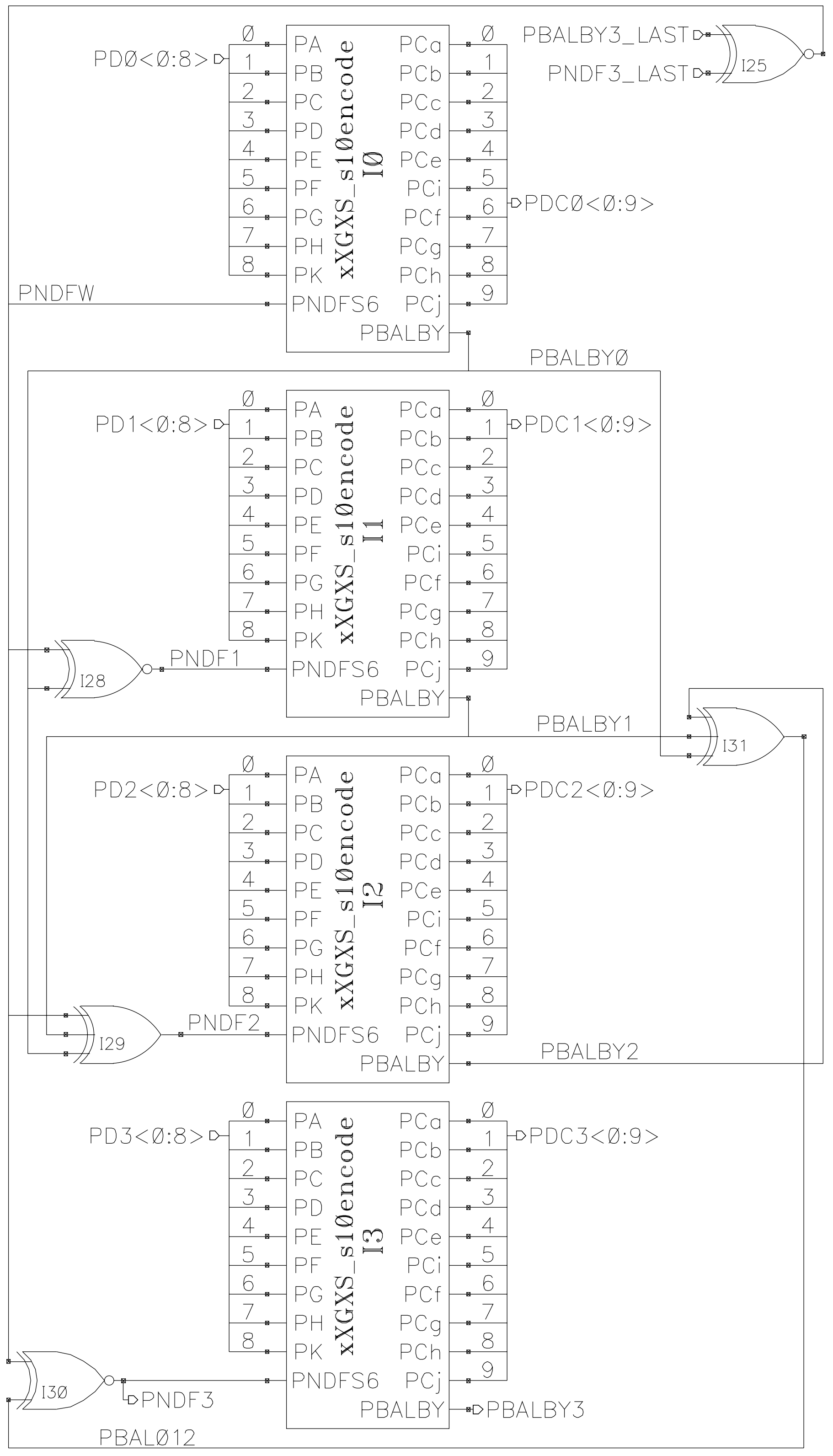
To better illustrate the fast approach to disparity operations, the more traditional way is also shown in FIG. 3 and is applicable where the higher performance is not needed. In traditional circuits, the ending disparity PNDEBY is derived within one and the same encoding cycle. Then only one parameter must be passed on to the next cycle with a single latch. The data input of this latch is PNDEBY and the output is PNDFBY, the disparity at the front of the next byte.

Disparity Circuit for 4-Byte Encoder, Fast Version

The circuit block of FIG. 4 shows four encoders operating in parallel on a 4-byte word. Again, this circuit takes advantage of the fact that it is not necessary that the starting disparity be known immediately for the encoding process. For this application, the circuit illustrated in FIG. 1B may be modified as follows: the gating by the signals PNDFS6 and PDFS6 may be moved forward in the logic chain. As an example, it might be advantageous, depending on the delay characteristics of the gates and interconnections, to move the respective gating to the AO gates I366 and I371 or even to the I320 gate, which would then be changed to and AOI gate.

The starting disparity of the block and of the first byte encoder I0 is given by the signal PNDFW (Neg. Disp. in Front of the Word) which is generated by the XNOR gate I25 at the top right. The starting disparity for each of the remaining 3 bytes is obtained by circuits operating in parallel from this reference point and the number of balanced bytes in between using a set of XOR and XNOR gates. Some designers have implemented daisy chains for disparity propagation from byte to byte with the associated serious delay penalties.

The signal PBAL012 (bottom left) is at the upper level if the block comprising the first 3 bytes I0, I1, and I2 is balanced. The running disparity PNDF3 in front of the last byte I3 and the balance signal PBALBY3 of the last byte are passed along to the next word cycle via a pair of latches, the output of which are labelled PNDF3_LAST and PBALBY3_LAST.



DISPARITY FOR 4-BYTE ENCODER, Fast Version 04-11-02

FIG.4

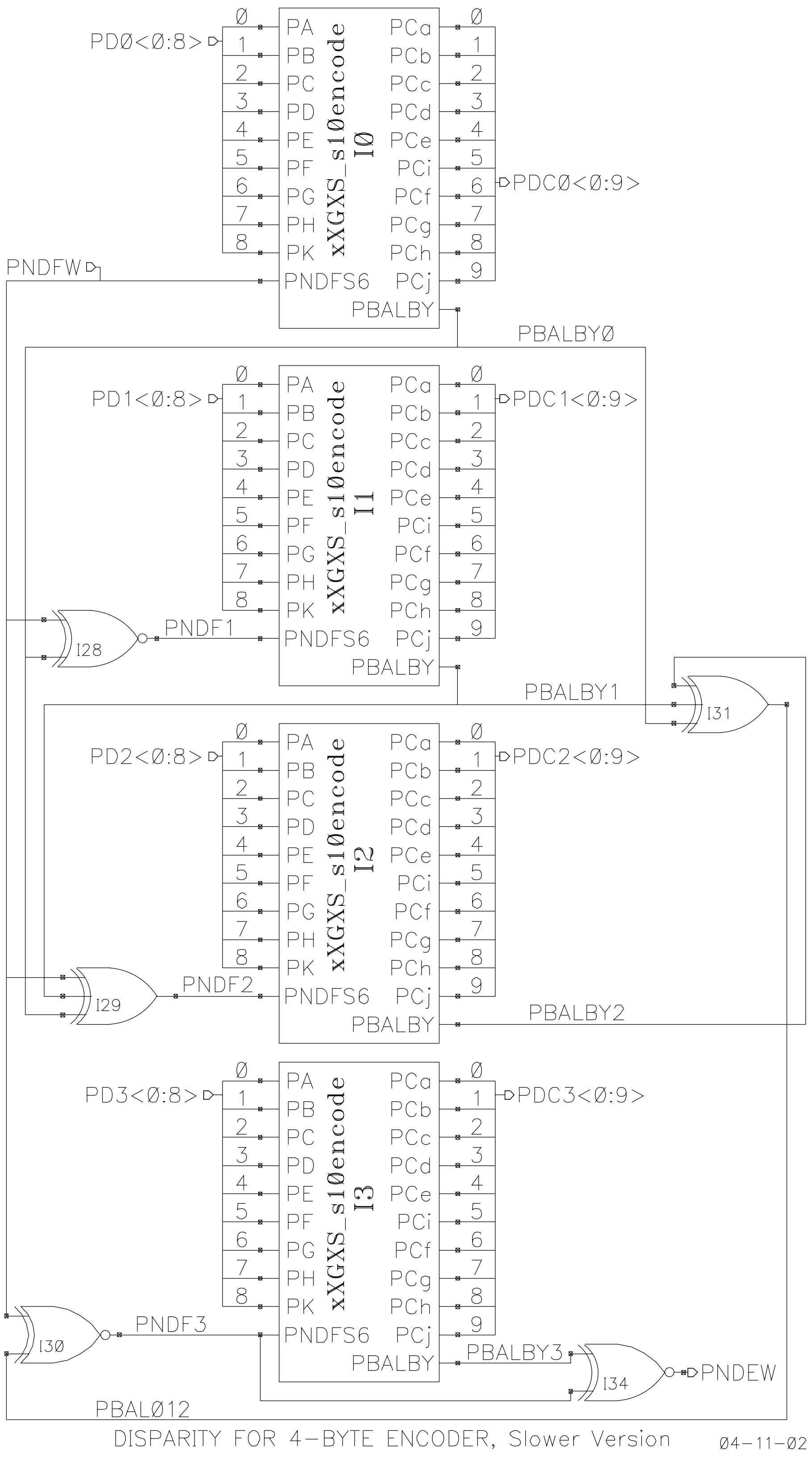


FIG.5

Disparity Circuit for 4-Byte Encoder, Slower Version

The circuit of FIG.5 shows also four encoders operating in parallel on a 4-byte word. But here the ending disparity PNDEW of the 4-byte block is generated in the same block cycle by the XNOR gate I34 at the lower right. The signal PNDEW is passed along to the next cycle via a latch the output of which is labelled PNDFW, the running disparity at the front of the next block.

10B/8B Decoder

The decoder consists of circuits to restore the original byte ABCDEFGH K, and circuits to indicate all transmission errors to the extent that they are detectable by the transmission code. The rules for the 6B/5B decoding are listed in Table III and the rules for 4B/3B decoding are listed in Table IV. Some changes in classifications have been made to enable faster operation at the expense of a few extra circuits.

The necessary bit translations for the decoding of a byte have been extracted from Tables III and IV and are listed in condensed form in Table V. The value of the decoded bits 'ABCDE FGH' is equal to the coded bits 'abcde fgh', respectively, except when the coded bits belong to one of the listed classifications of Table V and there is an 'F' for false in the respective bit column for that class.

The logic implementation of the decoder described in Ref 1 and 2 is an attempt to minimize the required circuit area which is accomplished by a reuse of basic logic definitions for various purposes such as decoding and error checking. That approach adds circuit delay and is not optimal for very high speed applications. The high speed implementation described here uses separate classifications for bit decoding, disparity classifications DR and DU, and each of these sets is implemented for minimum delay in CMOS technology.

Some of the 6B vector sets have been defined more broadly in the current design to allow invalid vectors in the set which leads to simpler logic terms with less delay. The new definitions operate identically for valid inputs, but for invalid inputs there may be different results which must be taken into account in the logic modelling and verification. These more inclusive categories also reduce the delay in the critical path of the disparity checks because the invalid vectors do not have to be added explicitly. There is a double win in using more comprehensive classifications: Simpler implementation and fewer logic terms.

The vector sets P3x and Px3 comprise all 4-bit vectors 'abcd' with 3 or 4 ones and 3 or 4 zeros, respectively, and replace the old sets of P13, P31. The sets P22, P04, and P40 are used only in the error checking circuits where P22 is derived from Px3 and P3x.

$P40 = a \bullet b \bullet c \bullet d$

$P04 = a' \bullet b' \bullet c' \bullet d'$

$P3x = P31 + P40 = a \bullet b \bullet c + a \bullet b \bullet d + a \bullet c \bullet d + b \bullet c \bullet d$

$Px3 = P13 + P04 = a' \bullet b' \bullet c' + a' \bullet b' \bullet d' + a' \bullet c' \bullet d' + b' \bullet c' \bullet d'$

$P22 = P3x' \bullet Px3'$

The vector sets P2x and Px2 are used to define the disparity classes and comprise all 3-bit vectors 'abc' with 2 or 3 ones and 2 or 3 zeros, respectively:

$P2x = P21 + P30 = a \bullet b + a \bullet c + b \bullet c$

$Px2 = P12 + P03 = a' \bullet b' + a' \bullet c' + b' \bullet c'$

Table III, 6B/5B Decoding and Table IV, 4B/3B Decoding, have been modified to reflect these changes. Table V, Decoder Equations, is based on Tables III and IV.

Table III. 6B/5B Decoding

Name	abcdei	Decoding Class	ABCDE K[1]	DR Class	DR	DU Class	DU
D0	011000	a’•b•c•d’•(e=i)	0**00**00 0	(a•b•c)’•d’•e’•i’	+	d’•e’•i’	–
D0	100111	a•b’•c’•d•(e=i)	**0**00**00** 0	(a’•b’•c’)’•d•e•i	–	d•e•i	+
D1	100010	Px3•i’	1000**0** 0	Px3•(e’+i’)	+	Px3•(e’+i’)	–
D1	011101	P3x•i	**1000**0 0	P3x•(e+i)	–	P3x•(e+i)	+
D2	010010	Px3•i’	0100**0** 0	Px3•(e’+i’)	+	Px3•(e’+i’)	–
D2	101101	P3x•i	**0100**0 0	P3x•(e+i)	–	P3x•(e+i)	+
D3	110001		11000 0		±		
D4	001010	Px3•i’	0010**0** 0	Px3•(e’+i’)	+	Px3•(e’+i’)	–
D4	110101	P3x•i	**0010**0 0	P3x•(e+i)	–	P3x•(e+i)	+
D5	101001		10100 0		±		
D6	011001		01100 0		±		
D7	111000		11100 0	a•b•c	–	d’•e’•i’	–
D7	000111	Px3•d•i	**11100** 0	a’•b’•c’	+	d•e•i	+
D8	000110	Px3•i’	0001**0** 0	Px3•(e’+i’)	+	Px3•(e’+i’)	–
D8	111001	P3x•i	**0001**0 0	P3x•(e+i)	–	P3x•(e+i)	+
D9	100101		10010 0		±		
D10	010101		01010 0		±		
D11	110100		11010 0		±		
D12	001101		00110 0		±		
D13	101100		10110 0		±		
D14	011100		01110 0		±		
D15	101000	a•b’•c•d’•(e=i)	1**1**1**1**0 0	(a•b•c)’•d’•e’•i’	+	d’•e’•i’	–
D15	010111	a’•b•c’•d•(e=i)	**1**1**1**1**0** 0	(a’•b’•c’)’•d•e•i	–	d•e•i	+
D16	011011	a’•b•c•d’•(e=i)	0**00**01 0	P2x•e•i	–	P2x•e•i	+
D16	100100	a•b’•c’•d•(e=i)	**0**00**01** 0	Px2•e’•i’	+	Px2•e’•i’	–
D17	100011		10001 0		±		
D18	010011		01001 0		±		
D19	110010		11001 0		±		
D20	001011		00101 0		±		
D21	101010		10101 0		±		
D22	011010		01101 0		±		
D/K23	111010		11101 x	P3x•(e+i)	–	P3x•(e+i)	+
D/K23	000101	Px3•e’	**11101** x	Px3•(e’+i’)	+	Px3•(e’+i’)	–
D24	001100	a’•b’•e’•i’	00**0**1**1** 0	Px2•e’•i’	+	Px2•e’•i’	–
D24	110011	a•b•e•i	**00**0**1**1 0	P2x•e•i	**–**	P2x•e•i	+
D25	100110		10011 0		±		
D26	010110		01011 0		±		
D/K27	110110		11011 x	P3x•(e+i)	–	P3x•(e+i)	+
D/K27	001001	Px3•e’	**11011** x	Px3•(e’+i’)	+	Px3•(e’+i’)	–
D28	001110		00111 0		±		
K28	001111	c•d•e•i	00111 **1**	(a’•b’•c’)’•d•e•i	–	d•e•i	+
K28	110000	c’•d’•e’•i’	**00111** **1**	(a•b•c)’•d’•e’•i’	+	d’•e’•i’	–
D/K29	101110		10111 x	P3x•(e+i)	–	P3x•(e+i)	+
D/K29	010001	Px3•e’	**10111** x	Px3•(e’+i’)	+	Px3•(e’+i’)	–
D/K30	011110		01111 x	P3x•(e+i)	–	P3x•(e+i)	+
D/K30	100001	Px3•e’	**01111** x	Px3•(e’+i’)	+	Px3•(e’+i’)	–
D31	101011	a•b’•c•d’•(e=i)	1**1**1**1**1 0	P2x•e•i	–	P2x•e•i	+
D31	010100	a’•b•c’•d•(e=i)	**1**1**1**1**1** 0	Px2•e’•i’	+	Px2•e’•i’	–

1. K = (K28 + Kx.7) = (c=d=e=i) + (e≠i) (i=g=h=j)

Table IV. 4B/3B Decoding

Name	fghj	Decoding Class	FGH K[1]	DR Class	DR	DU Class	DU
D/Kx.0	0100	(f≠g)•h'•j'	000 x	(f≠g)•h'•j'	+	h'•j'	–
D/Kx.0	1011	(f≠g)•h•j, f•g'•(h=j)	**000** x	(f≠g)•h•j	–	h•j	+
D/Kx.1	1001		100 x		±		
K28.1	0110	c'•d'•e'•i'•(h≠j)	**100 1**				
D/Kx.2	0101		010 x		±		
K28.2	1010	c'•d'•e'•i'•(h≠j)	**010 1**				
D/Kx.3	1100		110 x	f•g	–	h'•j'	–
D/Kx.3	0011	(f=g)•j	**110** x	f'•g'	+	h•j	+
D/Kx.4	0010		001 x	f'•g'	+	f'•g'•(h≠j)	–
D/Kx.4	1101	(f=g)•j	**001** x	f•g	–	f•g•(h≠j)	+
D/Kx.5	1010		101 x		±		
K28.5	0101	c'•d'•e'•i'•(h≠j)	**101 1**				
D/Kx.6	0110		011 x		±		
K28.6	1001	c'•d'•e'•i'•(h≠j)	**011 1**				
Dx.7	1110		111 0	f•g	–	f•g•(h≠j)	+
Dx.7	0001	(f=g)•j	**111** 0	f'•g'	+	f'•g'•(h≠j)	–
D/Kx.7	0111	(f≠g)•h•j	1**11** x	(f≠g)•h•j	–	h•j	+
D/Kx.7	1000	(f≠g)•h'•j', f•g'•(h=j)	1**11** x	(f≠g)•h'•j'	+	h'•j'	–

[1] K = (K28 + Kx.7) = (c=d=e=i) + (e≠i)•(i=g=h=j)

Table V. 10B/8B Decoder Equations

Classifications	A=a	B=b	C=c	D=d	E=e	F=f	G=g	H=h	K=0
a'•b•c•d'•(e=i)		F	F						
a•b'•c'•d•(e=i)	F			F	F				
Px3•i'					F				
P3x•i	F	F	F	F					
a•b'•c•d'•(e=i)		F		F					
a'•b•c'•d•(e=i)	F		F		F				
a'•b'•e'•i'			F		F				
a•b•e•i	F	F		F					
Px3•(d•i+e')+c'•d'•e'•i'	F	F	F	F	F				
(f≠g)•h•j						F			
(f≠g)•h'•j'							F		
f•g'•(h=j)								F	
(f=g)•j + c'•d'•e'•i'•(h≠j)						F	F	F	
K28[1] + Kx.7[2]									F

[1] K28 = c• d• e• i + c'• d'• e'• i'

[2] Kx.7 = (e≠i)• (i=g=h=j)

6B/5B Decoder and Error Check Equations

Logic Equations for the Generation of the decoded Bits A, B, C, D, E

Generally, A = a, B = b, C = c, D = d, E = e, except for the conditions listed in Table V, the complement of the respective unencoded bit is generated, e.g. A = a'. The following decoding equations are extracted from Table V:

1. A = a' iff (if and only if):

   $a\bullet b'\bullet c'\bullet d\bullet (e=i) + P3x\bullet i + a'\bullet b\bullet c'\bullet d\bullet (e=i) + a\bullet b\bullet e\bullet i + Px3\bullet (d\bullet i + e') + c'\bullet d'\bullet e'\bullet i'$

   Using Boolean manipulations, this logic statement can be restructured as follows:

   $(a\neq b)\bullet c'\bullet d\bullet (e=i) + P3x\bullet i + Px3\bullet (d\bullet i+e') + a\bullet b\bullet e\bullet i + c'\bullet d'\bullet e'\bullet i'$

   An examination of Table III shows that the first term in the above equation $[(a\neq b)\bullet c'\bullet d\bullet (e=i)]$ can be changed to $(a'+b')\bullet c'\bullet d\bullet (e=i)$. This modified expression overlaps with D7 ('1000111'b = $Px3\bullet d\bullet i$) which also requires complementation of the bit 'a'. So the term $Px3\bullet d\bullet i$ for the decoding of bit 'a=0' to 'A=1' could be eliminated and thus the delay in a non-critical path is reduced, but it has been included because it saves a gate. The equation implemented in the circuit diagram of FIG. 6A is:

   $(a'+b')\bullet c'\bullet d\bullet (e=i) + P3x\bullet i + Px3\bullet (d\bullet i+e') + a\bullet b\bullet e\bullet i + c'\bullet d'\bullet e'\bullet i'$

   In said logic diagram, the following abbreviations are used:

   $n0 = (a'+b')\bullet c'\bullet d\bullet (e=i) = c'\bullet d\bullet (e=i)\bullet n8$

   $n1 = Px3\bullet (d\bullet i+e')$

   $n2 = a\bullet b\bullet e\bullet i + c'\bullet d'\bullet e'\bullet i'$

   $n8 = (a'+b')$

   $CPLa = n0 + n1 + P3x\bullet i + n2$, where CPL suggests complementation

2. B = b' iff:

   $a'\bullet b\bullet c\bullet d'\bullet (e=i) + P3x\bullet i + a\bullet b'\bullet c\bullet d'\bullet (e=i) + a\bullet b\bullet e\bullet i + Px3\bullet (d\bullet i+e') + c'\bullet d'\bullet e'\bullet i'$

   Boolean reduction gives:

   $(a\neq b)\bullet c\bullet d'\bullet (e=i) + P3x\bullet i + Px3\bullet (d\bullet i+e') + a\bullet b\bullet e\bullet i + c'\bullet d'\bullet e'\bullet i'$

   By an examination of Table III, it can be verified that the first term $(a\neq b)\bullet c\bullet d'\bullet (e=i)$ can again be simplified to $(a+b)\bullet c\bullet d'\bullet (e=i)$. However, it turns out that because of signal polarity problems, $(a\neq b)$ can be generated with less delay than $(a+b)$, and so the former expression is used.

   In the logic diagram, the following abbreviations are used:

   $n3 = c\bullet d'\bullet (e=i)\bullet (a\neq b)$

   $CPLb = n3 + n1 + P3x\bullet i + n2$

3. C = c' iff:

   $a'\bullet b\bullet c\bullet d'\bullet (e=i)+P3x\bullet i+a'\bullet b\bullet c'\bullet d\bullet (e=i)+a'\bullet b'\bullet e'\bullet i'+Px3\bullet (d\bullet i+e')+c'\bullet d'\bullet e'\bullet i'$

   This logic statement can be restructured as follows:

   $a'\bullet b\bullet (c\neq d)\bullet (e=i) + P3x\bullet i + Px3\bullet (d\bullet i+e') + e'\bullet i'\bullet (a'\bullet b'+c'\bullet d')$

In the logic diagram, the following abbreviations are used:

$n4 = a' \bullet b \bullet (c \neq d) \bullet (e=i)$

$n5 = e' \bullet i' \bullet (a' \bullet b' + c' \bullet d')$

$CPLc = n4 + n1 + P3x \bullet i + n5$

4. D = d' iff:

   $a \bullet b' \bullet c' \bullet d \bullet (e=i) + P3x \bullet i + a \bullet b' \bullet c \bullet d' \bullet (e=i) + a \bullet b \bullet e \bullet i + Px3 \bullet (d \bullet i + e') + c' \bullet d' \bullet e' \bullet i'$

   This logic statement can be restructured as follows:

   $a \bullet b' \bullet (c \neq d) \bullet (e=i) + P3x \bullet i + a \bullet b \bullet e \bullet i + Px3 \bullet (d \bullet i + e') + c' \bullet d' \bullet e' \bullet i'$

   In the logic diagram, the following abbreviations are used:

   $n6 = a \bullet b' \bullet (c \neq d) \bullet (e=i)$

   $CPLd = n6 + n1 + P3x \bullet i + n2$

5. E = e' iff:

   $a \bullet b' \bullet c' \bullet d \bullet (e=i) + Px3 \bullet i' + a' \bullet b \bullet c' \bullet d \bullet (e=i) + a' \bullet b' \bullet e' \bullet i' + Px3 \bullet (d \bullet i + e') + c' \bullet d' \bullet e' \bullet i'$

   Boolean reduction reduces this to:

   $(a \neq b) \bullet c' \bullet d \bullet (e=i) + Px3 \bullet (d \bullet i + e' + i') + e' \bullet i' \bullet (a' \bullet b' + c' \bullet d')$

   An examination of Table III shows the same kind of overlap as described above for the decoding of bit 'a' to 'A'. In this case, a gate can be saved by taking advantage of the overlap and deleting the term $Px3 \bullet d \bullet i$. The equation can thus be reduced to:

   $(a' + b') \bullet c' \bullet d \bullet (e=i) + Px3 \bullet (e' + i') + e' \bullet i' \bullet (a' \bullet b' + c' \bullet d')$

   In the logic diagram, the following abbreviations are used:

   $n7 = Px3 \bullet (e' + i')$

   $CPLe = n0 + n7 + n5$

Logic Equation for invalid Vectors R6, INVR6

There are a total of 16 invalid R6 vectors:

$INVR6 = P40 + P04 + P3x \bullet e \bullet i + Px3 \bullet e' \bullet i'$

6B/5B Disparity Check Equations

The column 'DR' of Table III lists the required disparity at the start of the respective 6B vectors and the column 'DR Class' identifies the respective input bit patterns. The old column DB (Ref. 1) has been changed to DU and lists a positive or negative exit disparity for the disparity dependent vectors only. Disparity independent vectors have no entry in the DU column. In the old design, disparity independent vectors passed the input disparity to the output. In the new design, disparity independent vectors are ignored and bypassed for disparity purposes for shorter delay. Short delay is especially important for the DU outputs PDUR6 and NDUR6. To achieve this goal, a dedicated column 'DU Class' has been added which sorts the received vectors into DU classes in the most efficient way.

Logic Equations for Required Input Disparity DRR6

The terms PDRR6 and NDRR6 represent the R6 vectors which require a positive or negative running disparity, respectively, at the start of the vector. Contrary to previous implementations, all invalid vectors starting with P40 or P04 have been left out of the equations because of their redundancy in the overall error checking scheme.

$$PDRR6 = Px3 \bullet (e' + i') + a' \bullet b' \bullet c' + (a \bullet b \bullet c)' \bullet d' \bullet e' \bullet i' + Px2 \bullet e' \bullet i'$$

$$NDRR6 = P3x \bullet (e + i) + a \bullet b \bullet c + (a' \bullet b' \bullet c')' \bullet d \bullet e \bullet i + P2x \bullet e \bullet i$$

Boolean operations reduce this to:

$$PDRR6 = Px3 \bullet (e' + i') + a' \bullet b' \bullet c' + e' \bullet i' \bullet [Px2 + d' \bullet (a \bullet b \bullet c)']$$

$$NDRR6 = P3x \bullet (e + i) + a \bullet b \bullet c + e \bullet i \bullet [P2x + d \bullet (a' \bullet b' \bullet c')']$$

In the diagram of FIG. 6B, the following abbreviations are used for net names:

$$n20 = (a \bullet b \bullet c)' \bullet d'$$

$$n21 = (a' \bullet b' \bullet c')' \bullet d$$

$$n22 = PDRR4 \bullet NDRR6'$$

$$n23 = NDRR4 \bullet PDRR6'$$

Logic Equation for Monitoring Byte Disparity Violations

Bytes with only disparity independent vectors R6 and R4 are ignored for disparity checking purposes. There is a disparity violation DVBY at a specific byte under the following conditions:

1. The required entry disparity of the R6 vector does not match the running disparity at the front of the byte.
2. The R6 vector is disparity independent and the required entry disparity of the R4 vector does not match the running disparity in front of the byte.

For implementation with reduced delay in a critical path, the disparity dependence of the vector R6 is ignored, instead the entry disparity of vector R4 is compared with the disparity in front of the byte and an error is flagged on a mismatch except when the vector R6 has a required entry disparity which matches the running disparity.

A disparity violation internal to a byte from a disparity dependent R4 vector mismatched to a disparity dependent R6 vector is included in the set of invalid bytes, but not in DVBY. The disparity violation at a byte DVBY is thus given by the equation:

$$DVBY = NDFBY \bullet (PDRR6 + PDRR4 \bullet NDRR6') + PDFBY \bullet (NDRR6 + NDRR4 \bullet PDRR6')$$

The terms PDFBY and NDFBY represent a positive or negative running disparity, respectively, at the front of the byte.

If needed, one level of gating (I300 and I301, upper right corner of FIG. 6B) in the above equation can be eliminated by merging the gates I302 and I303 with the NDRR6 and

PDRR6 functions. The two AOI222 gates are then first duplicated, changed to the non-inverting version and expanded to AO2222 to generate PNDR and PDR directly.

Logic Equations for the assumed ending Disparities PDUR6 and NDUR6

The logic equations for the generation of the exit disparities PDUR6 and NDUR6, for invalid vectors PINVR6, and for the disparity violations are listed and explained on page 19 of the Research Report RC18855. Some simplifications have been made. For the expressions PDBR6 and NDBR6, the terms P40 and P04 have been replaced by P40•(e+i) and P04•(e’+i’), respectively. The terms P40 and P04 in the encoded domain can be generated only by at least one error. For the case of a single error and e=i, the R6 vector was obviously initially balanced and so should not generate PDUR6 or NDUR6 which would generate a superfluous code violation at the next disparity dependent vector in addition to the invalid vector at the actual error location. Therefore:

PDUR6 = P3x•(e+i) + d•e•i + P2x•e•i = P3x•(e+i) + e•i•(d + P2x)

NDUR6 = Px3 •(e’+i‘) + d’•e‘•i’ + Px2•e‘•i’ = Px3 •(e’+i‘) + e‘•i’•(d' + Px2)

4B/3B Decoder, Error Checks

Logic Equations for the Generation of the decoded Bits F, G, H, K

Generally, F = f, G = g, H = h, except for the conditions listed in Table V with an F (False) entry for which the complement of the respective unencoded bit is generated, e.g. H = h’.

In Table IV, the Decoding Class for the lines D/Kx.3 (0011), D/Kx.4 (1101), and Dx.7 (0001) have all been changed to (f=g)•j and similar changes have been made for other vector classifications. This leads to simplified equations on the last 5 rows of the Classifications column in Table V. With this change, the following decoding equations are extracted from Table V:

1. F = f’ iff (if and only if):

   (f≠g)•h•j + (f=g)•j + c’•d’•e’•i’•(h≠j)

   In the logic diagram, the following abbreviations are used:

   m0 = (f≠g)•h•j

   m7 = c’•d’•e’•i’•(h≠j)

   CPLf = m0 + (f=g)•j + m7
2. G = g’ iff:

   (f≠g)•h’•j’ + (f=g)•j + c’•d’•e’•i’•(h≠j)

   CPLg = (f≠g)•h’•j’ + (f=g)•j + m7
3. H = h’ iff:

   f•g’•(h=j) + (f=g)•j + c’•d’•e’•i’•(h≠j)

   Abbreviation:

   m2 = f•g’•(h=j)

$\mathrm{CPLh} = \mathrm{m2} + (f=g) \bullet j + \mathrm{m7}$

Logic Equation for Control Bit K

$K = (K28 + Kx.7) = (c=d=e=i) + (e \neq i) \bullet (i=g=h=j)$

For reduced delay, the above equation is implemented as follows:

$K = c \bullet d \bullet e \bullet i + c' \bullet d' \bullet e' \bullet i' + (e \neq i) \bullet (i \bullet g \bullet h \bullet j + i' \bullet g' \bullet h' \bullet j')$

In the logic diagram of FIG.7, the following abbreviations are used:

$m10 = i \bullet g \bullet h \bullet j + i' \bullet g' \bullet h' \bullet j'$

$Kx7 = m10 \bullet (e \neq i)$

Logic Equations for the required Disparity at the Front of the R4 Vector

The terms PDRR4 and NDRR4 represent the required positive or negative disparity, respectively, at the front of the R4 vector.

$PDRR4 = f' \bullet g' + (f \neq g) \bullet h' \bullet j'$

$NDRR4 = f \bullet g + (f \neq g) \bullet h \bullet j = f \bullet g + m0$

Logic Equations for the assumed ending Disparities PDUR4 and NDUR4

$PDUR4 = h \bullet j + f \bullet g \bullet (h \neq j)$

$NDUR4 = h' \bullet j' + f' \bullet g' \bullet (h \neq j)$

Logic Equation for invalid Vector R4, INVR4

There are a total of 2 inherently invalid R4 vectors: all ones or all zeros (f=g=h=j). Some invalid combinations of R6 and R4 vectors are also detected in this circuit complex and lumped together in the signal INVR4, such as violations of the S-function rules (e=i=f=g=h), and Kx.7 control characters with R6 vectors of the P22 class. (In previous designs, these unsuitable R6 vectors were classified as balanced vectors which leads to a more complicated circuit.) The K28•(f=g=h) character is a valid control character in terms of coding constraints, but is not included in the 8B/10B alphabet because it would require special encoding circuits. It is included in the group of invalid characters.

An invalid Kx.7 control character is recognized if the following conditions are met:

$VKx7 = (i \bullet g \bullet h \bullet j + i' \bullet g' \bullet h' \bullet j') \bullet (e \neq i) \bullet P22 = m10 \bullet (e \neq i) \bullet P22$

Other invalid R4 vectors are lumped together in the signal INVR4:

$INVR4 = (f=g=h=j) + (e=i=f=g=h) + K28 \bullet (f=g=h) + K28' \bullet (i \neq g=h=j)$

Abbreviations:

$m5 = K28 \bullet (f=g=h)$

$m6 = K28' \bullet (i \neq g=h=j)$

$INVR4 = (f=g=h=j) + (e=i=f=g=h) + m5 + m6$

Circuit Implementation of 10B/8B Decoding

Circuits for 6B/5B Decoding and 6B Error Checks

An example of a circuit implementation of the above 6B/5B decoding and error check equations is illustrated in FIGS. 6A and 6B, which represent a single circuit with net sharing which is identified by the block name eXGXS_r6cecode.

Notation for net names in the decoding circuit diagrams: For the Boolean operators, the identical letters are use as for the encoding diagrams, but they are capitalized (A, O, N, E, UE) to avoid confusion with some of the lower case letters abcdeifghj which represent the coded bits.

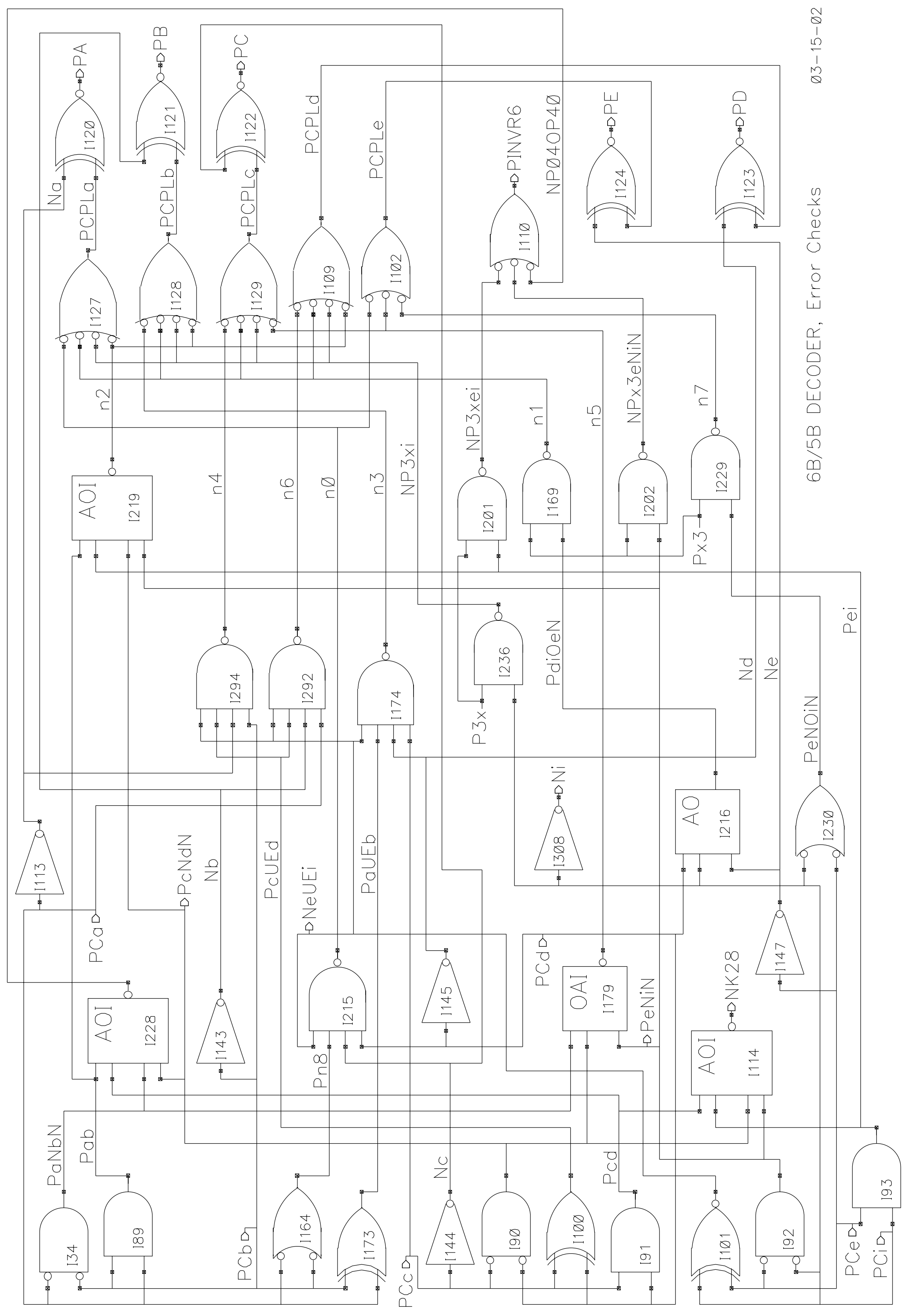

6B/5B DECODER, Error Checks
03-15-02
PA
PB
PC
PD
PE
PINVR6
NP04OP40
PCPLa
PCPLb
PCPLc
PCPLd
PCPLe
Na
I120
I121
I122
I123
I124
I127
I128
I129
I109
I102
I110
n2
n4
n6
n0
n3
NP3xi
NP3xei
n1
n5
NPx3eNiN
n7
AOI
I219
I201
I169
I202
I229
Px3
P3x
I236
PdiOeN
Nd
Ne
Pei
I294
I292
I174
PeNOiN
AO
I216
I230
Ni
I308
I113
PcNdN
Nb
PcUEd
NeUEi
PaUEb
PCa
PCd
AOI
I228
I143
I215
I145
OAI
I179
PeNiN
NK28
AOI
I114
I147
Pn8
PaNbN
Pab
Nc
Pcd
I34
I89
PCb
I164
I173
PCc
I144
I90
I100
I91
I101
I92
PCe
PCi
I93

FIG.6A

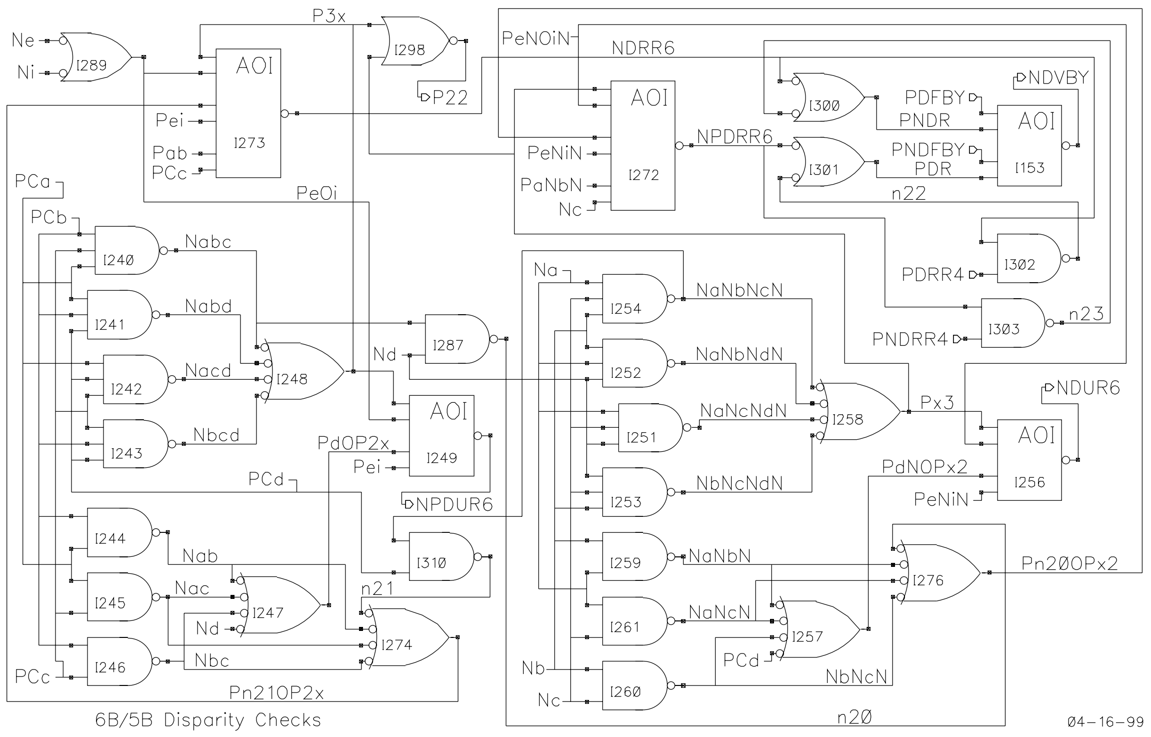


FIG.6B

Circuit Implementation of 4B/3B Decoder and related Error Checks

A circuit implementation of 4B/3B decoding equations and the related error checks is illustrated in the circuit diagram of FIG.7 and identified by the block name eXGXS_r4decode.

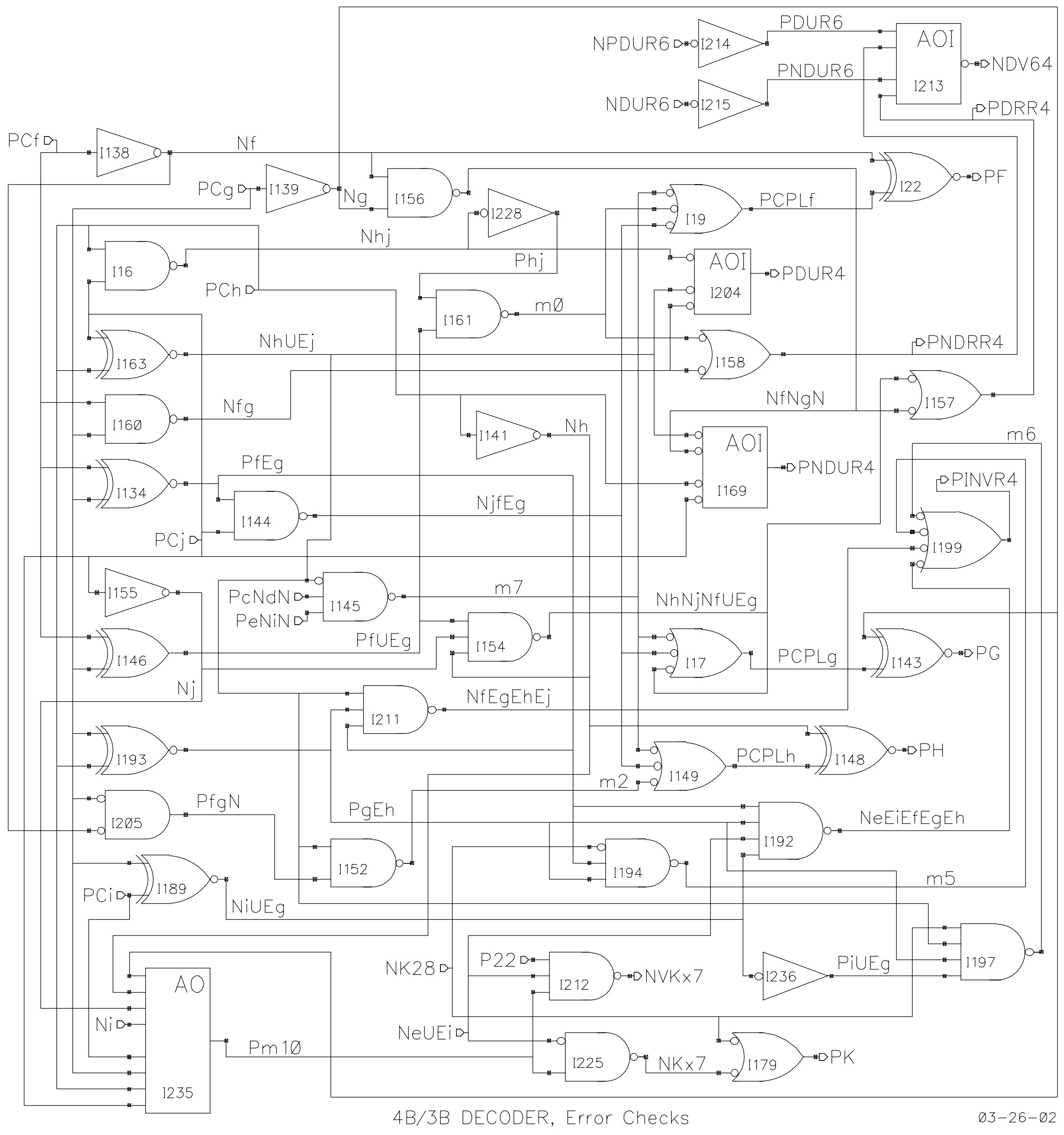


FIG. 7

Circuits for 10B/8B Byte-Level Error Checks

The circuit of FIG. 8 merges the 6B/5B and 4B/3B decoders into a byte decoder with the block name eXGXS_r10decode.

Error Reporting

The circuit of FIG. 8 generates the signal PINVBY which indicates an inherently invalid byte which includes disparity violations NDV64 which are evident from an examination of just the 10 coded bits of the current byte. The signal PVIOL signals either an invalid byte or a disparity violation NDVBY detected at this location which may result from an error in this or a preceding byte.

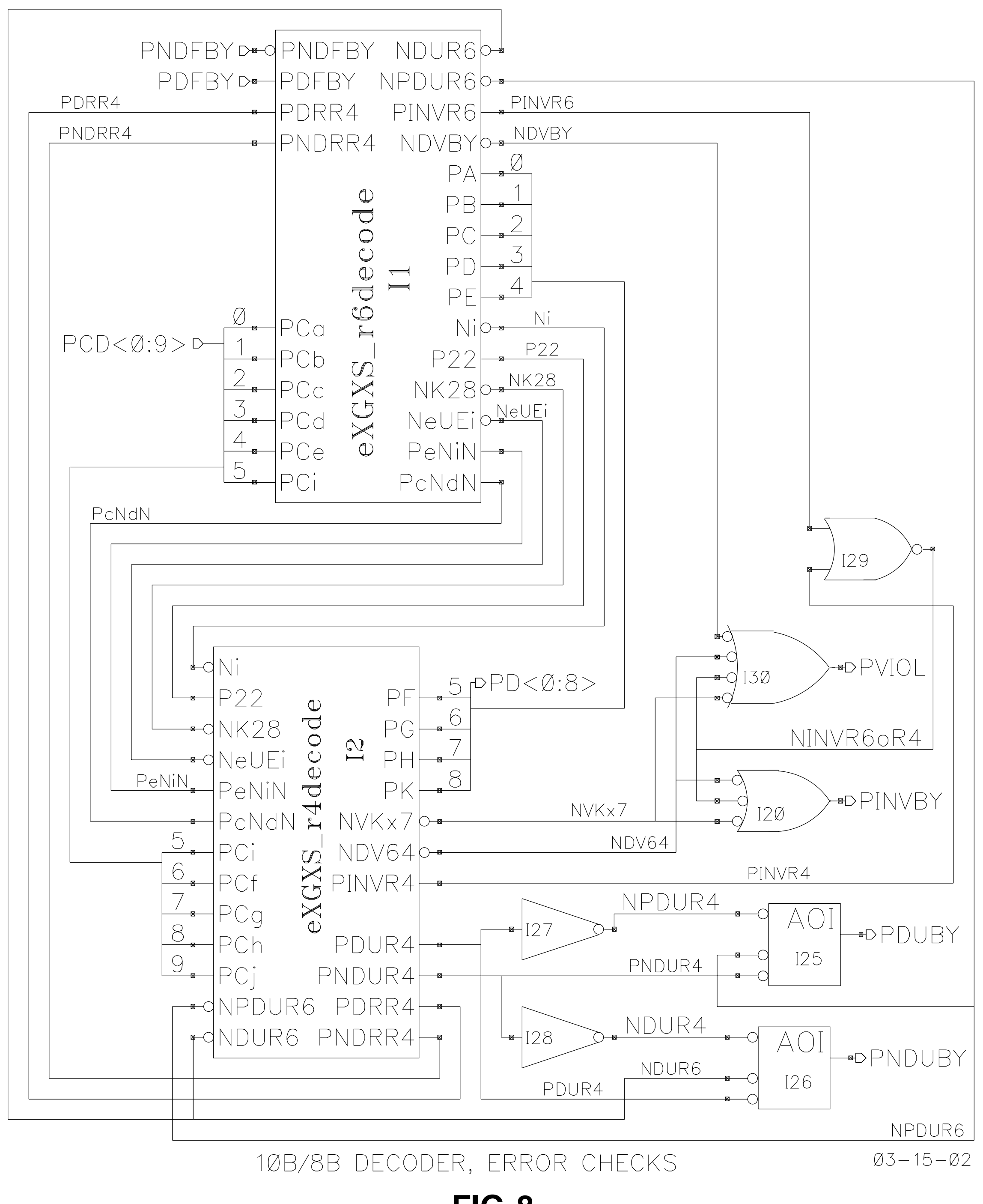


FIG.8

Disparity Monitoring

If either one or both of the vectors of a byte are disparity dependent, either PDUBY or PNDUBY are asserted to establish a positive or negative running disparity, respectively, at the end of the byte regardless of the running disparity at the front of the byte:

PDUBY = PDUR4 + PDUR6•NDUR4'

NDUBY = NDUR4 + NDUR6•PDUR4'Logic Equations for the Determination of the Disparity at the Start of the Byte

PDFBY = PDUBY_LAST + PDFBY_LAST • NDUBY_LAST’

Note that NDFBY and PDFBY are complementary: NDFBY = PDFBY’

The values of PDUBY and NDUBY are exclusive, none or one alone can be true.

Byte Disparity, Fast Version

The circuit of FIG.9 shows how the disparity at the front end of a new byte is generated based on three parameters passed along from the encoding process of the preceding byte. This circuit can operate at a higher cycle rate than the simpler circuit of FIG.10.

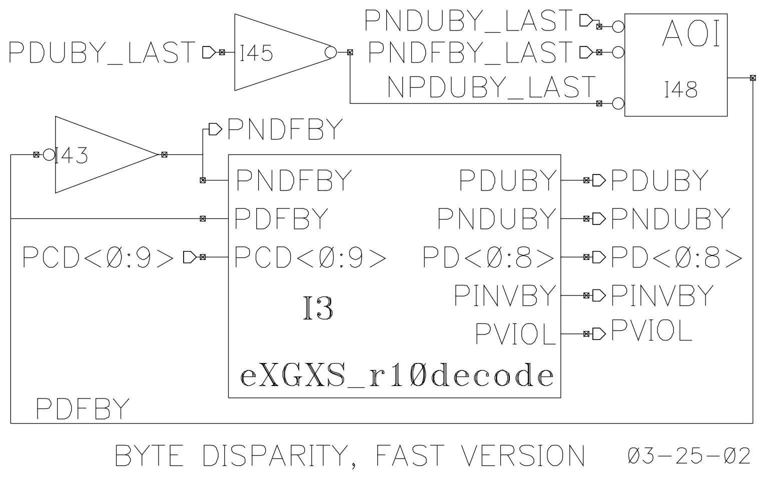


FIG.9

Notation: The signal names PDFBY and NDFBY refer to a positive and negative running disparity, respectively, in front of the byte. The signal names PDUBY and NDUBY refer the assumed positive or negative exit disparity, respectively, of the byte, regardless of the starting disparity at the front end. If neither the 6B vector nor the 4B vector of the byte is disparity dependent, none of the two outputs is asserted.

The values for the outputs NDFBY, PDUBY, and PNDUBY are stored in three latches (not shown in the diagrams) which are clocked concurrently with the decoded data output. The outputs of the latches are labelled NDFBY_LAST, PDUBY_LAST, and PNDUBY_LAST, respectively, and are used for the computation of the starting disparity NDFBY for the next byte.

Byte Disparity, Slower Version

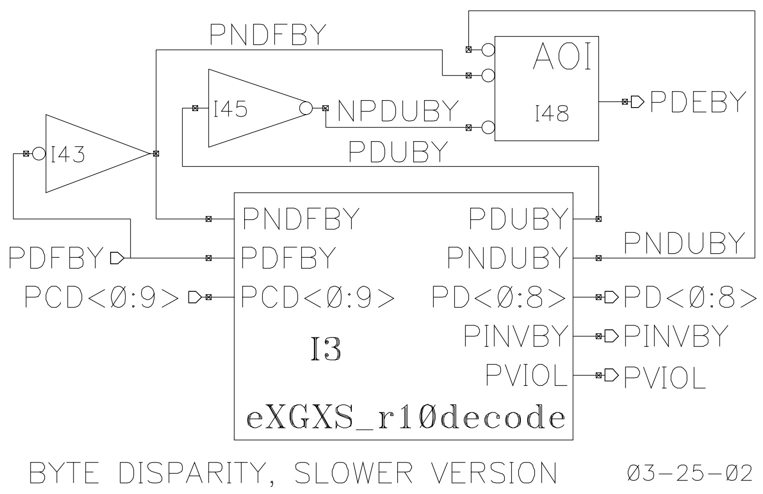


FIG.10

The incentive to use the slower version as shown in FIG.10 is the saving of two latches. If timing is not critical, the ending disparity PDEBY is generated in the same cycle as the decoding and the error checks. So only this single parameter must be passed on to next byte in the traditional manner. The output of this latch is the signal PDFBY, the disparity in front of the new byte. If the longest delay path is to the PDEBY output, the delays associated with one inverter plus one OAI21 gate will increase the critical path.

Four-Byte Word Decoder

The circuit of a four-byte word decoder is illustrated in FIG. 11.

Notation

The signal names PDFBY0 and NDFBY0 refer to a positive and negative disparity, respectively, in front of byte #0.

The signal names PDUBY0 and NDUBY0 refer the assumed positive or negative exit disparity, respectively, of byte #0. If neither the 6B vector nor the 4B vector of the byte is disparity dependent, none of the two outputs is asserted.

The values for the outputs PDFBY3, PNDFBY3, PDUBY3, and PNDUBY3 are stored in the latches which provide the signals PDFBY3_LAST, PNDFBY3_LAST, PDUBY3_LAST, and PNDUBY3, respectively, for the computation of the starting disparities PDFBY0 and PNDFBY0 for the next word cycle at the top of the diagram.

Logic Equations for the Determination of the Disparity at the Start of the Bytes

PDFBY0 = PDUBY3_LAST + PDFBY3_LAST • NDUBY3_LAST’

NDFBY0 = NDUBY3_LAST + NDFBY3_LAST • PDUBY3_LAST’

The values of PDFBY0 and NDFBY0 are complementary, but the values of PDUBY3 and NDUBY3 are exclusive, none or one alone can be true.

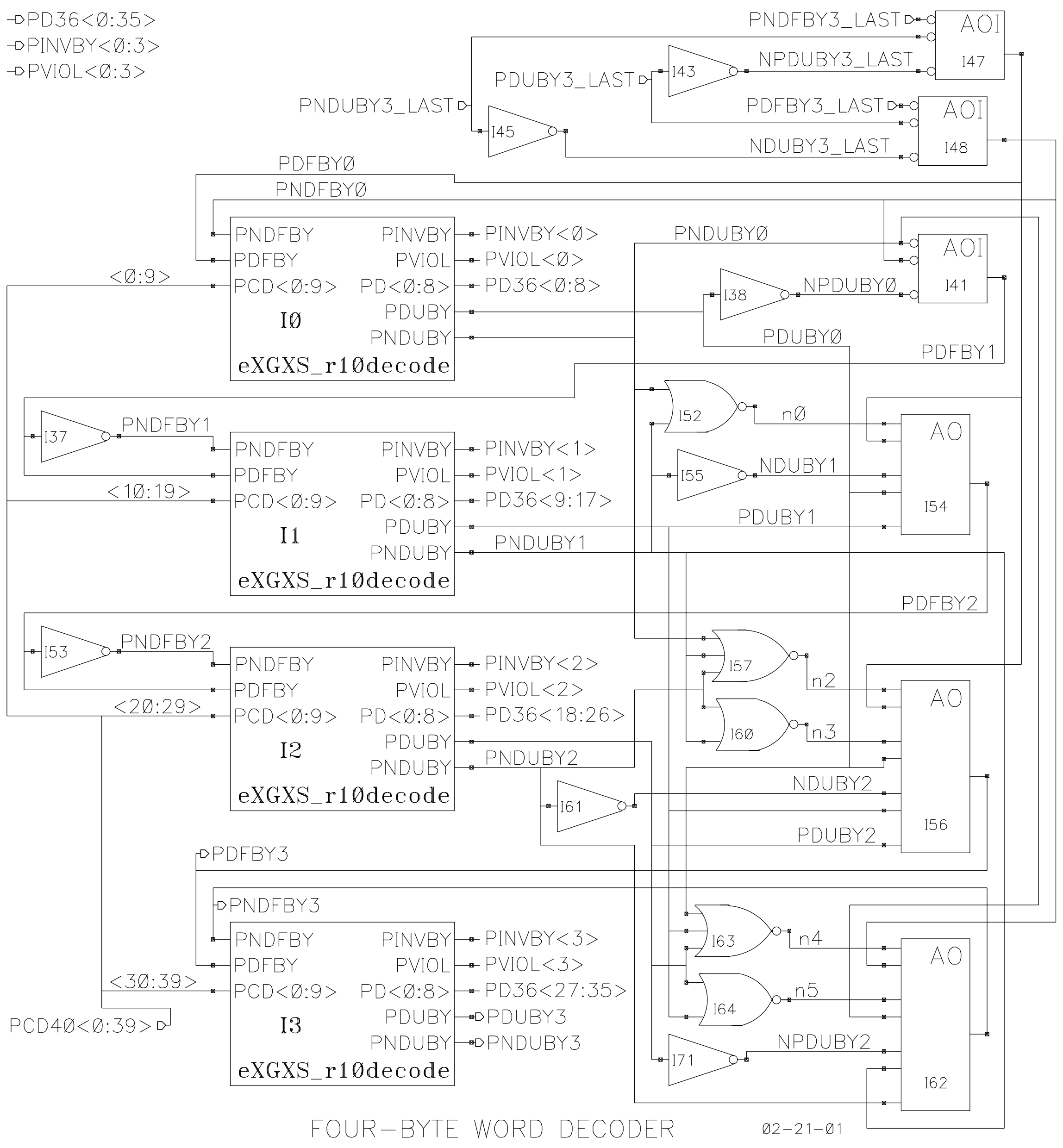


FIG.11

To minimize the circuit delays, the disparity values for the front of byte #1, #2, and #3 are determined not sequentially from byte to byte, but based on the disparity in front of byte #0 and the changes in disparity contributed by the byte(s) in between.

PDFBY1 = PDUBY0 + PDFBY0 • NDUBY0'
NDFBY1 = NDUBY0 + NDFBY0 • PDUBY0'

PDFBY2 = PDUBY1 + PDUBY0 • NDUBY1' + PDFBY0 • NDUBY0' • NDUBY1'
n0 = NDUBY0 + NDUBY1
PDFBY2 = PDUBY1 + PDUBY0 • NDUBY1' + PDFBY0 • n0'

NDFBY2 = NDUBY1 + NDUBY0 • PDUBY1' + NDFBY0 • PDUBY0' • PDUBY1'
n1 = PDUBY0 + PDUBY1
NDFBY2 = NDUBY1 + NDUBY0 • PDUBY1' + NDFBY0 • n1'

PDFBY3 = PDUBY2 + PDUBY1 • NDUBY2' + PDUBY0 • NDUBY1' • NDUBY2'
+ PDFBY0 • NDUBY0' • NDUBY1' • NDUBY2'
n2 = NDUBY0 + NDUBY1 + NDUBY2
n3 = NDUBY1 + NDUBY2
PDFBY3 = PDUBY2 + PDUBY1 • NDUBY2' + PDUBY0 • n3' + PDFBY0 • n2'
NDFBY3 = NDUBY2 + NDUBY1 • PDUBY2' + NDUBY0 • PDUBY1' • PDUBY2'
+ NDFBY0 • PDUBY0' • PDUBY1' • PDUBY2'
n4 = PDUBY0 + PDUBY1 + PDUBY2
n5 = PDUBY1 + PDUBY2
NDFBY3 = NDUBY2 + NDUBY1•PDUBY2' + NDUBY0 • n5' + NDFBY0 • n4'

The circuit implementation of shown in FIG. 11 takes advantage of the relationships:

PNDFBY1 = PDFBY1'

PNDFBY2 = PDFBY2'

These signals are not in the critical path and the added inversion does not decrease the maximum rate. For applications which have sufficient timing margin, the same simplifications can also be used for the signal PNDFBY3 and perhaps PNDFBY3 at a penalty of one inversion for each of those two signals.

Acknowledgment

The contribution of Chuck Haymes is gratefully acknowledged. He flushed out errors and verified most of the equations and circuits using logic design tools and then built a circuit macro of a single Coder/Decoder ready for porting to chip.

References

1. A. X. Widmer, The ANSI Fibre Channel Transmission Code, IBM RC 18855, 4/23/93
2. IBM US Patent 4,486,739, Franaszek and Widmer "Byte Oriented DC Balanced (0,4) 8B/10B Partitioned Block Transmission Code", Dec. 4, 1084.
3. IBM US Patent 5,245,339, R. D. Cideciyan, Flexible Encoding Method and Architecture for High Speed Data Transmission and Storage. 09/14/93.
4. IBM Microelectronics ASIC SA-27 Databook, SA14-2214-01
5. A. X. Widmer, 8B/10B CoDec for 12.5 Mbaud Operation, Memo to File, dated Jan. 27, 2000

File Name: Pat8B10B_RC