```
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Module: Error Check and Correction
;>
;>      BANK 1 MODULE
;>
;>      This module contains all the relevant files pertaining
;>      to the ECC method and algorithm used on Widget.
;>
;>      FUNCTION Ecc : BOOLEAN
;>      PROCEDURE ShiftAndXor( VAR R1,R2,R3,R4,R5,R6 : BYTE { !r1:6 } )
;>      PROCEDURE TestMod8( J : WORD { !!r8 } )
;>      PROCEDURE Test0( J : WORD { !!r8 } )
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: Ecc
;>
;>      This function is responsible for 1) checking if the data in the
;>      ReadBuffer is correctable and 2) correcting that data if it is
;>      correctable.
;>
;>      The method used was prepared by:
;>              Neil Glover
;>              Data Systems Technology Corp.
;>              1801 Aspen St.
;>              Broomfield, Co. 80020
;>              ( 303 ) 466-5228
;>
;>      Inputs: { none }
;>
;>      Outputs:
;>              Ecc : BOOLEAN { zero flag is set if not correctable }
;>
;>      K1 = BitLength( DataField ) + BitLength( CrcField ) +
;>              BitLength( EccField ) - 41
;>         = ( 1+532 )*8 + 2*8 + 6*8 - 41
;>         = 4287
;>
;>      Correction Span = 12 bits
;>
;>      R2Mask = $00
;>      R3Mask = $0F
;>
;>      Syndrome Bytes begin at ReadArray.RBuf1Ecc
;>
;>      Local Variables:
;>              R1      : BYTE { !r1 }
;>              R2      : BYTE { !r2 }
;>              R3      : BYTE { !r3 }
;>              R4      : BYTE { !r4 }
;>              R5      : BYTE { !r5 }
;>              R6      : BYTE { !r6 }
;>              Correctable : BOOLEAN { !r7/bit 7 }
```

```
;>              Aligned : BOOLEAN { !r7/bit 6 }
;>              Done    : BOOLEAN { !r7/bit 5 }
;>              J       : WORD { !!r8 }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       R1 := SyndromeByte[ 1 ] { most significant byte }
;>       R2 := SyndromeByte[ 2 ]
;>       R3 := SyndromeByte[ 3 ]
;>       R4 := SyndromeByte[ 4 ]
;>       R5 := SyndromeByte[ 5 ]
;>       R6 := SyndromeByte[ 6 ]
;>       IF ( R1=R2=R3=R4=R5=R6=0 )
;>        THEN Ecc := False
;>        ELSE
;>              J := K1
;>              Aligned := False
;>              Done := False
;>              Correctable := False
;>              WHILE ( R1 = 0 ) DO
;>                      ShiftRegsLeft 1 Byte  { Left-Hand justify syndrome }
;>                      J := J + 8
;>              While NOT( Done ) OR NOT( Aligned ) DO
;>                      ShiftAndXor
;>                      IF ( R1 = 0 )
;>                       THEN
;>                              IF ( R4=R5=R6=0 ) AND ( R2*R2Mask = 0 ) AND
;>                                      ( R3*R3Mask )
;>                               THEN
;>                                      Aligned := True
;>                                      TestMod8
;>                       IF NOT( Aligned ) THEN Test0
;>                       IF NOT( Done ) THEN J := J - 1
;>
;>              WHILE NOT( Done ) DO
;>                      ShiftAndXor
;>                      IF ( R1 = 0 )
;>                       THEN TestMod8
;>                       ELSE Test0
;>                      IF NOT( Done ) THEN J := J - 1
;>
;>              IF Correctable
;>               THEN
;>                      J := J DIV 8
;>                      Buffer1[ J ] := Buffer1[ J ] XOR R2
;>                      Buffer1[ J+1 ] := Buffer1[ J+1 ] XOR R3
;>                      Buffer1[ J+2 ] := Buffer1[ J+2 ] XOR R4
;>                      BlockMove( Buffer2, RBuffer )
;>
;>              Ecc := Correctable
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

K1              .Equ    4287
R2Mask          .Equ    0
```

```
R3Mask          .Equ    $0F
Ecc_Correctable .Equ    $80
Ecc_Aligned     .Equ    $40
Ecc_Done        .Equ    $20

Ecc:
                Clr     !r7     ;clear booleans
                Ld      !r8,#.HIBYTE. K1
                Ld      !r9,#.LOWBYTE. K1

                Ld      !r0,#0 ;get ready to check for all zero symdrome
                Ld      !rB,#6  ;load six bytes ;R1..R6 := Syndrome Bytes
                Ld      !rC,#.HIBYTE. RBuf1Ecc
                Ld      !rD,#.LOWBYTE. RBUf1Ecc
                Ld      !rA,#Wrk_Sys+$01 ;load syndrome bytes into registers

Ecc_Ld_Lp:      Ldei    @!rA,@!!rC
                Or      !r0,@!rA
                Djnz    !rB,Ecc_Ld_Lp
                Jp      Z,Ecc_End

Ecc_LHJ_While:  Or      !r1,!r1 ;WHILE ( R1 = 0 ) DO
                Jr      Nz,Ecc_Align

                Ld      !r1,!r2 ;shift left 1 byte
                Ld      !r2,!r3
                Ld      !r3,!r4
                Ld      !r4,!r5
                Ld      !r5,!r6
                Clr     !r6
                Add     !r9,#8  ;J := J + 8
                Adc     !r8,#0
                Jr      Ecc_LHJ_While

Ecc_Align:      Call    ShiftAndXor
                Jr      Nz,Ecc_Al_1

                Ld      !r0,!r4 ;IF ( R4=R5=R6=0 )
                Or      !r0,!r5
                Or      !r0,!r6
                Ld      !rF,!r3 ;AND ( R3*R3Mask = 0 )
                And     !rF,#R3Mask
                Or      !r0,!rF
                Jr      Nz,Ecc_Al_1

                Or      !r7,#Ecc_Aligned
                Call    TestMod8

Ecc_Al_1:       Tm      !r7,#Ecc_Aligned
                Jr      Nz,Ecc_Al_2

                Call    Test0

Ecc_Al_2:       Tm      !r7,#Ecc_Done
                Jr      Nz,Ecc_Crct

                Decw    !!r8 ; J := J - 1

                Tm      !r7,#Ecc_Done+Ecc_Aligned
                Jr      Z,Ecc_Align

Ecc_Shift:      Call    ShiftAndXor
                Jr      Nz,Ecc_Shft_Else
```

```
                Call    TestMod8
                Jr      Ecc_Shft_2

Ecc_Shft_Else:  Call    Test0

Ecc_Shft_2:     Tm      !r7,#Ecc_Done
                Jr      Nz,Ecc_Crct

                Decw    !!r8 ; J := J - 1
                Jr      Ecc_Shift

Ecc_Crct:       Tm      !r7,#Ecc_Correctable
                Jr      Z,Ecc_End

                Ld      !rA,#3  ; J := J DIV 8
Ecc_Div8:       Rrc     !r8
                Rrc     !r9
                Djnz    !rA,Ecc_Div8
                And     !r8,#$1F        ;mask off any unwanted carries

                Ld      !rC,#.HIBYTE. RDummy
                Ld      !rD,#.LOWBYTE. RDummy
                Add     !rD,!r9
                Adc     !rC,!r8

                Ld      !rB,#Wrk_Sys+$02 ;start with R2
                Ld      !rA,#3  ;correct 3 bytes

Ecc_Crct_Lp:    Lde     !r1,@!!rC
                Xor     !r1,@!rB
                Ld      @!rB,!r1
                Ldei    @!!rC,@!rB
                Djnz    !rA,Ecc_Crct_Lp

                 Ld     !r2,#.HIBYTE. RBuf_To_Buf2
                 Ld     !r3,#.LOWBYTE. RBuf_To_Buf2
                Call    Bank_Call
                Tm      !r7,#Ecc_Correctable ;set correctable flag

Ecc_End:        Jp      Bank_Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: ShiftAndXor
;>
;>      This procedure is used to shift the current syndrome bytes
;>      ( assumed to be located in !r1:6 ) to the right 1 bit and then
;>      Xor the syndromes with the reciprocal polynomial if needed.
;>
;>      Inputs:
;>              R1 : BYTE { !r1 }
;>              R2 : BYTE { !r2 }
;>              R3 : BYTE { !r3 }
;>              R4 : BYTE { !r4 }
;>              R5 : BYTE { !r5 }
;>              R6 : BYTE { !r6 }
;>
;>      Outputs:
```

```
;>              R1 : BYTE { !r1 }
;>              R2 : BYTE { !r2 }
;>              R3 : BYTE { !r3 }
;>              R4 : BYTE { !r4 }
;>              R5 : BYTE { !r5 }
;>              R6 : BYTE { !r6 }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       Shift the syndromes right 1 bit with carry
;>       IF the lsb of R6 was a 1  { if carry }
;>        THEN
;>              R1 := R1 XOR 140
;>              R2 := R2 XOR 12
;>              R3 := R3 XOR 10
;>              R4 := R4 XOR 40
;>              R5 := R5 XOR 24
;>              R6 := R6 XOR 8
;>
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

ShiftAndXor:
                Ld      !rC,#6  ;shift 6 bytes
                Ld      !rD,#Wrk_Sys+$01 ;start with R1

                Rcf

S_A_Xor_Lp:     Rrc     @!rD
                Inc     !rD
                Djnz    !rC,S_A_Xor_Lp

                Jr      Nc,S_A_Xor_End

                Xor     !r1,#140
                Xor     !r2,#12
                Xor     !r3,#10
                Xor     !r4,#40
                Xor     !r5,#24
                Xor     !r6,#8

S_A_Xor_End:    Or      !r1,!r1  ;IF ( R1=0 )...
                Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: TestMod8
;>
;>      This procedure is used to test if ( J MOD 8 = 0 ).
;>
;>      Inputs:
```

```
;>              J : WORD { !!r8 }
;>
;>      Outputs: { none }
;>
;>      Scoped Global Variables Changed:
;>              Done : !r7/bit 5
;>              Correctable : !r7/bit 7
;>
;>      Algorithm:
;>
;>      BEGIN
;>       IF ( J MOD 8 = 0 )
;>        THEN
;>              Done := True
;>              Correctable := True
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

TestMod8:
                Ld      !r0,!r9
                And     !r0,#07 ;get remainder from division
                Jr      Nz,TstMd8_Done

                Or      !r7,#Ecc_Done+Ecc_Correctable

TstMd8_Done:    Ret

                .LSTOFF

                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: Test0
;>
;>      This procedure is used to test if ( J = 0 ).
;>
;>      Inputs:
;>              J : WORD { !!r8 }
;>
;>      Outputs: { none }
;>
;>      Scoped Global Variables Changed:
;>              Done : !r7/bit 5
;>              Correctable : !r7/bit 7
;>
;>      Algorithm:
;>
;>      BEGIN
;>       IF ( J = 0 )
;>        THEN
;>              Done := True
;>              Correctable := True
;>      END
;>
```

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

Test0:
                Ld      !r0,!r8
                Or      !r0,!r9 ;IF ( J = 0 )..
                Jr      Nz,Tst0_Done

                Or      !r7,#Ecc_Done
                And     !r7,#$FF-Ecc_Correctable

Tst0_Done:      Ret

                .LSTOFF
```